Computing (2015) 97:691–711
DOI 10.1007/s00607-015-0445-x

Specification, verification, and quantification of security in model-based systems

Samir Ouchani · Mourad Debbabi

Received: 30 August 2013 / Accepted: 17 February 2015 / Published online: 28 February 2015

Abstract Modern systems are more and more complex and security has become a key component in the success of software and systems development. The main challenge encountered in industry as well as in academia is to develop secure products, prove their security correctness, measure their resilience to attacks, and check if vulnerabilities exist. In this paper, we review the state-of-the-art related to security specification, verification, and quantification for software and systems that are modeled by using UML or SysML language. The reviewed work fall into the field of secure software and systems engineering that aims at fulfilling the security as an afterthought in the development of secure systems.

Keywords Security · Modeling · Specification · Verification · Theorem proving · Model-checking · Temporal logic · Vulnerability · Attack · Security metrics · Security engineering · UML · SysML

Mathematics Subject Classification 68Nxx

1 Introduction

Integrating security concerns into software engineering practices [1], also known as secure software engineering, is a relatively recent research trend that aims at taking

S. Ouchani (✉)
Interdisciplinary Centre for Security, Reliability and Trust (SnT),
University of Luxembourg, Luxembourg, Luxembourg
e-mail: samir_ouchani@yahoo.com; samir.ouchani@uni.lu

M. Debbabi
Concordia University, 1455 de Maisonneuve Blvd. W., Montreal H3G 1M8, Canada
e-mail: debbabi@ciise.concordia.ca

security into consideration as a separate layer to build with the system as an afterthought. It discusses techniques and methodologies to provide support for the analysis and the design of security requirements and properties as well as for the specification of security aspects in the developed systems.

A major challenge in software and systems development process is to advance error detection at early stages of their life-cycles. It has been shown that the cost of repairing a software flaw during maintenance is approximately 500 times higher than fixing it at early design phases [2]. Yet, another more ambitious challenge is to accurately specify, express and measure the security level of a product based on its design artifacts. Various techniques have been proposed for the verification of software and hardware such as model checking [3], type checking, equivalence checking [4], theorem proving [5], as well as static and dynamic analysis.

In modern software engineering [6], UML [7] has become the de-facto standard for modeling object-oriented systems. It is common that many research initiatives aim at fulfilling the gaps between UML-based modeling and secure software engineering. Specifically, secure software engineering requires dealing with many challenges depending on the development phase, where the security of a given software or system depends on a set of pre-defined security constraints and requirements. Thus, the major challenges in secure software development: specifying security policies, verifying whether a given software satisfies the pre-defined policies, choosing the adequate verification method, and evaluating the resilience of a given software to an attack.

In this survey, we study and compare the state-of-the-art related to the aforementioned challenges. We split our review into the following four directions:

1. Specification of security requirements: It covers techniques that express security policies of a model,
2. Modeling of attacks: It provides different modelling formalisms that are used to design the different type of attacks,
3. Verification of security requirements: It cites mainly the automatic approaches that check the security requirement and prove their correctness of a model-based system, and
4. Security quantification: It elaborates tools and frameworks that are dedicated to assess security and analyze risks in a system.

This paper is organized as follows. Section 2 briefly presents the main techniques used by the reviewed approaches in favor of highlighting their strengths and shortcomings. Section 3 is dedicated to review relevant works to the UML-driven specification of security requirements and attack scenarios modeling, UML-based development of secure systems, as well as qualitative and quantitative verification of security on UML designs. In addition, it discusses the various criteria that we developed in order to compare the state-of-the-art initiatives in the aforementioned research directions. Finally, Sect. 4 summarizes this review with some important remarks on the promising research direction in secure software engineering.

2 Background

In this section, we briefly explain the background needed to understand the existing research initiatives in security specification and verification fields. First, we introduce the main modeling diagrams that are used in the surveyed works.

2.1 UML diagrams

The UML diagrams [7,8] are of two classes: structural and behavioural. The former models the static parts of the system such as the class and the package diagrams. The behavioural diagrams can be either Interaction, State Machine, or Activity diagrams. The interaction diagram answers the question: "When does who call whom and how?", the state machine diagram answers the question: " How does an object respond to events in a specific state?", and the activity diagram answers the question: "What happens in which sequence?". The interaction diagram describes the communication between objects declared in the package diagrams. The main elements of an interaction diagram are: lifelines and messages. A lifeline represents a communication role and a message is a communication between two lifelines. A state machine contains a set of states related by transitions. A state is to model a situation where some invariant condition holds. It takes the evaluation of attributes of an object instantiated from a class. A transition is a directed relationship specifying the system changes between states. Activity diagram can be used to model system's behaviour at various level of abstractions. It allows low-level modeling compared with other behavioural diagrams. An activity diagram notation can be decomposed into two basic categories: activity nodes and activity edges. In addition to UML diagrams, other modeling dialects extend UML to support more specific domains. More precisely, SysML [9] extends system features to UML activity diagram by using four systems stereotypes (probability, rate, nobuffer, and overwrite). In UML standard, the stereotype is an extensibility mechanism to define or refine the meaning of a model element.

2.2 UML-driven secure systems development

UMLsec [10–12] is one of the first efforts toward extending UML into a security profile for the development of security-critical systems. It supports the specification of security-relevant information within UML diagrams using stereotypes, tagged values, and constraints mechanisms. A number of stereotypes are defined to express general security requirements including confidentiality, integrity, non-repudiation, role-based access control, fair exchange, authenticity, and freshness. These stereotypes can be used for various UML diagrams such as use case, class, statechart, activity, sequence, and deployment diagrams. The UMLsec framework [13–15] integrates an automated theorem prover (e-Setheo and SPASS) and a model checker (SPIN) [16] that automatically establish whether the security requirements hold or not. The framework input is a .zargo or .xmi file containing UML diagrams created with the UML tool named ArgoUML. SecureUML [17] is another paradigm that supports the development of secure software, also known as Model Driven Security (MDS). It integrates a pro-

posed secure modeling language for access control as a generalization of the role based access control (RBAC) within the UML class diagrams.

2.3 Security policies specification

A security policy is generally considered as a set of rules and guidelines that specify how to achieve the needed security requirements for a system or an organization. It might include rules for virus detection and prevention, granting and revoking access to system resources, protecting critical information from unauthorized users. Security policies can be classified into high-level policies and low-level requirements. High-level security covers policies such as, confidentiality, integrity, authentication, secrecy, freshness, authorization and availability. Low-level security requirements concern safety vulnerabilities that can be introduced in the software source code during the implementation phase including memory safety, input validation, user interface, etc. In the literature, usually a mathematical logic or a graph-based representation is used to formally specify/express security policies. As example of the formal logic, one can find the first and the higher order logics, the linear (LTL) [18] and the branching time logics (CTL, CTL*) [19]. Further, others extend them by probability and real time such as pCTL and CSL. The most important challenge in the security property specification is the user-friendliness of its expressiveness. Hence, other formalisms have been developed including W-automaton, OCL, and UML diagrams.

