SEVENTH FRAMEWORK PROGRAMME

RASEN

Compositional Risk Assessment and Security Testing of Networked Systems

Deliverable D3.3.3 and D4.3.3

RASEN Tools for:

Compositional security risk assessment and security test result aggregation v.3

and

Compositional Security Risk Assessment and Security Testing v.3

Brief Description, Documentation and Installation Guide

Project title:	RASEN
Project number:	316853
Call identifier:	FP7-ICT-2011-8
Objective:	ICT-8-1.4 Trustworthy ICT
Funding scheme:	STREP – Small or medium scale focused research project

Work package:	WP3 and WP4
Deliverable number:	D3.3.3 and D4.3.3
Nature of deliverable:	Prototype & Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2015-09-30
Actual delivery date:	2015-09-30
Responsible partner:	UFC

Contributors

Editors	Fabien Peureux (UFC)
Contributors	Fabien Peureux (UFC), Fredrik Seehusen (SINTEF), Alexandre Vernotte (UFC), Johannes Viehmann (Fraunhofer), Frank Werner (SAG)
Quality assurors	Fredrik Seehusen (SINTEF), Bjørnar Solhaug (SINTEF)

Version history

Version	Date	Description
0.1	15-09-15	ToC and contents overview
0.2	15-09-28	Contents provided to all sections
0.3	15-09-29	Complete version finalized for internal review
1.0	15-09-30	Final version

Abstract

This report provides some basic information about the tools of the WP3 and WP4 prototype deliverable month M36. The delivered tools are CORAS from SINTEF, RACOMAT and RASEN Security Dashboard from Fraunhofer FOKUS, Smartesting CertifyIt and its plugin from UFC, and ARIS Business Architect from Software AG.

Keywords

Security, security risk assessment, security testing, test-based risk assessment, tool support, prototype

Executive Summary

The overall objective of RASEN WP3 and WP4 is to develop tools and techniques to support test-based and compositional security risk assessment. Deliverable D3.2.3 presents the WP3 techniques that were developed during the first two years of the project and deliverable D4.2.3 shows the WP4 techniques that were developed during the first two years of the project, while D5.3.1 presents the methodologies that the WP3 techniques should support. This report accompanies the D3.3.3 and D4.33 prototype deliverable, and gives an overview and introduction to the four tools of that deliverable.

The set of tools are as follows:

- The CORAS tool from SINTEF, supporting model-driven risk analysis
- The RACOMAT tool from Fraunhofer FOKUS, which is a tool for the bidirectional combination of low level risk assessment with security testing that could also be used as an integration platform for other tools in such a process
- The Security plugins developed by UFC to extend the tool Smartesting CertifyIt devoted to model-based security testing
- The ARIS Business Architect from Software AG, supporting security assessment and risk modeling of software and IT systems
- The RASEN Security Dashboard provided by Fraunhofer FOKUS to aggregate test results and to provide test metrics and measurements.

Table of contents

TABLE OF CONTENTS **5**

1 INTRODUCTION **6**

2 CHARACTERISTICS OF THE IDENTIFIED WP3 AND WP4 TOOLS **7**

2.1 RACOMAT TOOL 7
2.2 SMARTESTING CERTIFYIT WITH UFC SECURITY PLUGIN 8
2.3 ARIS BUSINESS ARCHITECT WITH RASEN PLUGIN 9
2.4 CORAS 10
2.5 RASEN SECURITY DASHBOARD 11

3 DESCRIPTION OF THE PROTOTYPE TOOLS **12**

3.1 RACOMAT TOOL 12
3.1.1 Features of the RACOMAT Tool 13
3.1.2 Installation of the RACOMAT Tool 14
3.1.3 Documentation of the RACOMAT tool 14
3.2 SECURITY PLUGINS FOR SMARTESTING CERTIFYIT 18
3.2.1 Presentation 18
3.2.2 Installation Guidelines 18
3.2.3 User Guide 19
3.2.3.1 Modeling Activity 19
3.2.3.2 Test Purpose Activity 20
3.2.3.3 Test Generation Activity 22
3.2.3.4 Test Concretization and Execution Activities 23
3.3 ARIS BUSINESS ARCHITECT – RASEN PLUGIN 25
3.3.1 Presentation 25
3.3.2 Installation Guidelines 25
3.3.3 User Guide 26
3.4 CORAS TOOL SET 29
3.4.1 Presentation 29
3.4.2 Installation Guidelines 30
3.4.3 User Guide 31
3.4.3.1 Core CORAS Tool 31
3.4.3.2 Capec2Coras Tool 31
3.4.3.3 CorasAnalyzer Tool 33
3.5 SECURITY DASHBOARD 42
3.5.1 Presentation 42
3.5.2 Installation Guidelines 42

4 CONCLUSION **43**

1 Introduction

This report is a brief documentation and introduction to the RASEN WP3 and WP4 prototype tools. The prototypes are developed to provide support for the methods and techniques that are also developed in this work package. The reader is referred to deliverables D3.2.3 and D4.2.3 for an overview of the WP3 and WP4 research results after the three years of the RASEN project.

The prototypes presented in this report serve as a part of the RASEN tool-box for security risk assessment and security testing that is developed in the context of WP5. WP5 defines the common RASEN data meta-model that will be used for integrating the tools by facilitating the communication between them. An important part of the RASEN prototype development is therefore to provide support for exporting and importing data to and from the common RASEN data model.

The tools that are presented in the next two sections are RACOMAT, Smartesting CertifyIt and its RASEN plugins, ARIS Business Architect, CORAS, and the RASEN Dashboard. In Section 2 we give a brief overview of the main characteristics of the WP3 and WP4 tools. In Section 3 we present the tools in some more details, give installations and user guidelines, and explain the planned and current status for the development of tool features relevant in WP3 and WP4. Finally, in Section 4 we conclude.

2 Characteristics of the Identified WP3 and WP4 Tools

In this section we give an overview of the RASEN tools relevant for WP3 and WP4, providing some basic general information and technical information, as well as other information that may be useful.

2.1 RACOMAT Tool

General information	
Name	RACOMAT (Risk Analysis COMbined with Automated Testing) Tool
Provider	Fraunhofer FOKUS
Topic addressed	The entire combined test-based risk assessment (TBRA) and risk-based security testing (RBST) process.
Description	Utilizing static and dynamic analysis techniques, the RACOMAT tool generates initial editable fault trees or CORAS risk graphs with linked system models for programs, libraries, components and web interfaces. Users decide which elements from existing risk related libraries like MITRE CAPEC and CWE should be added. The RACOMAT tool supports compositional risk analysis with simple drag and drop for existing artefacts and it calculates likelihoods for dependent incidents by performing Monte Carlo simulations. Security test patterns are automatically associated with risk analysis artefacts as well as system model components (e.g. input and output ports) and their priority is calculated. If no appropriate test patterns exist in the library, the tool allows its users to create new test patterns within the tool and to upload them to the library for sharing. Given an appropriate test pattern, test generation, execution and result aggregation are at least semi-automated. Indeed, there is no need for additional manual work at all for testing many common issues like overflows or SQL injections. Security testing metrics suggested by the test patterns can be used to analyze and evaluate test results. New security testing metrics can be created and edited in the RACOMAT tool. With appropriate security testing metrics, it is possible to update the risk graphs automatically with more precise likelihood estimates or new faults / incidents based on the test results. Besides using the RACOMAT tool as a stand-alone tool, it is possible to use the RACOMAT tool as an integration platform and to utilize other eventually more specialized tools for some steps in the combined TBRA and RBST process.
License	RASEN project partner can obtain a license for the project duration. The final license model is not yet decided.
Website	http://www.rasenproject.eu/the-racomat-tool-2/
Technical information	
Download site	Public beta planned for fourths quarter 2015
OS	Windows Vista or newer Windows version
Technology environment	.Net 4.5, WPF
Other dependencies	none
Additional information	
Known issues/risks	RACOMAT security testing is meant to attack SUTs – use with caution!
Additional useful information	Since there is not yet a public security test pattern library server online, only a local test pattern library can be used in the delivered RACOMAT tool version.

2.2 Smartesting CertifyIt with UFC Security plugin

General information	
Name	Smartesting CertifyIt
Provider	Smartesting
Topic addressed	Model-Based Testing solution
Description	The tool is composed by a Rational Software Architect plugin for modelling activities, a standalone Java application for the test generation and test case management, and dedicated RASEN plugin to manage DASTML specifications, vulnerability test generation from test purposes and RASEN test publisher.
License	RASEN project partner can obtain a license for the project duration.
Website	www.smartesting.com
Technical information	
Download site	www.smartesting.com
OS	Win or Linux
Technology environment	Rational Software Architect v8.0.x and v8.5.x, EMF compliant JAVA 1.6, 1.7 compliant
Other dependencies	N/A
Additional information	
Known issues/risks	N/A
Additional useful information	N/A

2.3 ARIS Business Architect with RASEN Plugin

General information	
Name	ARIS Business Architect + RASEN Plugin
Provider	Software AG
Topic addressed	Security Assessment and Risk Modeling of Software and IT Systems
Description	The model extension is based on the ARIS Business Process Analysis Platform, a proprietary solution of Software AG and provides an interface to model risk assessment of IT security systems.
License	Proprietary
Website	http://www.softwareag.com/corporate/products/aris_alfabet/bpa/overview/default.asp
Technical information	
Download site	Download site of the RASEN Model Extension https://project.sintef.no/eRoomReq/Files/ikt2/RASEN/0_48ae6/Y3-SAG%20Tool%20Deliverable%20RASEN.zip
OS	Web-based solution, server installation supports Windows, Linux, (Any OS supported by the ARIS Business Architect)
Technology environment	None
Other dependencies	None
Additional information	
Known issues/risks	None
Additional useful information	None

2.4 CORAS

General information	
Name	CORAS tool
Provider	SINTEF
Topic addressed	Model-based risk assessment
Description	The tool is based on Eclipse and the GMF/EMF framework, but it is distributed as a stand-alone tool
License	Eclipse Public License v1.0 (http://www.eclipse.org/legal/epl-v10.html)
Website	http://coras.sourceforge.net
Technical information	
Download site	http://coras.sourceforge.net/downloads.html
OS	Tested on Windows; a non-tested distribution is also available for Linux
Technology environment	The tool needs Java
Other dependencies	None
Additional information	
Known issues/risks	None
Additional useful information	None

2.5 RASEN Security Dashboard

General information	
Name	RASEN Security Dashboard
Provider	Fraunhofer FOKUS
Topic addressed	RASEN Security Dashboard, visualizing security test metrics and generating aggregated test reports.
Description	The RASEN Security Dashboard processes the models generated during the RISK security process of RASEN, namely: risk models, test report models and test procedures in the exchange format, also specified in the RASEN project. It generates different metrics and visualizes them in different view formats and provides an export of an aggregated test report for items of interest.
License	RASEN project partner can obtain a license for the project duration. The final license model is not yet decided.
Website	N/A
Technical information	
Download site	N/A
OS	Windows
Technology environment	Java, Eclipse Juno (Eclipse Modelling Tools)
Other dependencies	CORAS tool in case of installed CORAS bridge
Additional information	
Known issues/risks	Not recommended installation component of the CORAS tool
Additional useful information	N/A

3 Description of the Prototype Tools

3.1 RACOMAT Tool

Risk assessment might be difficult and expensive, and it often depends on the skills and estimates of the analysts. Testing is one analysis method that might yield more objective results, but security testing itself might be difficult and expensive, too, because security testing means to test for unwanted behavior and there is usually no specification what to expect. Besides that manual testing is itself error prone and infeasible for large scale systems, even highly insecure system can produce lots of correct test verdicts if the “wrong” test cases have been created and executed. Therefore, it makes sense to do Risk Assessment COMbined with Automated Testing, i.e. to use RACOMAT Tool. The RACOMAT Tool is intended to cover the entire process of combined test-based risk assessment (TBRA) and risk-based security testing (RBST) within a single standalone tool. It allows applying and evaluating the RASEN Method without troubling about interoperation and interaction between different tools.

However, more specialized tools might be more powerful than the RACOMAT Tool for some tasks. Eventually, use case partners do already use some existing tools for some steps of the risk assessment and security testing process. Therefore, we try to build bridges from the RACOMAT tool to other tools, especially to those developed or used by our project partners. The idea is to make the RACOMAT Tool the central integration platform which can be used to control the overall combined TBRA and RBST process while for some tasks within that process, other tools are used. Since the RACOMAT Tool itself implements the entire process, there will be no gaps, for sure, even if for some steps within the process there were no other tools than the RACOMAT Tool available.

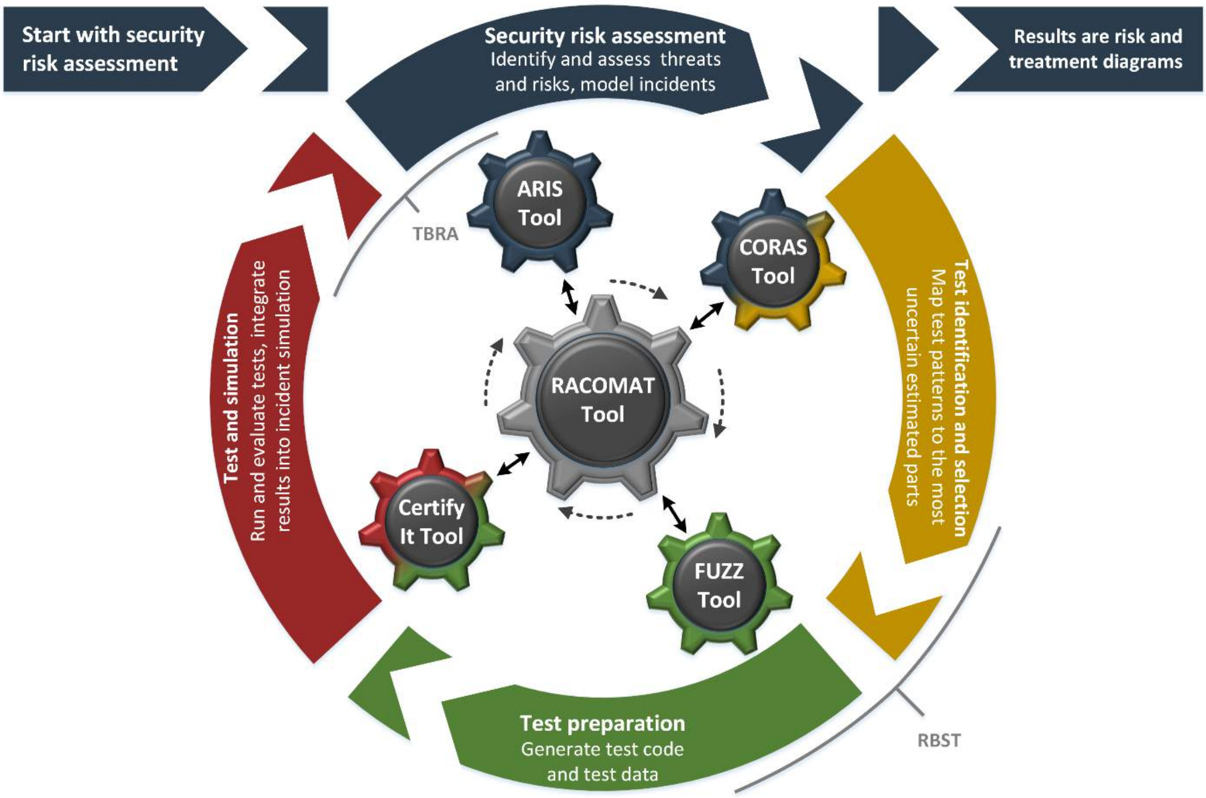

Start with security risk assessment
Security risk assessment
Identify and assess threats and risks, model incidents
Results are risk and treatment diagrams
TBRA
ARIS Tool
CORAS Tool
RACOMAT Tool
Certify It Tool
FUZZ Tool
Test identification and selection
Map test patterns to the most uncertain estimated parts
Test and simulation
Run and evaluate tests, integrate results into incident simulation
Test preparation
Generate test code and test data
RBST

Figure 1 – The process of the RACOMAT Method with the RACOMAT tool as a platform integrating various other tools

3.1.1 Features of the RACOMAT Tool

Current Status

- Component based, low level system analysis and risk assessment
    - Automatically creates interface models for programs, APIs, components, Web-Pages or Web-Services
    - Generates semi automatically initial fault trees (FTA), event trees (ETA) or CORAS risk graphs
    - Reusable risk assessment artifacts – uses existing risk catalogues (MITRE CWE / CAPEC, BSI IT-Grundschutz …)
    - Compositional risk analysis,edit and compose per Drag and Drop
    - Calculates likelihoods for dependent incidents automatically
        - Supports timing issues – likelihoods might change over time
    - Identifies and prioritizes elements worth further investigation
    - Allows to model relations between risk artefacts and with system components
- Supports security testing as a part of the risk analysis process
    - Security Test Pattern instantiation
        - Suggests test patterns for identified threat scenarios
        - Assisted association with risk artefacts and system components by drag and drop
            - Indicates where to stimulate and what to observe
    - Execution of tests, observation
        - Once a test pattern is instantiated, generating, executing and evaluating tests woks at least semi automatically
            - Often no manual work is required at all, e. g. for overflows or (SQL-) Injections
        - Automatic observation and basic aggregation of raw test results
- Updates the risk picture based upon the test results semi automatically
    - Makes suggestions using the metrics associated with the security test pattern
    - More precise likelihood values
    - Allows to add unexpected observations as new faults or unwanted incidents by dragging them to the risk graph
        - Automatically creates relations to related threat scenarios that were tested
- Create, edit and share reusable artefacts
    - Threat interfaces
    - Security test patterns
    - Security testing metrics
- Flexible dashboard showing final risk assessment results for the management
- Interoperability
    - Import and export in XML RASEN exchange formats
    - Import from and export to ARIS Business Architect (Software AG case study)

Planned Features

The most relevant upcoming features expected within the first public beta release are the following:

- Calculate how much test effort should be spend for which tests (monetary value)
- Further import / export functionality
- Integration into Microsoft Visual Studio IDE
  - Plug-in
  - Use powerful Visual Studio code editor, debugging tools etc. for creating and editing test patterns and testing metrics directly within Visual Studio
- Public RACOM server to share threat interfaces, test pattern and testing metrics

3.1.2 Installation of the RACOMAT Tool

To install the RACOMAT tool, just execute the setup executable contained in the RACOMAT.zip file on a computer running Microsoft Windows Vista or a newer Microsoft Windows version. For best compatibility, we recommend using the latest Microsoft Windows 10 version.

