Compositional Risk Assessment and Security Testing of Networked Systems

The PMVT approach: a RASEN innovation for security Pattern and Model-based Vulnerability Testing

Bruno Legeard[a,b], Fabien Peureux[a,b], Martin Schneider[c], Fredrik Seehusen[d], and Alexandre Vernotte[b]

[a]Smartesting, [b]Institut FEMTO-ST, [c]Fraunhaufer FOKUS, [d]SINTEF ICT

Vulnerability detection can be classified into two complementary categories: static and dynamic application security testing. Static Application Security Testing (SAST) are white-box approaches including source, byte and object code scanners. Dynamic Application Security Testing (DAST) includes black-box applications scanners, fuzzing techniques and emerging model-based security testing. This white paper introduces a DAST approach, called Pattern-driven and Model-based Vulnerability Testing (PMVT for short), proposed within the RASEN project to generate and execute vulnerability test cases. It combines model-based testing and a fuzzing technique, and drives the test generation by security test patterns selected from risk assessment. This approach is supported by processes and tools that effectively automate the detection of vulnerabilities and allow getting feedback about risk estimation.

Based on the current state of the art on security and on the security reports such as OWASP Top 10 2013, CWE/SANS 25 and WhiteHat Website Security Statistic Report 2013, Web applications are the most popular targets when speaking of cyber-attacks. The fact that modern society relies on the Web a little more everyday highlights the challenges of IT security, particularly in terms of data privacy, data integrity and service availability. The mosaic of technologies used in current Web applications increases the risk of security breaches. This situation has led to significant growth in application-level vulnerabilities. Application-level vulnerability testing is first performed by developers, but they often lack the sufficient in-depth knowledge in recent vulnerabilities and related exploits. This kind of tests can also be achieved by companies specialized in security testing, in penetration testing for instance. But they mainly use manual approaches, making the dissemination of their techniques very difficult, and the impact of this knowledge very low. Finally, Web application vulnerability scanners can be used to automate the detection of vulnerabilities, but since they often generate many false positive and negative, human investigation is also required.

The RASEN approach we propose, called Pattern-driven and Model-based Vulnerability Testing (PMVT for short), aims to improve the accuracy and precision of security and vulnerability testing, by proposing a model-based testing approach driven by risk assessment. It extends the traditional model-based security testing techniques by driving the testing process using security test patterns selected from risk assessment results. The adaptation of such techniques for risk-based vulnerability testing defines novel features and perspectives in this research domain.

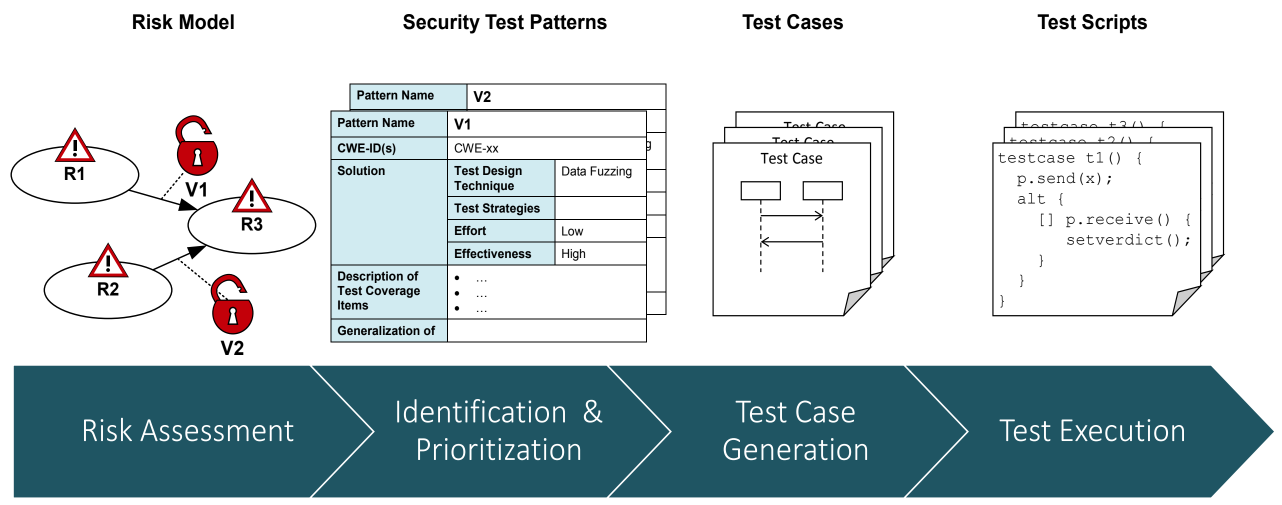

Figure 1. Overall Picture of the PMVT risk-based vulnerability testing process.

