SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2016; **26**:119–148
Published online 24 July 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.1580

Model-based security testing: a taxonomy and systematic classification

Michael Felderer[1,*,†], Philipp Zech[1], Ruth Breu[1], Matthias Büchler[2] and Alexander Pretschner[2]

[1]*Institute of Computer Science, University of Innsbruck, Technikerstr. 21a, Innsbruck, Austria*
[2]*Department of Software Engineering, TU München, Boltzmannstr. 3, Garching, Germany*

SUMMARY

Model-based security testing relies on models to test whether a software system meets its security requirements. It is an active research field of high relevance for industrial applications, with many approaches and notable results published in recent years. This article provides a taxonomy for model-based security testing approaches. It comprises filter criteria (i.e. model of system security, security model of the environment and explicit test selection criteria) as well as evidence criteria (i.e. maturity of evaluated system, evidence measures and evidence level). The taxonomy is based on a comprehensive analysis of existing classification schemes for model-based testing and security testing. To demonstrate its adequacy, 119 publications on model-based security testing are systematically extracted from the five most relevant digital libraries by three researchers and classified according to the defined filter and evidence criteria. On the basis of the classified publications, the article provides an overview of the state of the art in model-based security testing and discusses promising research directions with regard to security properties, coverage criteria and the feasibility and return on investment of model-based security testing. Copyright © 2015 John Wiley & Sons, Ltd.

Received 3 September 2014; Revised 10 June 2015; Accepted 10 June 2015

KEY WORDS: model-based security testing; security testing; model-based testing; classification; taxonomy

1. INTRODUCTION

Modern information technology systems based on concepts such as cloud computing, location-based services or social networking are permanently connected to other systems and handle sensitive data. These interconnected systems are subject to security attacks that may result in security incidents with high severity affecting the technical infrastructure or its environment. Exploited security vulnerabilities can cause drastic costs, for example, due to downtimes or the modification of data. A high proportion of all software security incidents is caused by attackers who exploit known vulnerabilities [1]. An important, effective and widely applied measure to improve the security of software is security testing techniques, which identify vulnerabilities and ensure security functionality.

In this article, testing refers to active, dynamic testing, where the behaviour of a system under test (SUT) is checked by applying intrusive tests that stimulate the system and observe and evaluate the system reactions [1]. Security testing validates software system requirements related to security properties such as confidentiality, integrity, availability, authentication, authorization and non-repudiation [2, 3]. Sometimes, security properties come as classical functional requirements, for example, 'user accounts are disabled after three unsuccessful login attempts', which approximates one part of an authorization property and is aligned with the software quality standard ISO/IEC

*Correspondence to: Michael Felderer, Institute of Computer Science, University of Innsbruck, Technikerstr. 21a, 6020 Innsbruck, Austria.
†E-mail: michael.felderer@uibk.ac.at

9126 [4] defining security as a functional quality characteristic. However, it seems desirable that security testing directly targets the previous security properties, as opposed to taking the detour of functional tests of security mechanisms. This view is supported by the ISO/IEC 25010 [2] standard that revises ISO/IEC 9126 and introduces security as a new quality characteristic that is not included in the characteristic functionality any more.

Because the former kind of (non-functional) security properties describes all executions of a system, this kind of security testing is intrinsically hard. Because testing cannot show the absence of faults, an immediately useful perspective directly considers the *violation* of these properties. This has resulted in the development of specific testing techniques such as penetration testing that simulates attacks to exploit vulnerabilities. Penetration tests are difficult to craft because tests often do not directly cause observable security exploits, and because the testers must think like an attacker [5], which requires specific expertise. During penetration testing, testers build a mental model of security properties, security mechanisms and possible attacks against the system and its environment. It seems intuitive that security testing can benefit from specifying these security test models in an explicit and processable way. Security test models provide guidance for the systematic and effective specification and documentation of security test objectives and security test cases, as well as for their automated generation and evaluation.

The variant of testing that relies on explicit models that encode information on the system under test and/or its environment is called model-based testing (MBT) [6, 7]. Especially in the context of security testing, the process of deriving tests tends to be unstructured, not reproducible, not documented, lacking detailed rationales for the test design and dependent on the ingenuity of single testers or hackers, motivating the use of explicit models.

Model-based security testing (MBST) is model-based testing of security requirements. MBST is a relatively new research field and of high interest for industrial applications [1], where many approaches and remarkable results were published in recent years. To sketch which parts of the research area seem to be well understood and evaluated, and which ones do not seem so yet, a systematic classification of existing work on MBST seems valuable, in order to guide further research and to foster industrial application.

Existing taxonomies for model-based testing [8] focus on functional testing and are not sufficient to cover all specific aspects of model-based security testing. The goal of this article is to define classification criteria for MBST and to provide a comprehensive overview of the state of the art in model-based security testing by assigning existing work to these criteria in a systematic way. The defined classification criteria for MBST complement existing classification schemes for MBT by security-specific aspects.

Contribution. This article provides specific classification criteria for model-based security testing approaches, that is, filter and evidence criteria. The schema extends established existing taxonomies and classifications of model-based testing [6, 8] but additionally considers specifics of security testing and the evidence of testing approaches. The defined criteria are evaluated by systematically classifying existing model-based security testing approaches reported in the literature according to them. For this purpose, systematic search and selection of model-based security testing publications were performed and classified the identified 119 papers. For each paper, it was possible to classify the covered MBST approach uniquely according to the defined filter and evidence criteria. This indicates the adequacy of the classification scheme itself and provides a characterization of the state of the art of model-based security testing.

Intended Limitations. In this article, only model-based testing approaches with explicit models are covered, but not classical fuzz or penetration testing without explicit and machine-processable models, static analysis (which only considers the validation or verification of the model itself) or monitoring (which only passively observes a running system). In addition, the article only considers model-based testing of security properties but not related properties such as robustness, safety or trust.

Structure. Section 2 presents background and related work on model-based testing, security testing and their classification. Section 3 defines filter and evidence criteria for model-based security testing approaches. Then, in Section 4, relevant publications are systematically selected and

DOI: 10.1002/stvr

classified according to the criteria defined before. Section 5 presents and discusses the results, and Section 6 draws conclusions.

2. BACKGROUND AND RELATED WORK

Testing is the evaluation of software by observing its execution [9]. The executed system is called *system under test* (SUT). Testing consists of the *dynamic* verification of the actual behaviour of a system against its *expected* behaviour [10]. This is performed with a *finite* set of *finite* test cases, a so-called *test suite*, suitably *selected* from the usually infinite set of intended execution traces. As mentioned in the introduction, this article only considers dynamic testing where the behaviour of a SUT is checked by applying intrusive tests that stimulate the system and observe and evaluate the system reactions. After running a test case, the actual and intended behaviours of an SUT are compared with each other, which then results in a *verdict*. Verdicts can be *pass* (behaviours conform), *fail* (behaviours do not conform) or *inconclusive* (not known whether behaviours conform) [11]. A *test oracle* is a mechanism for determining the verdict.

According to the granularity of the SUT, one can distinguish between *unit testing*, that is, testing single programme units; *integration testing*, that is, ensuring that several units work together correctly; and *system testing*, that is, testing the system as a whole. *Regression testing* performs selective retesting to verify that modifications have not caused unintended effects and that the system under test still complies with the specified requirements [12]. Regression testing can be performed for any type of test independent of the granularity of the SUT. According to accessibility of test design artefacts one can distinguish *black box testing*, that is, deriving test cases from external descriptions of software such as specifications, from *white box testing*, that is, deriving test cases from internal descriptions such as implementation code.

The remainder of this section presents background and related work on MBT and its classification (Section 2.1), on security testing and its classification (Section 2.2) and on existing classifications of MBST (Section 2.3).

2.1. Model-based testing

This section presents basic concepts and classifications of model-based testing.

2.1.1. Basic concepts of model-based testing. In MBT, manually selected algorithms automatically and systematically generate test cases from a set of models of the system under test or its environment [7]. MBT is an active area of research [6, 8] and offers big potential to improve test processes in industry [7, 13, 14]. Its prospective benefits include early and explicit specification and review of system behaviour, better test case documentation, the ability to automatically generate useful (regression) tests and control test coverage, improved maintenance of test cases and shorter schedules and lower costs [7].

The process of MBT consists of three main steps integrated into the overall process of test design, execution and evaluation. (1) A model of the SUT and/or its environment is built from informal requirements, existing specification documents or an SUT. The resulting model of the SUT dedicated to test generation is often called *test model*. (2) If they are executable, one execution trace of such a model acts as a test case: input and expected output for an SUT. Because there are usually infinitely many and infinitely long execution traces, models can therefore be used to generate an infinite number of tests. To cut down their number and length, *test selection criteria* are applied. These guide the generation of tests. (3) Once the test model and the test selection criteria are defined, a set of test cases is generated from the test model as determined by the chosen test selection criteria. Test generation is typically performed automatically. The generated test cases are traces of the model and thus possibly at a higher level than the events or actions of an SUT. If necessary, the generated test cases are further refined to a more concrete level or adapted to the SUT to support their automated execution. In accordance with this process, this article does not consider static model analysis, for example, consistency checking of test models, as an MBT approach. Instead, the focus is on approaches where at least abstract test cases are generated.

DOI: 10.1002/stvr

2.1.2. Classification of Model-Based Testing. Utting *et al.* [8] present a broad taxonomy for MBT approaches motivated by the model-based testing process. In this taxonomy, three general classes are identified: *model specification*, *test generation* and *test execution*. Each of the classes is divided into further categories. The model specification class consists of *scope*, *characteristics* and *paradigm* categories. The test generation class consists of *test selection criteria* and *technology* categories. Finally, the test execution class contains the category *on/offline*. Figure 1 shows an overview of the taxonomy of Utting *et al.* Zander *et al.* [15] complement the taxonomy with *test evaluation* as an additional class referring to comparing the actual SUT outputs with the expected SUT behaviour based on a test oracle. The test evaluation class consists of *specification* and *technology* categories. Furthermore, the test generation class is extended with an additional category *result of the test generation*. Also, the class model specification is adapted and considers a category *MBT basis* indicating what specific element of the software engineering process is the basis for the MBT process.

Dias-Neto and Travassos [6] performed a systematic review and classification of the MBT research literature. They consider the following dimensions: *type of experimental evidence* ranging from speculation to experimentation; *testing level* with values unit, integration, system and regression testing; *use of supporting tool*; *model to represent software* divided in UML-based and non-UML model types; and non-orthogonal *software execution platforms* such as embedded systems, distributed systems, Web services or Web application. In addition, Dias-Neto and Travassos consider the *type of non-functional requirements* evaluated by MBT including security, reliability, efficiency and usability. The review and the classification of Dias-Neto and Travassos complement the taxonomy of Utting *et al.* because it provides a very detailed view of MBT, while the taxonomy gives a higher-level way of classifying both existing and future MBT approaches. For instance, the review lists 48 different MBT modelling notations, while the taxonomy of Utting *et al.* groups these into seven modelling paradigms [8]. The main aim of the review and MBT classification of Dias-Neto and Travassos is to support the selection of MBT techniques for software projects. It is therefore very pragmatic and covers purely technological criteria such as the use of supporting tools or the software execution platforms as well.

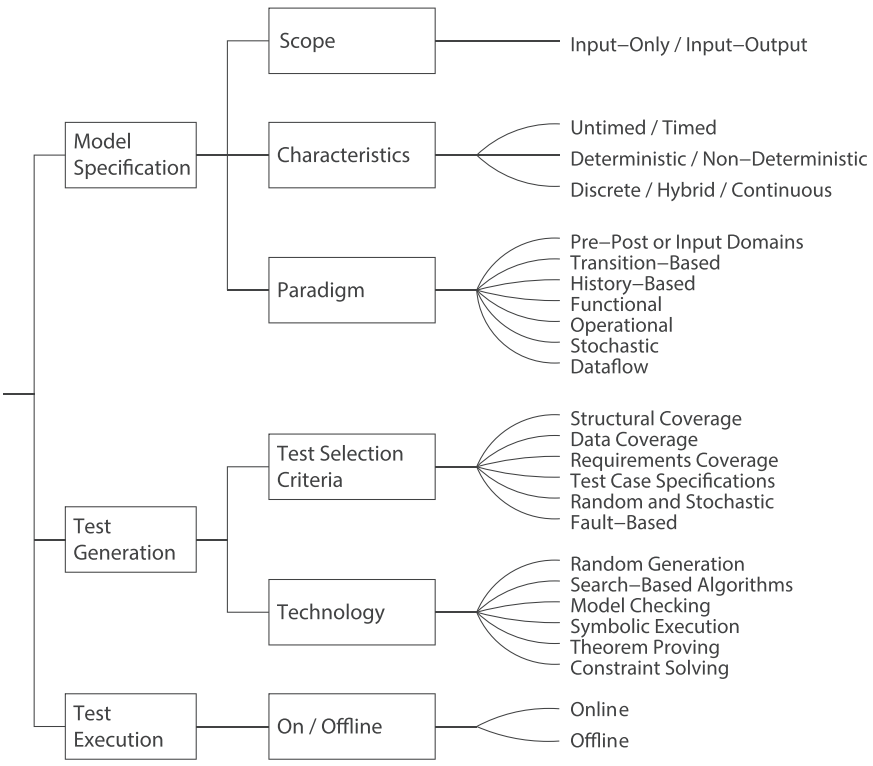

Figure 1. Overview of the model-based testing taxonomy of Utting *et al.* [8].