The RACOMAT tool requires Microsoft .NET 4.5 or a later, compatible version. The RACOMAT setup program should automatically download and install the .NET framework if no appropriate version has been installed before. Eventually the user has to confirm the installation of .NET and accept the related Microsoft license terms. The latest version of .NET can also be obtained manually from https://www.microsoft.com. Note that an internet connection will be required anyway to download the .NET setup program since it is not included in the RACOMAT.zip file.

3.1.3 Documentation of the RACOMAT tool

RACOMAT is still a prototype. The online documentation is currently under construction. However, there are already some tool tips in the program and there is a short Video Demo / Tutorial showing the basic workflow included in the tool deliverable ZIP file.

Here we also give a short tutorial introducing the most important concepts. For demonstration, a web application called Damn Vulnerable Web Application (DVWA) will be used here, because it has many weaknesses. DVWA can be obtained from here: http://www.dvwa.co.uk/.

After starting RACOMAT, users will see an empty risk graph. By dragging elements from the toolbar on the left to the risk graph, it is possible to manually create a risk model. Drag one threat interface to the graph. Choose “HTTP interface” from the small dialog that pops up.

The http interface assistant will be opened, which is most appropriate to create the initial system model for an application having an http based web interface like DVWA.

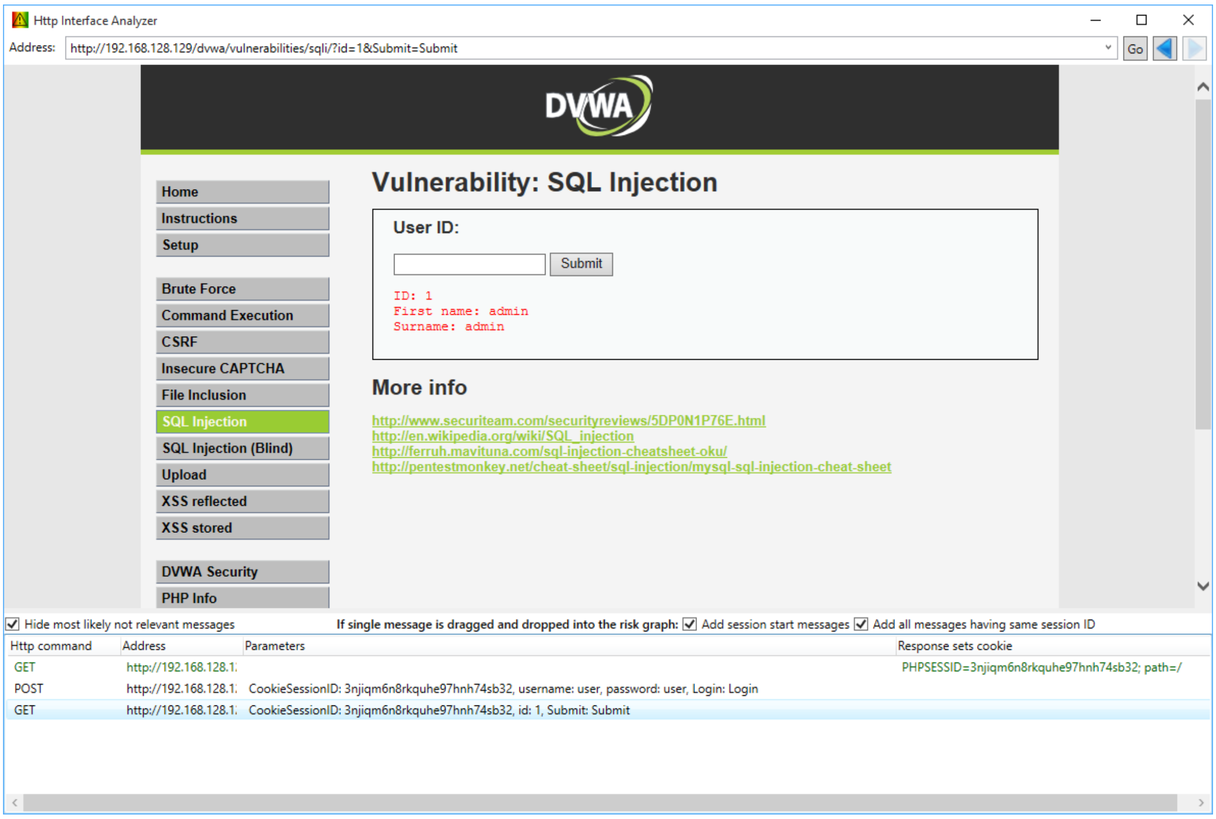
The http assistant basically contains a web browser. Just enter an URI in the address field and hit return or click the “Go” button. While browsing the web, RACOMAT automatically records messages that could eventually be used to test the interface. These recorded messages will be displayed in the list view at the bottom of the http assistant window. Drag and drop one or more of the recorded messages to the risk graph.

In the example shown here, we just log in to DVWA, choose SQL Injection from the menu and submit the value “1”. We drag the only the last message to the graph.

Close the http assistant window when done by clicking the cross in the upper right corner of that window.

So far, the risk graph only contains a system model. In order to add risk related information, click on one of the “String” buttons in the input column of some added threat interface instance. A window with a list of MITRE CWE based weaknesses typically related to that kind of input parameters will be shown. If some weakness in the list is not applicable, right click it. Else, drag it into the input field for which it might be relevant. Thereby, a new vulnerability will be added to the risk graph.

In the example, we drag the SQL injection weakness to the input field “ID”.



http://192.168.128.129/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit

Vulnerability: SQL Injection

User ID:

```
ID: 1
First name: admin
Surname: admin
```

More info

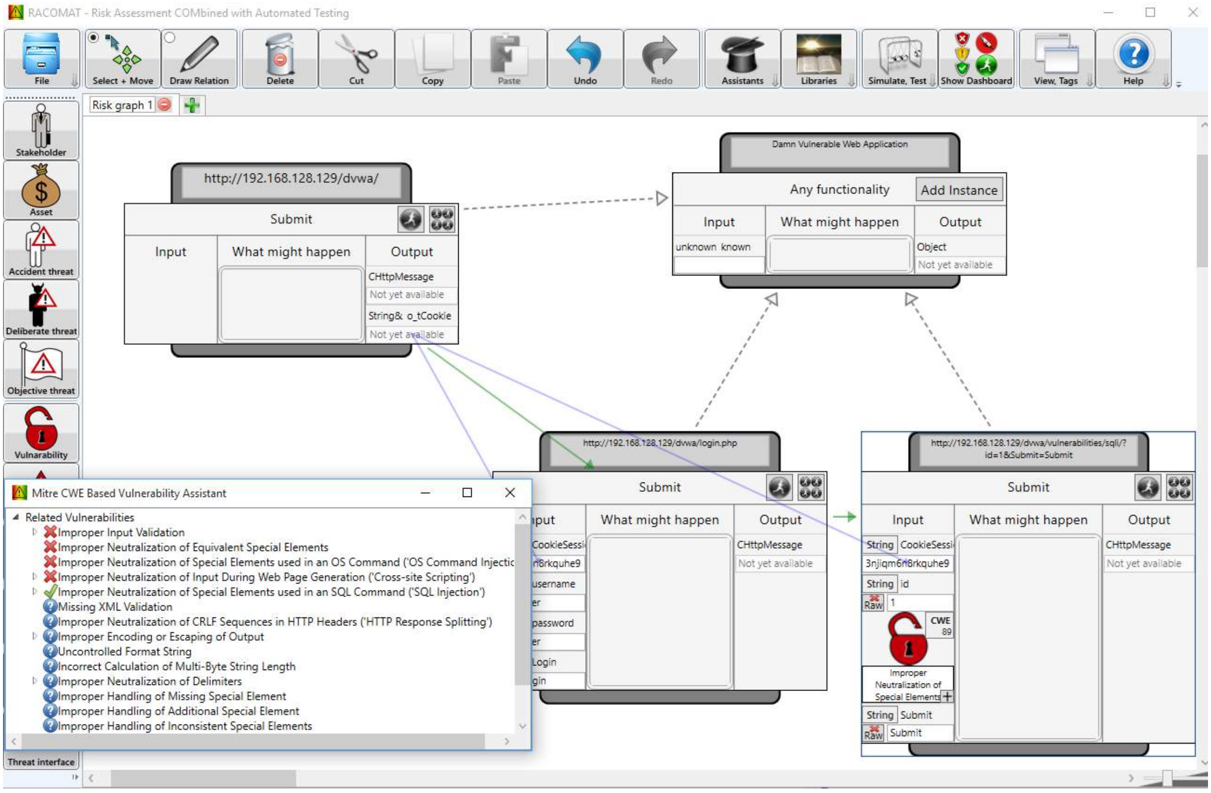
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

☑ Hide most likely not relevant messages

If single message is dragged and dropped into the risk graph: ☑ Add session start messages ☑ Add all messages having same session ID

Http command	Address	Parameters	Response sets cookie
GET	http://192.168.128.1		PHPSESSID=3njiqm6n8rkquhe97hnh74sb32; path=/
POST	http://192.168.128.1	CookieSessionID: 3njiqm6n8rkquhe97hnh74sb32, username: user, password: user, Login: Login	
GET	http://192.168.128.1	CookieSessionID: 3njiqm6n8rkquhe97hnh74sb32, id: 1, Submit: Submit	

Figure 2 – The http assistant



RACOMAT - Risk Assessment COMbined with Automated Testing

Mitre CWE Based Vulnerability Assistant

- Related Vulnerabilities
  - ✗ Improper Input Validation
  - ✗ Improper Neutralization of Equivalent Special Elements
  - ✗ Improper Neutralization of Special Elements used in an OS Command ('OS Command Injectic
  - ✗ Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
  - ✓ Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - ? Missing XML Validation
  - ? Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
  - ? Improper Encoding or Escaping of Output
  - ? Uncontrolled Format String
  - ? Incorrect Calculation of Multi-Byte String Length
  - ? Improper Neutralization of Delimiters
  - ? Improper Handling of Missing Special Element
  - ? Improper Handling of Additional Special Element
  - ? Improper Handling of Inconsistent Special Elements

Figure 3 – Adding risk information with the MITRE CWE based assistant

Close the MITRE CWA assistant and click on one of the "CWE" button of some vulnerability. In the menu that opens choose CAPEC based threat scenarios and drag any relevant threat scenarios to the "What might happen" column of the threat interface. This will add threat scenarios to the risk graph and create relations to the weaknesses from which the threat scenarios were created. In the example we add the attack pattern "SQL Injection".

The threat scenarios have a "CAPEC" button. Clicking such a "CAPEC" button opens another menu, which can be used to add unwanted incidents. Add one or more of these unwanted incidents by dragging them to the risk graph, e.g. to some field in the output column of the threat scenario. In the example we select the unwanted incident "SQL injection most likely occurred", which is actually defined by some test pattern.

Each vulnerability, threat scenario and unwanted incident has a small button with a plus sign in its lower right corner. Clicking on that button will show more detailed information. To start security testing, open the detail area for some threat scenario by clicking on its "+" button. From the combo box on the very top of that detail area, it is possible to choose a test pattern that should be applied. If there is already at least one fitting test pattern existing for that threat scenario and associated with the CAPEC ID, then by default some test pattern will already be selected. Otherwise, a test pattern has to be selected manually. Eventually, even a new test pattern has to be created. In the example, the Basic SQL Injection test pattern is already pre-selected. So there is nothing that needs to be done manually.

Just hit the execute button (Figure 4) in order to start the testing process.

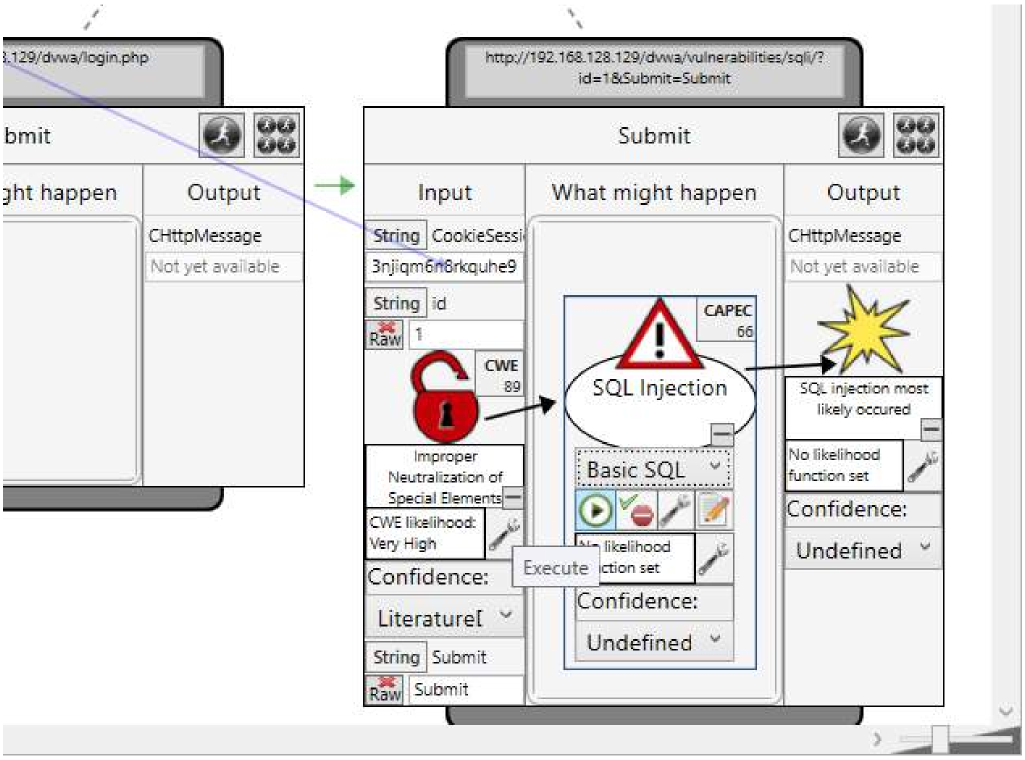
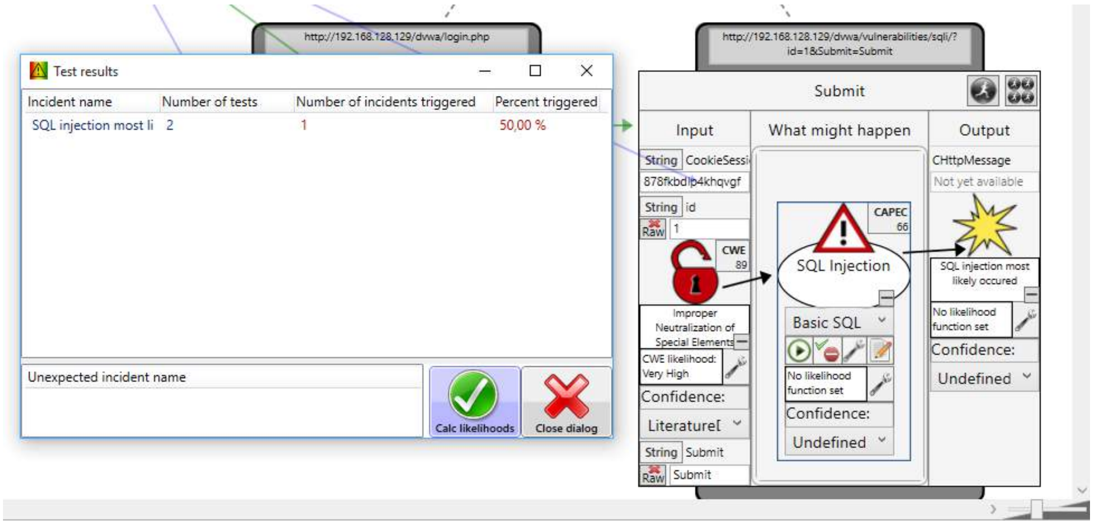


Figure 4 – Executing tests

The test results will be shown in a separate window. Hit the “Calc likelihoods” button to update likelihood values in the risk graph. If there are any unexpected incidents detected, then these should be dragged to the risk graph.


