Evolution of Security Requirements Tests for Service–Centric Systems

Michael Felderer, Berthold Agreiter, and Ruth Breu

Institute of Computer Science, University of Innsbruck, Austria
{michael.felderer,berthold.agreiter,ruth.breu}@uibk.ac.at

Abstract. Security is an important quality aspect of open service–centric systems. However, it is challenging to keep such systems secure because of steady evolution. Thus, security requirements testing, considering system changes is crucial to provide a certain level of reliability in a service–centric system. In this paper, we present a model–driven method to system level security testing of service–centric systems focusing on the aspect of requirements, system and test evolution. As requirements and the system may change over time, regular adaptations to the tests of security requirements are essential to retain, or even improve, system quality. We attach state machines to all model elements of our system-and test model to obtain consistent and traceable evolution of the system and its tests. We highlight the specifics for the evolution of security requirements, and show by a case study how changes of the attached tests are managed.

1 Introduction

Security requirements testing \[1\], i.e., the dynamic validation of security requirements such as authentication or integrity, is of high interest, especially for service–centric systems which are open. Basically, a service–centric system consists of a set of independent peers which provide and call services \[2\]. Service–centric systems are widely used, and they can be subject to modifications which may harm their security.

The work at hand shows how the evolution of a system, its infrastructure or its requirements influence security requirements and their tests.

According to \[3\], security requirements can be classified as follows:

- **Confidentiality** is the assurance that information is not disclosed to unauthorized individuals, processes, or devices.
- **Integrity** is provided when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.
- **Authentication** is a security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual’s authorization to receive specific categories of information.
- **Authorization** provides access privileges granted to a user, program, or process.
- **Availability** guarantees timely, reliable access to data and information services for authorized users.
- **Non-repudiation** is the assurance that none of the partners taking part in a transaction can later deny of having participated.

This paper investigates the test evolution of security requirements, and regression testing based on the above classification. Note that our approach is independent of the concrete security requirements classification and can be based on other classifications, e.g., [4] as well. Defects or deviating behavior is normally detected by checking specific properties of a system during execution. These checks are called **assertions**. Security requirements are an adequate source for the definition of such assertions and provide additional constraints on functional requirements. Current approaches mostly tackle penetration or vulnerability tests and aim at the automatic generation of tests. We instead want to cover security requirement tests, called **security tests** in the sequel, from an acceptance point of view, i.e., testing whether security requirements are satisfied with regard to a functional requirement.

Our security test methodology takes into account that it is practically impossible to test security requirements completely, i.e., to provide evidence that security flaws are absent [5]. Instead of automatically generating test cases to be executed on the system, our approach allows for defining dedicated scenarios, i.e., functional tests, in which the compliance with security requirements is observed. We consider evolving scenarios and the consequences for security testing.

The remainder of this document is organized as follows. Section 2 details our testing approach by explaining our metamodel and the security test evolution process. The description of a case study in Section 3 applies the process to a real-world example. Finally, Sections 4 and 5 present related work and conclusions.

## 2 Test Evolution Methodology

On a very abstract level, we can distinguish among three different types of evolution for service-centric systems: evolution of **requirements**, evolution the **system**, and evolution of the **infrastructure**. For instance, the integration of a new security requirement is a requirements evolution, the adaptation of a service interface is a system evolution, and the modification of the implementation of a service or its deployment environment is an infrastructure evolution. Obviously, a change in the requirements may trigger changes in the system design or infrastructure. But a change in the design or in the infrastructure may also occur independently of a change in the requirements. Additionally, also changes on tests may occur.

In our approach, we attach a state machine to each changeable artifact, that defines its actual state, and triggers resp. receives events to compute new states. Changes in requirements are indicated by triggers on requirement elements and changes of the infrastructure or the system are indicated by triggers on service elements. Also tests itself have a state. Following the widely used classification in [6], the **type of a test** can be **evolution**, for testing novelties of the system,