Security Deliberations in Software Development Lifecycle

S. Shanmuga Priya
J.J. College of Engineering and Technology
Tiruchirappalli, Tamilnadu

P. D. Sheba Keiza Malarchelvi
J.J. College of Engineering and Technology
Tiruchirappalli, Tamilnadu

ABSTRACT
Security is a serious problem in software development which when not taken into consideration, exploits vulnerabilities in software. Such security related problems need to be addressed as early as possible while building software. Security problems exist for many reasons. A major thing is that, software cannot resist security attacks. Software security vulnerabilities are often caused due to the flaws that might be in specification, design, implementation or testing. These flaws are unknowingly injected by the software developers during development or left unnoticed by the software testers while testing for defects in software. This requires that developers and testers use methods that consistently produce secure software, which results in a defect less product. Security must be integrated into the software development life cycle from the beginning and must persist until the product is in use. This paper brings out the security deliberation that have to be paid due attention in the various phases of software development life cycle while developing a software.

Keywords
Software Development Life Cycle, Requirements, Design, Development, Threat Modeling, Security Testing.

1. INTRODUCTION
As computer's rule the modern world, software's become inevitable. The software's are developed in large – scale which results in low quality, high cost, and unreliable product that fail to meet the end user requirements. Mainly much of the developed software fails to look into the security issues that happen in real time which results in a defective product. People use software bearing in mind that it is reliable, can be trust upon and the operation they perform is secured. Application security deals with the protection of software after it's built, whereas software security is a process of designing, building and testing software for its security. Testing software for its correct functionalities is an essential routine that is carried out in any software built. In practice, security goes unnoticed that results in several undesired functionalities and such product vanishes from the software market in a shorter span. As in practice, the security considerations in software has always been addressed only in the production environment and henceforth security is always considered to be a non-functional requirement. Sometimes, this might result in serious causes which lead the product to failure.

In order to overcome the security defects in any software, the stakeholders must also think at right time about the required security solutions that a product must satisfy. The right time could possibly be the initial phase of the software development life cycle and must be followed in the subsequent phases too that could result in high quality, low cost, least effort and defect less product built on-time. The formal software development life cycle (SDLC) techniques include requirement analysis and gathering, designing, developing, testing, deployment and maintenance of the software. Whatever the software process methodology that has been chosen, security issues have to be thought of and suitable security solutions have to be incorporated right from the initial phase of the software development life cycle. In this paper, the security considerations to be addressed in the software development life cycle are presented.

1.1 Software Security Rules
In [1] Banerjee and Pandey, has given twenty one security rules which if practically applied from the beginning of SDLC i.e., from requirement analysis phase will certainly make room to produce secure and reliable development of software. All the stakeholders who involves in developing the software must obey these rules to ensure that vulnerabilities are not introduced into the software system. The rules are:

i) Rule of Awareness - The rule suggests a constant acquisition of new information and updating the existing knowledge relating to security aspect for the software development team which includes software architecture, software developers and software testers [2].

ii) Rule of Prevention - The rule suggests that the security in software must be synchronized in such a way that it must be able to prevent any kind of threat from internal as well as external source rather then let it happen and later on cure it.

iii) Rule of Accountability - The rule of accountability suggests that a log needs to be maintained for all the tasks/activities/acts performed during an operation/action with the purpose of prevention of the security policy violations and enforcement of certain liabilities for those acts [3].

iv) Rule of Confidentiality - The rule suggests that confidentiality must be maintained by ensuring that information is not accessed by unauthorized persons [4].

v) Rule of Integrity - The rule suggests that integrity must be maintained by ensuring that information is not altered by unauthorized persons in a way that is not detectable by authorized users [4].

vi) Rule of Availability - The rule states that a balanced approach needs to be maintained between security and availability providing a system that is highly secure and available at all the times [5].

vii) Rule of Non-repudiation - The rule states that the objective of non-repudiation is to ensure undesirability of a transaction by any of the parties involved where a trusted third party can play an important role [5].

viii) Rule of Access Control - The rule suggests that access to resources and services must be permission based and the user if given permission must be permitted/allowed to access those resources and services and these eligible users must not be

denied access to services that they legitimately expect to receive [4].

ix) Rule of Identification & Authentication – The rule suggests that the process of identification and authentication must be implemented to determine who can log on to a system and their legitimate association which various users with respect to their granted access rights [6].

