Cutting Edge Practices for Secure Software Engineering

Kanchan Hans khans@amity.edu
Amity Institute of Information Technology
Amity University, Noida,
201301, India

Abstract

Security has become a high priority issue in software engineering. But, it is generally given a side thought. Security features are implemented after engineering the whole software. This paper discusses that security should be implemented right from the inception of software and planned for each phase of SDLC in software Engineering.The paper also suggests recommendations for implementing security at each phase of lifecycle of software. If each phase of the software engineering includes the appropriate security analysis, defenses and countermeasures, it will definitely result in a more robust and reliable software.

Keywords: Requirements analysis, security engineering, Design, Secure Software Engineering, Security vulnerabilities, risk analysis

1. INTRODUCTION

Security is considered as a very critical issue for software systems.Software is itself a resource and thus must be afforded appropriate security. But security often isn't the highest priority in software engineering. It is often seen as a task that a team performs in the testing phase at the tail end of the software development lifecycle (SDLC), after the developers have completed the code. Security is generally an afterthought i.e. given due attention only after going through all the phases of engineering a software. But, since the new technologies are coming up using networking, distributed capabilities of systems and also popularity of off the shelf components (COTS), security issues have rather become most important [9]. Security should be considered as an integral part of all phases of software development lifecycle. Also it has been observed that if security is implemented right from the inception of software, it saves the economy billions of dollars. Industry is thus in need of a model to examine security and quality requirements in the development stages of the production lifecycle. Therefore, it is rightly said by –Gene Spafford
"Security is like adding brakes to cars. The purpose of brakes is not to stop you: it's to enable you to go fast!"

2. PRACTICES FOR SECURE SOFTWARE ENGINEERING

Software that is developed with security in mind is typically more resistant to both intentional attack and unintentional failures. [10] However, it is incredibly hard to show that a particular system is 100% secure. Presently, there is no single solution for secure software engineering. However, there are specific approaches which improve the likelihood that a system is secure. The security of software is threatened at various points throughout its life cycle, both by inadvertent and intentional choices and actions taken by "insiders"—individuals closely affiliated with the

organization that is producing, deploying, operating, or maintaining the software. Both research and real-world experience indicate that correcting weaknesses and vulnerabilities as early as possible in the software's life cycle is far more cost-effective over the lifetime of the software than developing and releasing frequent security patches for deployed software. If Security mechanisms are fitted into a pre-existing design at a later stage, it will lead to design challenges that usually translate into software vulnerabilities [14]. Therefore security is a necessary property from the beginning of the system's life cycle (i.e., needs and requirements definition) to its end (retirement). Most approaches in practice today encompass training for developers, testers, and architects, analysis and auditing of software artifacts etc. The following section discusses the various security measures to be taken at each phase of SDLC in software engineering.