DOI: 10.1002/stvr

Anand *et al.* [16] perform an orchestrated survey of methodologies for automated software test case generation. They distinguish three types of *MBT approaches*, that is, *axiomatic*, *finite state machine* and *labelled transition system* approaches, as well as three types of used *modelling notations*, that is, *scenario-oriented*, *state-oriented* and *process-oriented* notations. This cited paper does not provide additional categories of MBT approaches compared with the taxonomy of Utting *et al.* Its types of MBT approaches and modelling notations can be subsumed under the categories 'paradigm' and 'test selection criteria' proposed by Utting *et al.*

Hierons *et al.* [17] provide a survey on using formal specifications to support testing. They present popular types of specification languages, that is, *state-based*, *hybrid* and *algebraic* languages, and discuss how their use can be integrated into the testing process by concrete MBT approaches of each language type. This extensive survey does not propose additional categories of MBT approaches; the presented types of specification languages can be subsumed under the model specification class of Utting *et al.*

Hartman *et al.* [18] provide several decision criteria, corresponding to classification criteria, to choose a language for test behaviour and test selection modelling. Their decision criteria, *visual versus textual language*, *proprietary versus standard language* and *commercial versus open-source tool* are technological and economic criteria. In addition, Hartman *et al.* distinguish *model-driven versus model-based testing*, that is, UML-based or not, *online versus offline testing*, *system versus test-specific language* and *domain-specific versus generic language*. Except for the last criterion, *domain-specificity*, the other non-technological criteria are covered by the classification criteria of Utting *et al.*

Summarizing the discussion of the available MBT classification schemes, the taxonomy of Utting *et al.*, shown in Figure 1 and motivated by the model-based testing process with or without its extension by Zander *et al.*, seems well suited to classify MBT approaches on a technology-independent level and to identify trends on MBT approaches. This is due to several reasons. First, the classification of Utting *et al.* provides high-level criteria not considering very fine-grained gradings like the modelling notation in the classification of Dias-Neto and Travassos. Second, the classification is abstract and does not consider technological details like the tool criteria of Dias-Neto and Travassos as well as Hartman *et al.* Third, its classification criteria subsume the abstract criteria defined in the other classifications.

2.2. Security testing

This section presents basic concepts and classifications of security testing.

2.2.1. Basic concepts of security testing. Security testing validates software system requirements related to security properties such as confidentiality, integrity, availability, authentication, authorization and non-repudiation. Security testing identifies whether the specified or intended security features are implemented correctly. According to Tian-yang *et al.* [19] two principal approaches can be distinguished, that is, *security functional testing* and *security vulnerability testing*. Security functional testing validates whether the specified security requirements are implemented correctly, in terms of both security properties and security mechanisms. Security vulnerability testing addresses the identification of unintended system vulnerabilities.

A *vulnerability* is a characteristic of a system without which a specified class of security breaches would not be possible. Examples include buffer overflow or Structured Query Language Injection (SQLI) vulnerabilities. The characteristic refers to flaws in system design, implementation, operation or configuration. A *threat* is a potential cause of an unwanted incident that harms or reduces the value of an asset. For instance, a threat may potentially corrupt the integrity or availability of a Web service. An *attack* is defined by the steps a malicious or inadvertently incorrectly behaving entity performs to the end of turning a threat into an actual corruption of an asset's properties. This is usually performed by exploiting a vulnerability.

Security vulnerability testing uses the simulation of attacks and other kinds of *penetration testing* attempting to compromise the security of a system by playing the role of a hacker trying to

DOI: 10.1002/stvr

attack the system and exploit its vulnerabilities [20]. Security vulnerability testing requires specific expertise, which makes it difficult and hard to automate [21]. By identifying risks in the system and creating tests driven by those risks, security vulnerability testing can focus on parts of a system implementation in which an attack is likely to succeed.

Risks are often used as a guiding factor to define security test processes. For instance, Potter and McGraw [21] consider the process steps creating security misuse cases, listing normative security requirements, performing architectural risk analysis, building risk-based security test plans, wielding static analysis tools, performing security tests, performing penetration testing in the final environment and cleaning up after security breaches. Also, the Open Source Security Testing Methodology Manual [22] and the Open Web Application Security Project (OWASP) Testing Guide [23] take risks into account for their proposed security testing activities.

2.2.2. Classification of security testing. It was sketched before that two principal security testing approaches can be distinguished, that is, *security functional testing* and *security vulnerability testing* [19]. Potter and McGraw [21] operationalize these principal approaches and distinguish *testing security mechanisms* to ensure that their functionality is properly implemented, and *performing risk-based security testing* motivated by understanding and simulating the attacker's approach. Testing security mechanisms can be performed by standard test organizations with classical functional test techniques, whereas risk-based security testing requires specific expertise and sophisticated analysis [21].

Besides the two principal approaches, Tian-yang *et al.* [19] list major methods of security testing. They consider *formal security testing*, *model-based security testing*, *fault-injection-based security testing*, *fuzz testing*, *vulnerability scanning testing*, *property-based testing*, *white box-based security testing*, and *risk-based security testing*. The provided categories are rather ad-hoc; that is, they are not based on classification criteria; they concentrate on a few security testing approaches and are not disjoint. For instance, model-based security testing overlaps with formal security testing, fuzz testing, white box-based security testing and risk-based testing.

Shahriar and Zulkernine [24] propose seven comparison criteria for security vulnerability testing approaches, that is, *vulnerability coverage*, *source of test cases*, *test case generation method*, *testing level*, *test case granularity*, *tool automation* and *target applications*. Tool automation is further refined into the criteria *test case generation*, *oracle generation* and *test case execution*. The authors classify 20 informally collected approaches according to these criteria. The main aim of the criteria is support for security practitioners to select an appropriate approach for their needs. Therefore, Shahriar and Zulkernine blend abstract criteria such as source of test cases or test case generation method with technological criteria such as tool automation or target applications.

Bau *et al.* [25] define the security vulnerability testing criteria *class of vulnerabilities tested*, *effectiveness against target vulnerabilities* and *relevance of the target vulnerabilities* to vulnerabilities in production systems. The authors compare eight security vulnerability testing approaches based on these criteria. The aim of the criteria is to provide evidence to support the selection of a vulnerability testing approach as well as to assess the potential impact of future research.

Felderer and Fourneret [26] present classification criteria for security regression testing. They consider the seven criteria *abstraction level*, *security issue*, *regression testing technique*, *tool support*, *evaluated system*, *maturity of evaluated system* and *evaluation measures*. The authors classify 17 systematically collected security regression testing approaches according to these criteria. The first four criteria determine the security regression testing approach, whereas the last three criteria determine its evaluation. Therefore, the aim of the criteria is to characterize security regression testing approaches and their evidence to support researchers and practitioners in the field.

From the presented classifications of security testing, it can be concluded that security vulnerability testing requires more specific testing techniques and classifications than security functional testing, which is more or less classical testing of security requirements such as confidentiality, integrity or availability. In security vulnerability testing, attacks that exploit vulnerabilities have to be defined and evaluated. This requires specific testing techniques to be selected and configured on the basis of classification criteria. The presented classification schemes for security vulnerability

DOI: 10.1002/stvr

testing suggest that risk-based testing, fault-injection, fuzz testing, vulnerability coverage and evidence of the approaches, for example, measured by their effectiveness and the evaluated system, are relevant aspects to be considered in a security testing classification.

2.3. Existing classifications of model-based security testing

Model-based security testing is a relatively new research field, where many approaches were published in recent years. Although a systematic classification of existing work on MBST to guide further research and to foster industrial application does not exist and to provide one is the aim of this article, there are already some ad-hoc classifications of existing approaches to MBST.

The ad-hoc classification presented by Felderer *et al.* [27] raises the security test classification of Potter and McGraw [21], who distinguish functional and risk-based security testing, to the model-level and defines the dimensions *risk* and *automated test generation*. The risk dimension addresses whether risks are *integrated* or *not integrated* into the test models. The automated test generation dimension addresses the degree of automation of test generation and can be *complete*, that is, all security tests are derived from a model. It can also be *partial*, that is, some security tests are generated from a model, or *missing*, that is, security tests are not derived from a model. As this classification also considers the case that model-based test generation is missing, it is not specific to MBST but in principle a classification of security testing in general with the aim to extend the classification of functional and risk-based security testing to the model-level. In addition, this classification focuses only on one aspect of model-based security testing, namely the risk integration, but misses other aspects such as mutation. As the risk integration is covered by the filter criterion 'explicit test selection' (Section 3.1) and because it is a classification of security testing in general, it is not considered further here.

Another MBST classification that considers different perspectives used in securing a system is defined by Schieferdecker *et al.* [1]. The authors claim that MBST needs to be based on different types of models and distinguish three types of input models for security test generation, that is, *architectural and functional models*, *threat, fault and risk models* and *weakness and vulnerability models*.

Architectural and functional models of the SUT are concerned with security requirements and their implementation. They focus on the expected system behaviour. In particular, security testing approaches based on access and usage control models fall into this category. Threat, fault and risk models focus on what can go wrong and concentrate on causes and consequences of system failures, weaknesses or vulnerabilities. Specifically, security testing based on fault and attack modelling approaches such as CORAS [28] or fault/attack tree analysis [29] fall into this category.

Weakness and vulnerability models describe weaknesses or vulnerabilities by themselves. The information needed to develop such models is provided by databases such as the Common Vulnerabilities and Exposures [30] database. Mutated models of the system under test are often used to represent weaknesses or known vulnerabilities, and to generate security tests. The classification scheme of Schieferdecker *et al.* defines a subset of the filter criteria of Section 3.1, blending the criteria 'model of system security', 'security model of the environment' and 'explicit test selection'. The two classification schemes are compatible, however, because there is a clear correspondence between the categories of Schieferdecker *et al.* and the categories of this article: Their architectural and functional models correspond to two categories of this article's system security model, namely 'functionality of security mechanisms' and parts of 'security properties'. Threat, fault and risk models correspond to security models of the environment, additionally considering the 'risk-based' test selection criterion. Finally, weakness and vulnerability models correspond to this paper's vulnerabilities category.

Test selection criteria for MBST are not explicitly covered by Schieferdecker *et al.* This renders the alignment with the established MBT classification of Utting *et al.* difficult. Differing from this article's classification, models of system security and security models of the environment are not explicitly distinguished by Schieferdecker *et al.* However, this facilitates the classification of concrete MBST approaches as presented in the next section. Moreover, the evidence aspect of MBST

DOI: 10.1002/stvr

is not covered by Schieferdecker *et al.* This article's classification scheme can therefore be considered to be more comprehensive, to facilitate the classification of concrete MBST approaches and to be aligned with the MBT taxonomy of Utting *et al.* However, as the classification scheme presented in the next section is compatible with that of Schieferdecker *et al.*, the classification results of concrete MBST approaches performed in Section 4 are carried over to the classification of Schieferdecker *et al.*

3. CLASSIFICATION CRITERIA FOR MODEL-BASED SECURITY TESTING

Model-based security testing is model-based testing of security requirements. MBST approaches can therefore be classified according to any classification scheme for MBT presented in Section 2.1.2. Specifically, Dias-Neto and Travassos explicitly consider MBST approaches because they classify security as a subtype of non-functional requirements and collect 16 MBST approaches. However, none of the existing MBT classification schemes (Section 2.1.2) consider specific aspects of security testing discussed before, including risk-based testing, fault-injection, fuzz testing, vulnerability coverage and evidence. This section defines criteria for security concerns in model-based testing to the end of systematically classifying MBST approaches. The criteria complement existing MBT classification schemes and are aligned with the taxonomy of Utting *et al.* They aim at characterizing and comparing existing as well as future MBST approaches and trends.

Literature reveals that test case generation technology is not domain-specific, that is, not tailored to security testing. It appears, hence, reasonable to concentrate on modelled security aspects to select (filter) security test cases and the available evidence of the approaches. This gives rise to two criteria groups, *filter criteria* and *evidence criteria*. Each criterion has several *values* and a corresponding *scale type*, which defines how the values are assigned to specific approaches. The scale type is *single-select* or *multi-select*.

Filter criteria comprise the specification of the *model of system security*, the *security model of the environment* and *explicit test selection criteria*. In combination, these models determine security-specific filters on the system model and define a finite subset of the traces of an SUT that is of interest for security testing purposes. Filter criteria address specific techniques of security vulnerability testing, for example, risk-based testing, fault-injection, fuzz testing and vulnerability coverage. Filter criteria are discussed in detail in Section 3.1.

Evidence criteria comprise *maturity of evaluated system*, *evidence measures* and the *evidence level*. In combination, these criteria assess the industrial applicability and utility of MBST approaches as well as the actual state in research. Evidence criteria are discussed in detail in Section 3.2.

Figure 2 gives an overview of the MBST criteria and their dimensions introduced in this section.

3.1. Filter criteria

Filter criteria provide means to select relevant test cases. They are called filters because they define 'interesting' subsets of all the traces of an SUT, or a model of this SUT. Filter criteria comprise security-specific models of a system and/or its environment as well as explicit test selection criteria. These criteria formally define *security test objectives* and describe *what* is modelled: parts of the SUT, its environment or, directly, the basis for test case selection. In the following, the filter criteria 'model of system security', 'security model of the environment' and 'explicit test selection criteria' as well as their scale types and values are described in more detail.

3.1.1. Model of system security. Models of system security are partial models of an SUT that focus on security aspects. These models consist of security properties, vulnerabilities and functionality of security mechanisms. As these models can be combined, the scale type of this criterion is *multi-select*. In particular, the types of system security models are as follows:

- *Security Properties*: These are models of security properties such as confidentiality, integrity, availability, authentication, authorization or non-repudiation.

DOI: 10.1002/stvr

- *Vulnerabilities*: These are models of characteristics of a system without which a specified class of security breaches would not be possible. Vulnerabilities can be described similar to how faults are described by fault models, usually as a deviation from a correct programme or system [31]. Examples for models of vulnerabilities include security mutation operators.
- *Functionality of Security Mechanisms*: These are models of functional security measures such as access control or usage control means. The model here usually is provided by the policy that configures the respective mechanism.

