Cybersecurity Practices from a Software Engineering Perspective

Aakanksha Rastogi, Kendall E. Nygard
Department of Computer Science
North Dakota State University
Fargo, ND, USA
Email: Aakanksha.Rastogi@ndsu.edu, Kendall.Nygard@ndsu.edu

Abstract. Building a software application that is equipped with up-to-date security measures and is compliant with standard security rules, regulations and standards is a complex task. Patching software after a hacker attack may often be the only a situational reasonable solution. However, incorporating security requirements while building the software in the first place can be helpful in shielding from hacking attacks. Determining when and how to integrate security considerations is a debatable issue. We assert that including security aspects at each phase of the software development life cycle adds significant protection to the software. This paper provides a survey of relevant literature of secure software development practices that have been developed and utilized to build software applications. We also addressed related issues of trust and autonomous systems.

Keywords. Cybersecurity, Trust, Security, Cyber Attacks, Software Development Life Cycle (SDLC), Analysis.

I. INTRODUCTION

Incorporating cybersecurity protections into software applications during development is a complex issue. In the ever-expanding digital age, virtually every aspect of human endeavor relies on secure transactions and operations. However, consideration of cybersecurity issues is often inadequate, leading to problems such as financial losses, data losses, and privacy breaches. From a systems and networking view, enormous efforts have been made to develop tools to combat specific types of cyber-attacks as they appear. However, hackers tend to think differently than developers of applications and are constantly and proactively developing increasingly notorious and creative attack strategies. Such attacks in planting malicious pieces of code that corrupt the application, steal sensitive customer information, or introduce malware such as viruses, worms and spyware, phishing, extortion schemes, and spam, [1], can be exploit vulnerabilities introduced at any step of the development process.

Software applications that are vulnerable to cyber-attacks can drive potential customers and users of the application away. To gain user trust in purposeful applications, it is important to carry out application development while carefully addressing security issues at each step. Software developers tend to focus on functional requirements, with little emphasis on non-functional requirements, such as security. Some authors report benefits of addressing security measures at the distinct phases of the software development life cycle [2][3][4][5][6]. Futcher and von Solms proposed guidelines for secure software development [6]. In this paper we provide a survey of literature that is relevant to secure software development practices. Several security issues, concerns, challenges, and solutions at different phases of the software development life cycle as described in the literature on cybersecurity are also presented. However, the scope of this paper is limited to Analysis, Design, Implementation, and Testing phases of the Software Development Life Cycle (SDLC).

The organization of rest of the paper is as follows. Section II describes the characterization of trust in cybersecurity. Section III describes the phases of a typical software development life cycle. Section IV describes the security concerns encountered during the Analysis phase of the Software Development Life Cycle and potential ways to address those concerns. Section V discusses future work and Section VI wraps up the paper with the conclusion.

II. CYBERSECURITY AND TRUST

With technology advancement and mass digitalization of user personal data, establishing user trust has become an important factor in the use of software systems. Most software systems are potentially vulnerable to attacks even if there is strict adherence to leading edge principles of encryption and decryption. Security of software systems is classified into three categories: Confidentiality, Integrity and Availability [7][8][1]. These categories are also collectively known as the CIA triad. Confidentiality is defined as “Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information…” [9]. Integrity is defined as “Guaranteeing against improper information modification or destruction, and includes ensuring information non-reputation and authenticity…” [9]. Availability is defined as “Ensuring timely and reliable access to and use of information…” [9].

Security is often interwined with trust. In the context of software systems, trust refers to the level of confidence or reliability that a person places in a software system, including the expectations that they have for the software fulfilling its purpose. The software system can be of multiple elements, including programs, configuration files, and documentation. In addition, the concept of trust in the context of cyber security includes expectations that people have from all aspects of software development, including requirements, design, platform-specific issues and networks, for which various security practices, processes and technologies are in use.

Trust also refers to a relationship that a person forms with software applications that are online or over a network. The
trust relationship is betrayed if the user’s expectations from these applications are not met. This raises questions concerning the kinds of expectations that users have with the usage of an application and the factors that diminish trust. One factor arises from any negative risks that are associated with the usage of an application. There are traditional ways of assessing risk in cybersecurity. Oltramari et al. [7] identified endpoint users as key introducers of risk in an application network, since humans, such as software developers, attackers and users of the application are included as a component of the system. In addition, low skill level or exhausted software developers tend to increase cybersecurity risk, while users can substantially decrease cybersecurity risks by being aware and attentive to the means of protecting their personal assets from, phishing or spam efforts [5]. Again, insiders within an organization are also known to sometimes support and execute malicious attacks for which outsiders have minimal knowledge. As described by Colwill, “A malicious insider has the potential to cause more damage to the organization and has many advantages over an outside attacker” [10]. These human concerns in information and cybersecurity make it important to learn to distinguish between regular users, potential hackers and insiders who can pose a great threat.

