

Compositional Risk
Assessment and Security
Testing of Networked Systems

Deliverable D3.2.3

Techniques for Compositional Test-Based Security Risk Assessment v.3

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
Deliverable number:	D3.2.3
Nature of deliverable:	Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2015-09-30
Actual delivery date:	2015-09-30
Responsible partner:	Software AG

Contributors

Editor(s)	Bjørnar Solhaug (SINTEF)
Contributor(s)	Fredrik Seehusen (SINTEF), Bjørnar Solhaug (SINTEF), Johannes Viehmann (Fraunhofer), Frank Werner (Software AG)
Quality assuro(s)	Samson Esayas (UiO), Erlend Eilertsen (Evry)

Version history

Version	Date	Description
0.1	15-02-13	Table of contents
0.2	15-09-04	Draft version
0.3	15-09-07	Revision of all sections
0.4	15-09-11	Finalized for internal review
0.6	15-09-22	Revision after internal review
1.0	15-09-30	Final revision and quality check preformed

Abstract

This deliverable reports on the main results of RASEN WP3 from the third and final year of the project. The tasks that have been addressed are: (T3.1) the development of techniques for compositional security risk assessment, (T3.2) the development of techniques for test-based security risk assessment, and (T3.3) the development of techniques for continuous risk assessment by means of test-based indicators.

The RASEN approach to component-based and test-based security risk assessment has been further developed, including the tool-support. In particular, this deliverable documents the following WP3 contributions. Tool-supported techniques for component-based security risk assessment supported by testing; security test result aggregation using test metrics and risk metrics; a tool supported approach to component-based security risk assessment for composition of risk assessment results.

Keywords

Security, security risk assessment, component-based security risk assessment, test-based security risk assessment, test result aggregation

Executive Summary

The overall objective of RASEN WP3 is to develop tools and techniques to facilitate compositional security risk assessment and component-based security risk assessment supported by security testing. This includes developing tools and techniques i) for compositional and for component-based security risk assessment and security testing, ii) for identifying, estimating and verifying security risks based on security test results, and iii) for reuse of risk assessment and security test results, as well as dynamic updates of the security risk assessment based on test results and using metrics and risk metrics.

This deliverable reports on the WP3 results after the third year of the project. The results cover all of the WP3 research tasks, namely (T3.1) the development of techniques for compositional security risk assessment, (T3.2) the development of techniques for test-based risk identification and estimation in order to complement the risk picture based on test results, and (T3.3) the development of techniques for continuous risk assessment of large scale systems by the use of test-based indicators. In particular, the deliverable makes the following contributions.

- Tool-supported techniques for component-based security risk assessment supported by testing. Two alternative approaches are offered, each of which makes use of the same software product component model where components are decomposed into a hierarchy.
- Security test result aggregation using test metrics and risk metrics. A process and documentation format is introduced for specifying metrics and aggregation functions, so as to enable the aggregation of test results to risk information that can be fed to the security risk model.
- Tool-supported approach to component-based security risk assessment. The overall security risks for a software system are derived by aggregating the security risks for the individual components that the system is composed of. The approach moreover allows the reuse of component risk assessment results in the different contexts and systems where the component in question may be used.

The WP3 results contribute to support and facilitate the overall RASEN methodology that is presented in the context of WP5. The WP3 tools are moreover integrated into the RASEN tool-box and have therefore the potential to be used in combination with other RASEN tools and techniques.

Table of contents

TABLE OF CONTENTS	5
1 INTRODUCTION	6
2 COMPONENT-BASED RISK ASSESSMENT COMBINED WITH TESTING	7
2.1 RISK AGGREGATION IN ARIS.....	8
2.2 RISK AGGREGATION VIA RACOMAT.....	10
3 TEST RESULT AGGREGATION USING TEST METRICS AND RISK METRICS	12
3.1 THE PROCESS FOR DEFINING MEASUREMENT AGGREGATION	12
3.1.1 Step I: Identify Source and Target Data Model Elements.....	12
3.1.2 Step II: Specify Metrics and Dependencies.....	12
3.1.3 Step III: Specify Aggregation Functions.....	13
3.2 A CATALOGUE OF COMMON MEASUREMENTS, DEPENDENCIES AND AGGREGATION FUNCTIONS	14
3.2.1 Common Source and Target Data Model Elements	14
3.2.2 Common scales and dependencies	14
3.2.3 Common Aggregation Functions	17
3.3 THE APPLICATION OF THE PROCESS TO AN EXAMPLE.....	18
3.3.1 Step I: Identify Source and Target Data Model Elements.....	18
3.3.2 Step II and Step III: Specify Metrics and Dependencies and Aggregation Functions.....	19
3.3.3 Aggregation of test results.....	21
4 REUSABLE RISK ASSESSMENT ARTIFACTS AND RISK AGGREGATION WITH THE HELP OF TAGS AND SCOPES	23
4.1 TAGGING RISK ASSESSMENT ARTIFACTS	23
4.2 THE SCOPE OF FAULTS AND UNWANTED INCIDENTS	24
4.3 RISK AGGREGATION USING TAGS AND SCOPES.....	26
5 CONCLUSION	28
REFERENCES	29
APPENDIX A: IMPORT/EXPORT FORMAT FOR ARIS IN XSD	30

1 Introduction

A main objective of RASEN WP3 is to develop techniques and tools that facilitate security risk assessment of large-scale and complex software systems. To fulfill this objective WP3 has conducted R&D tasks in three directions. First, we have developed techniques and modeling support for compositional security risk modeling and assessment, as well as component-based techniques. Such techniques should allow large system to be decomposed into smaller sub-systems or components that can be analyzed separately. For this we need methods for deriving the combined results of the individual analyses. Second, we have developed techniques for test-based risk identification and estimation, so as to complement the risk picture based on the test results. These techniques involve the aggregation of test results by using test metrics and risk metrics. Third, we are investigating techniques for continuous security risk assessment by leveraging the techniques for compositional security risk assessment, and by means of test metrics.

The current WP3 status and the third and final year results of these R&D activities are presented in this deliverable. The activities correspond to research tasks T3.1 (compositional security risk assessment), T3.2 (test-based risk identification and estimation) and T3.3 (continuous risk assessment) of RASEN WP3. More specifically, the technical contents of this deliverable are as follows.

Section 2 presents techniques for tool-supported component-based security risk assessment supported by testing. We describe two different approaches to this that complement each other. Each of them makes use of the same software product model, where components are decomposed into hierarchies. The first approach allows security risk aggregation in ARIS in which security risks are estimated at four abstraction layers, namely application, system, network and enterprise. In the second approach the software component risks are obtained by security testing using the RACOMAT tool. The tool imports the software product component hierarchy from ARIS, and the security risks are aggregated accordingly.

Section 3 presents the RASEN approach to security test result aggregation using test metrics and risk metrics. We introduce a process for defining measurement aggregation using the RASEN data models for security testing and security risk assessment. The relevant model elements, metrics and aggregation functions are specified and documented using a well-defined table format. We introduce and explain all these formats, before exemplifying their instantiation.

