CHAPTER 6

Requirements Gathering and Secured Software Development

6.1 Introduction

In the age of Internet, businesses are compelled to connect to the suppliers, vendors, remote users etc. The extensive network usage among various users can bring wide range of security threats that can affect the information owned by the business. The major cause of increasing security incidents is the presence of vulnerabilities in the software that mainly owe to defective specifications, design and insecure development [RD04]. The normal development processes leaves many vulnerabilities in software that may be later exploited by the intruders. Moreover, the software developers are mainly concerned about the deadlines resulting from the large business pressure. It involves very less attention towards the security flaws the software might have [SOL11]. It has been argued that building security in the product during development process is easy and cheaper in terms of both time and money [CSR06, GHS, LINK58, MHS05]. Adding security at later stage may disrupt ongoing operations of an organization [NIST800-12].

The development of secured software product begins with requirements gathering. Formal approach to gather security requirements involve the use of requirements gathering tools such as Security Requirements Quality Engineering (SQUARE), checklists etc. [G+03, MHS05]. As the software development proceeds, security must be considered during all its stages. Some of the secured development processes include Secure Development Lifecycle (SDL), Comprehensive, Lightweight Application Security Process (CLASP), Touchpoint etc. [ML06, MAC06]. It has been observed that security requirements are non-functional requirements, and are usually not considered in detail along with
functional requirements. Also, the requirements gathering do not consider the stakeholders. The secured development processes followed are complex and specific to certain domain. Moreover, the development processes do not consider the insecurity arising during various stages.

Software security is consequence of security requirements gathered from the stakeholders and secured development process followed. We discuss a brief background on the related work in Section 6.2. In order to ensure right security requirements to meet business needs, we suggested *Software Security Requirements Gathering Instrument (SSRGI)* and discussed it along with three case studies in Section 6.3. In Section 6.4, the flaws that may arise during SDP have been emphasized resulting in Secured Software Development Process (SSDP). A case study demonstrating SSDP is presented along with. The results and conclusion are drawn in Section 6.5. Finally, we conclude with the summary in Section 6.6.

### 6.2 Background

Studies have shown that errors made while gathering requirements contribute to 40% to 60% of all defects found in software projects [BP98]. The main reason being lack of user input, incomplete user requirements and changing requirements [NEI05]. Thus, gathering security requirements is the first step towards developing secured software. The requirements can be identified using the Software Requirements Specification (SRS), security assessment instrument, *SQUARE* methodology etc. [SRS98, MHS05]. Software security checklist ensures that all the security aspects are included during SDP [G+03]. As a part of security assessment instrument, a process has been mentioned which enables the developers to identify, analyze, and finalize the security requirements using software security components [HMS08].

*SQUARE* methodology implements security requirements in some discrete steps. It treats security requirements as add-ons to the functional requirements [MHS05]. Common Criteria (CC) allows for the establishment of
the methodology to apply security standards to an IT system or product. The methodology can be used at the architectural level of the security requirements [WAL, MZR07]. Most of the research deals with only the security requirements, enhancing existing security standards, or security to be considered as a non-functional requirement in SRS. It has also been revealed that security is not considered in detail while gathering requirements. Moreover, the stakeholders are usually not involved during security requirements gathering process. It identifies the need to develop an instrument that can be used to gather various types of security requirements from the stakeholders in detail.

Many efforts have been put forth to include security during software development process. SDL divides various security practices mapping to the software development activities [HL06]. Moreover, the organization can implement security during the development of software using the fundamental practices such as standard security toolkits, cryptography, threat modeling, misuse cases etc. [SIM11]. The Touchpoints or the best security practices for software security aim to fill the gap between the state of art and state of practice. Touchpoints such as code review, architectural risk analysis, penetration tests, abuse cases etc. can be followed during SDP [MAC06].

A report on secured development focuses on education and training; development, sharing and skillfully managing the product and processes; developing incentives to create security awareness; and making patching process simple [TFR04]. A software security initiative acts as guideline to relevant engineering activities and the knowledge required in achieving secured software [DHS06]. Another unified model, Secure Software Development Model (SSDM) integrates software security with security engineering. It separates the security specifications from the functional specifications [SOA06].

Secured software development can be based on security checklist and security assessment tools. The tools include Vulnerability Matrix (Vmatrix), flexible modeling framework (FMF) with model-based verification (MBV) and property-based testing (PBT) tool for the requirements specification, development
and testing phases of the life cycle [GKB00, GKPB01, GP02]. The efforts of the authors have been proved beneficial but most of the work is towards the development of specialized products. Some of them have considered security as specific process to be included in the SDP. It has been observed from the literature that the causes of insecurity or loopholes in the development process are not considered. Identifying the reasons for insecurity may help address security during development process of the software in a better way.

6.3 Security Requirements Gathering

Security requirements lay the foundation for development of secured software product. Requirements can be gathered with the help of SRS, brainstorming sessions, questionnaires, etc. It may also utilize attack trees, misuse cases etc. To gather security requirements systematically, we propose Software Security Requirements Gathering Instrument (SSRGI). The instrument considers various stakeholders during requirements gathering as developing secured software system is an organizational perspective. The instrument is integrated with IEEE 830-1998 SRS, so that security requirements can be considered along with other functional requirements. In the following subsections, SRS is discussed followed by our proposed SSRGI and its components. Further, SSRGI is integrated with SRS to gather security requirements along with the functional requirements. We then present case studies to illustrate SSRGI.

6.3.1 IEEE Standard 830-1998

IEEE Std. 830-1998 recommends the practice for Software Requirements Specification (SRS). It results in gathering unambiguous and complete requirement specifications. SRS helps customers, vendors, developers and other stakeholders to specify the requirements of the software for in-house and commercial products. It is described in three parts consisting of introduction, overall description and specific requirements along with additional information. The introduction delineates the purpose and overview of the SRS along with the
organization of rest of the SRS. The introduction contains following subsections viz. purpose; scope; definitions, acronyms, and abbreviations; references; and overview.