Within the RASEN project, a dedicated tool environment has been developed to support the end-to-end PMVT process. It automates the detection of such vulnerabilities, provides feedback about risk assessment, and therefore makes it possible to automatically complement the related risk picture of the application under test.

Principles of the PMVT Approach

The PMVT testing process, 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 as shown in Figure 1. 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, test cases are generated by combining information from the risk model, security test patterns, a test model and test generation techniques. The latter are composed of 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 application under test. Hence, the PMVT approach integrates the tools of the project partners: *CORAS*[1] from SINTEF ICT (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 fuzzing techniques). This tool chain is depicted in Figure 2.

Risk identification and prioritization

The PMVT process starts with the risk model as a result from the risk assessment. Risk may be defined as an unwanted incident which may occur at a given likelihood and impact an asset with a given consequence. Especially for complex systems, there are not sufficient resources to test all vulnerabilities and threat scenarios identified during risk analysis.

Each threat scenario of the *CORAS* risk model is linked to a security test pattern defining the testing procedure to detect the threat in the application under test.

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 with respect to risk estimation. *CORAS* is a model-driven method for risk analysis featuring a tool-supported modeling language specially designed to document risks and their causes.

[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: Fuzzing library Fuzzino on Github https://github.com/fraunhoferfokus/Fuzzino

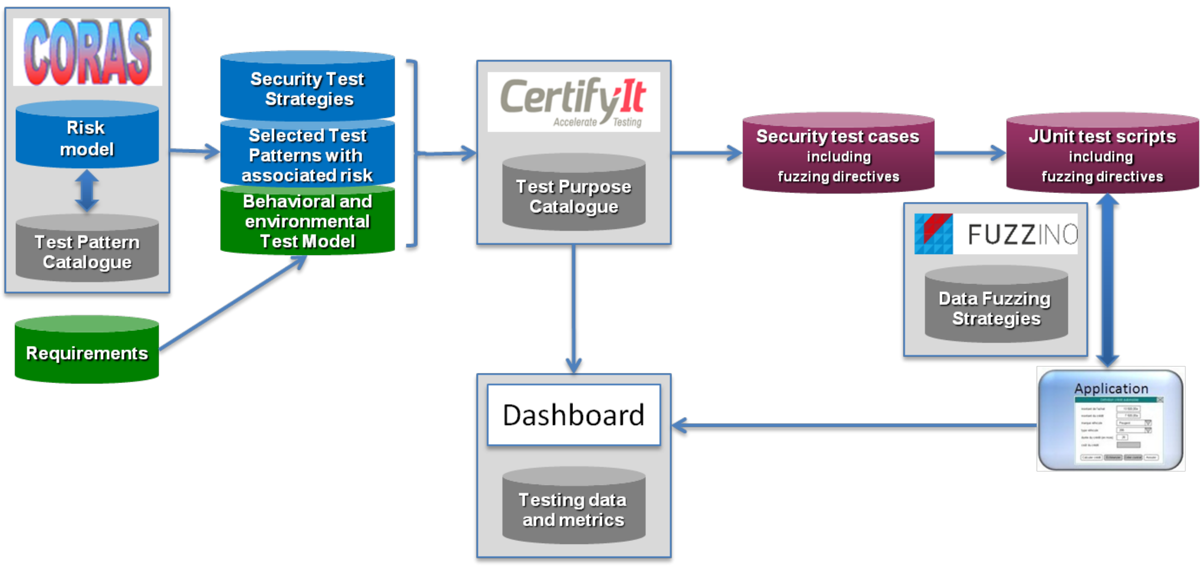
CORAS
Risk model
Test Pattern Catalogue
Requirements
Security Test Strategies
Selected Test Patterns with associated risk
Behavioral and environmental Test Model
CertifyIt Accelerate Testing
Test Purpose Catalogue
Security test cases including fuzzing directives
JUnit test scripts including fuzzing directives
FUZZINO
Data Fuzzing Strategies
Application
Dashboard
Testing data and metrics

Figure 2. Overall Picture of PMVT risk-based vulnerability testing tool.

Risk model elements such are vulnerabilities are linked to dedicated security test patterns. 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. This test pattern catalogue typically addresses the Top 10 weaknesses of the Open Web Application Security Project (OWASP)[4].

Test purpose catalogue

Security test patterns, based on prioritized vulnerabilities from the *CORAS* risk model, provide a starting point for security test case derivation by giving information how appropriate security test cases can be created from risk analysis results.

A test purpose allows formalizing the intention of a given security test pattern and thus allows to automatically generate the expected test cases with model-based testing techniques.

This way, the security test patterns, which are used by the test generation algorithm, are gathered from the risk model elements of each *CORAS* model. Moreover, likelihood and consequence are also collected from the *CORAS* model and used as a basis for prioritization. The test generation tool *CertifyIt* is supported by a catalogue of generic test purposes that formalize test patterns.

