American Journal of Applied Sciences 5 (12): 1622-1629, 2008
ISSN 1546-9239

Secured Requirement Specification Framework (^{S}RSF)

[1]R.A. Khan and [2]K. Mustafa
[1]Department of IT, Babasaheb Bhimrao Ambedkar University, Lucknow, UP, India
[2]Department of Computer Science, Jamia Millia Islamia, New Delhi, India

Abstract: Generally, software engineers are poorly trained to elicit, analyze and specify security requirements, often confusing them with the architectural security mechanisms that are traditionally used to fulfill them. One of the most ignored parts of a security-enhanced software development lifecycle is the security requirements engineering process. Security should begin at the requirements level and must cover both overt functional security and emergent characteristics. A critical review of literature on the attempts in this regard reveals that there is no standard framework or model available for delivering secured software requirement specification. This study presents a framework for the security requirement specification called Secured Requirement Specification Framework (^{S}RSF), which is prescriptive in nature.

Key words: Software security, security requirement, risk analysis, use cases, abuse cases, secure software development life cycle

INTRODUCTION

Security is important in all aspects of life and the increasing pervasiveness and capability of information technology makes IT infrastructure security increasingly so[1]. The continual and increasing publicity given to failures of IT security demonstrate the importance of developing and assuring software to appropriate levels of security. The problem with most software today is that it contains numerous flaws and errors that are often located and exploited by attackers to compromise the software's security and other required properties. An article by C. Mann 'Why is Software So Bad', concludes that bad habits and inadequate software life cycle processes have led to the development of poor software[2]. No doubt, there is advancement in the software engineering process and tools, but the literature survey reveals that the progress in improving the quality of software is still lagging[3]. This assessment can be made with respect to security by answering the question why is software so insecure and vulnerable?

Secure software is software that is able to resist most attacks, tolerate the majority of attacks it cannot resist and recover quickly with a minimum of damage from the very few attacks it cannot tolerate. Secure software cannot be intentionally subverted or forced to fail. It remains dependable in spite of intentional efforts to compromise that dependability[4]. Software security matters because so many critical functions have come to be completely dependent on software. This makes software a very high-value target for attackers, whose motives may be malicious, criminal, adversarial, or terrorist. What makes it so easy for attackers to target software is the virtually guaranteed presence of vulnerabilities, which can be exploited to violate one or more of the software's security properties, or to force the software into an insecure state[4]. Most of the successful attacks on software result from successful targeting and exploitation of known but non-patched vulnerabilities or unintentional misconfiguration[4].

Development of high assurance security software requires knowledge and techniques not commonly taught to or practiced by most software developers. The lack of rigor and discipline in the software development process, driven by the focus on short time-to-market, performance and functionality, has produced rampant security vulnerabilities that gravely affect a large range of computing environments, from small deeply embedded safety applications to large enterprise software platforms[6].

In the traditional software development lifecycle (SDLC), security is often an afterthought and security estimation and prediction efforts are delayed until after the software has been developed. Vulnerabilities are an emergent property of software which appears throughout the development phases. Therefore, it is highly desirable to adopt a 'before, during and after' approach of software security to software development process[7]. A life cycle process that includes security

Corresponding Author: R.A. Khan, Department of IT, Babasaheb Bhimrao Ambedkar University, Lucknow, UP, India

assurance is needed for improving the overall security of software[3].

Requirements engineering is critical to the success of any major development project. Several efforts have been made to prove that requirements engineering defects cost 10 to 200 times as much to correct once fielded than if they were detected during requirements development[4]. It is also proven by the researchers and industry personals that reworking requirements defects on most software development projects costs 40 to 50% of total project effort and the percent age of defects originating during requirements engineering is estimated at more than 50%. The total percent age of project budget due to requirements defects is 25 to 40%[4]. The need to consider security from the ground up is a fundamental tenet of secure software development. While many development projects produce next versions that build on previous releases, the requirements phase offers the best opportunity to build secure software. Therefore, it is highly desirable to define security requirements during software requirement specification.

SECURITY DEVELOPMENT LIFE CYCLE

Applications developed with security in mind are safer than those where security is an afterthought[9]. Researchers and practitioners working in the area of Software Security Engineering have focused on using so-called best practices in the software lifecycle. These are the security-enhanced software development methodology which provides an integrated framework, or in some instances, phase-by-phase guidance for promoting security-enhanced development of software throughout the life cycle phases.

