A Classification for Model-Based Security Testing

Michael Felderer, Berthold Agreiter, Philipp Zech and Ruth Breu
Institute for Computer Science
University of Innsbruck
Technikerstr. 21a, A-6020 Innsbruck, Austria
{michael.felderer, berthold.agreiter, philipp.zech, ruth.breu}@uibk.ac.at

***Abstract*—Security testing defines tests for security requirements of software. Security requirements are non-functional, and thus require a different way of testing compared to functional requirements. Model-based testing applies model-based design for modeling test artifacts or the automation of test activities. Although model-based testing techniques improve security testing, these two testing activities have rarely been combined systematically. Like functional system models improve functional testing, risk models can improve security testing. This paper first gives an overview of existing security testing approaches, and based on that, develops a novel classification for model-based security tests along the two dimensions risk and automated test generation. The classification allows for understanding which areas of model-based security testing are already well-covered by research and practice, and furthermore, can serve as a guideline for deciding which testing approach fits specific circumstances. Based on the classification, we identify tasks for interesting future research.**

***Keywords*-Secure Systems, Verification and Testing, Security Testing, Model-based Testing**

I. INTRODUCTION

Security testing means to test for security requirements of software. By the increasing use of software in more and more areas of our daily life, also the amount of sensitive data which is processed automatically, e.g., in electronic health or e-government applications, increases. Consequently, the adherence to security requirements, which aim to protect sensitive data and the systems processing that data, is constantly gaining importance. Several classifications of security requirements can be found in literature, e.g., [1]–[3]. The most prominent list of security requirements (cf. [3]) distinguishes six types of security requirements:

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

Security testing aims at checking whether these requirements are satisfied under various conditions. Due to the openness of modern service-oriented systems, security testing has gained much interest in the last years [4] and has become a vast field of research.

Model-based testing (MBT) applies model-based design for the *modeling of test artefacts*, or the *automation of test activities* [5]. MBT supports the early definition and automatic validation of tests on the abstract model level. Most of today's model-based testing approaches consider the automated generation of test cases from a functional system description. MBT itself is well-covered in literature, and many tools are already on the market applying model-based approaches [6]. Furthermore, also partial test models are often encountered in practice, where domain expertise by a test engineer is needed for designing tests. Although model-based testing increases the level of abstraction in different aspects, supports a systematic, model-based, a-priori security test design, and reduces the required expertise for security testing, it is not widely used for testing security requirements today. So far, *model-based security testing* (MBST), i.e., model-based testing of security requirements, is more or less only used for testing access control policies in academia (see Section II on security testing approaches).

MBT re-uses functional system knowledge which is provided by models, so that the test engineer can abstract from many aspects in that respect. However, for testing security requirements, the test engineer further needs security knowledge for being reasonably able to design tests. Since security is tightly coupled with risk, risk models can potentially fill this gap. Based on an overview of existing classical and model-based security testing approaches, the contribution at hand defines a novel classification of model-based security testing approaches, and systematically identifies interesting future research directions to promote model-based security testing. We classify model-based security testing approaches along the two dimensions *risk* and *automated test generation*. For each category of security tests, we describe typical approaches and identify fields which offer the potential to put the discipline of security testing forward.

 ISBN: 978-1-61208-168-7

Since we want to classify security testing approaches in this contribution, we first detail what *security* is. Security is a *non-functional* property, and defines how a system is supposed to be, in contrast to what a system is supposed to do. Security threats are caused by faults and flaws. *Faults* may lead to *failures* which harm security requirements, and *flaws* are security problems which may lead to *vulnerabilities*. Security can be classified into three main levels. Each level exhibits its own security threats, but also offers the corresponding security requirement to deal with these threats. The three security levels network security, operating system security, and application security (cf. [7]) are characterized as follows:

- *Network Security* involves tackling threats which target the network. The main threats are (distributed) denial-of-service, network intrusion, or attacks during message transport (cf. [8]).
- *Operating System Security* is related to the basic services of operating systems and includes protection against all sorts of malware (virus, worm, spyware, etc.). The protection mechanisms involve antivirus, anti-malware, operating system level access control mechanisms, firewalls, etc. (cf. [9])
- *Application Security* deals with the threats targeting a specific application. It includes unauthorized access, information theft, and misuse of the application. The security mechanisms include access control mechanisms and policies, application level encryption/decryption, etc.