Test results

Incident name	Number of tests	Number of incidents triggered	Percent triggered
SQL injection most li	2	1	50,00 %

Unexpected incident name

Figure 5 – Interpreting test results in the terms of risk assessment

3.2 Security Plugins for Smartesting CertifyIt

This section introduces the Security plugins developed by UFC for the tool CertifyIt provided by the company Smartesting. It enables to generate security test cases from vulnerability risk assessment, and to produce test results, which can actively contribute to identify, estimate and finally treat the risk of the tested application

It should be noted that a more detailed presentation of the tool features including tutorial, and tips to use it efficiently, are provided with the tool (modeling guide, UML/OCL reference guide and software documentation).

3.2.1 Presentation

Smartesting CertifyIt is a tool suite that automatically generates test cases based on a model of system requirements. Manual test design is labor intensive and error prone; this manual work can be avoided for complex applications by modeling the key concepts (abstraction) and allowing Smartesting CertifyIt to automate your test design work. Since the model is more expressive and simpler than the system-under-test, it can more readily be reviewed for correctness and coherency, as well as be updated more easily. Some plugins have been developed during RASEN project for the deployment of the RASEN approach in order to assess its accuracy and precision regarding risk-based objectives. Smartesting CertifyIt including these plugins supports UML/OCL models as the specification modeling language, and generates test cases to cover security test patterns used as test objectives. Finally, the generated test cases can be exported in UML sequence diagrams and can be fully integrated into the RASEN Security Dashboard introduced in section 0.

More precisely, the Smartesting plugins developed by UFC concern the following features:

1. Test Purpose language extension, enabling sets creation using OCL code evaluated on the initial state of the system
2. Keyword creation to be used by Test Purposes. This mechanism enables to create generic Test Purposes, and to help for maintenance and reuse.
3. Capability to link a Test Purpose to a requirement identifier to ensure the traceability through the all test generation process.
4. Test Purpose catalogue import/export to reuse and apply Test Purposes on several systems under test.
5. Generation and Animation standalone Java API to enable fuzzed test sequences validation regarding the model.
6. A DSML allows the validation engineer to easily create the UML model. This DSML specifically addresses web-based applications.
7. A RASEN dedicated Wizard assist the user during the model creation, and pattern selection regarding the Risk analysis.
8. A RASEN publisher makes it possible to export the test result into the RASEN Security Dashboard.

Those features are presented in a more detailed way in the D4.2.3 RASEN deliverable.

3.2.2 Installation Guidelines

Smartesting test generation solution works with models edited by an Eclipse-based UML modeler. Smartesting provides a plug-in that checks whether the model fits the Smartesting CertifyIt restrictions on UML, and exports a model to be used by the Smartesting CertifyIt tool to generate the test cases. To use Smartesting test generation solution, both the Eclipse-based modeler plug-in for IBM RSA and the Smartesting CertifyIt software are needed.

IBM RSA Modeler Plug-in installation

1. This plug-in must be installed with the Eclipse update site tool. The following steps describe the installation of a new release.
2. Start RSA modeling tool.
3. From the tool menu, select Help->Software Updates->Find and Install...
4. A dialog opens. Select "Search for new features to install", and click "Next". Select "New Archived Site ...".
5. Specify the path to a .zip file available in the 'install' folder of the Smartesting CertifyIt installation directory. Once the path is selected, the following window should appear. You can use the 'Name' field in order to define your own Local Site name.
6. Click the "Finish" button of the "Install" window to launch the installation.
7. Check the Smartesting CertifyIt plug-in, and click « Next ».

   At this stage, you may have warnings indicating that some features cannot be installed (due to unavailable dependencies). In that case, check the “System Requirements and Supported Software”, and upgrade RSA modeler if required.
8. Accept terms of the license agreement, and click « Next ».
9. Click the « Finish » button to install the plug-in.

If verification messages appear, just accept those in order to complete the installation. It should be noted that write permissions for the install location are needed. Afterwards, restart RSA modeling workbench. The Smartesting CertifyIt plug-in should be successfully installed at this point.

Smartesting CertifyIt software with Security plugin installation

To install Smartesting CertifyIt software and the dedicated Security plugins, simply run the setup program and follow the installation instructions (this implies accepting the terms of the displayed license agreement).

Before using Smartesting CertifyIt, a license needs to be defined. Prerequisites are: Smartesting CertifyIt must be installed and the RSA Eclipse-based modeler plug-in must be installed. There are two types of licenses: floating license server or capacity-based license.

3.2.3 User Guide

This section introduces each step of the testing process supported by the tool.

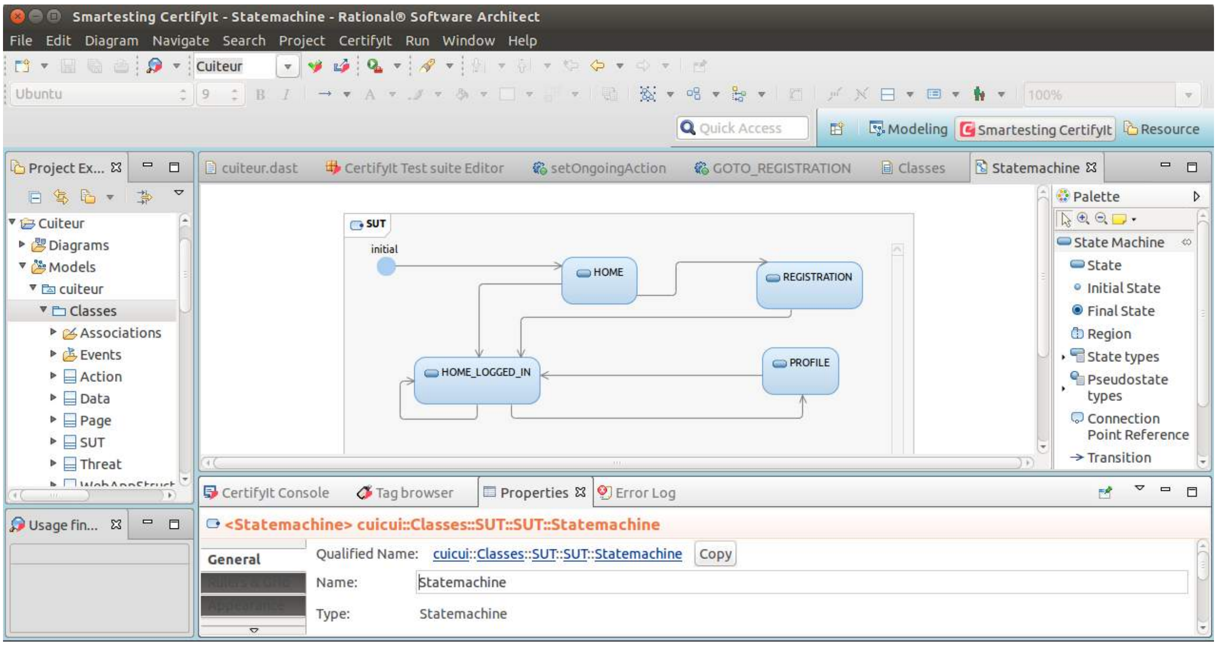
3.2.3.1 Modeling Activity

The first action in the process is Web application modeling. A domain-specific modeling language called DASTML enables to create textual UML models. Therefore, any text editor can be used for model design. Designing a DASTML model does not require any testing skills, and consists of doing a manual crawling of the Web Application under test to gather relevant information that will be used for vulnerability detection purposes.

The first step of this manual process should be about creating a “map” of the Web application, that is to say the page flow. This can be done by clicking on links and filling Web forms, the goal is to identify each unique page that requires testing (e.g., because it provides user interactions that are not purely navigational) or would be part of an attack process (e.g., a result page).

Then, the second step is to do reconnaissance on each page that has been modeled to gather information in order to design a DASTML model, in a text file. Once the model is complete, or a first subset of the Web application has been modeled, the file can be processed by RSA to translate its content in a UML test model.

Figure 6 shows the RSA interface, which has been has been extended with several plugins for testing purposes.


Smartesting CertifyIt - Statemachine - Rational® Software Architect

<Statemachine> cuicui::Classes::SUT::SUT::Statemachine

Qualified Name: cuicui::Classes::SUT::SUT::Statemachine

Name: Statemachine

Type: Statemachine

Figure 6 – IBM RSA Modeling Environment

The first plugin, Smartesting for RSA, enables to create UML test models, animate them, check their compliancy with the UML meta-model, and export them in a standardized XML file. It is used to manage generated UML entities and potentially make adjustments in special cases where experimented test engineers need to create new test purposes. It is also possible to animate models to check whether they are accurate regarding the Web application under test.

The second plugin, Test Purposes connector for RSA, provides a graphical interface to manage test purposes: creation, import / export, validity check. It is more discussed in the next section.

The third plugin, PMVT Connector for RSA, is responsible for the translation of DASTML models in UML test models. DASTML models are parsed using an ANTLR3 grammar, which creates an Abstract Syntax Tree (AST) out of it. Then, a dedicated algorithm visits the AST to create the corresponding UML entities on the fly. This is delivered to users of the solution in two ways.

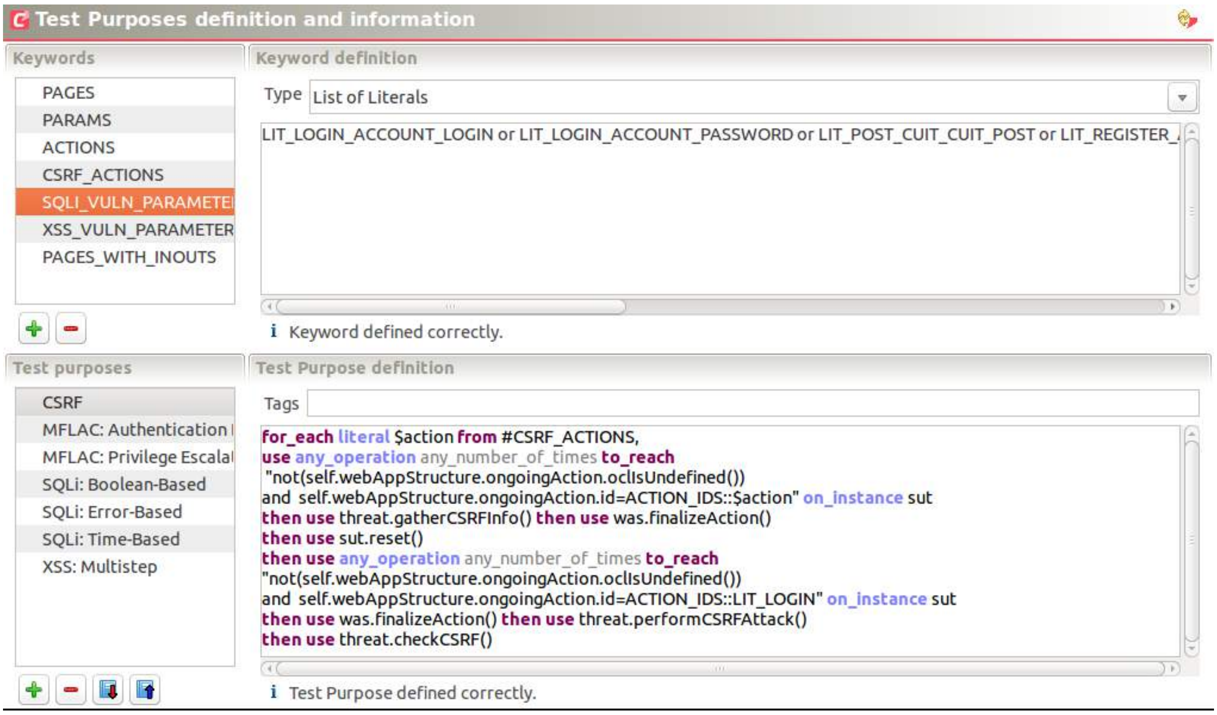
First, test engineers can choose to create a new modeling project. An eclipse wizard has been created and provides two inputs: a DASTML model file, and a test purpose catalog. From these inputs, the wizards automatically create the corresponding UML test models, and integrate the test purposes from the catalog. Second, it is possible to import a DASTML model in an existing project. However, model evolution is not yet supported at this point and test engineers must remove any previously generated entity before importing a new DASTML model.

Next step of the process is the selection and / or creation of test purposes, as described in the next section.

3.2.3.2 Test Purpose Activity

Once the model has been designed, test engineers have to proceed to test selection. This is done using test purposes. The design and selection of test purposes is also done through RSA. A graphical interface has been created, which is depicted in Figure 7. The upper part of the interface concerns keywords lists. Such list can contain a set of Enumeration literals, instances, states, behaviors, etc. Lists content is then used by test purposes iterators to compute test cases based on a refined set of model elements, therefore allowing test engineers to generate precise test suites that only address the selected elements. Note that all the visible keywords lists visible in the figure have been generated automatically.

The lower part of the interface is the test purpose editor. The left side lists all the created or imported test purposes, and on the right side is the actual editor. It is shipped with test purposes and OCL expressions on-the-fly syntax check.



Type List of Literals

LIT_LOGIN_ACCOUNT_LOGIN or LIT_LOGIN_ACCOUNT_PASSWORD or LIT_POST_CUIT_CUIT_POST or LIT_REGISTER_

ℹ Keyword defined correctly.

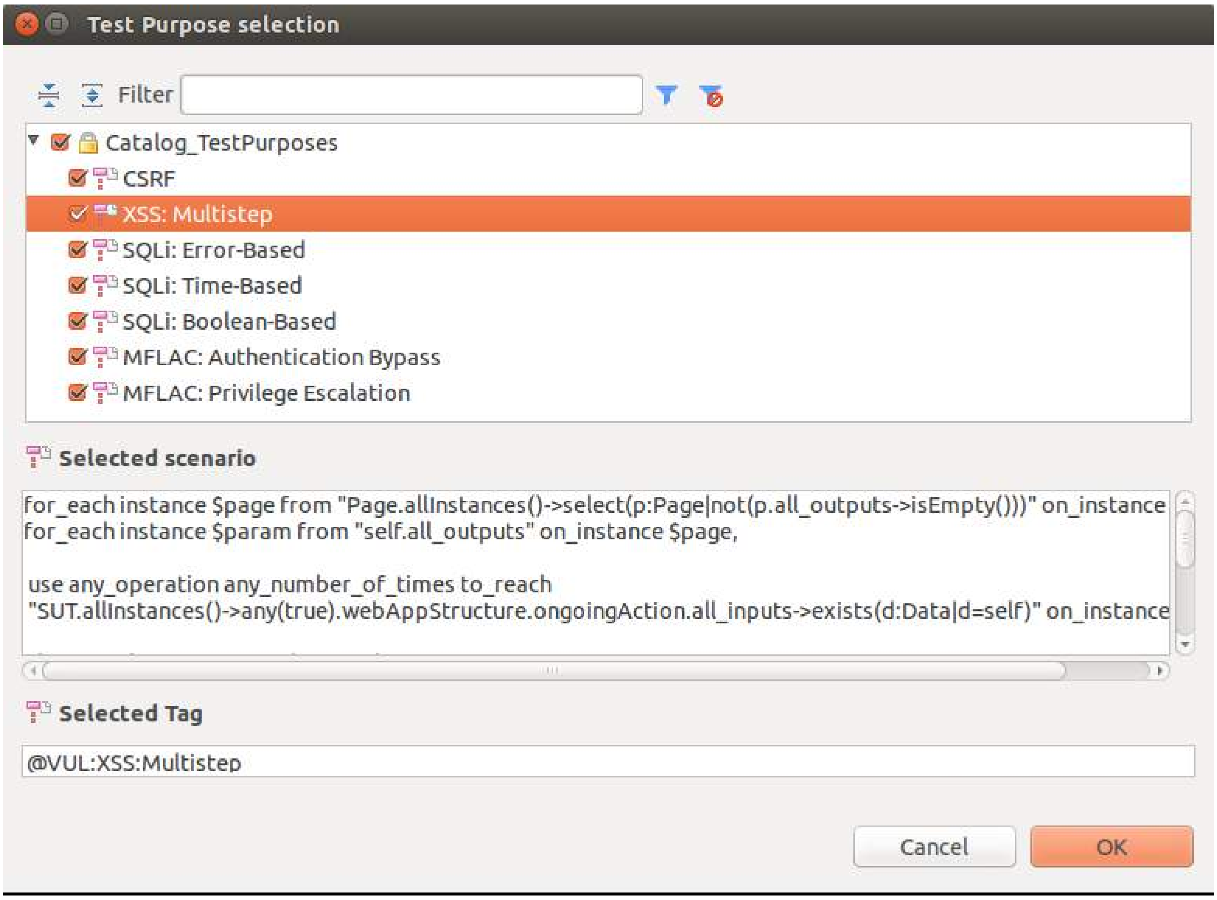
Tags

```
for_each literal $action from #CSRF_ACTIONS,
use any_operation any_number_of_times to_reach
 "not(self.webAppStructure.ongoingAction.oclIsUndefined())
and  self.webAppStructure.ongoingAction.id=ACTION_IDS::$action" on_instance sut
then use threat.gatherCSRFInfo() then use was.finalizeAction()
then use sut.reset()
then use any_operation any_number_of_times to_reach
"not(self.webAppStructure.ongoingAction.oclIsUndefined())
and  self.webAppStructure.ongoingAction.id=ACTION_IDS::LIT_LOGIN" on_instance sut
then use was.finalizeAction() then use threat.performCSRFAttack()
then use threat.checkCSRF()
```

ℹ Test Purpose defined correctly.