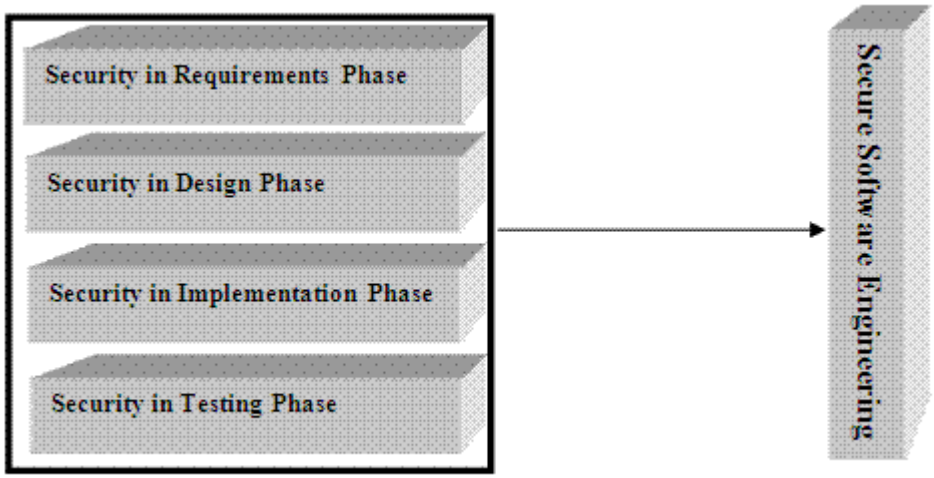

FIGURE 1: Secure Software Engineering

2.1 Security in Requirements Phase

Building a 100% secure system is hardly possible. In SDLC lot of problems arise because of inadequate Requirements Analysis. It is one of the main causes of over budgeted and delayed projects. Also problems at this phase cause poor quality applications and have reduced scope [1]. So, this phase should be given the utmost importance. From this layer itself, security features should be planned. Detailed requirements of a specific system with respect to security policy should be specified. Planning for such features and adding them at a later point in the life cycle makes this task a lot more difficult. A recent study found that the return on investment when security analysis and secure engineering practices are introduced early in the development cycle, ranges from 12 to 21 percent, with the highest rate of return occurring when the analysis is performed during the application and design.

Recommendations

- Policy on disclosure of information
- Authentication and password management-Use strong passwords. Support password expiration periods and account disablement. Do not store credentials (use one-way hashes with salt). Encrypt communication channels to protect. [13]
- Authorization and role management-Use least privileged accounts. Consider authorization granularity. Enforce separation of privileges. Restrict user access to system level resources.
- Network and data security- Encrypt sensitive data over the wire. Secure the communication channel. Provide strong access controls for sensitive data stores. Code integrity and validation testing.

- Cryptography and key management. Use strong passwords. Support password expiration periods and account disablement. Do not store credentials .Encrypt communication channels to protect.
- Ongoing education and awareness. This involves educating software engineers on general security concepts. Cases on previous security breaches and their consequences should be presented to them in order to appreciate the need for adequate protection of software products. [12]
- Requirements specification review/inspection to find security errors by possibly using a checklist of potential requirements specification security errors.[11]
- Build the abuse *cases* [6]. Similar to use cases, abuse cases describe the system's behavior under attack and building them requires explicit coverage of what should be protected, from whom, and for how long.

2.2 Security in Design Phase

At the design and architecture level, a system must be coherent and present a unified security architecture that takes into account security. Designers, architects, and analysts must clearly document assumptions and identify possible attacks.

Recommendations

- Risk should be covered using multiple defensive strategies. In case one layer of protection turns out to inadequate, the next level of defensive strategy will prevent a full breach.
- Secure design guidelines and principles should be followed while developing the initial design. Secure design patterns should either be followed or used for guidance. Secure design decisions should be specified using a secure design specification language.[11]
- The system should be divided into sub parts so that amount of damage that can be done to a system when a unit is compromised [4].
- External review (outside the design team) is often necessary to identify security errors. [8].
- Threat Analysis (risk analysis) should be undertaken that helps identifying issues before code is committed so that they can be mitigated in early SDLC. It involves identifying the conditions that cause incidents and analyzing them. [15]
- Brainstorming discussions should be conducted.
- Standard proven security toolkits (cryptographic and protocol libraries) should be selected.

2.3 Security in Implementation Phase

For the implementation phase, a secure programming language should be selected to minimize security errors. Moreover, secure coding standards and guidelines should be followed. [11] This phase also requires full consideration as per security is concerned. Here, the focus must be on implementation flaws. One may make use of tools that scan source code and detect common vulnerabilities. A security team should perform a code walkthrough with the developers and, in some cases, the system architects. A code walkthrough is a high-level walkthrough of the code where the developers can explain the logic and flow. It lets the code review team obtain a general understanding of the code, and it lets the developers explain why certain things were developed the way they were. Some more efforts that can be done are:

Recommendations

- Use of unsafe functions should be avoided or minimized. Look for potential buffer overflows, array out of bound errors, integer underflow and overflow, as well as data truncation errors. [13]

- Latest compiler toolset should be used [6].
- Manual code review must be done. Identify malicious behavior. Know what good traffic looks like.
- All inputs and outputs must be validated. Do not trust input; consider centralized input validation. Do not rely on client-side validation. Be careful with canonicalization issues. Constrain, reject, and sanitize input. Validate for type, length, format, and range. [13]
- Some logging and tracing activity should be followed. Audit and log activity through all of the application tiers. Secure access to log files. Back up and regularly analyze log files.

- Static and dynamic code analysis tools can be used to aid code review process to find vulnerabilities. Static analysis tool, also called source code analyzer examine a program's text without attempting to execute it. Dynamic analysis requires actually running the code. [16]

- Make no assumption: don't suppose something would never happen [9].

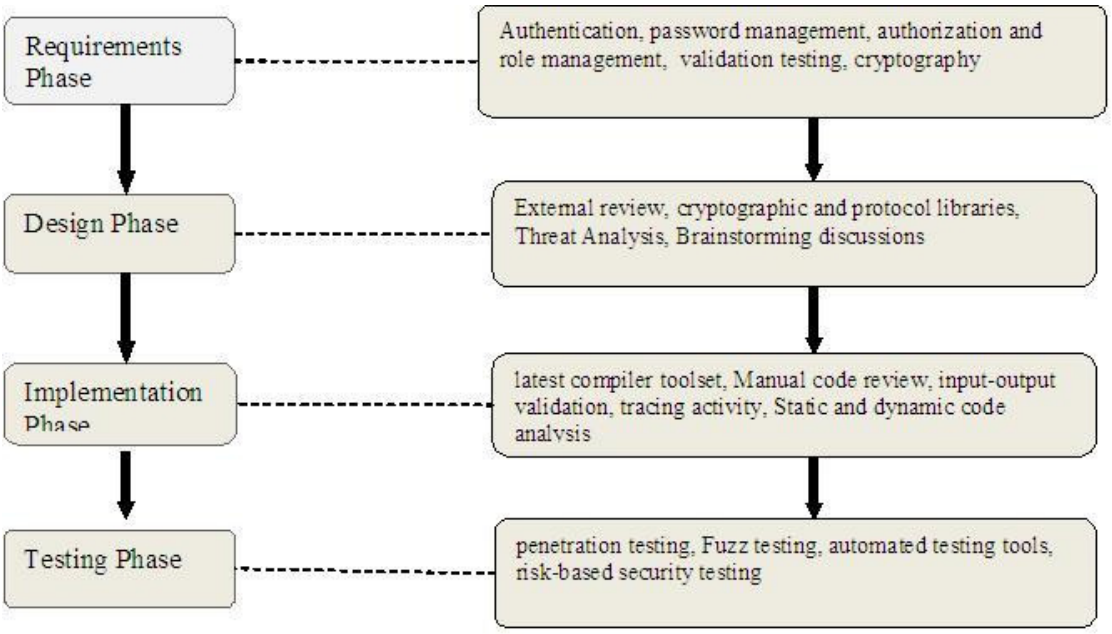

FIGURE 2: Integrating Security into each phase of Software Engineering

2.4 Security in Testing Phase

During this phase, various tests are conducted to check the security aspect of software being built. A good testing program engages a security team that is predominantly made up of the development team itself. Testing for security involves various techniques.

