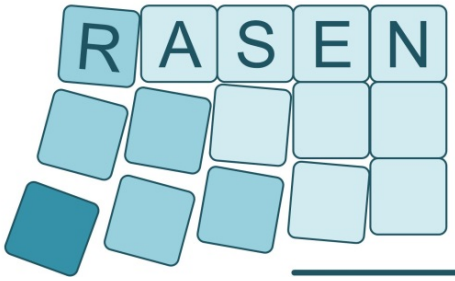

Compositional Risk Assessment and Security Testing of Networked Systems

Automated Risk-based Security Testing – Finding Vulnerabilities That Are Worth Being Found

Fabien Peureux[a], Martin Schneider[b], and Fredrik Seehusen[c]

[a]Smartesting, [b]Fraunhofer FOKUS, [c]SINTEF ICT

Fixing and even testing for every possible vulnerability is far too expensive. Especially when considering complex and networked systems, testing has to focus on vulnerabilities that would cause more damage when being exploited than it would cost to fix them.

Finding vulnerabilities in complex software requires either a lot of time or a lot of knowledge. Automated penetration testing usually leads to a large number of results with a significant amount of false positives. Each of these results has to be assessed individually. This requires, on the one hand, a lot of time to manage the large number of results and, on the other hand, much knowledge about the system under test to determine whether a result actually hints to a vulnerability. Hand-crafted vulnerability tests require a lot of knowledge about both the system under test and how to trigger vulnerabilities.

The possible impact of a vulnerability being exploited has to be considered as well, for example, on access to personal data or otherwise valuable data, as well on the reputation of a company.

A risk analysis provides information about the likelihoods for possible vulnerabilities and their impact on such assets when a vulnerability would be exploited. Today's systems are of such a size that testing for every possible bug is neither feasible nor economic. Therefore, risk analysis is an essential means to focus the testing strategy. Risk-based security testing allows for economic security testing by leveraging risk models resulting from risk analysis to optimize testing for security-relevant vulnerabilities. A risk model enables to estimate whether it is reasonable to test for a certain vulnerability or not.

Consider, for example, a system that uses a database to store information. A common vulnerability when accessing a database with user input is SQL injection. Such a vulnerability enables an attacker to access the database's content or to modify it. However, whether it is useful to test for this vulnerability strongly depends on the value of the data in the database and what it is used for. If a database contains data such as statistical data of a website (number of visitors, page views), a vulnerability where these data are disclosed is less critical than for user data, such as names, e-mail addresses and credit card numbers. Therefore, testing for a vulnerability depends on the impact and would be useful and economic in the second case but not in the first case.

Another requirement for efficient risk-based security testing is automation. Despite the fact that it is possible to perform risk-based security testing manually, this is not efficient. There are many techniques and tools that separately support the different steps of risk-based security testing, mainly risk analysis and security testing. Within the RASEN project, research partners and tool providers fill the gap between risk analysis and security testing in order to enable efficient risk-based security testing, providing automation where possible.