3.1.2. Security model of the environment. Security models of the environment are models of security aspects of the technical, organizational and operating context of an SUT. They concentrate on the causes and potential consequences of system behaviour and interfaces to the SUT. Note the difference between modelling a vulnerability, that is, a part of a (partial) model of the SUT, and modelling exploits of that vulnerability, that is, parts of the environment of the SUT. Environment models for security testing are always bound to specific vulnerabilities and describe how to target a specific vulnerability (or class of vulnerabilities). The scale type of security models of the environment is *multi-select* with values threat and attack model, which are defined as follows:

- *Threat Models*: These models describe threats, that is, potential causes of an incident, that may result in harm of systems and organization [32].
- *Attack Models*: These models describe sequences of actions to exploit vulnerabilities. Threats may be considered as their causes. Attack models may be testing strategies such as string or behaviour fuzzing, possibly on the grounds of grammars, or syntax models, that generate test cases.

The relationship between threats and attacks can be illustrated by attack trees: an attack tree represents a conceptual view on how an asset might be attacked. It describes a combination of a threat and corresponding attacks. Usually, the top-level event (root node) of an attack tree represents the threat. In contrast, each cut of an attack tree, with the root node removed from this cut, is an attack. Several non-root nodes of the tree exploit vulnerabilities. The children of a node express conditions that need to be fulfilled in order to exploit the vulnerability at the parent node. If all nodes in a cut (including the root node) are made true, the corresponding attack is possible and leads

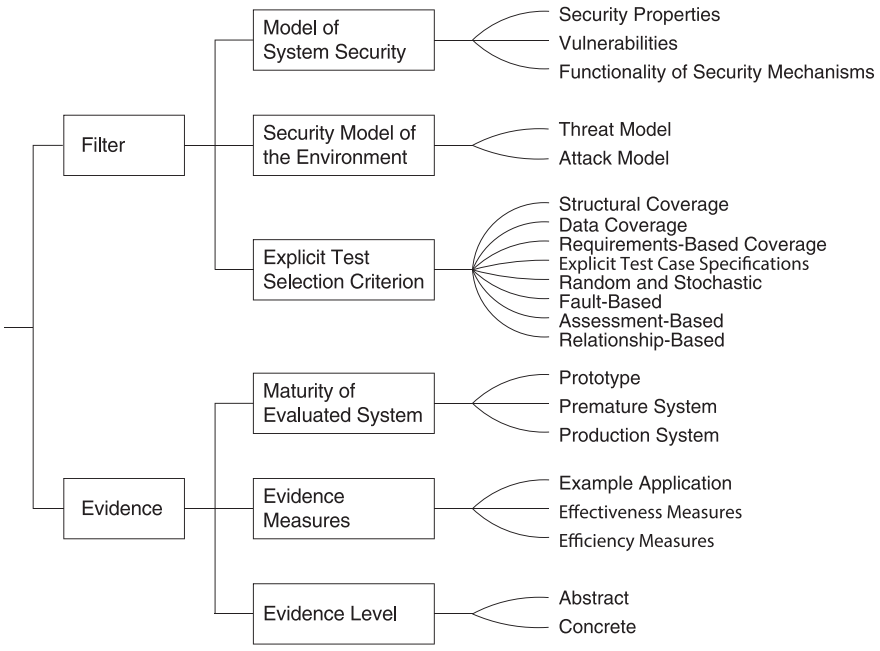

Figure 2. Overview of the model-based security testing classification criteria.

DOI: 10.1002/stvr

to the described threat. Note that vulnerabilities are inherently bound to an SUT while attacks are inherently bound to an SUT's environment. Test selection is performed using both.

3.1.3. Explicit test selection criteria. Test selection criteria define how to further cut down the number of possible traces defined by properties, security mechanisms, vulnerabilities and attacks. Only if these sets are sufficiently small, they can be used for testing purposes. They are models of what makes an SUT's traces interesting as test cases. The types of explicit test selection criteria are as follows:

- *Structural Coverage*: These criteria exploit the structure of a model, such as the nodes and arcs of a transition-based model, or conditional statements in a model in pre/post notation.
- *Data Coverage*: These criteria deal with how to choose a few test values from a large data space.
- *Requirements-Based Coverage*: These criteria deal with coverage of informal requirements, which is possible when elements of the model can be explicitly associated with informal requirements.
- *Explicit Test Case Specifications*: Test case specifications in some formal notation are used to determine which tests will be generated. For example, they may be used to restrict the paths through the model that will be tested, to focus the testing on critical scenarios or to ensure that particular paths will be tested.
- *Random and Stochastic*: These criteria consider directly or indirectly modelled probabilities such as probabilities of actions modelling a usage profile or randomly generated invalid or unexpected input.
- *Fault-Based*: These criteria address the explicit coverage of unintentionally added and artificially injected faults. The most common fault-based criteria are *mutation coverage* criteria. This involves mutating a model, then generating tests that would distinguish between the mutated model and the original model. The mutants are similar to vulnerabilities, and the generated tests simulate attacks.
- *Assessment-Based*: These criteria select test cases on the basis of calculated and/or estimated values associated with test cases. Popular assessment-based criteria for test case selection are based on risk values that *determine probabilities and consequences of certain threats* addressed by test cases. Other metrics for assessment might include costs, priorities or severities as well as size, complexity or change measures.
- *Relationship-Based*: These criteria select tests on the basis of the relationship between models. These models can be on different abstraction levels such as a design and an implementation level model or be different versions of one model in the case of *regression testing*.

Because these models can be combined, the scale type of this criterion is *multi-select*. For instance, *fuzz testing* [33] is an effective technique for finding security vulnerabilities and provides an example of how test selection criteria are combined. Traditionally, fuzz testing tools apply random mutations to well-formed inputs of a programme and test the resulting values to uncover vulnerabilities. Thus, fuzz testing combines fault-based and random test selection.

The listed explicit test selection criteria are based on the test selection dimension of the MBT taxonomy of Utting *et al.* In addition, it contains the additional criterion 'risk-based', which is of particular importance for security testing [34], and 'relationship-based' to cover relationships between models of different levels of abstraction and versions.

3.1.4. Relationship. It was described before how a combination of a model of system security, a security model of the environment and test selection criteria define *filters* according to which security tests are selected or generated. Such a filter defines a relevant finite subset and truncation of the possible execution traces of the SUT and can be achieved on the basis of several operations:

- *Relating a security property to SUT specification*: Security properties, expressed as statements in any form of mathematical logic, can be imposed by computing their logical conjunction with the (formal) specification of the SUT. This is usually interpreted as intersection of the trace sets

DOI: 10.1002/stvr

of the property and the SUT specification. By intersection with the traces of a security property, the set of traces of the SUT can only become smaller or remain identical. For testing purposes that aim at violating properties, negated or mutated properties are often used.

- *Exploiting vulnerabilities*: In order to exploit vulnerabilities, the system must first be brought into a state where it is possible to perform the exploit. Models of vulnerabilities therefore restrict the number of traces to those that visit states in which vulnerabilities can potentially be exploited.
- *Defining functionality of security mechanisms*: A model of the functionality of a security mechanism, for example, an access control mechanism, reduces the possible traces by forbidding those system transitions that are not permitted with respect to a policy. For testing purposes, negated or mutated policies are often used.
- *Composing an SUT model with an environment model*: This is achieved by connecting the input channels of the SUT model to the output channels of the environment model, and by connecting the output channels of the SUT model to the input channels of the environment model. If the environment model does not allow arbitrary sequences of inputs to be sent to the SUT—and this is usually the purpose of an environment model—then this composition reduces the set of relevant traces of the SUT to those that are permitted by the environment.
- *Explicitly specifying selection criteria*: Explicitly specified test selection criteria define how to further cut down the trace set of the SUT. For instance, one can specify that only commands identified as critical are executed, that the filter applied to all traces yields all those traces that satisfy a coverage criterion or that no repetition of events may occur.

In the literature, threats, attacks and vulnerabilities are encoded by all these means and combinations thereof. Note that different explicit test selection criteria may have varying applicability for combination with system or environment models. For instance, fault-based criteria are mostly applicable to system models, because the goal is to find faults in the SUT. But, random and stochastic criteria are mostly applicable to environment models, because it is the environment that describes the usage patterns and input of the SUT.

3.2. Evidence criteria

Evidence criteria determine how much evidence is available regarding the usefulness of an MBST approach. As evidence has several facets, these criteria comprise *maturity of evaluated system*, *evidence measures* and the *evidence level* and are assessed for different applications of an MBST approach. Evidence is not specific to MBST but relevant to MBT in general. This is shown by the fact that Dias-Neto and Travassos already consider the classification criterion ‘analysis per type of experimental evidence’ [6] with category values ‘speculation’, ‘example’, ‘proof of concept’, ‘experience/industrial reports’ and ‘experimentation’. But, especially for security testing where the selection and application of suitable test techniques is very critical and complex, the documentation of evidence requires a more many-faceted classification scheme. Therefore, the following three *orthogonal* evidence criteria helping in this respect are defined: maturity of evaluated system, evidence measures and evidence level. Considering not only the industrial application but also the evidence reported in research, each of the three evidence criteria is *optional* as specific results may not be available as in speculative approaches. The defined criteria take a related classification scheme for security regression testing into account [26], where security regression testing approaches are classified with regard to the evidence criteria evaluated system, maturity of evaluated system and evaluation measures, and adopt it to model-based security testing and the level of abstraction required for a generic taxonomy. In the following, the evidence criteria maturity of evaluated system, evidence measures and evidence level as well as their respective scales and values are discussed.

3.2.1. Maturity of evaluated system. The criterion maturity of evaluated system captures the technical soundness and degree of deployment of the most mature system under test used to evaluate an MBST approach. Its scale is *single-select* with values prototype, premature system and production system, which are defined as follows:

DOI: 10.1002/stvr

- *Prototype*: This is an early sample or release of a system with the purpose to be replicated, learned from or to evaluate other approaches. Prototypes have some well-known limitations (for instance, with regard to security). They are in a development state or have been developed without industrial or technical pressure.
- *Premature System*: This is a running system, not yet performing tasks valuable to stakeholders on a regular basis. This intermediate value is introduced for systems that are already more mature than a prototype but not yet productive.
- *Production System*: This is a running system, performing tasks valuable to stakeholders on a regular basis.

3.2.2. Evidence measures. Evidence measures determine the qualitative or quantitative assessment criteria to evaluate an MBST approach in a specific application. These measures are 'example application', and with increased significance 'effectiveness measures' as well as 'efficiency measures'. As these measures can be combined, the scale type of this criterion is *multi-select*, and its values are example application, effectiveness measures and efficiency measures defined as follows [26]:

- *Example Application*: An approach has been used in a documented context indicating the feasibility of the approach application typically in a qualitative way.
- *Effectiveness Measures*: In general, they measure whether the intended or expected result (effect) is produced. In the context of software testing, effectiveness measures judge the effect of tests on the system under test. A common effectiveness measure is the number of faults found, sometimes also divided by the number of test cases executed. For security testing purposes, especially the number of found vulnerabilities or mutations is of interest.
- *Efficiency Measures*: In general, they measure performing or functioning in the best possible manner with the least waste of time and effort. In the context of software testing, efficiency is measured by relating artefacts such as faults, test cases or tested model elements to required time or cost, ideally by contrasting it to other approaches. For security testing purposes, especially the number of found vulnerabilities or mutations in relation to time or cost required for their identification is of interest.

3.2.3. Evidence level. The evidence level determines whether the approach is evaluated on the level of non-executable abstract or executable concrete test cases [13]. Evaluation on the abstract level and on the level of executable test cases can be combined. The scale of this criterion is therefore *multi-select* with values abstract and executable, which are defined as follows:

- *Abstract*: On this level, an approach is evaluated on the basis of non-executable test cases. For instance, if generated test cases are sequences of non-executable nodes of a transition-based model, then evidence can already be measured (efficiency can, for example, be measured on the basis of a cost or time model). But, the concrete effect of the approach on the deployed system under test cannot be measured.
- *Executable*: On this level, an approach is evaluated on the basis of test cases, which are executable against the system under test. Abstract test cases are often generated as an intermediate step. Compared with the abstract level, the set of possible evidence measures is not limited.

4. SYSTEMATIC SELECTION AND CLASSIFICATION OF PUBLICATIONS ON MODEL-BASED SECURITY TESTING

This section describes the systematic selection of model-based security testing publications and their classification according to the filter and evidence criteria defined in the previous section. Threats to validity of the selection and classification procedure are discussed. The resulting classification of MBST approaches covered in the selected publications is then used in the next section to indicate the adequacy of the defined criteria and for further discussions.

DOI: 10.1002/stvr

4.1. Selection of publications

The selection of relevant peer-reviewed primary publications comprises the definition of a search strategy and paper selection criteria as well as the selection procedure.

4.1.1. Search strategy. The conducted search of the relevant publications included an automatic search in the following digital libraries:

- IEEE Digital Library (http://ieeexplore.ieee.org/);
- ScienceDirect (http://www.sciencedirect.com/);
- Springer Link (http://link.springer.com);
- ACM Digital Library (http://portal.acm.org/); and
- Wiley (http://onlinelibrary.wiley.com/).

These libraries were chosen because they cover the most relevant sources in software and security engineering [35].