x) Rule of Accuracy - The rule suggests that the software development team must perform the various actions, activities, methods, process & tasks correctly and accurately every time [2].

xi) Rule of Consistency - The rule suggests that the various requirements, protocols or standards or policies designed for securing the software system must be consistent in any case.

xii) Rule of Authorization – The rule suggests that the process of authorization must be implemented to determine what a subject can do on the system.

xiii) Rule of Privacy - The rule states that privacy can **e**nsures that individuals maintain the right to control what information is collected about them, how it is used, who has used it, who maintains it, and what purpose it is used for [4].

xiv) Rule of Assessment/Evaluation - The rule suggests that each and every process irrespective of size must be evaluated and assessed after it has been created by the software developer [2].

xv) Rule of Excellence - The rule suggests that security is a subset of quality and the control and variability of the security features will depend on the quality [2].

xvi) Rule of Flexibility - The rule suggests that the various requirements regarding security must not be rigid and must be flexible as well as realizable [2].

xvii) Rule of Fortification (Protection) – The rule suggests that the various process used in security engineering process must be secured in individuality and totality [2].

xviii) Rule of Unambiguity - The rule suggests that for easy implementation of software security, the details pertaining to it must be clear and concise [2].

xix) Rule of Error Classification - The rule suggests that errors must be categorized & classified according to a schema containing a set of security rules for better understanding of the problem which might have an impact on the security of the software [7, 8].

xx) Rule of Auditability - The rule suggests that auditability must be implemented to judge the accountability feature of software security and aids in redesign a full proof security policy and procedures for implementing a secure software system [9].

xxi) Rule of Interoperability - This rule suggests that if more software are interacting or communicating with each other then all the software involved in the interaction or communication must be secured.

The organization of the paper is as follows. Section 2 explains the Security Requirement Elicitation. Section 3 highlights Design Level Security. Section 4 describes the Security Development Considerations. Section 5 elaborates on Security Testing. Section 6 gives the Conclusion.

2. SECURITY REQUIREMENT ELICITATION

Requirement Engineering is the main building block for any software. Requirement gathering phase in SDLC is considered to be the most important and serious one, as this phase directly deals with the customer. Security requirements can vary, depending on the system construction purpose. Traditionally security requirements have been considered to be “nonfunctional” or “quality” requirements like reliability, scalability, robustness etc. Usually security requirements are prepared after a product is finished and sold, that leads to software vulnerabilities. The requirement elicitation activity involves interaction with the customer to discover, confirm, document and analyze the requirements. This phase is the key and base to the rest of the phases in SDLC, which when made a strongest foundation, the other phases could be built securely that result in a quality product.

The software security requirements are intended to be: (a) Testable and verifiable (Functional Security Requirements are testable whereas Non – Functional Security Requirements are non testable), (b) Clear, concise and non-ambiguous, (c) Implementable by software engineers even without security knowledge, (d) Capable of preventing modern software vulnerabilities when used correctly in development.

2.1 Categories of Security Requirements

In [11] Paco Hope and Peter White have classified security requirements into three categories as: *Functional Security Requirements* is a security related description that is integrated into each functional requirement. Typically, this also says what shall not happen. This requirement artifact can, for example, be derived from misuse cases. *Non – Functional Security Requirements* lists the properties that are security related architectural requirements, like "robustness" or "minimal performance and scalability". This requirement type is typically derived from architectural principals and good practice standards. *Derived Security Requirements* is derived from functional and non-functional security requirements.

In [12], Malik Imran Daud, has categorized the security requirements as: i) Functional Security Requirements, ii) Non-Functional Security Requirements, iii) Derived Security Requirements, iv) User Stories (Mainly used for developing Agile Software), and v) Abuse Case.

2.2 Steps for Security Requirement Elicitation

According to Kotonya .G and Sommerville .I [13], have given the requirement engineering process which includes activities as: i) Requirement Elicitation ii) Requirement Analysis and Negotiation, and iii) Requirement Validation

In [14], Asoke K. Talukder et al. has given 8 steps for security requirement elicitation which are as follows:

Step 1: Identify Assets
Step 2: Functional Requirements
Step 3: Security Requirements
Step 4: Threat and Attack Tree
Step 5: Rate the risk
Step 6: Decision on In-Vivo vs In-Vitro
Step 7: Non-functional Requirements
Step 8: Iterative – which insists that the step 1 through 7 can be repeated until it’s identified that all the security requirements are collected.