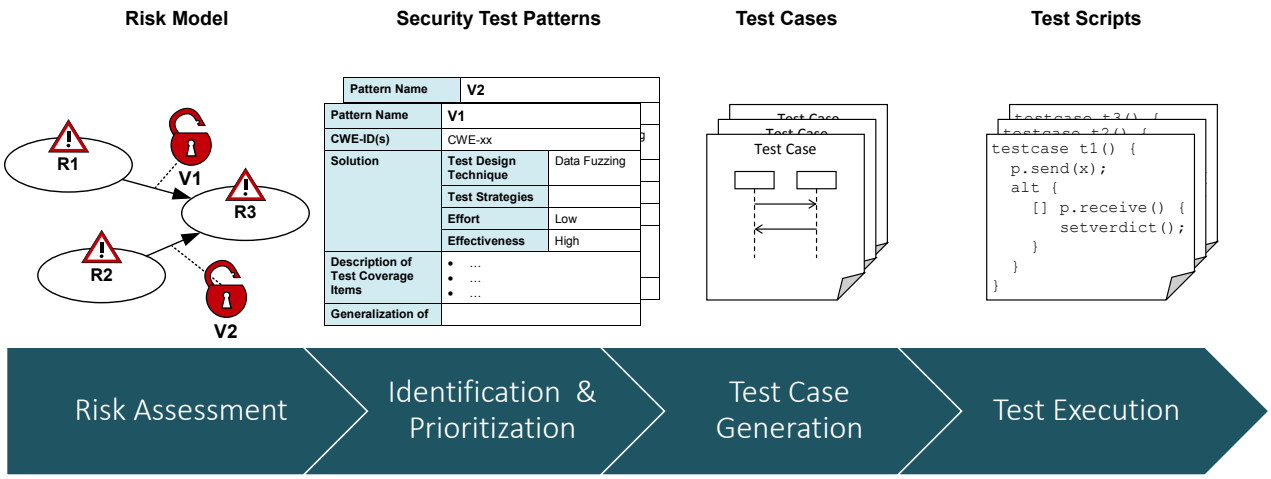

Figure 1. Overall Picture of RASEN risk-based security testing process. In the first step, a risk model is created using the *CORAS* method that identifies threat scenarios (R1, R2, R3) and potential vulnerabilities (V1, V2). The risk model is then used for the identification and prioritization of appropriate security test patterns. Based on the security test patterns, actual test cases are generated by combining information from the risk model, security test patterns, a system model and test generation techniques. The latter are test purposes generated from Smartesting *CertifyIt* and fuzzing techniques from Fraunhofer FOKUS's fuzzing library *Fuzzino*. In the last step, test scripts are generated, compiled and executed against the system under test.

Example: A test manager has been assigned to introduce a cost-efficient process of security testing. So far, risk analysis has been performed implicitly and security testing is done manually–no explicit risk analysis is performed. Testers design test cases based on their experiences and on different knowledge levels. Hence, test cases, results and revealed vulnerabilities heavily depend on the test designer that performed security testing. In order to reduce the cost for security test design, he decided to use a test tool for security testing that automatically generates security test cases. This dramatically improves time for generating test cases. However, there is actually no tool that actually guides the test generation process based on the analyzed risks.

The RASEN Approach

The testing approach, developed and promoted by the RASEN project for deriving test cases from risk assessment results, aims to make interrelated and synergetic risk management activity and security testing. To achieve this, the RASEN approach integrates the tools of the project partners: *CORAS*[1] from SINTEF (to address risk assessment), *CertifyIt*[2] from Smartesting (to perform risk and model-based test generation) and *Fuzzino*[3] from Fraunhofer FOKUS (to apply data and behavioral fuzzing techniques). This approach is depicted in Figure 1.

Identification and prioritization from risk assessment

The process starts on the left with the risk model as a result from the risk assessment. Risk may be defined as the combination of the impact of the severity (consequence) and the likelihood (probability) of a given hazardous vulnerability. Especially for complex systems, there are not sufficient resources to test all vulnerabilities and threat scenarios identified during risk analysis. Hence, a *CORAS* risk model (in relation with associated generic security test pattern and vulnerability catalogues) enables to select security test purposes and to prioritize them regarding risk estimation. *CORAS* is a model-driven method for risk analysis featuring a tool-supported modeling language specially designed to model threat scenarios.

[1] M.S. Lund, B. Solhaug, and K.Stølen. Model-Driven Risk Analysis: The CORAS Approach. Springer (2011). http://coras.sourceforge.net/

[2] E. Bernard, F. Bouquet, A. Charbonnier, B. Legeard, F. Peureux, M. Utting, and E. Torreborre. Model-based testing from UML models. Int. Workshop on Model-based Testing, LNCS, vol. 94. pages 223–230. Dresden, Germany (Oct. 2006) http://www.smartesting.com/

[3] Fraunhofer FOKUS: FuzzinglibraryFuzzinoonGithub https://github.com/fraunhoferfokus/Fuzzino

Each identified threat scenario, corresponding to a specific vulnerability, is linked to a dedicated security test pattern. Security test patterns express the testing procedures of recurring problems in security testing and thus allow the identification of the corresponding threat in a Web application. For risk-based security testing issues, we have adapted its initial structure, provided within the ITEA2 DIAMONDS[4] project, especially by adding some fields targeting testing automation needs (data vector libraries, test metrics to complete test coverage criteria, …). This test pattern catalogue is currently being extended to address the Top 10 weaknesses of the Open Web Application Security Project.

Identification and prioritization of risks allow focusing the testing on threat scenarios that would have the biggest impact on a system with respect to the likelihood that they will be exploited.

Deriving abstract test cases from security test patterns

Security test patterns based on prioritized vulnerabilities from the *CORAS* risk model thus provide a starting point for security test case derivation by providing information how appropriate security test cases can be created from risk analysis results. Indeed the test generation tool *CertifyIt* proposes generic test purposes to formalize each targeted vulnerability imported from risk assessment.

Test purposes drive the generation of abstract test cases based on security test patterns that form the starting point for security test cases.