The search string applied to the digital libraries is as follows:

```
( "model based" OR automata OR "state machine" OR
  "specification based" OR policy OR policies OR
  "threat model" OR mutation OR risk OR fuzzing )
AND ( security OR vulnerability OR privacy OR cryptographic )
AND ( test OR testing )
```

This search string combines terms for (security) models and artefacts with terms for security and testing. It is based on the search string of Dias-Neto and Travassos used in their systematic review on model-based testing [6]. The search string was piloted and iteratively improved to cover all relevant publications from a reference database on model-based security testing from a research project that was additionally extended by key publications known by the three researchers who performed the search and are all experts in the field of model-based security testing. The reference database contained 76 relevant publications from all five searched digital libraries. The final version of the search string covered all relevant publications of this reference database and therefore had a recall, that is, ratio of the retrieved relevant items and all existing relevant items [36], of 100 percent. The start year of the search was set to 1996, which is the publication year of the earliest model-based security testing approach listed in the survey of Dias-Neto and Travassos [6]. The last year of inclusion is publications within 2013. The search fields title, abstract and keywords were considered.

4.1.2. Paper selection. On the set of papers found with the search strategy, suitable inclusion and exclusion criteria for selecting relevant primary publications were applied. Included were peer-reviewed publications written in English that cover a model-based security testing approach as defined in Section 1. Excluded were related approaches to classical penetration testing, static analysis, monitoring and testing of properties such as robustness, safety or trust; these do not cover a model-based security testing approach according to the definition in this article.

As illustrated in Figure 3, the initial search on the basis of the search strategy delivered the following results. In the ACM Digital Library, 3038 publications were found, in the IEEE Digital Library 1769, in Science Direct 111, in Springer Link 1000 and in Wiley 10. On the basis of this result set, the relevant publications were selected in three phases as shown in Figure 3 by three researchers.

In phase 1, duplicates and irrelevant papers according to the selection criteria were removed on the basis of the *title*. In this phase, 5268 papers were excluded from the originally retrieved publications leaving 660 filtered papers.

In phase 2, irrelevant papers according to the selection criteria were removed on the basis of the *abstract*. In this phase, 336 papers were excluded from the filtered publications leaving 324 papers for full text reading.

In phase 3, irrelevant papers according to the selection criteria were removed on the basis of the *full text*. Papers with a length below four pages were additionally removed. In this phase, 205 papers

DOI: 10.1002/stvr

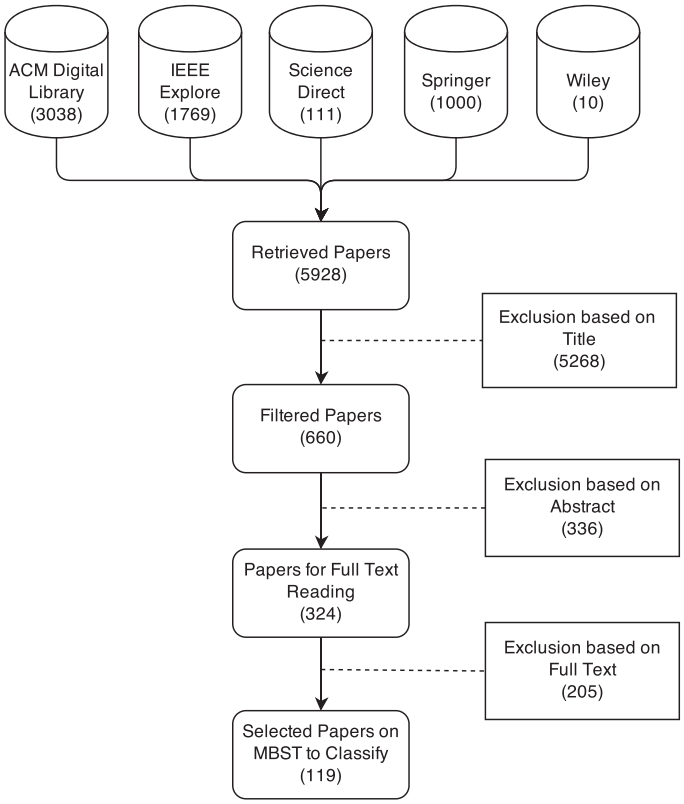

Figure 3. Paper selection procedure. MBST, model-based security testing.

were excluded leaving 119 papers for further classification according to the classification criteria for MBST.

4.2. *Paper classification*

The selected 119 primary publications on model-based security testing were classified according to the filter and evidence criteria defined in Section 3 by the three researchers who also performed the paper selection. For this purpose, the selected primary publications accepted for classification were randomly divided into three sets of equal size for data extraction and classification. A data extraction and classification spreadsheet for the classification scheme was designed. Besides *bibliographic information* (title, year and publisher), the form contains *classification fields* for each defined criterion, that is, a field for model of system security, security model of the environment, explicit test selection criterion, maturity of evaluated system, evidence measures, and evidence level. The criterion model of system security was classified as security properties only if at least one of the security properties confidentiality, integrity, availability, authentication, authorization or non-repudiation was explicitly defined in the model. To avoid input errors and misclassification, the scale and characteristics of each classification criterion were additionally implemented as a respective item list for each classification field. Each list also contained the item 'not specified', to cater for the fact that a specific criterion is not defined or classification with regard to it is not possible due to the information available in the publication. Problems during the classification were remarked in an additional *comment field*. The resulting classification of all publications was reviewed independently by all three researchers. Remarks on the classification of specific papers were also annotated in the comment field. Finally, in a group session, all comments were discussed and resolved among all three researchers.

DOI: 10.1002/stvr

4.3. Threats to validity

Different factors may influence the results of this paper's systematic classification, such as the search string applied. Threats to validity include publication bias as well as the identification and classification of publications.

4.3.1. Publication bias. This threat refers to the general problem that only certain approaches are published, for instance, those with positive research outcomes or those promoted by influential organizations [37]. The authors regard this threat as moderate as the sources of information are not restricted to a certain publisher, journal or conference. Therefore, it can be assumed that the breadth of model-based security testing approaches is covered sufficiently. However, to consider the trade-off between considering as much literature as possible and, at the same time, accumulating reliable information, gray literature (technical reports, work in progress, unpublished or not peer-reviewed publications) was excluded [37]. In addition, the minimum page number for selected approaches was set to four to guarantee that enough information is available to classify the approaches appropriately.

4.3.2. Threats to the identification of publications. The strategy to construct the search string aimed to retrieve as many relevant papers as possible related to model-based security testing. Because it is impossible to know all existing relevant publications, the recall of the search string was estimated by conducting a pilot search as described in Section 4.1.1; that is, the search string was iteratively improved to cover all relevant publications from a reference database on model-based security testing. In order to additionally reduce the threat of missing important publications, we informally checked papers referenced by the selected papers. We did not become aware of any missed frequently referenced paper. Further optimizations of the search string reduced its precision, that is, the degree how well the search identifies only relevant items, significantly, and therefore, the search string was not further adapted. In addition, the full text of a few papers could not be retrieved within the scheduled time frame. Single missing publications seem like a moderate threat to validity as the aim of this article is not to perform a comprehensive systematic literature review but validate the classification scheme and characterize the state of the art of MBST.

4.3.3. Threats to classification of publications. Because of the high number of primary publications that have to be classified according to several criteria with—to some extent—a high number of characteristics, the threat of misclassification has to be considered. Several measures mitigate this threat. First, the classification scheme is clearly defined in Section 3. Second, for each classification field in the classification form, an item list to select the respective characteristics was implemented. Third, the classification was conducted in parallel by three researchers, who are all experts in the field of model-based security testing, and the classification result was reviewed independently by all of them. Comments were discussed and resolved in a final group session among all three researchers.

5. RESULTS AND DISCUSSION

This section presents and discusses the classification results of the selected MBST publications. The complete classification of all 119 identified MBST publications, in terms of the proposed taxonomy and its intended abstraction level, is shown in the appendix of this article (Table B.1 in Section 6) and is also available online.[‡]

Table I shows the frequencies of all possible combinations of models of system security and security models of the environment currently appearing in reported MBST approaches. Because both criteria (i.e. 'model of system security' and 'security model of environment') do not represent a mandatory artefact in an MBST approach, we additionally added the value 'n.s.' (i.e. not specified) to make explicit that a specific type of model is not considered. Table I shows that every approach specifies either a model of system security or a security model of the environment. In addition, the

[‡]http://qe-informatik.uibk.ac.at/mbst-classification.

DOI: 10.1002/stvr

Table I. Number of model-based security testing approaches per type of security model of the environment and model of system security, where '+' represents a combination of different types of security models of the environment or models of system security and 'n.s' stands for 'not specified'.

	SecP	V	FSecM	SecP+V	SecP+FSecM	V+FSecM	n.s.	Sum
T	1	0	3	0	0	0	2	6
A	2	0	1	0	0	0	20	23
T+A	1	1	0	0	1	0	3	6
n.s.	15	7	42	2	9	9	0	84
Sum	19	8	46	2	10	9	25	119

'T', threat model; 'A', attack model; 'SecP', security properties; 'V', vulnerabilities; 'FSecM', functionality of security mechanisms.

Table II. Number of publications reporting a specific type of test selection criterion per type of security model.

	SecP	V	FSecM	T	A	Sum
SC	13	7	38	7	16	81
DC	0	1	10	2	1	14
RC	3	0	4	2	0	9
TCS	9	1	9	1	6	26
RS	1	1	5	0	0	7
FB	6	9	12	3	5	35
AB	0	1	0	0	3	4
RB	2	2	2	0	1	7

'SC', structural coverage; 'DC', data coverage; 'RC', requirements coverage; 'TCS', explicit test case specifications; 'RS', random and stochastic; 'FB', fault-based; 'AB', assessment-based; 'RB', relationship-based; 'SecP', security properties; 'V', vulnerabilities; 'FSecM', functionality of security mechanisms; 'T', threat model; 'A', attack model.

majority of approaches (109 out of 119) focus on one model, and only a few approaches (10 out of 119) consider both types of models at the same time.

The by far most widely used type is models of functionalities of security mechanisms (FSecM), that is, 65 of overall 119 consider FSecM, after all 42 of these use solely FSecM models. Because of the high number of approaches deriving test cases from access control models, this high number of approaches related to FSecM is not surprising. In general, models of system security are much more frequent than security models of the environment as 84 MBST approaches do not consider environment models at all. From the remaining 35 MBST approaches, which take security models of the environment into account, most approaches, that is, 29, consider attack models. So, it is more common and even practical for automatic test derivation to model sequences of actions to exploit vulnerabilities, that is, attacks, than potential causes of incidents, that is, threats. Finally, Table I shows that only 10 papers consider a combination of 'model of system security' and 'security model of environment'.

5.1. Results

For each publication, it was possible to classify the covered MBST approach uniquely according to the defined filter and evidence criteria. This indicates the adequacy of the defined criteria for the classification of MBST approaches providing a framework to understand, categorize, assess, compare and select MBST approaches for specific purposes. Besides validation of this paper's filter and evidence criteria, the classification of MBST approaches enables, as it was performed in a systematic and comprehensive way, an aggregated view and analysis of the state of the art of model-based security testing. Tables I–III provide an overview of the security models and test selection criteria used in MBST as well as the available evidence.

DOI: 10.1002/stvr

Table III. Evidence reported in model-based security testing publications per maturity of evaluated system, evidence measures and evidence level , where '+' represents a combination of evidence levels

		Abs	Exec	Abs+Exec	Sum
Prototype					
	Ex	36	22	23	81
	Effe	3	15	7	25
	Effi	3	9	7	19
	Sum	42	46	37	125
Premature					
	Ex	1	0	1	2
	Effe	0	2	1	3
	Sum	1	2	2	5
Production					
	Ex	3	3	1	7
	Effe	2	7	0	9
	Effi	1	0	0	1
	Sum	6	10	1	17

'Premature', premature system; 'Production', production system; 'Ex', example application; 'Effe', effectiveness measures; 'Effi', efficiency measures; 'Abs', abstract; 'Exec', executable.

Table II shows the explicit test selection criteria used for specific security models of the system or environment. The dominant criterion for test selection is structural coverage. It is used in 81 approaches, especially for models of functionality of security mechanisms, but also for security properties and attacks. Similar to MBT [6], also in MBST, many models are graphs, and structural coverage, for example, of nodes and arcs, therefore seems like the natural strategy to systematically cover it and derive test cases. Compared with structural coverage, complementing data coverage is reported comparatively rare, that is, used in 14 approaches. The application of fault-based selection and explicit test case specification is also rather common and specific for MBST where testing often explicitly addresses vulnerabilities or attack scenarios. The selection criteria requirements coverage, random and stochastic, relationship-based, as well as assessment-based are rarely used but offer high potential for future work.

Table III summarizes the evidence reported for the selected MBST approaches. By far the most evidence is reported on the basis of prototypical systems under test. Only very few papers evaluate the presented approach on a premature or production system under test. Of those papers that evaluate the approach with a prototype, 81 papers show the success of the approach with the help of examples. Twenty-five papers use effectiveness measures, whereas 19 papers consider efficiency measures for the evaluation. This correlates with the difficulty of the evaluation. In order to show the effectiveness, the approach has to be applicable to examples. Efficiency puts effectiveness into context. If an approach is evaluated on the basis of premature or production system, effectiveness measures are applied more often. In addition, efficiency is almost never considered as evaluation measure for premature and production systems.

Table IV relates evidence to the different security model types and refines the results shown in Table III. Overall, only for one MBST approach [38], no evidence is specified. This shows the high importance of reporting the application and evaluation in papers on MBST approaches. For all model types, most approaches are evaluated on the basis of prototypes, especially many models of security mechanisms are evaluated in this way. Premature systems are rarely used for evaluation; for vulnerability models, they are not used at all. Attack models represent a high ratio of approaches evaluated on production systems. A reason for this could be that especially attack models, which describe sequences of actions to exploit vulnerabilities, often require complex environments for evaluation, which can best be provided by production systems. Effectiveness and efficiency measures are often defined for security mechanism, and the ratio of attack models with regard to example application is rather high. Finally, regarding the evidence level, it can be observed that the ratio of security

DOI: 10.1002/stvr

Table IV. Number of publications per type of security model reporting evidence per maturity of evaluated system, evidence measures and evidence level, where '+' represents a combination of evidence levels.