Figure 7 – Test Purpose Editor

Users can add and remove test purposes, and export their test purposes as a catalog in an XML file that is linked with CORAS. This way, they can directly import them from risk assessment. For that, we provide to users of the RASEN approach a predefined catalog containing the existing test purposes. Users can import all or part of this catalog. The test purposes stored this catalog are usable as is to test any Web application, without the need for adaptation. Figure 8 shows the interface for importing test purposes from the catalog in a test project.



☑ Catalog_TestPurposes
- ☑ CSRF
- ☑ XSS: Multistep
- ☑ SQLi: Error-Based
- ☑ SQLi: Time-Based
- ☑ SQLi: Boolean-Based
- ☑ MFLAC: Authentication Bypass
- ☑ MFLAC: Privilege Escalation

Selected scenario

```
for_each instance $page from "Page.allInstances()->select(p:Page|not(p.all_outputs->isEmpty()))" on_instance
for_each instance $param from "self.all_outputs" on_instance $page,

 use any_operation any_number_of_times to_reach
 "SUT.allInstances()->any(true).webAppStructure.ongoingAction.all_inputs->exists(d:Data|d=self)" on_instance
```

Selected Tag

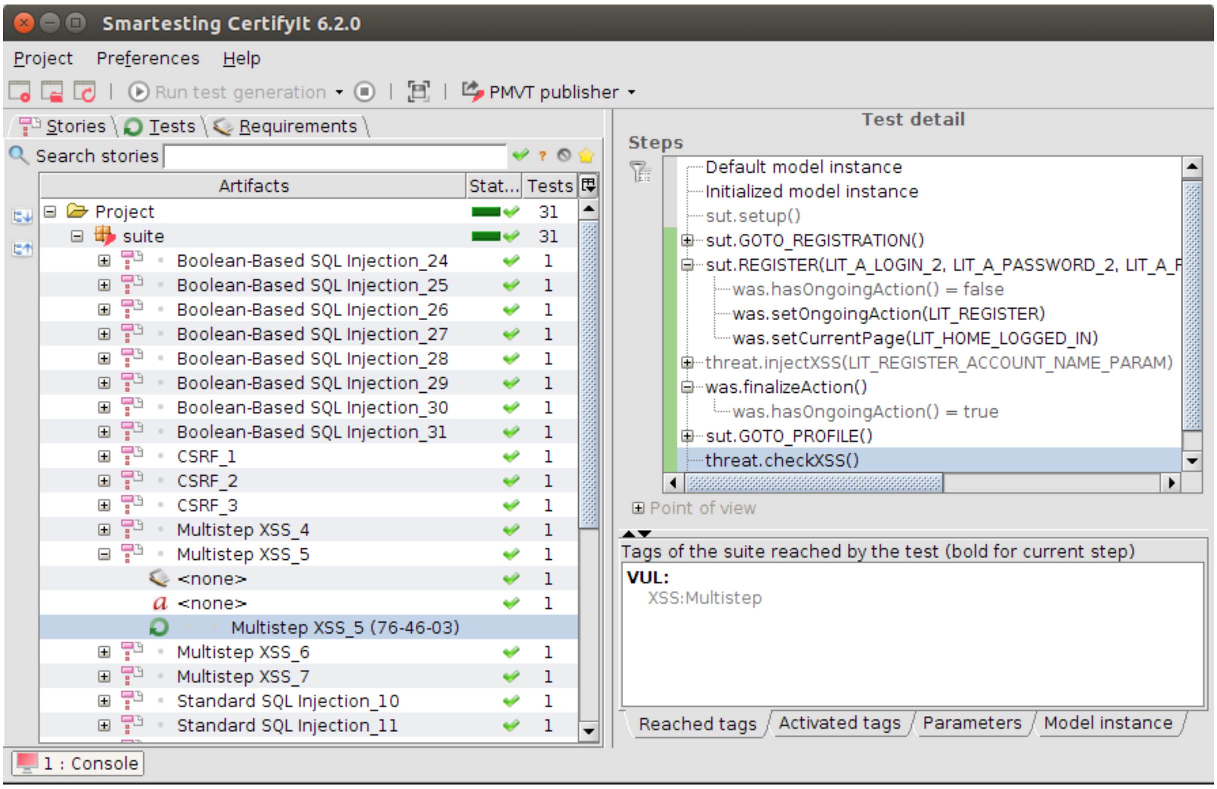
@VUL:XSS:Multistep

Figure 8 – Test Purpose Catalog

When the model and the test purposes have been defined, Test engineers can check the validity of their project. If it has been successfully checked, the next step is to export this project in a standardized XML file and pass it on to the test generation engine.

3.2.3.3 Test Generation Activity

The test generation engine that is used as part of the process is supported by the tool CertifyIt provided by Smartesting. This tool generates abstract test cases by finding paths in the test model to reach the test targets that have been derived from the test purposes. Figure 9 shows its interface.



Artifacts	Stat...	Tests
Project	✓	31
suite	✓	31
Boolean-Based SQL Injection_24	✓	1
Boolean-Based SQL Injection_25	✓	1
Boolean-Based SQL Injection_26	✓	1
Boolean-Based SQL Injection_27	✓	1
Boolean-Based SQL Injection_28	✓	1
Boolean-Based SQL Injection_29	✓	1
Boolean-Based SQL Injection_30	✓	1
Boolean-Based SQL Injection_31	✓	1
CSRF_1	✓	1
CSRF_2	✓	1
CSRF_3	✓	1
Multistep XSS_4	✓	1
Multistep XSS_5	✓	1
<none>	✓	1
<none>	✓	1
Multistep XSS_5 (76-46-03)		
Multistep XSS_6	✓	1
Multistep XSS_7	✓	1
Standard SQL Injection_10	✓	1
Standard SQL Injection_11	✓	1

```
Default model instance
Initialized model instance
sut.setup()
sut.GOTO_REGISTRATION()
sut.REGISTER(LIT_A_LOGIN_2, LIT_A_PASSWORD_2, LIT_A_F
    was.hasOngoingAction() = false
    was.setOngoingAction(LIT_REGISTER)
    was.setCurrentPage(LIT_HOME_LOGGED_IN)
threat.injectXSS(LIT_REGISTER_ACCOUNT_NAME_PARAM)
was.finalizeAction()
    was.hasOngoingAction() = true
sut.GOTO_PROFILE()
threat.checkXSS()
```

Tags of the suite reached by the test (bold for current step)

VUL:
XSS:Multistep

Figure 9 – Test Generation Environment

The left part lists all the test targets that the tool extracted from the model. If a test target is successfully reached, a test case is produced and all its steps are displayed on the right part of the tool.

CertifyIt interprets test purposes using a two-step process. First, during the importation of a test project, the test generation unfolds the test purposes: if a test purpose contains iterators, then the test generation engine creates a test target for each element of the list linked to the iterator. Second, the test generation engine tries to reach each test target by animating the model. When test generation is completed, test engineers must export the generated abstract test cases in executable test scripts, which are introduced in the next section.

3.2.3.4 Test Concretization and Execution Activities

Abstract test cases are not executable as is on the real system, therefore they must be concretized (adapted) and exported as test scripts. Test engineers must implement each operation from the model, provide concrete data to match each abstract data, and may need to develop a test harness in order to simplify test execution.

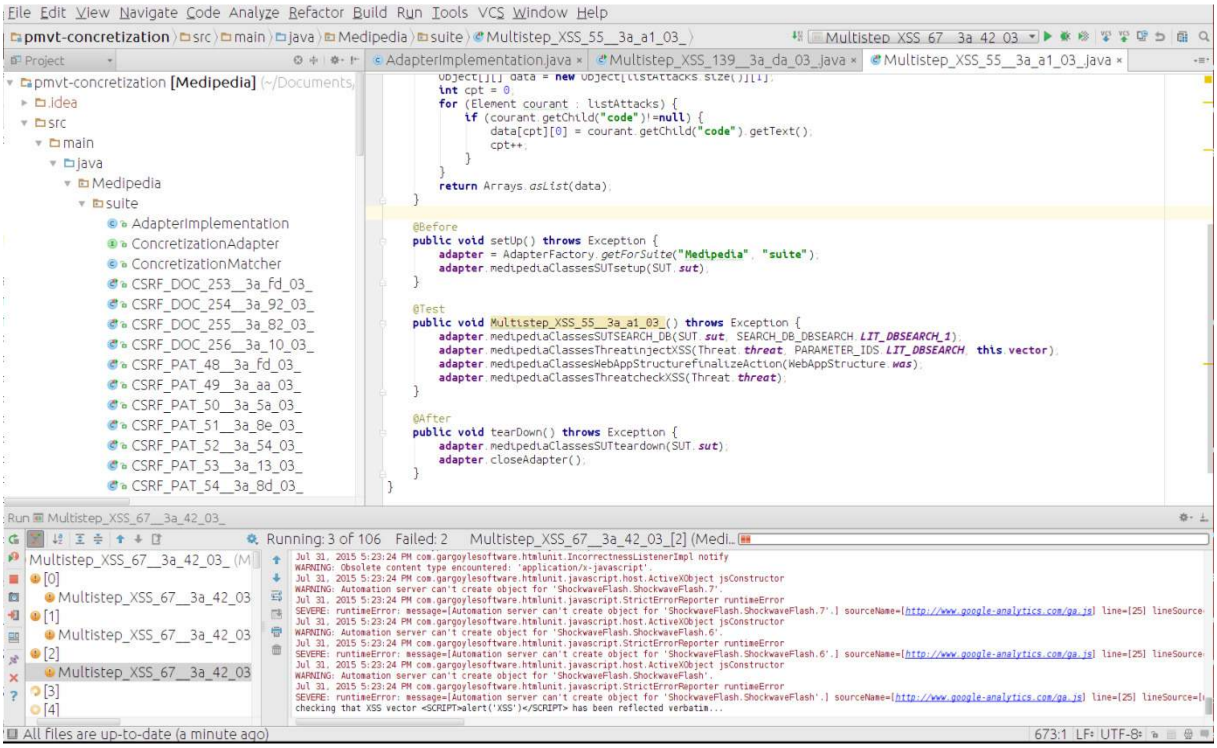
In the context of RASEN, generated abstract test cases are exported as JUnit test scripts. These scripts allow to automatically interacting with the Web application through its GUI, as a normal user would do. To accomplish this programmatically, test scripts use libraries such as Selenium[1] or HTMLUnit[2], which enable to operate a headless browser using primitives (i.e. methods such as filling form fields and clicking buttons / anchors).

Test concretization activity represents a considerable amount of work to obtain executable test scripts. A dedicated test publisher has therefore been designed in order to automate most of the process. When all abstract data have been matched with concrete data and all methods have been implemented, the obtained test scripts can be executed on the real Web application.

[1] http://www.seleniumhq.org/ [Last visited: August 2015]
[2] http://htmlunit.sourceforge.net/ [Last visited: August 2015]

Test execution and verdict assignment are fully automated, thanks to the test harness generated by the publisher. Test engineers execute the test suite, and collect the results.

Both the test concretization and execution activities are performed preferably using an IDE (in our case, IntelliJ Idea), as depicted in Figure 10. Such tool provides a graphical view of the test results, and an "export" function to store the results, for instance in an XML file. For instance, a RASEN exporter allows engineers to produce a XML files including test cases and results that can be imported into the RASEN Security Dashboard presented in section 0.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
pmvt-concretization › src › main › java › Medipedia › suite › Multistep_XSS_55__3a_a1_03_
Multistep XSS 67 3a 42 03
Project
AdapterImplementation.java ×
Multistep_XSS_139__3a_da_03_.java ×
Multistep_XSS_55__3a_a1_03_.java ×
pmvt-concretization [Medipedia] (~/Documents,
.idea
src
main
java
Medipedia
suite
AdapterImplementation
ConcretizationAdapter
ConcretizationMatcher
CSRF_DOC_253__3a_fd_03_
CSRF_DOC_254__3a_92_03_
CSRF_DOC_255__3a_82_03_
CSRF_DOC_256__3a_10_03_
CSRF_PAT_48__3a_fd_03_
CSRF_PAT_49__3a_aa_03_
CSRF_PAT_50__3a_5a_03_
CSRF_PAT_51__3a_8e_03_
CSRF_PAT_52__3a_54_03_
CSRF_PAT_53__3a_13_03_
CSRF_PAT_54__3a_8d_03_
Run Multistep_XSS_67__3a_42_03_
Running: 3 of 106 Failed: 2 Multistep_XSS_67__3a_42_03_[2] (Medi...
Multistep_XSS_67__3a_42_03_ (M
[0]
Multistep_XSS_67__3a_42_03
[1]
Multistep_XSS_67__3a_42_03
[2]
Multistep_XSS_67__3a_42_03
[3]
[4]
All files are up-to-date (a minute ago)
673:1 LF÷ UTF-8÷

```
        Object[][] data = new Object[listAttacks.size()][1];
        int cpt = 0;
        for (Element courant : listAttacks) {
            if (courant.getChild("code")!=null) {
                data[cpt][0] = courant.getChild("code").getText();
                cpt++;
            }
        }
        return Arrays.asList(data);
    }

    @Before
    public void setUp() throws Exception {
        adapter = AdapterFactory.getForSuite("Medipedia", "suite");
        adapter.medipediaClassesSUTsetup(SUT.sut);
    }

    @Test
    public void Multistep_XSS_55__3a_a1_03_() throws Exception {
        adapter.medipediaClassesSUTSEARCH_DB(SUT.sut, SEARCH_DB_DBSEARCH.LIT_DBSEARCH_1);
        adapter.medipediaClassesThreatinjectXSS(Threat.threat, PARAMETER_IDS.LIT_DBSEARCH, this.vector);
        adapter.medipediaClassesWebAppStructurefinalizeAction(WebAppStructure.was);
        adapter.medipediaClassesThreatcheckXSS(Threat.threat);
    }

    @After
    public void tearDown() throws Exception {
        adapter.medipediaClassesSUTteardown(SUT.sut);
        adapter.closeAdapter();
    }
}
```

```
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.IncorrectnessListenerImpl notify
WARNING: Obsolete content type encountered: 'application/x-javascript'.
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.host.ActiveXObject jsConstructor
WARNING: Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash.7'.
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.StrictErrorReporter runtimeError
SEVERE: runtimeError: message=[Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash.7'.] sourceName=[http://www.google-analytics.com/ga.js] line=[25] lineSource
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.host.ActiveXObject jsConstructor
WARNING: Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash.6'.
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.StrictErrorReporter runtimeError
SEVERE: runtimeError: message=[Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash.6'.] sourceName=[http://www.google-analytics.com/ga.js] line=[25] lineSource
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.host.ActiveXObject jsConstructor
WARNING: Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash'.
Jul 31, 2015 5:23:24 PM com.gargoylesoftware.htmlunit.javascript.StrictErrorReporter runtimeError
SEVERE: runtimeError: message=[Automation server can't create object for 'ShockwaveFlash.ShockwaveFlash'.] sourceName=[http://www.google-analytics.com/ga.js] line=[25] lineSource=[
checking that XSS vector <SCRIPT>alert('XSS')</SCRIPT> has been reflected verbatim...
```

Figure 10 – Test Concretization and Execution Environment

3.3 ARIS Business Architect – RASEN Plugin

3.3.1 Presentation

The ARIS Business Architect (ABA) is proprietary software from Software AG. A screenshot of the ABA with the different modeling aspects is depicted in Figure 11. It shows on the left hand side a list of all possible Common Weakness Enumerations (CWEs)[3] which can directly be aligned to a software component. On the right hand side (cf. Symbol Box) a selection of other components can be selected and drawn into the modeling pane in the center of the screen.