The second section of SRS describes the general factors that affect the product and provides background for the specific requirements. To understand the product and its requirements, overall description consists of six subsections viz. Product Perspectives, Product Functions, User Characteristics, Constraints, Assumptions and Dependencies, and Apportioning of Requirements. Product perspectives describe how software operates in various constraints such as System interfaces, User interfaces, Hardware interfaces, Software interfaces, Communication interfaces, Memory, Operations, and Site Adaptation requirements. Product function illustrates the major functions of the software product, while the user characteristics give an account of general features of the indented user such as educational level, experience and technical expertise. The constraints section focus on general description of the checks that developer should consider. Some of the constraints include regulatory policies, hardware limitation, interfaces to other applications, audit functions, parallel functions, safety and security considerations, and so on.

The specific requirements section of SRS must contain all the software requirements in sufficient detail. It may help the designers to incorporate these requirements in the design, while the testers can test the system to satisfy the mentioned requirements. The requirements must contain minimum description of input and output. It focuses on external interfaces, functions, performance requirements, logical database requirements, design constraints, software system quality attributes, and object oriented models. The external interface details all input into and output from the system. Functional requirements define fundamental actions in accepting and processing the inputs and generating the outputs. Some of the examples include fundamental checks such as validity checks, sequence of operations, error handling, etc. The performance requirements indicate the expected performance in numbers such as number of terminal supported, amount and type of information to be handled during normal
and peak hours, etc. Logical database requirements involve types of information used by various functions, frequency of use, data entities and interrelationships, etc. Standards and compliance specifies the requirements derived from existing standards or regulations. The software system attributes include Reliability, Availability, Security, Maintainability and Portability.

6.3.2 Software Security Requirements Gathering Instrument (SSRGI)

In this section, we propose a course of actions to gather security requirements. SSRGI starts with focus on the security policy that can support the identification of security needs and objectives. It may further help in deciding security requirements. We identified security requirements to be gathered from the different roles. The categories for security requirements can be Secure Functional Requirements (SFR), Drivers, Functional Security Requirements (FSR), Non-Functional Security Requirements (NFSR), Security Development Requirements (SDR), and Security Testing Requirements (STR). The different roles considered are customers, managers, designers, coders, and QA/testers. Figure 6.1 illustrates the proposed SSRGI that provides a systematic methodology to gather security requirements.

6.3.2.1 Security Policy

Security policy allows an organization to outline specific security requirements and set security practices and procedures. It must be met by the resultant software system so as to reduce the likelihood of attack. Policies may define roles and responsibilities of users, managers, designers, coders, and quality assurance team in achieving security objectives of the software system as well as the organization. It can help in safeguarding the business by maintaining confidentiality, integrity, and availability of data and information. An organization may develop its own and/or adopt security policies for securing the software resources. A number of security policies include regulatory acts for industries as Sarbanes-Oxley, HIPAA etc. The security policies identify the security needs and objectives.
Fig 6.1: Software Security Requirements Gathering Instrument (SSRGI)
6.3.2.2 Security Needs and Objectives

The identification of needs and objectives makes the understanding and development of comprehensive security plan easier for an organization. Security objectives define constraints that can fulfill the security goals for achieving confidentiality, integrity and availability of data and other resources. It may also include resource protection, authentication, authorization, non-repudiation, auditing security objectives etc. Further, the objectives define the roles along with the responsibilities or privileges.

6.3.2.3 Security Requirements

The needs and objectives decide the security requirements that have to be gathered from various roles viz. customer, manager, designers, coders, and quality assurance/ tester. Several types of security requirements may be gathered from these roles and are discussed below:

Secure Functional Requirements (SFR)

Secure Functional Requirements are security related description to be included in the functional requirements and is determined by the help of customers. It acts as constraints on the functional requirements. It specifies what shall not happen when software is executing. Such requirements can be gathered through misuse cases that capture negative requirements. The misuse cases counter measures the design decisions [SO01]. SFR may include constraints on sensitive data transmission such as use of Kerberos, logon and passwords (passwords should contain at least one special character), logout (logout of inactive user), account creation, submitting information to website (the data must be filtered to avoid spamming) etc. For example, the misuse case may depict the privacy issue of password being stolen while login data is in transit. It generates the need for secure data transmission. The customer expects to implement security measures as insecurity causes loss in monetary value.
Drivers

The drivers can determine the security needs of the software product as per the industry standards. The drivers for security requirements may include regulatory compliance, industry regulations and standards, company policies, and security features. Regulatory compliance may include Sarbanes-Oxley, HIPAA, Gramm-Leach-Bliley Act etc. Some of the industry regulations and standards are ISO 17799, OASIS etc. Company policies may include privacy policies, coding standards, patching policies, data classification policies etc. Security features include authentication and authorization model, role-based access control, and administrative interfaces etc. For example, Gramm-Leach-Bliley act requires that the financial companies should explain the information sharing practices to its customer and safeguard sensitive data. Thus, the drivers can be used by managers to decide upon the security requirements necessary for the product.

Functional Security Requirements (FSR)

Functional Security Requirements can be gathered with the help of the managers using the security drivers. It mainly focuses on the system under inspection. It can include requirements regarding authentication, authorization, backup, server-clustering, access control, encryption, data integrity etc. For example, managers must decide the intended users of the system along with the access rights, set up the backup policy, need of encryption etc.