	Prot	Pre	Prod	Ex	Effe	Effi	Abs	Exec	Abs+Exec
SecP	27	3	1	25	5	5	17	5	9
V	17	0	2	13	9	4	4	11	4
FSecM	56	3	6	49	19	13	24	24	16
T	10	1	1	11	3	3	5	4	3
A	21	2	5	25	4	2	10	10	8
Sum	131	9	15	123	40	27	60	54	40

'SecP', security properties; 'V', vulnerabilities; 'FSecM', functionality of security mechanisms; 'T', threat model; 'A', attack model; 'Prot', prototype; 'Pre', premature system; 'Prod', production system; 'Ex', example application; 'Effe', effectiveness measures; 'Effi', efficiency measures; 'Abs', abstract; 'Exec', executable.

properties is rather high on the abstract level and of vulnerabilities as well as attack models on the executable level. A reason for this could be that security properties can often be evaluated statically by a formal calculus, whereas vulnerabilities and attack models exploiting these require dynamic systems for evaluation.

5.2. Discussion

The main purpose of this article is to provide a classification schema for security testing approaches. It can also be used to characterize the current state of the art. A comprehensive discussion of the state of the art—as opposed to a characterization, which is provided in this article—is necessarily incomplete. In the authors' opinion, in order to suggest necessary research directions, it is worth concentrating on the kind of security properties that are tested, on the use of coverage criteria and on the feasibility and return on investment (ROI) of model-based security testing.

5.2.1. Security properties and vulnerabilities. Firstly, considerably more research is devoted to testing based on mechanisms or properties than to testing based on vulnerabilities. On the one hand, this appears surprising because many approaches to test case derivation are inherently fault-based [31], and a fault naturally corresponds to a vulnerability. On the other hand, with the exception of testing object-oriented software, many approaches to fault-based test case derivation are not based on explicit models, and the derivation is often manual rather than automatic. One can speculate that the reason for this is that vulnerability models are hard to design—there is no simple analogon to the simple fault models 'division-by-zero' or 'null-pointer dereferencing'. In any case, it is a surprising finding given that most security analyses in practice start by hypothesizing about security vulnerabilities in a system.

Secondly, the previous aggregated numbers also show that considerably more research is devoted to testing the functionality of security mechanisms than to testing security properties as such. It is hard to theoretically characterize the difference between testing an access control mechanism and testing for an authentication property or a specific trace-based interpretation of confidentiality (both define trace sets that are intersected with the trace set of an SUT, see Section 3.1.4). However, the intuitive difference is the one between testing a functional property and a non-functional property. It seems fair to speculate that the reason for this bias in research is due in part to historical reasons: model-based testing technology has been around for a long time, and it is of course appealing to apply it to specific pieces of functionality [6]. Moreover, testing a security mechanism intuitively is simpler because it targets one specific component rather than the entire system. On the downside, it is not always clear that a security mechanism indeed implements a system-wide security property such as confidentiality, integrity or availability.

Thirdly, collecting all classes of addressed security properties during a further analysis on the 31 papers classified as 'property-based' yields the following distribution. Authentication and access control properties are addressed by 13 papers each. Authorization is closely following with 10

DOI: 10.1002/stvr

papers, and confidentiality is addressed by eight papers. Integrity and availability are security properties that are covered by three papers each. The focus in research is therefore rather on confidentiality than on integrity and availability. While it is true that buffer overflow or string format vulnerabilities that are detected by fuzz testing can be used for compromising all three, confidentiality, integrity and availability, the current research focus rather clearly is on confidentiality. The authors of the present article deem this to be interesting, given that also integrity plays such an important role in current information technology systems. (Note that in different research domains, confidentiality is often defined as non-interference. Non-interference is a hyper property, i.e., a property of sets of traces than of traces, and as such cannot be falsified by one single test. When it comes to testing, usually a less demanding trace-based definition of confidentiality is chosen in the literature: some data may not leave a specific boundary.)

The authors of the present article do see room for research that (1) explicitly addresses vulnerabilities, (2) targets understanding the precise relationship between (local) security mechanisms and (system-wide) security mechanisms and (3) directly targets system-wide security properties. Moreover, as far as availability is a concern, there seems to be room for cross-fertilization with the research communities working on software reliability and stress testing.

5.2.2. Coverage criteria. A further observation is that the use of coverage criteria for model-based security testing enjoys great popularity. This is also true for model-based testing in general. One of the reasons is that coverage criteria naturally lend themselves to the automated generation of test cases. However, whether or not coverage criteria ultimately lead to test cases with good fault detection effectiveness, is subject to an ongoing debate; see, among others, respective related work [31, 39–45]. The intuition is that because coverage criteria in general are not defined using an underlying fault model, their fault detection ability depends on how faults are distributed in a programme, and this in general cannot be predicted. In the context of security testing, coverage helps to identify potentially vulnerable programme parts to which malicious input can be applied, regardless of whether explicit models are used, for example, for fuzz testing [46].

The authors of the present article do see room for research to better understand which variants of coverage criteria can yield effective and efficient security tests in the specific context of *model-based* security testing.

5.2.3. Feasibility and ROI of model-based testing. To date, the ROI of model-based testing has not been sufficiently understood, and only some rough ROI calculations are available in the literature [14, 47]. This is reflected in little evidence about efficiency measures. The idea to build and manage models and push a button for test case generation, as opposed to build and manage test suites, is appealing and has been shown to be beneficial, specifically when reuse is considered. However, there are several practical concerns. These include finding an adequate level of abstraction, necessary education to do the modelling, efficiency of test case generation, debugging aids for models, concurrent development of models, synchronization of model versions with the system versions they represent, the embedding in the development process and the overall cost-effectiveness. In many cases, also in model-based security testing, a model of the SUT is needed, both (1) for guiding test case generation and (2) as an oracle. If such a model of the SUT is required, then all concerns regarding model-based testing are inherited by model-based security testing. Moreover, in addition to a common intuition that security is often particularly domain-, data- and application-specific, there is the perspective that security testing is fundamentally bound to the expertise of security analysts [48]. This means that it is hard for some to imagine that model-based security testing, used by non-experts, will turn out as an effective and efficient security assessment technique.

The authors of the present article do see room for research to understand precisely which kinds of models are necessary and beneficial for model-based security testing; if generic security

DOI: 10.1002/stvr

models can be built that can be reused in different contexts; and to empirically understand the return on investment.

6. CONCLUSIONS

This article presented a taxonomy for model-based security testing, showed its adequacy by classifying systematically selected model-based security testing approaches and characterized the current state of the art. The taxonomy is based on a comprehensive analysis of existing classification schemes for model-based testing and security testing. The taxonomy complements existing classification schemes by the definition of filter and evidence criteria. Filter criteria comprise the specification of the *model of system security*, the *security model of the environment* and *explicit test selection criteria*. In combination, these models determine security-specific filters on the system model and define a finite subset of the traces of an SUT that is of interest for security testing purposes. Evidence criteria comprise *maturity of evaluated system*, *evidence measures* and the *evidence level*. The systematic selection of available model-based security testing approaches, which is based on five digital libraries covering the most relevant sources in software and security engineering, identified 119 approaches. For each approach, it was possible to uniquely classify it according to the defined filter and evidence criteria. The complete classification is shown in the appendix of this article and is also available online.[§] On this basis, this article also provided an overview of the state of the art in model-based security testing, as reflected in 119 selected papers. Finally, the article discussed promising research directions with regard to security properties, coverage criteria, and the feasibility and return on investment of model-based security testing. Because model-based security testing is an active research field [1], a taxonomy as provided in this article delivers a valuable starting point for further research on model-based security testing and helps to clarify the key issues of the field and to show possible alternatives and directions. We further see the necessity for future research to classify the selected papers into more fine-grained criteria, for instance with regard to the types of systems or the types of vulnerabilities the identified approaches deal with, to allow more fine-grained search for specific model-based security testing approaches.

APPENDIX A: ABBREVIATIONS FOR FILTER AND EVIDENCE CRITERIA

Tables A.1 and A.2 define the abbreviations for filter and evidence criteria used for the classification of MBST publications shown in Table B.1 of Section 6.

Table A.1. Abbreviations for filter criteria.

	Abbreviation	Meaning
Model of system security	MSSec	
	SecP	Security properties
	V	Vulnerabilities
	FsecM	Functionality of security mechanisms
	n.s.	not specified
Security model of environment	SecME	
	A	Attack model
	T	Threat model
	n.s.	not specified
Test selection criteria	TS	
	SC	Structural coverage
	DC	Data coverage
	RC	Requirements coverage
	TCS	Explicit test case specifications

[§]http://qe-informatik.uibk.ac.at/mbst-classification.

DOI: 10.1002/stvr

Table A.1. Continued.

Abbreviation	Meaning
RS	Random and stochastic
FB	Fault-based
AB	Assessment-based
RB	Relationship-based
n.s.	not specified

Table A.2. Abbreviations for evidence criteria.

	Abbreviation	Meanings
Maturity of evaluated system	MES	
	Prot	Prototype
	Pre	Premature system
	Prod	Production system
	n.s.	not specified
Evidence measures	EM	
	Ex	Example application
	Effe	Effectiveness measures
	Effi	Efficiency measures
	n.s.	not specified
Evidence level	EL	
	Abs	Abstract
	Exec	Executable
	n.s.	not specified

APPENDIX B: CLASSIFICATION OF SELECTED PUBLICATIONS ON MODEL-BASED SECURITY TESTING

Table B.1 shows the classification of the selected model-based security testing publications sorted by author names and applying the abbreviations from Tables A.1 and A.2 of Section 6. The list of model-based security testing approaches is also available online[¶] as interactive table supporting filtering and sorting on each column.

Table B.1. Classification of selected model-based security testing publications.

	Filter criteria			Evidence criteria		
Paper	MSSec	SecME	TSC	MES	EM	EL
Abassi & Fatmi [49]	SecP+FSecM	n.s.	TCS	Prot	Ex	Abs
Abbassi & El Fatim [50]	SecP	n.s.	TCS	Prot	Ex	Abs
Al-Azzani & Bahsoon [51]	n.s.	A	TCS	Prot	Ex	Abs
Al-Shaer *et al.* [52]	FSecM	n.s.	RS	Prot	Effe+Effi	Abs+Exec
Allen *et al.* [53]	V	n.s.	SC	Prot	Ex+Effe	Exec
Antunes & Neves [54]	FSecM	n.s.	RS	Prot	Ex	Abs+Exec
Armando *et al.* [55]	SecP+FSecM	n.s.	SC	Pre	Effe	Exec
Bartel *et al.* [56]	V	n.s.	AB	Prot	Ex	Abs+Exec
Belhaouari *et al.* [57]	FSecM	n.s.	SC+DC+RS	Prot	Ex+Effi	Exec
Bertolino *et al.* [58]	FSecM	n.s.	DC	Prot	Effe+Effi	Exec
Bertolino *et al.* [59]	FSecM	T	FB	Prot	Ex+Effe+Effi	Abs+Exec
Beyer *et al.* [60]	SecP	n.s.	SC	Prot	Effe+Effi	Abs+Exec

[¶]http://qe-informatik.uibk.ac.at/mbst-classification.

DOI: 10.1002/stvr

Table B.1. Continued.

