A Comparative Overview of the Most Common Methodologies for Secure Software Development

Golnaz Aghaee Ghazvini

Computer Engineering Department, Science and Research Branch, Islamic Azad University, Tehran, Iran.

Aghaee@iauda.ac.ir

Abstract

Security is a property of an entire system in context. Rather than a software product; so a thorough understanding of system security risk analysis is necessary in the entire software life cycle for a successful project. This aspect is strongly related to costs, risks and reputation of an organization. Focus on security in the software development life cycle can be divided into two distinct categories: Internal security and External security. Internal security of software depends on the security of codes and applications used in software development. External security of software includes measurements to maintain security in the interaction with the external environment. In recent years, many methods have been developed to maintain internal security of software. The main goal of this paper is to present and compare the most common methodologies provided to ensure the internal security of the software systems.

Keywords: Software, Security, Internal Methodology, Development Process.

I. Introduction

Secure software is that which is in and of itself robust against attack. This means that the software will remain dependable even in situations where dependability is threatened. In other words, this software lacks faults or weaknesses that can be exploited either by human attackers or malicious codes. Compared to other software properties, security is not yet well understood: In the software development life cycle, developers or users are not able to determine with 100 percent certainty that the software is secure or not. Software security is a dynamic property that is why authors want to propose secure-by-design principle by using innovative and proven development approaches in the whole system life cycle to maintain a high security level. The software that is secure in a particular environment within a particular threat landscape may no longer be secured if that environment or threat landscape changes or if the software itself changes. In terms of “testability,” security is also difficult to gauge. Software testers can run 10,000 hours of testing and ultimately be very confident that the software passing those tests will operate reliably. This cannot be said about the software’s security. Security testing techniques for software are still immature collectively representing an incomplete patchwork of coverage of all security issues that need to be tested for (Mercedes et al., 2007). Security experts usually focus on ensuring the security of the software associated with the external environment. To achieve this goal, professionals used different access controls to prevent the unauthorized access for confidential data and applications. In this paper, we focused on internal security methodologies for developing secure software systems.

The rest of the paper is organized as follows: Section II describes model driven methodologies for developing secure software systems. Section III and IV reviews formal and risk driven methodologies for secure software development. Section V, discusses about the features of the
methodologies to characterize the use of each methodology and Section VI concludes the paper and point out direction of future research.

II. Model Driven Methodologies for Developing Software System

Model is a representation of system or real-world entity, and it is the heart of the system design (Kaugers et al., 2010). The modeling of systems in many engineering disciplines, particularly in software engineering, is important. But modeling alone is not useful, and certainly does not come for free: Sufficient time and expertise are needed to build the optimal models, and efforts should be made to keep them synchronized with end products. Model driven development has been introduced in order to increase the quality and thereby the security of software systems (Kasal et al., 2011). In model Driven development, system designs are done using graphical modeling languages like UML, and system artifacts as code and configuration data are automatically generated from the models.

A. Model Driven Architecture

Model Driven Architecture (MDA) was developed by Object Management Group (OMG) in 2001. OMG has been an international, open membership, nonprofit computer industry consortium since 1989 (Magobleh et al., 2014). This methodology helps to specify software systems specifications regardless of hardware infrastructure and platforms for their implementation. This means that, it uses models for software system development. For this purpose, it provides a set of guidelines for structuring specification of the main system components, which are expressed as models. These models can be used in other phases of software development.

MDA consists of three major types of models (figure1):

- Computation Independent Model (CIM). It involves specification of system functionality.
- Platform Independent Model (PIM). A formal specification of the structure and functionality of a system that abstract regardless technical details. Usually UML used to describe the system specification.
- Platform Specific Model (PSM); Specifies how the functionality specified in a PIM is realized on a particular platform. Platform specific UML profiles are usually used for this purpose.
As shown in fig1, Computation Independent Model (CIM) is mapped on Platform Independent Model (PIM) and then PSM is generated from PIM. The Platform Specific Model (PSM) may then be coded manually, the frame code is generated automatically and the methods are coded manually, or all the codes generated automatically.