Section 4 presents the RACOMAT tool-supported approach to component-based security risk assessment. The approach facilitates security risk assessment of large-scale systems by allowing the decomposition of systems into smaller components of sizes that are manageable. The idea is further that risk assessment results for individual components can be reused in different context and for different software systems in which the component is used. The challenge that we address is how to aggregate the individual results and derive the correct security risk picture for the combined system.

Finally we conclude in Section 5 before presenting the import/export format for ARIS in the appendix.

The presented methods and techniques support various parts of the overall RASEN methodology as presented in the context of WP5. The WP3 tools are moreover integrated into the RASEN tool-box, which means that they can be used in combination with other RASEN tools. The WP3 tool portfolio is provided in prototype deliverable D3.3.3.

2 Component-Based Risk Assessment Combined With Testing

Component-based risk assessment allows a software product to be assessed by addressing its individual components separately. In this section we describe two different ways in which we can conduct component-based security risk assessment in RASEN. The two alternatives eventually provide a combination of risk aggregation along the product hierarchy, as well as security risk estimation by evaluating the risk tree of the product. What the approaches have in common is that they share the same ARIS model as input. This facilitates their use and increases the potential for exploitation; apart from the RASEN methodology, no additional modelling conventions are required.

The interface between ARIS Business Architect from Software AG and the RACOMAT tool from Fraunhofer FOKUS is realized using a standardized RASEN format which can be used in both directions. The current version uses a format based on JSON¹ (Java Script Object Notation) which is considered to be an open standard format using human-readable text to transmit data objects consisting of attribute-value pairs. The exchange format contains a product hierarchy in arbitrary depth, it fully implements the vignette, it allows for components along with their common weakness score (CWS), and it has capabilities of expressing risk ratings. An XSD variant is detailed in Appendix A.

As the approach to the modeling of large-scale network systems is already described in detail in past deliverables (D3.2.1 and D3.2.2) we will refrain from repeating ourselves, rather focusing on the new aspects of the import of the Common Weakness Enumerations (CWEs) [4], as well as the computation of risks using a) the risk graph from RACOMAT, and b) a set of aggregation functions.

Figure 1 gives an overview the two approaches along with the required steps.

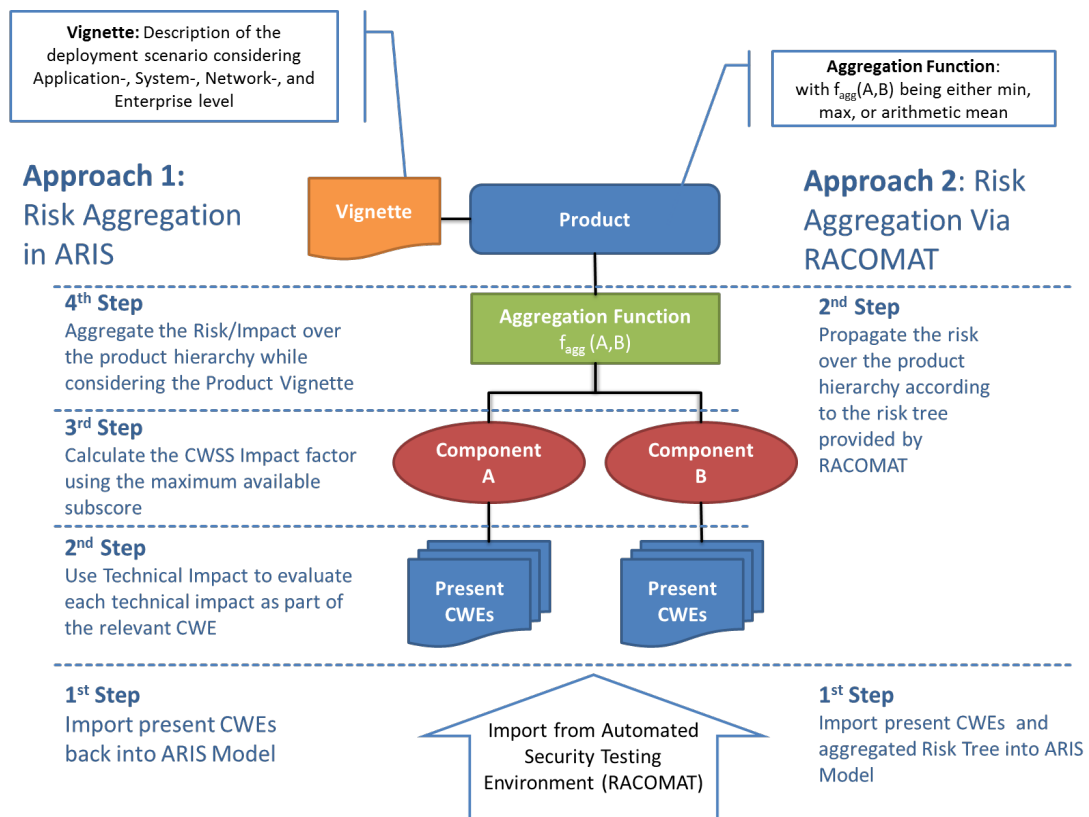


Figure 1 – Comparing component-based risk aggregation from CWEs and RACOMAT

¹ <https://tools.ietf.org/html/rfc7159>

The essential elements of the two approaches are the CWEs. For each component, the weaknesses (in terms of CWE-IDs) are assigned using one or more of the following methods following an automated generation process:

1. Collecting all CWE-IDs of a CWE category: For the specified category, weaknesses are collected recursively following the tree structure.
2. Searching for CWE-IDs using a search term: CWE weaknesses have a headline and a short summary description. All CWE weaknesses that contain the search term are added to the component type.
3. Collecting all CWE-IDs that share a specified value: for example operating system, framework, programming language, etc.
4. Adding CWE-IDs manually: Some CWE-IDs are added manually to component types. This is necessary because not all CWE-IDs can be assigned automatically using one of the methods described above. The most important reasons for that is incomplete and/or inconsistent data in the CWE database.

2.1 Risk Aggregation in ARIS

The main characteristic for doing the security risk aggregation in ARIS is the use of a Product Vignette. During the process of a software product security risk assessment, risk model is assessed in which the product hierarchy is represented, as well as a vignette being unique for that product. The vignette describes the deployment scenario in terms of “*Where can data be read on network level?*”, “*How can unauthorized code be executed on Enterprise level?*”, “*How can privileges be gained on application level?*” etc.

The vignette is a placeholder which contains all kinds of possible deployment scenarios, following a stringent capturing of exploitation scenarios of four distinct layers:

- Application layer: This is the lowest abstraction layer where exploitation scenarios are defined based on a single application.
- System layer: This layer is defined just above the application layer, considering a software system which potentially hosts many different applications, e.g., a server OS
- Network layer: This layer reflects exploitations on peer to peer basis e.g., through networks, transmission protocols, etc.
- Enterprise layer: This is the highest level of abstraction where product security risks are exposed to the whole company.

On each of these levels, the risk impact is quantified according to the following list. The estimates are used to calculate the CWSS (Common Weakness Scoring System) [5]. The CWSS provides a mechanism for scoring weaknesses in a consistent, flexible, open manner while accommodating context for various business domains. As it is part of the CWE project, it can be interlinked with information already available in the models to allow quantitative measures of available weaknesses present within a software component.

