International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:9, 2016

Importance of Hardware Systems and Circuits in Secure Software Development Life Cycle

Mir Shahriar Emami

Abstract**—**Although it is fully impossible to ensure that a software system is quite secure, developing an acceptable secure software system in a convenient platform is not unreachable. In this paper, we attempt to analyze software development life cycle (SDLC) models from the hardware systems and circuits point of view. To date, the SDLC models pay merely attention to the software security from the software perspectives. In this paper, we present new features for SDLC stages to emphasize the role of systems and circuits in developing secure software system through the software development stages, the point that has not been considered previously in the SDLC models.

Keywords**—**Systems and circuits security, software security, software process engineering, SDLC, SSDLC.

I. Introduction

THOUGH software engineering methodologies have been a challenging research area in the past 45 years, , the improvement on software process engineering methods still have been the most challenging effort in the field of software engineering. As the literature shows [1], almost all of the SDLC methodologies have less concrete attention to hardware security issues during the developing of the software. This will cause to increase the risks of software systems in terms of security. Such circumstances let many threats not only would cause physical and logical destructions but also in a larger perspective could bring about devastating catastrophes in social activities and business economics.

The chief importance of this research is to include systems and circuits security perspectives into the SDLC model in order to develop secure software systems. Developing secure software systems involves a concrete attention to the security issues in the hardware system which the developing software will run on it. Thereby, considering the hardware security issues during the SDLC is a vital need for the secure software development.

In this paper, in Section II, we summarize the cyber security requirements. In Section III, we study today's world cyber security situation. In Section IV, we have a survey on SDLC models and their characteristics. In Section V, we summarize the hardware vulnerabilities and their effects on software systems. In Section VI, we present features for SDLC based on systems and circuits perspectives and the roles of them on software development stages during SDLC.

Mir Shahriar Emami is with the Faculty of Engineering, Computer Engineering Department, Islamic Azad University (Roudehen Branch), Roudehen, Tehran, Iran (e-mail: mshemami@riau.ac.ir).

II. Cyber Security Requirements

One common sentiment across the current researches on cyber security is the need to distinguish the role of security requirements areas. These security requirements consist of the following issues:

- Information Security.
- Software Security.
- Network Security.
- Hardware Security.
- Organization Security.
- Vulnerability Assessment.
- Independent Verification Activities.
- Threat Analysis.

In this paper, the mutual relation between hardware security concepts and secure software development has been studied. In fact, considering hardware security issues in software development stages let software developers to develop secure platforms to ensure the acceptable level for cyber security. This also can lead the hardware designers to include new security elements and features for their new products. The process of manufacturing can be categorized in several levels as:

- Box Level: Developing a secure computer.
- Board Level: Manufacturing secure appliances.
- Components Level: Designing secure chips, processors, memories, and off-chip and on-chip communication circuits and networks.

Manufacturing of such new hardware elements can guarantee the security of the future software development systems.

III. Today's Cyber Security Situation

Nowadays, cyber security is in serious danger. Spear phishing, malicious intrusions, SQL injection, cross-site scripting, worms, Trojans and viruses are quite enough to endanger security of cyber environments. The SANS Institute published a report in September 2009 based on attack data, which collected between March 2009 and August 2009, from software and appliances in more than 6,000 organizations [13]. Fig. 1 shows a number of cyber-attacks in this period of time. This figure shows a total number of near 38,000,000 cyber-attacks just only for three types of them in a period of six months in the United States [13]. Meanwhile, antiviruses and firewalls sometimes cause many problems against their benefits [14]. These situations are enough to convince us to accept that our cyber security is in danger seriously. In such an air, it is better to think of secure development of cyber environments rather than secure them after their development.

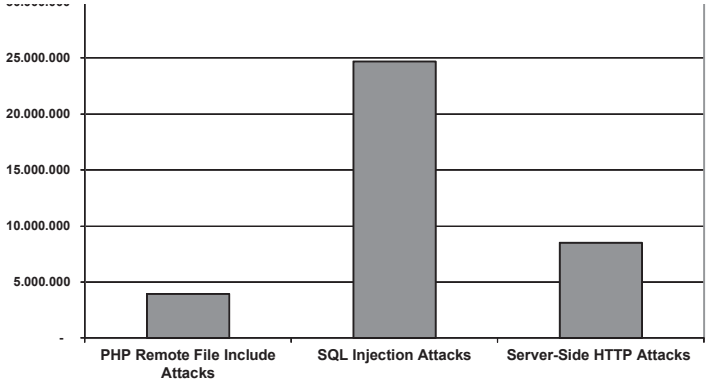

Fig. 1 A number of some cyber-attacks in USA between March 2009 and August 2009 [13]

