SECURITY IN THE SOFTWARE DEVELOPMENT LIFECYCLE

Swati Kaushik*1, Tanu Mohan2

1M.Tech. Scholar, Bhagwant Institute of Technology, Muzaffarnagar, India.
2Computer Science & Engineering, Uttar Pradesh Technical University, Lucknow (U.P), India.

ABSTRACT

The purpose of this paper is to help you understand the important role that security plays in the Software Development Life Cycle (SDLC). The paper defines security as it applies to the SDLC and discusses overall SDLC security issues. It then covers each phase of the SDLC and specific security controls and issues for each phase. Note that the SDLC acronym is also used to represent System Development Life Cycle. In many cases, a decision is made to purchase or outsource the software and associated hardware and network systems needed to implement a new application. This is often referred to as the “buy or build” decision. Buy means you will purchase a finished product from a vendor and build means you will hire someone to develop it for you. This paper focuses on the Software Development Life Cycle, but most of the phases, terms, and issues discussed apply to both the “buy” and “build” process.

Keywords: Integrity, sdlc, threats, proxy, network, security protocols.

1. INTRODUCTION

When defining security in the SDLC, two areas must be addressed. The first area is the SDLC process itself. The second area is application operational security. You must understand the SDLC process and associated security activities and the specific application and operational security controls that are available to the application designer. Software is ubiquitous. Many functions within the public and private sectors are highly dependent on software to handle the sensitive and high-value data on which people’s privacy, livelihoods, and very lives depend [1]. National security—and by extension citizens’ personal safety—relies on increasingly complex, interconnected, software-intensive information systems—systems that in many cases use the uncontrolled Internet or Internet-exposed private networks as their data bus. Software-intensive systems run the nation’s critical infrastructure- electrical power grids, water treatment and distribution systems, air traffic control and transportation signal systems, nuclear, biological, and chemical laboratories and manufacturing plants, etc [2]. Those systems, too, are increasingly being interconnected via the Internet. Security systems for banks and prisons are similarly software-intensive and networked via the Internet.

2. SECURITY IN THE SDLC PROCESS

The SDLC process consists of six phases (discussed in detail later). In each phase, specific security related activities take place to ensure that security is built into the software system under development. For example, in one phase, SDLC Project Initiation Phase, a security related activity is to define the sensitivity of the information that the software system will process [12]. By placing this security related activity early in the SDLC process, later decisions are made based on the security needs of the organization and not as an afterthought.

The objective of software security is to design, implement, configure, and support software systems in ways that enable them to-

i- Continue operating correctly in the presence of most attacks by either resisting the exploitation of faults or other weaknesses in the software by the attacker, or tolerating the errors and failures that result from such exploits.
ii- Isolate, contain, and limit the damage resulting from any failures caused by attack-triggered faults that the
software was unable to resist or tolerate, and recover as quickly as possible from those failures [7].

3. APPLICATION OPERATIONAL SECURITY

As the development team moves through the phases of the SDLC, decisions are made to add security controls to the application to ensure the proper protections to availability, integrity, and confidentiality. These application and operational controls can be administrative controls, physical controls, or technical controls [6]. An example of an administrative control designed into the application in the Operations and Maintenance Phase is Separation of Duties.

Several lower-level properties may be seen as attributes of security as a software property-

i- Availability: The software must be operational and accessible to its intended, authorized users (humans and processes) whenever it is needed.

ii- Integrity: The software must be protected from subversion. Subversion is achieved through unauthorized modifications by authorized entities, or any modifications by unauthorized entities. Such modifications may include overwriting, corruption, tampering, destruction, insertion of unintended (including malicious) logic, or deletion. Integrity must be preserved both during the software’s development and during its execution [5].

iii- Confidentiality: In the context of software security, confidentiality applies to the software itself rather than to the data it handles. Confidentiality for software means either its existence, its characteristics (including its relationships with its execution environment and its users), and/or its content are obscured or hidden from unauthorized entities, most often to prevent them from learning enough about it, e.g., through reverse engineering, to craft effective attacks against it [4].