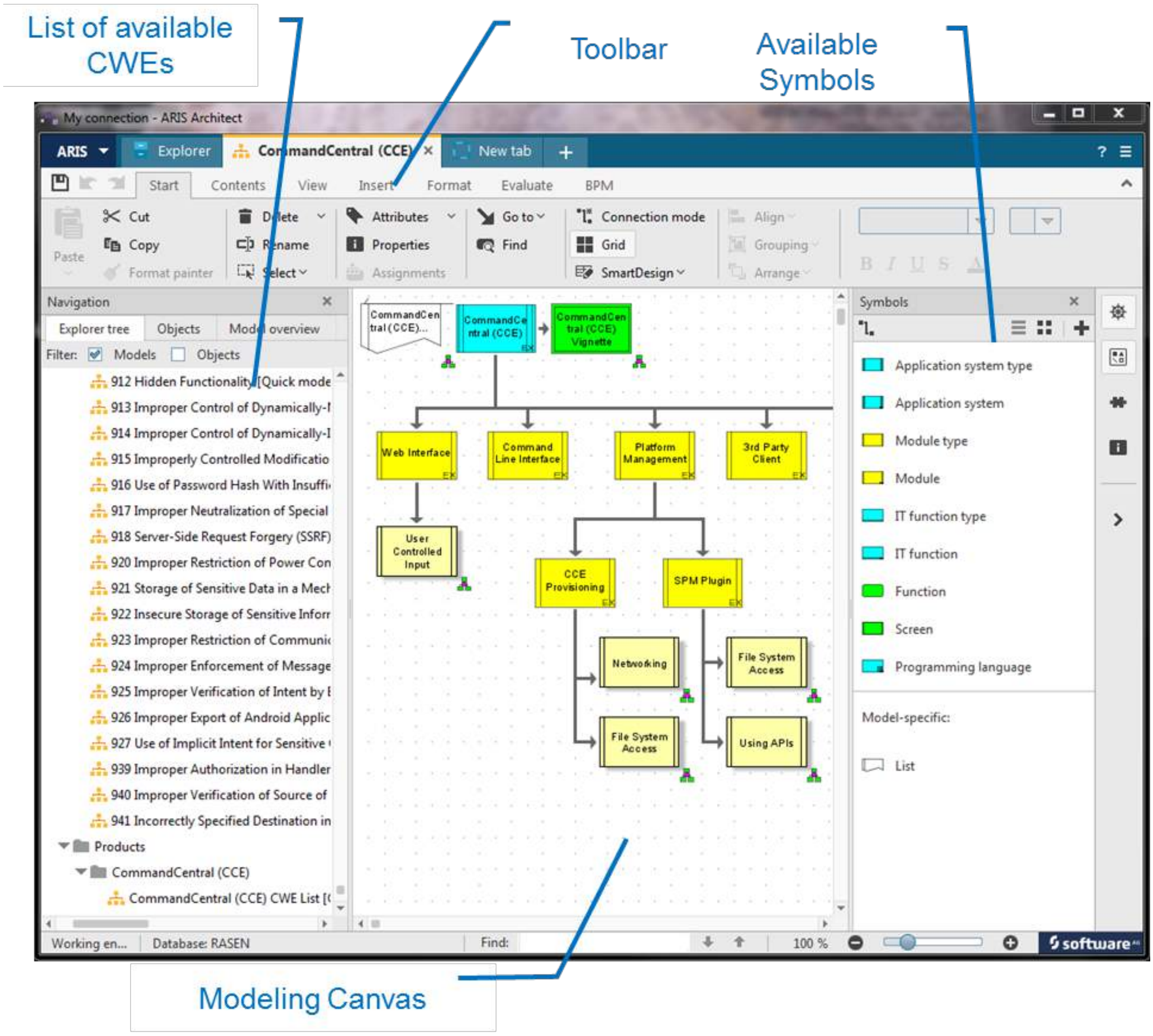

List of available CWEs
Toolbar
Available Symbols
Modeling Canvas
My connection - ARIS Architect
ARIS
Explorer
CommandCentral (CCE)
New tab
Start
Contents
View
Insert
Format
Evaluate
BPM
Paste
Cut
Copy
Format painter
Delete
Rename
Select
Attributes
Properties
Assignments
Go to
Find
Connection mode
Grid
SmartDesign
Align
Grouping
Arrange
Navigation
Explorer tree
Objects
Model overview
Filter: ☑ Models □ Objects
912 Hidden Functionality [Quick mode
913 Improper Control of Dynamically-I
914 Improper Control of Dynamically-I
915 Improperly Controlled Modificatio
916 Use of Password Hash With Insuffi
917 Improper Neutralization of Special
918 Server-Side Request Forgery (SSRF)
920 Improper Restriction of Power Con
921 Storage of Sensitive Data in a Mech
922 Insecure Storage of Sensitive Inforr
923 Improper Restriction of Communi
924 Improper Enforcement of Message
925 Improper Verification of Intent by I
926 Improper Export of Android Applic
927 Use of Implicit Intent for Sensitive
939 Improper Authorization in Handler
940 Improper Verification of Source of
941 Incorrectly Specified Destination in
Products
CommandCentral (CCE)
CommandCentral (CCE) CWE List [
CommandCentral (CCE)...
CommandCentral (CCE)
CommandCentral (CCE) Vignette
Web Interface
Command Line Interface
Platform Management
3rd Party Client
User Controlled Input
CCE Provisioning
SPM Plugin
Networking
File System Access
File System Access
Using APIs
Symbols
Application system type
Application system
Module type
Module
IT function type
IT function
Function
Screen
Programming language
Model-specific:
List
Working en...
Database: RASEN
Find:
100 %
software

Figure 11 – Screenshot of the Security Risk Assessment in RASEN model

3.3.2 Installation Guidelines

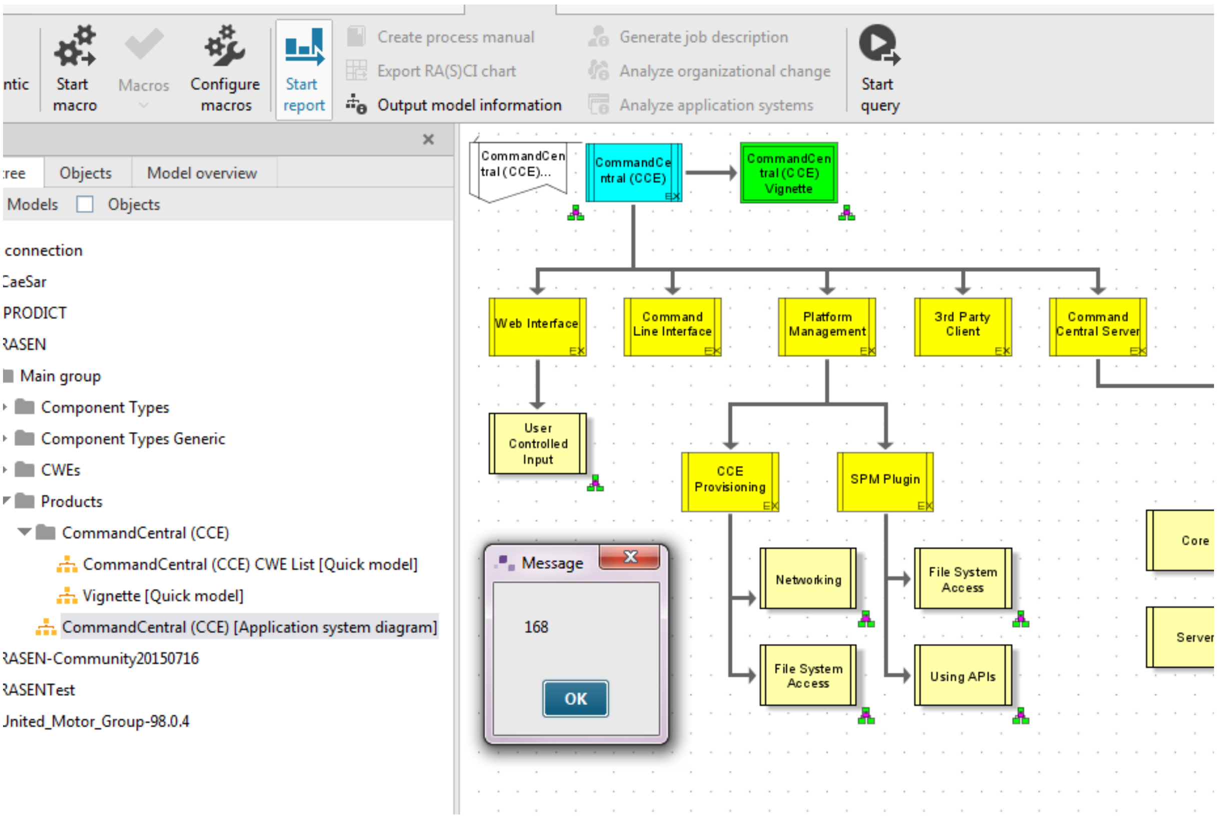
An individual installation program can be started using the provided Setup.exe which is guiding the used through the provided setup-routine. If system files are changed during installation, you are prompted to reboot your computer after installation. In addition to that there is a detailed installation guide of how to install the ARIS Business Architect on most spread operating systems.

[3] http://cwe.mitre.org

On top of the base installation of the ARIS Business Architect, the RASEN methodology is added by importing the RASEN artefacts from a ZIP file which contains the base package, consisting of the reports, the definition of necessary modeling elements, the macro, a preliminary set of already defined CWEs, and a predefined set of generic component types previously generated.

3.3.3 User Guide

The Product Model builds the base of the proposed modeling approach. The respective product is modeled as the root entry of the model; the corresponding components are modeled as child entries (cf. Figure 15). An exemplary RASEN model of the Comment Central Component is shown in the figure below, providing a full detailed description of the Software AG's product, the CCE Vignette, the component tree and the final CWSS scoring.



Message

168

Figure 12 – The final product model with the resulting CWSS Score (CCE Score 164)

At least one so called Component Template which specifies a special type of component is assigned to each component. Additionally a list of CWEs is assigned to the component. This list represents the union of all relevant CWE for this component which are automatically derived from the Component Template. This list can be modified in sense of deleting irrelevant CWEs or extending them according to the needs of the security expert doing the risk assessment.

The present model type represents a program under test as a set of components with their hierarchical relation. This relation is not restricted to one level but as several layers of sub-components building are feasible (subcomponents, sub-subcomponents, etc.), this can also be reflected within the model. Each component has a list of CWEs defined by the Component Template as denoted earlier. The initial content of these lists is defined by the connected Component Types provided as a Risk Template.

The product is now tested for all CWEs in its components lists. Afterwards the test results are imported into the tool and consequently all irrelevant CWEs where deleted from the property list. A CWE Model contains all relevant information as the ID, name and technical impacts about one CWE from the CWE database. The technical impacts where mapped to 8 technical impacts to fit the vignette schema and later on used for score computations. A clipping of the vignette is shown in the figure below.

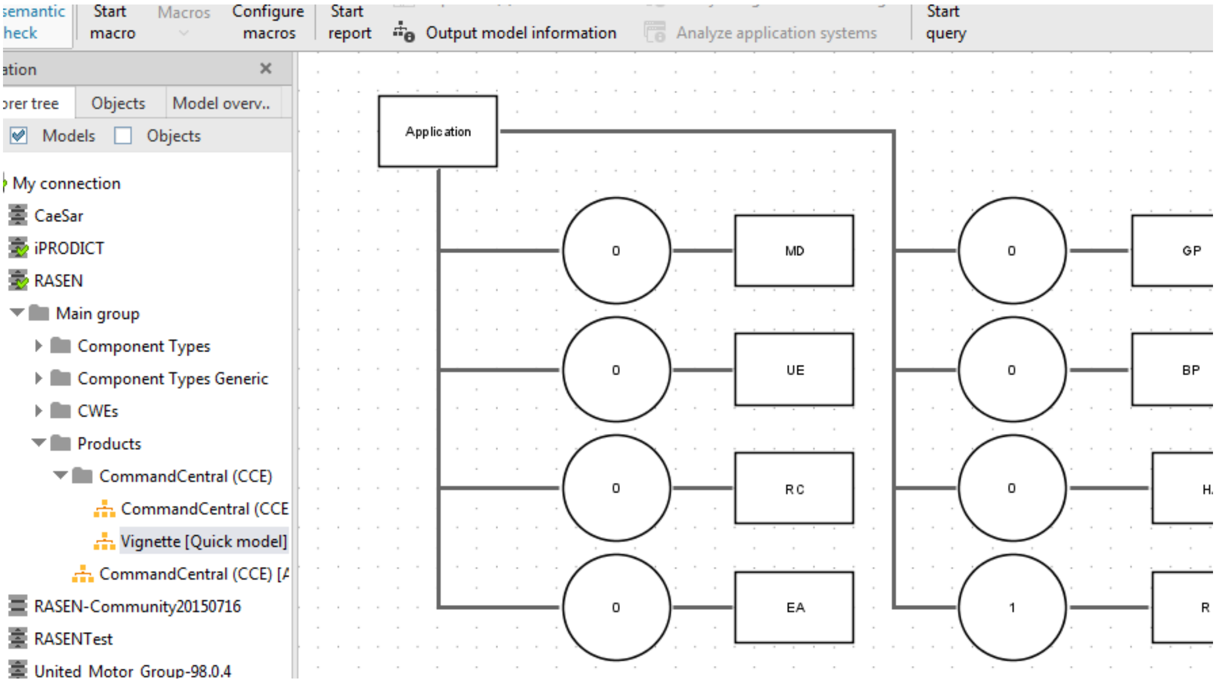


Figure 13 – The Product's Vignette containing information about the deployment scenario

The most frequently used components are defined as generic components which can be used as a kind of library to quickly instantiate a new product with components and accelerate the modelling effort. As shown in the figure below, each Generic Component Type consists of a set of assigned CWEs and the assignment follows best practices.