Recommendations

- Risk-based security testing based on attack patterns and threat models. It involves creating security abuse/misuse cases, performing architectural risk analysis risk-based security test plans. [17]

- Testing security functionality with standard functional testing techniques. Security test plans are prepared for both the strategies.
- Penetration testing should be done. While the application may be secure, a small aspect of the configuration could still be at a default install stage and vulnerable to exploitation. This is where automated black-box scanners are actually effective, looking for the known holes through poor configuration management [3] [7]. Testing most often involves running a series of dynamic functional tests to ensure proper implementation of the application's features. [14]
- Fuzz testing should also be done that provides invalid, unexpected, or random data to the inputs of a program. If the program fails (for example, by crashing or failing) the defects can be noted [2].
- Operations people should carefully monitor fielded systems during use for *security breaks*. So monitoring software behavior is an excellent defensive technique [5].
- Usage of automated testing tools is also recommended.

As risks can emerge up during all stages of the software life cycle, so a constant *risk analysis* thread, with continual risk tracking and monitoring activities, is highly recommended. More important, software development staff on critical software security issues [5].

3. CONCLUSION

Hence it is concluded that security must be integrated throughout the software development lifecycle (SDLC) in order to provide the user community with the best, most secure software. We must not leave security requirements to be dealt with as an afterthought. It is needed to incorporate security engineering throughout the software life cycle. If each phase of the software engineering includes the appropriate security analysis, defenses and countermeasures, it will definitely result in a more robust and reliable software. Therefore, security is rightly considered as life blood of software engineering

4. REFERENCES

[1] Nancy R. Mead, T. Stehney. *"Security Quality Requirements Engineering (SQUARE) Methodology"*. Software Engineering for Secure Systems -- Building Trustworthy Applications (SESS'05), 2005

[2] Fuzz Testing [Online]. Available at: http://en.wikipedia.org/wiki/Fuzz_testing

[3] Penetration test [Online]. Available at: http://en.wikipedia.org/wiki/Penetration_testing

[4] Jian Chen. *"Security Engineering for Software".* isis.poly.edu/courses/cs996-management /Lectures/SES.pdf

[5] G. McGraw. *"Software Security, Building Security"*. In published by IEEE Computer Society, 2004

[6] G. Blitz, Jarry, M. Coles, Dhillon, C. Fagan. *"Fundamental Practices for Secure Software Development: A guide to most effective secure practices today".* Safe Code Software Forum for Excellence in Code, 2008

[7] G. McGraw. *"Testing for Security during Development: Why We Should Scrap Penetrate-and-Patch".* IEEE Aerospace and Electronic Systems, 13(4):13–15, 1998

[8] G. McGraw. *"Building Secure Software: Better than Protecting Bad Software ".* IEEE Software, 19(6):57–59, 2002

[9] D. J. Hulme, B. Wassermann. *"Software Engineering for Security".* Available at: www.cs.ucl.ac.uk/ staff/ucacwxe/lectures/3C05-01-02/aswe17.pdf

[10] Allen, Julia, Barnum, Sean, Ellison, Robert, McGraw, Gary, Mead, Nancy. *"Software Security Engineering: A Guide for Project Managers".* Addison-Wesley, 2008

[11] M. U. A. Khan, M. Zulkernine. "*A Survey on Requirements and Design Methods for Secure Software Development"*. Technical Report No. 2009 – 562 , School of Computing, Queen's University, Kingston, Ontario, Canada, 2009

[12] Sodiya, Onashoga, Ajayi. *"Towards Building Secure Software Systems, Issues in Informing Science and Information Technology"* . 3: 2006

[13] J. D. Meier, A. Mackman, B. Wastell, P. Bansode, J. Taylor, R. Araujo. "Software *Engineering Explained: Patterns and Practices"*. Microsoft

[14] G. McGraw. *"Software Penetration Testing, Building Security In".* published by IEEE Computer Society, 2005

[15] Barbato, A. Montes, Vijaykumar. *"Methodologies and Tools for Software Vulnerabilities Identification"*

[16] G. McGraw. ***"****Automated Code Review Tools Used for Security, How Things WorK".* Cigital, 2005

[17] G. McGraw. *"Software Security Testing, Building Security In"*. published by IEEE Computer Society, 2004