All these security levels can be subject to software tests. In this paper, we focus on application security and disregard the network and operating system, nevertheless this does not limit the applicability of the classification we introduce later.

The remainder of this paper is organized as follows: In Section II, we give an overview of existing security testing approaches. In Section III, we define a novel classification of model-based security testing with the dimensions risk and automated test generation. Finally, in Section IV, we summarize and sketch future work in the area of model-based security testing arising from our classification.

II. Security Testing Approaches

In this section, we give a representative overview of actual security testing approaches to motivate model-based security testing that considers risks and partial test generation. Security testing is often fundamentally different from traditional software testing because it emphasizes what an application should not do rather than what it should do. This fact was also pointed out by Alexander in [10], where the author distinguishes between positive and negative requirements modeled as use cases and misuse cases.

For testing positive security requirements, i.e., functional security properties that are defined in the requirements specification, classical functional testing techniques can be applied (Michael and Radosevich [11] provide a detailed listing of functional testing techniques for testing positive security requirements, e.g., equivalence testing or decision tables). Testing positive security requirements can be performed by the traditional test organization [12].

Negative security requirements express what a system should not do, respectively what should not happen. The set of negative requirements is therefore infinite on principle, and this makes it impossible to achieve complete test coverage. A promising way to overcome this problem is the derivation of tests based on a risk-analysis [11]. Due to this fact, risk-based testing (RBT) techniques [13] are highly relevant for security testing [14]. Based on a threat model, or based on abuse cases [15], vulnerabilities can be identified and prioritized relying on a risk analysis.

Tests can be designed in a classical operational way or, very frequently, as penetration tests which attempt to compromise the security of a system [16] by acting like an attacker trying to penetrate the system and exploit its vulnerabilities. Specifically, it tests missing functionality or side-effects of the system. Often also the environment of a system is the target of attacks instead of the system itself (e.g., exploiting an unpatched operating system). Furthermore, there are several penetration testing strategies [17], e.g., internal, external, or blind testing strategy, and penetration testing tools available [18], e.g., port or vulnerability scanners. Moreover, several standards for penetration testing exist, among which the Open Source Security Testing Methodology Manual (OSSTMM) [19] is the most prominent one. The OSSTMM methodology covers the whole process of risk assessment involved in a penetration test, from initial requirements analysis to report generation.

Besides penetration testing, another well-known approach to functional security testing is fuzzing [20]. The very basic idea behind a fuzzer is to test a protocol implementation on possible security flaws due to improper handling of malicious input. However, as Takanen et al. show in their book [20], fuzzing may also be used for testing other types of software in terms of security. Yet, what makes fuzzers difficult to use is the fact that a fuzzer by design cannot be general-purpose. Hence, for each new software to test, a specific fuzzer has to be implemented from scratch, which in fact is a challenging task. For instance, Taber et al. [21] present a fuzzer dedicated to security testing SIP-based VOIP applications. Yet, fuzzers still suffer from their randomness, put another way, testing with a fuzzer lacks all aspects of a structural approach. However, as done in [22], combining the idea of fuzzing with the concept of model–based testing, allows for systematic and automated testing of software applications.

On a more abstract level, model-based testing approaches have been applied for testing access control policies.

In [23], the authors describe a model–based testing approach for checking whether access control policies are properly enforced by the system under test (SUT). The functional model is written in the B language [24] and used for the security test generation from so called *test purposes*. Test purposes are defined as regular expressions and describe a general sequence

of operation calls to induce a certain situation on the SUT. The approach aims at the automated generation of test cases from the SUT.

A similar approach is used in [25] for testing smart cards where the access control rules are defined by test purposes. Another model-based approach for testing role-based access control policies is presented in [26]. The creation of test targets (actual access requests) is based on three different strategies: (1) only taking into account roles and permissions, (2) considering all rules in a policy and (3) completely at random. The tests target the policy decision point to check whether its decisions are correct or not.