The list of possible impacts contains the following eight elements:

- Modify data
- Read data
- DoS: unreliable execution
- DoS: resource consumption
- Execute unauthorized code or commands
- Gain privileges / assume identity
- Bypass protection mechanisms
- Hide activities

When assessing the component-based risks obtained from the security testing, the following four-step process is conducted:

1. The model is imported from RACOMAT using the common exchange format. At this point the ARIS model contains a list of all potential CWEs which, in the subsequent process, is replaced by a list of actually present CWEs.
2. Using the technical impact from the previously imported CWEs, the impact is evaluated by combining the CEWs impact and a product impact which is derived from the product wide vignette. Note that the vignette represents/characterizes the deployment scenario of the product and is static for computation over the whole product hierarchy.
3. The CWSS impact factor is calculated by using the maximum available sub-scores over all child components and CWEs.
4. After the CWSS Risk Rating is available, the risk is further aggregated using aggregation functions along the product hierarchy. A set of possible functions is depicted in Figure 2.

Following this process, the overall software product security risk is eventually obtained and displayed at the product level.

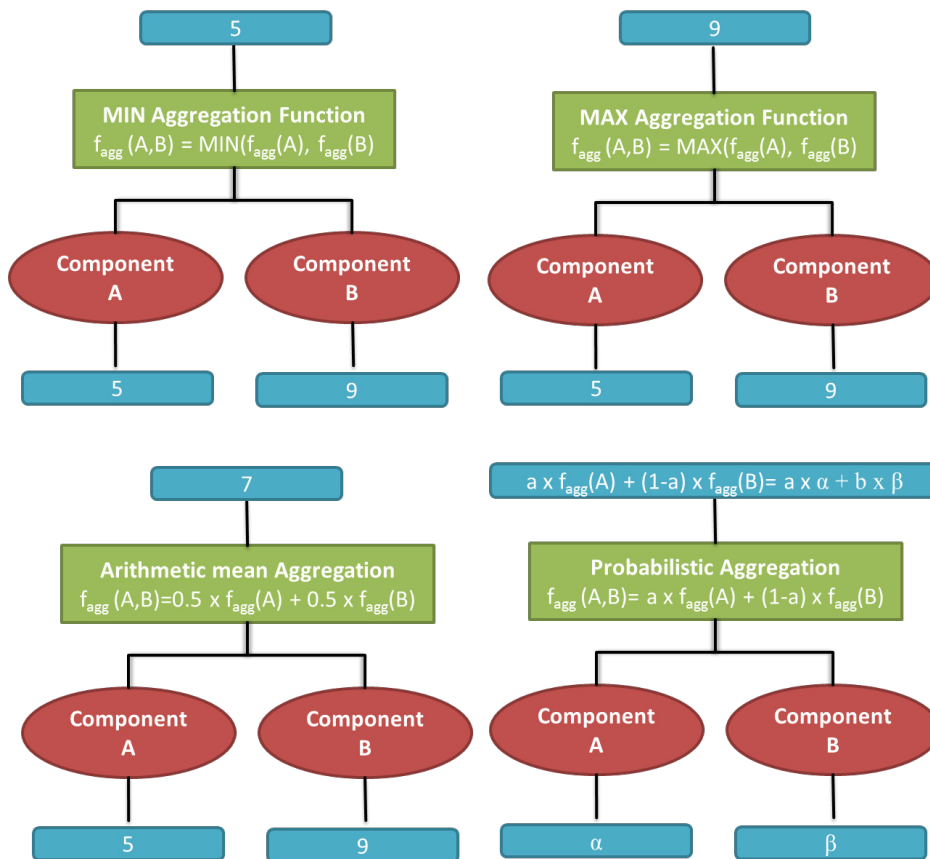


Figure 2 – Examples of different risk aggregation models

As the above mentioned aggregation functions revealed its limits, the 2nd approach introduces the component based risk assessment where risks are obtained directly from the security testing.

The approach of security testing in large software environments needs maximum degree of automation, the processes of which should be designed to require only minimal user interaction. To comply with this, the above depicted aggregation models can be automatically extracted from the risk models in ARIS and used for security testing. This is in particular convenient as no further human

interference is involved. On the other hand, when considering the level of detail and the meaningfulness of the obtained aggregated results, it becomes visible that results can only be treated as an estimate of the software system security risks. Here the aggregation function for the minimal and maximum risk ($f_{agg(min)}$ and $f_{agg(max)}$) deliver upper and lower values, and the risk lies somewhere in between. One major drawback of the above stated aggregation functions is that only risks along the hierarchy are considered which for example completely neglects inter-dependencies of software components residing in different branches.

To combat this downside, a more sophisticated approach is taken by considering the risk along the hierarchy in a risk-based graph. As this solution delivers realistic results, it has been considered within the evaluation phase and is detailed in the following and in Section 4.

2.2 Risk Aggregation via RACOMAT

For the approach to security risk aggregation using RACOMAT, the CWEs are imported as in the alternative approach described above, but the risks are not aggregated using the same functions. Instead RACOMAT computes the security risks based on the complete risk product graph obtained during the automated testing phase. As risks are already precomputed, they can be immediately used in the model [7].

This approach follows a two-step process:

1. The present CWEs are imported in analogy to the alternative approach as described in the previous sub-section. In addition, the product risk graph is imported into the ARIS tool. The risk graph contains pre-computed, aggregated risks for all components, and therefore also eventually the also the risk rating for the top level product.
2. In accordance to the risk graph, the risks are propagated over the complete product hierarchy. The graphical notation of the approach moreover reflects the actual risks at each of the components.

An illustration of the transformation process on high level is shown in the following figure.