Secure software development is the term largely associated with the process of producing reliable, stable, bug and vulnerability free software. There are a number of ways that this can be undertaken within traditional application development, but the most common procedures involve phased security assessments and reviews that encompass knowledge share; design and implementation assessment and regular security health checks. There are several reasons why organizations choose to follow a secure software development program including the followings[27]:

- Mitigating of the risk of a serious application flaw exposing the organization or its data
- Providing a better quality in the completed product or service, thereby reducing any risk of liability or negative publicity
- Reducing IT security costs after implementation and ultimately provides a better return on IT security investment
- Improving maintenance time by reducing the effort needed to fix bugs after delivery
- Improving productivity and allocating resource. Less development work is required to engineer solutions to problems identified early. Their root causes may be determined, resolved and adapted to prevent reoccurrence
- Shortening delivery times by reducing the time spent in the integration and system test/debug phases
- Therefore, a Secure Development Process should be integrated with all phases of the software development lifecycle. It ensures that security is a consideration at all stages of software development lifecycle, from requirement analysis through design and implementation to deployment in production environments
- Application security and insecurity, is a rapidly evolving area. In order to successfully integrate security to the development process a comprehensive understanding of the potential issues and failures is required, together with intrinsic knowledge of the existing development processes
- Literature survey reveals that much work has been done in developing such a methodology[14-21]. Following section describes security enhanced software development methodologies proposed by various researchers and practitioners

Microsoft's framework (SDL): Microsoft developed a trustworthy computing Security Development Life Cycle (SDL) in 2002 during its security pushes. The framework encompasses the addition of a series of security-focused activities and deliverables to each of the phases of Microsoft's software development process. The entire product team focuses on updating the product's threat models, performing code reviews and security testing and revising documentation. The major objective of the proposed framework was to confibbrm the validity of the product's security architecture documentation through a focused, intensive effort, uncovering any deviation of the product from that architecture and identify and remediate any residual security vulnerabilities. The framework comprises activities that would normally be distributed across multiple SDLC phases into a single relatively short time period.

Oracle's framework (OSSA): Like Microsoft, Oracle Corporation has made an extensive effort in developing a framework for secured software development. Oracle's product development and maintenance process includes a comprehensive set of security assurance mechanisms and processes. The goals of these processes are to improve the strength of security mechanisms and reduce the likelihood of security flaws in products. Collectively these assurance mechanisms and processes are known as Oracle Software Security Assurance (OSSA).

Comprehensive, Lightweight Application Security Process (CLASP): John Viega, chief security architect and vice president of McAfee, Inc, made an effort in developing a framework for secured software development in 2004[14]. He developed a Comprehensive, Lightweight Application Security Process (CLASP) to insert security methodologies into each phase of software development life cycle. CLASP provides a well-organized and structured approach to moving security concerns into the early stages of software development life cycle, whenever possible. CLASP consists a set of 30 process pieces that can be integrated into any software development process. It takes a prescriptive approach and documents activities that organization should be doing and finally provides an extensive wealth of security resource that make implementing those activities reasonable.

Gary McGraw's approach: Gary McGraw describes Seven Touch points for Software Security in his book Software Security: Building Security In[22]. This is nothing but a lightweight best practice to be applied to various software development artifacts. This set of software security best practices referred to as touch points. Putting software security into practice requires making some changes to the way organizations build software. These security best practices have their basis in good software engineering and involve explicitly pondering the security situation throughout the software life cycle.

TSP-secure: The SEI's Team Software Process (TSP) provides a framework, a set of processes and disciplined methods for applying software engineering principles at the team and individual level[23]. TSP for Secure Software Development (TSP-Secure) extends the TSP to focus more directly on the security of software applications. The TSP-Secure framework is a joint effort of the SEI's TSP initiative and CERT program. The principal goal of this framework is to develop a TSP- based method that can predictably produce secure software.

Secure Software Development Model (SSDM): It has been observed that producing secure software requires integrating Software Engineering (SE) process with Security Engineering[17]. Simon Adesina Sodiya, a researcher at the Nigerian University of Agriculture developed a Secure Software Development Model (SSDM), which integrates security engineering with software engineering so as to ensure effective production of secure software products[2,12]. SSDM is a unified model that combines some existing software security techniques. It is structured towards developing secure software. The model shows clearly how software development should be linked to security engineering in order to come up with the secured software.

