A Strategy Oriented Process Model for Software Security

Syed Mohammad Shahe Alam¹, Dr. S.K Singh², Dr. Suheil A.Khan³
¹Research Scholar, Amity University Lucknow, INDIA
²Professor, AIT, Amity University, INDIA
³Assistant Professor, Department of Computer Science, Integral University, INDIA

ABSTRACT

Traditionally, security of software is not considered from the very beginning of a software development life cycle, and it is only incorporated in the later stages of development as an afterthought. As a consequence, there are increased risks of security vulnerabilities that are introduced into software in various stages of development. To avoid security vulnerabilities, there are many secure software development efforts in the directions of secure software development life cycle processes, security specification languages, security requirements engineering processes, secure design languages, and secure design guidelines. In this paper, we compare and contrast various secure software development processes based on a number of characteristics that such processes should have. We also analyze security specification languages with respect to desirable properties of such languages. Furthermore, we identify activities that should be performed in a security requirements engineering process to derive comprehensive security requirements. We compare different security requirements engineering processes based on these activities. Finally, we investigate the state-of-the-art in secure design languages and secure design guidelines. Our analysis shows that many of the secure software requirements and design methods lack some of the desired properties. The comparative study presented in this paper will provide guidelines to software developers for selecting specific methods that will fulfill their needs in building secure software applications.

Keywords: Security breaches, Software system, Software security, Software design, Design defects

I. INTRODUCTION

Introduction

Security of software has become an important issue with the increasing integration of software in various aspects of human society. Software is considered to be secure if it does not allow the confidentiality, integrity, and availability (widely referred to as CIA) of its data, code, or service to be compromised [1]. However, most of today’s software are not secure and contain security vulnerabilities that can be exploited by people with malicious intent to cause financial and/or physical harm. Traditionally, security of software is not considered from the very beginning of a software development life cycle (SDL). It is only incorporated in the later stages of development as an afterthought. As a consequence, there are increased risks of security vulnerabilities that are introduced into software in various stages of development. The traditional approach to security of software has led to penetrate-and-patch approaches. In a penetrate-and-patch approach, security of software is assessed by attempting to break into the software from its environment by exploiting known security vulnerabilities. If such a penetration attempt is successful, a patch is developed and deployed for the vulnerability that allowed the break-in. Penetrate-and-patch approaches have many major drawbacks [2-6]. First, developing and deploying a patch to remove an error that causes a security vulnerability can be up to 200 times more expensive [7] than if the error had been removed as soon as it was introduced during development. Second, there is no assurance that the developed patch itself does not have any new security vulnerabilities (assuming that the traditional approach of software development was followed). Third, there is no guarantee that all the existing security vulnerabilities have been identified. Finally, major harm could have already occurred before a security vulnerability is even detected [6]. Secure software engineering aims to avoid security vulnerabilities in software by considering security aspects from the very beginning and throughout the SDLC. Secure software engineering is the process of designing, building, and testing software so that it becomes secure. Software security concerns are different from “application security” issues. Application security is about protecting software after it is developed and deployed. It usually includes input filters, intrusion detection systems, firewalls, and other protection mechanisms [2-5].

Computer software systems are increasingly faced with both internal and external penetrations. One major reason for this is the fact that software systems are still

Copyright © 2016. Vandana Publications. All Rights Reserved.
Software security includes secure software development life cycle (SSDLC) processes and secure software development (SSD) methods. A SSDLC process is either a SDLC process augmented with various SSD methods or a set of independent SSD methods. An SSD method provides ways to incorporate security in software during its development. An SSD method maybe a security specification language, security requirements engineering process, secure design specification language, set of secure design guidelines, secure design pattern, secure coding standard, and software security assurance method (e.g., penetration testing, static analysis for security, and code reviews for security). In this paper, we analyze the existing SSDLC processes, requirements methods, and design methods for secure software development based on their strengths and weaknesses. We first identify a number of characteristics of a SSDLC process that make it complete and effective. These characteristics are then used to compare and contrast various SSDLC processes. We also present a detailed comparison of security specification languages based on a number of desirable properties. This comparison focuses on identifying languages that can effectively specify security requirements. Moreover, we identify activities that should be performed as part of any security requirements engineering process. Based on these activities, we compare various security requirements engineering processes and identify their strengths and weaknesses. Finally, we provide an analysis of the state-of-the-art of different secure design languages and guidelines. The analysis presented in this paper can be useful in a number of ways. The comparison of various SSD methods will help software developers in selecting a particular SSDLC process, security specification language, security requirements engineering process, or secure design language for their particular development scenarios. The identified properties of software security specification and design languages can be useful to translate one specification language into another. Such a translation is particularly useful when a user of one language intends to use security tools developed for other languages. The rest of this paper is organized as follows. Chapter 2 defines the software security terms used in this paper. Chapter 3 presents an in-depth critical analysis of SSDLC models or processes. Chapter 4 analyzes methods for 2 software security requirements engineering. Methods for secure software design are discussed in Chapter 5. Chapter 6 concludes this paper by summarizing our findings and identifying the corresponding open research issues.