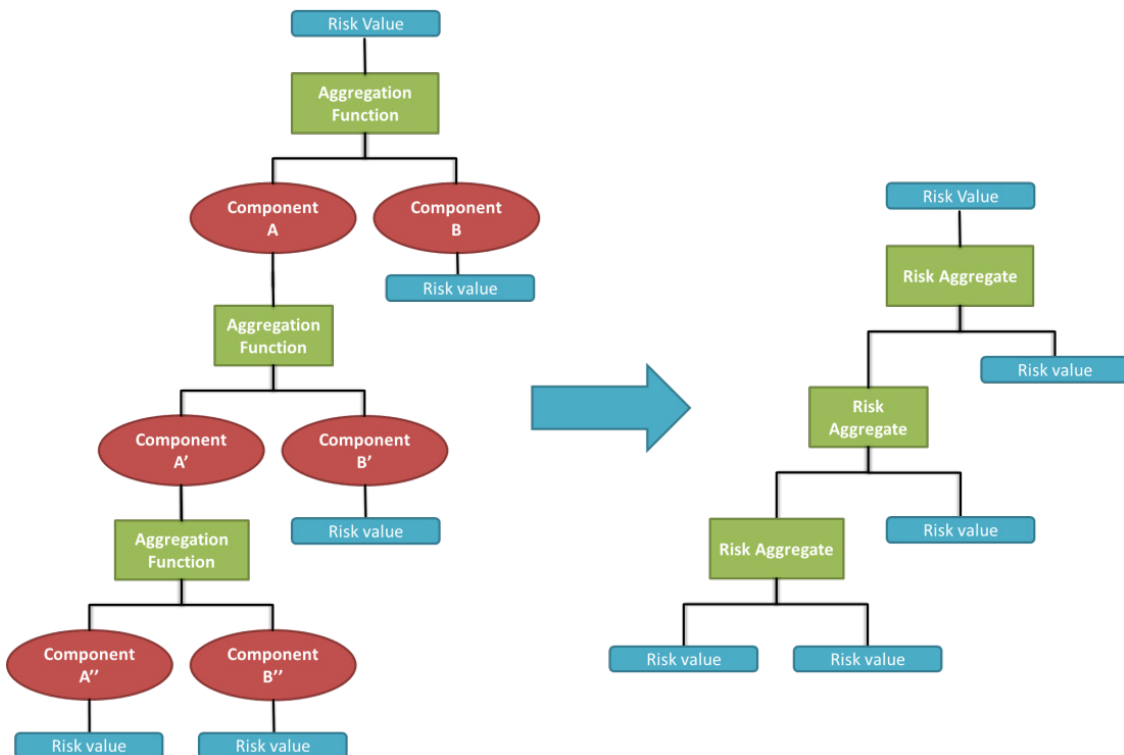


Figure 3 – Transformation of the Product tree into the Risk Graph using RACOMAT

As usability and applicability has been a major concern, the risk aggregation via RACOMAT seems to be more sophisticated as its aggregation reflects applied security test pattern and security testing metrics when computing risks on component level. The RACOMAT tool as well as its underlying way of aggregating risk is described in more detail in Section 4.

3 Test Result Aggregation Using Test Metrics and Risk Metrics

In order to understand what security test results mean in terms of impact on risks, we need techniques for bridging the low-level technical details obtained through testing with the higher level information of the risk assessment. For this purpose, RASEN develops metrics which are intended to support the propagation/aggregation of the test results to the risk assessment level. In this section we first, in Section 3.1, describe a process for defining measurement aggregation functions in general. Then, in Section 3.2, we define a catalogue of common measurements, and finally in Section 3.3, we demonstrate the process on a small example. Tool support related to the test result aggregation techniques is documented in RASEN deliverable D3.3.3-D4.3.3.

3.1 The Process for Defining Measurement Aggregation

The process for defining measurement aggregations is conducted over three steps. The results of each step are document in dedicated table templates as described in the following.

3.1.1 Step I: Identify Source and Target Data Model Elements

The process assumes that the source data (test result data which will be input to the transformation) and the target data (risk assessment data which is the output of the transformation) are defined in a data model. The first step of the process is then to identify the data model elements which will be the source and target of the transformation, and to document this in the template of Table 1.

Table 1 – Template for defining model elements

Data model element ID	Kind	Description
<i>Element identifier</i>	<p><i>Type of model element.</i></p> <p><i>There are three possible types: source, target, and intermediate. Source type refers to elements that are input to the aggregation, target type refers to output elements, and intermediate type refers to data elements that are used as intermediate calculation steps by the aggregation.</i></p>	<i>Description of model element</i>

3.1.2 Step II: Specify Metrics and Dependencies

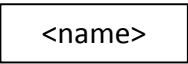
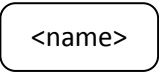
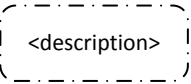

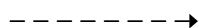
The second step of the method is to define metrics which will be used as intermediate calculation data points for aggregating the source data to the target data. The first activity is to document this in a *dependency* graph following these steps:

1. List all the target data elements at the top of the graph.
2. List all the source data elements at the bottom.
3. Connect the target data points to the source data points using metrics.

That the third step should be performed in a top-down fashion; we start with the target data elements and break these down into metrics until metrics whose values can be calculated on the basis of the source data elements are identified. We take note of metrics whose value cannot be calculated on the basis of source data elements, but that nevertheless are needed in order to calculate the target data elements. Such metrics will have to be manually estimated during measurement aggregation, or defined as a source of uncertainty.

The resulting dependency graph should be document using the notation of Table 2.

Table 2 – Dependency graph notation

Graphical notation	Description
	Model element (typically corresponds to a class in a meta model)
	Metric
	Comment
	Association between model elements
	Dependency arrow, used to denote that a model element or measurement may depend on another model element or measurement

The metrics and their types should additionally be documented using the templates of Table 3(metric type) and Table 4 (metric).

Table 3 – Template for defining types

Literal name	Description	Definition
<i>Name of type literal</i>	<i>Description of type literal</i>	<i>Precise definition of literal</i>

Table 4 – Template for defining metrics

Metric ID	Metric name	Description	Scale/Type
<i>Metric identifier</i>	<i>Metric name</i>	<i>Description of metric</i>	<i>Type of measurement for metric</i>

3.1.3 Step III: Specify Aggregation Functions

After having defined the dependency graph, the next step is to define aggregation functions that describe how the target data elements can be calculated from the source data elements using metrics. The functions should be documented using the template of Table 5.

<i>Name of aggregation function</i>	
Source measurements	<i>Definition of measurements that are input of the aggregation function.</i>
Target measurements	<i>Definition of the measurements that are the output of the function</i>
Description	<i>Textual description of the function</i>
Definition	<i>Precise definition of the function</i>

Table 5 – Template for defining aggregation function

3.2 A Catalogue of Common Measurements, Dependencies and Aggregation Functions

In this section we define source and target data model elements, measurements, measurement dependencies, and aggregation functions which have been identified in the RASEN case studies.

3.2.1 Common Source and Target Data Model Elements

In the RASEN case studies, we were always interested in updating the conditional likelihood values of the relations in the risk models. In the RASEN data model, this information is expressed by the `data::riskassessment::RiskRelation` and the `data::foundation::Parameter` data elements. Furthermore, the test report with associated test cases and test items were used to express the test results. The relevant data elements of the RASEN data model are shown in Table 6.

Table 6 – Common data source and target data elements of the RASEN data model

Data model element ID	Kind	Description
<code>data::riskassessment::RiskRelation</code>	Target	The risk relation data element corresponds to an arrow in a CORAS diagram. It contains the source and target nodes of the arrow, as well as arrow annotations, i.e. likelihood and vulnerabilities. The data item that we want to update is the likelihood on the relation. In the data model, this likelihood is represented by as a parameter to the relation.
<code>data::foundation::Parameter</code>	Target	We are interested in parameters whose name attribute is "Vulnerability", to represent names of vulnerabilities, and "Likelihood" to represent conditional likelihoods of risk relations.
<code>data::testing::TestReport</code>	Source	Contains all relevant information about the test results.

3.2.2 Common scales and dependencies

For those measurements which may be automatically calculated are usually defined as probability intervals. For measurements which must be assessed by expert judgement, and in addition are not suited in automated calculation, it may be useful to use a qualitative scale. Two common scales that we have used often are described in Table 7.