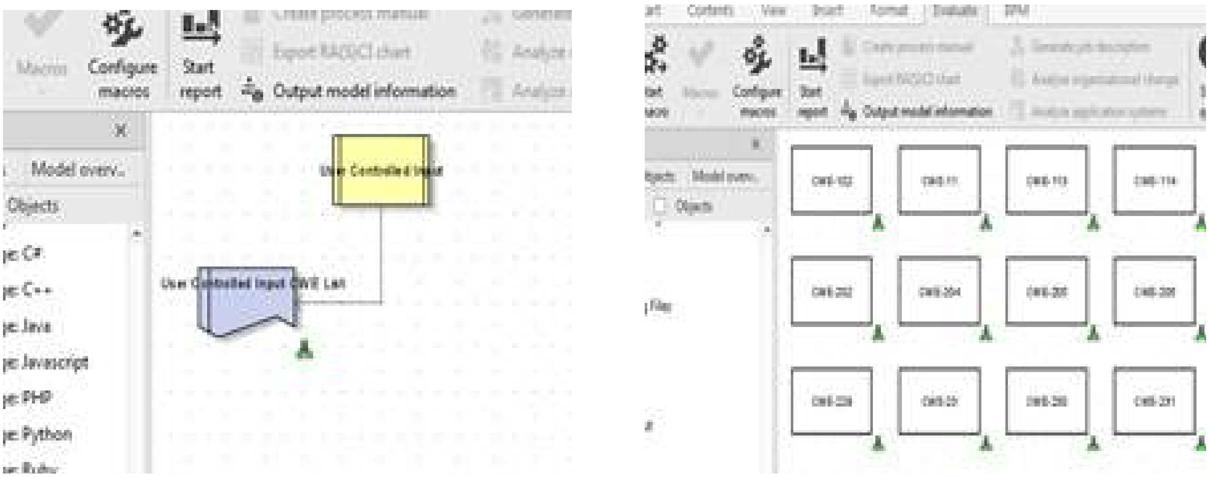


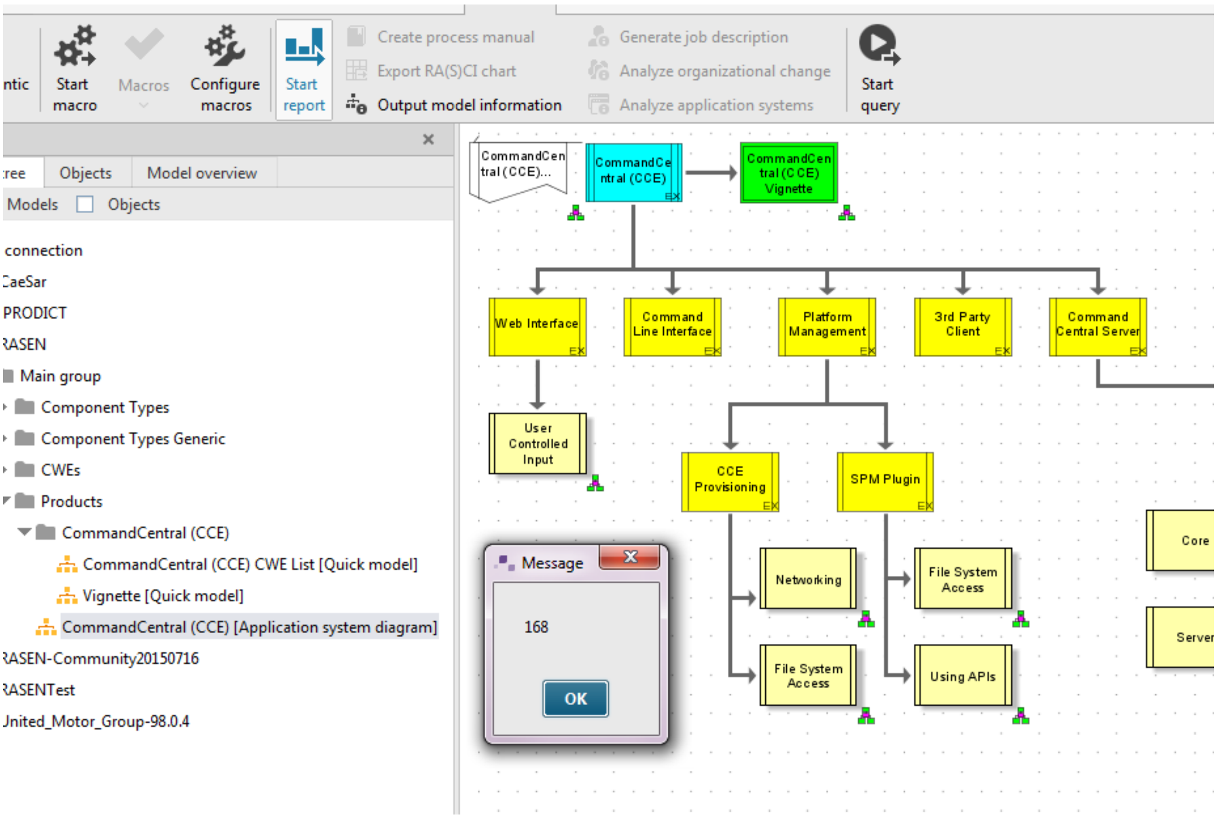
Figure 14 –ARIS Security Risk Assessment: the left the associated CWE list for the generic "Authentication" type; right: Clipping of CWEs linked to the "Authentication" type

The following functionality is implemented as either Reports (run on the ARIS server) or as Marcos (executed in the ARIS client):

- New Product – A wizard which supports the creation all necessary templates and directory for the security risk assessment of a new product

- New Component Type – A Wizard to support the creation of defining user-specific component types
- Import CWE Database --.Import and update the ARIS RASEN CWE database with inputs obtained from the MITRE CWE database
- Import Component Types – Import the set of generic component types from an external file or database
- Union Components – Compute the aggregation of all components
- Evaluate: The functionality which computes from the lower level aggregated and the vignette the CWSS score of the modeled product

An exemplary RASEN model of the Comment Central Component is shown in the figure below, providing a full detailed description of the Software AG's product, the CCE Vignette, the component tree and the final CWSS scoring.



Message

168

Figure 15 – The final product model with the resulting CWSS Score (CCE Score 168)

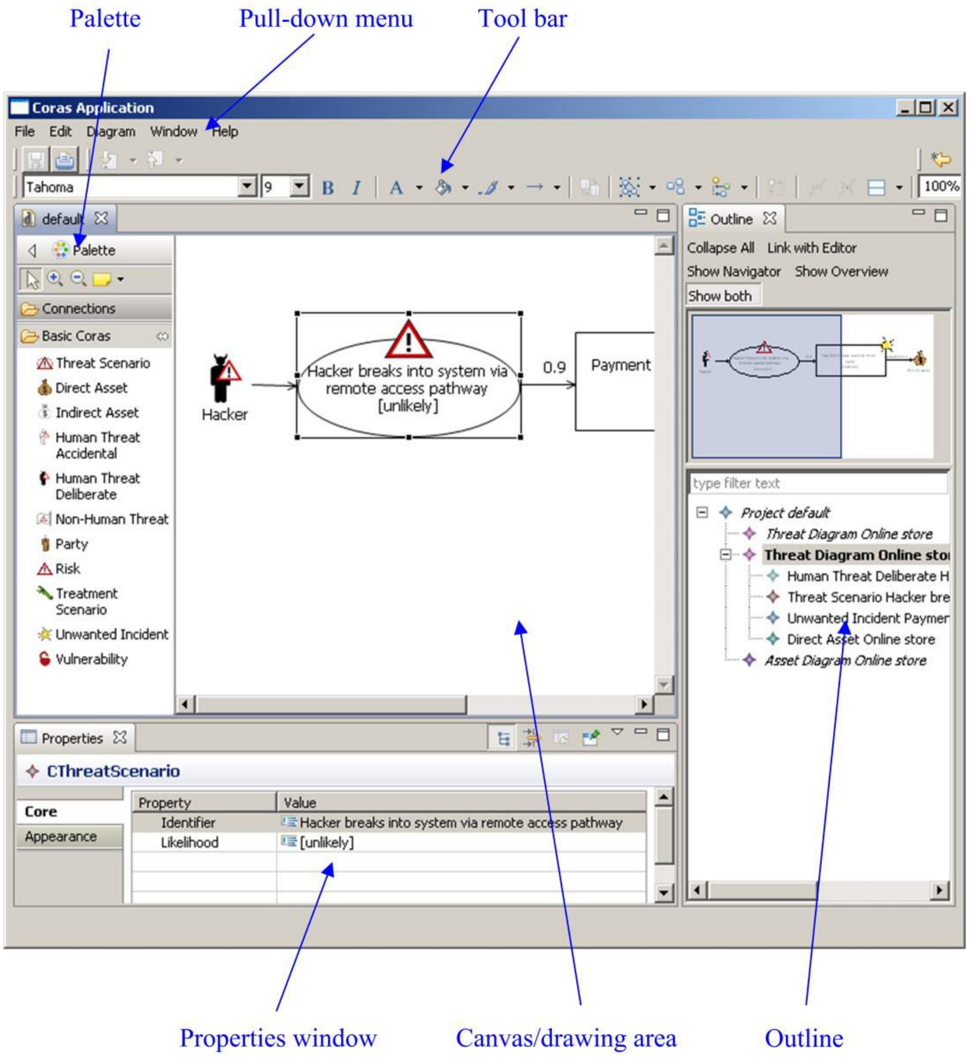
3.4 CORAS tool set

In this section we describe the CORAS tool set for model based risk assessment. The toolset consists of three stand-alone tools:

- The CORAS risk diagram editor (referred to as the CORAS tool for short). This is tool represents the core of the CORAS tool set. The other two tools may be seen as extensions of this core tool
- The Capec2Coras tool for automatically translation security attack patterns of the CAPEC catalog in to CORAS risk models
- The CorasAnalyzer for performing analysis functions on CORAS diagrams, and for combining CORAS risk diagrams with testing related activities.

In the following, we will in Section 3.4.1 describe the main functionality of the core Coras tool. Then, in Section 3.4.2 we provide installation guidelines of the three tools. Finally in Section 3.4.3 we provide user guidelines for the three tools.

3.4.1 Presentation



Property	Value
Identifier	Hacker breaks into system via remote access pathway
Likelihood	[unlikely]

Figure 16 – Screenshot of the CORAS tool

The CORAS tool is an open source diagram editor that supports the CORAS risk analysis language. The CORAS language is a graphical language whose constructs correspond to notions that are relevant during a risk analysis, e.g. threats, vulnerabilities, risks, unwanted incidents, threat scenarios and assets. The CORAS tool is intended to be used intensively during workshops where information is gathered through structured brainstorming. The tool is also intended to be used to document a risk analysis and to present the risk analysis results.

The CORAS tool is designed to support on-the-fly modeling using all five kinds of basic CORAS diagrams, thus facilitating the entire CORAS risk analysis process. A screenshot of the CORAS diagram editor is given in Figure 16. As indicated in the figure, the editor has six main parts:

- A pull-down menu that offers standard functions such as open, save, copy, cut, paste, undo and print.
- A tool-bar that offers easy access to the standard functions of the pull-down menu.
- A palette that contains the model elements and relations for drawing the diagrams.
- A drawing area or canvas for drawing the diagrams.
- A properties window that lists the properties of a selected element, and that can be used to edit the values of the properties.
- An outline that presents a project and its diagrams as a tree.

Except for the pull-down menu and the tool bar, all parts of the tool can be closed or hidden.

In the tool, a project is a collection of diagrams, and each diagram must belong to a project. A project must therefore be created before any diagrams are created.

The outline contains a tree representation of the project. The diagrams of the project are listed at the first level, and under each diagram all the diagram elements are listed. When a new element is created in the drawing area, it is automatically added to the tree under the correct diagram.

The drawing area is the part of the tool where the diagrams are made by inserting, editing, annotating and deleting elements. This is also where likelihoods and consequences are inserted to diagrams as part of the risk estimation, and it is also where risk levels are inserted as part of the risk evaluation.

3.4.2 Installation Guidelines

CORAS tool version 1.4:

- Download the zip-file containing the current version of the tool.
- Extract the zip-file to any folder of your choosing.
- Double click the file "Coras.exe" located in the Coras folder of the distribution.

Capec2Coras

- Download the zip-file containing the current version of the tool.
- Extract the zip-file to any folder of your choosing.
- Double click the file "Capec2Coras.exe" located in the Capec2Coras folder of the distribution.

Coras Analyzer

- Download the zip-file containing the current version of the tool.
- Extract the zip-file to any folder of your choosing.
- Double click the file "RUN_CORAS_ANALYZER.BAT" located in the Coras Analyzer folder of the distribution.

3.4.3 User Guide

3.4.3.1 Core CORAS Tool

Creating a new project

Before the CORAS tool can be properly used, a new project must be created. To create a new project:

- Select File -> New ->Coras Project in the File menu located near the top left corner of the window.
- Enter the desired file name and folder of the new project.
- Check “High Level CORAS”, “Dependent CORAS”, or “Legal CORAS” if you want to use these extensions of the basic CORAS language.
- Press “Finish” when done.

Creating a new diagram

The first time a new project is created, a new threat diagram called unnamed is automatically created within the project as shown in the outline view of the right hand side of the Figure 16. To create an additional diagram:

- Right-click the project entry in the tree outline located on the right hand side of the CORAS tool window.
- In the pull-down menu, select “new X diagram” (where X is the type of diagram that can be created e.g. threat, asset).
- The new diagram should appear in the tree outline. In order to open the new diagram, double click the diagram in the tree outline.

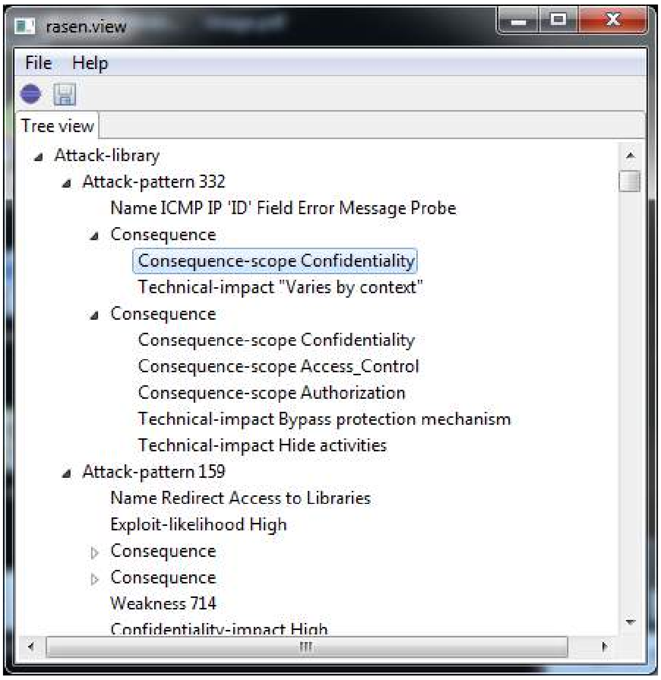
The difference between the diagrams is the kind of risk model elements that can be created within them. Note that the diagram type "Treatment" can contain (almost) all risk model elements; all other diagrams are subsets of this diagram (except the asset diagram).

Editing diagrams

Once a project and an appropriate diagram have been created, risk model elements can be added to the diagram by selection the appropriate element in the palette on the left hand side of the window, and left-clicking the diagram canvas. Relations can be created by selecting them from the pallet, or by holding the mouse over a source or target element until two small boxes appear on the border of the element. Holding down the left-mouse button on one of these small boxes will allow you to create a new relation.

3.4.3.2 Capec2Coras Tool

Capec2Coras is program that enables the user to transform an XML file containing CAPEC attack patterns into a CORAS file which can be read by the CORAS tool. The tool allow the CAPEC-file to be viewed and edited before being transformed into CORAS. Parameters which are needed by the transformation can also be configured.



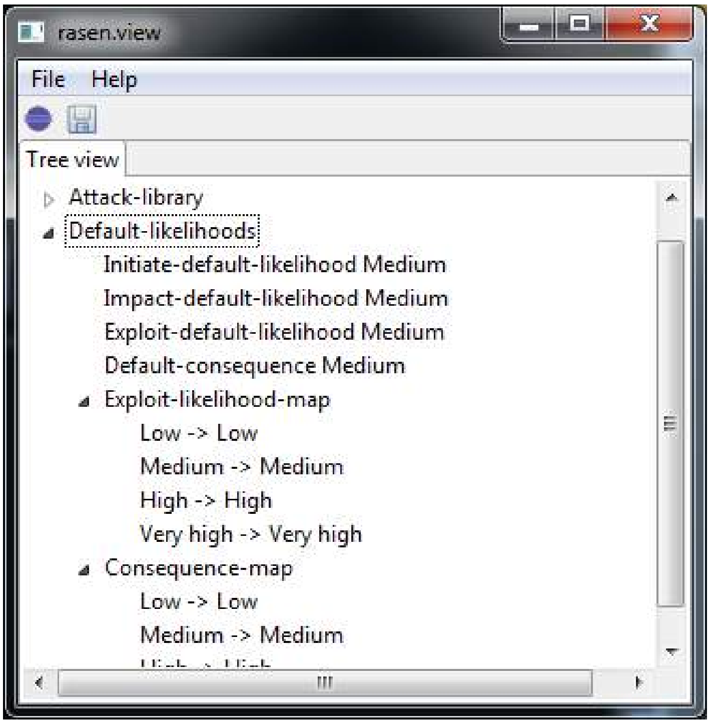
```
Attack-library
    Attack-pattern 332
        Name ICMP IP 'ID' Field Error Message Probe
        Consequence
            Consequence-scope Confidentiality
            Technical-impact "Varies by context"
        Consequence
            Consequence-scope Confidentiality
            Consequence-scope Access_Control
            Consequence-scope Authorization
            Technical-impact Bypass protection mechanism
            Technical-impact Hide activities
    Attack-pattern 159
        Name Redirect Access to Libraries
        Exploit-likelihood High
        Consequence
        Consequence
        Weakness 714
        Confidentiality-impact High
```

Figure 17 – Screenshot of Capec2Coras tool

After starting the Capec2Coras tool, the user is presented with a tree view as shown in Figure 17. This is where the CAPEC attack patterns and the parameters of the transformation can be edited. The tool has two main functions:

- Open: This function allows two kinds of files to be opened and displayed in the tree view of the Capec2Coras tool. The first kind of file is an XML-file containing a CAPEC library of attack patterns. The second kind is a file (not XML) which contains a previously edited version of the imported CAPEC library into the Capec2Coras tool.
- Save as: This function allows two kinds of saves (depending on the extension of the file being saved). The first kind transforms and saves the file into a CORAS-file (if the file extension ".coras_project" is used). The second kind saves the CAPEC library into a file so that he edits made to the file are not lost (if the file extension is not ".coras_project").