TABLE I
SDLC MODELS AND THEIR CHARACTERISTICS

SDLC Model	Specification
Pure Waterfall model [1], [2]	• Stage by stage software system development. • Two distinguishable phases: Analysis and Coding. • High degree of documentation.
V model [1]	• An extension of the pure waterfall model. • Testing activities begin with the commencement of the project prior to coding stage. • Various types of functional testing activities are accomplished through the different stages.
Iterative Enhancement [4]	• Top-down, stepwise improvement approach to software development. • The total scope of a project is decomposed to smaller chunks of tasks. • Supports a strategy to distribute maintenance updates and services to disperse user communities.
Parallel development model [1]	• In this model different subsystems are developed in a parallel way. • This model does not support changing requirements effectively.
Cleanroom [5]-[7]	• The term is derived from the process used to produce semiconductors.
Spiral Model [8]	• An evolutionary model. • Distinguishing the process model differed from software method. • Ability of navigation between each two stages. • Software development start at the center position then moves clockwise in traversals in a spiral manner to complement the process. • In each traversal some proportion of the work is completed. • Each traversal often results in a deliverable such as specification, prototype and so on therefore by a spiral shape movement the related activities will mature towards the outer traversal. • A risk-driven approach to the software process rather than document-driven or code-driven process.
Extreme Programming (XP) [9], [10]	• An Agile method. • Simple rules. • Ability to change customer requirements in a lightweight manner. • Software developers can confidently respond to modifying user requirements, even late in the life cycle. • Short time to advocate frequent software product releases in short development cycles which is called Time Boxing. • Ability of building working software at the very beginning time of the software development.
UML [12]	• Object oriented perspective. • Pay more attention on production.
RUP [11]	• Iterative software development methodology. • Provides a disciplined approach to assigning tasks and responsibilities within a development organization. • Its activities create and maintain models instead of large amount of paper documents. • It strives to produce the software components as soon as possible. • Through each stage, several iterations may occur.
Synchronize and Stabilize.	• A mixture approach of planning-driven and agile methods. • The software development process does not start at a complete specification. • Daily build approach to frequently synchronize the works produced via building a new version of the complete system at the end of each day. • Software development starts at the vision of development team about what they want to do. This vision leads team's manager to an initial functional specification by a schedule that has multiple milestones whereas each of them shows development sub cycle progress. • Software development and software testing are done in parallel. • A buffer time is provided prior to software release intending to illustrate that the team can easily satisfy the demands of market, for instance adding a feature. • In the final milestone, the user interface is frozen, and the code is run for a final test, for debugging, and stabilizing sub cycles and issue a final release.

IV. SDLC Models

SDLC was demonstrated in 1970 by Royce [1]. The Royce's model was called Pure Waterfall model. This model showed how a software system can be developed stage by stage. Royce by presenting Waterfall model illustrated that software development process consists of two basic stages: analysis and coding. He depicted that these two main stages were obviously different in the way they executed [1]. Table I shows a brief look on current SDLC models [3] and their characteristics.

V. Systems and Circuits Security Perspectives

Despite the fact that implementing the security into the hardware architecture has been the key scope of the hardware researches, which can be used to guarantee the cyber-security; thinking of such hardware, which can be released during the SDLC to improve the cyber-security, is much more important. As a matter of fact, software systems have been manifesting themselves in a number of improving and new hardware-based functions and services. However, previous SDLC approaches have less attention in terms of hardware vulnerabilities and their terrible effects on software systems. In this sense, more studies about those are in a high importance. To date, in software system development, the employed hardware was considered attack-free and safe enough but the previous practices showed that hardware is as vulnerable as software. This means that hardware circuit vulnerabilities are possible and terrible. In fact, harmful logics which cause system problems can be embedded within the hardware design similar to software viruses and Trojans which bring about software system security vulnerabilities. Though the developments of encryption technologies increase the security of software systems, there are several security-attacks which can endanger the software system. For example, data may be leaked from the source of data by physical reading of the permanent memory before encryption. Table II shows the hardware vulnerabilities and their origins.

TABLE II
Hardware Circuits Vulnerabilities

Item	System Problem	Origin
1	Malicious programming of system Flash Rom	Firmware
2	Cause the System to Output incorrect data	Hardware
3	Creating Wrong Port or Address	Hardware-Software
4	Changing the System Internal Timing	Hardware
5	Disabling the System Clock	Hardware
6	Disabling the System Bus	Hardware
7	Forcing Stress Factors	Hardware
8	Physical Reading of Permanent Memories	Hardware
9	Consumption and Electromagnetic Radiations	Hardware-Software
10	Adding Extra Connections	Hardware
11	Using External Mass Storage Devices	Hardware-Software
12	Information Leakage	Hardware-Software
13	Modifying Internal Circuit Structures	Hardware

VI. Proposed SSDLC Features

As we mentioned in Section IV, none of the current and traditional SDLC models have any attention to the hardware issues during the software development. In Section V, we clarified the hardware circuit vulnerabilities and the effect of those on software systems. Now, we are going to consider hardware circuits' issues during the software development process but before that, let us call the main stages of secure software development life cycle (SSDLC) as: URS, SRS, ADS and DDS. In Table III we present new features for each stage of SSDLC.