2.4 Security quantification

Security evaluation that involves only qualitative results may not necessarily coincide with real objectives. "At a high-level, metrics are quantifiable measurements of some aspect of a system. For a part of the system, which security is a meaningful concept, there are some identifiable attributes that collectively characterize the security of that entity" [20]. More precisely, a security metric is a quantitative measure indicating to which extend the considered entity possesses the attribute of being secure. Many criterions have been proposed to evaluate security risk including the Trusted Computer System Evaluation Criteria (TCSEC), Information Technology Security Evaluation Criteria (ITSEC), and Systems Security Engineering Capability Maturity Model (SSE-CMM). Recently, several commonly recognized metrics are used such as attack surface metrics [21]. In addition to the metric-based approaches, one can use the quantified version of formal verification techniques, namely probabilistic model-checking and probabilistic theorem proving to quantify the security of a system.

2.5 Attack scenario

As defined in [20], an attack is an attempt to gain unauthorized access to an Information System's (IS) services, resources, or information; or the attempt to compromise an IS's integrity, availability, or confidentiality. Different attack models have been deployed: attack tree, attack graph, and network attack graph. An attack tree [22] is a tree where

nodes represent attacks. The root node of the tree is the global goal of the attack. Children of a node are refinements of this goal, and leafs therefore represent attacks that can no longer be refined. A refinement can be done by either an aggregation or a choice. An attack graph [23] is a graph where each vertex represents the entire network state and the arcs represent state transitions caused by an attacker's actions. Moreover, a vertex can not represent the entire state of a system but rather a system condition in a predicate form and arcs represent the relation between the system conditions. In this case, the attack graph is called a dependency attack graph. A network attack [24] is an attack model [24] composed of the computer network, the attacker, and the defender. A state transition in a network attack model corresponds to a single action by the intruder, a defensive action by the system administrator, or a routine network action.

2.6 Formal verification

The strengths of formal verification approaches reside in the mathematical principles used to demonstrate the correctness of a formal software condition with respect to its specifications [25]. The basic idea is to construct a *formal model* of the system that captures all possible behaviours of the system and to write the related requirements representing the desirable behaviour. Various formal models have been proposed to capture the semantics of software such as automata [26], Petri nets [27], and process algebra [28]. Formal verification can be performed either using model checking or theorem proving.

Model Checking is a prominent automated formal verification technique for assessing functional and non-functional properties of information and communication systems. It requires a model of the system under consideration and a desired property to systematically check whether the given model satisfies the property or not. It checks the absence of errors (i.e., property violations) and alternatively can be considered as an intelligent and effective debugging technique. Among the most popular model checkers include: SPIN [16], NuSMV [29], UPPAAL [30], and PRISM [31]. For applying model checking to a system, three phases required:

- Specify the system using a model description language, and formalize the property to be checked using the property specification language.
- Check the validity of the property in the system model.
- Analyze the satisfiability or the violation of the property.

Theorem Proving is another technique used to formally verify a given system where both the system and its desired properties are expressed as formulas in mathematical logic. This logic is given by a formal system, which defines a set of axioms and a set of inference rules [32]. Theorem proving is the process of finding the proof of a property from the axioms of the system. The steps in the proof rely to axioms and rules, and possibly a derived definitions and intermediate lemmas. While proofs can be constructed by hand, we focus here only on machine-assisted theorem proving. Theorem provers are increasingly being used in the mechanical verification of systems and software

designs against safety-critical properties. Among existing automatic theorem provers, one includes Coq,[1] Isabelle,[2] and HOL.[3]

3 Related work

In this section, we review the existing works that have been conducted in the state-of-the-art of the specification, the verification, and the quantification of security at the design-level of software and systems. For security specification, we illustrate the existing formalisms that are used to model and to express security requirements in UML-based systems. For verification, three main approaches are adopted from the literature: model checking, theorem proving, and simulation. For security quantification, the existing approaches are based on the probabilistic verification, and stochastic metrics analysis. First, we study different techniques and approaches that model and specify security requirements in UML and SysML diagrams.

3.1 Specification of security requirements

Different mechanisms are proposed to specify security requirements such as temporal logic, OCL, and security templates (security patterns). In addition, other mechanisms integrates security services (confidentiality, availability, ...) within the design such as UMLsec by using profiles and stereotypes. In this section, we survey the works related to these both mechanisms. Mainly, we focus on formal languages and techniques that are used to express security properties. Also, we check how security is modeled in UML diagrams and which requirement is more suitable to specify and to check for a given approach.

Cheng et al. [33,34] propose a collection of security policies modeled as UML design patterns [35]. These patterns are shaped to include security aspects and constraints in order to facilitate the integration of security knowledge. The integrated security requirements support the security principles developed by Viega and McGraw [36] including authentication, confidentiality, integrity, availability that are designed as a state machine or a sequence diagram. The structure of the pattern is specified by using UML class diagram where the constraints are expressed in LTL. The application-dependent security properties are instantiated from the security templates and they are verified in Hydra framework [37]. The results of the analysis are processed and visualized on the original UML diagrams using the MINERVA tool [38]. These requirements are specified for the authentication of an automated teller machine (ATM) application modeled by a state machine diagram.

Zisman [39–41] proposes an extension to UMLsec in order to model peer-to-peer applications along with their security aspects and to statically verify security properties using a Static Verification Framework (SVF). It relies on the concept of abuse

[1] http://coq.inria.fr/.

[2] http://labsoc.comelec.enst.fr/turtle/.

[3] http://hol.sourceforge.net/.

cases defined as UML use cases and state machine diagrams to represent attack scenarios. Abuse cases serve both identification and description of security properties. The framework includes a property editor that allows the specification of security properties by using a graphical template language based on patterns defined in [42]. The design models that are expressed as class and state machine diagrams are translated into Promela and their related properties are mapped into LTL, for the purpose of using SPIN. Also, the verification of the underlying security protocols is performed by using the AVISPA[4] tool. And, the supported security properties are expressible in UMLsec and thus include but not limited to authentication, confidentiality, integrity, and role based access control. The results of the verification process are presented to the designer and in a case of property violation, the counterexample is highlighted in the design. In addition, the framework allows the user to add a custom code that is written in the action language. This is applied on a peer-to-peer protocol composed from one sender and one receiver without specifying its reference.