Non-Functional Security Requirements (NFSR)

Non-Functional Security Requirements are usually the constraints on the functional requirements [LINK73]. These relate to the non-functional architectural requirements for security such as robustness, reliability, data authenticity etc. These can be derived from the architectural principals and good security practices as mentioned in SDL, Touchpoints etc. Some of the examples include security log, encryption, audit trail etc. In object oriented programming, creational, structural or behavioral design patterns can be used to generate NFSR. These also identify system’s resilience and level of immunity to attack [LLO05]. The designers may require NFSRs’ during the system design.
Secure Development Requirements (*SDR*)

It ensures that the resultant product is not vulnerable. The activities required during secured software development can be considered from coding guidelines. The coding guidelines such as *CLASP*, taxonomy of software security can be used by the coders to identify *SDR*. Some of the vulnerabilities mentioned in *OWASP Top 10* relate to coding flaws such as SQL injection and Cross Site Scripting (XSS). Since, most of the vulnerabilities owe to lack of secure coding, *SDRs* concentrates on coders mainly.

Security Testing Requirements (*STR*)

Secure Testing Requirements mainly focus on testing security requirements as mentioned in SRS. It tries to identify security vulnerabilities in a given environment. The QA/ testers needs to gather information about hardware architecture, software architecture, and user models to develop security test cases. *STR* may include the test cases to evaluate the effectiveness of the implemented security measures and controls. The security testers should also consider security use cases or software security functions as well as misuse cases. *STR* can act as security check list by the testing team. Some of the requirements to be tested are permissible multiple logons, time-out after inactive sessions, testing validity of login names and passwords etc.

6.3.3 Case Study

Our proposed *SSRGI* elaborates on gathering security requirements from the various stakeholders. To support gathering security requirements along with the functional requirements, *SSRGI* is merged in the system security attribute of SRS. Figure 6.2 shows the said integration of *SSRGI* into IEEE 830-1998 SRS. We describe three different cases illustrating Web-based, LAN based Client/server, and Single User system demonstrating the use of our *SSRGI*. The cases are selected on the basis of the diversity of requirements. In the cases, we elaborate on the product description and functions, assumptions, constraints, hardware and software requirements, memory requirements, etc. Further, we focus on gathering various security requirements with the help of *SSRGI*. An integrated approach of
Fig. 6.2: Integration of SSRGI into SRS

Specific requirements
- external interfaces
- functional requirements
- performance requirements
- logical database requirements
- design constraints
- software system quality attributes
  - reliability
  - availability
  - security
  - maintainability
  - portability
- object oriented models.
SSRGI with SRS has been presented for each of the three cases.

**Case I: Web-based System**

Journal Publishing System (JPS) is a web-based publishing system designed for scholarly researchers worldwide to be used by researchers, scholars, institutions, and other interested users. The system is designed to assist researchers and editors to automate the article submission, review process and publishing process thereby maximizing efficiency. The software facilitates the communication between authors, editors, and reviewers via preformatted reply forms and e-mail. The system maintains the database of the subscribers, contributors, reviewers, editors, and the articles along with the review report. The main objectives of JPS include accepting articles online and forwarding the articles to the reviewers according to the area of interest. Other objectives include e-mail acceptance/ modification/ rejection letter to the contributors based on the feedback from reviewers, accept the camera ready copy of the article if accepted/ modified as per the feedback via e-mail along with the publication fee and publish the article. The published article is made available to the contributor free of charge for 1 year. The journal can be subscribed for 6 months or 1 year based on the subscription fees. The system accepts subscription/ publication fee from the subscribers/ authors. The fee can be submitted through Visa/ Master credit card using online system, swift transfer or DD/ Cheque.

Considering it to be a moderately used system with approximately more than 10000 users and having more than 1000 users online in 5 minutes, minimum server configuration include 3.2 GHz Single Processor, Quad Core, 4 GB RAM and at least 300GB HDD. Web server required is Apache 2.x and above supported by Redhat Linux EL4 and above as operating system. The software is developed in PHP 5.x and above, with database as MySQL 5.x. The minimum requirement at client side includes Windows XP/ Windows 7 Professional, IE 7.0 and above/ Firefox 3.0/ Google Chrome, MS Office and Adobe Reader. The predefined messages can be sent using Outlook Express.

The various security requirements of JPS have been specified in security requirements section of SRS. The SRS of JPS explains the purpose and features of
the system, the interfaces, the outputs and the inputs along with the validity constraints, constraints under which the system shall operate, and the interaction of the system with the external environment. SFR includes logon page, password security for various users, password reset, account creation for new contributor and reviewer, contacting the editor, submitting information to website, preventing spamming of website submission, and backup of the data. JPS being a web-based project, the managers can take help of OWASP to gain knowledge about the current security vulnerabilities for gathering security requirements (NFSR and SDR). The managerial level focus on reliability, therefore decide the need for secure data transmission, and securing user identification for the contributors, reviewers and editors using Secure Socket Layer (SSL) protocol. According to the policies of the publisher, the authorization rights of read/ write/ delete can be granted to the users. Editors are permitted read/ write/ delete access while reviewers are provided read/ write access of the articles in the specialized research area. Once the article is published, read only permission is specified for all the users. But, the chief editor/ administrator may be allowed to change the file location.

The editors need to identify the unauthorized users who can pose threats to the system. The system also suffers from the technical and business impacts such as loss of revenues, loss of identity of the authors, loss of honorium of the reviewers, etc. FSR such as backups, authentication, authorization, access control, and encryption should be specified by the managers. The SDR can be specified by the developers using CLASP and OWASP. SDR should also specify the security related issues regarding formatting of data, exceptional conditions etc. To evaluate the security of the software, the tester specifies the security test plan. The test plan must contain the test procedures, software test reports, and acceptance criteria for the users in STR specification. The test plan shall address issues regarding password cracking, password reset, account lockout to prevent brute force attack, URL manipulation, session timeout, SQL injection, XSS, spoofing, spamming, etc.
It has been shown in the literature that one of the biggest problems of security is by placing the secure login boxes on insecure pages by 47% of the banks considered for the study. Also, 55% of the banks put contact information and security advice on insecure pages. As a result, an attacker could change the information and set up his own call center to gather private data from customers who need help from banks [LINK74]. To prevent such vulnerability, JPS indicates the need for SSL protocol as it gathers personal information from the authors and the subscribers. SQL injection tops the list of flaws from OWASP that affects the databases. It is caused by accepting untrusted data through an SQL query. SQL Injection attack can be avoided by checking and validating input to SQL statements. For example, concatenated SQL strings may suffer from SQL injection attacks, instead parameterized queries should be used. Such flaws can be avoided by the coders, so it should be included in the SDR. XSS affects the websites most that lacks the proper validation of the user supplied data to the active browser. To avoid such flaw, the system designer must include requirement of filtering the input from the sender. The testing requirements include manual code review along with the static and dynamic code reviews, and check the keywords that validate the input.