A test purpose is a high-level expression that formalizes a testing objective to drive the automated test generation on the test model. Basically, such a test purpose can be seen as a partial algorithm that defines a sequence of significant steps that has to be executed by the test case scenario. Each step takes the form of a set of operations or behaviors to be covered, or specific state to be reached on the test model in order to assess the robustness of the application under test with respect to the related vulnerability to be tested.

Behavioral and environmental test model

To apply model-based testing, a behavioral and environmental test model is used to instantiate the selected test purposes and therefore to automatically generate abstract test cases. The PMVT approach, based on *CertifyIt* technology, requires a model designed using the UML4MBT[5] notation: UML class diagrams specify the static structure, while state diagrams describes the behavioral aspects. To ease and accelerate the modeling activity, a Domain Specific Modeling Language (DSML), called DASTML, has been especially created.

A Domain Specific Modeling Language (DSML) dedicated to Web application description makes the design of the test model easier and less time consuming.

[4] See https://www.owasp.org

[5] F. Bouquet et al. A subset of precise UML for model-based testing. In the 3rd int. Workshop on Advances in Model Based Testing, London, UK, July 2007. ACM Press

DASTML enables the modelling of the global structure of a Web application: the available pages, the available actions on each page, the user inputs of each action potentially used to inject attack vectors, etc. It solely represents all the structural entities needed to generate the test model. Such a DASTML instantiation is automatically translated into a UML4MBT test model using a dedicated Smartesting plug-in.

To manage traceability between risk assessment and the generated test cases, the test model also embeds all the related information about prioritization and test procedure, which are inherited from the *CORAS* model and the test pattern.

Generation of the abstract test cases

The test generation process enables automatic generation of abstract vulnerability test cases, including the expected results. It consists of instantiating the vulnerability test purposes on the test model of the application under test. To achieve that, the test model and the test purposes are both translated into elements and data directly computable by the test generator *CertifyIt*.

Test case generation is performed by instantiating the selected test purposes on the behavioral UML4MBT test model specifying the Web application under test.

Notably, test purposes are transformed into test targets, which are defined by a sequence of intermediate objectives used by the test generation engine. The test targets are then executed on the test model to generate the abstract test cases. In this way, each test purpose produces one or more abstract test cases verifying the test purpose specification and the behavioral test model constraints. Such 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 (the abstract test cases can also be described and handled using UTP sequence diagrams).

The abstract test cases are finally exported into the execution environment: this consists of automatically creating a JUnit test suite, in which each abstract test case is exported as a JUnit test case skeleton. The JUnit test suite also embeds the security test strategies (from security test patterns) that is next used to apply data fuzzing strategies during the generation and the execution of the test scripts.

Generation of the executable test scripts

During the modeling activity, all data used by the application are modeled at an abstract level. As a consequence, the test cases are abstract and cannot be executed as they are. To bridge the gap, the test case generator produces a file containing the prototype of each operation of the application and links the abstract structures / data of the test cases to the concrete ones. The test automation engineer is in charge of implementing each operation and data of this interface.

Moreover, the fuzz test data generation *Fuzzino* automatically determines the attack vectors to generate by evaluating the security test strategies applied to the operation arguments. Typically, only a few arguments of these messages contain attack vectors. However, this would result in a large number of test cases that differ merely in these fuzzed data. This approach has the advantage to keep clean the test model and to externalize the definition of the attack vectors, that can be independently updated or parameterized. In the end, each abstract fuzzed test case defines an executable JUnit test case, which can be executed by a JUnit environment supporting the management of the executable test suite and its computation on the application under test to assign the execution verdict.

Exploitation of the test results for risk assessment

The last phase of the PMVT process thus consists of exporting and executing the test cases in the execution environment. The test results are then gathered and displayed in a dashboard that allows to provide various security testing metrics. Security testing metrics are a concept for the transfer of information from security testing to risk assessment. Typically, they concern the number of test cases executed, passed or failed, the percentage of the attack surface elements (coverage) that are addressed by the test cases, the characteristics of the attack vector, the coverage of the paths to execute the attack, etc.

Traceability between the initial targeted security test patterns, the corresponding test cases and the test results enables to evaluate and complete the risk picture.

The PMVT process ensures the traceability between the test case, the verdict of the execution and the targeted vulnerabilities identified during risk assessment. The results of the security testing metric functions should help to characterize the security risks of the system under test. Hence, it should be possible to improve a risk assessment based upon the results since these functions can be used, for instance, to calculate and update the likelihood values regarding the exploitability of some threat scenarios.

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)
- **Institut FEMTO-ST,** University of Franche-Comté, France (www.femto-st.fr)
- **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