Jürjens and Shabalin [12,15] provide a support for an automated verification of UMLsec diagrams by using the SPIN model-checker. Also, an additional specific cryptography-related information are extracted from different types of diagrams. The Dolev–Yao model of an attacker is included with UMLsec to model the interaction with the outside environment [43]. In [44], Jürjens verifies UMLsec models for security requirements: authentication, confidentiality, integrity, availability, and secrecy. More precisely, the constraints associated with security stereotypes in UMLsec are verified using an automated theorem prover. To accomplish, UMLsec diagrams are translated into the first-order logic (FOL) formulas that can be automatically analyzed using the FOL prover on a sender/receiver communication protocol by including a Dolev–Yao attack model.

Hassine et al. [45] propose a high level pattern-based approach to describe a property by using the concept of Use Case Maps (UCM). The properties are described in terms of occurrence, ordering, and temporal scopes of actions with respect to their architectural scope. Then, they provide a mapping of the UCM patterns to the CTL, TCTL, or Architectural TCTL form (ArTCTL), which is an extension of TCTL. This technique is proposed to express all the design requirements including security specification and it is applied on an ATM UML diagrams. This properties pattern has been expressed for a simple telephone system root map.

Irbis et al. [46] present a tool for Data Property Specification (DaProS) that assists in the specification and the refinement of properties that can be used to check data quality in terms of accuracy. It uses Disciplined Natural Language (DNL) to specify properties within the help of common type representations as patterns to validate if the specifications capture the intended meaning of data or not. The data property specification is a four-step process: property category selection, property scope selection, property pattern selection and specification (timed and un-timed), and property visualization.

Heather et al. [47] generate linear temporal logic properties that specify the latent behaviour of an existing UML diagram. The key component of their approach is an

[4] http://www.avispa-project.org/.

evolutionary-computation called MARPLE. It uses a novel search to discover a set of properties that describe UML diagrams. As a result, the generated LTL properties describe the behaviour stated in the requirements and the unacceptable latent behaviours. The LTL properties are generated for a door-locking system and a robot navigation system.

The specification patterns created by France et al. in [48] are a meta-model-based patterns to characterize UML diagrams. First, they specialize the abstract syntax by sub-typing the UML meta-model classes and by making the well-formed rules more restrictive. The result is an abstract syntax for models describing pattern solutions. Then, they define a parameterized templates for OCL constraints [49] to represent requirements that are characterized by the specialized meta-model which capture the semantics of patterns. The result is a pattern specification consists of a structural pattern specification that specifies the class diagram view of the pattern solutions, and a set of interaction pattern specifications that specifies interactions in the pattern solution. The specification of the proxy design pattern is shown as an example of this proposal.

Another way is defined to specify temporal properties by using a modified version of OCL called temporal OCL (TOCL) developed by Ziemann and Gogolla in [50]. The pre and post conditions defined in OCL specification [49] are used to define the syntax and the semantics of invariants and operators of TOCL expressions. Flake and Müller express in [51] a property specification with OCL by mapping a specific hierarchically ordered pattern to CTL. The ordered pattern is specified by two classes: occurrence and order. The occurrence class contains the absence, the existence, and the universality operators. The order class specifies the precedence, and the response features. The specification by OCL is applied on an informal example composed of a class diagram and a state machine.

Van Lamsweerde [52] elaborates security requirements by constructing intentional anti-models. He addresses malicious obstacles (called anti-goals) set up by attackers to threaten security goals. In this case, threat trees are built through anti-goal refinement until leaf nodes are derived. The leaf nodes represent either software vulnerabilities or anti-requirements. Security requirements are elaborated by: (a) instantiate specification patterns associated with property classes, (b) derive anti-model specifications threatening such specifications, (c) derive alternative countermeasures to such threats and define new requirements by selection of alternatives that best meet other quality requirements from the model. The specification patterns can be formalized in a first-order, real-time linear temporal logic augmented with epistemic constructs for security-related predicates. Preda et al. [53] propose a formal technique that combines the use of access control policies expressed in the Organization-Based Access Control (OrBAC) [54] language together coupled with specifications based on the B-Method [55]. As application, the security requirements are expressed for the IPsec tunnels modeled by a sequence diagram.

Ouchani et al. [56] generate the PCTL temporal logic expressions from attacks templates designed as SysML activity diagrams. They model a selected set of CAPEC[5] as an application-independent attacks. Then, they use the system model to instantiate the

[5] http://capec.mitre.org, Common Attack Pattern Enumeration and Classification sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

appropriate dependent-application attack scenario in order for an attack-system composition. For verification, the interaction model within the generated PCTL properties from the application-dependent CAPEC attack are given to PRISM. This approach generates the PCTL properties related to the real time streaming protocol in the presence of a spoofing attack.

After showing the existing approaches that are dedicated to security specification and modeling, we study the modeling techniques that are used to design attacks of a given UML/SysML diagram.

3.2 Modeling of attacks

Modeling attacks is a main peace to analyze the weakness/the strength of a system. In this section, we survey the state-of-the-art that deals with attack modeling. In addition, we explore which UML diagram is suitable for this use, and we highlight which kind of attack modeling is preferred from those defined in Sect. 2.

In [57], attack graphs have been used to assess the probability that an attacker reaches particular attack step. Pamula et al. [58] analyze the security of system configurations in terms of the weakest adversary that can compromise the network. Frigault et al. [59] model probability metrics based on attack graphs as a special Bayesian Network. Each node of the network represents vulnerabilities as well as the pre and post conditions resulting from the exploitation of such vulnerabilities. Probabilities on nodes are inferred from the Common Vulnerability Scoring System (CVSS)[6] standard and denote the success likelihood of a specific attack goal.

Gegick and Williams [60] identify security vulnerabilities in code level by tailoring attack patterns based on software components. These patterns take the form of regular expressions that are generic representations of vulnerabilities. To encapsulate the steps that can form an attack, a matching of a sequence of components in a system design with symbols in a regular expression is applied to deduce the sequence of events in the attack pattern that can occur. If a match exists, then the vulnerability may exist in the application being analyzed. The attacks patterns given in [60] are mainly based on code-level vulnerabilities especially the buffer-over flow and the SQL injection attacks.

Grunske and Joyce [61] propose a risk-based approach that creates modular attack trees for each component in the system. These trees are specified as parametric constraints, which allow quantifying the probability of security breaches that occur due to internal and external vulnerabilities. The probability of a successful attack is determined with respect to a set of attack profiles that are chosen to represent potential attackers and corresponding to environmental conditions.

Saqui-Sannes et al. [62] verify the security of the communication systems designed in UML by using AVISPA[7] and TURTLE [8] tools. AVISPA uses the Dolev–Yao intruder model to detect security flaws. And, TTool checks the generated dynamic

[6] http://nvd.nist.gov/cvss.cfm.

[7] http://www.avispa-project.org.

[8] http://labsoc.comelec.enst.fr/turtle/.

timed automata from the model against temporal requirements. The implementation in AVSIMA and TTool models and checks Dolev–Yao intruder for a secure group communication protocol modelled in UML.