Case II: LAN based Client/ server System

Patient Management System (PMS) is a LAN based Client/ server system allowing the hospitals to maintain data regarding patients. The system also registers and maintains records of the doctors, nurses, dieticians, and social workers. PMS helps in the management of personalized patient lists and assists the doctors, nurses, dieticians, and social workers. The system generates the reports on patients along with the ailments, medications and test reports (if suggested), consulting doctors, and nurses in-charge. PMS permits to view the case history and fix appointment with the doctor. The system generates overall reports on birth and death records, test reports, diagnosis with medicines, and billing and payments intended for the hospital management.
Considering 100 patients daily including old and new ones, the minimum server configuration include 2.8 GHz Single Processor, Quad Core, 4 GB RAM and at least 300GB HDD. The software is developed using DotNet Framework with SQL Sever 2008 Express as database software. The client machine must be Intel Pentium Dual Core (E5800), 2 GB DDR3 SDRAM, 320 GB SATA Hard drive (7200 rpm) and Broadcom Integrated Gigabit Networking (BRCM 57780). The software requirements include Windows XP/ Window 7 Professional with DotNet Framework, while printers required are dot matrix and laser.

In PMS, SFR includes information regarding logon page, password security for the various types of users, password reset, account creation of new user etc. The company policy facilitates the decision regarding the access rights for all the users as the security threats are mainly from the internal users. Access control is one of the major design issues in a client/ server based system. Access control list can be developed by threat modeling. The designers must understand how access control works on the target platform. Depending on the security required, managers need to decide on the need of encryption and password strength. The requirements gathering team gathers security requirements from the managers by help of misuse cases. Misuse cases support the identification of threats to the system from authorized/ unauthorized users. The misuse case specifies the threat of stealing patient’s data violating privacy. The vulnerable points in the software must be specified and ensure that the unused features are disabled by default. The designers must specify the audit mechanisms to identify the intruders. Such mechanisms act as counter measure for misuse cases. Additionally, the designers can use CLASP to gather security requirements. The coders may specify SDR by help of software security taxonomy as well as mention exceptional cases and input/ output validations. The tester needs to state the test plans to evaluate security of the software, the test procedures, software test reports, and the acceptance criteria in STR specification. The test plans shall address code review, communication threats, password cracking etc.
Case III: Single User System

Shop Management System (SMS) is a single user system designed for a shopkeeper to keep track of the inventory, sales, and accounts, as well as prepares ledger and balance sheet. The system allows maintaining list of inventory available and required, daily/weekly/monthly sales, raises purchase indent, and maintains outstanding amount. It generates reports on daily/weekly/monthly sales, outstanding amounts, purchase details, list of suppliers, customers with addresses (for home delivery of goods) and the respective credit amount. For installing such a system, the minimum requirements include 2 GHz processor, 2 GB RAM, MS-Access, Windows XP with SP3 or Vista, Office 2003, 40 GB disk space and Broadband Internet connection with IE 7.0/IE 8.0/Firefox 3.0/Google Chrome, bill and laser printers.

The shop owner, being the only user of the system, is the administrator and has all rights to access the data. FSR includes logon and password facility as well as finger print recognition for accessing the computer system, and backup of data at regular intervals. Finger print recognition pocket device can be used to access the software to make it safe from the helpers in the shop. Although there are no threats form the outsiders (unless connected to Internet), the input and output needs to be validated for the correct working of the system. NFSR, SDR and STR are mainly the concerns for multi-user systems.

6.4 Secured Software Development Process

Secured software has been the result of secured practices that can be implemented during design and coding. Thus, secured development involves considering security during all phases of software development. The software development phases include requirements gathering and analysis, design, coding/implementation, system integration and testing, operations and maintenance, and documentation. In this section, one of the existing secured development processes is discussed. Further, our proposed secured software development process that considers security during all the phases is discussed. We also discuss the various
flaws generated during the software development stages along with the methods and tools to identify the flaws. A case study is presented that illustrates our secured software development process.

6.4.1 Existing Secured Development Process

Software practices followed during each stage of software development process can help improve software security. In a paper, business researchers discuss the policies adopted by the member organizations in developing secured software. It focuses on all the stages of SDP with main emphasis on design, development and testing [SIM08, SIM11]. The requirements phase targets on gathering security requirements using misuse cases, customer inputs, company policy, best practices, and security improvement goals. The design practices followed are threat analysis, free-form discussion and misuse-cases. The other design options include selecting standards, proven security toolkits and avoid building own security technologies and protocols. The secured design principles followed are threat modeling, least privilege, and sandboxing. The safe coding practices include minimize the use of unsafe string and buffer functions as in C and C++, validate input and output etc. The testing should be able to determine the attack surface and the robustness of the software product. The testing methods adopted are fuzz and penetration testing, and automated testing tools. The integrity principles further facilitates in achieving the secured software. The source code must be kept in protected control systems such as repositories to protect the active code. To discover potential threats and malicious activities, event and audit logs are to be monitored. Vulnerabilities generating code must be resolved promptly. The documentation should include the security posture and must contain certain information regarding how to securely configure the software.