AEGIS: Developing a secure software system is a complex and time-consuming process that seeks to accommodate frequently competing factors, such as functionality, scalability, simplicity, time-to-market, etc. Appropriate and Effective Guidance for Information Security (AEGIS) is a software development process to develop secure and usable software system. AEGIS is formulated to be a lightweight process that can fit into any software development process. It was integrated into an incremental development process[16].

Rational unified process-secure: The Rational Unified Process (RUP) is one of the most popular and complete process models being used by developers in recent years. Most of the guidelines and activities in this process model is based on software engineering related standards that have been proposed by ISO and IEEE. This process model is extended to be used in developing secure software systems by researchers at Amirkabir University of Technology (Tehran Polytechnic)[24,25] and named as RUPSec. Requirement Discipline of RUP is extended to improve RUP for developing secure software systems. These extensions are adding and integrating a number of Activities, Roles and Artifacts to RUP in order to capture, document and model threats and security requirements of system[24].

Clear and stepwise activities are introduced to developers to assure that security requirements are captured and modeled. These models are used in design, implementation and test activities[24]. The major objective of the RUPSec is to define a software process model in which security requirements are considered in all development phases of a computer-based system: business modeling, requirements, analysis and design, implementation and testing.

Table 1: Activities to be carried out for Securing the Requirement Phase

Microsoft SDL	Oracle Secure Software Assurance	CLASP	McGraw's 7 Touch points	TSP-Secure
Review, recommend and ensures for security team plans; Identifies critical objectives; Identify security feature requirements; Conduct risk analysis of requirements	Ensure developers are security aware; Ensure security standards exist and documented; Ensure security tools and libraries are available.	Specify operational environment Perform security analysis of requirements Detail misuse cases	Develop security requirement specification Build abuse cases	Design security specifications Identify assets Develop use cases and abuse cases

Security extension to MBASE: Model-Based Architecting and Software Engineering (MBASE) is a set of guidelines that describe software engineering techniques for the creation and integration of development models for a software project. The models to be integrated extend beyond Product (development) models such as object oriented analysis and design models and traditional requirements models, to include Process models such as lifecycle and risk models, Property models such as cost and schedule and most notably success models such as business-case analysis and stakeholder win-win. MBASE was originally introduced by the University of Southern California's (USC) Centre for Software Engineering in 1999. MBASE is comprehensive and risk driven approach, which combines various models and demonstrate their capabilities and feasibility, but so far lacks specific guidelines for developing a secure system

Secure software engineering: Secure software engineering (S2e), a process-oriented approach to software development, improves secure software and reduces attack surfaces and vulnerability entry points. In secure software engineering a predictable, manageable process replaces ad-hoc penetrate and patch methodologies[25,26]. Its techniques are tailored to each project are phased in gradually. Therefore, provision is there to adopt all or parts of the secure software engineering approach, depending on its needs[26]. The basic objective of S2e is to significantly reduce the number of vulnerabilities in the software that results. S2e is also intended to benchmark using a CMM such as ISO/IEC 21827 SSE-CMM[25]. Secure software engineering dramatically improves software quality by respecting security aspects and reduces post-release maintenance and service costs. It is built upon security and antisecurity experts' knowledge

SECURITY AT SOFTWARE REQUIREMENT SPECIFICATION

Developing secure software is a complex and time-consuming process that seeks to accommodate frequently competing factors including functionality, scalability, simplicity, time-to-market, etc. Software engineering research has recently focused on improving the modeling abilities in terms of non-functional requirements such as stability[10], performance[11], fault tolerance[12] and security[13].

One of the most ignored parts of a security-enhanced software development lifecycle is the security requirements engineering process. Unfortunately, security is assumed to be a technical issue and therefore best handled during architecture and design or, better still, during implementation. Since software requirements are often written by non-technical business analysts, this is a common conclusion[8]. Software that does not have its requirements elicited, enumerated and well-documented will most likely is of low quality. It is important to have a clear idea of secured requirements to build a good threat model. An extensive literature survey reveals that a lot of work has already been done on how to effectively elicit, validate and document software requirements, which may be extended to include security at requirement specification[8].