Table 7 – Common scales

Literal name	Description	Definition
Probability interval	A pair [p,q] of two probabilities p and q such that p is less than or equal to q	{0,...,1}, {0,...,1}
Qualitative scale	Ordinal scale of three values.	{low, middle, high}

In Figure 4, we have illustrated a dependency graph showing the dependencies between common metrics. The metrics are intended to be used as intermediate steps for updating the conditional likelihood of a risk model (expressed by the Parameter class) based on the test results (contained in the TestReport). The highest level metric is called "Attack success likelihood". This is intended to be a

probability estimate which directly corresponds to the likelihood of the risk model relation. The "Attack success likelihood" is further decomposed/dependent on metrics related to the vulnerabilities that may be exploited in an attack. Each vulnerability associated with an attack has a "Vulnerability severity" metrics, estimating the likelihood that the vulnerability will be exploited in an attack if the attack is initiated. This metric is further broken down into "Vulnerability existence likelihood", estimating the likelihood that a vulnerability exists in the system under test, and "vulnerability exploitability", estimating how easy it is to exploit the vulnerability if it exists. Security test results will most often contain information about the presence/absence of vulnerabilities. Thus the "Vulnerability existence likelihood" metric is further broken down into metrics that are related to the testing activity. The metrics "#passed test cases" and "#failed test cases" can be derived from the TestReport containing the test results, whereas the other metrics must typically be manually estimated.

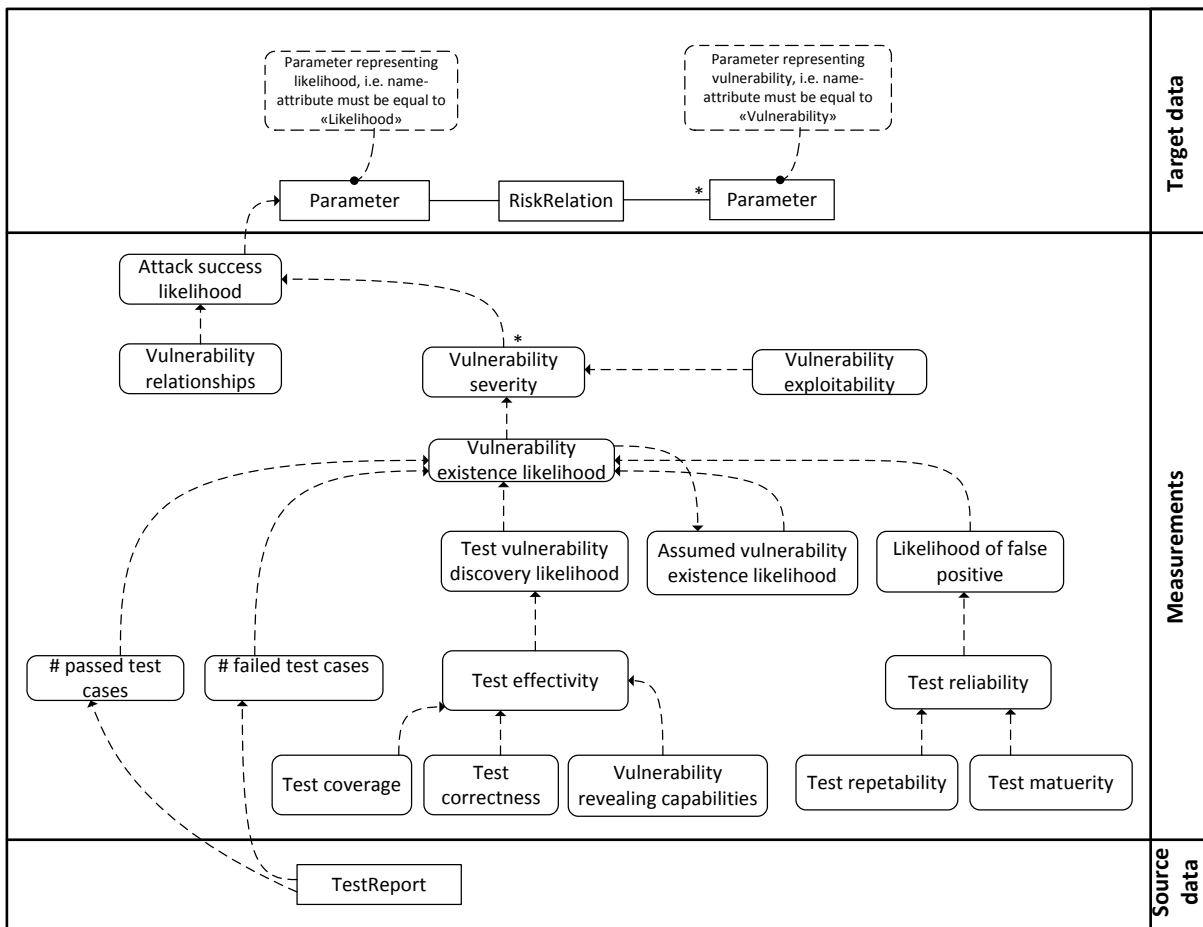


Figure 4 – Dependency graph

In Table 8, we have defined the metrics of Figure 4 more precisely.

Table 8 – Common metrics

Metric ID	Metric name	Description	Scale/Type
R1	Attack success likelihood	Specifies the likelihood that a security attack will succeed in exploiting a set of vulnerabilities	Probability interval
R2	Vulnerability relationships	Specifies the dependencies of a set of vulnerabilities e.g. whether it is sufficient to exploit one of them to achieve a successful attack, or whether a combination of them must be exploited.	Text

R3	Vulnerability severity	Specifies the likelihood that a vulnerability exists and is successfully exploited if this is attempted	Probability interval
R4	Vulnerability exploitability	Specifies the likelihood that that an attacker will be able to exploit the vulnerability if this is attempted.	Probability interval
R5	Vulnerability existence likelihood	Specifies the likelihood that a vulnerability exists in the SUT	Probability interval
RT1	Test vulnerability discovery likelihood	Probability that a test fails if the SUT contains the vulnerability which we are testing for	Probability interval
RT2	Likelihood of false positive	Probability that a test fails if the SUT does not contain the vulnerability which we are testing for	Probability interval
RT3	Assumed vulnerability existence likelihood	Assumed probability that system has vulnerability (before testing)	Probability interval
T1	# passed test cases	This indicates that no vulnerability has been found	Integer
T2	# failed test cases	This indicates that a vulnerability has been found	Integer
T3	Test Effectivity	The capability of the specified test procedure to fulfill a given test purpose thus to reveal vulnerabilities when they are exist.	Qualitative scale
T4	Test Reliability	The capability of the specified test procedure to maintain a specific performance under different conditions.	Qualitative scale
T5	Existence of failed test cases	Specifies if a test run has failed test cases	Boolean
T3.1	Test Coverage	Coverage constitutes a measure for test completeness and measures the degree to which the test specification covers the pre-defined coverage items (coverage items may be requirements, system elements, attack vector classifications, or test purpose descriptions)	Qualitative scale (or percentage of covered coverage items or #test cases per coverage item)
T3.2	Test Correctness	Test correctness denotes the correctness of the test specification with respect to the system specification or the test purposes and when it always returns correct test verdicts.	Qualitative scale
T3.3	Vulnerability Revealing Capabilities	Capability of a test procedure to actually reveal vulnerabilities	Qualitative scale
T4.1	Test Repeatability	The capability of the specified test procedure to yield the same results in different test runs.	Qualitative scale

