ARTICLE IN PRESS

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Cloud security engineering: Early stages of SDLC

Shadi A. Aljawarneh[a,*], Ali Alawneh[b], Reem Jaradat[c]

[a] *Software Engineering Department Jordan University of Science and Technology, Irbid, Jordan*
[b] *MIS & CIS - Faculty of IT, Philadelphia University, Jordan*
[c] *Reem Jaradat, Faculty of IT, Isra University, Jordan*

ARTICLE INFO

Article history:
Received 1 July 2016
Received in revised form
21 August 2016
Accepted 4 October 2016
Available online xxxx

Keywords:
Software engineering
Cloud security
SaaS
Architectural decisions
Applicability
Security vulnerabilities

ABSTRACT

Security vulnerabilities and defects are results of poorly constructed software that can lead to easy exploitation by the cyber criminals. A large number of Cloud software systems are facing security threats, and even the sophisticated security tools and mechanisms are not able to detect it. Such prevailing problem necessitates the monitoring and controlling of the software development process and its maintenance. Security is considered to be one of the nonfunctional requirements that have significant effect on the architectural designing of the Cloud Software as a Service (SaaS). In addition, there is prevalence of differential views between the two software engineering concepts, i.e., conventional and contemporary and then this presents a significant challenge for the software development team to deal with security at the implementation and maintenance stage of the SDLC. Thus, we have discussed a real world case study includes 103 failed real cases that were generated manually or automatically by real applications through various testing techniques and we have illustrated some preliminary results. The evaluation results showed appearance of a significant number of security vulnerabilities in the early stages of Cloud Software/Service Development Life Cycle (CSDLC). Hence, this needs to be maintained in advance. Based on such results, this paper presents a generic framework to deal with such security at the early stages of the CSDLC. This framework aims at adding an extra security level at the early stages of the CSDLC, which has been further illustrated by a case study showing the applicability of the framework.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Since the last decade, the software engineering field has witnessed software architecture to be the most researched topics of all times. Many researchers have talked off the fundamental keys to software architecture is that the architectural decisions [1–3]. Cloud software architecture has evolved into a decision-centered perspective from a structural representation [4].

The Software as a Service (SaaS) defines architectural design as an identifying parameter for the Cloud service sub-services and the framework for the control and communication of sub-services. At the early stage of the service design process, the architectural design is performed taking into consideration the relationship between the final service specification and design processes [5].

Furthermore, the service goal at the analysis phase can be met by such noble architecture that provides the essential functional requirements. This assists in detecting and avoiding the security vulnerabilities at the analysis phase. When the architectural design reaches the implementation phase, the service designer is able to decide the functional requirements involving a security technique.

To resist the cyber security attacks, the designers and customers need to work in collaboration. To bring out the stakeholders' perspective, a development team was built with the assistance of the agile methods. Each individual perspective can result in delivery of a potential solution [5–7]. However, architectural conflict might result between the security and availability, making security setting a difficult process, particularly at the implementation and deployment phases [5,7].

The architectural design of SaaS is a creative process, which varies depending on the type of service being developed. For example, the online payment service process differs from that of the online learning service, since security as non-functional requirements have different policies, standards and priorities [5,7].

However, there are certain standard decisions that span across all designing processes, such as the decision in which all services need to have a security kind at different levels. As a functional requirement, the user needs to correctly, reliably and securely authenticate its service.

* Corresponding author.
E-mail addresses: saaljawarneh@just.edu.jo (S.A. Aljawarneh), aalawneh@philadelphia.edu.jo (A. Alawneh).

http://dx.doi.org/10.1016/j.future.2016.10.005
0167-739X/© 2016 Elsevier B.V. All rights reserved.

However, security requirement is one of the primary handlers of architectural decision-making. Such nonfunctional requirement is enhanced or impaired by architectural decisions (ADs). Including or excluding parts of the architecture are considered to be its limitations (for e.g., logical components or technologies). Determination of effect of AD or affected architectural parts by a security constraint is considered a difficult task. But at completion, identification of the nonfunctional requirements becomes difficult along with the forced constraints including all ADs.

Often the requirements engineering faces problems like partial requirements specification stipulated without any analysis or the one limited to the functional end-user requirements, which are difficult to prove [8]. Hence, the addition of security occurs at post-development of the system [9]. However, security requirements are found to be constraints on the system functions, operationalizing one or more security requirements [10].

Secure design consists of component security architecture, physical, and logical. The physical security architecture is concerned with data model specifications and structures (i.e., signatures, messages, tables), rules (i.e., procedures, actions, conditions), security mechanisms (i.e., encryption, access control, virus scanning), security technology infrastructure, and time dependency (i.e., time intervals, events). The logical security architecture specifies confidentiality and integrity protection, domains, entities, and security processing cycle (registration, login, and session management). In addition, the security of the overall system can be affected by the component integration into the system. Thus, analyzing the component's security, as well as security of each identified component and all other components has been necessitated [11].