II. OVERVIEW OF EXISTING LITERATURE RECENTLY

There has been a lot of interest in building software securely. Pauli and Xu (2005) provided a threat-driven architectural design of secure information systems. The work presented threat modeling using misuse cases and stated that the findings can be used in detailed design and validation of implementation of software system. Wilander and Gustavsson (2005) mentioned in their own work that security requirements are normally poorly specified due to these three things: inconsistency in the selection of requirements, inconsistency in the level of detail and almost no standard requirements on some security solutions. The result of the study shows that security mainly treated as a functional aspect composed of security features such as login, backup, and access control. They stated that requirement on systems through assurance measures are left out. Devanbu and Stubblebine (2000) in their work emphasized that software engineering and security engineering must be integrated in order to have secure software systems. The two software articles of Ghosh, Howell, and Whittacker (2002) and Mead and McGraw (2003) also emphasized the need to build software securely from the ground. Apart from stating and emphasizing the security needs of software system, many researchers have actually proposed methods for improving software security. One method is the correctness-by-construction proposed by Praxis Critical Systems Limited (Hall & Rod, 2004). The method operates on the principles that errors should not be introduced in first place and that errors should be removed as close as possible to the point they are introduced. The method also incorporates formal notations to specify system and design components with review and analyses for consistency and
correctness. Hall and Rod (2004) reported that correctness-by-correction method produced defect densities ranging from 0.04 to 0.75 defects per thousand lines of code. Another method for achieving secure software system is the Cleanroom (Linger & Stacy, 2004). Cleanroom incorporates incremental development, functional-based specification and design, correctness verification, and statistical testing. Redwine and Davis (2004) reported the overall performance of application of cleanroom as ranging from 0.1 errors/KLOC (KLOC means Kilos of Lines of Codes) with full application to 0.4 defects/KLOC with partial application. Capability Maturity Models (CMMs) have also been used as process models to guide organizations in improving the capability to perform a particular process. An example of security related CMM is Systems Security Capability Maturity Model (SSE-CMM), which was presented in Hefner (1997). Golderson and Gibbon (2003) reported that CMMs have also helped in the overall reduction in design and implementation defects of software products. Similar to CMM is threat modeling. Threat modeling is used to analyse potential threat to computer systems in order to guide against attacks or penetrations. The usefulness of threat modeling has demonstrated in some previous works (Mogilevsky, Lee, & Yurcik, 2005; Myagmar, Lee, & Yurcik (2005). Attack trees or graphs have also been used in determining what security measures to deploy in a system (Schneir, Lippman, & Wing, 2002; Sheyner & Wing, 2004). However, attack trees model a selected set of attacks via a finite state machine and feasible only in small scenarios. They also require compiling a list of potential threats before generating attack trees. The empirical performance of both threat modeling and attack trees are not readily available. Another method for improving security of software is the Software Engineering Institute’s Team Software Process (TSP) (Davis & Mullaney, 2003). The process incorporates the idea of managing and removing specification, design and implementation defects throughout development lifecycle, controlling and monitoring of process, and using predictive measures of removing defects. Davis and Mullaney (2003) reported an average of 0.06 delivered design and implementation defects per thousand lines of codes produced. It has shown from the literature that software systems are still with defects (though minimal in some situations). The fact is that an attacker only needs to find one security flaw to compromise the whole system. It is therefore important to design a bug-free software because the implication of a bug might be catastrophic and results in loss of large amounts of money.

III. SOFTWARE SECURITY ISSUES

Software security is concerned with protection of software and its resources. A security risk is the probability of sustaining a loss of a specific magnitude during a specific time period due to a failure of security system (Rodgers, 2002). Software Security Goals. The three security goals of computer system are Confidentiality, Integrity and Availability. These goals are commonly referred as the CIA of computer security (Sodiya, Longe, & Akinwale, 2004) and these goals also apply to computer systems. a. Confidentiality:- This has to do with the prevention of unauthorized disclosure of software resources such as codes, data, documents, files, GUI, and so on. b. Integrity:- This is prevention of unauthorized modification of software resources. c. Availability:- This is concerned with the unauthorized denial of the services of software resources. Over the years, there have been several security mechanisms to achieve these goals. Some of these mechanisms are authentication and authorization, access control, encryption, and so on, but the rate and magnitude of attacks on computer system is increasingly alarming. Some of the interests of attacker are financial and knowledge gains, malicious intention, competitive edge and so on. The situation is even worse now that we have mobile codes in distributed network and exposure to Internet.