	Filter criteria			Evidence criteria		
Paper	MSSec	SecME	TSC	MES	EM	EL
Blome *et al.* [61]	n.s.	A	TCS	Prot	Ex	Exec
Bortolozzo *et al.* [62]	FSecM	n.s.	FB	Prod	Effe	Exec
Botella *et al.* [63]	FSecM	n.s.	SC+RC	Prot	Ex+Effi	Abs+Exec
Bozic & Wotawa [64]	n.s.	A	SC	Prot	Ex	Exec
Bracher & Krishnan [65]	SecP	n.s.	RC+TCS	Prot	Ex	Abs
Brucker *et al.* [66]	FSecM	n.s.	SC+DC	Prot	Effi	Abs
Brucker & Wolff [67]	FSecM	n.s.	SC+DC	Prot	Effi	Abs
Brucker *et al.* [68]	FSecM	n.s.	SC+DC	Prod	Ex	Abs
Brucker *et al.* [69]	FSecM	n.s.	SC+DC	Prot	Ex	Abs
Büchler *et al.* [70]	SecP	n.s.	FB	Prot	Ex	Abs
Büchler *et al.* [71]	SecP	n.s.	FB	Prot	Ex+Effe+Effi	Abs+Exec
Chen *et al.* [72]	SecP+FSecM	n.s.	SC	Prot	Ex	Abs+Exec
Dadeau *et al.* [73]	FSecM	n.s.	FB	Prot	Ex	Abs+Exec
Darmaillacq *et al.* [74]	FSecM	n.s.	SC	Prot	Ex	Abs
Darmaillacq [75]	FSecM	n.s.	TCS	Prot	Ex	Abs+Exec
El Kateb *et al.* [76]	SecP+FSecM	n.s.	TCS	Prot	Ex	Abs+Exec
El Maarabani [77]	FSecM	n.s.	SC	Prot	Ex	Exec
Elrakaiby *et al.* [78]	FSecM	n.s.	FB	Prot	Ex+Effe	n.s.
Falcone *et al.* [79]	FSecM	n.s.	SC	Prot	Ex	Exec
Faniyi *et al.* [80]	n.s.	A	TCS	Prot	Ex	Abs
Felderer *et al.* [81]	SecP	n.s.	SC+RB	Prot	Ex	Abs
Fourneret *et al.* [82]	SecP+FSecM	n.s.	SC	Prot	Ex	Abs
Gilliam *et al.* [83]	SecP+V	n.s.	n.s.	Prot	Ex	Abs
Hanna *et al.* [84]	SecP	n.s.	SC	Prot	Ex	Abs
Hanna *et al.* [85]	V	n.s.	FB+DC	Prot	Effi	Exec
He *et al.* [86]	n.s.	A+T	SC	Prot	Ex	Abs
Hong *et al.* [87]	FSecM	n.s.	SC	Prot	Ex	Abs
Hsu *et al.* [88]	FSecM	T	SC+DC	Prot	Ex	Abs+Exec
Hu & Ahn [89]	SecP+FSecM	n.s.	TCS	Prot	Ex	Abs
Hu *et al.* [90]	FSecM	n.s.	SC	Prot	Ex	Abs
Hu *et al.* [91]	SecP+FSecM	n.s.	SC	Prot	Ex	Abs+Exec
Huang & Wen [38]	n.s.	A	SC	n.s.	n.s.	n.s.
Hwang *et al.* [92]	FSecM	n.s.	SC+RS	Prot	Effe+Effi	Exec
Hwang *et al.* [93]	SecP	n.s.	SC	Prot	Ex	Abs+Exec
Hwang *et al.* [94]	FSecM	n.s.	SC+Random	Prot	Effe+Effi	Exec
Julliand *et al.* [95]	SecP+FSecM	n.s.	TCS	Prot	Ex	Abs
Julliand *et al.* [96]	FSecM	n.s.	SC+DC	Prod	Effe	Abs
Jürjens [97]	SecP	T	SC+RC	Prot	Ex	Abs
Jürjens & Wimmel [98]	FSecM	T	RC	Prot	Ex	Abs
Jürjens & Wimmel [99]	FSecM	n.s.	TCS	Prot	Ex	Abs
Kam and Dean [100]	n.s.	A	FB	Pre	Effe	Exec
Le Traon *et al.* [101]	V+FSecM	n.s.	SC+FB	Prot	Effe	Exec
Le Traon *et al.* [102]	V+FSecM	n.s.	SC+FB	Prot	Ex+Effe	Exec
Lebeau *et al.* [103]	FSecM	n.s.	SC+RC	Prot	Ex	Abs
Li *et al.* [104]	FSecM	n.s.	SC	Prot	Ex	Abs
Mallouli & Cavalli [105]	SecP	n.s.	TCS	Prot	Effe	Abs
Mallouli *et al.* [106]	SecP	n.s.	FB	Prot	Effe+Effi	Exec
Mallouli *et al.*[107]	SecP	n.s.	FB	Prot	Effe+Effi	Exec
Mammar *et al.* [108]	SecP	n.s.	SC+RC	Prot	Ex	Exec
Marback *et al.* [109]	n.s.	T	SC+DC	Prod	Ex+Effe	Exec
Martin & Xie [110]	V+FSecM	n.s.	SC	Prot	Effe+Effi	Exec
Martin *et al.* [111]	SecP+FSecM	n.s.	FB	Prot	Ex+Effe+Effi	Abs
Masood *et al.* [112]	FSecM	A	SC	Prot	Effe+Effi	Abs+Exec
Masson *et al.* [113]	SecP+V	n.s.	TCS	Prot	Ex	Abs
Morais *et al.* [114]	n.s.	A	SC	Prot	Ex	Abs+Exec
Morais *et al.* [115]	n.s.	A+T	SC	Prot	Ex+Effi	Exec

DOI: 10.1002/stvr

Table B.1. Continued.

	Filter criteria			Evidence criteria		
Paper	MSSec	SecME	TSC	MES	EM	EL
Mouelhi *et al.* [116]	V+FSecM	n.s.	FB	Prot	Ex	Exec
Mouelhi *et al.* [117]	FSecM	n.s.	FB	Prot	Ex	Exec
Mouelhi *et al.* [118]	V+FSecM	n.s.	SC+FB	Prot	Effe	Exec
Mouelhi *et al.* [119]	V+FSecM	n.s.	RB	Prot	Ex+Effe	Exec
Nguyen *et al.* [120]	V+FSecM	n.s.	FB	Prot	Ex	Exec
Noseevich & Petukhov [121]	SecP	n.s.	SC	Prot	Ex	Abs+Exec
Pari–Salas & Krishnan [122]	FSecM	n.s.	SC	Prot	Ex	Abs
Pretschner *et al.* [123]	FSecM	n.s.	DC	Prot	Ex+Effe	Exec
Rehkis *et al.* [124]	FSecM	n.s.	TCS	Prot	Ex	Abs
Rosenzweig *et al.* [125]	FSecM	n.s.	SC	Prot	Ex	Abs
Saidane & Guelfi [126]	SecP	A+T	SC	Prot	Ex	Exec
Salas *et al.* [127]	SecP	A	TCS	Prot	Ex	Abs
Salva & Zafimiharisoa [128]	V+FSecM	n.s.	RB	Prot	Ex+Effe+Effi	Abs+Exec
Savary *et al.* [129]	V	n.s.	FB	Prod	Effe+Effi	Abs
Schneider *et al.* [130]	n.s.	A	FB	Prot	Ex	Exec
Schneider *et al.* [131]	n.s.	A	SC	Prod	Ex	Abs+Exec
Senn *et al.* [132]	FSecM	n.s.	SC	Prod	Ex	Exec
Shahriar & Zulkernine [133]	n.s.	A	SC	Prod	Effe	Exec
Shu & Lee [134]	SecP	n.s.	RB	Pre	Effe	Abs+Exec
Shu *et al.* [135]	SecP	A	SC	Prod	Ex	Abs
Shu *et al.* [136]	V	n.s.	SC	Prod	Effe	Exec
Singh *et al.* [137]	SecP	n.s.	RS	Prot	Ex	Abs
Stepien *et al.* [138]	n.s.	A	RB	Prot	Ex	Abs+Exec
Tang *et al.* [139]	V	n.s.	RS	Prot	Ex	Exec
Traore & Aredo [140]	FSecM	n.s.	RC	Prot	Ex	Exec
Tuglular & Belli [141]	V+FSecM	n.s.	SC	Prot	Ex	Abs+Exec
Tuglular & Gercek [142]	FSecM	n.s.	SC	Prot	Ex	Abs+Exec
Tuglular & Gercek [143]	FSecM	n.s.	SC	Prod	Effe	Exec
Tuglular *et al.* [144]	FSecM	n.s.	SC	Prot	Ex	Exec
Turcotte *et al.* [145]	V	n.s.	FB	Prot	Ex	Abs+Exec
Wang *et al.* [146]	n.s.	A+T	SC+TCS	Prot	Ex	Abs
Wang *et al.* [147]	FSecM	n.s.	SC	Prot	Ex+Effe	Exec
Weber *et al.* [148]	FSecM	n.s.	SC	Prot	Ex+Effe	Abs
Wei *et al.* [149]	n.s.	A	SC	Prot	Ex	Abs+Exec
Whittle *et al.* [150]	n.s.	A	SC	Prot	Ex	Abs
Wimmel & Jürjens [151]	V	A+T	FB	Prot	Ex	Abs
Xu *et al.* [152]	FSecM	n.s.	SC	Prot	Ex+Effe	Abs+Exec
Xu *et al.* [153]	n.s.	T	SC	Prot	Effe	Exec
Xu *et al.* [154]	FSecM	n.s.	SC	Prot	Effe	Exec
Yan & Dan [155]	FSecM	n.s.	SC	Prot	Ex	Abs
Yan *et al.* [156]	FSecM	n.s.	TCS	Prot	Ex	Abs
Yang *et al.* [157]	FSecM	n.s.	TCS	Pre	Ex	Abs
Yang *et al.* [158]	n.s.	A	DC	Prot	Ex	Exec
Yang *et al.* [159]	n.s.	A	SC	Prod	Ex+Effe	Exec
Yu *et al.* [160]	FSecM	n.s.	DC	Prod	Effe	Exec
Zech [161]	n.s.	A	SC+AB	Prot	Ex	Abs+Exec
Zech *et al.* [162]	n.s.	A	AB	Prot	Ex	Abs+Exec
Zech *et al.* [163]	n.s.	A	SC+AB	Prot	Ex	Abs
Zhang *et al.* [164]	n.s.	A	FB	Prot	Ex	Exec
Zhou *et al.* [165]	SecP+FSecM	A+T	FB	Pre	Ex	Abs+Exec
Zulkernine *et al.* [166]	n.s.	A	TCS	Prod	Ex	Abs

'SecP', security properties; 'V', vulnerabilities; 'FSecM', functionality of security mechanisms; 'T', threat model; 'A', attack model; 'Prot', prototype; 'Pre', premature system; 'Prod', production system; 'Ex', example application; 'Effe', effectiveness measures; 'Effi', efficiency measures; 'Abs', abstract; 'Exec', executable; SC, structural coverage; DC, data coverage; RC, requirements coverage; TCS, explicit test case specifications; RS, random and stochastic; FB, fault-based; AB, assessment-based; RB, relationship-based; n.s., not specified.

DOI: 10.1002/stvr

ACKNOWLEDGEMENTS

This research was partially funded by the research projects QE LaB—Living Models for Open Systems (FFG 822740) and MOBSTECO (FWF P 26194-N15). In addition, the authors are grateful to Boban Celebic for his support to create the interactive online version of the classification table of the identified model-based security testing approaches.

REFERENCES

1. Schieferdecker I, Grossmann J, Schneider M. Model-based security testing. *Proceedings 7th Workshop on Model-Based Testing,* Tallinn, Estonia, 2012; 1–12.
2. ISO/IEC. ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.
3. Hafner M, Breu R. *Security Engineering for Service-Oriented Architectures*. Springer: Berlin Heidelberg, Germany, 2009.
4. ISO/IEC. ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model, 2001.
5. McGraw G. Software security. *IEEE Security & Privacy* 2004; **2**(2):80–83.
6. Dias-Neto AC, Travassos GH. A picture from the model-based testing area: concepts, techniques, and challenges. *Advances in Computers* 2010; **80**:45–120.
7. Schieferdecker I. Model-based testing. *IEEE Software* 2012; **29**(1):14–18.
8. Utting M, Pretschner A, Legeard B. A taxonomy of model-based testing approaches. *Software Testing Verification and Reliability* 2012; **22**(2):297–312.
9. Ammann P, Offutt J. *Introduction to Software Testing*. Cambridge University Press: Cambridge, UK, 2008.
10. Bourque P, Dupuis R (eds.) *Software Engineering Body of Knowledge (SWEBOK)*. IEEE: EUA, 2004. Available from: http://www.swebok.org/ [accessed 26 March 2013].
11. ISO/IEC. Information technology – open systems interconnection – conformance testing methodology and framework: International ISO/IEC multi–part standard No. 9646, 1994.
12. IEEE. *Standard Glossary of Software Engineering Terminology*. IEEE: New York, NY, USA, 1990.
13. Utting M, Legeard B. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2007.
14. Grieskamp W, Kicillof N, Stobie K, Braberman V. Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability* 2011; **21**(1):55–71.
15. Zander J, Schieferdecker I, Mosterman PJ. *Model-Based Testing for Embedded Systems*, Vol. 13. CRC Press: Boca Raton, FL, USA, 2012.
16. Anand S, Burke EK, Chen TY, Clark J, Cohen MB, Grieskamp W, Harman M, Harrold MJ, McMinn P. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software* 2013; **86**(8):1978–2001.
17. Hierons RM, Bogdanov K, Bowen JP, Cleaveland R, Derrick J, Dick J, Gheorghe M, Harman M, Kapoor K, Krause P, Simons AJH, Vilkomir S, Woodward MR, Zedan H. Using formal specifications to support testing. *ACM Computing Surveys (CSUR)* 2009; **41**(2):9.
18. Hartman A, Katara M, Olvovsky S. Choosing a test modeling language: a survey. In *Hardware and Software, Verification and Testing*. Springer: Berlin Heidelberg, 2007; 204–218.
19. Tian-yang G, Yin-sheng S, You-yuan F. Research on software security testing. *World Academy of Science, Engineering and Technology Issue* 2010; **69**:647–651.
20. Arkin B, Stender S, McGraw G. Software penetration testing. *IEEE Security & Privacy* 2005; **3**(1):84–87.
21. Potter B, McGraw G. Software security testing. *IEEE Security & Privacy* 2004; **2**(5):81–85.
22. Herzog P. The open source security testing methodology manual 3, 2010. Available from: http://www.isecom.org/mirror/OSSTMM.3.pdf [accessed 15 January 2013].
23. OWASP Foundation. OWASP testing guide v4. Available at https://www.owasp.org [accessed 11 April 2015].
24. Shahriar H, Zulkernine M. Automatic testing of program security vulnerabilities. *33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09*, Vol. 2, IEEE: New York, NY, USA, 2009; 550–555.
25. Bau J, Bursztein E, Gupta D, Mitchell J. State of the art: automated black-box web application vulnerability testing. *2010 IEEE Symposium on Security and Privacy (SP)*, IEEE: New York, NY, USA, 2010; 332–345.
26. Felderer M, Fourneret E. A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer* 2015; **17**(3):305–319.
27. Felderer M, Agreiter B, Zech P, Breu R. A classification for model-based security testing. *The Third International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011),* Barcelona, Spain, 2011; 109–114.
28. Lund MS, Solhaug B, Stølen K. *Model-Driven Risk Analysis: The CORAS Approach*. Springer: Berlin Heidelberg, Germany, 2011.
29. Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault tree handbook. *Technical Report*, DTIC Document, 1981.
30. MITRE. Common vulnerabilities and exposures. Available from: http://cve.mitre.org.