6.4.2 Proposed Secured Software Development Process

The existing SDP lack the focus on security concerns of SDLC. Focusing on security concerns may support the development team to be more responsible towards the inclusion of security. It may support the development team during all
the stages of SDP. We propose security development process converging on various flaws generated during SDP that compromise security along with the tools to restrain the flaws.

**Software Requirements and Analysis**

The requirements can be gathered from strategic business level and the business area level. The major flaws resulting in insecurity are primarily due to the business level. The business level is generally not able to identify security issues as per the business needs. For example, requirements specification must state who shall be the authorized users along with the authorization rights. Poor security owes to the attack scenarios that may not have been understood well e.g. analyzing the ways an unauthorized user may break the security. The security requirements may consider the identification of roles and responsibilities based on governance and various security aspects as identified referring to SRS framework (refer Section 5.4). Mostly, requirements statements describe architectural and implementation constraints compared to security specifications. Also, the business levels as well as developers emphasize more on functional requirements rather than the non-functional requirements.

Specific security requirements must be gathered to effectively engineer secured software along with the constraints and validations for handling input and output. The popular tools to generate security requirements are misuse/ abuse cases and attack trees. The requirements gathering stage do not normally consider all the stakeholders. Therefore, security requirements can be gathered by the help of **SSRGI** (Section 6.3). Moreover, it is not clearly specified to the requirements gathering team as to what data is to be protected. The above mentioned problems are mainly due to lack of awareness of the security issues by the business owners. If security requirements are not explicitly stated in this stage, it may be overlooked in the design stage [CHA04]. Table 6.1 lists the above mentioned issues and proposes the actions and the tools to reduce insecurity due to requirements gathering stage.
### Table 6.1(a): Proposed Secured Software Development Process

<table>
<thead>
<tr>
<th>Steps</th>
<th>Security Issues</th>
<th>Proposed Actions</th>
<th>Tools required</th>
</tr>
</thead>
</table>
| **Software Requirements and Analysis** | *• All the stakeholders are not considered for security requirements  
  • Type of security required  
  • Attack scenarios are not understood well  
  • Specific security issues are not considered* | *• Brainstorming sessions of stakeholders may generate requirements.  
  • Security requirements must be expressed as technical features, assurances or operational practices.  
  • Must include requirements for constraints on input, passwords, account lockouts, expiring inactive sessions, data retention period etc.  
  • Must include requirements for protection from malicious attack and for system maintenance* | *• Abuse/ Misuse cases  
  • Attack trees  
  • Threat modeling/ analysis* |
| **Software Design**    | *• Security is not considered as prime objective due to lack of knowledge of various security concerns such as security principles, guidelines and attack patterns  
  • Least permissions not taken during requirement phase  
  • Sensitive data not encrypted  
  • Security aspects associated with acquired or reused software, if used any, is not addressed  
  • Not considering platform flaws* | *• Map the attack patterns into the design  
  • Assume all data to be malicious and hence validate all input  
  • Error handling and exception handling must be included in design  
  • Unused components of reused software components must be removed and thereby avoiding its misuse  
  • Error pages should not leak important information  
  • Evaluate system design for security by testing it with help of test cases  
  • Recognize the security implication of various platform and hence design the software  
  • Incorporate the failure mechanisms as well as mechanisms to survive attacks* | *• Check the design with the attack patterns  
  • Human analysis  
  • Design testing tools  
  • Perform risk assessment  
  • Documentatio n of the risk analysis and test cases* |

Cont…
<table>
<thead>
<tr>
<th>Steps</th>
<th>Security Issues</th>
<th>Proposed Actions</th>
<th>Tools required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding / Implementation</td>
<td>• Improper error handling, illogical access control.</td>
<td>• Programming language chosen should effectively manage memory and avoid the use of pointers</td>
<td>• Check the code with the code checkers</td>
</tr>
<tr>
<td></td>
<td>• Not able to code the security requirements as per the design</td>
<td>• The input from cross language boundaries must be handled very carefully</td>
<td>• Apply coding and testing standards</td>
</tr>
<tr>
<td></td>
<td>• Initialization errors, incorrect input validation, API abuse, buffer overflows, race conditions etc.</td>
<td>• The memory should be released explicitly after deallocation of objects</td>
<td>• Manual review</td>
</tr>
<tr>
<td></td>
<td>• Use of unsafe functions in C/ C++.</td>
<td>• Compiler generated warnings should be taken care of.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Lack of education about the coding flaws</td>
<td>• Train the programmer to use unsafe functions efficiently.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Not utilizing security features of the platforms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Integration and Testing</td>
<td>• Many test methods do not take architectural risk analysis into account</td>
<td>• Test the software for all its functionality as per the design</td>
<td>• Traceability matrix</td>
</tr>
<tr>
<td></td>
<td>• Most of the test methods are used after integration of all modules</td>
<td>• The software should be tested for non-regular conditions such as low memory, insufficient privileges, interrupting a transaction etc. to identify the unexpected behavior</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Most quality assurance organizations focus on features rather than security</td>
<td>• Test the most exposed entry points and worst case scenarios.</td>
<td>• Penetration testing</td>
</tr>
<tr>
<td>Operations and Maintenance</td>
<td>• Insecure configuration of Operating System, databases, networks etc.</td>
<td>• A list of allowable commands must be created during runtime and only allow mentioned commands</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Changes in the system or environment</td>
<td>• The servers must use secure channels for data transfer</td>
<td>• Fuzz testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ensure secure configurations based on user roles</td>
<td>• Ballista Method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Automated tools</td>
</tr>
<tr>
<td>Documentation</td>
<td>• Lack of uniformity between the design, code and documentation</td>
<td>• Document should contain the instructions for security controls and the responsibilities of the various users</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Documentation must be updated after every modification</td>
<td>• Risk assessment and testing of the changed area using testing tools</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Educate the users on the security aspects</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Programmers must use the traceability matrices to identify the cause of flaw in changed area</td>
</tr>
</tbody>
</table>
Software Design

The design step identifies the functional and architectural design elements. The functional design elements include hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudocode, and a complete entity relationship diagram with full data dictionary. The design elements describe the software in sufficient detail so that skilled programmers may develop right software as per the specifications.