Usually, models that are used for test generation are supposed to be correct and attack-resistant. In an attack-driven approach, a common practice is to modify the model such that it contains errors that can be revealed by attacks. The model mutation makes it possible to simulate attack scenarios on the model such that the response provides an observable answer to the attack. If a test representing an attack is executed against the SUT and reacts as expected from the modified model, then a vulnerability has been identified. In the mutation-based testing approach of DeMillo et al. [27], modified models are used in the standard test generation process. The mutation guides the test generation with a focus on the introduced errors.

Traon et al. [28] present a set of security mutations in access control policies used to drive the test generation. The generated tests are used to request access to secure data. The success of the access, allows to conclude the presence of failures in the SUT.

Jurjens [29] uses fault-injection techniques to introduce security faults in UMLsec models [30], and model-checkers are then used to build traces leading to the faults. The traces are then used as test cases for the system.

The model-based testing approaches mentioned above focus on the test generation from *complete* models where all security tests are derived fully automated from a formal model. Such models are costly to create and hard to maintain, and therefore only rarely applied in practice. Thus, we also consider *partial* security test generation where some security tests are generated automatically from a security model, and others are added manually. In classical approaches no test models are created, and the automatic test generation is *missing*.

Fig. 1 shows and compares the three degrees of automatic security test generation from software (components), i.e., complete, partial, and missing.

In the case of complete test generation, all tests are generated from a formal security model which is depicted as graph in Fig. 1. In the case of partial test generation, tests are generated from a formal security model and manually which is depicted by a graph and a cloud as source of test generation. The graph and the cloud are linked to show that the design of the test model and the manual test definition informally influence each other. Finally, if automatic test generation is missing, then all tests are generated manually

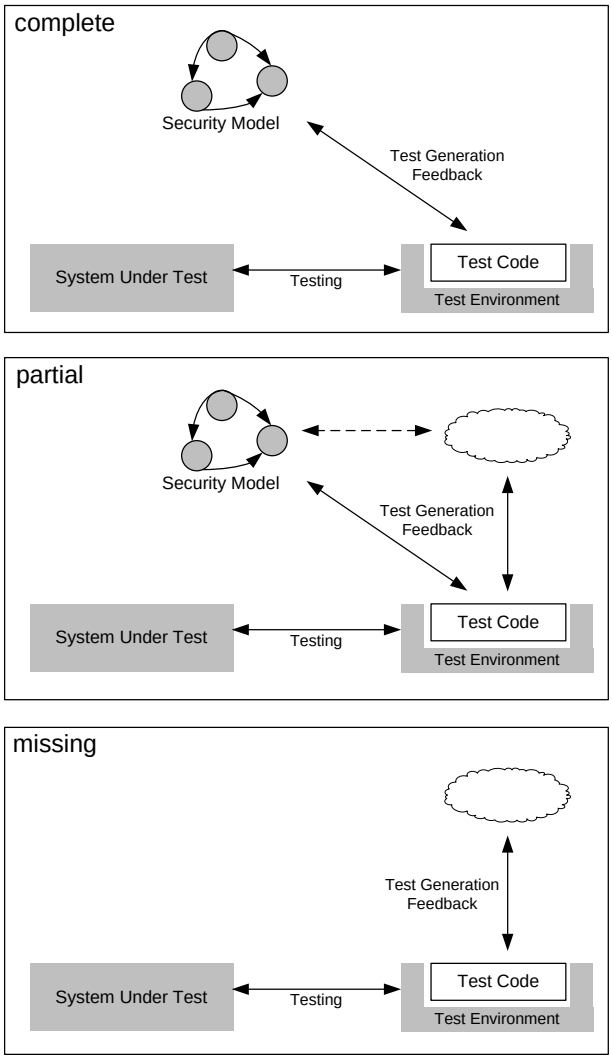

Fig. 1. Complete, Partial, and Missing Security Model

which is depicted by a cloud only. Assuming that the security model is provided, the skills needed by a test designer increase from complete to missing.