Two additional properties commonly associated with human users are also required in software entities that act as users (e.g., proxy agents, Web services, peer processes). These properties are-

iv- Accountability: All security-relevant actions of the software-as-user must be recorded and tracked, with attribution of responsibility. This tracking must be possible both while and after the recorded actions occur. The audit-related language in the security policy for the software system should indicate which actions are considered “security relevant” [3].

v- Non-repudiation: Pertains to the ability to prevent the software-as-user from disproving or denying responsibility for actions it has performed. Non-repudiation measures were originally intended to ensure that users could not deny having sent or received email messages. However, the value of non-repudiation for activities other than message transmission is being increasingly recognized [9].

4. SDLC SECURITY ISSUES

The goal of a good SDLC process is to capture, verify, and implement all the requirements needed to make the application useful to the organization. These requirements include security needs defined around confidentiality, integrity, and availability of the information system [6]. If security requirements are correctly identified and the proper security controls added are to the application to meet these requirements, the result is a secure application. But in reality, developing applications involves trade-offs to meet budget, resource, and time constraints placed on the project. In many cases security is the first requirement to be dropped. Security issues directly caused or affected by the unique aspects of embedded software, firmware, and real time software. Such characteristics include the unique computational models, operational constraints, development languages, tools, and practices, etc., used in the creation and operation of such software [9]. These unique aspects may make it difficult for the creators of these types of software to apply some of the methodologies, principles, and practices described here. Embedded software in particular may not be subject to the same threats as other software. Moreover, it may be subjected to its own unique set of threats.

Software is subject to two general categories of threats-

i- Threats during development (mainly insider threats): A “rogue” developer can sabotage the software at any point in its development life cycle, through intentional exclusions from, inclusions in, or modifications of
the requirements specification, the threat models, the design documents, the source code, the assembly/integration framework, the test cases and test results, or the installation/configuration instructions and tools. The secure development processes, methodologies, and practices described in this document are, in part, designed to help reduce the exposure of software to insider threats during its development process.

**ii-Threats during operation (both inside and external threats):** Any software system that runs on a network-connected platform is likely to have its vulnerabilities exposed to attackers during its operation. The level of exposure will vary depending on whether the network is public or private, Internet-connected or not. The larger, more “open” and uncontrolled the network, the more exposed the software will be to external threats. But even if the network is private, small, and closely managed, there may still be a threat from untrustworthy elements in the software’s authorized user community (i.e., “malicious insiders”).

**5. SDLC PHASES**

**i- Requirement Gathering and Analysis** - In this phase business requirement are gathered. This phase is the main focus of the project managers and stack holders. Meetings with managers, stake holders and users are held in order to determine the requirements like, who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirement gathering phase [8]. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model [13].

**ii- Design Analysis** - In this phase the system and the software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model [9].

**iii- Implementation/Coding** - On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of Software Development Life Cycle.

**iv- Testing** - After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During the phase unit testing, integration testing, system testing, acceptance testing are done [14].

**v- Deployment** - After successful testing the product is delivered/deployed to the customer for their use.

**vi- Maintenance** - Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance [10].

**6. SUMMARY**

If companies follow the SDLC phases and incorporate the appropriate security activities in each phase, systems can be developed in a secure manner. If security is not built in to the system in this manner, the security shortcomings of the system will be discovered. IT auditors might find the problems and dictate that costly
system changes be implemented. Or, attackers might find the problems and cause damage to company assets and reputation [7]. Due to the current state of security measurement capabilities, it is difficult to directly assess the degree to which a particular methodology, process, practice, or technology affects a software-intensive system’s security, but many provide enough information about existing “security-enhanced” practices and security-minded approaches to general practices to enable the reader to begin identifying and comparing potential new practices that can be adopted and/or adapted to augment, adjust, eliminate, or replace his/her organization’s current non-secure practices.

REFERENCES