MDA has several advantages which are very important for business environments. Some of these features are listed below (Alfrod, 2014 and Moral, 2014)

- Fast adaption with changes into the model
- Portability of Architecture among several platforms
- Easy to identify and correct errors
- Interoperability, MDA can improve interoperability to enhance product data exchange between heterogeneous systems
- Reusability
- Increases system maintainability and security level of system, and it will have a significant impact on IT systems in future

B. Model Driven Security

Model Driven Security is a methodology that can model security requirements in a high level of abstraction. Security requirements can be obtained from some sources such as stakeholders and other available sources of security information. The security information is expressed in Domain Specific Languages (DSL), and the security rules will be extracted from DSL with as little human intervention as possible. This methodology uses secure UML to generate the system architectures from the system models. The secure UML is an extension to the UML for integrating security related information in UML specifications. This information can be used for the model based security engineering. Security information will be added using stereotypes and covers many security properties including secure information flow, confidentiality and access control.

Many researchers believe that MDS can have a significant impact on information. The security infrastructure is required to become increasingly real-time, automated and adaptive to changes in
an organization and its environment. The information security infrastructure can support the adaptive changes that are essential for many recent technologies being built to support adaptive changes (e.g. Service Oriented Architectures).

According to the above, some of the most important features of MDS are listed below (Basin et al., 2006 and Luciu et al., 2014):

- Auto-generate domain boundary security policies
- Support adaptive changes
- Reduce the number of human errors
- Increase the flexibility of policies
- Lack facilities for automatic conversion model to code

III. Formal methodologies for developing secure software system

Formal methodologies are available in several different levels:

- Formal specifications: They are mathematically based on the specifications used to describe the behavior of the software system. These specifications have a formal syntax and are able to be used to elicit important information about software system.
- Formal development: Using formal methods, developers can produce software programs in a formal manner. This may be the most appropriate to produce a high quality software system.

Methodologies that will be discussed in this section use formal methods for software development. Formal methods are mathematically based on techniques for specifications of requirements and verification of software. The use of the formal methods for software development is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design.

A. Software Security Assessment Instrument Methodology

This methodology for developing secure software uses tools and information resources available. Online vulnerability database is the first information source used by this methodology. The security assessment instrument includes a vulnerability matrix that is commonly called Vmatrix. The information in this matrix is necessary to obtain database vulnerabilities. Database vulnerabilities include information about different vulnerabilities, exploits and signatures. The information of this matrix displays the aggregate number of real vulnerabilities and potential vulnerabilities for the software being developed.

The security checklist is another source used in this methodology. The checklist can be used to help reduce the security vulnerabilities and to verify the correctness of software during the software development process. It also uses a model checking called flexible modelling framework. Model checking is an increasingly important software quality assurance technique that can complement existing testing and specification methods (Notander et al., 2013 and Beger, 2007). This methodology provides a property-based testing tool to test the software system.
which uses the security properties specified in the security checklist or flexible modelling framework as a basis to test the software (Gilliam et al., 2003).

B. **Correctness by Construction Methodology**

Correctness by Construction is a radical, effective methodology for building software with demonstrable integrity for security applications. This approach, which uses formal methods at all of the software life cycle, applies formal description language to describe the system specification and unequivocal programming language and bug prevention that allows precise analysis very early in the software development process. Most researchers believe that the development methodology that focuses on bug prevention rather than bug detection can both improve quality and save time. For this purpose, this methodology uses a programming language whose source code has a precise meaning: This makes it possible to provide tool support in the form of static analyzers that can be applied very early in the coding process before dynamic testing begins. This method follows that easily validated software must be designed and produced. The code should be simple and directly traceable to the specification. Because the test cases are also obtained from the specification, the process of verification is simplified. As shown in figure 2, the method eliminates feedback from the various validation activities that can clearly affect any earlier deliverable and cause re-entry to any previous activity. However, because of the high cost of it, only security critical components of software systems are developed by using this methodology (Lamprecht et al., 2013)

![Correctness by Construction activities](image)

**Figure 2.** Correctness by Construction activities

IV. **Risk Driven Methodology for developing secure software system**

The risk driven methodology helps developers follow a middle path, one that avoids wasting time on techniques that help their projects only a little, but ensures that project-threatening risks are addressed by appropriate techniques. The key element of the risk-driven methodology is the promotion of risk to prominence. The main goal of this approach is summarized below:

- Identification and prioritization of risks
- Selecting and applying a set of techniques to increase security and reduce the risk of software
- Evaluation of risk reduction

In the next section, one of the most common risk-driven methodologies for secure software development has been described.