Partial security test generation is very promising because it does not require complete models and integrates the expertise of security testers. But partial security test generation is hardly employed in practice. Additionally, actual MBST approaches do not consider risk values for the test case generation. To increase the acceptance of MBST approaches in industry, and to consider existing risk-based security testing approaches, we extend the view on model-based security testing. In a novel classification of MBST approaches, we incorporate risks and partial test generation. Today, especially testing negative requirements strongly depends on expertise and experience [12]. We intend to lower the required level of expertise needed for security testing by defining new approaches based on our classification.

 ISBN: 978-1-61208-168-7

III. Model-Based Security Testing Classification

As motivated before, in this section, we classify model-based security tests along the two dimensions *automated test generation* and *risk*. The automated test generation dimension describes how much of the system and the security requirements is captured by formal models. The fewer information on system and security requirements is available, the more individual knowledge by the test engineer is needed to specify meaningful test cases. Fully automated test generation is only possible with formal and complete models which are typically not available. However, a possibility to support the test engineer in test design is the consideration of risk models, on the one hand, for deriving test cases, and, on the other hand, for prioritizing test execution. This results in the classification of model-based security testing approaches shown in Figure 2.

The degree of "automated security test generation" can be *complete*, i.e., all security tests are derived from a model, *partial*, i.e., some security tests are generated from a model, and others are added manually, or *missing*, i.e., all tests are defined manually. Assuming that the security model is provided, needed test design skills increase from *complete* to *missing* test generation. The more complete a security model is, the more functional are the security requirements. Note that the degree of automated test generation may vary for different system components.

The dimension "risk" can have the values *integrated* into the model or *not integrated* into the model. Note that the boundaries between the different characteristics are fuzzy. In the following sections we provide examples for each category.

A. Individual Knowledge

If a model does not support automated test generation and does not consider risk values, then individual knowledge determines the design of security tests and the selection of appropriate functional [11] and penetration testing techniques [17]. The efficiency of such techniques heavily relies on the experience and the specific domain knowledge of the test designers. However, employing the idea of pattern based testing allows to, although no better than very rudimentary, tweak this otherwise quite random testing techniques in terms of testing process itself.

B. Adapted RBT

If automated test generation is not possible because of a missing security model but risks have been evaluated, then a prioritization of tests is possible. Most actual risk-based testing approaches, e.g., [13] assign risk values to design elements and can therefore be categorized as risk enhanced model-based testing approaches without automated test generation. Wysopal et al. [14] define an adapted risk-based security testing approach based on threat models. The risk values in case of adapted RBT approaches are often only defined as additional informal artifacts, e.g., in spreadsheets. But the risk assessment itself is typically systematized in such approaches [31].

C. Scenario-Based MBT

The partial automated generation is supported e.g., by the Telling TestStories approach [32]. Telling TestStories supports the automated generation but also the manual definition of so called test stories, i.e., test scenarios modeled as UML activity diagrams or UML sequence diagrams plus assigned test data in a tabular form. Telling TestStories has been applied to model security tests of service-centric systems [33]. However, the idea of scenario-based MBT is not only restricted on UML diagrams, also the application of control flow graphs allows to derive scenarios by employing graph coverage criteria, geared towards covering high security sensitive execution traces. Obviously, following such an approach the graph replaces the classical perception of a software model, based on notions of UML. Tuglular et al. [34] suggest an approach to firewall testing based on Event Sequence Graphs, used to directly generate test cases.

D. Risk Enhanced Scenario-Based MBT

The Telling TestStories approach mentioned before is scalable for integrating risks into the test model to define a risk enhanced partial test generation process. We consider the integration of risks into Telling TestStories as future work.

E. Adapted MBT

The adapted model-based testing for security models is supported by the automated test generation approaches for access control policies discussed in Section II, e.g., [23], [26], [27], [29], which automatically generate test cases but do not consider risk values.

However, the application of adapted MBT in the field is quite rare, as often a *complete* model, as required by MBT approaches, does not exist.

F. Automated RB Security Test Generation

RiteDAP [35] is a model-based approach to risk-based system testing that employs annotated UML activity diagrams for automated test case generation and prioritization. RiteDAP is therefore an approach that manages risks on the model level and supports the complete automated test generation. But it does so far not consider the automated test generation of security tests.