T4.2	Test Maturity	Frequency of failure of the of the test procedure.	Qualitative scale
------	---------------	--	-------------------

3.2.3 Common Aggregation Functions

In this section, we define three aggregation functions which may be calculated automatically. The first shown in Table 9, propagates metrics associated with a set of vulnerabilities to a metric estimating the likelihood of a successful attack. The second function, shown in Table 10, is used to calculate how the likelihood of a vulnerability being exploited, given the likelihood that the vulnerability exists, and the likelihood that it will be exploited if it exists. Finally, the third aggregation function, shown in Table 11, can be used to calculate the likelihood that a vulnerability exists based on test results. This function takes as input the number of passed and failed test cases (where failed indicates that a test has found a vulnerability), the likelihood that the system under test has a vulnerability if the test fails, the likelihood that the system does not have a vulnerability if the test fails, and assumed likelihood that the system under test has a vulnerability. Number of passed and failed test cases can be derived from a test report, whereas the other measurements may have to be estimated manually based on expert judgement. Based on the given input, the aggregation function shown Table 11, will use Bayesian inference to update the assumed likelihood of vulnerability existence based on the evidence provided, i.e. number of passed and failed test cases.

Table 9 - Aggregation function AF1: Attack likelihood aggregation

Attack likelihood aggregation	
Source measurements	<ul style="list-style-type: none"> vr: Vulnerability relationships (R2) $\{(v1, s1), \dots, (vn, sn)\}$: (data::riskassessment::Parameter, Vulnerability severity (R3))*
Target measurements	<ul style="list-style-type: none"> asl : Attack success likelihood (R1)
Description	For simplicity, we assume that it is sufficient to exploit one vulnerability to achieve a successful attack. We therefore let the attack success likelihood be equal to the sum of the vulnerability severity values.
Definition	<ul style="list-style-type: none"> $asl = s1 + \dots + sn$

Table 10 – Aggregation function AF2: Vulnerability severity aggregation

Vulnerability severity aggregation	
Source measurements	<ul style="list-style-type: none"> (v,e): (data::riskassessment::Parameter, Vulnerability exploitability (R4)) (v,l): (data::riskassessment::Parameter, Vulnerability existence likelihood (R5))
Target measurements	<ul style="list-style-type: none"> (v, s): (data::riskassessment::Parameter, Vulnerability severity (R3))
Description	The vulnerability severity is defined as the multiplication of the exploitability e and the likelihood that the vulnerability exists l
Definition	<ul style="list-style-type: none"> $s = l * e$

Table 11 – Aggregation function AF3: Vulnerability existence aggregation

Vulnerability existence aggregation	
Source measurements	<ul style="list-style-type: none"> • <i>V</i>: data::riskassessment::Paramenter • <i>P</i>: #passed test cases, • <i>F</i>: #failed test cases, • <i>DL</i>: Test vulnerability discovery likelihood • <i>FP</i>: Likelihood of false positive • <i>AL</i>: Assumed vulnerability existence likelihood
Target measurements	<ul style="list-style-type: none"> • <i>oV</i>: data::riskassessment::Paramenter • <i>oVE</i>: Vulnerability existence likelihood
Description	This function will use Bayesian inference to update the initial assumed estimate of the vulnerability existence (as specified by <i>AL</i>) by taking into account the number of passed test cases (given by <i>P</i>) and failed test cases given by (<i>F</i>). The aggregation function will be defined in terms of the binominal function BINOMDIST.
Definition	<ul style="list-style-type: none"> • $oV = v$ (the name of the output vulnerability is equal to the name of the input vulnerability) • $Pa = AL * BINOMDIST(F, P+F, DL)$ (Probability that a system has vulnerabilities given sample) • $Pb = (1 - AL) * BINOMDIST(F, P+F, FP)$ Probability that a system has no vulnerabilities given sample) • $oVE = Pa / (Pa + Pb)$ (Vulnerability existence likelihood, i.e. normalized probability that a system has vulnerability given sample)

3.3 The Application of the Process to an Example

In this section we illustrate the process described in Sect. 3.1 with an example.

3.3.1 Step I: Identify Source and Target Data Model Elements

The first step of the process is to identify data elements that should be updated based on test results. In this example, we assume that the risk model to be updated is the one shown in Figure 5, and that we are furthermore only interested in updating the three conditional likelihoods on the edges highlighted by the red ovals. In the RASEN risk model, this conditional likelihood is contained in the element data::foundation::Parameter which is attached to the data::riskassessment::RiskRelation data element corresponding to relations in the risk model.

Assume further that we have a test report which only contains information about how many tests have passed and failed regarding tests related to existence of vulnerabilities w.r.t. the three relations highlighted in Figure 5. In the RASEN data model, the test report is represented by the data::testing::TestReport.

The source and data elements are summarize in Table 12.

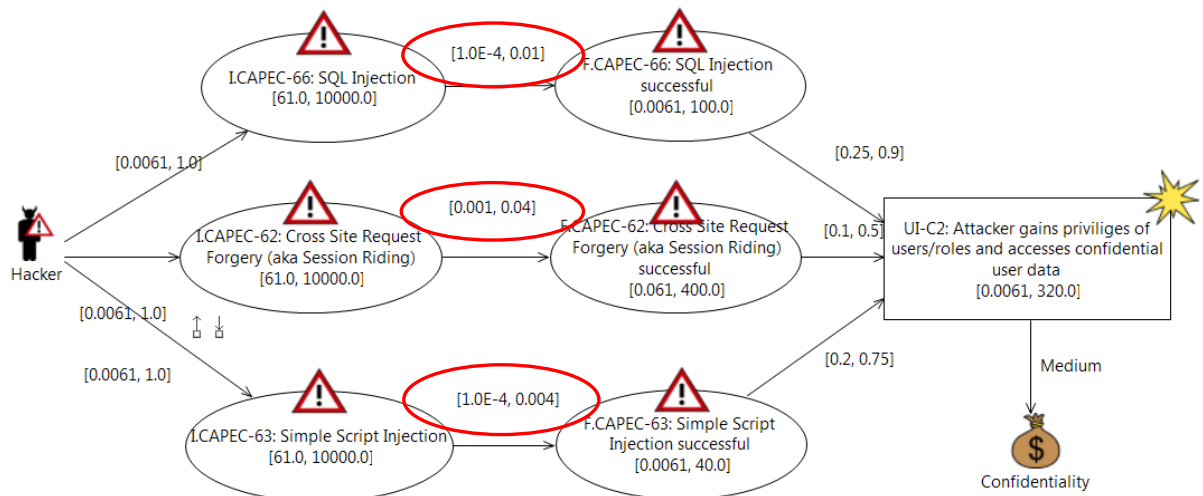


Figure 5 – Risk model to be updated

Table 12 – Source and target data elements

Data model element ID	Kind	Description
data::riskassessment::RiskRelation	Target	The risk relation data element corresponds to an arrow in a CORAS diagram. The data item that we want to update is the likelihood on the relation. In the data model, this likelihood is represented by as a parameter to the relation.
data::foundation::Parameter	Target	We are interested in parameters whose name attribute is "Likelihood" to represent conditional likelihoods of risk relations.
data::testing::TestReport	Source	We are interested in the number of passes and failed tests for each relation that is tested for existence of vulnerabilities.