Security architecture is regarding what the software is not expected to do. The design should be inline with the security architecture and must able to reduce the attack surface of the software. It may help in making the software system attack resistant and tolerant. For example, design should include Access Matrix, Access Control List or Role-Based Access Control to identify the access rights of authorized users. The designer must have high level of knowledge and experience to develop proactive design. No specific flaws of design stage are addressed in the literature. Still, number of design flaws exist as security is not considered to be the prime objective. It may be due to the lack of knowledge regarding the security concerns such as security principles, security guidelines, and attack patterns [A+08]. Design flaws may be the result of not considering PoLP. To develop secured design, various attack patterns must be identified and mapped into the design. The design should also address security aspects associated with acquired or reused software, if any, so that unused components shall be isolated and its misuse can be avoided. For example, in a Kiosk based system, file creation may not be allowed on the system. A number of security defects are present in each platform. Failure to recognize the security implications of different languages, tools, or techniques and how it can be used when implementing the software may provide some level of insecurity [DHS06]. Insecurity exists as the potential threats are not taken care of in the design.

To have secured system, every input must be checked against pre-defined constraints. It may ensure correct input to the system and may reduce
the chance of exploitation. When the software interfaces with external entities, error handling and exception handling must be incorporated in the design of the software. The system design should be able to resist the attack patterns when provided as input. It must be evaluated for security by conducting a series of tests developed for user inputs that lead to insecurity. It can help to recover the system if some security breach does occur. A number of security flaws have been identified and the measures to avoid them are mentioned in Table 6.1. The flaws in design stage can be identified mainly by human analysis along with testing tools such as Alloy. Proper documentation of the risk analysis and all the test cases may help in testing the system against the future defects.

**Coding/ Implementation**

During coding/ implementation, the developers write the code, test it and integrate the modules. The major security issues include illogical access control, breaking the design into different modules without considering the after effects and not able to code the security requirements as per the design. The coding flaws may attribute to use of unsafe functions, improper use of pointers, allocation and deallocation of memory, initialization errors, incorrect input validation, buffer overflows, improper error handling, race conditions etc.

A number of practitioners and researchers have mentioned various coding flaws that need to be addressed while developing secured software system. Some of the coding flaws include API abuse, injection, buffer overflow, etc. [TCM05, OWASP11, CWE11]. Such lists create awareness among the developers regarding prevention of security vulnerabilities. A thorough understanding of the coding errors and programming efficiency can avoid most of the coding flaws.

Normally, the code is tested for functionality using unit testing, module testing, validation testing, and integration testing. Coding errors can be identified using automatic code checking tools as it is not possible to check all the combinations of input/ output and the final conditions [KFN99]. Some of the static code analyzers to detect common programming errors are RATS,
Flawfinder, ITS4 etc. The static code checkers are aimed at identifying only specific set of flaws. Therefore, manual review of each module is necessary that can be conducted by trusted third party. While testing the code, the exposed code must be reviewed first. The other reasons for coding flaws and the flaw avoidance tools are listed in Table 6.1.

**System Integration and Testing**

System integration helps in transforming the subsystems into an application system. The security flaws at this stage arise due to insecure configuration of operating system, communication links, databases etc. After system integrating, the software and hardware are tested to ensure that the system meets the requirements, is implemented as per the design and works as expected. It is the final step towards all the unaddressed issues of the errors and negligence occurred in the above steps. The defects identified are the result of escaped flaws during the previous software development phases; hence the testing should be applied during all the phases of software development. The testing process helps to judge the quality of the end-product. Security testing aims to identify the robustness of the software, i.e. the data and resources are secure, and continue to function in an acceptable manner even in case of attacks from unintentional/ malicious users, hackers, and other attackers [A+08].

Security test strategies should be determined by security specifications from requirements phase. The test cases can be determined using threat models, attack patterns, misuse/ abuse cases, and requirements specifications and design. Flaws due to functional as well as non-functional requirements can be identified using white-box and black-box testing techniques. Security testing techniques include penetration testing for testing real time software to ensure that it can resist attacks; fuzz testing to identify design and implementation defect; and ballista testing for exception handling. The test cases should be designed to test exposed entry points, as well as worst case scenarios. The flaws may remain unidentified in the software even after testing as the black box testing may not
take architectural risk analysis into account [TFR04]. The other testing issues and the tools used for testing are mentioned in Table 6.1.

**Operations and Maintenance**

Operations provide a product or service to end users. The activities performed during operations include performing backups, organizing training sessions, monitoring user administration and access privileges, updating security software, implement secure settings etc. A list of allowable commands created during run-time can prevent use of any command other than mentioned in the list [CWE09]. The allowable commands should be associated with the access controls of the users. Further, security protocols can be used for secure communication by the servers to ensure security during the operation of software system.

Software maintenance is defined as the process of modifying a software product, component, or system after delivery and installation to correct faults, improve performance or other attributes, or adapt to the changed environment. In other words, maintenance includes changes and enhancements in the system. Even if developers have taken enough care during design and development of a system, errors may be introduced during maintenance as the human errors are unavoidable. The errors are result of changes in system or environment. During maintenance, if major changes are made, the software must be analyzed thoroughly for security. Risk assessment and testing, especially of the changed area, must be performed to assess the quality and security of the software. For large software, it can be very costly and time taking. Other approach is to test the software (after all the changes have been made) by making use of old as well as new test cases for the changed area. It may make identification of vulnerability difficult for the tester owning to the changes made in the software. For minor changes, informal evaluation of the software may be sufficient. The testers may use traceability matrix to ensure that all the security requirements have been tested. Users must be educated regarding the security aspects of the
system that includes specific security controls and user responsibilities associated with using the system.