The approach, suggested by Zech [36] actually attempts to go one step further as RiteDAP by additionally supporting the automated generation of test cases. Based on attack patterns and threat profiles, enabling the generation of a risk model from a system model, test case are generated out of misuse cases, automatically derived from the aforementioned risk model. Hence, this approach can be considered as an approach to automated RB security test generation, based on a tailored security model. Again, such an approach, building on a *complete* model currently lacks acceptance in the field, as most of the time, as already mentioned before, a complete

 ISBN: 978-1-61208-168-7

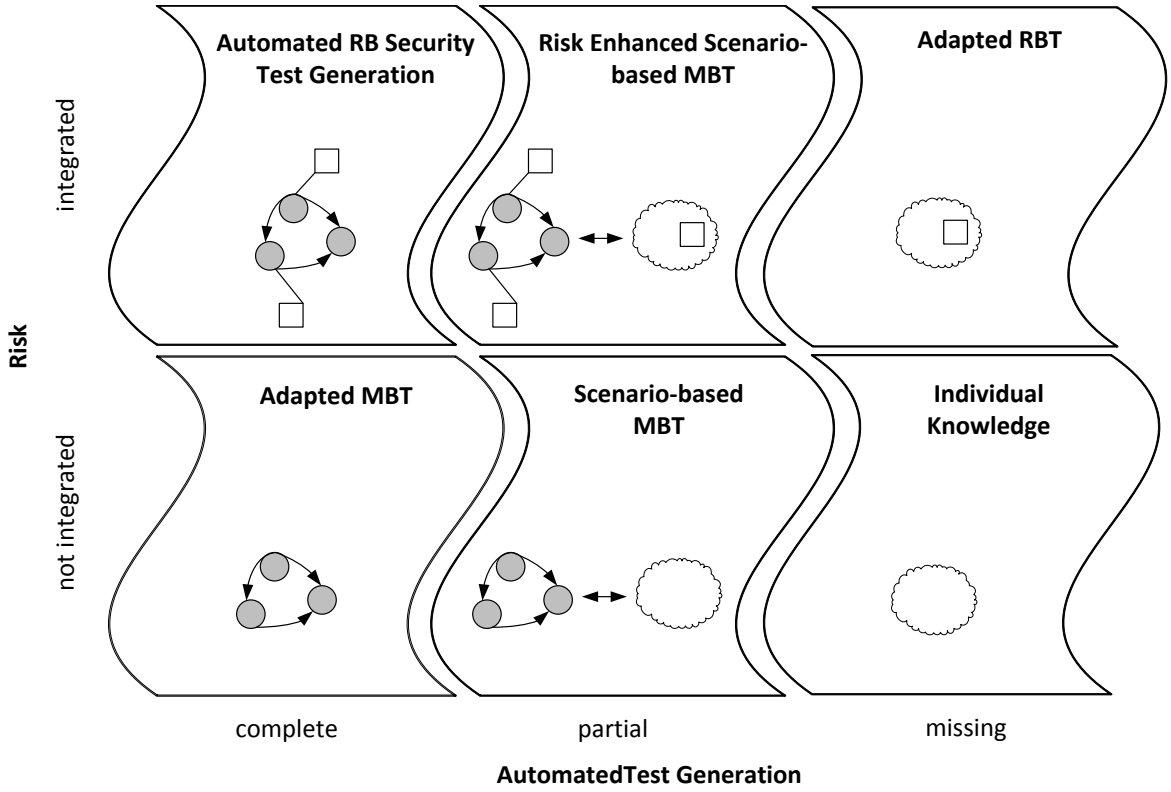

Fig. 2. Model-Based Security Testing Classification

model, capable of being employed for test generation, simply does no exist.

According to the presented categorization there exist only very few model-based security testing approaches that integrate complete or partial automated test generation and risks. Both, a risk enhanced scenario-based MBT approach and an automated RB security test generation approach contribute to utilize the full potential of model-based security testing of positive and negative security requirements. We therefore motivate the further investigation and development of risk-aware and automated MBST approaches based on existing frameworks such as Telling TestStories or RiteDAP.