Security should begin at the requirements level and must cover both overt functional security and emergent characteristics. One way to cover the emergent security space is to build abuse cases. Similar to use cases, abuse cases describe a system's behavior under attack, providing explicit coverage of what should be protected, from whom and for how long. Table 1 describes the activities proposed by various researchers and practitioners in the requirement phase of the software development life cycle to come up with the secured requirement.

THE FRAMEWORK

Literature survey reveals that security mechanism should be implemented at the user interface level as well as at the application-under-development level. At the user interface level security mechanism started with an analysis of user's security requirements. In order to accomplish the goal of the theme on security at requirement phase, following objectives are set forth:

- To ensure that users and client applications are identified and identities are properly verified

- To ensure that users and client applications can only access data and services for which they have been properly authorized
- To detect intrusion attempts by unauthorized users and client applications
- To ensure that the unauthorized malicious programs (e.g., viruses) do not infect the application or component.
- To ensure that communications and data are not intentionally corrupted
- To ensure that parties to interactions with the application or component cannot later repudiate those interactions.
- To ensure that confidential communications and data are kept private
- To enable security personnel to audit the status and usage of the security mechanisms
- To ensure that applications and centers survive attack, possibly in degraded mode
- To ensure that centers and their components and personnel are protected against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism, sabotage, or terrorism)
- To ensure that system maintenance does not unintentionally disrupt the security mechanisms of the application, component, or center

Taking into account the objectives discussed above a roadmap or framework for developing secured software specification, an integrated and prescriptive framework ^{S}RSF is hereby proposed. ^{S}RSF has been attempted to be highly implementable and prescriptive in nature. It has been structured into a hierarchical description including premises, generic guidelines and secured requirement specification process to be followed in order as follows.

Premises: The following premises have been considered when the proposed framework is being used to develop a secured software requirement specification:

- There is no universally agreed-upon definition for each of high-level security requirement attributes
- The set of security attributes used in the development of the framework has been defined operationally in the context
- A common set of features for the desired requirement specification may be used to form the basis for its development
- The recourse optimization in SDLC depends on the early use of procedure for requirement specification and uncovering of vulnerabilities as far as possible
- The approach to risk estimate should be more applicable to identifying low security software than the highly secured code

Generic guidelines: The guidelines before following the process to develop the secured software specification may be listed as follows:

- Assure compliance/ adherence to collect a generally-accepted set of characteristics that good requirements possess
- Identify and persist with all the security-specific issues involved in requirements engineering
- Identify policies and standards as a source of software security requirement
- Assure to control somehow all the extraneous and intervening factors that may affect the outcome based prediction

Requirement specification development process: The development process of the security requirement is comprised of five phases together with prescriptive steps for each and has been depicted pictorially in ^{S}RSF, Fig. 1. Such a framework has been proposed on

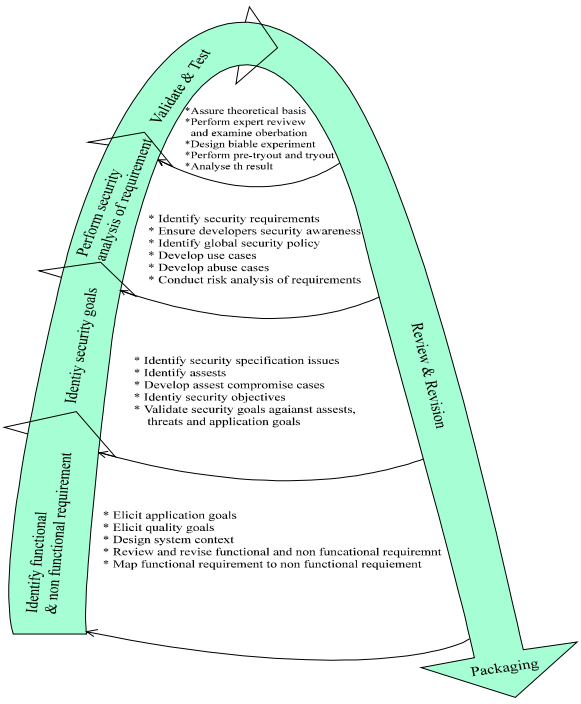

Fig. 1: Secured Requirement Specification Framework (^{S}RSF)

the basis of integral and basic components for designing secured requirement specification. The first phase starts with the identifying functional and non-functional requirements. Identifying security goals for the desired specification is treated as an important task and has been putforth as a second phase, followed by the phases termed as perform security analysis of requirement, validation and testing, review and revision and packaging. An attempt has been made to symbolically represent the spirit of developing the secured requirement specification make the framework prescriptive in nature followed by a brief description of each o f the phases comprising the depicted steps in the special reference to development of the same.