3.3.2 Step II and Step III: Specify Metrics and Dependencies and Aggregation Functions

In step II, we specify measurements that are partially derived from the test results and which can be used to update the conditional likelihoods on the three relations highlighted in the risk model of Figure 5.

In this example, we will assume that the conditional likelihood on the risk model relations we are interested in is equal to the likelihood that there is a vulnerability which can be exploited in order for the source of the relation to lead to/cause the target of the relation. This means, for instance, that the likelihood [0.001, 0.04] on the relation going from the threat scenario *I.CAPEC-62: Cross-site request forgery* to *F.CAPEC-62: Cross-site request forgery successful* in Figure 5, specifies the likelihood that there is a vulnerability which will be exploited to cause a successful cross-site request forgery attack if it is initiated.

In this example, we make use of the aggregation function *Vulnerability Existence Aggregation* specified in Table 11. In addition to the number of passed on failed test executions, this function takes

the following measurements as input: test vulnerability discovery likelihood, likelihood of false positives, and assumed vulnerability existence likelihood.

Table 13 – Metrics needed in the example

Metric ID	Metric name	Description	Scale/Type
R5	Vulnerability existence likelihood	Specifies the likelihood that a vulnerability exists in the SUT	Probability interval
RT1	Test vulnerability discovery likelihood	Probability that a test fails if the SUT contains the vulnerability which we are testing for	Probability interval
RT2	Likelihood of false positive	Probability that a test fails if the SUT does not contain the vulnerability which we are testing for	Probability interval
RT3	Assumed vulnerability existence likelihood	Assumed probability that system has vulnerability (before testing)	Probability interval
T1	# passed test cases	This indicates that no vulnerability has been found	Integer
T2	# failed test cases	This indicates that a vulnerability has been found	Integer

A summary of the metrics needed in the current example is given in Table 13. Furthermore, the dependencies between the metrics are shown in Figure 6. Note here that the metric "Assumed vulnerability existence likelihood" depends on the "Parameter" data element. The reason for this is that we let the assumed likelihood metric represent the likelihood value of the risk model which we are interested in updating based on the test results. Thus the "Assumed vulnerability existence likelihood" will actually be equal to a likelihood value represented by a Parameter element in the data model.

Since the two metrics #passed test cases and # failed test cases can be derived automatically from the test report, the only metrics which will have to be estimated manually be expert judgement are "Test vulnerability discovery likelihood" and "Likelihood of false positive" which are highlighted in gray color in Figure 6.

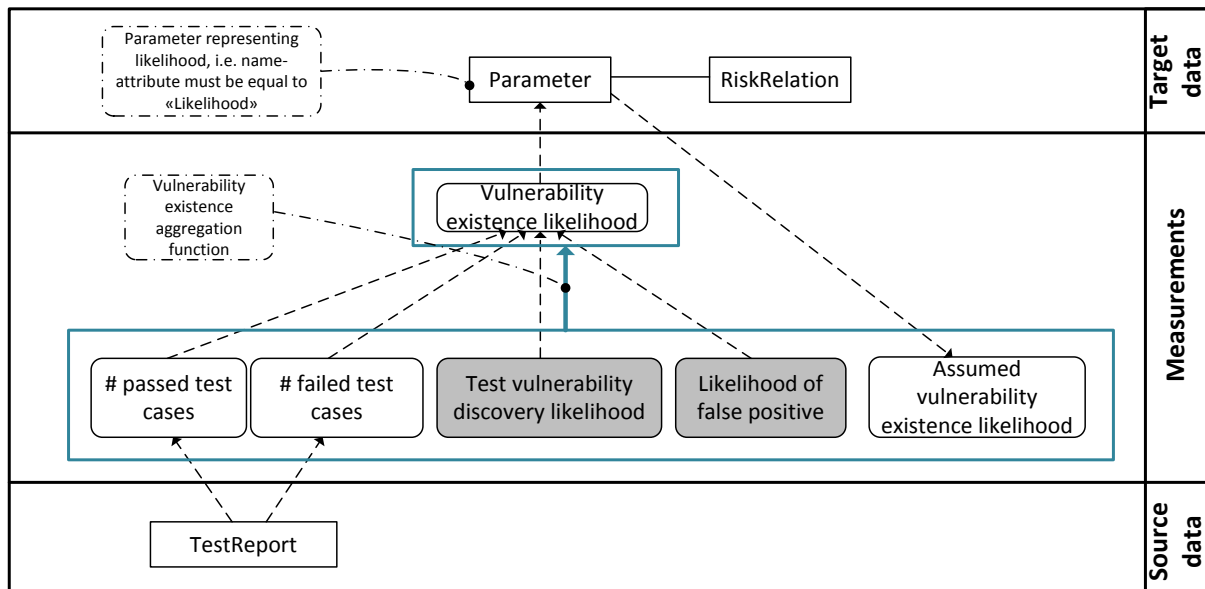


Figure 6 – Dependency graph of the example

3.3.3 Aggregation of test results

Having specified source and target data elements, measurements, and aggregation functions, we may apply the aggregation functions to a given set of test results in order to analyze how this impacts the risks of the risk model.

To illustrate this, assume that we have the test results of Table 14, where we for each relation we are testing, have zero fail test executions, meaning that the tests have revealed no vulnerabilities. This is often the case in practice. However, the fact that the tests did not reveal any vulnerabilities, does not mean that the system under test has no vulnerabilities. However, the more we test without finding any vulnerabilities, the more confident we can be that no vulnerabilities exist in the system.

Table 14 – Example of test results

Name	Passes	Fails
CAPEC 66: SQL injection	1000	0
CAPEC 62: Cross site request forgery	1500	0
CAPEC 63: Simple script injection	2000	0

In order to apply the aggregation function Vulnerability Existence Aggregation, we need, in addition to the number of passed and failed test cases, the measurements shown in Table 15. Here the "assumed vulnerability existence likelihood" measurement value is obtained from the risk model in Figure 5. The two other measurements need to be estimated based on expert judgement since there is no information about these in the test report assumed in the current example.

Table 15 – Example of measurement values

Name	False positives likelihood	Vulnerability test discovery likelihood	Assumed vulnerability existence likelihood
CAPEC 66: SQL injection	0.000001	0.001	[0.001, 0.04]
CAPEC 62: Cross site request forgery	0.000001	0.001	[0.0001, 0.01]
CAPEC 63: Simple script injection	0.000001	0.001	[0.0001, 0.004]

Given the measurement values of Table 15 and Table 16, we can apply the aggregation function to automatically calculate the "Vulnerability existence likelihood". The results of this is shown in Table 16.