IV. Concluding Remarks

In this contribution, we have first explained the benefits of model-based testing, which is already widely used today. To employ model-based approaches for testing security requirements, security models are needed which describe how a system is supposed to behave. The security model lowers the level of required security expertise of the test designer due to several reasons.

First, such a security test model improves the test designer's understanding of the software's security aspects which results in more efficient test cases. By using models, the level of abstraction is raised which enables more people to design tests. Finally, the model can be employed to automatically generate test cases. Additionally, security models are often created in conjunction with a risk analysis. This risk information can on the one hand be used for deriving test cases, and on the other hand for prioritizing test execution. We have developed a categorization along these two dimensions and provided examples for each category.

Based on an overview of security testing approaches, we have motivated the increasing importance of model-based security testing. We especially pointed out that the integration of risks into test models has not been investigated in detail, but has high potential for practical application in security testing.

According to our classification and existing security testing approaches we identify the following future research tasks in the area of model-based security testing:

- Development of a risk enhanced scenario- and model-based security testing approach on top of a scenario-based MBT approach.
- Development of an automated risk-based security test generation approach grounded on a model-based approach to risk-based testing.
- Integration of manually, semi-automatically and automatically determined metrics for the assessment of risks values in the risk model which is the basis for the integration of risks into a test model.
- Application and evaluation of model-based security testing for service-centric systems such as service-oriented architectures or cloud applications.

ACKNOWLEDGEMENTS

This work is sponsored by the SecureChange project (EU grant number ICT-FET-231101), the MATE project (FWF project number P17380), and QE LaB - Living Models for Open Systems (FFG 822740).

REFERENCES

[1] D. Firesmith, "Engineering Security Requirements," *Journal of Object Technology*, vol. 2, no. 1, 2003.

[2] Common Criteria Recognition Arrangement, "Common Criteria for Information Technology Security Evaluation," 2009, http://www.commoncriteriaportal.org/thecc.html [accessed: April 7, 2011].

[3] Committee on National Security Systems, "National Information Assurance Glossary," CNSS, Tech. Rep., 2006.

[4] G. Canfora and M. D. Penta, "Testing Services and Service-Centric Systems: Challenges and Opportunities," *IT Professional*, vol. 8, pp. 10–17, 2006.

[5] T. Roßner, C. Brandes, H. Götz, and M. Winter, *Basiswissen Modellbasierter Test*. dpunkt Verlag, 2010, in German.

[6] H. Götz, M. Nickolaus, T. Roßner, and K. Salomon, *iX Studie Modellbasiertes Testen*. Heise Zeitschriften Verlag, 2009, in German.

[7] T. Mouelhi, "Testing and Modeling Security Mechanisms in Web Applications," Ph.D. dissertation, RSM, University of Rennes, 2010.

[8] W. Stallings, *Network Security Essentials*. Prentice Hall, 2002.

[9] T. Jaeger, *Operating System Security*. Morgan & Claypool, 2008.

[10] I. Alexander, "Misuse cases: Use cases with hostile intent," *Software, IEEE*, vol. 20, no. 1, pp. 58–66, 2002.

[11] M. C. C. and R. Will, "Risk–based and Functional Security Testing," Cigital, Tech. Rep., 2009, https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing/255-BSI.pdf [accessed: April 7, 2011].

[12] B. Potter and G. McGraw, "Software Security Testing," *IEEE Security & Privacy*, 2004.

[13] S. Amland, "Risk-based testing: : Risk analysis fundamentals and metrics for software testing including a financial application case study," *Journal of Systems and Software*, vol. 53, no. 3, pp. 287–295, 2000.

[14] C. Wysopal, L. Nelson, D. D. Zovi, and E. Dustin, *The Art of Software Security Testing*. Addision–Wesley, 2006.

[15] D. Firesmith, "Security use cases," *Journal of Object Technology*, vol. 2, no. 1, pp. 53–64, 2003.

[16] M. Bishop, "About Penetration Testing," *IEEE Security & Privacy*, vol. 5, no. 6, 2007.