The topic of the paper is of interest especially at a time when cloud computing is an evolving paradigm changing the way digital files are stored shared and being accessed. The idea of proposing a framework can support the elicitation of security requirements at an early stage of the system development. In addition, such framework can support the building of service architecture design correctly and reliably. The proposed framework is a 5-level framework. The framework addresses security at service level, storage level, record level followed by hypervisor and datacenter levels. On the whole the proposed framework consists of 17 components. These components form the entire framework and are incorporated to be the components in each Cloud protection level.

The case study discusses the applicability of proposed Cloud security framework. The security feature is added at early stages of SDLC in the Cloud deployment. This paper is arranged as follows. Section 2 discusses the related works. Section 3 discusses the results of a real case study. Section 4 presents an overview of the proposed framework. At last, Section 5 includes conclusion of the paper.

2. Related work

In a previous study, the researchers proposed a method to diminish the security requirements gap that combines software engineering approaches with the principles of state-of-the-art security engineering. This resulted in establishment of a precise arrangement between the security engineering principles, nonfunctional goal, and implementation of security architecture. The method proposed designing of security architecture of a system that is based on a small, precisely defined, and application-specific trusted computing base. However, the Cloud systems and services could not be fit by the proposed method. Therefore, if the Cloud security requirements are not taken into consideration, then the coming up with a precise definition of the Cloud system architecture will not be possible.

Based on the key security considerations, another research presented a practical security model by looking at the number of infrastructure aspects of Cloud Computing, such as Utility, SaaS, Platform and Managed Services, Web, Service commerce platforms and Internet Integration. Such model can be applied as a proposed shared security approach in the system development life cycle (SDLC) focusing on the plan-built-run scope, with added security at the implementation phase.

In another study conducted by Moradian and Håkansson [12] highlighted the development of a multi-agent security support system that helped in emergence of new systems along with current system modifications. Therefore, the requirement goals are verified and validated by the multi-agent system during different SDLC phases. This whole process involves information searching and mapping. Information searching with relation to the security and project documents, such as threat list, security risks and security vulnerabilities that is performed by the software agents. Cases and mapping of attack patterns are used for comparing and analyzing the system requirements. The multi-agent system support integrity, confidentiality, availability, accountability, and non-repudiation. Such a system enhanced the security and supported the developers in every phase of SDLC, throughout the engineering process. However, the security was not considered by the developed system at the early stages of system development.

Mouratidis et al. [13] proposed an agent oriented software methodology known as the Tropos by using principles of authorization, access control, and availability. It is a security oriented extension defining the security constraints as well as the secure dependencies. The secure Tropos process allows model and design validation.

3. Real test cases: data collection and analysis

In this study, we used a dataset collected by The Software Assurance Reference Dataset (SARD). The Dataset source is from https://samate.nist.gov/SARD/testsuite.php. The Software Assurance Reference Dataset (SARD) provides a set of known security flaws to its users, software security assurance tool developers, and researchers. This enables the end users and tool developers to evaluate their tools and test their methods, respectively. These test cases (designs, source code, binaries, etc.) are derived from all phases of the SDLC. The dataset consists of test cases such as wild (production), academic (from students) and synthetic (written to test or generated). The dataset also encompasses a wide variety of possible vulnerabilities, platforms, languages, and compilers. It is considered as a large-scale effort by gathering test cases from many of the contributors.

We have collected and analyzed the dataset and we have taken randomly a number of real test cases. Therefore, our improved Dataset is a real world case study includes the 103 worst failed real cases that were generated manually or automatically by real applications through various testing techniques as shown in Table 1 and we have illustrated some preliminary results. In addition, we have classified each failed test cases and added the problem phase column and then solution column as shown in Table 2.

A failed test is a service failure, for example, the service could not correctly deliver the expected result. In case of the test has been conducted in certain conditions, then the service failure depicts that a service has an error or fault such as requirements and design flaws.

In Table 1, a random sample includes the test case id, descriptions and weaknesses. The mentioned failed test cases have generated manually or automatically through a number of software applications. As shown in Table 2 this table has been classified by finding the Problem phase at the Cloud Software Development Life Cycle, the appropriate solution phase and the solution for each test case mentioned in Table 1. Therefore, this table has been developed to include the proposed solution for each

Download English Version:

https://daneshyari.com/en/article/4950388

Download Persian Version:

https://daneshyari.com/article/4950388

Daneshyari.com