DOI: 10.1002/stvr

31. Pretschner A, Holling D, Eschbach R, Gemmar M. A generic fault model for quality assurance. *Proceedings of ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems*, 2013.
32. ISO/IEC. ISO/IEC 27005:2011 Information technology – Security techniques – Information security risk management, 2011.
33. Miller BP, Fredriksen L, So B. An empirical study of the reliability of unix utilities. *Communications of the ACM* 1990; **33**(12):32–44.
34. Felderer M, Schieferdecker I. A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer* 2014; **16**(5):559–568.
35. Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software* 2007; **80**(4):571–583.
36. Saracevic T. Evaluation of evaluation in information retrieval. *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM: New York, NY, USA, 1995; 138–146.
37. Kitchenham B. *Procedures for Performing Systematic Reviews*, Vol. 33, Keele, UK, Keele University, 2004.
38. Huang B, Wen Q. An automatic fuzz testing method designed for detecting vulnerabilities on all protocol. *2011 International Conference on Computer Science and Network Technology (ICCSNT)*, Vol. 2, IEEE: New York, NY, USA, 2011; 639–642.
39. Weyuker EJ, Jeng B. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering* 1991; **17**(7):703–711.
40. Gutjahr WJ. Partition testing vs. random testing: the influence of uncertainty. *IEEE Transactions on Software Engineering* 1999; **25**(5):661–674.
41. Inozemtseva L, Holmes R. Coverage is not strongly correlated with test suite effectiveness. *ICSE,* Hyderabad, India, 2014; 435–445.
42. Mockus A, Nagappan N, Dinh-Trong TT. Test coverage and post-verification defects: a multiple case study. *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, IEEE: Washington, DC, USA, 2009; 291–301.
43. Malaiya Y, Li M, Bieman J, Karcich R. Software reliability growth with test coverage. *IEEE Transactions on Reliability* 2002; **51**(4):420–426.
44. Heimdahl M, Whalen M, Rajan A, Staats M. On MC/DC and implementation structure: an empirical study. *IEEE/AIAA 27th Digital Avionics Systems Conference, 2008. DASC 2008,* Prague, Czech Republic, 2008; 5.B.3–1–5.B.3–13.
45. Dupuy A, Leveson N. An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software. *The 19th Digital Avionics Systems Conference, 2000. Proceedings. DASC*, Vol. 1, Philadelphia, PA, USA, 2000; 1B6/1–1B6/7 vol.1.
46. Godefroid P, Levin MY, Molnar DA. Sage: whitebox fuzzing for security testing. *Communications of the ACM* 2012; **55**(3):40–44.
47. Mlynarski M, Güldali B, Engels G, Weißleder S. Model-based testing: achievements and future challenges. *Advances in Computers* 2012; **86**:1–39.
48. Doupé A, Cova M, Vigna G. Why Johnny can't pentest: an analysis of black-box web vulnerability scanners. *DIMVA,* Berlin Heidelberg, Germany, 2010; 111–131.
49. Abassi R, Fatmi S. Using security policies in a network securing process. *2011 18th International Conference on Telecommunications (ICT)*, IEEE: New York, NY, USA, 2011; 416–421.
50. Abbassi R, El Fatmi SG. Towards a test cases generation method for security policies. *ICT '09 International Conference on Telecommunications, 2009*, IEEE: New York, NY, USA, 2009; 41–46.
51. Al-Azzani S, Bahsoon R. Using implied scenarios in security testing. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, ACM: New York, NY, USA, 2010; 15–21.
52. Al-Shaer E, El-Atawy A, Samak T. Automated pseudo-live testing of firewall configuration enforcement. *IEEE Journal on Selected Areas in Communications* 2009; **27**(3):302–314.
53. Allen WH, Dou C, Marin GA. A model-based approach to the security testing of network protocol implementations. *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, IEEE: New York, NY, USA, 2006; 1008–1015.
54. Antunes J, Neves NF. Using behavioral profiles to detect software flaws in network servers. *2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE)*, IEEE: New York, NY, USA, 2011; 1–10.
55. Armando A, Pellegrino G, Carbone R, Merlo A, Balzarotti D. From model-checking to automated testing of security protocols: bridging the gap. In *Tests and Proofs*. Springer: Berlin Heidelberg, Germany, 2012; 3–18.
56. Bartel A, Baudry B, Munoz F, Klein J, Mouelhi T, Le Traon Y. Model driven mutation applied to adaptative systems testing. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2011; 408–413.
57. Belhaouari H, Konopacki P, Laleau R, Frappier M. A design by contract approach to verify access control policies. *2012 17th International Conference on Engineering of Complex Computer Systems (ICECCS)*, IEEE: New York, NY, USA, 2012; 263–272.
58. Bertolino A, Daoudagh S, Lonetti F, Marchetti E. Automatic XACML requests generation for policy testing. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2012; 842–849.

DOI: 10.1002/stvr

59. Bertolino A, Daoudagh S, Lonetti F, Marchetti E, Martinelli F, Mori P. Testing of PolPA authorization systems. *2012 7th International Workshop on Automation of Software Test (AST)*, IEEE: New York, NY, USA, 2012; 8–14.
60. Beyer D, Chlipala AJ, Henzinger TA, Jhala R, Majumdar R. Generating tests from counterexamples. *Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society: New York, NY, USA, 2004; 326–335.
61. Blome A, Ochoa M, Li K, Peroli M, Dashti MT. Vera: a flexible model-based vulnerability testing tool. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2013; 471–478.
62. Bortolozzo M, Centenaro M, Focardi R, Steel G. Attacking and fixing PKCS# 11 security tokens. *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ACM: New York, NY, USA, 2010; 260–269.
63. Botella J, Bouquet F, Capuron JF, Lebeau F, Legeard B, Schadle F. Model-based testing of cryptographic components–lessons learned from experience. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2013; 192–201.
64. Bozic J, Wotawa F. Xss pattern for attack modeling in testing. *2013 8th International Workshop on Automation of Software Test (AST)*, IEEE: New York, NY, USA, 2013; 71–74.
65. Bracher S, Krishnan P. Enabling security testing from specification to code. In *Integrated Formal Methods*. Springer: Heidelberg, Germany, 2005; 150–166.
66. Brucker AD, Brugger L, Kearney P, Wolff B. Verified firewall policy transformations for test case generation. *2010 Third International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2010; 345–354.
67. Brucker AD, Wolff B. Test-sequence generation with hol-testgen with an application to firewall testing. In *Tests and Proofs*. Springer: Berlin Heidelberg, Germany, 2007; 149–168.
68. Brucker AD, Brügger L, Wolff B. Model-based firewall conformance testing. In *Testing of Software and Communicating Systems*. Springer: Berlin Heidelberg, Germany, 2008; 103–118.
69. Brucker AD, Brügger L, Kearney P, Wolff B. An approach to modular and testable security models of real-world health-care applications. *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, ACM: New York, NY, USA, 2011; 133–142.
70. Büchler M, Oudinet J, Pretschner A. Security mutants for property-based testing. In *Tests and Proofs*. Springer: Berlin Heidelberg, Germany, 2011; 69–77.
71. Buchler M, Oudinet J, Pretschner A. Semi-automatic security testing of web applications from a secure model. *2012 IEEE Sixth International Conference on Software Security and Reliability (SERE)*, IEEE: New York, NY, USA, 2012; 253–262.
72. Chen Z, Guo S, Fu D. A directed fuzzing based on the dynamic symbolic execution and extended program behavior model. *Proceedings of the 2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, IEEE Computer Society: New York, NY, USA, 2012; 1641–1644.
73. Dadeau F, Héam PC, Kheddam R. Mutation-based test generation from security protocols in HLPSL. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2011; 240–248.
74. Darmaillacq V, Fernandez JC, Groz R, Mounier L, Richier JL. Test generation for network security rules. In *Testing of Communicating Systems*. Springer: Berlin Heidelberg, Germany, 2006; 341–356.
75. Darmaillacq V, Richier J, Groz R. Test generation and execution for security rules in temporal logic. *ICSTW '08 IEEE International Conference on Software Testing Verification and Validation Workshop, 2008*, IEEE: New York, NY, USA, 2008; 252–259.
76. El Kateb D, El Rakaiby Y, Mouelhi T, Traon YL. Access control enforcement testing. *2013 8th International Workshop on Automation of Software Test (AST)*, IEEE: New York, NY, USA, 2013; 64–70.
77. El Maarabani M, Hwang I, Cavalli A. A formal approach for interoperability testing of security rules. *2010 Sixth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, IEEE: New York, NY, USA, 2010; 277–284.
78. Elrakaiby Y, Mouelhi T, Le Traon Y. Testing obligation policy enforcement using mutation analysis. *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2012; 673–680.
79. Falcone Y, Mounier L, Fernandez JC, Richier JL. j-POST: a Java Toolchain for property-oriented software testing. *Electronic Notes in Theoretical Computer Science* 2008; **220**(1):29–41.
80. Faniyi F, Bahsoon R, Evans A, Kazman R. Evaluating security properties of architectures in unpredictable environments: a case for cloud. *2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, IEEE: New York, NY, USA, 2011; 127–136.
81. Felderer M, Agreiter B, Breu R. Evolution of security requirements tests for service-centric systems. In *Engineering Secure Software and Systems*. Springer: Berlin Heidelberg, Germany, 2011; 181–194.
82. Fourneret E, Ochoa M, Bouquet F, Botella J, Jurjens J, Yousefi P. Model-based security verification and testing for smart-cards. *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, IEEE: New York, NY, USA, 2011; 272–279.
83. Gilliam DP, Powell JD, Kelly JC, Bishop M. Reducing software security risk through an integrated approach. *Proceedings 26th Annual NASA Goddard Software Engineering Workshop, 2001*, IEEE: New York, NY, USA, 2001; 36–42.

DOI: 10.1002/stvr

84. Hanna A, Ling HZ, Furlong J, Yang Z, Debbabi M. Targeting security vulnerabilities: from specification to detection (short paper). *QSIC '08 The Eighth International Conference on Quality Software, 2008*, IEEE: New York, NY, USA, 2008; 97–102.
85. Hanna A, Ling HZ, Yang X, Debbabi M. A synergy between static and dynamic analysis for the detection of software security vulnerabilities. In *On the Move to Meaningful Internet Systems: OTM 2009*. Springer: Berlin Heidelberg, Germany, 2009; 815–832.
86. He K, Feng Z, Li X. An attack scenario based approach for software security testing at design stage. *ISCSCT '08 International Symposium on Computer Science and Computational Technology, 2008*, Vol. 1, IEEE: New York, NY, USA, 2008; 782–787.
87. Yu H, Song H, Bin H, Yi Y. Using labeled transition system model in software access control politics testing. *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC),* Harbin, China, 2012; 680–683.
88. Hsu Y, Shu G, Lee D. A model-based approach to security flaw detection of network protocol implementations. *ICNP 2008 IEEE International Conference on Network protocols, 2008*, IEEE: New York, NY, USA, 2008; 114–123.
89. Hu H, Ahn G. Enabling verification and conformance testing for access control model. *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, ACM: New York, NY, USA, 2008; 195–204.
90. Hu VC, Martin E, Hwang J, Xie T. Conformance checking of access control policies specified in XACML. *COMPSAC 2007. 31st Annual International Computer software and Applications Conference, 2007*, Vol. 2, IEEE: New York, NY, USA, 2007; 275–280.
91. Hu VC, Kuhn DR, Xie T. Property verification for generic access control models. *EUC '08. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008*, Vol. 2, IEEE: New York, NY, USA, 2008; 243–250.
92. Hwang J, Xie T, Chen F, Liu AX. Systematic structural testing of firewall policies. *SRDS '08. IEEE Symposium on Reliable Distributed Systems, 2008*, IEEE: New York, NY, USA, 2008; 105–114.
93. Hwang J, Xie T, Hu V, Altunay M. ACPT: a tool for modeling and verifying access control policies. *2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*, IEEE: New York, NY, USA, 2010; 40–43.
94. Hwang J, Xie T, Chen F, Liu AX. Systematic structural testing of firewall policies. *IEEE Transactions on Network and Service Management* 2012; **9**(1):1–11.
95. Julliand J, Masson PA, Tissot R. Generating security tests in addition to functional tests. *Proceedings of the 3rd International Workshop on Automation of Software Test*, ACM: New York, NY, USA, 2008; 41–44.
96. Julliand J, Masson PA, Tissot R, Bué PC. Generating tests from B specifications and dynamic selection criteria. *Formal Aspects of Computing* 2011; **23**(1):3–19.
97. Jürjens J. Model-based security testing using UMLsec: a case study. *Electronic Notes in Theoretical Computer Science* 2008; **220**(1):93–104.
98. Jurjens J, Wimmel G. Formally testing fail-safety of electronic purse protocols. *Proceedings 16th Annual International Conference on Automated Software Engineering, 2001. (ASE 2001)*, IEEE: New York, NY, USA, 2001; 408–411.
99. Jürjens J, Wimmel G. Specification-based testing of firewalls. In *Perspectives of System Informatics*. Springer: Berlin Heidelberg, Germany, 2001; 308–316.
100. Kam BW, Dean TR. Linguistic security testing for text communication protocols. In *Testing–Practice and Research Techniques*. Springer: Berlin Heidelberg, Germany, 2010; 104–117.
101. Le Traon Y, Mouelhi T, Fleurey F, Baudry B. Language-specific vs. language-independent approaches: embedding semantics on a metamodel for testing and verifying access control policies. *2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2010; 72–79.
102. Traon YL, Mouelhi T, Baudry B. Testing security policies: going beyond functional testing. *ISSRE '07 The 18th IEEE International Symposium on Software Reliability, 2007*, IEEE: New York, NY, USA, 2007; 93–102.
103. Lebeau F, Legeard B, Peureux F, Vernotte A. Model-based vulnerability testing for web applications. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2013; 445–452.
104. Li K, Mounier L, Groz R. Test generation from security policies specified in Or-BAC. *31st Annual International Computer Software and Applications Conference, 2007. COMPSAC 2007*, Vol. 2, IEEE: New York, NY, USA, 2007; 255–260.
105. Mallouli W, Cavalli A. Testing security rules with decomposable activities. *HASE '07 10th IEEE High Assurance Systems Engineering Symposium, 2007*, IEEE: New York, NY, USA, 2007; 149–155.
106. Mallouli W, Lallali M, Morales G, Cavalli AR. Modeling and testing secure web-based systems: application to an industrial case study. *SITIS '08. IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008*, IEEE: New York, NY, USA, 2008; 128–136.
107. Mallouli W, Mammar A, Cavalli A. A formal framework to integrate timed security rules within a TEFSM-based system specification. *APSEC '09 Asia-Pacific Software engineering conference, 2009*, IEEE: New York, NY, USA, 2009; 489–496.
108. Mammar A, Mallouli W, Cavalli A. A systematic approach to integrate common timed security rules within a TEFSM-based system specification. *Information and Software Technology* 2012; **54**(1):87–98.