After surveying the main works related to the specification of security requirements and modeling attacks, we have to explore different techniques that are proposed to verify the specified requirements and to check the presence of possible attacks.

3.3 Verification of security requirements

The previous two sections are the main essence of the verification part, without security specification, the verification can not be completed. In this section, we focus more on the semantic model extracted for the used diagram. Also, we look forward for which technique is more appropriate to verify a given diagram.

Ray [63] presents a framework of security verification in software architecture based on a discrete time labeled transition system (DTLTS) as its formal representation. They propose an abstraction technique to extract only small units of the system, and put them in a "security harness" that exercises relevant executions within a unit, later, model-checking is applied to get more tractable units of three classes of security properties: safety, quasi-liveness, or bounded response. A given property is traversed while the irrelevant parts of the system are abstracted away by applying minimization or action hiding techniques that should satisfy a subset of temporal CCS and covers the abstracted part of the system. The CWB-NC[9] model-checker is used for property checking and attack tracing that is marked as suspicious behaviours is used subsequently for intrusion detection.

Thapa et al. [64] present an approach to verify security and time-related requirements. Both, UMLsec and MARTE profiles [65] are used to address security and timing requirements. This combined meta-model is converted to a form of UML-based Specification Environment (USE) specification so that it can be used for verifying models. Security properties such as authenticity, authorization, secrecy, integrity and fair exchange with freshness properties are supported. These properties are verified on an authentication protocol modeled by an interaction diagram.

Dong et al. [66] present a model-checking based approach to verify a composition of security patterns. Initially, they define the behavioural aspect of security patterns using sequence diagrams. A set of general rules are proposed to define the synchronous and asynchronous messages, and alternative flows within UML sequence diagrams. Then, these rules are used to transform sequence diagrams into a CCS-based [67] representation in order to be input to the CWB-NC model-checker. This requires from the user to specify the systems in CCS and the security properties in the GCT temporal logic. As a case study, the author chose the exchange fair protocol for there experiments.

Moebius et al. [68] present a verification method that allows to ensure the security for security-critical systems based on cryptographic protocols. An application is modeled with UML extended by a UML profile and the Model Extension Language

[9] http://www.scss.tcd.ie/Matthew.Hennessy/rsexternal/tools.php.

(MEL) is used to describe protocols on an implementation-independent level. A formal specification is automatically generated from the UML models and then imported by KIV[10] theorem prover. The formal specification of the system contains: a static and a dynamic part. In the first part, UML class and deployment diagrams are specified using algebraic specifications. In the dynamic part, sequence and activity diagrams are translated into an abstract state machine. However, the verification approach requires the interaction of the user for developing lemmas in in order to use the theorem prover. A copycard application is selected to verify security properties.

Bauer et al. [69] propose an approach for model-based security assurance that supports security verification at both levels: design and implementation. At the specification level, the design models are verified formally against high-level security requirements such as secrecy and authentication. At the implementation level, it relies on run-time verification technique to ensure if the implementation conforms to the properties expressing run-time behaviour. The uncovered security weakness during run-time verification is removed using aspect-oriented security hardening transformations. Thus, the approach supports the evolution of software since it updates the traceability mapping between the design specification and the implementation when refactoring operations. The design expressed in UMLsec are formally analyzed with the UMLsec environment. The online run-time verification technique is used for the verification of the code. The first phase follows seven steps: (1) specify and verify security protocols using UMLsec tool suite, (2) an implementation is linked to this UML model, (3) temporal logic formulae are derived from the UML model, (4) a security monitor is generated from the temporal logic formulae, (5) the relation between the UML model and the code is maintained as the implementation evolves over time, (6) errors in the implementation can be corrected using AOP, and (7) the security monitor is updated with respect to the changes arising from the previous step (step 6). The traceability phase has four steps: (1) verifying traceability from security requirements to design, (2) verifying traceability of security requirements from design to execution time, (3) verifying traceability of security requirements from one version of the implementation to another through system evolution, (4) hardening the traceable security hardening for code-level security vulnerabilities. In the experimental phase, the handshake protocol of SSL3 using RSA and a server authentication is selected to express and verify security requirements.

Rajkumar et al. [70] present an approach to verify the correctness of the usage control implementation using a semi-formal property verification. First, irrelevant code details of the usage control are abstracted and the usage control state space of the application is isolated. Next, The action LTL logic is used to verify the security properties on the abstracted model of the application generated by an action-based abstraction that takes the form of a UML-state machine.

A practical verification framework of a composition of UML behavioural diagrams (state machine, activity diagram, and sequence diagram) is proposed by Ouchani et al. [71]. Systematically, the semantic model, which is a kind of transition system is constructed based on a compositional operator. They verify the security properties by

[10] http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/.

instantiating them from an application-independent templates. The counterexample obtained from NuSMV is showed on the composition of diagrams that model ATM behaviour.

Lima et al. [72] verify UML 2.0 sequence diagrams by mapping each fragment into a process in PROMELA code where the send/receive event are specified by a communication between process. The security properties are specified using LTL temporal logic to be verified in SPIN. The counterexample is mapped to a trace in the model to be analyzed later by the user. Alalfi et al. [73] verify a reversed PHP code to a SecureUML model in the form of .XMI file. First, the resulting model is converted to a formal state model using UML2Alloy tool. Next, Alloy is employed to analyze the security properties.

The previous verification approaches check only the satisfiability or not of security requirements, and the presence or not of a given work. But our intention is to look forward techniques that quantify security by providing more precise answers as those presented in the next section.

3.4 Security quantification

We consider the initiatives cited in the previous section as a qualitative evaluation of security. In this section, we study how security is measured by using either probabilistic model checking, or statistical and qualitative analysis.

Umair et al. [74] use vulnerability occurrence to calculate the vulnerability index in software development life cycle) artifacts that provide an indication about the existing vulnerabilities. It is calculated by using the combined potential damage that can be caused by vulnerabilities. The approach is composed of the following steps: (1) identify a list of vulnerabilities and errors, (2) identify their occurrence relationships, (3) identify security requirements, (4) calculate the vulnerability index as the total number of vulnerability occurrences, (5) identify damage to assets and implementation cost of security requirements, (6) the total damage is the total potential damage caused by an occurrence of a given vulnerability, (7) prioritize the vulnerabilities based on their damages, and (8) the security index is measured based on the previous metrics to measure the extent of the damage that may be caused due to the remaining vulnerabilities.

Georg et al. [75] advocate the use of the aspect-oriented risk-driven development methodology for developing secure systems. It consists of a formal security evaluation and a trade-off analysis that help system designers to position alternative security solutions against each other. The first uses Alloy analyzer to provide assurance that an incorporated security mechanism performs as expected and makes the system resilient to previously identified attacks. The trade-off analysis based on Bayesian Belief Network (BBN) topology allows equally effective security mechanisms to be compared against system security requirements and other factors such as time-to-market and budget constraints.