2.2.1 Kinds of Security Requirements

In [15] Donald G. Firesmith has given the kinds of security requirements as: i) Identification Requirements, ii) Authentication Requirements, iii) Authorization Requirements, iv) Immunity Requirements, v) Integrity Requirements, vi) Intrusion Detection Requirements, vii) Non-repudiation Requirements, viii) Privacy Requirements, ix) Security Auditing Requirements, x) Survivability Requirements, xi) Physical Protection Requirements and xii) System Maintenance Security Requirements

2.3 Threat Modeling for Security Requirement Elicitation

In [16] Suvda Myagmar et al. proposed how threat modeling can be used as a foundation for the specification of security requirements. Threat modeling involves understanding and identifies the various threats to a system. During gathering and analysis of security requirements, these threats were analyzed that makes decisions whether to mitigate or accept the risk associated with the threat. The threat modeling and security requirement identification could provide basis for the rest of the security system phases. In [17], Lee M. Clagett gave the threat modeling process which starts by identifying the assets of a system, and potential threats to those assets. A threat exists when an entry point leads to the access of an asset. Attacks to achieve that threat can be illustrated using different diagrams. For example, a system may store passwords which are an asset to an adversary, and the threat is that an adversary steals those passwords. A diagram would then be made to represent the different attacks that could achieve the threat of stolen passwords. Once the diagram is complete, it highlights the areas that need mitigation to prevent the overarching threat from being realized. Any attack that is not mitigated, or is mitigated improperly, has a vulnerability that could be exploited to gain access to the asset that the system protects.

2.4 Modeling Security Requirements with Abuse Cases

In [18], Chun Wei (Johnny) Sia, stated that the misuse and abuse cases could be used to elicit security requirement at the earliest. Business Analyst must analyze the business, identify critical assets and security services, identify vulnerabilities and must analyze misuse case to propose security requirement mechanisms. An abuse case is a use case where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders in the system. An interaction is harmful if it decreases the security (confidentiality, integrity, or availability) of the system.

In [19], Martyn Fletcher et al. have proposed how to effectively combine functional and security requirements, the interactions and iterations that is needed between functional and non-functional requirement processes to meet the objectives of Distributed Aircraft Maintenance Environment (DAME) system. The requirement process must focus on things: i) Functional Requirements in that consider the system as 'Black box', ii) Asset concerns, iii) Consider non-functional goals of the system, and iv) Identify the threat environments by identifying the potential attackers of the system.

2.5 Difficulties in Security Requirements Gathering

i) Security is constantly changing.
ii) Software security requirements must be stated in a positive tone.
iii) Software security requirements must be language and platform independent.
iv) Software security requirements must be testable and verifiable for the development process to work.
v) A project may only require some software security requirements, but not all.

3. DESIGN LEVEL SECURITY

The design phase falls next to requirement elicitation. In this, the architects, developers and designers make a complete study on the requirements specification, and they propose secure design elements, software architecture, secure design review and conduct threat modeling as per the requirements specified. Design phase typically intend the functionalities and follows as per the specifications given by the customers. The designer prepares a design specification which is very technical focusing on how to implement the system. The functional and non-functional requirement specifications are needed to describe the system's security features.

3.1 Security Design Principles

There are various security design principles in existence which provides guidelines on how to design a secure system. Security design guidelines must be known in before hand and can be incorporated them in advance in SDLC. The Security Design Principles as described by Saltzer and Schroeder [21] are:

i) Principle of Least Privilege: Subject must be given privileges that are necessary for completing its task and those rights must be discarded after use.
ii) Principle of Fail-Safe Defaults: This principle means that the default is lack of access permission. The protection scheme identifies conditions for which access permission could be granted. If action fails, system as secure as when action begins.
iii) Principle of Economy of Mechanism: Keep the design mechanisms as simple as possible, called as KISS principle.
iv) Principle of Complete Mediation: Every access must be checked and ensured if they are authorized to enjoy the privileges.
v) Principle of Open Design: Design must not be secret. The security mechanisms must be independent on the ignorance of potential attackers, but rather on the possession of specific, more easily protected by using passwords or other security implementations.
vi) Principle of Separation of Privilege: Requires multiple conditions and to grant privilege that do not depend on a single condition.
vii) Principles of Least Common mechanism: Insists that the mechanism must not be shared. If shared, every shared mechanism represents a possible information path between users and must be designed with great care to be sure it does not unintentionally compromise security.
viii) Principles of Psychological Acceptability: Adding security mechanisms especially in the human interface must not produce additional complexity to the system and the correct protection mechanisms must be applied automatically.