**Documentation**

Documentation emphasize on working of the software, configurations etc. Other than the working of the software, the document should contain the instructions for the security controls and the user responsibilities according to the roles of the users. The software development team can use traceability matrices to maintain consistency between requirements, design, code and documentation [ESA95]. It can also be used for maintaining the modifications in the software product during maintenance. The security issues and the proposed actions to incorporate security during this phase are listed in Table 6.1.

**6.4.3 A Case Study**

In this section, we presented secured SDP that helps to integrate security during the various phases of software development. Security issues of the SDP phases along with certain actions and tools to combat security are also suggested. To study SSDP, we present a case study covering the security issues of various stages of software development process. The security requirements are gathered considering SSRGI and SPS framework. SSRGI and SPS framework are discussed in Sections 6.3 and 5.4 respectively. The design utilizes security requirements gathered from previous stage and utilizes secured architecture/ framework. The coders code the software on the basis of design and implements considering the security objectives. The software is integrated and tested for secured functioning as per design and requirements. Once the software is made operational, the access controls must be updated while the software must be tested in case of any changes/ upgradation made in the software. In this regard, we present a case study on Journal Publishing System (JPS) illustrating our secured software development lifecycle. The development team of JPS gathers security requirements with the help of SSRGI (Section 6.3.2) and focuses on SPS framework (Section 5.4) for secured
development. We elaborate on the security aspects as well as tools to be considered during all the stages of software development.

**Security Requirements and Analysis Phase**

The development team along with the JPS team determines the security requirements with the help of SSRGI through the brainstorming sessions/ interviews/ questionnaires. The security requirements are based on confidentiality, integrity and availability. The password reset facility utilizes security question and the e-mail id through which the new password can be set. The JPS team decides to implement separation of duties. Security requirement may also include means to identify unauthorized users and the malicious attacks. Such requirements have been gathered by misuse cases to identify possible threats, attack trees to model threat against computer systems, and threat analysis to support in risk analysis. E.g. JPS can suffer attack from the ‘Failed Password’ functionality of the system as shown in Fig. 6.3. The security requirements include secure transmission channel for avoiding sniffing attack, input validation to avoid SQL injection, and delay of five seconds to reenter password and maximum three attempts to valid login to avoid brute force attack. The journals can be subscribed through the online payment system using credit card. Therefore, the bank uses payee registration confirmation for security and uses secure channels for data transmission.

**Software Design**

The design of JPS includes validating all inputs such as logon and password information, personal details etc. Moreover, the design of JPS considers mechanisms to secure e-mails, and combat platform flaws. E-mail needs to be secured through SSL/ TLS POP/ SMTP as the correspondence between the editors, researchers and reviewers is through e-mail. It ensures confidentiality and authenticity of the e-mails. Separation of duties is implemented by deciding the roles and responsibilities of various users. The software is developed in PHP as it is free and easy to use which brings down the development cost significantly. PHP supports HTML5 and above, and output
**Misuse Case:** Attack on Failed Password functionality

**Summary:**
An attacker tries to gain access to the JPS by using Forgot Password functionality and acquiring a new password.

**Author:** xxxxx

**Date:** xxxxx

**Possible Attacks:**
- SQL Injection
- Brute force attack to gain access
- Sniffing attack on e-mail sent for password recovery

**Assumption:**
- An attacker can attack anytime.
- An attacker is an unauthorized malicious user.

**Worst Case Scenario:**
Attacker gains access to JPS and downloads all the research articles without paying.

**Post condition:** The attacker cannot gain access to the journal database.
An attacker cannot identify the e-mail address of the authorized users/subscribers.

---

**Fig 6.3: Example of Misuse Case for ‘Failed Password’ Facility**
pdf and text files providing better integration to web applications. The PHP related vulnerabilities are listed in National Vulnerability Database. As a result, advanced protection patches such as Suhosin and Hardening-Patch are available to check errors. Besides, the PHP web applications can be protected by use of software such as PHPIDS from external third party threat. Design checking tool Alloy has been used for checking the design. Alloy uses high level (summary) coding notation of the design to check errors and identify unusual conditions. Additionally, the design is checked manually by peer group to identify the design flaws.

**Coding/ Implementation**

JPS may suffer some of the coding errors due to PHP. The coding errors may relate to SQL Injection, XSS, session hijacking, etc. SQL injection can be avoided by PDO defined statements, while XSS and session hijacking by validating input. Session hijacking can be prevented by changing the session id often by running session_regenerate_id() function and notifying the client. The security issues due to remote file being included can be reduced through proper settings of the flags in php.ini. Static code analysis tool Pixy has been used to detect SQL injection and XSS related flaws. In addition, the code is reviewed manually to check for the flaws.

**System Integration and Testing**

Based on security requirements, design and common attacks, a number of test cases have been developed. The test cases evaluate the software for regular and non-regular conditions to identify unexpected behavior. The test cases should map the requirements and design by maintaining traceability matrix. Some of the sample test cases are described below:

**Test Case I: Login to test valid end user.**

It is expected that the home page should be displayed with appropriate access to the journal articles.

After execution of SSDP, it results into access to only authorized users.
**Test Case II**: Test the time period of inactive session of the valid end-user.

It is expected that the inactive session must be valid only for 15 min.

After execution of the test case for SSDP, it results in session expire after 15 minute of inactivity.

**Test Case III**: Test for XSS by validating input

The Expected result is that the un-trusted data is not accepted except for allowed remote locations thus preventing XSS.

After execution of the test case for SSDP, system is able to defend against XSS attacks.