Chen et al. [76] present threats modeling method based on attacking path analysis (T-MAP) which quantifies security threats by calculating the total severity weights of relevant attacking paths for commercial off the shelf (COTS) based systems. First,

key stakeholders and values propositions are identified. After that, a set of security evaluation criteria are established. Then, based on a database of vulnerabilities, the attack paths are analyzed. Finally, the attackers are modeled to assess the level of difficulty of the exploiting paths and the T-MAP weighting system is measured using the Tiramisu tool. They demonstrate the T-MAP process through a case study on a production server that communicates to several sensitive databases.

Liu and Traore [77,78] propose a framework to measure the protection of software and systems at the design level. Basically, they analyze the user interactions in a sequence diagram based on the User System Interaction Effect (USIE) model. In [77], they take into consideration only the confidentiality requirement by measuring the number of significant information leakage channels between two roles communication via messages. And in [78], they provide the measure of privilege system based on the mechanism strength in a pattern. Their framework is applied on a doctor read record protocol under a concurrent control modeled by a sequence diagram.

Buchholtz et al. [79] verify the security requirements by a qualitative analysis and then compute the performance measure by a performance analysis of UML sequence diagrams. The used technique for security analysis is a static analysis procedure. For quantitative evaluation, a Continuous-Time Markov Chain (CTMC) is generated from the model and then solved for its equilibrium probability distribution using procedures of numerical linear algebra such as the pre-conditioned the bi-conjugate gradient method, or successive over-relaxation. This framework has been applied on a shopping-online-system.

Ouchani et al. [80] quantify the security level of a system modeled as SysML activity diagrams. They model a selected set of CAPEC as an application-independent attacks. Then, they use the system model to instantiate the appropriate dependent-application attack scenario in order for an attack-system composition. And, they express in PCTL the security properties related to the system/attack composition. For verification, they use PRISM for the PCTL expressions and the model composition. The security properties are evaluated on a real-time steaming protocol model in SysML activity diagram.

After reviewing the main existing initiatives that specify, verify, and quantify security in model-based system. We summarize this survey by comparing them in order to conclude it with important new research directions.

3.5 Summary

We divide this section into two parts. The first one develops the challenging criteria come across within the state-of-the-art, and the second part compares the studied contributions with respect to the developed criteria.

3.5.1 Comparison criteria

In order to understand and to compare approaches targeting the security policies specification, or the security verification and quantification; we propose a set of comparison criteria. Further, we focus on whether if they are included or not for each approach.

- Criteria for security specification: This class is based on five basic features, which are:
  - Scope: It specifies whether the proposed approach is used for the specification of security requirements and properties or for the specification of security aspects in the software development.
  - Technique: It highlights the leveraged technique in security specification such as: profile, template, intermediate language, meta-model, temporal logic, OCL, formal model, etc.
  - Security properties: It pinpoints different security properties supported by the reviewed work including authentication, confidentiality, integrity, non-repudiation, role-based access control, fair exchange, authenticity, freshness, secure communication, guarded access, etc.
  - Tool support: It distinguishes the research initiatives that involved in the development and implementation of a tool supporting their proposed approach.
  - Verification technique: It relates the studied work on security specification to the initiatives aiming at supporting this effort with a verification technique that is also discussed within this survey.
- Criteria for security verification: This class contains six main features that are described as follows:
  - Verification technique: It specifies the technique used to perform the verification of security on UML design. Among the verification techniques, we can find model-checking, theorem proving, simulation, equivalence checking, type checking, hybrid (the use of more than one technique), empirical, etc.
  - Model representation: It represents the semantic model of the studied UML models (automata, formal language, Petri nets, logic, etc).
  - Tool support: It informs about the existence of a tool to perform the verification process.
  - Diagrams: It highlights the supported UML diagrams.
  - Type of verification: It distinguishes between approaches providing qualitative results and those providing quantitative results.
  - Counterexample: Tracing the counter example (C-E) in the system's model when the security property is violated.

3.5.2 Classification

We classify in this section the different works cited in this section. We compare them in Table 1 from the security specification point of view, and, in Table 2 we compare them in term of verification.

From Table 1, we observe that the most of the studied researchers express the security requirements and integrate them on the design (column 2, R/D). The most used technique for specification is security templates (column 3). Generally, they focus more on the existing security templates already mentioned in the background section. Some of them use profiles by using the stereotyped attributes to express security properties introduced by the user. Also, few of them use OCL that helps to write a security property as a constraint. The most used tool to check the security requirement is the model checkers rather then theorem provers (column 5), and, more especially

Table 1 Security specification in the state of the art

Contribution	Scope	Technique	Security properties	Verification technique	Tool support
Cheng et al. [33,34]	R/D	Templates	All	Model-checker	SPIN
Zisman [39–41]	R/D	Templates	All	Model-checker	SPIN + Avispa
Jürjens and Shabalin [12,15]	R/D	Profile	All	Model-checker	SPIN
Jürjens [44]	R/D	Profile	All	Theorem-prover	UMLsec suite
Hassine et al. [45]	R/D	Templates	General	Unspecified	Unspecified
Irbis et al. [46]	R/D	Templates	General	Unspecified	DaProS
Heather et al. [47]	R/D	UML model	General	Model-checker	LTL-support
France et al. [48]	R/D	Meta-model + OCL	All	Unspecified	Unspecified
Ziemann and Gogolla [50]	R/D	OCL	All	Unspecified	USE
Flake and Müller[51]	R/D	OCL pattern	All	Unspecified	Unspecified
Lamsweerde [52]	R/D	Templates	General	Prover + Checker	Unspecified
Preda et al. [53]	R/D	Profile	General	Prover	Unspecified
Ray [63]	R/D	Behavior	General	Checker	CWB-NC
Thapa et al. [64]	R/D	Behavior	All	Unspecified	USE
Dong et al. [66]	R/D	Templates	All	Checker	CWB-NC
Moebius et al. [68]	R/D	Profile	General	Prover	KIV
Bauer et al. [69]	R/D	Profile	General	Checker + Prover	UMLsec suite
Rajkumar et al. [70]	R/D	Language	Unspecified	Checker	Support LTL
Ouchani et al. [56,71,80]	R/D	Template	General	Checker	NuSMV/PRISM
Lima et al. [72]	R/D	Behaviour	General	Checker	Spin
Alalfi et al. [73]	R/D	Profile	General	Analyzer	Alloy
Georg et al. [75]	R/D	Aspect	General	Analyzer	Alloy
Chen et al. [76]	R/D	Attack	General	Quantification	Tiramisu

SPIN and PRISM (column 6). The motivation behind this is that the end user needs an automatic way to verify a design which is one of the favour of using the model checkers.