3.2 Threat Modeling for Design Level Security

Threat modeling is an iterative process for modeling security threats and identifies design flaws that can be exploited by these threats so that systems can be securely designed and countermeasures implemented to mitigate these threats. Even though the threat modeling can be added in every phases of SDLC, it's considered essentially in designing phase. The system, at design time, allows system architects to validate and discover whether the design meets the level of acceptable risks. Flaws in the design can be exposed, and the information thus gathered is used to improve the security quality of the design before the system is ever implemented. The designers, program managers and architects could participate in threat modeling.

The major advantages of threat modeling in design level security are it identifies the security problems, investigates potential threats and vulnerabilities, helps in planning for security tests depending on the identified threats, helps in reducing the software support costs by identifying vulnerabilities during design and development, as when the product gets into production, security defects might be reduced considerably.

3.3 Systematic Approach to Create a Threat Model

In [22] Meier J. D. et al. proposed five major threat modeling steps as follows which is an iterative approach that could be used through out the development life cycle to discover more about the design. The Fig. 1 shows the iterative threat modeling process.

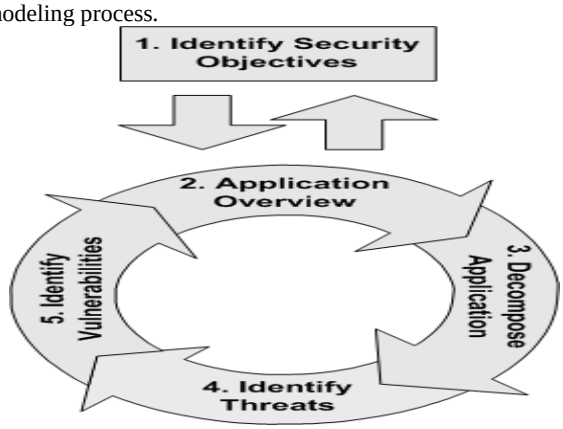

Fig. 1 The iterative threat modeling process [22]

The five threat modeling steps are:
Step 1: Identify security objectives. On identifying the security objectives clearly, focus on the threat modeling activity that helps to analyse the amount of effort that is need to spent on subsequent steps.
Step 2: Create an application overview. This step helps to list out the important characteristics of an application which helps to identify the relevant threats used during step 4.
Step 3: Decompose your application. A detailed application study and understanding the mechanism of the application makes it easier to uncover detailed threats.
Step 4: Identify threats. Use details from steps 2 and 3 to identify threats relevant to your application scenario and context.
Step 5: Identify vulnerabilities. Review the layers of your application to identify weaknesses related to your threats. Use vulnerability categories to help you focus on those areas where mistakes are most often made.
In [10], Shawn Hernan et al. proposed threat modeling using STRIDE acronym for Spoofing identity, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege was mapped to the security policies and guards against them. The mapping is shown in the Table 1:

Table 1 Mapping of Threat to Security Policies

Threat	Security Policy (Property)
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

The STRIDE model with Threat Modeling helps in identifying designing flawless in software architecture. The DFD or use case diagrams can be used in designing, which depicts the flow of system. Vulnerability analysis is also an important part in security design phase which can occur in any phase of SDLC.

3.4 Security Patterns for Design Phase

In [23], Joseph Yoder and Jeffrey Barceló were first to adapt design patterns for information security. It is easy to document what the system is required to do and difficult in identifying to list out what the system is not supposed to do. They proposed security design patterns for information security. The security patterns are:

i) Single Access Point: Providing a security module and a way to log into the system. This pattern suggests that keep only one way to enter into the system.
ii) Check Point: Organizing security checks and their repercussions. Authentication and authorization are two basic entity of this pattern.
iii) Roles: Organizing users with similar security privileges.
iv) Session: Localizing global information in a multi-user environment.
v) Full View with Errors: Provide a full view to users, showing exceptions when needed.
vi) Limited View: Allowing users to only see what they have access to.
vii) Secure Access Layer: Integrating application security with low-level security.
viii) Least Privilege: Privilege state must be shortest lived state.
ix) Journaling: Keep a complete record of usage of resource.
x) Exit Gracefully: Designing systems to fail in a secure manner.