Trust and human factors in cybersecurity all also of great concern in the rapidly expanding area of autonomous systems, many of which utilize advanced methods of artificial intelligence. Examples of autonomous systems include floor cleaning robots, agent software, military and private drones, surgery-performing robots and self-driving cars. Autonomous systems are managed and supervised independently by a single administrator, entity, or organization [11] [12]. Each autonomous system has a unique identifying label that can be used during data packet transfer between two systems [12]. Some autonomous systems can make decisions and perform tasks in unstructured environments with no need for human control or guidance. Stormont [13] illustrated a spectrum of autonomy ranging from a low end of fully remote control to a high end with no human intervention. Figure 1 illustrates the spectrum.

![Examples of degrees of autonomy.](image)

In addition, Stormont questioned the need for high autonomy for robots in hazardous environments such as combat zones and identified factors contributing to human trust in robots [13]. In Unmanned Aerial Systems (UAS), vehicles are highly restricted in the national airspace, and are not allowed to autonomously release weapons in military battlefield operations, which illustrate lack of trust in autonomous systems.

The legal and moral ramifications of autonomous systems are of rapidly increasing concern today. For example, as self-driving cars become reality, in addition to the direct cybersecurity issues, we have broader issues of trust, reliability, and ethical decision making [14]. This leads to questions surrounding who to hold responsible if autonomous systems exhibit faults, experience security breaches, injure innocents or damage property. This calls into question whether or not autonomous systems should be allowed to make life and death decisions [13]. When autonomous systems, which are already limited in reliability and are lacking human trust, are victims of hacking attacks and security vulnerabilities, they become dangerous.

There is an inter domain routing protocol called BGP (Border Gateway Protocol) that each autonomous system in a network can follow to reach every block of an IP address. Although BGP is a secure protocol, it is still vulnerable to potential hijacking, malicious and cryptography related attacks. This possibility diminishes human trust in autonomous systems. In [15], a method called pretty good BGP (PGBGP) for detecting anomalies and responding to them is described. The approach can potentially detect and stop the propagation of invalid origin autonomous systems and invalid paths.

III. SECURE SOFTWARE DEVELOPMENT LIFE CYCLE

The Software development follows a software development life cycle, commonly referred to as a SDLC. A SDLC typically consists phases for Analysis, Design, Implementation, Testing and Maintenance. In the Analysis phase of a business application, potential stakeholders are identified and requirements are gathered from the customers, who are future users of the software. Requirements gathering involve customers meeting the business managers and analysts to state their expectations for the software. These requirements are analyzed and a requirements specification document is formulated and used throughout the life cycle of the software, including usage by the software design team, architects, developers, testers, and the end users of the software.

The software design phase involves the creation of a blueprint of the software, which is based upon the customer requirements. A document is created that defines the guidelines for the design of the system. The Implementation phase involves the coding activities required to create the software, following the customer requirements. The Testing phase follows, which tests the system for fault removal and to provide assurance that the system functionality adheres to requirements. Finally, a Maintenance phase is entered, in which future enhancements, bug fixes and version controls are undertaken. We assert that incorporating security concerns at every phase of the software development life cycle can significantly
To follow a secure software development life cycle, the software project requires knowledgeable developers who are current with security standards and follow secure coding practices. During the design phase, developers can also incorporate secure implementation by starting out with the basics, such as having the code that they wrote, reviewed by threat advisors, as well as managers and fellow developers. Code reviews improve the coding style of the developers, but also enhances code security by pin-pointing known secure coding errors. Review feedback and hacking demonstrations enhance code security [16].

According to Grégoire et al. [17], code reviews provide an opportunity to find security related bugs and fix them relatively early, and are also useful in teaching secure coding practices and security vulnerabilities to developers [17]. Other than code reviews, risk assessment at each phase, asset identification and valuation; threat modelling and categorizing threats are also important factors that should be taken into consideration. Figure 2, based on information in [18], depicts a model of a secure software development life cycle infused with security at every stage, as suggested by Jones and Rastogi [18].

Jones and Rastogi followed the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) protocol for categorizing threats to each attack target node [18]. Spoofing threats refer to the susceptibility of the target to their authentication information being illegallly accessed and used by another person. Tampering threats refer to user sensitive data being maliciously modified. Repudiation threats refer to users denying that they have performed illegal operations in a system that lacks traceability, unless proved otherwise. Information Disclosure threats involve sensitive personal information being exposed to users who are not authorized access. Denial of Service attacks refer to denying services to valid users, typically by flooding platforms with extraneous data. Elevation of Privilege threats enables unprivileged users to gain privileged access to a system, which makes it sufficient for them to destroy or compromise the integrity of a system.