In Table 2, the most studied diagrams are behavioural and less the structural ones (column 2). The intuition behinds this is that they look on the execution part of the system. This intuition enforces the reason of proposing in the most case a state-based transition systems as semantic models (column 3). Since the most mentioned works do not cover quantitative features such as time, probability, and cost, the normal temporal logic such as LTL (column 4) is the most used to express security properties that could be checked within a model checker tool (column 5) such as SPIN (column 6). The

Table 2 The use of security verification approaches in the state of the art

Contribution	Behaviour diagram	Semantic model	Logic	Technique	Tool	C-E
Cheng et al. [33,34]	Class + Behaviour	Promela	LTL	Checker	SPIN	Yes
Zisman [39–41]	Class + State machine	Promela	LTL	Model-checker	SPIN + Avispa	Yes
Jürjens and Shabalin [12,15]	Calss + Behaviour	Promela	LTL	Model-checker	SPIN	Yes
Jürjens [44]	Class	Predicate	FOL	Theorem-prover	UMLsec suite	No
Siveroni et al. [39]	State machine	Promela + Predicates	LTL	Checker	SPIN	Yes
Buchholtz et al. [79]	Sequence diagram	CTMC	Unspecified	Unspecified	Static analysis	No
Saqui-Sannes et al. [62]	Communication system	Timed automata	Unspecified	Model-checker	AVISPA and Turtle	No
Ray [63]	Software architecture	DTLTS	GCT	Checker	CWB-NC	No
Thapa et al. [64]	Class	Unspecified	Unspecified	Unspecified	USE	No
Dong et al. [66]	Sequence diagram	CCS	GCT	Model-checker	CWB-NC	No
Moebius et al. [68]	UML and MEL	Algeb. + ASM	FOL	Prover	KIV	No
Bauer et al. [69]	UMLsec	Logic formula	FOL	Checker + Prover	UMLsec suite	Yes
Rajkumar et al. [70]	State machine	Unspecified	A-LTL	Model-checker	Unspecified	No
Ouchani et al. [71]	UML-beh. diagrams	LTS	CTL	Model-checker	NuSMV	Yes
Georg et al. [75]	Aspect model	Bayesian topology	Unspecified	Analyzer	Alloy	No
Lima et al. [72]	Sequence	Promela	LTL	Model-checker	SPIN	Yes
Alalfi et al. [73]	SecureUML	Alloy support	Alloy support	Analyzer	Alloy	Yes
Chen et al. [76]	Threat model	Unspecified	Unspecified	Quantification	Tiramisu	No
Liu and Traore [77,78]	Sequence	Unspecified	Unspecified	Quantification	USIE	No

main objective of this part of verification is to prove the correctness and the security of a design. In the case of a security property violation, we found that only 40 % (8 from 19 contributions) who provide the counter example (column 7) and show the vulnerabilities on the design.

After this summary and comparison, we present in the next section the deduced conclusions and the future directions that are inferred from our survey and comparison study.

4 Conclusions and future directions

From the studied contributions, we found that the most of them target explicitly a single diagram, and look toward a precise domain. Further, they deal especially with a specific technique instead of combining more than one in order to gain the advantage and to reduce the limitation of each one, such as the case of model checking and theorem proving. For verification, we observed that the most used technique is model checking compared to theorem proving or other approaches such as static analysis. Of course, the former is automatic but from a practical point of view, a theorem prover built up a model checker will reduce the verification time and size complexity. For security specification, the most used specification formalism is the temporal logic especially the linear one such as LTL. In this case, one cannot express a probabilistic or timed security property for example. Especially for a designer, it is suitable to specify a security property with the same design language. In addition, a new standardized OMG models such as SysML behavioural or UML timing diagrams are not studied yet for security verification and quantification. From this study, we believe that facilitating the expressiveness and the verification of security for the new standard diagrams is challenging. Another aspect is that the majority of the cited works inherit the existing limitations of the used tools. Especially, rarely whose trace the anomalies in the diagrams. Based on these conclusions, we propose four main promising research directions that are described as follows.

1. Developing techniques to automatically instantiate and express security requirements is a promising research direction. The objective is to help developers and security experts to avoid writing complex security properties in a mathematic formalism such as temporal logic. Especially, generating security properties from a rich design (time, probability, conditions, etc) to a specific temporal logic is not a straightforward task. It needs to develop the appropriate semantics of the design under test, and to develop techniques that convert this semantic (for example a PTA) to an equivalent temporal logic formula (such as extended PCTL) which is a challenging task. The correctness of any approach in this sense should be proved by finding an equivalent relation between both semantics of the design and the target temporal logic.
2. Improving the existing approaches and tools especially the model checkers that suffer from the state of explosion. In this direction, we propose to avoid the model checkers limitations (especially state explosion) by developing techniques such as abstraction and compositional verification for UML diagrams. The main issue in this direction is shaping the verification problem into sub-problems. In addition,

proving the soundness of both abstraction or compositional verification to prove that the satisfiability of the security properties is always preserved on both the concrete, and, the abstract or composed diagrams. In the case of abstraction and compositional verification, using theorem provers to prove the correctness of the the proposed algorithms, is also challenging, because it needs to provide a deep mathematical formalism for UML. Also, since the counter example is a main part of verification. Showing the counter example in the case of the probabilistic and timed systems is challenging.

3. Studying the security aspect for the recent standardized diagrams such as block, time, requirements, types, and parametric SysML diagrams. These diagrams are prominent in both software and hardware modeling. In addition, exploring security specification and verification in those diagrams is a new direction to be explored. To achieve this goal, first, the formal semantics of each diagram should be provided. Further, it should cover all artifacts and features of the studied diagram. Then, providing or adapting security specification and verification techniques for them.
4. Generating the secure code of a diagram. Providing the correct and the secured diagram is the goal of any modeling strategy. In this research path, the main intention is to generate automatically the secure code representing the low level description of the design level. The challenge of this problem is the complexity of the design and the specific domain of the target platform. Another challenge is proving formally the correctness of the generated secure code, by showing that the generated code is fully secure. In addition, the equivalence between both semantics of the design and the code which needs to provide the adequate semantics for both levels.
5. Hardening and securing a diagram. This direction is very prominent especially it advances many state-of-the-arts such as: security, software and hardware modeling, and formal verification. The main idea is to produce a fault tree instead of a simple counter example. To achieve that, the verification procedures should produce all the possible prone errors at one phase. Then, the correction of the design can be achieved by the aspect-modeling-language strategy and the help of the fault tree. The soundness of any hardening strategy can be proved by showing that the design after the correction is always more secure than before.

Those research directions are considered as hot research topics in security and formal verification in both software and hardware modeling for the next years.