After a CAPEC library has been opened in the Capec2Coras tool, a tree view containing two top nodes is displayed as shown in Figure 18. These are "Attack-library" and "Default-likelihoods". The former contains all the attack-patterns which are defined in the CAPEC library, the second contains values which are used as parameters by the transformation. Specifically, default values which are produced by the transformation to CORAS can be configured as shown in Figure 18.



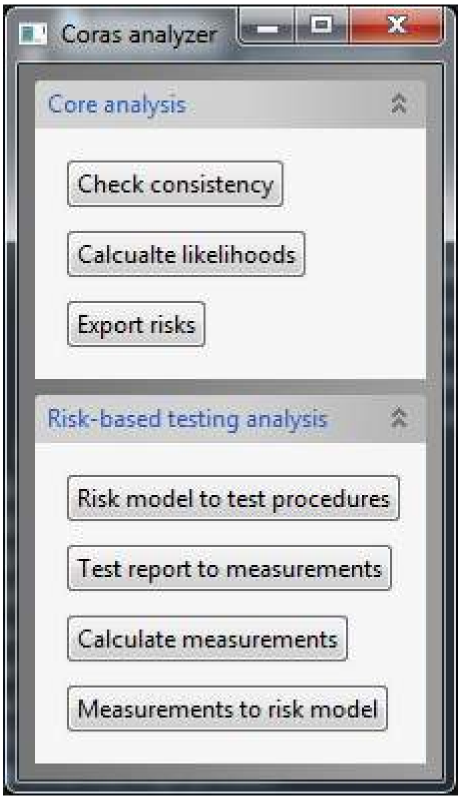
Attack-library
Default-likelihoods
Initiate-default-likelihood Medium
Impact-default-likelihood Medium
Exploit-default-likelihood Medium
Default-consequence Medium
Exploit-likelihood-map
Low -> Low
Medium -> Medium
High -> High
Very high -> Very high
Consequence-map
Low -> Low
Medium -> Medium
High -> High

Figure 18 – Screenshot showing transformation parameters

3.4.3.3 CorasAnalyzer Tool

CorasAnalyzer is a program that can be used to analyze Coras risk models. A screenshot of the tool is shown in Figure 19. Here we can see that the analysis functions are separated into two areas:

- *Core analysis* allows the user to perform the basic analysis functions: Check consistency, calculate likelihoods, and export risks.
- Risk-based-testing functions which allow the user to generate prioritized test procedures from a risk model and to aggregate test results to the risk model. Specifically, the following analysis features are supported: "Risk model to test procedures", "Test report to measurements", "Calculate measurements", and "Measurements to risk model".



Coras analyzer

Core analysis

Check consistency

Calcualte likelihoods

Export risks

Risk-based testing analysis

Risk model to test procedures

Test report to measurements

Calculate measurements

Measurements to risk model

Figure 19 – Screenshot showing main Coras analysis window

In the following, we describe each analysis feature in more detail.

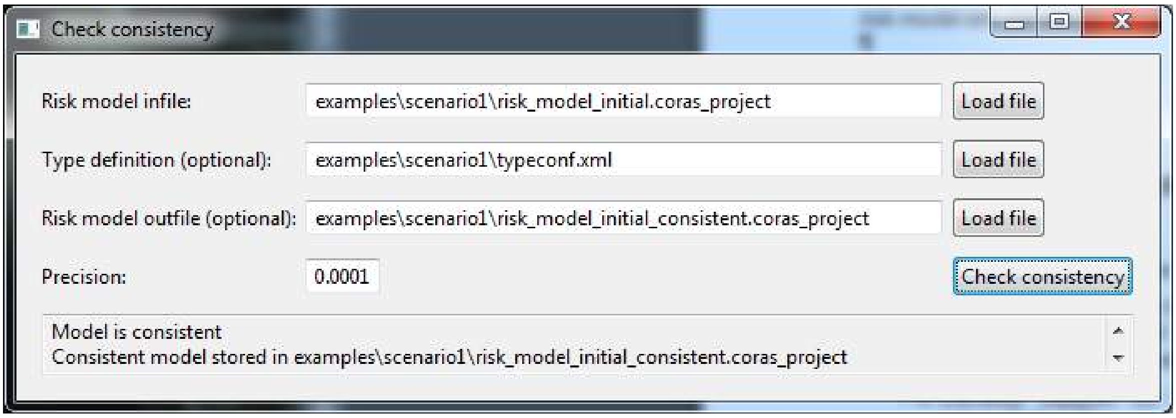
3.4.3.3.1 Check Consistency

The check consistency feature allows the user to check whether the likelihood estimates of a Coras risk model are consistent. A screenshot of the check consistency feature is shown in Figure 20. Here we can see that the function takes three files as input (two of which are optional) in addition to one numeric parameter:

- Risk model in file: Refers to a Coras risk model file (with the .coras_project extension) whose consistency we want to check.
- Type definition (optional): Refers to an xml file defining the types of the likelihood values of the risk model. See Section 3.4.3.3.8 for more details.
- Risk model outfile (optional): This refers to a Coras risk model file which will contain an example of a consistent risk model which has been generated on the basis of the input risk model.
- Precision: This parameter specifies the precision with which consistency is checked.

By pressing the "Check consistency" button, the tool will load the risk model in file and check whether its likelihood values are consistent up to the specified level of precision. If the model is found to be consistent, and the risk model out file is specified, the tool will store an example of a consistent risk model to that file.

Example: The check consistency function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml which are located in the CorasAnalyzer distribution.



examples\scenario1\risk_model_initial.coras_project

examples\scenario1\typeconf.xml

examples\scenario1\risk_model_initial_consistent.coras_project

0.0001

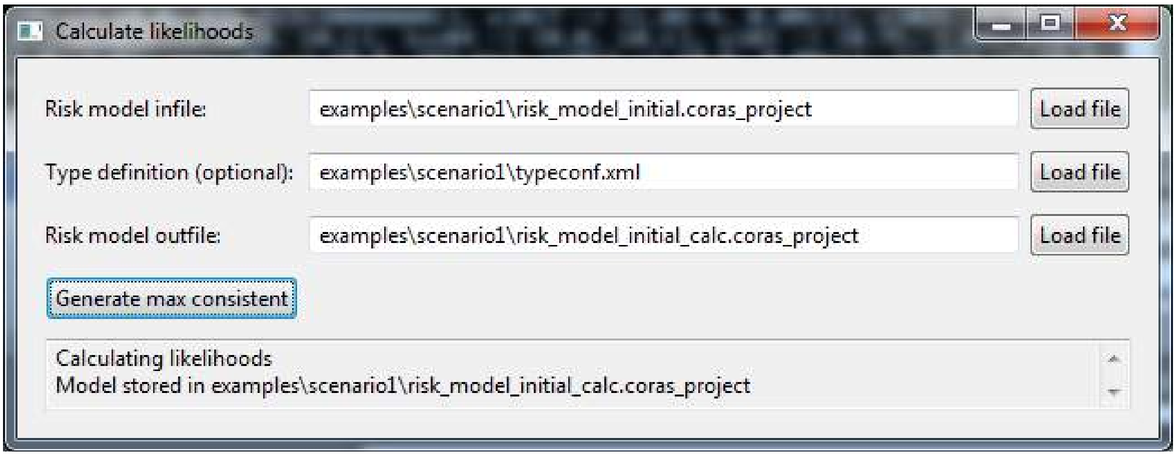
```
Model is consistent
Consistent model stored in examples\scenario1\risk_model_initial_consistent.coras_project
```

Figure 20 – Screenshot of check consistency feature

3.4.3.3.2 Calculate Likelihoods

The calculate likelihoods function allows the likelihoods of a risk model which have not been estimated to be derived based on likelihood values that have been estimated. In addition, the function ensures that likelihood estimate ranges never include values which would always lead to inconsistency. As shown in Figure 21, the function takes three input values

- Risk model in file: Refers to the Coras risk model whose likelihood values will are to be calculated.
- Type definition (optional): Refers to an xml file defining the types of the likelihood, consequence, and risk values of the risk model. See Section 3.4.3.3.8 for more details.
- Risk model out file: Refers to the Coras risk model where the results of the likelihood calculation will be stored.



examples\scenario1\risk_model_initial.coras_project

examples\scenario1\typeconf.xml

examples\scenario1\risk_model_initial_calc.coras_project

```
Calculating likelihoods
Model stored in examples\scenario1\risk_model_initial_calc.coras_project
```

Figure 21 – Screenshot of the Calculate likelihoods function

If the "Generate max consistent" button is pressed, the tool will read the risk model in file and the type definition file (if any), calculate the likelihoods, and store the results in the risk model out file.

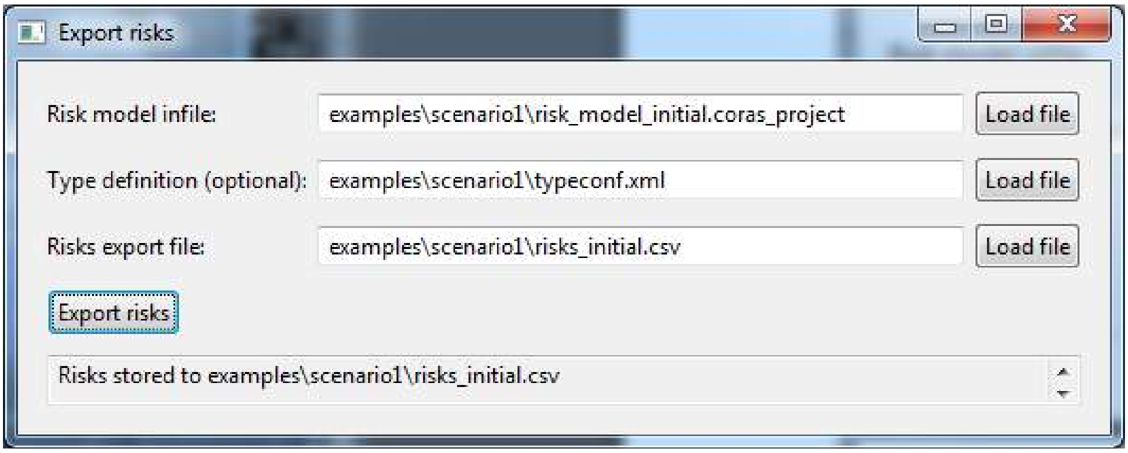
Example: The calculate likelihoods function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml which are located in the CorasAnalyzer distribution.

3.4.3.3.3 Export Risks

The export risks function allows the Coras risk model to be exported to comma-separated file containing all relevant information about all risk risks in the risk model. A risk in a Coras risk model is an unwanted incident which is connected to an asset. The function assumes that each unwanted incident constituting a risk is connected to exactly one asset. As shown in Figure 22, the export risks function takes three parameters:

- Risk model in file: Refers to the Coras risk model file whose risks will be exported
- Type definition (optional): Refers to an xml file defining the types of the likelihood values of the risk model. See Section3.4.3.3.8 for more details.
- Risks export file: Refers to the file where the risks will be stored. The file should have the extension .csv, since the risks are exported as comma-separated text file.

If the user presses the "Export risks" button, to tool will read the risk model infile and the type definition file (if any), calculate the risk values, and export the risks with the risk values to the risks export file in a comma separated text format.



Risk model infile: examples\scenario1\risk_model_initial.coras_project

Type definition (optional): examples\scenario1\typeconf.xml

Risks export file: examples\scenario1\risks_initial.csv

Risks stored to examples\scenario1\risks_initial.csv

Figure 22 – Screenshot of export risks function

Example: The export risks function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml which are located in the CorasAnalyzer distribution.

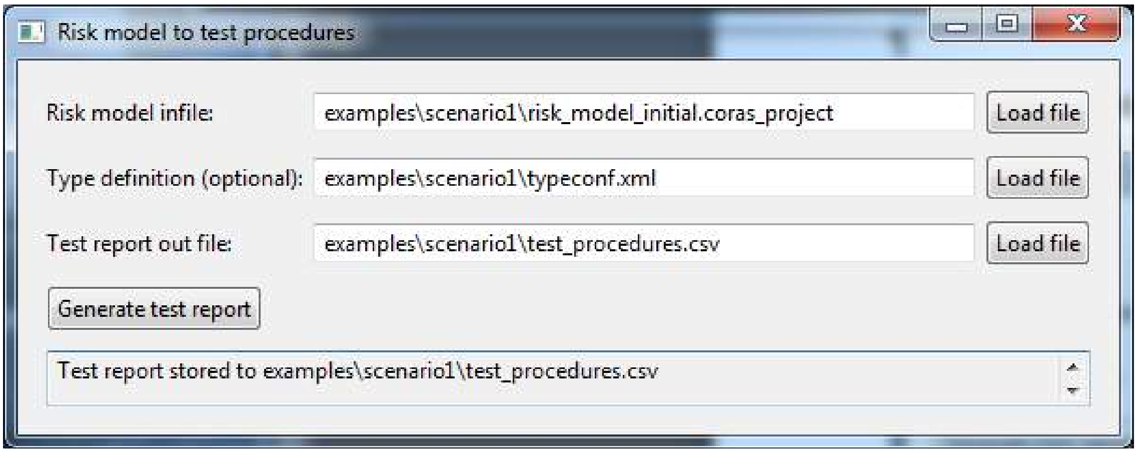
3.4.3.3.4 Risk Model to Test Procedures

The risk model to test procedures function allows the user to generate a prioritized list of test procedure from a Coras risk model. These test procedures are meant to be used as a starting point for a test design and implementation activity. As shown in Figure 23, the function takes three input parameters:

- Risk model in file: Refers to the CORAS risk model on the basis of which the test procedures will be generated.
- Type definition (optional): Refers to an xml file defining the types of the likelihood, consequence, and risk values of the risk model. See Section 3.4.3.3.8 for more details.
- Test report out file: Refers to the file where the test procedures will be stored. Two different file formats are supported: If the out file ends with the .csv extension, then a comma-separated text file will be generated. Otherwise the test procedures will be stored in an XML file which conforms to the RASEN Test report XSD schema described in deliverable D5.4.3. This file

may be seen as an initial empty test report which can be populated with actual test results in the test execution phase.

If the user presses the button "Generate test report", then the tool will read the risk model in file and the type definition file (if any), generate a prioritized list of test procedures, and store the results to the test report out file.



Risk model infile: examples\scenario1\risk_model_initial.coras_project

Type definition (optional): examples\scenario1\typeconf.xml

Test report out file: examples\scenario1\test_procedures.csv

Test report stored to examples\scenario1\test_procedures.csv

Figure 23 – Screenshot of risk model to test procedures function

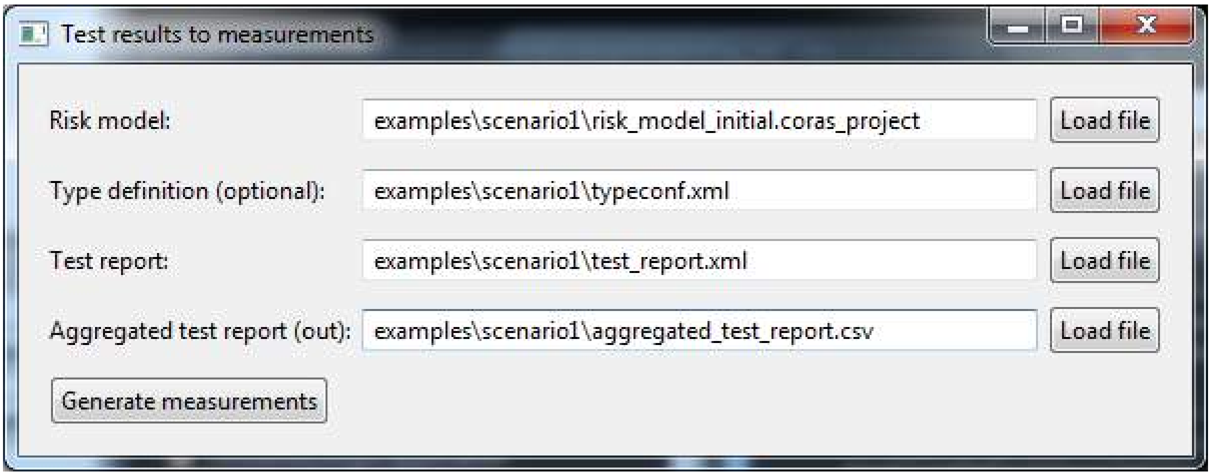
Example: The "risk model to test procedures" function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml which are located in the CorasAnalyzer distribution.