At the end of the design, the attack surface is analyzed. If the attack surface area is high, above process is repeated until the attack surface is reduced to the minimum level.

In [24], Nobukazu Yoshioka et al. proposed security patterns in terms of security concepts for each phase of software development. They have also shown the patterns for requirement phase, design phase and implementation phase and described the achievements of researches on utilizing those proposed security patterns, including methodologies to develop secure software systems on adopting those security patterns.

3.5 Design Review

The Project Manager oversees periodic system design reviews of the system functions, performance requirements, security requirements, and platform characteristics. A system/subsystem design review is held at the end of the design phase to resolve open issues regarding one or more of the system-wide or subsystem-wide design decisions and architectural design of a software system or subsystem

A software design review is held at the end of the design phase to resolve open issues regarding one or more of the following: i) Software-wide design decisions, ii) Architectural design of a software item, and iii) Detailed design of a software item or portion thereof (such as a database) .

4. SECURITY DEVELOPMENT CONSIDERATIONS

A well secured architecture produced by using threat modeling provides good guidelines during development phase. During the implementation phase, the product team codes, tests, and integrates the software. Provided a good security requirements and designing details, but poorly coded, leads to vulnerabilities which results in non-secure software as end product that is undesirable. The threat modeling of designing phase provides important guidance during the implementation phase. Developers pay attention to ensure the correctness of code that mitigates high-priority threats and testers focus their testing on ensuring that such threats are blocked or mitigated. In addition to these, the developer must constantly monitor the security requirements and upgrade it in implementation in order to produce an up to date product.

4.1 Implementation Elements of Secure Development Life Cycle

In [25], Steve Lipner and Michael Howard gave the elements of Secure Development Life Cycle that can be applied in the implementation phase. The coding standards help developers to avoid flaws being injected to the software that can lead to security vulnerabilities. Apply security-testing tools including fuzzing tools provides structured but invalid inputs to the software being developed to maximize the error detection that might lead to software vulnerabilities. Apply static-analysis code scanning tools can be used to detect some kinds of coding flaws that result in vulnerabilities, including buffer overruns, integer overruns, and uninitialized variables. These tools can assist in discovering the vulnerabilities which needs special attention. Code reviews supplement automated tools and tests by applying the efforts of trained developers to examine source code for detecting and removing potential security vulnerabilities. They are a crucial step in the process of removing security vulnerabilities from software during the development process.

In [26], Agrawal and Khan gave a software vulnerability detection and analysis framework (SVDAF) which is independent to development life cycle. The proposed framework concentrates on vulnerability analysis that analyse vulnerable inputs at each phase and reports vulnerabilities that can be sent as a feedback in the software life cycle that the vulnerable inputs could be modified to secure outputs. Security Checklists of various SDLC phases were also listed that could be cross checked for all the phases which when followed could result in a secured software.

The developer can test the code in the development environment. While testing, developer can play the role of an attacker and test the code by giving invalid inputs which breaks the security of the system being developed. As well as the developer can play the developer role and could make the code tightly secure to prevent the attackers intruding the system.

5. SECURITY TESTING

Security testing must not concentrate on whether the developed system satisfies the documented requirements alone and must try to work beyond it to cover the full aspect of the system intended to build. Software security testing must essentially be risk-based rather than requirement based. It is difficult to find security bugs in developed software. For example, buffer overflow problems which occur during construction phase (if bound checking is not done in code) often remain invisible during standard testing. Lack of awareness about security in most developers and unavailability of proper approach for secure software development are also big reasons why security bugs generated that remain undetected. Applying security from starting stages is more costly approach in maximum software development that's why companies and customer are not concentrating about security issues and security bugs remain undetected.

5.1 Methods of Security Testing

Two major methods of security testing are: i) *Functional Security testing* which is adopted to check whether the software behaves according to certain specific functional requirements as expected. ii) *Risk-based Security testing* that addresses the negative requirements that are expected, what the software must not do. Negative test requirements are also derived from the risk analyse and tests are conducted to cover it. During software security testing, Test Case, Test Plan and Test Suite can be generated in a way as it could cover both functional security testing and risk-based testing inputs.

Malik Imaran Daud [12] has proposed few other testing methods. They are: a) *Penetration Testing* that is performed to find out the vulnerabilities in the software. The vulnerabilities when discovered must be viewed seriously and counter measures can be implemented. The various penetration testing are: i) Target Testing, ii) External Testing, iii) Internal Testing, iv) Blind Testing, and v) Double Blind Testing. b) *Fuzz Testing* which is implemented by tools called fuzzers that are programs or scripts which submits some combination of inputs to test a system.