References

1. Endler D, Collier M (2007) Hacking exposed VoIP: voice over IP security secrets & solutions. McGraw-Hill, New York
2. Baier C, Katoen JP (2008) Principles of model checking. The MIT Press, New York
3. Clarke Jr. EM, Grumberg O, Peled DA (1999) Model checking. The MIT Press, New York
4. Huang HSY, Cheng KTG (1998) Formal equivalence checking and design debugging. In: Frontiers in electronic testing, FRET 12. Kluwer Academic, New York
5. Newborn M (2001) Automated theorem proving—theory and practice. Springer, New York
6. Lange CFJ, Chaudron MRV (2005) Managing model quality in UML-based software development. In: Software technology and engineering practice, pp 7–16

7. OMG (2007) OMG unified modeling language (OMG UML) superstructure, V2.1.2. Object Management Group. OMG available specification, Needham
8. Holt J, Perry S (2007) SysML for systems engineering. Institution of Engineering and Technology Press, London
9. OMG (2014) OMG systems modeling language (OMG SysML) specification. Object Management Group, OMG available specification, Needham
10. Jürjens J (2005) Secure systems development with UML. Springer, New York
11. Jürjens J, Shabalin P (2004) Tools for critical systems development with UML (tool demo). In: Nunes NJ, Selic B, Rodrigues da Silva A, Toval Álvarez JA (eds) UML modeling languages and applications, UML 2004 satellite activities, Lisbon, 11–15 October 2004. Revised selected papers, vol 3297 of Lecture notes in computer science, pp 250–253. Springer, New York
12. Jürjens J, Shabalin P (2007) Tools for secure systems development with UML. Int J Softw Tools Technol Transf 9:527–544
13. Jürjens J, Shabalin P, Alter E, Gilg A, Höhn S, Kopjev D, Lehrhuber M, Schwarzmüller S, Shen S (2004) UMLsec tool. http://inky.cs.tu-dortmund.de/main2/jj/umlsectool/index.html. Accessed June 2011
14. Jürjens J, Schreck J, Yu Y (2008) Automated analysis of permission-based security using UMLsec. In: Proceedings of the theory and practice of software, the 11th international conference on fundamental approaches to software engineering, FASE'08/ETAPS'08, Heidelberg, pp 292–295. Springer-Verlag, Berlin
15. Jürjens J, Shabalin P (2004) Automated verification of UMLsec models for security requirements. In: UML 2004—the unified modeling language, vol 2460 of LNCS. Springer, New York, pp 412–425
16. SPIN Team (2011) SPIN. http://spinroot.com. Accessed June 2011
17. Basin D, Doser J, Lodderstedt T (2006) Model driven security: from UML models to access control infrastructures. ACM Trans Softw Eng Methodol 15:39–91
18. Vardi MY (1996) An automata-theoretic approach to linear temporal logic. In: Logics for concurrency: structure versus automata, vol 1043 of Lecture notes in computer science, pp 238–266. Springer-Verlag, New York
19. Pnueli A (1977) The temporal logic of programs. In: Proceedings of the 18th IEEE symposium on foundations of computer science, pp 46–57
20. Jansen W, Jansen W, Gallagher PD, Deputy Director (2009) Directions in security metrics research
21. Manadhata PK, Wing JM (2011) An attack surface metric. IEEE Trans Softw Eng 37(3):371–386
22. Mauw S, Oostdijk M (2005) Foundations of attack trees. In: International conference on information security and cryptology, ICISC 2005. LNCS, vol 3935, pp 186–198. Springer, New York
23. Sawilla R, Defence R&D Canada Ottawa (2007) Googling attack graphs. Defence R&D Canada, Ottawa (technical memorandum)
24. Sheyner OM (2004) Scenario graphs and attack graphs. PhD thesis, Pittsburgh (AAI3126929)
25. Drechsler R (2004) Advanced formal verification. Kluwer Academic Publishers, Norwell
26. Gabbar HA (2006) Modern formal methods and applications. Springer-Verlag, Secaucus
27. Murata T (1989) Petri nets: properties, analysis and applications. Proc IEEE 77(4):541–580
28. Bergstra JA (2001) Handbook of process algebra. Elsevier, New York
29. NuSMV Team (2011) NuSMV. http://nusmv.fbk.eu/. Accessed June 2011
30. UPPAAL Team (2011) UPPAAL. http://www.uppaal.org. Accessed June 2011
31. PRISM Team (2011) PRISM—probabilistic symbolic model checker. http://www.prismmodelchecker.org. Accessed June 2011
32. Jeannette EC, Clarke EM, Wing JM et al (1996) Formal methods: state of the art and future directions. ACM Comput Surv 28:626–643
33. Cheng BHC, Konrad S, Campbell LA, Wassermann R (2003) Using security patterns to model and analyze security. In: IEEE workshop on requirements for high assurance systems, pp 13–22
34. Wassermann R, Cheng BHC (2003) Security patterns. In: Technical report, Michigan State University, Computer Science and Engineering, East Lansing
35. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software
36. Viega J, Mcgraw G (2002) Building secure software: how to avoid security problems the right way
37. McUmber WE, Cheng BHC (2001) A general framework for formalizing UML with formal languages. In: Proceedings of the 23rd international conference on software engineering, ICSE '01, pp 433–442. IEEE Computer Society, Washington, DC