Fig. 2. Secure Software Development Life Cycle

IV. SECURITY IN THE ANALYSIS PHASE

In To incorporate security during the analysis phase, it is essential that security requirements are generated as part of initial requirements gathering and analysis. If the requirements are gathered keeping security aspects of the software product formalized, other phases of the SDLC can be driven towards higher security. Nearly every software project is carried out in a time crunch for which security is given minimal to no attention. Defining security requirements in the requirements gathering use cases and performing an initial risk analysis encourages the elevation of consideration of security aspects in the rest of the SDLC phases [3].

In the requirement phase, Futcher and von Solms argued for the importance of gathering and including all known security-related information, including business and legal requirements [6]. They also define CLASP (Comprehensive, Lightweight Application Security Process), SDL (Security Development Lifecycle) and TSP-Secure (Team Software Process-Secure) for describing the capturing of security requirements as a best practice. We also emphasize the provision of security education and training for developers, acknowledging and endorsing the training, and providing a list of global security requirements that can be used as a baseline for software projects [6]. An excellent comparison of CLASP and SDL is presented by Grégoire et. al. [17].

Security being a non-functional requirement reveals that it is a crosscutting concern that can potentially impact the multiple functional requirements across a multi-component system. A crosscutting concern is a common functionality that supports different operations in a system, including authentication, authorization, communications, caching, logging, exception management, and validation. These functionalities span layers and tiers [19]. In most cases, requirements for security are specified in terms of achievability rather than in terms of the problem being solved. This results in lack of clarity as to how the security requirements affect functional requirements [20].

Haley et. al. used aspect-oriented software development crosscutting concepts and problem frames and presented an approach to deriving security requirements from them [20]. They identified threat descriptions, concerns, join points (locations of objects or assets which are being shared by functional requirements and threat descriptions) and vulnerabilities that are a composition of functional requirements and threats and are found at join points. With the help of several examples, one of them being the result of an application of CIA concerns conflicting with ease-of-use and revenue; the authors recognized, interpreted, and clarified conflicts of security requirements with each other. The representation of security requirements as crosscutting threat descriptions illustrates how they assist with the composition of these requirements with functional requirements. The approach allows general security concerns to be converted into instantiations that are closely related to functional requirements. Rosenhainer demonstrated an approach to identify and improve and enhance the overall quality of the software product.
document early crosscutting concerns at the requirements engineering level from an aspect-mining point of view [21].

Moreira et. al. focused on the need to clearly identify and separate non-functional concerns. These include quality attributes such as security, reliability, accuracy, and response time, and address their crosscutting nature, since these properties affect the entire system [22]. To manage conflicts that arise due to tangled representations, specifications, and code at the requirements stage of software development, they proposed a model consisting of three activities namely identifying, specifying, and integrating requirements. Their approach first identified all of the requirements and selected relevant quality attributes, followed by using use cases from the world of Unified Modeling Language for specifying functional requirements and using special templates for describing quality attributes [22]. This helps to identify all of the quality attributes that can crosscut functional requirements. Finally, they proposed a set of models that represented an integration of functional requirements with crosscutting quality attributes [22].

V. FUTURE WORK

The premises of this survey study were primarily limited to web software applications. Our next steps are to extend this work of surveying security issues, concerns, challenges and their solutions in hardware and networking, as they interface with software. We plan to proactively implement ways to secure software, hardware and networks and develop tools and techniques to prevent hacker attacks more broadly. Another is to survey security issues, concerns, challenges, and their solutions in emerging technologies that apply software engineering methodologies in cloud computing, social networking, safety-critical software applications, healthcare software applications, internet banking applications, mobile applications, telecommunication, and smartphone technology.

VI. CONCLUSION

Cybersecurity is of high importance in the development and usage of software applications. For a software application, from gathering requirements and through design, architecture, development, testing, quality assurance, distribution and actual usage by endpoint users, security is a core issue. Stakeholders, development teams, customers, and users must be attentive to security issues. We recognize that security is a non-functional requirement. From inception to delivery of a software product, security poses concerns of great importance, and heavily impact user’s trust in a software application and its continued usage.

The survey presents the core concepts from the literature dealing with security related issues, concerns, challenges, and their solutions from a software engineering perspective. We conclude that patching a software product for emerging issues is not enough and that there is a strong need to incorporate security requirements from the inception of the software life cycle. We also conclude that implementing security requirements at each phase in the software development life cycle and conducting risk analysis at each step is a promising contribution towards building a secure and robust software application.

REFERENCES