A. **Security Requirement Engineering Process**
The Security Requirements Engineering Process is a risk based methodology for the establishment of the security requirements in the secure software system development process. It describes how to integrate the common criteria into the software life cycle model together with the use of a security resources repository to support reuse of security requirements, assets, threats and countermeasures. The methodology has been designed to integrate security concepts in the initial phases of the development process. This methodology consists of 9 phases (figure3): Agreement on security definition, Identification of Vulnerability, Identification security objectives and dependencies, Risk assessment, Eliciting security requirements, Categorization of and prioritization of requirements, Requirement Inspection and Repository improvement. As shown in figure3, each phase may include much iteration, and each iteration is like a mini-project and it may contain all the core workflows (requirements, analyses, designs, implementation, and tests), but with different emphases depending on where the iteration is in the life cycle. One of the striking features of this methodology is being iterative and incremental. Because of this feature, at the same time, all of the criteria components are integrated into the software life cycle and Software Quality Assurance activities are performed along all the phases of the software development life cycle (Mellado et al.,2006 and Mellado et al.,2007 and Hijazi et al.,2012).

![Figure3. Security Requirement Engineering Process methodology](image)

V. Comparison and Discussion

This paper focused on the well-known methodologies for secure software development. Each of these methodologies has the specific features that distinguish methodologies from the others. In this section, we intend to elucidate the benefits of each methodology. By doing so, developers can select the suitable methodology for software development easily. Model driven methodologies in section 2, were suitable for the large, complex software systems in which teams consisting of large number of people participate. Because of uses abstractions models as
the main artifacts during the development process, it is suitable for enhancing productivity, reusability, portability, maintainability and interoperability of software system. Model driven methodologies allow developing and deploying software projects without having previous knowledge about specific programming languages or platform technologies. They are appropriate for distributed systems, which are run on different platforms. These systems require integration at different points in their deployment. Formal methodologies as discussed in section 3, allow the development of complex software under a firm mathematical foundation resulting in high quality in software. Hence, the resulting systems are consistent, complete and unambiguous. These systems can also be validated automatically before implementation level. Finally, Risk driven methodologies in section 4 are suitable for such systems which are vulnerable from threats. In table 1, some of the key features of these methodologies such as Ease of use, Domain application, Support tools and scalability, are listed.

<table>
<thead>
<tr>
<th>Ease of use</th>
<th>Domain Application</th>
<th>Support tools</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDA **</td>
<td>Distributed and platform independent</td>
<td>-</td>
<td>Large system</td>
</tr>
<tr>
<td>MDS **</td>
<td>General software system</td>
<td>CASE tools</td>
<td>Large system</td>
</tr>
<tr>
<td>SAI *</td>
<td>Critical software system</td>
<td>PBT tools</td>
<td>Small system</td>
</tr>
<tr>
<td>CbyC *</td>
<td>Critical software system</td>
<td>-</td>
<td>Small system</td>
</tr>
<tr>
<td>SREP **</td>
<td>Risk based software system</td>
<td>SREP tools</td>
<td>Medium system</td>
</tr>
</tbody>
</table>

Note 1: The notations, *, indicates that methodology is easy to apply and the notation,***, indicates the difficulty of applying the methodology in secure software development.

VI. Conclusion
In this paper an overview of most common methodologies for secure software development was presented. These methodologies are generally divided into three categories. Of course, there are some categories that are not mentioned in this paper, because our main purpose was to compare between the most popular software methodologies for secure software development. The comparison is done based on key parameters mentioned in the text. This can guide developer to choose an appropriate methodology according to the software requirements. As future work, it is necessary to develop a new category based on more secure software development methodologies and also for some methodologies more Tools should be developed to automatically transform requirements to secure models.

References