Gu Tian-Yang et al. [27] have proposed several methods of security testing as i) Formal security testing, ii) Model-based security testing, iii) Fault injection-based security testing, iv) Fuzzy testing, v) Vulnerability scanning testing, vi) Property-based testing, vii) White box-based security testing, and viii) Risk-based security testing.

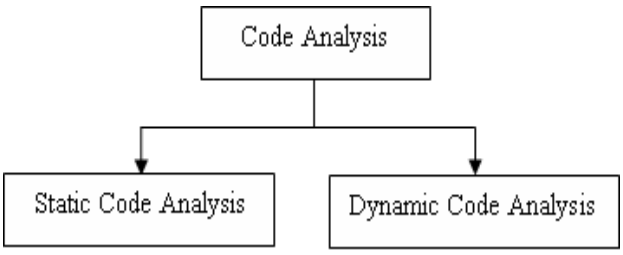

Fig. 2 Types of code analysis

Fig. 2 shows the basic types of code analysis. Static Code Analysis involves pre-compiled source code to be evaluated for conformance to identify security requirements whereas dynamic code analysis is performed on post-compiled source code that is actually running which is capable of identifying defects on codes while it is actually executed.

There are two ways of testing that are manual or automation. Manual testing carried out by the testers. Testers test the software manually for the defects. It requires a tester to play the role of an end user and use most of all features of the application to ensure its correct behavior. They follow a written test plan that leads them through a set of important test cases [28]. The problems with manual testing are, it is very time consuming process, not reusable, has no scripting facility, great effort required, and some errors remain uncovered [29].

Automation testing covers all the problems of manual testing. Automation testing automates the steps of manual testing using automation tools such as QuickTest Pro (QTP) and Test

Complete (TC). It increases the test execution speed, more reliable, repeatable, programmable, comprehensive and reusable. Hence this testing process must be done very efficiently. There must be benefits in all aspects while testing software. It would be better to automate the testing process instead doing it manually which is being needed for the emerging fields of Search based software engineering, Security testing can be manual or automated using several testing tools available in the market which makes testing easier and faster.

5.2 Security Testing Approach

In [30], Aaron Marback et al. have proposed a security testing approach that derives test cases from design-level. The approach has four folded activities as: i) Build threat trees from threat modeling, ii) Generating security tests from threat, iii) Trees generate test inputs contains valid and invalid inputs, and iv) Assigning input values to parameters.

5.3 Use of Threat Modeling in Security Testing

Threats identified during threat modeling allow the identification of security tests to verify both new and existing security flaws. Penetration tests can be driven by attack vectors for the vulnerabilities identified during threat modeling. Threats and misuse cases can drive unit test cases during implementation so that vulnerabilities in the system can be avoided as the threats to the system have been identified.

6. CONCLUSION

Security measures and controls in the software's life cycle must not be constrained to the requirements, design, code, and test phases. It is essential to continue performing code reviews, security tests, configuration control, and quality assurance during deployment and operations to ensure that updates and enhancements do not introduce security weaknesses or vulnerability issues in the software. In future, we have planned to reuse threat modeling which are used in the requirement elicitation and design phase to automatic generation of security test case in the testing phase.

7. REFERENCES

[1] Banerjee C., Pandey S. K., "Software Security Rules: SDLC Perspective", International Journal of Computer Science and Information Security (IJCSIS), Vol. 6, No.1,.

[2] Sodiya A. S., Onashoga S. A., and Ajayi O. B., "Toward Building Secure Software Systems", Vol. 3, pp. 636 – 645, 2006.

[3] Vladimir Golubev, "Using of Computer Systems Accountability Technologies in The Fight against Cybercrimes", Computer Crime ResearchCenter. Available: http://www.crimeresearch.org/ library/Using.htm.

[4] http://www.albion.com/security/intro-4.html

[5] Neil Daswani, Christoph Kern, Anita Kesavan, "Foundations of security What Every Programmer Needs to Know", APRESS, pp. 44, 2007.

