Information and Software Technology 52 (2010) 274–295

Contents lists available at ScienceDirect

Information and Software Technology

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

Automated verification of security pattern compositions

Jing Dong *, Tu Peng, Yajing Zhao

Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA

ARTICLE INFO

Article history:
Received 11 July 2008
Received in revised form 3 October 2009
Accepted 5 October 2009
Available online 14 October 2009

Keywords:
Design pattern
Security
Logics
Process algebra
Model checking

ABSTRACT

Software security becomes a critically important issue for software development when more and more malicious attacks explore the security holes in software systems. To avoid security problems, a large software system design may reuse good security solutions by applying security patterns. Security patterns document expert solutions to common security problems and capture best practices on secure software design and development. Although each security pattern describes a good design guideline, the compositions of these security patterns may be inconsistent and encounter problems and flaws. Therefore, the compositions of security patterns may be even insecure. In this paper, we present an approach to automated verification of the compositions of security patterns by model checking. We formally define the behavioral aspect of security patterns in CCS through their sequence diagrams. We also prove the faithfulness of the transformation from a sequence diagram to its CCS representation. In this way, the properties of the security patterns can be checked by a model checker when they are composed. Composition errors and problems can be discovered early in the design stage. We also use two case studies to illustrate our approach and show its capability to detect composition errors.

1. Introduction

With increasing security attacks, security becomes a critical requirement for successful software system development. Studies [37] have shown that design flaws and errors are commonly the main source of security holes that are explored by attackers. Therefore, secure software architecture and design are at the heart of software security. Security becomes one of the most important factors that affect software quality. Security patterns [33,35] document good practices to solve security problems arising frequently in software development and encourage reusing expert solutions. A security pattern is a recipe of solving a particular security problem. It is a design pattern [19] that generally describes a group of participants as well as their relationships and collaborations, which achieve some security goals. Each participant in the group is defined generically in terms of the role it plays in the security pattern. The benefits of security patterns include the reuse of security design solutions instead of the reuse of just a piece of code, documentation of expert design experience, recording of security design tradeoffs, capturing of security decisions, and improvement of communication.

Multiple security patterns can be used in large software systems to solve many security problems. Combining security patterns may help to reuse expert solutions on different security problems in the same system. While each security pattern describes expert experience on solving a particular security problem, the composition of these security patterns is not always a good solution. There can be inconsistencies among security patterns such that some critical security properties may no longer hold. The inconsistencies between security patterns may cause problems in the design. Discovering these problems and errors early in the design stage is important because such design errors are very difficult to find and correct when they are transformed to implementation errors. Analysis techniques that help to find such design errors are crucial to the quality of the software systems.

There are several automated verification techniques, such as model checking and theorem proving. Model checking has been initially applied in the hardware community to verify safety and liveness properties [3,5,16]. It has also been used in the software community, e.g., in the verifications of web service composition [20,21,8], in distributed cache coherence analysis [38], in hypermedia applications [11], and in security property analysis [23]. In this paper, we use model checking techniques to analyze the consistency of security pattern compositions. More specifically, we formally specify the behavioral aspect of the security patterns in the Calculus of Communicating Systems (CCS) [28], as well as the properties of each security pattern. We provide a general rule for specifying security pattern behavior modeled by a sequence diagram in CCS. We define the synchronous message, asynchronous message, and alternative flows of a UML sequence diagram and transform them into CCS specifications. We also prove the faithfulness of the CCS specification with respect to the sequence

* Corresponding author.
E-mail addresses: jdong@utdallas.edu (J. Dong), txp051000@utdallas.edu (T. Peng), yxz045100@utdallas.edu (Y. Zhao).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.10.001

diagrams. A model checker is used to perform the analysis and check whether the characteristics of each security pattern still hold after they are composed. Our analysis results show that our approach is able to find the design errors that may lead to security holes and flaws.

The remainder of this paper is organized as follows: the next section describes our analysis techniques on security pattern compositions. Section 3 presents two case studies to illustrate our approach and show the discovery of several subtle security problems in the design. The last two sections cover related work and conclusions.

2. Our security analysis techniques

Many security patterns have been identified to solve the recurring security problems, such as authentication, authorization, and confidentiality during communication. One of the common security goals is secure communication between two parties. Unsecured communications are often exposed to eavesdropping, spoofing, sniffer, and replay attacks. The replay attacks copy the legitimate transactions and resend them. The sniffer attacks just capture sensitive information for later use. Many of these attacks can be categorized as man-in-the-middle attacks which cannot only harm the unsecured network but also VPN where data is exposed at the end points. This exposed data is still subjected to disclosure, modification, or duplication. Some of these attacks are easy to carry out, even for novices. As a consequence, these attacks may result in huge losses for businesses that need to communicate sensitive data.