3.4.3.3.5 Test Report to Measurements

The "test report to measurements" function takes a test report as input, and generates an aggregated test report which contains measurements that can be used for the purpose of updating a risk model. As shown in Figure 24, the function takes four arguments:

- Risk model: Refers to a Coras risk model which was used to produce an initial version of the test report (using the risk model to test procedures function described in Section Figure 23.
- Type definition (optional): Refers to an xml file defining the types of the likelihood, consequence, and risk values of the risk model. See Section 3.4.3.3.8**Error! Reference source not found.** for more details.
- Test report: An XML file containing the test results on the basis of which the aggregated test report with measurements will be generated. The test report must confirm to the Test_Report XSD schema specified in deliverable D5.4.3.
- Aggregated test report (out): Refers to the file where the aggregated test report with the measurements is stored. Two formats are support. If the file ends with ".csv", the aggregated test report will be stored in a comma-separated text file. Otherwise the aggregated test report will be stored in an XML file which conforms to the Aggregated_Test_Report XSD schema specified in deliverable D5.4.3.

If the user presses the "Generate measurements" button, the tool will load the risk model, the test report, and the type definition file (if any), generate the measurements, and stored these as an aggregated test report in the measurement file. Note that the reason the risk model is loaded as opposed to only the test report, is that some of the measurements may be generated on the basis of information contained in the risk model (but not in the test report).



Test results to measurements

Risk model: examples\scenario1\risk_model_initial.coras_project

Type definition (optional): examples\scenario1\typeconf.xml

Test report: examples\scenario1\test_report.xml

Aggregated test report (out): examples\scenario1\aggregated_test_report.csv

Figure 24 – Screenshot of the test results to measurements function

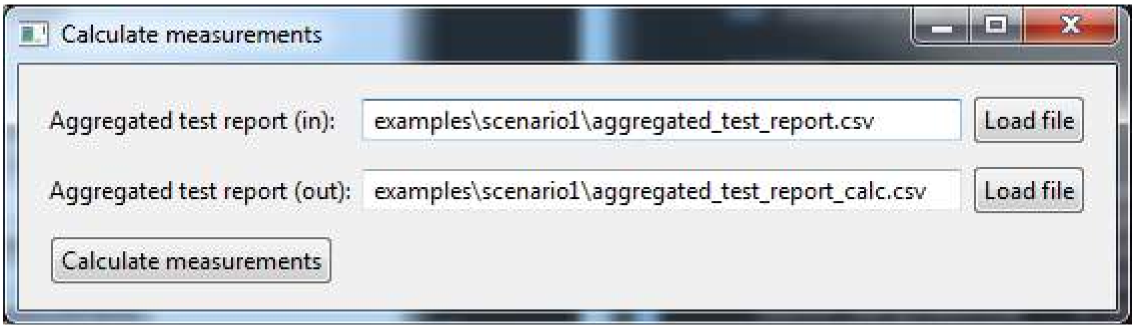
Example: The "test results to measurements" function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml, and the test report file examples/scenario1/test_report.xml" which are located in the CorasAnalyzer distribution.

3.4.3.3.6 Calculate Measurements

The "calculate measurements" function takes measurements of an aggregated test report as input, and based on some of these measurements, derives new measurements which are needed in order to update the a risk mode, and stores these together with the input measurements into a new aggregated test report file. As shown in Figure 25, the function takes two arguments

- Aggregated test report (in): The aggregated test report containing the measurements that are the basis for the derivation/calculation of new measurements. Two file formats are supported. If the file ends with the .csv extension, then it assumed that the aggregated test report is in a comma separated text file. Otherwise, the tool assumes that aggregated test report is in an XML file conforming to the Aggregated_Test_Report XSD schema documented in deliverable D5.4.3.
- Aggregated test report (out): Refers to the file where the new derived measurements together with the input measurements will be stored. As with the input file, xml and csv file formats are supported.

If the "calculate measurements" button is pressed, the tool will read the aggregated test report in file, calculate new measurements, and store these together with the input measurements to the out file.



Calculate measurements

Aggregated test report (in): examples\scenario1\aggregated_test_report.csv

Aggregated test report (out): examples\scenario1\aggregated_test_report_calc.csv

Figure 25 – Screenshot of the calculate measurements function

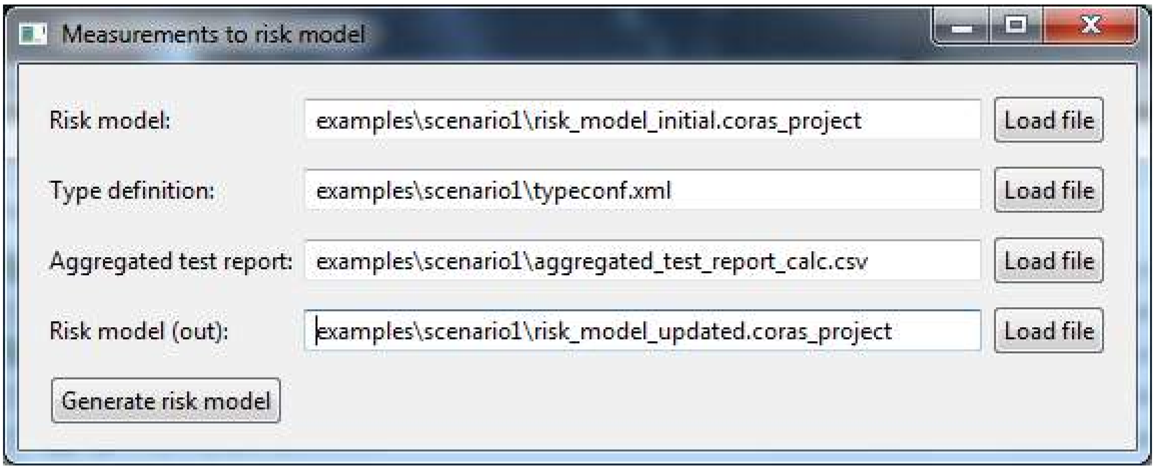
Example: The "calculate measurements" function can be tested on the aggregated test report in file examples/scenario1/aggregated_test_report.csv which is located in the CorasAnalyzer distribution.

3.4.3.3.7 Measurements to Risk Model

The "measurements to risk model" function takes a risk model and an aggregated test report as input, and updates the likelihood estimates of the risk model based on the measurements contained in the aggregated test report. As shown in Figure 26, the function takes the following parameters:

- Risk model: Refers to a Coras risk model which contains the likelihood estimates that may be updated.
- Type definition (optional): Refers to an xml file defining the types of the likelihood, consequence, and risk values of the risk model. See Section 3.4.3.3.8 for more details.
- Aggregated test report: An XML or CSV file an aggregated test report with the measurements that will be used to update (some of) the likelihood values of the risk model. This file should be generated using the "Calculate measurements" function described in Section 3.4.3.3.6.
- Risk model (out): Refers to the CORAS file where the updated risk model will be stored.

If the "Generate risk model" button is pressed, the tool will read the input risk model, the type definition, and the aggregated test report, update the likelihood values of the risk model based on the measurements of the aggregated test report, and store the results in the risk model out file.



Risk model: examples\scenario1\risk_model_initial.coras_project

Type definition: examples\scenario1\typeconf.xml

Aggregated test report: examples\scenario1\aggregated_test_report_calc.csv

Risk model (out): examples\scenario1\risk_model_updated.coras_project

Figure 26 – Screenshot of the measurements to risk model function

Example: The "test results to measurements" function can be tested on the risk model in file examples/scenario1/risk_model_initial.coras_project and the type definition file examples/scenario1/typeconf.xml, and the test aggregated report file examples/scenario1/aggregated_test_report_calc.csv" which are located in the CorasAnalyzer distribution.

3.4.3.3.8 The Type Configuration File

The type configuration file is an XML will which contains list of *type definitions*, and list of *type assignments* to elements of the risk model. The main purpose is to specify whether frequency or probability estimates should be used, the specify the risk function kind, and to specify precisely defined human readable qualitative values such as "Low", "High", etc. to numeric values which are need for automated analysis.

Each type definition specifies:

- The *name* of the type.
- A *basetype*, where the following values are possible: Frequency, Probability, Real, RiskFunctionAddition, RiskFunctionProduct. The latter two specifies whether the risk level be calculated by adding or multiplying the likelihood and consequence values that constitute the risk.
- A list of literal definition which map qualitative human readable values such as "Low", "High" into specific values. A literal definition consists of a *name*, a *description*, and a value definition.

An example of a type definition is shown below:

```
<typedef name="NodeLikelihoodType" basetype="Frequency">
<literal name="Seldom" description="Attack initiation scale">
      <value><intervalReal min="0.0" max="0.1"/></value>
</literal>
<literal name="Unlikely" description="Attack initiation scale">
      <value><intervalReal min="0.1" max="1.0"/></value>
</literal>
<literal name="Possible" description="Attack initiation scale">
      <value><intervalReal min="1.0" max="13.0"/></value>
</literal>
<literal name="Probable" description="Attack initiation scale">
      <value><intervalReal min="13.0" max="61.0"/></value>
</literal>
<literal name="Certain" description="Attack initiation scale">
      <value><intervalReal min="61.0" max="10000.0"/></value>
</literal>
</typedef>
```

In this example, we define a type with name NodeLikelihoodType, and basetype Frequency. In addition, there are five literals defined with names such as "Seldom", "Possible", etc. If these names are used in the risk model, they will be automatically replaced by their value as defined by the literals during analysis by the tool. As in the example, literals are usually defined as an interval with a minimum and maximum value.

The type assignments assign types to elements of the risk model. Each type assignment consists of:

- a *name*defining the kind of risk element that will be assigned a type, where the following values are possible: NodeLikelihood (referring to likelihood values of nodes in the risk model), EdgeLikelihood(referring to likelihood values of edges in the risk model), NodeConsequence(referring to consequence values of nodes in the risk model), NodeRisk(referring to risk values of nodes in the risk model).
- a*type* referring to the name of a type definition list.

The following shows an example of a list of type assignments:

```
<typeassignment name="EdgeLikelihood"  type="EdgeLikelihoodType"/>
<typeassignment name="NodeLikelihood"  type="NodeLikelihoodType"/>
<typeassignment name="NodeConsequence" type="NodeConsequenceType"/>
<typeassignment name="NodeRisk"        type="NodeRiskType"/>
```

In this example, four type assignments are defined. For instance, here the likelihood type of edges is set to the type with name EdgeLikelihoodType and the likelihood type of nodes is set of the type name NodeLikelihoodType.

The assignments may also be used to define mappings values to intervals. This is useful when risk evaluation criteria are defined in risk matrices. As an example, consider the risk matrix shown in the following table:

	Seldom	**Unlikely**	**Possible**	**Probable**	**Certain**
Insignificant	🟩 1	🟩 2	🟩 3	🟩 4	🟨 5
Small	🟩 2	🟩 3	🟩 4	🟨 5	🟥 6
Medium	🟩 3	🟩 4	🟨 5	🟥 6	🟥 7
High	🟩 4	🟨 5	🟥 6	🟥 7	🟥 8
Critical	🟨 5	🟥 6	🟥 7	🟥 8	🟥 9

Here columns represent likelihood values, and the rows represent consequence values. Combinations of likelihood and consequence values are mapped to one of the three risk levels: green, yellow, or red. In order to determine the risk level given a likelihood and a consequence value, we need to map these onto the appropriate rows and columns. This is the purpose of the mapping assignments.

A mappings assignment consists of

- a name defining the kind map assignment, where the following values are possible: LikelihoodMap (specifying that the map applies to likelihood values), ConsequenceMap (specifying that the map applies to consequence values), RiskMap (specifying that the map applies to risk values).
- a*type* referring to the name of a type definition list.

As an example, consider the following definitions:

```
<typeassignment name="LikelihoodMap"   type="NodeLikelihoodType"/>
<typeassignment name="ConsequenceMap"  type="NodeConsequenceMap"/>
<typeassignment name="RiskMap"         type="NodeRiskType"/>
```

For instance, here the map for likelihood values is defined by the type NodeLikelihoodType. The intervals of the likelihood values is defined by the literals in the NodeLikelihoodType. If we assume that this type is the one defined in the example above, then we can see that likelihood values in the range between 0 and 0.1, will be mapped to 1 (the first literal in the list), likelihood values in the range between 0.1 and 1.0 will be mapped to 2 etc.

The XSD schema defining the format of type definitions can be found in the folder *misc* located in the Corasanalyzer tool distribution folder.

3.5 Security Dashboard

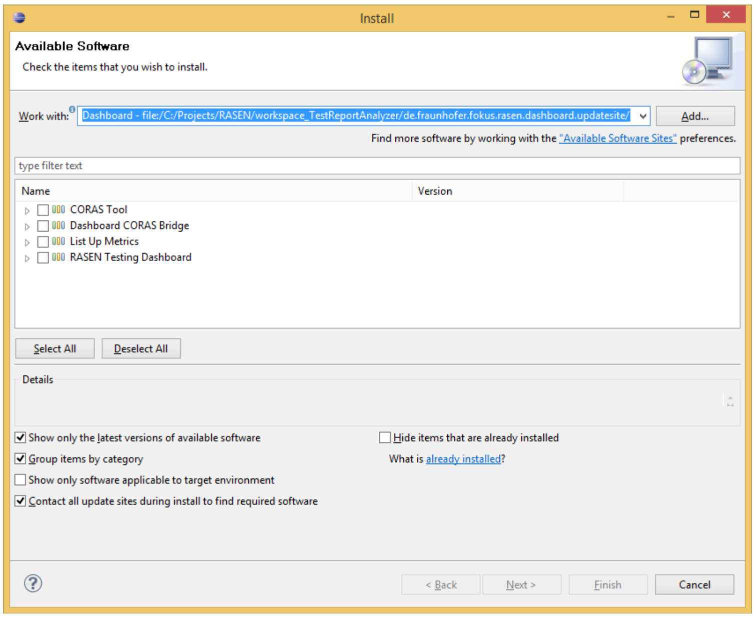
This section introduces the RASEN Testing Dashboard developed and provided by the Fraunhofer FOKUS institute. The Dashboard manages the referenced models for the RASEN security assessment, generates test metrics and measurements for the security elements of interest, visualizes them in different views and provides an exporter for aggregated results.

3.5.1 Presentation

The Testing Dashboard is designed as a Java plugin for the Eclipse environment. It consists of a risk test model analyzer, a metric generator and two different views embedded in the Eclipse workbench: a dashboard metric table view and a metric chart view. The risk test model analyzer processes registered models for metric generation and visualization. The GUI part of the Testing Dashboard provides different user interaction options, including selecting elements and metrics for the analysis, and setting different parameters for their visualization. Selected elements and their metrics can be exported as aggregated test reports.

3.5.2 Installation Guidelines

The Testing Dashboard has to be installed from the update site. Therefore, select the menu Help -> Install new Software -> Add… -> Archive …. and select the file or enter the link of the update site.



Install

Available Software

Check the items that you wish to install.

Work with: Dashboard - file:/C:/Projects/RASEN/workspace_TestReportAnalyzer/de.fraunhofer.fokus.rasen.dashboard.updatesite/

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
□ CORAS Tool	
□ Dashboard CORAS Bridge	
□ List Up Metrics	
□ RASEN Testing Dashboard	

Details

☑ Show only the latest versions of available software

☑ Group items by category

□ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

□ Hide items that are already installed

What is already installed?

Figure 27 – Update Categories

After this, it can be chosen between four categories. The required component *RASEN Testing Dashboard* installs all core components such as the security model analyzer, the metric generator, the metric table view and the metric chart view. Also required component *List up metrics* provides the dashboard with list up metrics for analysis processing. Metrics such as coverage metrics and efficiency metrics are planned for further development. The Testing Dashboard is designed to interact with the CORAS editor: CORAS risk models can be directly added to the dashboard and risk elements can be selected in the CORAS editor to use them for the metric chart view. The *Dashboard CORAS Bridge* realizes this interaction. The *CORAS tool* can also be installed from here, but this is not recommended. The installed version causes failures because it is not an approved installation. To install the CORAS editor see the tool description for the CORAS tool.

4 Conclusion

In this report we have given an overview of the RASEN WP3 and WP4 tools. The tools presented in this deliverable are CORAS (SINTEF), RACOMAT (Fraunhofer FOKUS), Security plugins (UFC) for Smartesting CertifyIt, ARIS Business Architect (Software AG) and RASEN security dashboard (Fraunhofer FOKUS). In addition to give an introduction to the features and purposes of these tools, we have described the further development goals and the current status of these tools.