[6] http://en.wikipedia.org/wiki/Access_control

[7] http://www.fortify.com/vulncat/en/vulncat/index.html

[8] http://www.fortify.com/security-resources/taxonomy.jsp

[9] Elizabeth Wasserman, "The Role of Auditing in IT and Security",Available:http://www.ciostrategycenter.com/Board/smarts/role_of_audit/index.html

[10] Shawn Hernan, Scott Lambert, Tomasz Ostwald, Adam Shostack, "Uncover Security Design Flaws using The STRIDE Approach", msdn.microsoft.com, Nov. 2006. Available: http://msdn.microsoft.com/en-us/magazine/cc163519.aspx.

[11] Paco Hope and Peter White, "Software Security Requirement the foundation for security", Cigital Inc., Available: http://sqgne.org/presentations/2007-08/Hope-Sep-2007.pdf

[12] Malik Imran Daud, "Secure Software Development Model: A Guide for Secure Software Life Cycle", Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol. I, IMECS, Hong Kong, March 17-19, 2010.

[13] Kotonya G. and Sommerville I., "Requirement Engineering Process and Techniques", John Wiley and Sons, 1998.

[14] Asoke K. Talukder, Vineet Kumar Mayura, Santhosh Babu G., Jangam Ebenezer, Muni Sekhar V., Jevitha K. P., Saurabh Samanta, Alwyn Roshan Paris, "Security-aware Software Development Life Cycle (SaSDLC) – Processes and Tools", Accepted for Presentation at WOCN 2009, Cairo, Egypt, 28-30 April 2009.

[15] Donald G. Firesmith, "Engineering Security Requirements", Firesmith Consulting, U.S.A Vol. 2, No. 1, January-February 2003. Available: http://www.jot.fm/issues/issue_2003_01/column6.

[16] Suvda Myagmar, Adam J. Lee, and William Yurcik, "Threat Modeling as a Basis for Security Requirements", IEEE Symposium on Requirements Engineering for Information Security (SREIS), August 2005.

[17] Lee M. Clagett, "Security Requirements for the Prevention of Modern Software Vulnerabilities and a Process for Incorporation into Classic Software Development Lifecycles", Thesis dissertation.

[18] Chun Wei (Johnny), Sia, "Misuse Cases and Abuse Cases in Eliciting Security Requirements", 25 Oct 2005.

[19] Martyn Fetcher, Howard Chivers, Jim Austin, "Combining Functional and Security Requirements' Processes", ROLLS ROYCE PLC-REPORT-PNR, Vol. 93025, 2005.

[20] Swapnesh Taterh, Yadav K. P., Sharma S. K., "Threat Modeling and Security Pattern used in Design Phase of Secure Software Development Life Cycle", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 4, April 2012.

[21] Saltzer, Jerome H. and Schroeder, Michael D., "The Protection of Information in Computer Systems", Proceedings of the IEEE 63, pp. 1278-1308, September 1975.

[22] Meier J.D., Alex Mackman, Blaine Wastell, "Threat Modeling Web Applications Patterns & Practices

Library", Microsoft Corporation, May 2005. Available: http://msdn.microsoft.com/en-us/library/ff648006.aspx

[23] Joseph W. Yoder and Jeffrey Barcalow (1997), "Architectural Patterns for Enabling Application Security", Proc., 4th Conference on Patterns Languages of Programs (PLoP'97) Monticello, Illinois.

[24] Nobukazu Yoshioka, Hironori Washizaki and Katsuhiasa Maruyama, "A survey on security patterns — progress in informatics", No.5, pp.35-47, (2006).

[25] Steve Lipner and Michael Howard, "The Trustworthy Computing Security Development Lifecycle", Security Engineering and Communications, Security Business and Technology Unit, Microsoft Corporation, March 2005.

[26] Agrawal A. and Khan R. A., "A Framework to Detect and Analyze Software Vulnerabilities – Development Phase Perspective", International Journal of Recent Trends in Engineering, Vol. 2, No. 2, November 2009.

[27] Gu Tian-yang, Shi Yin-sheng, and Fang You-yuan, Research on Software Security Testing, World Academy of Science, Engineering and Technology, 2010.

[28] Chilenski J. and Miller S., "Applicability of modified condition/decision coverage to software testing", Software Engineering Journal, pp. 193–200, September 1994.

[29] Mark Fewster and Dorothy Graham. Software test automation: effective use of test execution tools. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[30] Aaron Marback, Hyunsook Do, Ke He, Samuel Kondamarri, Dianxiang Xu, Security Test Generation using Threat Trees, Proc. Fourth Int'l Workshop Automation of Software Test (AST '09), May 2009.