Identify functional and non-functional requirement: One of the foremost tasks of this comprehensive problem-solving activity is to identify the functional and non-functional requirements. This phase will elicit the application goals and quality goals. System context will be designed. Revision of the identified functional and non-functional requirement will be based on the review of the same. Importance of this phase lies in the fact it serves as the basis for evolving initial set of specifications to subsequent phases of development.

Identify security goals: There are five general steps required to identify the security goals including identification of security specification issues, identification of the assets, development of asset compromise cases, identification of the security objectives and validation of security goals against assets, threats and application goals. The result is a set of security goals, which are validated by ensuring that the business goals remain satisfied.

Perform security analysis of requirement: Before performing a security analysis, one must understand what is to be built. This task should involve reviewing all existing high-level system documentation. If other documentation such as user manuals and architectural documentation exists, it is advisable to review that material as well. This phase comprises of the sub activities including identification of security requirements, ensuring developers security awareness, identification of global security policy, conducting risk analysis of requirement.

Validate and test: Common wisdom, intuition, speculation and proof of concepts may not be reliable sources of credible knowledge, hence it is necessary to place the specified requirement under testing. Testing is one of the best empirical research strategies, performed through quantitative analysis of experimental data on implementation. Testing is crucial for the success of any software measurement project. This phase comprises of assuring theoretical basis, performing expert review and examination observation, designing viable experiment, performing pre-tryout and tryout and analyzing the result and finalizing the specification.

Review and revision: This phase is informal and has been placed as the fifth phase with free-to-enter at any of the earlier phases. Basic idea of such a prescription is to have adequate enough exposure and then turn back for better review, in the light of all the previous phases. However, informal reviews and revisions may be carried out at any of the stages in the requirement specification development process.

Packaging: This phase is the last and conclusive phase of the specification development process. During this phase the developed requirement specification is prepared with the needed accessories to become a ready-to-use product, like any other usable product.

VALIDATION OF THE FRAMEWORK

A key verification step for the framework described in this paper is the ability to show that the system can satisfy the security requirements. An experimental tryouts and statistical analyses at a large scale with typical representative samples may be needed to standardize the framework. More developmental activities using the framework may be carried out by the researchers and practitioners. Review of already developed or underdevelopment requirement specification may be guided by the framework and this framework may form the basis for the development of better-refined roadmap.

CONCLUSION

Application designed with security in mind is safer than those where security is an afterthought. Traditionally, security issues are first considered during the Design phase of the software development life cycle once the software requirement specification has been frozen. This paper has presented a prescriptive framework for security requirement specification comprising of six steps including identification of functional and non-functional requirement, identifying security goals, performing security analysis of requirement, validation and testing, review and revision and packaging. The developed framework may be used

to ensure the software requirement specification contains the security specifications which helps improve the security of application and reduce the cost of re-work later.

