(IJISE) 2016, Vol. No. 4, Jul-Dec **e-ISSN: 2454-6402, p-ISSN: 2454-812X**

A SECURITY APPROACH IN SYSTEM DEVELOPMENT LIFE CYCLE

Mathan Kumar M[1], Dr. Anu Bharti[2]

[1] *Research Scholar, Dept. of Computer Science & Engineering, Sunrise University, Alwar*

[2] *Asso. Prof., Dept of Computer Science & Engineering, Sunrise University, Alwar*

ABSTRACT

Many software organizations today are confronted with challenge of building secure software systems. Traditional software engineering principles place little emphasis on security. These principles tend to tread security as one of a long list of quality factors that are expected from all professionally developed software. As software systems of today have a wide reach, security has become a more important factor than ever in the history of software engineering can no longer be treated as Separate Island. There is an imperative necessity to incorporative security into software engineering. Incorporating security into software engineering necessitates modification of existing software engineering principles, as these have to be tailored to take into account the security aspect. All phases of software engineering are likely to be impacted. In this paper we tried a novel security mechanism in system development life cycle.

Keyword: *Security, Design Phase, SDLC.*

INTRODUCTION

In the software industry that requirements engineering is critical to the success of any major development project. Security requirements are identified during the system development lifecycle. However, the requirements tend to be general mechanisms such as password protection, spam and Phishing detection tools. Often the security requirements are developed independently of the rest of the requirements engineering activity, and hence are not integrated into the mainstream of the requirements activities. As a result, security requirements that are specific to the system and that provide for protection of essential services and assets are often neglected. The requirements elicitation and analysis that is needed to get a better set of security requirements seldom takes place.

Users may not have aware of the security risks, risks to the mission and vulnerabilities associated with their system. To define requirements, systems engineers may, in conjunction with users, perform a top-down and bottom- up analysis of possible security failures that could cause risk to the organization as well as define requirements to address vulnerabilities. Fault tree analysis for security is a top-down approach to identifying vulnerabilities. In a fault tree, the attacker's goal is placed at the top of thetree.

(IJISE) 2016, Vol. No. 4, Jul-Dec **e-ISSN: 2454-6402, p-ISSN: 2454-812X**

Then, the analyst documents possible alternatives for achieving that attacker goal. For each alternative, the analyst may recursively add precursor alternatives for achieving the sub goals that compose the main attacker goal. This process is repeated for each attacker goal. By examining the lowest level nodes of the resulting attack tree, the analyst can then identify all possible techniques for violating the system's security preventions for these techniques could then be specified as security requirements for the system.

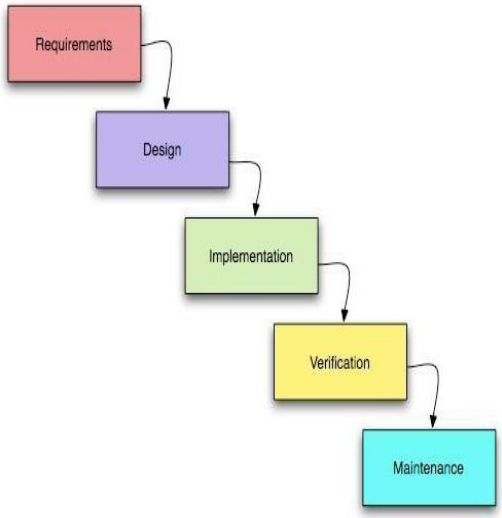
Failure Modes and Effects Analysis is a bottom-up approach for analyzing possible security failures. The consequences of a simultaneous failure of all existing or planned security protection mechanisms are documented, and the impact of each failure on the system's mission and stakeholders is traced. Other techniques for developing system security requirements include threat modeling and misuse and abuse cases. Requirements may also be derived from system security policy models and system security targets that describe the system's required protection mechanisms.