TABLE III
New Features in SSDLC Stages

Item	URS	SRS	ADS	DDS
1	×	×		
2	×	×	×	×
3	×	×	×	×
4	×	×		
5	×	×		
6	×	×		
7	×	×		
8	×	×		
9	×	×		
10	×	×	×	×
11			×	×
12			×	×
13	×	×		

In URS, we attempt to find out the user requirements. While we are trying to recognize the future end users, we can think of the kind of hardware which they will use or we can think about their knowledge about the hardware circuits and systems. We can look at the roles in the organizational chart more precisely focusing on their experiences about hardware and make some questions about them. For example, we can ask ourselves if they can program the Flash Rom of a PC, or who can cause the system to output incorrect data, or who can probably create wrong port or hardware address inside the organization.

In SRS we analyze the system requirements. We can think of the best platform for the developing software system and we can consider hardware vulnerabilities in that platform. We can make questions again. If the system internal timing changes by a hardware attacker what would be the effects of that on the software system, or if the hardware circuits sustain forcing stress factors like temperature, voltage, glitches, light, laser and particles, what kind of information leakage may be occurred and whether this information could be harmful for software systems. In this stage, we can also think of external media. For example, if the software system will use smart cards, we can think of smart card physical attacks like modifying the material of the silicon part of its microchip or cutting the internal wires in order to disconnect the sensors. If these happen, we should find the probable effects of such kind of attacks on the software system and the ways that we can protect the side effect of such kind of attacks.

In ADS which software engineers to bring users' perspectives into the software architecture, it can be said that software architecture relates directly to hardware structure. Hence, any circuit vulnerability can affect directly on software architecture. In this stage, we can think of hardware

requirements of the employed software architecture and their vulnerabilities in the time we attempt to develop convenient software architecture for our system.

In DDS the software system is developed in details. In this stage, we can think of forms of displaying the information if any information leakage occurs. This means whether the attackers can see the leaked information through the webpage which we are designing now or whether they use an external hardware for reading the information leakage, and the kind of ports that these attackers may exploit for retrieving the leaked information.

VII. Conclusions and Future Works

In this paper, a brief look on current SDLC models has been done and we found that previous and traditional SDLC models do not pay attention to the systems and circuits' vulnerabilities during the software development stages. Although developing a fully secure software system is not possible, considering the hardware security issues during the software system development increases the software system acceptability in terms of security issues. In this paper, we were successful to propose new features for the SSDLC model based on hardware vulnerabilities. This enables secure software system development. In our future work, we attempt to present an SSDLC model which includes systems and circuits security issues during each of the SSDLC stages.

References

[1] M. S. Emami, N. B. Ithnin, O. Ibrahim, Software Process Engineering: Strength, Weaknesses, Opportunities and Threads, 6th International Conference on Networked Computing, IEEE Seoul Section, pp.148-152, 2010.
[2] W. W. Royce, "Managing The Development of Large Software Systems", Proceedings, IEEE Wescon, 1970.
[3] H. V. Vliet, Software Engineering Principles and Practices, Book, John Wiley & Sons, ISBN 978-0-470-03146-9, 2008.
[4] V. R. Basili and A. J. Turner, " Interactive Enhancement: A Practical Technique for Software Development ", Journal, IEEE Transaction on Software Engineering, IEEE Computer Society, 1975.
[5] H.D. Mills, M. Dyer, R.C. Linger, "Cleanroom Software Engineering", Journal, IEEE Software, IEEE Computer Society, 1987.
[6] R. S. Oshana and R. C. Linger, "Capability Maturity Model Software Development Using Cleanroom Software Engineering Principles - Results of an Industry Project", Proceedings, 32nd Hawaii International Conference on System Sciences, IEEE Computer Society, 1999.
[7] R. C. Linger, "Clean room Software Engineering for Zero- Defect Software", Proceedings, 15th International Conference on Software Engineering, IEEE Computer Society, 1993.
[8] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", Journal, Computer, IEEE Computer Society, 1998.
[9] D. Wells, "Extreme Programming: A gentle introduction", www.extremeprogramming.org
[10] "Design Patterns and Refactoring", Lecture, University of Pennsylvania, http://www.cis.upenn.edu, USA, 2003.
[11] "Rational Unified Process Best Practices for Software Development Teams", Rational Software Corporation, www.ibm.com/developerworks/rational
[12] S. R. Schach, Object-Oriented Classical Software Engineering, Book, Mc Graw Hill, ISBN: 0-07-319126-4, 2007.
[13] "The Top Cyber Security Risks", SANS, www.sans.org , Sep 2009
[14] P. A. Strassmann, "Cyber Security for the Department of Defence", June 2009.

Mir Shahriar Emami is Assistant Professor (Senior Lecturer) at RIAU University and an invited professor at SRBIAU University in Iran, and an active researcher and entrepreneur in Elites Technology Incubator at Pardis Technology Park (PTP) in Pardis, Iran. He is postdoc fellow and a computer science researcher in the field of multimedia security and a key note speaker in the field of cloud computing in the National Conference on Interdisciplinary Researches in Computer, Electronics, Mechanical and Mechatronic Engineering (IRCEM 2016) in Iran. His current research is about digital image watermarking algorithms for digital asset authentication, ownership identification and copyright protection. He is also interested in secure software development. In addition, he is interested in quantum computing and quantum algorithms. For more information about this author, the site: www.mirshahriaremami.com is available.