REFERENCES

1. U.S. Department of Homeland Security, The National Strategy to Secure Cyberspace, February 2003.
2. Mann, C., 2002. Why Software Is so Bad, Technol. Rev. (July/August).
3. David P. Gilliam, Thomas L. Wolfe, Josef S. Sherif, 2003. Software security checklist for the software life cycle, proceedings of the twelfth IEEE international workshops on enabling technologies: Infrastructure for collaborative enterprises (WETICE'03) IEEE.
4. Nancy R. Mead, 2007. How to compare the security quality requirements engineering (SQUARE) method with other methods, technical note, CMU/SEI-2007-TN-021.
5. Security In The Software Lifecycle, Making Software Development Processes-and Software Produced by Them-More Secure, Department of Homeland Security, DRAFT Version 1.1 - July 2006.
6. Irvine, C.E., T.E. Levin, T.D. Nguyen and G.W. Dinolt, 2004. The trusted computing exemplar project. Proceedings of the 2004 IEEE Systems. Man and cybernetics information assurance workshop, West Point, NY, pp: 109-115.
7. Elfriede Dustin <http://www.devsource.com/author_bio/0,1908,a=6041,00.asp>, The Secure Software Development Lifecycle, November 11, 2006. http://www.devsource.com/article2/0,1895,2055993,00.asp
8. Rudolph Araujo, Security Requirements Engineering: A Road Map, Security Feature (July 2007) http://www.softwaremag.com/L.cfm?Doc = 1067-7/2007.
9. Roshan Chandran, Security at Software Requirement Specification, AUGUST 2004. http://Palisade.Plynt.Com/Issues/2004aug/Security-Requirements/
10. Jazayeri, M., 2002. On Architectural Stability and Evolution. Reliable Software Technologies-Ada-Europe, Vienna, Austria, pp: 17-21. http://www.infosys.tuwien.ac.at/Staff/mj/papers/archstab.pdf
11. Denaro, G., A. Polini and W. Emmerich, 2004. Performance testing of distributed component architectures. Beydeda, S. and V. Gruhn (Eds.). Building Quality into COTS Components-Testing and Debugging. Springer. http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/BeyadaGruhn/PerformanceTesting.pdf
12. Guerra, P.A.D.C., C. Rubira and R. de Lemos, 2003. A Fault-Tolerant Software Architecture for Component-Based Systems. Lecture Notes in Computer Science. 2677: 129-149. Springer.
13. Jürjens, J., 2003. UMLsec: Extending UML for Secure Systems Development. LNCS.
14. Oracle Software Security Assurance [web page] (Redwood Shores, CA: Oracle Corporation). http://www.oracle.com/security/software-security-assurance.html
15. James, W. Over (CMU SEI), TSP for Secure Systems Development (presentation at CMU SEI, Pittsburgh, PA). http://www.sei.cmu.edu/tsp/tsp-secure-presentation/
16. Ivan Flechais, Cecilia Mascolo and M. Angela Sasse, 2006. Integrating Security and Usability into the Requirements and Design Process, Proceedings of the Second International Conference on Global E-Security, London, UK, http://www.softeng.ox.ac.uk/personal/Ivan.Flechais/downloads/icges.pdf
17. Sodiya, Adesina Simon, Onashoga, Sadia Adebukola, Ajayi, Olutayo Bamidele. Towards building secure software systems, Proceedings of Issues in Informing Science and Information Technology; 2006 June 25-28; Salford, Greater Manchester, England. Vol. 3. http://informingscience.org/proceedings/InSITE2006/IISITSodi143.pdf
18. Mohammad Zulkernine and Sheikh Iqbal Ahamed, 2006. Software Security Engineering: Toward Unifying Software Engineering and Security Engineering, chap. XIV in Enterprise Information Systems Assurance and System Security: Managerial and Technical Issues, Merrill Warkentin and B. Rayford Vaughn, (Eds.). (Hershey, PA: Idea Group Publishing).
19. Royce, Managing the Development of Large Software Systems, op cit.
20. Dan Wu, Ivana Naeymi-Rad and Ed Colbert (University of Southern California), Extending MBASE to Support the Development of Secure Systems, in Proceedings of the Software Process Workshop, Beijing, China, May 25-27, 2005. www.cnsqa.com/cnsqa/jsp/html/spw/download/Copy%20of%20MBASE_Sec_Ext_danwu_abstract%5B1%5D.v1.revisedv2.1.pdf

21. Secure Software Engineering portal. http://www.secure-software-engineering.com/
22. Gary McGraw, 2006. Software Security: Building Security In, Addison Wesley.
23. Humphrey, Watts S., 2002. Winning with Software: An Executive Strategy. Boston, MA: Addison Wesley, (ISBN 0201776391).
24. Reza, M., A. Shirazi, P. Jaferian, G. Elahi, H. Baghi and B. Sadeghian, 2005. RUPSec: An Extension on RUP for Developing Secure Systems-Requirements Discipline, Proceedings of World Academy of Science, Engineering and Technology Vol. 4: ISSN 1307-6884, pp: 208-212.
25. Software Security Assurance, 2007. State-of-the-Art Report (SOAR) Information Assurance Technology Analysis Center (IATAC) Data and Analysis Center for Software (DACS) Joint endeavor by IATAC with DACS.
26. Thorsten Schneider, 2006. Secure Software Engineering Processes: Improving the Software Development Life Cycle to Combat Vulnerability, sqp VOL. 9, NO. 1/©, ASQ, pp: 4-13.
27. Glyn Geoghegan, 2004. Secure Development Framework, A Corsaire White Paper.