The SQUARE process involves the interaction of a team of requirements engineers and the stakeholders of project. The requirements engineering team can be thought of as external consultants, though often the team is composed of one or more internal developers of the project. When SQUARE is applied, the user should expect to have identified, documented, and inspected relevant security requirements for the system or software that is being developed. SQUARE may be more suited to a system under development or one undergoing major modification than one that has already been fielded, although it has been used both ways. Software life-cycle models describe phases of the software cycle and the order of execution of those phases. Many models are being adopted by software companies, but most of them have similar patterns. Typically each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during the implementation phase and is driven by the design. Code is finally tested against requirements to ensure quality. In this paper we were implemented some security principles in Waterfall method.

PROPOSED METHOD

The Waterfall Model is the old method of structured system development. It's the base for all models although it has come under attack in recent years for being too rigid and unrealistic when it comes to quickly meeting customer's needs and development, the Waterfall Model is still widely used because of easy model for developers. It is attributed with providing the theoretical basis for other Process Models, because it most closely resembles a generic model for software development.

(IJISE) 2016, Vol. No. 4, Jul-Dec **e-ISSN: 2454-6402, p-ISSN: 2454-812X**



The Procedure of Waterfall Model for software development:

- **System Requirements:**

System Requirement refers to the consideration of all aspects of the targeted business function or process, with the goals of determining how each of those aspects relates with one another, and which aspects will be incorporated into the system. What is the essential thing needed in developing system

- **System Analysis:**

This step refers to the gathering of system requirements, with the goal of determining how these requirements will be accommodated in the system. Extensive communication between the customer and the developer is essential. Developer has to understand the exact requirement of user.

- **System Design:**

Once the requirements have been collected and analyzed, it is necessary to identify in detail how the system will be constructed to perform necessary tasks. More specifically, the System Design phase is focused on the data requirements, the software construction and the interface construction.

- **Coding:**

Allies name is programming, this step involves the creation of the system software. Requirements and systems specifications from the System Design step are translated into machine readable computer code.

- **Testing**

As the software is created and added to the developing system, testing is performed to ensure that it is working correctly and efficiently.

In every system development has its own strategy and methodology but where lies the security aspect in system development life cycle? This is the very big problem in system development in this paper

(IJISE) 2016, Vol. No. 4, Jul-Dec **e-ISSN: 2454-6402, p-ISSN: 2454-812X**

we provide some security methods to enhance the water fall model. **Security in Design Phase:**

In every phase we have to include the security enhancement. Even though it's very essential for every phase depth security features needed from the design phase onwards. In requirement gathering phase information fetching mostly happen between user and developer due to that need of security level will be less. Apart from the user collecting information should be trust worthy as well as valid information should be taken for the system development. In analysis phase, the information's what we obtain from the requirement phase that will give to the analysis phase. Collected information will be analysis used some valid documents materials, white papers, existing methods, etc.

Information grouped into the structure form from the unstructured form. In the analysis phase itself we have to estimate what kind of security requirements need for our system. Security elements and features should be included in every aspect of the system like user, data, module, design, testing, etc.

Developers need to know secure software design principles and how they are employed in the design of resilient and trustworthy systems. Two essential concepts of design include abstraction and decomposition of the system using the architecture and constraints to achieve the security requirements obtained during the requirements phase. Most of the readers are probably familiar with these concepts.

Abstraction is a process for reducing the complexity of a system by removing unnecessary details and isolating the most important elements to make the design more manageable. Decomposition is the process of describing the generalizations that compose an abstraction. One method, top-down decomposition, involves breaking down a large system into smaller parts. For object-oriented designs, the progression would be application, module, class, and method. Other secure software design principles are detailed in a multitude of books, white papers, web portals, and articles. In this paper we are providing some techniques to improve the SDLC.

First thing is minimize the no of high consequence targets. Minimizes the number of actors in the system granted high levels of privilege, and the amount of time any actor holds onto its privileges. Ensures that no single entity should have all the privileges required to modify, delete, or destroy the system, components and resources. Separation of domains makes separation of roles and privileges easier to implement.