In this section, we first describe model checking techniques. We then introduce our approach on modeling the behavior of a security pattern, which may then be checked by a model checker. We describe a general way to specify security pattern behavior in CCS and prove the faithfulness of the transformation from the sequence diagrams of a security pattern to our specification.

2.1. Model checking

In order to analyze the compositions of security design patterns, we apply model checking techniques. Model checking is a method of verifying algorithmically a formula against a logic model [5]. This verification technique can be automated by model checking tool (model checker). In our case, we assume a logic model representing the security patterns and their compositions, as well as a logic formula representing a property of these patterns. In order to determine whether the pattern-based system satisfies the properties, it is checked whether the formula holds in the logic model of security pattern compositions.

There are several model checking tools, such as SMV [25], SPIN [22], XMC [30], and CWB-NC [40]. In this paper, we will concentrate on the CWB-NC model checker since we can use it to specify security patterns in process algebra and analyze their composition by model checking. CWB-NC is a software tool which is capable of not only model checking but also behavioral equivalence verification. As a model checker, CWB-NC requires the user to specify the systems in CCS and the temporal properties with GCTL [29,40] (extension of computational tree logic, CTL [17]). It will then check whether the system satisfies the properties. In behavioral equivalence verification mode, CWB-NC requires the user to specify two systems in CCS. It will then check whether the two systems are equivalent under certain constraints. In this paper, we will use CWB-NC as a model checker. The syntax of CCS is as follows:

spec: *binding_list*
binding: "*proc*" *id* "=" *agent* |"*set*" *id* "=" *id_set*
agent: "*nil*"|*id* |*act* "." *agent* |*agent* "+" *agent* |*agent* "|" *agent* |*agent* "\" *restriction* |"(" *agent* ")"
act: *id* |"'" *id*
restriction: *id_set*

where the system specification consists of a list of bindings. Each binding is a formula, which either defines a process by using keyword "proc" or defines a set by using keyword "set". Each agent defines an expression of actions and operators. An action name is a user defined id, which represents a system activity. The prime symbol ('), which is placed in front of an action name, denotes that this action is an output of the system. The operations between agents include sequential composition ".", non-deterministic choice "+", parallel composition "|", and restriction "\". Restriction is used together with parallel composition, to denote that the messages being restricted are internal messages. Let us consider an example specification of the behavior of the Observer pattern [19] in CCS as follows:

```
* observer *
proc OBSERVER=OSUBJECT|OOBSERVER\{osetstate,oupdate}
proc OSUBJECT=osetstate.ochangestate.'notify.
OSUBJECT1
proc OSUBJECT1=notify.'oupdate.OSUBJECT
proc OOBSERVER=ochange.'osetstate.OOBSERVER+oupdate.
ogetstate.OOBSERVER1
proc OOBSERVER1=OOBSERVER
```

The system specification of the Observer pattern consists of two processes, OSUBJECT and OOBSERVER. The interaction of these two processes is that the OOBSERVER process sends out osetstate message to the OSUBJECT which changes its state and sends out oupdate to all OOBSERVERs. The OOBSERVER processes perform action ogetstate to update their states and thus keep it consistent with the OSUBJECT.

Temporal properties are expressed in GCTL, which is an extension of CTL. The syntax of GCTL is

$$S ::= p \mid \neg p \mid S \wedge S \mid S \vee S \mid \mathrm{A}p \mid \mathrm{E}p \mid \mathrm{G}p \mid \mathrm{F}p$$
$$P ::= \theta \mid \neg\theta \mid S \mid P \wedge P \mid P \vee P \mid \mathrm{X}P \mid P \cup P \mid PRP$$

where S is state formula, P is path formula. p is atomic proposition, and θ is atomic action proposition. A is a universal quantifier which means that the formula is true in all paths starting from the current state. E is an existential quantifier which means that there exists a path following the current state, such that the formula is true. G is a path universal quantifier which means that the formula is true for all the states along the path from the current state. F is a path existential quantifier which means the formula is true in some state in the path from the current state. X is a path quantifier which means the formula is true in the next state in the path from the current state. G, F and X are always used together with A and E. We illustrate the use of quantifiers in Fig. 1.

2.2. Overview of our approach

Fig. 2 illustrates the main characteristics of our approach to analyzing security pattern compositions. Initially, each security pattern is formally specified using CCS. The security pattern specifications are generic in the sense that they capture good design practice in a domain-independent way. These declarative representations, which constitute the models of the security patterns, are then instantiated into concrete domain-specific representations. In this way, security design practice can be reused. The instances of security patterns are integrated to form a model Σ of the composition of the security patterns, which is then submitted