38. Campbell LA, Cheng BHC, Mcumber WE, Stirewalt K (2002) Automatically detecting and visualising errors in UML diagrams. Requir Eng 7:264–287
39. Siveroni I, Zisman A, Spanoudakis G (2008) Property specification and static verification of UML models. In: Proceedings of the 2008 third international conference on availability, pp 96–103. IEEE Computer Society, Reliability and Security, Washington, DC
40. Siveroni I, Zisman A, Spanoudakis G (2010) A UML-based static verification framework for security. Requir Eng 15:95–118
41. Zisman A (2007) A Static verification framework for secure peer-to-peer applications. In: Proceedings of the 2nd international conference on internet and web applications and services, ICIW '07, p 8. IEEE Computer Society, Washington, DC
42. Dwyer MB, Avrunin GS, Corbett JC (1999) Patterns in property specifications for finite-state verification. In: Proceedings of the 21st international conference on software engineering, ICSE '99, pp 411–420. ACM, New York
43. Houmb SH, Islam S, Knauss E, Jürjens J, Schneider K (2010) Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and UMLsec. Requir Eng 15:63–93
44. Jürjens J (2005) Sound methods and effective tools for model-based security engineering with UML. In: Proceedings of the 27th international conference on software engineering, ICSE '05, pp 322–331. ACM, New York
45. Hassine J, Rilling J, Dssouli R (2009) Use case maps as a property specification language. Softw Syst Model 8:205–220. doi:10.1007/s10270-007-0076-6
46. Gallegos I, Gates A, Tweedie C (2010) DaProS: a data property specification tool to capture scientific sensor data properties. In: Trujillo J, Dobbie G, Kangassalo H, Hartmann S, Kirchberg M, Rossi M, Reinhartz-Berger I, Zimányi E, Frasincar F (eds) Advances in conceptual modeling—applications and challenges, vol 6413 of Lecture notes in computer science, Springer, Berlin, pp 232–241. doi:10.1007/978-3-642-16385-2-29
47. Goldsby HJ, Cheng BHC (2010) Automatically discovering properties that specify the latent behavior of UML models. In: Proceedings of the 13th international conference on model driven engineering languages and systems: part I, MODELS'10, Heidelberg, pp 316–330. Springer-Verlag, Berlin
48. France RB, Kim D-K, Ghosh Sudipto, Song E (2004) A UML-based pattern specification technique. IEEE Trans Softw Eng 30(3):193–206
49. OMG (2006) Object constraint language, V2.0. OMG available specification. Object Management Group
50. Ziemann P, Gogolla M (2002) An extension of OCL with temporal logic. In: Critical systems development with UML, pp 53–62
51. Flake S, Müller W (2003) Expressing property specification patterns with OCL. In: Software engineering research and practice, pp 595–603
52. van Lamsweerde A (2004) Elaborating security requirements by construction of intentional anti-models. In: Proceedings of the 26th international conference on software engineering, ICSE 2004, pp 148–157
53. Preda S, Cuppens-Boulahia N, Cuppens F, Garcia-Alfaro J, Toutain L (2010) Model-driven security policy deployment: property oriented approach. In: Massacci F, Wallach D, Zannone N (eds) Engineering secure software and systems, vol 5965 of Lecture notes in computer science, pp 123–139. Springer, Berlin. doi:10.1007/978-3-642-11747-3-10
54. Kalam AAE, Baida RE, Balbiani P, Benferhat S, Cuppens F, Deswarte Y, Miege A, Saurel C, Trouessin G (2003) Organization based access control. In: Proceedings IEEE 4th international workshop on policies for distributed systems and networks, POLICY 2003, pp 120–131
55. Abrial J-R (1996) The B-book: assigning programs to meanings. Cambridge University Press, New York
56. Ouchani S, Mohamed OA, Debbabi M (2013) A security risk assessment framework for SysML activity diagrams. In: 2013 IEEE 7th international conference on software security and reliability (SERE), pp 227–236
57. Jha S, Sheyner O, Wing J (2002) Two formal analyses of attack graphs. In: Proceedings of the 15th computer security foundation workshop, pp 49–63. IEEE, London
58. Pamula J, Jajodia S, Ammann P, Swarup V (2006) A Weakest–Adversary security metric for network configuration security analysis. In: Proceedings of the 2nd ACM workshop on quality of protection, QoP'06, pp 31–38, New York

59. Frigault M, Wang L (2008) Measuring network security using Bayesian network-based attack graphs. In: 32nd annual IEEE international conference on computer software and applications, COMPSAC '08, pp 698–703
60. Gegick M, Williams L (2007) On the design of more secure software-intensive systems by use of attack patterns. Inf Softw Technol 49:381–397
61. Grunske L, Joyce D (2008) Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles. J Syst Softw 81:1327–1345
62. Saqui-Sannes P, Villemur T, Fontan B, Mota S, Bouassida MS, Chridi N, Chrisment I, Vigneron L (2010) Formal verification of secure group communication protocols modelled in UML. Innov Syst Softw Eng 6:125–133
63. Ray A (2003) Security check: a formal yet practical framework for secure software architecture. In: Proceedings of the 2003 workshop on new security paradigms, NSPW '03, pp 59–65. ACM, New York
64. Thapa V, Song E, Kim H (2010) An approach to verifying security and timing properties in UML models. In: 15th IEEE international conference on engineering of complex computer systems (ICECCS), pp 193–202
65. OMG (2008) A UML profile for MARTE: modeling and analysis of real-time embedded systems, beta 2 (convenience document without change bars). Object Management Group, OMG adopted specification, Needham
66. Dong J, Peng T, Zhao Y (2010) Automated verification of security pattern compositions. Inf Softw Technol 52:274–295
67. Milner R (1982) A calculus of communicating systems. Springer-Verlag, Secaucus
68. Moebius N, Stenzel K, Reif W (2010) Formal verification of application-specific security properties in a model-driven approach. In Massacci F, Wallach D, Zannone N (eds) Engineering secure software and systems, vol 5965 of Lecture notes in computer science, pp 166–181. Springer, Berlin. doi:10.1007/978-3-642-11747-3-13
69. Bauer A, Jürjens J, Yu Y (2011) Run-time security traceability for evolving systems. Comput J 54:58–87
70. Ghosh SK, Rajkumar PV, Dasgupta P (2009) Application specific usage control implementation verification. Int J Netw Secur Appl (IJNSA) 01(03)
71. Ouchani S, Mohamed OA, Debbabi M, Pourzandi M (2010) Verification of the correctness in composed UML behavioural diagrams. In: SERA (selected papers), pp 163–177
72. Lima V, Talhi C, Mouheb D, Debbabi M, Wang L, Pourzandi M (2009) Formal verification and validation of UML 2.0 sequence diagrams using source and destination of messages. Electron Notes Theor Comput Sci 254:143–160
73. Alalfi MH, Cordy JR, Dean TR (2009) A verification framework for access control in dynamic web applications. In: Canadian conference on computer science and software engineering, pp 109–113
74. Ahmed Khan MU, Zulkernine M (2008) Quantifying security in secure software development phases. In: Proceedings of the 2008 32nd annual IEEE international computer software and applications conference, pp 955–960. IEEE Computer Society, Washington, DC
75. Georg G, Anastasakis K, Bordbar B, Houmb SH, Ray I, Toahchoodee M (2010) Verification and trade-off analysis of security properties in UML system models. IEEE Trans Softw Eng 36:338–356
76. Chen Y, Boehm B, Sheppard L (2003) Measuring security investment benefit for off the shelf software systems—a stakeholder value driven approach
77. Liu MY, Traore I (2004) UML-based security measures of software products. In: International workshop on methodologies for pervasive and embedded software (MOMPES'04), 4th international conference on application of concurrency to system design (ACSD-04), Hamilton
78. Liu MY, Traore I (2005) Measurement framework for software privilege protection based on user interaction analysis. In: Proceedings of the 11th IEEE international software metrics symposium, p 10. IEEE Computer Society, Washington, DC
79. Buchholtz M, Gilmore S, Haenel V, Montangero C (2005) Endto-end integrated security and performance analysis on the DEGAS choreographer platform. In: Proceedings of the international symposium of formal methods Europe (FM 2005), vol 3582 in LNCS. Springer-Verlag, New York, pp 286–301
80. Ouchani S, Jarraya Y, Aït-Mohamed O (2011) Model-based systems security quantification. In: PST, pp 142–149