DON'T EXPOSE VULNERABLE OR HIGH- CONSEQUENCE COMPONENTS:

- Keep program data, executables, and configuration data separated. .Reduces the likely hood that an attacker who gains access to program data will easily locate and gain access to program executables or control/configuration data.

- Segregate trusted entities from un trusted entities**,** Reduces the exposure of the software's high- consequence functions from its high-risk functions, which can be susceptible to attacks.
- Assume environment data is not trustworthy, reduces the exposure of the software to potentially malicious execution environment components or attacker- intercepted and modified environment data.
- Use only safe interfaces to environment resources; this practice reduces the exposure of the data passed between the software and its environment.
- Minimize the number of entry and exit points; this practice reduces the attack surface.

DENY ATTACKERS THE MEANS TO COMPROMISE

- Simplify the design; this practice minimizes the number of attacker-exploitable vulnerabilities and weaknesses in the system.
- Hold all actors accountable, this practice ensures that all attacker actions are observed and recorded, contributing to the ability to recognize and isolate/block the source of attack patterns.
- Avoid timing, synchronization, and sequencing issues, this practice reduces the likelihood of race conditions, order dependencies, synchronization problems, and deadlocks.
- Make secure states easy to enter and vulnerable states difficult to enter, this practice reduces the likelihood that the software will be allowed to inadvertently enter a vulnerable state.
- Design for controllability, this practice makes it easier to detect attack paths, and disengage the software from its interactions with attackers.
- Design for secure failure, Reduces the likelihood that a failure in the software will leave it vulnerable to attack.

In large distributed systems, scale-up problems related to security are not linear because there may be a large change in complexity. A systems engineer may not have total control or awareness over all systems that make up a distributed system. This is particularly true when dealing with concurrency, fault tolerance, and recovery. Problems in these areas are magnified when dealing with large distributed systems. Controlling the concurrency of processes presents a security issue in the form of potential for denial of service by an attacker who intentionally exploits the system's concurrency problems to interfere with or lock up processes that run on behalf of other principals. Concurrency design issues may exist at any level of the system, from hardware to application. Some examples of and best practices for dealing with specific concurrency problems, includes

- **Processes Using Old Data:** Propagating security state changes is a way to address this problem.

- **Conflicting Resource Updates**: Locking to prevent inconsistent updates is a way to address this.
- **Order of Update in Transaction-Oriented Systems and Databases**: Order of arrival and update needs to be considered in transaction- oriented system designs. System Deadlock, in which concurrent processes or systems are waiting for each other to act this, is a complex issue, especially in dealing with lock hierarchies across multiple systems. However, note that there are four necessary conditions, known as the Coffman conditions.

In above passage we provided some of security enhancement in design phase in future work we will consider the throughout the development life cycle.

CONCLUSION:

In this paper we started our initiation process of our research and we gave some suggestion to enhance the security mechanism to improve the system development life cycle. In our forthcoming papers we will give security principles throughout the lifecycle. It's our faith it will give better result than the ordinary development life cycle models. Compare with SQUARE and CLASP methods this one is different in functionalities.

REFERENCES:

[1] Mead, N.R., Viswanathan, V., Padmanabhan, D., and Raveendran, A., Incorporating Security Quality Requirements Engineering (SQUARE) into Standard Life-Cycle Models. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2008.

[2] Ambler, S. W. *A Manager's Introduction to Rational Unified Process*,2005.

[3] Kruchten, P. *The Rational Unified Process: An Introduction*, 3rd ed. Boston, MA: Addison-Wesley,2003.

[4] Mead, N. R., E. Hough, and T. Stehney. *Security Quality Requirements Engineering (SQUARE) Methodology*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[5] *Rational Unified Process: Best Practices for Software Development Teams*. Rational Software White Paper TP026B, Rev 11/01,2001.