**Test Case IV**: Test for SQL injection by checking the input to dynamically generated queries.

The result being expected is that the probable information is passed into the query to prevent SQL injection.

After executing the test case, SQL injection is prevented.

The penetration testing tools are used to test the software for security e.g. netcat evaluates the networks by testing the software system against all network key connections, sensitive data, and key access points. Further, it determines the unauthorized access to file systems and data. The software system has been fuzz tested by random input. The system passed fuzz testing as it did not hang or provided unexpected results.

**Operations and Maintenance**

During operational phase, the developers provide maintenance till the software is finally accepted and then the maintenance department becomes responsible for maintaining the system. Security flaws identified during operations are categorized on the basis of the security domains identified in Chapter 3. It enables the identification of the reasons for flaws and prioritizes the testing efforts. Besides, it supports the development of metrics that help in risk assessment based on security posture of the organization. Some of the
metrics include number of virus attacks per month, number of failed login attempts, number of audits per year, ratio of firewalls to access points etc. Further, the allowable commands are listed in the server configuration file. Each command is also associated with access control list for authorization. Moreover, the users are educated on security and safeguarding the system from unintentional breaches.

Documentation

The security aspects, access control lists, secure configuration, secure transmission and traceability matrix are maintained as part of documentation. The traceability matrix supports in mapping the approved written security requirements to all phases of software development; while changes and enhancements maps to the new requirements as well as updated test cases.

6.5 Results and Conclusion

Specific security requirements form the basis for secured software. In this regard, we identified 6 types of security requirements viz. Secure Functional Requirements, drivers, Functional Security Requirements, Non-Functional Security Requirements, Secure Development Requirements and Secure Testing Requirements. A software security requirements gathering instrument is developed to gather security requirements with the help of various stakeholders. The security requirements are generated on the basis of organization’s policies as well as needs and objectives. Exploratory case study based research has been conducted to analyze SSRGI for web enabled, client/server and single user systems. Based on the case studies, following observations can be made:

- Web enabled software has the highest need for security requirements as it has a large number of users not limited to geographical area. It mainly suffers threats from unauthorized malicious users.
• **SSRGI** helps to gather requirements on the basis of the industry standards, identification of unauthorized users, need to use secure communication channel, secure payment policy etc.

• A client/server based system suffers threats from internal users. Hence, **SSRGI** focuses on gathering requirements for access control based on the user roles, secure internal communication, password strength, backup policy etc.

• In a single-user system, **SSRGI** helps to gather requirements for password, backup etc.

• It has been observed that our **SSRGI** is flexible and it enables gathering security requirements in a systematic manner from various stakeholders.

• Moreover, **SSRGI** enables gathering security requirements along with the functional requirements when integrated with SRS.

Further, a general view of incorporating security has been developed that considers security during all the stages of software development leading SDP to SSDP. SSDP discusses security issues during the various stages of software development, proposed actions for dealing with the security issues and tools required to analyze security. The case study leads to the following observations:

• The case study emphasizes that misuse cases, attack trees, threat analysis and **SSRGI** can help gather security requirements in an effective manner.

• The design can be checked for security using attack patterns, design testing tools and human analysis.

• By exploiting the security features of the development platform, the coding flaws have been addressed in a systematic manner.
After integration, the software is tested on the basis of security requirements and design.

During the operations phase, the vulnerabilities can be reduced by permitting only the list of permissible commands. The software must be tested for security during the maintenance phase by testing the access rights and the changed part of the software or whole as required.

The inconsistencies among the requirements, design and implementation can be located using traceability matrix.

Finally, it can be observed that our SSDP focus on security issues related to each development phase and helps to overcome them by use of appropriate actions and tools.

Security flaws in a software product may destroy customer value and trust which leads to the need for secured software. The backbone of secured software is security requirements. We developed SSRGI to gather security requirements from various stakeholders on the basis of organization’s security policy and the security needs and objectives. On the basis of case studies, it can be revealed that SSRGI shall ensure systematic and detailed security requirements along with the functional requirements. SSRGI also allows gathering security requirements for different kinds of software as illustrated in Fig. 6.4. Further, the software vulnerabilities can be prevented by understanding the causes for introduction of flaws during various stages of SDP. Additionally, it includes rigorous set of reviews for design, implementation and testing stages that may enable finding unexpected logic, exploitable faults, and other weaknesses. Code reviews, security tests, and strict configuration control must be continued during the post-release support phase. Our SSDP focuses on security issues of each of the phases of software development along with suggested actions and tools to combat security. It may ensure systematic security consideration throughout SDP. Also, updates and patches do not add security weaknesses or malicious logic to existing software products.
Fig. 6.4: SSRGI for Various Case Studies
6.6 Summary

Lack of security requirements is one of the major causes for insecure software system. In this Chapter, we developed Software Security Requirements Gathering Instrument, SSRGI, that aims to frame security policy, identify needs and objectives, and determine requirements from the various stakeholders. The instrument can help to gather various types of security requirements viz. Secure Functional Requirements, drivers, Functional Security Requirements, Non-Functional Security Requirements, Secure Development Requirements and Secure Testing Requirements from the stakeholders. SSRGI is integrated with SRS so that security requirements can be gathered along with the functional requirements. Exploratory case study based research on SSRGI reveals that it can systematically gather detailed security requirements. It is also flexible and easy to use. Further, the developers might unknowingly inject flaws during Software Development Process (SDP), therefore security must inform all the phases. We developed secured software development process that focuses on understanding the causes for insecurity during various stages of SDP. It proposes actions for incorporating security and suggests tools required to combat security issues. Case study has been conducted to illustrate the inclusion of security during Secured Software Development Process (SSDP). It has been revealed that SSDP promotes systematic security consideration throughout the development stages. It may ensure that updates and patches do not add security weaknesses or malicious logic to existing software products.