IV. PROBLEMS OF PRODUCING SECURE SOFTWARE

Producing secure software is complex and requires high integration of security and software engineering. Based on our experience, we present the problems of producing secure software as follows:- a. Attackers’ Knowledge:- As software security continue to gain the attention of security experts and organizations, the tools and techniques used by attackers are becoming increasingly sophisticated and invasive. b. Complexity of software:- Software development is a complex process because it involves many activities and specialties. It involves different units, stages and personalities, and all these have to be integrated to achieve successful software production. The situation is even worse with the connection to Internet and the use of mobile codes. c. Security Education:- Producing secure software requires a lot of security training, education and experience. Many software developers are not well grounded in computer security. To what extent of the techniques and tools of attackers do they understand? Also, many programming books do not teach how to write secure programs. Many curricular of schools offering computer science do not properly address computer security. d. Attitude of Software Engineers:- In the past, software developers were only interested in producing software product with the desired quality. Little attention was paid on making secure products. But, the situation is changing now that everybody is aware of the implications of vulnerabilities in
software. e. Inadequacy of Computer Security Models:- Many security models are still not adequate for producing secure software products. Implementing some of them is complex and difficult to realize.

V. SECURE SOFTWARE DEVELOPMENT & THEIR MODEL

It has been said that producing software requires integrating Software Engineering (SE) process with Security Engineering. In doing this, a careful understanding of software development process is important. We now present an overview of software development process.

Similarity with Software Specification Languages

To integrate security concerns during software development seamlessly, it is essential that security specification languages are as close as possible to the software specification languages. The major reason for this is that developers would have to spend very little time in becoming well versed in these languages. For example, all the languages based on UML (such as Abuse cases, Misuse cases, UML sec, Secure UML, and UM Lintr) would be much easier to learn for the software practitioners who know UML. The languages based on the notion of extended finite state machine and the ones with syntax close to existing programming languages (e.g., STATL [47]) may gain popularity more quickly among developers.

VI. SSDM SECURITY ENGINEERING PROCESS

As shown in Figure 1, the security engineering path is divided into five stages.

a. Security Training

Not all software engineers have adequate knowledge of computer security. The essence of security training is to provide adequate security education to stakeholders in software development. The key training requirements are:

- Security awareness
- Knowledge of attackers on previous related applications
- Understanding attackers’ interests on software being developed
- Knowledge about secure development practices

b. Threat Modeling

A threat model is used to effectively and comprehensively identify attackers and their capabilities. Every software development that must be secure must have its own threat model because the common security criteria might not be suitable for all software products. SSDM threat model is divided into three parts. The classification is similar to what is presented in Myagmar et al., 2005, but our own seems to be more detailed.

i. Understand the nature of the software
ii. Identify attackers/threats
iii. Identify possible vulnerabilities

c. Security Specification (SS)

This entails stating the guidelines and procedures that guarantee security of the system. The SS should contain the following:

i. Security needs
ii. State security policies
iii. How to coordinate security implementation
iv. How to make the system to adapt to the changing landscape of security
v. How to monitor security postures of the software system

d. Review SS

This is to check whether the design content of the software is in accordance with the SS. This is important because if the design is defective, then the software design must incorporate the security specifications. If the design has not guaranteed the items in the SS, then the design will be reviewed.

e. Penetration Testing

Capabilities of the software in preventing attacks are tested here. It is important so as to:

- Test the security of the software and its resources
- Determine if the current security posture of the software is actually detecting and preventing attacks.
Penetration testing is carried out in a way that all identified attacks and future attack patterns are initiated on-line into the software, and the capability of the software in preventing these attacks is monitored.

VII. FUTURE WORK AND CONCLUSIONS

In this paper, we examined the problem of producing secure software. The research in software security is gaining much attention because of the implication of security breaches. Problems towards producing secure software products were identified and stated in this work. This is important for researchers so as to know the real problems and the direction to follow in providing the methodologies for providing secure software product. We also proposed an integrated model for providing secure software products and presented a case study to ascertain that the model works. The model has some interesting features that guarantee the successful production of secure software products. SSDM has shown a way of separating security specification from functional specification. It is believed that if this model is carefully implemented; it will result in the production of secure software products. It was shown in this work that implementing SSDM improves the security of software systems. The model presented in this work still needs to be more tested so as to really ascertain its performance. Some of the components of SSDM are still major areas of research that should be studied.

REFERENCES