A test purpose is a high level or regular expression that formalizes a test objective (in terms of states to be reached, behaviors to be activated and operations to be called) to drive the automated test generation on the behavioral model of the application under test.

[4]ITEA2 European Project no. 09018 (2010-2013)
DIAMONDS: Development and Industrial Application of Multi-Domain Security Testing Technologies
http://www.itea2-diamonds.org

The behavioral model uses the UML notation to describe the application: class diagrams specify the static structure and define the points of control and observation, while state diagrams graphically describes its behavioral characteristics. Security test patterns employ test purposes to automatically generate abstract test sequences: each test purpose produces one or more test cases verifying (i) the test purpose specification and (ii) the behavioral model constraints. An abstract test case takes the form of a sequence of steps, where a step corresponds to an operation call representing either an action or an observation of the application under test. Such test case, depicted by a UML sequence diagram, also embeds the security test strategies (from security test patterns) that is next used to apply data and behavioral fuzzing strategies during test scripts generation and execution.

Security test patterns are the missing link between risk analysis and security testing and pave the way for automated risk-based security testing.

Applying data and behavioral fuzzing techniques

Applying data fuzzing on abstract test cases, using the fuzz test data generation *Fuzzino*, consists to produce *JUnit* executable test cases that do not differ in the messages exchanged with the application under test but only in the values for arguments of these messages. *Fuzzino* determines the fuzz test data to generate by evaluating the security test strategies applied to a message arguments, the type description of the message argument and possibly valid values if provided. Only a few arguments of these messages contain fuzz test data. If for each of the different fuzz test data to be used to stimulate the system under test, this would result in a large number of test cases that differ merely in these fuzz test data. Hence, all the test cases have much in common. This blows up the model that can be avoided by specifying the basic message sequence and where fuzz test data shall be inserted at test execution time by using security test strategies. Thus, the model is kept clean, and at test execution time this message sequence is submitted to the system under test. In each iteration, different fuzz test data is used for the message arguments marked to carry fuzz test data using security test strategies.

Moreover, the presented approach is not limited to data fuzzing. For instance, behavioral fuzz test cases can be generated by the presented approach. Behavioral fuzzing means to generating invalid message sequences in contrast to data fuzzing. In case of behavioral fuzzing, strategies are used for test case generation by applying the corresponding behavioral fuzzing operators to the already generated test sequences in form of UML sequence diagrams. This computation results in a set of sequence diagrams that are behavioral fuzz test cases.

Security test strategies are a model-based way to guide the security test case generation.

Exploitation of test results for security assessment

The last phase thus exports and executes the test cases in the execution environment. In our case, it consists in creating a *JUnit* test suite, where each abstract fuzzed test case is exported as a *JUnit* test case, and creating an interface. This interface defines the prototype of each operation of the application and links the abstract structures and data of the test cases to the concrete ones. Since this process ensures the traceability between the verdict of the test case execution and the targeted vulnerabilities identified during risk assessment, the test results are gathered to automatically complement the risk picture of the system under test. Finally, this overall testing process to address large scale systems ensures scalability by the capacity to support a compositional testing approach and is experimented using the industrial case-studies proposed by the RASEN project partners (Software AG, InfoWorld and Evry).

The RASEN Project

The main overall objective of the RASEN project is to strengthen European organizations' ability to conduct security assessments of large scale networked systems through the combination of security risk assessment and security testing, taking into account the context in which the system is used, such as liability, legal and organizational issues as well as technical issues.

Consortium

The RASEN project is coordinated by SINTEF ICT and consists of the following partners:

- **EVRY**, Norway (www.evry.no)
- **Fraunhofer FOKUS**, Germany (www.fokus.fraunhofer.de)
- **Department of Private Law**, University of Oslo, Norway (www.jus.uio.no/ifp)
- **Info World**, Romania (www.infoworld.ro)
- **SINTEF ICT**, Norway (www.sintef.no)
- **Smartesting**, France (www.smartesting.com)
- **Software AG**, Germany (www.softwareag.com)

Contact

Visit the RASEN website or contact us by email.

- www.rasenproject.eu
- contact@rasenproject.eu

The project can also be followed on LinkedIn and Twitter.

- @RASENProject
- #RASENProject

www.linkedin.com/groups?home=&gid=7429037

Acknowledgments

The RASEN project (2012-2015) is funded by the European Commission via the Seventh Framework Programme, grant agreement no. 316853.

SEVENTH FRAMEWORK PROGRAMME

RASEN