[17] SearchNetworking.com, "Penetration Testing Strategies," 2011, http://searchnetworking.techtarget.com/tutorial/Penetration-testing-strategies [accessed: April 7, 2011].

[18] K. van Wyk, "Penetration Testing Tools," 2008, available at https://buildsecurityin.us-cert.gov/bsi/articles/tools/penetration/657-BSI.pdf [accessed: April 7, 2011].

[19] P. Herzog, *The Open Source Security Testing Methodology Manual 3*, 2010, http://www.isecom.org/mirror/OSSTMM.3.pdf.

[20] A. Takanen, J. DeMott, and C. Miller, *Fuzzing for Software Security Testing and Quality Assurance*, 1st ed. Norwood, MA, USA: Artech House, Inc., 2008.

[21] S. Taber, C. Schanes, C. Hlauschek, F. Fankhauser, and T. Grechenig, "Automated Security Test Approach for SIP-based VoIP Softphones," *Advances in System Testing and Validation Lifecycle, International Conference on*, vol. 0, pp. 114–119, 2010.

[22] Y. Yang, H. Zhang, M. Pan, J. Yang, F. He, and Z. Li, "A model-based fuzz framework to the security testing of tcg software stack implementations," in *Proceedings of the 2009 International Conference on Multimedia Information Networking and Security - Volume 01*, ser. MINES '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 149–152. [Online]. Available: http://dx.doi.org/10.1109/MINES.2009.111

[23] J. Julliand, P.-A. Masson, and R. Tissot, "Generating Security Tests in Addition to Functional Tests," in *AST '08: Proceedings of the 3rd international workshop on Automation of software test*. ACM, 2008.

[24] K. Lano, *The B language and method: a guide to practical formal development*. Springer-Verlag New York, 1996.

[25] P.-A. Masson, M.-L. Potet, J. Julliand, R. Tissot, G. Debois, B. Legeard, B. Chetali, F. Bouquet, E. Jaffuel, L. Van Aertrick, J. Andronick, and A. Haddad, "An access control model based testing approach for smart card applications: Results of the POSÉ project," *JIAS, Journal of Information Assurance and Security*, vol. 5, no. 1, pp. 335–351, 2010.

[26] A. Pretschner, T. Mouelhi, and Y. Le Traon, "Model-Based Tests for Access Control Policies," in *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, 2008.

[27] R. DeMillo, R. Lipton, and F. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Tutorial software quality assurance: a practical approach*, 1985.

[28] Y. L. Traon, T. Mouelhi, and B. Baudry, "Testing Security Policies: Going Beyond Functional Testing," in *The 18th IEEE International Symposium on Software Reliability*, 2007, pp. 93–102.

[29] J. Jürjens, "Model–based Security Testing Using UMLsec," *Electron. Notes Theor. Comput. Sci.*, vol. 220, no. 1, 2008.

[30] ——, "UMLsec: Extending UML for Secure Systems Development," in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*. Springer-Verlag, 2002.

[31] E. Veenendaal, "Practical Risk-Based Testing PRoduct RIsk Management: the PRISMA method," Improve Quality Services BV, Tech. Rep., 2006.

[32] M. Felderer, P. Zech, F. Fiedler, and R. Breu, "A Tool-based methodology for System Testing of Service-oriented systems," in *VALID 2010*, 2010, pp. 108–113.

[33] M. Felderer, B. Agreiter, and R. Breu, "Security Testing by Telling TestStories," in *Modellierung 2010*, 2010.

[34] T. Tuglular, O. Kaya, C. A. Muftuoglu, and F. Belli, "Directed Acyclic Graph Modeling of Security Policies for Firewall Testing," *Secure System Integration and Reliability Improvement*, vol. 0, pp. 393–398, 2009.

[35] H. Stallbaum, A. Metzger, and K. Pohl, "An automated technique for risk-based test case generation and prioritization," in *Proceedings of the 3rd international workshop on Automation of software test*, 2008, pp. 67–70.

[36] P. Zech, "Risk–Based Security Testing in Cloud Computing Environments," *2011 Fourth IEEE International Conference on Software Testing, Verification, and Validation*, pp. 411–414, 2011.

 ISBN: 978-1-61208-168-7