Secure Agile Software:
A methodology and good practices for the software development life cycle

Carlos Manuel Gutiérrez Galán

Dr. Alejandra Guadalupe Silva Trujillo

Dr. Francisco Javier Torres Reyes

*M.S. in Computer Engineering*

*Facultad de Ingeniería*

*UASLP*

**Abstract**

Security standards are an important factor in various aspects of the development of software. Various investigations have found effective methods for producing and evaluating secure software systems, and various aspects of software security have been studied by applying security at various stages of the software development life cycle. Most security attacks take advantage of human errors, such as poorly chosen passwords and weak security settings, or software deployment failures. Therefore, the software must be developed in accordance with security requirements. Software vulnerabilities could be minimized by examining and monitoring every step of the system development life cycle. Security for a software system has always been addressed only in the production environment through security, such as firewalls, proxies, intrusion prevention systems, and antiviruses. An application must be security conscious in order to protect itself from security threats.

This implies that security must be built into the application.
1 Introduction

1.1 Software security

1.1.1 Software breaches

The term "software data breach" or "software breach" is used to denote an event where a particular piece of software is hacked, and its information accessed by an external individual or group of individuals. In 2019, there were more than one hundred reported breaches [12] that compromised hundreds of millions (and some speculations dare to say that it goes up to billions) of records[7]. E-mail addresses, passwords, full names, and social security numbers are amongst the most valuable Personally Identifiable Information that is sold in the black market, and thus the most desired by the previously mentioned individuals. In order to perform the software breaches, those individuals exploit software vulnerabilities, which are discussed in the next section.

1.1.2 Software vulnerabilities

As Ira Winkler and Araceli Treu say in the book "How to hack computers" [3]

Software vulnerabilities involve bugs in software. All software has bugs of one form or another. Some bugs cause the system to crash, some cause connectivity to fail and some do not let a person to log in. Some bugs create information leakage or elevate user privileges or grant otherwise unauthorized access.

These are security vulnerabilities.

If all software has bugs, and some bugs will cause security vulnerabilities, then there is a risk that any given piece software has vulnerabilities. Almost all devices have software in them, from the applications that they run[11], down to the very lowest levels in their firmware[6]; All devices have, then, the risk of being vulnerable.

1.2 Software development

Software development -in general- is a chaotic activity, often characterized by the phrase "code and fix". Software used to be written without an underlying plan, and the system’s design was cobbled together from many short term decisions. This plan works pretty well if the system is small, but as the system grows, it becomes increasingly difficult to add new features to the system. Furthermore, bugs become increasingly prevalent and increasingly difficult to fix. A typical sign of such a system is a long test phase after the system is “feature complete”. Such a long test phase plays havoc with schedules as testing and debugging is impossible to schedule.
1.2.1 The agile methodology

The original movement to try to change this way of developing software introduced the notion of a methodology. These methodologies impose a disciplined process upon software development, intending to make software development more predictable and more efficient. They do this by developing a detailed process with a strong emphasis on planning inspired by other engineering disciplines.[8]

In February 2001, a group of seventeen software experts got together in Snowbird, Utah, to discuss the growing field of what used to be called lightweight methods. They decided to use the term agile to describe this new breed of methods. They also wrote the Manifesto for Agile Software Development, setting out the values and principles of these agile processes.[9]

The manifesto consists of four principles that are derived from a broad range of software development frameworks, and represent the most valuable aspect of them [5]. Those principles are the following:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

1.2.2 The scrum framework

Scrum is an agile framework for developing, delivering, and sustaining complex products where multiple teams are involved. It consists of scrum teams and their associated roles, events, artifacts, and rules, which are responsible for binding the previous components. Each component within the framework serves a specific purpose and is essential to scrum’s success and usage.

1.2.3 The software development life cycle

The software development life cycle (SDLC) phases are somewhat ambiguous; they can vary depending on to whom you ask. For this work, they are going to be defined as shown in the figure 1 on page 3. These phases are described as follows:

- **Requirements**
  All the information that’s relevant to the project is collected from the customer and the interested parts.

- **Design**
  The requirements gathered are used to plan the different components of the software and the interactions between them and the outside.
• **Implementation**
The design plans are then translated into source code.

• **Maintenance**
Any issues or flaws that come to surface once the software is deployed on its target environment are fixed.

• **Testing**
The code is evaluated in order to detect possible flaws and to determine if it satisfies the requirements.

• **Deployment**
Once the code is approved, it is deployed into production.

![Figure 1: The phases of the SDLC in their proposed order](image)

1.3 Literature review

The development of secure software is one of the most critical issues that IT organizations are facing. Several studies have been realized in order to find methods to produce secure software. Software security was considered as only being part of software testing, but later, with the help of those researches, a consensus was reached, and it stated that security should be part of the SDLC. Therefore, various security aspects have been studied [2].

There are many critical factors involved in the development of secure software, e.g., design and implementation of robust authentication protocols, devising effective trust models and security policies. Despite these challenges, most of the security attacks are facilitated by human error[1] or flaws in the software implementation. Because of such errors, software systems need to be designed and implemented according to a security requirement.

Agile software development is the denomination given to a collection of processes, methods, and practices [4], and nowadays is the *de facto* standard in the industry. Scrum [10], which is one of the most popular agile frameworks, helps teams work together by defining roles, events, and artifacts that are part of the development effort. The objective of this work is to define the previously mentioned security requirement as a set of good practices and a methodology that can be applied in the different scrum events by the members in charge of the development.
2 Secure Agile Software

Based on the previous articles, we have concluded that a strong knowledge in both software engineering and software security is required, and it is why we propose for Carlos to take complementary courses using online platforms such as Pluralsight and Coursera. In addition to that knowledge, a software suite is going to be required for the automation of different tasks performed along the SDLC. For those, we propose the use of Jenkins, which is an open-source automation server.

Given the lack of knowledge about this software, additional courses will be required in order to use it.

2.1 Schedule of activities

<table>
<thead>
<tr>
<th>Activity or course</th>
<th>Aug-Dec ’20</th>
<th>Jan-Jun ’21</th>
<th>Aug-Dec ’21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Coding Practices Specialization</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cybersecurity Specialization</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building a Modern CI/CD Pipeline with Jenkins</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experiment with code reviews</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Deployment of Jenkins on a local machine</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References