DOI: 10.1002/stvr

109. Marback A, Do H, He K, Kondamarri S, Xu D. A threat model-based approach to security testing. *Software: Practice and Experience* 2013; **43**(2):241–258.
110. Martin E, Xie T. A fault model and mutation testing of access control policies. *Proceedings of the 16th International Conference on World Wide Web*, ACM: New York, NY, USA, 2007; 667–676.
111. Martin E, Hwang J, Xie T, Hu V. Assessing quality of policy properties in verification of access control policies. *Annual Computer Security Applications Conference, 2008. ACSAC 2008*, IEEE: New York, NY, USA, 2008; 163–172.
112. Masood A, Ghafoor A, Mathur AP. Conformance testing of temporal role-based access control systems. *IEEE Transactions on Dependable and Secure Computing* 2010; **7**(2):144–158.
113. Masson PA, Julliand J, Plessis JC, Jaffuel E, Debois G. Automatic generation of model based tests for a class of security properties. *Proceedings of the 3rd International Workshop on Advances in Model-Based Testing*, ACM: New York, NY, USA, 2007; 12–22.
114. Morais A, Martins E, Cavalli A, Jimenez W. Security protocol testing using attack trees. *CSE '09 International Conference on Computational Science and Engineering, 2009*, Vol. 2, IEEE: New York, NY, USA, 2009; 690–697.
115. Morais A, Cavalli A, Martins E. A model-based attack injection approach for security validation. *Proceedings of the 4th International Conference on Security of Information and Networks*, ACM: New York, NY, USA, 2011; 103–110.
116. Mouelhi T, Le Traon Y, Baudry B. Mutation analysis for security tests qualification. *Testing: Academic and Industrial Conference Practice and Research Techniques-Mutation, 2007. TAICPART-MUTATION 2007*, IEEE: New York, NY, USA, 2007; 233–242.
117. Mouelhi T, Fleurey F, Baudry B. A generic metamodel for security policies mutation. *ICSTW '08. IEEE International Conference on Software Testing Verification and Validation Workshop, 2008*, IEEE: New York, NY, USA, 2008; 278–286.
118. Mouelhi T, Fleurey F, Baudry B, Le Traon Y. A model-based framework for security policy specification, deployment and testing. In *Model Driven Engineering Languages and Systems*. Springer: Berlin Heidelberg, Germany, 2008; 537–552.
119. Mouelhi T, Le Traon Y, Baudry B. Transforming and selecting functional test cases for security policy testing. *ICST '09 International Conference on Software Testing Verification and Validation, 2009*, IEEE: New York, NY, USA, 2009; 171–180.
120. Nguyen PH, Papadakis M, Rubab I. Testing delegation policy enforcement via mutation analysis. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2013; 34–42.
121. Noseevich G, Petukhov A. Detecting insufficient access control in web applications. *Syssec Workshop (SYSSEC), 2011 First*, IEEE: New York, NY, USA, 2011; 11–18.
122. Pari-Salas PA, Krishnan P. Testing privacy policies using models. *Sixth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2008)*, IEEE: Cape Town, South Africa, 2008; 117–126.
123. Pretschner A, Mouelhi T, Le Traon Y. Model-based tests for access control policies. *2008 1st International Conference on Software Testing, Verification, and Validation*, IEEE: New York, NY, USA, 2008; 338–347.
124. Rekhis S, Bennour B, Boudriga N. Validation of security solutions for communication networks: a policy-based approach. *2011 10th IEEE International Symposium on Network Computing and Applications (NCA)*, IEEE: New York, NY, USA, 2011; 115–122.
125. Rosenzweig D, Runje D, Schulte W. Model-based testing of cryptographic protocols. In *Trustworthy Global Computing*. Springer: Berlin Heidelberg, Germany, 2005; 33–60.
126. Saidane A, Guelfi N. Towards improving security testability of AADL architecture models. *2011 5th International Conference on Network and System Security (NSS)*, IEEE: New York, NY, USA, 2011; 353–357.
127. Salas PAP, Krishnan P, Ross KJ. Model-based security vulnerability testing. *18th Australian Software Engineering Conference, 2007. ASWEC 2007*, IEEE: New York, NY, USA, 2007; 284–296.
128. Salva S, Zafimiharisoa SR. Data vulnerability detection by security testing for android applications. *Information Security for South Africa, 2013*, IEEE: New York, NY, USA, 2013; 1–8.
129. Savary A, Frappier M, Lanet JL. Detecting vulnerabilities in Java-card bytecode verifiers using model-based testing. In *Integrated Formal Methods*. Springer, 2013; 223–237.
130. Schneider M, Großmann J, Tcholtchev N, Schieferdecker I, Pietschker A. *Behavioral Fuzzing Operators for UML Sequence Diagrams*. Springer: Berlin Heidelberg, Germany, 2013.
131. Schneider M, Grossmann J, Schieferdecker I, Pietschker A. Online model-based behavioral fuzzing. *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2013; 469–475.
132. Senn D, Basin D, Caronni G. Firewall conformance testing. In *Testing of Communicating Systems*. Springer: Berlin Heidelberg, Germany, 2005; 226–241.
133. Shahriar H, Zulkernine M. PhishTester: automatic testing of phishing attacks. *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, IEEE: New York, NY, USA, 2010; 198–207.
134. Shu G, Lee D. Testing security properties of protocol implementations–a machine learning based approach. *ICDCS '07 27th International Conference on Distributed Computing Systems, 2007*, IEEE: New York, NY, USA, 2007; 25–25.

DOI: 10.1002/stvr

135. Shu G, Lee D. Message confidentiality testing of security protocols–passive monitoring and active checking. In *Testing of Communicating Systems*. Springer: Berlin Heidelberg, Germany, 2006; 357–372.
136. Shu G, Hsu Y, Lee D. Detecting communication protocol security flaws by formal fuzz testing and machine learning. In *Formal Techniques for Networked and Distributed Systems–FORTE 2008*. Springer: Berlin Heidelberg, Germany, 2008; 299–304.
137. Singh S, Lyons J, Nicol DM. Fast model-based penetration testing. *Proceedings of the 36th Conference on Winter Simulation*, IEEE: New York, NY, USA, 2004; 309–317.
138. Stepien B, Peyton L, Xiong P. Using TTCN-3 as a modeling language for web penetration testing. *2012 IEEE International Conference on Industrial Technology (ICIT)*, IEEE: New York, NY, USA, 2012; 674–681.
139. Tang W, Sui AF, Schmid W. A model guided security vulnerability discovery approach for network protocol implementation. *2011 IEEE 13th International Conference on Communication Technology (ICCT),* Jinan, China, 2011.
140. Traore I, Aredo DB. Enhancing structured review with model-based verification. *IEEE Transactions on Software Engineering* 2004; **30**(11):736–753.
141. Tuglular T, Belli F. Protocol-based testing of firewalls. *2009 Fourth South-East European Workshop on Formal Methods (SEEFM)*, IEEE: New York, NY, USA, 2009; 53–59.
142. Tuglular T, Gercek G. Feedback control based test case instantiation for firewall testing. *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, IEEE: New York, NY, USA, 2010; 202–207.
143. Tuglular T, Gercek G. Mutation-based evaluation of weighted test case selection for firewall testing. *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, IEEE: New York, NY, USA, 2011; 157–164.
144. Tuglular T, Kaya O, Muftuoglu CA, Belli F. Directed acyclic graph modeling of security policies for firewall testing. *Third IEEE International Conference on Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009*, IEEE: New York, NY, USA, 2009; 393–398.
145. Turcotte Y, Tal O, Knight S, Dean T. Security vulnerabilities assessment of the X. 509 protocol by syntax-based testing. *2004 IEEE Military Communications Conference, 2004. MILCOM 2004*, Vol. 3, IEEE: New York, NY, USA, 2004; 1572–1578.
146. Wang L, Wong E, Xu D. A threat model driven approach for security testing. *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, IEEE Computer Society: New York, NY, USA, 2007; 10.
147. Wang W, Zeng Q, Mathur AP. A security assurance framework combining formal verification and security functional testing. *2012 12th International Conference on Quality Software (QSIC)*, IEEE: New York, NY, USA, 2012; 136–139.
148. Weber S, Paradkar A, McIntosh S, Toll D, Karger PA, Kaplan M, Palmer ER. The feasibility of automated feedback-directed specification-based test generation: a case study of a high-assurance operating system. *19th International Symposium on Software Reliability Engineering, 2008. ISSRE 2008*, IEEE: New York, NY, USA, 2008; 229–238.
149. Wei T, Ju-Feng Y, Jing X, Guan-Nan S. Attack model based penetration test for SQL injection vulnerability. *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, IEEE: New York, NY, USA, 2012; 589–594.
150. Whittle J, Wijesekera D, Hartong M. Executable misuse cases for modeling security concerns. *ICSE '08. ACM/IEEE 30th International Conference on Software engineering, 2008*, IEEE: New York, NY, USA, 2008; 121–130.
151. Wimmel G, Jürjens J. Specification-based test generation for security-critical systems using mutations. In *Formal Methods and Software Engineering*. Springer: Berlin Heidelberg, Germany, 2002; 471–482.
152. Xu D, Sanford M, Liu Z, Emry M, Brockmueller B, Johnson S, To M. Testing access control and obligation policies. *2013 International Conference on Computing, Networking and Communications (ICNC)*, IEEE: New York, NY, USA, 2013; 540–544.
153. Xu D, Tu M, Sanford M, Thomas L, Woodraska D, Xu W. Automated security test generation with formal threat models. *IEEE Transactions on Dependable and Secure Computing* 2012; **9**(4):526–540.
154. Xu D, Thomas L, Kent M, Mouelhi T, Le Traon Y. A model-based approach to automated testing of access control policies. *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, ACM: New York, NY, USA, 2012; 209–218.
155. Yan C, Dan W. A scenario driven approach for security policy testing based on model checking. *International Conference on Information Engineering and Computer Science, 2009. ICIECS 2009*, IEEE: New York, NY, USA, 2009; 1–4.
156. Yan M, Xuexiong Y, Yuefei Z, Guoliang S. A method of TTCN test case generation based on TP description. *WCSE '09 Second International Workshop on Computer Science and Engineering, 2009*, Vol. 1, IEEE: New York, NY, USA, 2009; 255–258.
157. Yang Y, Chen Y, Xia M, Ma J. An automated mechanism of security test on network protocols. *IAS '09 Fifth International Conference on Information Assurance and Security, 2009*, Vol. 1, IEEE: New York, NY, USA, 2009; 503–506.
158. Yang Y, Zhang H, Pan M, Yang J, He F, Li Z. A model-based fuzz framework to the security testing of TCG software stack implementations. *MINES '09 International Conference on Multimedia Information Networking and Security, 2009*, Vol. 1, IEEE: New York, NY, USA, 2009; 149–152.

DOI: 10.1002/stvr

159. Yang D, Zhang Y, Liu Q. Blendfuzz: a model-based framework for fuzz testing programs with grammatical inputs. *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM)*, IEEE: New York, NY, USA, 2012; 1070–1076.
160. Yu H, Xiao-Ming L, Song H, Chang-You Z. Data oriented software security testing. *Proceedings of the 2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, IEEE Computer Society: New York, NY, USA, 2012; 676–679.
161. Zech P. Risk-based security testing in cloud computing environments. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*, IEEE: New York, NY, USA, 2011; 411–414.
162. Zech P, Felderer M, Breu R. Towards risk-driven security testing of service centric systems. *2012 12th International Conference on Quality Software (QSIC)*, IEEE: New York, NY, USA, 2012; 140–143.
163. Zech P, Felderer M, Breu R. Towards a model based security testing approach of cloud computing environments. *2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C)*, IEEE: New York, NY, USA, 2012; 47–56.
164. Zhang Z, Wen QY, Tang W. An efficient mutation-based fuzz testing approach for detecting flaws of network protocol. *2012 International Conference on Computer Science & Service System (CSSS)*, IEEE: New York, NY, USA, 2012; 814–817.
165. Zhou L, Yin X, Wang Z. Protocol security testing with SPIN and TTCN-3. *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE: New York, NY, USA, 2011; 511–519.
166. Zulkernine M, Raihan MF, Uddin MG. Towards model-based automatic testing of attack scenarios. In *Computer Safety, Reliability, and Security*. Springer: Berlin Heidelberg, Germany, 2009; 229–242.

DOI: 10.1002/stvr