Table 16 - Example of aggregated measurement values

Name	Vulnerability existence likelihood
CAPEC 66: SQL injection	[2.23314E-5, 0.00225]
CAPEC 62: Cross site request forgery	[3.68296E-4, 0.015104]
CAPEC 63: Simple script injection	[1.35482E-5, 5.43763E-4]

We may now update the conditional likelihoods of the three relations that are tested in the risk model, and recalculate the likelihood values of the risk model. The result of this is shown in Figure 7. Note in particular that likelihood values of the risk "UI-C2: Attacker gains privileges of users/roles and accesses confidential user data" is now calculated to be [0.002247, 99.853963] whereas the value was [0.0061,320] (as shown in Figure 7) prior to the update of the conditional likelihoods. Thus we see precisely how the test results have impacted the likelihood value of the risk in the risk model.

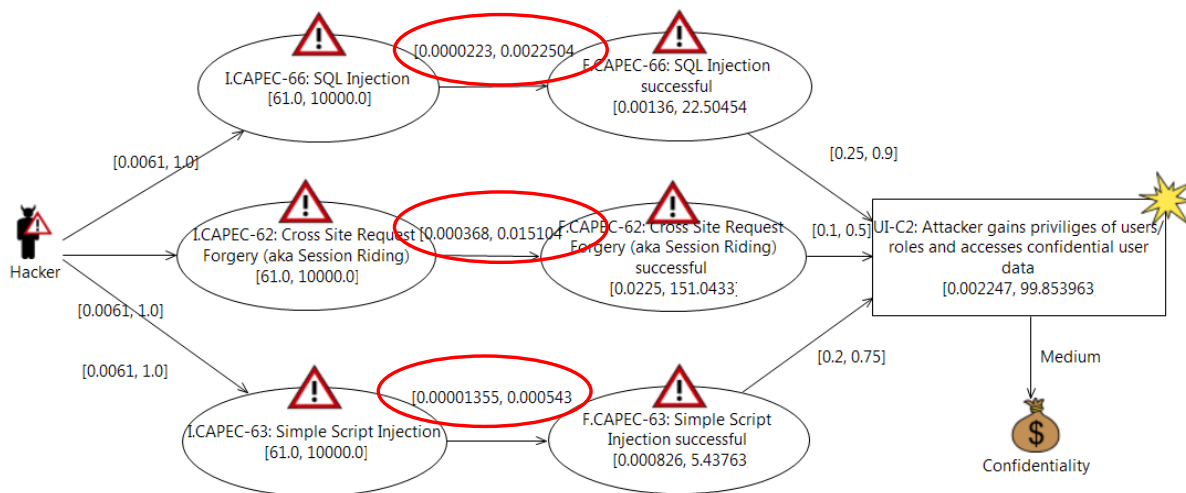


Figure 7 – Updated risk model

4 Reusable Risk Assessment Artifacts and Risk Aggregation with the Help of Tags and Scopes

Risk assessment of large scale ICT systems is a difficult task. One obvious approach to deal with its complexity and to keep its costs at a reasonable level is to identify the manageable small sub-components that the complex system consists of, and then analyze each of these sub-components separately. The results for the individual sub-components can later potentially be used as an adequate basis for a risk assessment of the entire system.

The idea is that even if the components are used to build different systems and if they are used in very different contexts, the risk analysis artifacts created for the individual components should be reusable for compositional risk assessment, i.e. for analyzing the overall risks without looking again into details of any sub-component. Finally it should be possible to aggregate and to evaluate the risks for entire products, configurations and usage scenarios at a very high level in order to support the management in their decision making processes.

But how exactly can such a composition and aggregation be done? Certainly, the risks of entire systems are not just the unions of the risks identified for their sub components. Depending on how the components are used, certain risks might not be relevant in a particular configuration. One possible reason might for example be that other components treat and mitigate the risk already. However in a slightly different configuration, the risk of exactly the same component might be unacceptably high.

For the aggregation of individual risk values, for example overall product level risks, it is not a sound idea to just sum up all the risk values for sub components and to calculate average high level risk values. Some components are eventually more critical than other components. Even summing them up in a weighted fashion according to the criticality of the components might be misleading. A security breach like execution of unauthorized code, even though exploiting a weakness in the most unimportant and least crucial component, could affect all other components that use the same logical system or database server. Weighting risk values low just because the components for which they were identified themselves seem to be uncritical would probably lead to completely wrong results.

Of course, it is possible to model the dependencies between the components in detail, as it is described for example in [6]. However, doing so requires a substantial amount of manual work for each and every system that is build out of the components. For large scale systems, graphical modelling might become unintuitive and analysts will probably get lost simply because the models get too complex.

Since one of the major requirements to the RASEN tooling, as specified by the case study provider Software AG, is exactly to avoid such manual work as much as possible, modelling all dependencies and relations in detail is not an option. The goal is rather to model very basic isolation levels and spread scopes in a simple, but nevertheless flexible, way.

4.1 Tagging Risk Assessment Artifacts

The isolation is modeled with the help of tags, which contain information about the involved resources like logical systems and databases. The tagging for risk assessment artifacts proposed here uses a simple format consisting of a tag category and a tag value. A number of different categories that might be useful in many scenarios have been identified so far:

- Component
- Product
- Configuration
- Physical system
- Logical system
- Process
- Network segment

- Database
- Database server
- Operating system
- Programming language
- Framework
- Third party API / library
- Task
- Building block

The RACOMAT tool supports these predefined categories, and other categories can also be specified as needed. Tags of predefined categories, as well as tags of user defined categories, can for example be used for grouping results in the dashboard view.

Besides the tag category, for each tag an arbitrary string identifier can be specified as a tag value. For example, a tag for the category *Database server* could have a value *PaymentTransactionServer*. However, the analyst must make sure that the same element is always identified with the same string in order to enable correct calculations. The RACOMAT tool therefore automatically suggests previously used values that users can apply with single mouse clicks.

Any node or relation in a risk graph can be tagged within the RACOMAT tool with multiple different tags. Even tags themselves can have tags, which are especially intended to support the modelling of different configurations. For each artifact, there may be multiple alternative tags having the same tag category, only with different tag values. It is, for example, possible to specify several different logical system tags for the same component. This might even make sense if the component will never run on more than a single logical system: Tags on the alternative tags can be used to express which alternative tag value should be used. Hence, with the help of configuration tags having different values, it is possible to make sure that for each configuration value, only exactly one of the alternative logical system tag values becomes applicable.

Most tags can typically be specified for entire components, which are represented by threat interfaces. So the amount of manual work should be reasonably low. For the RASEN use case work, for example, the RACOMAT tool imports information about systems, components and products from the ARIS tool, and it generates appropriate tags automatically. So the effort for tagging in the Software AG's case study is actually zero for some tag categories.

Certain tag categories are probably invariant for some systems, i.e. no matter where the system will be used and how it will be used, the tag values will always stay the same. Invariant categories typically specify information about system internals, e.g. used APIs. When reusing the risk assessment artifacts once created for some system, tags for the invariant categories do not have to be adopted even if the configuration and the environment of that system are changed.

On the other hand, there are tag categories which should be used to describe exactly where and how some system is used. Of course, these tag values that typically describe external aspects have to be adapted if the system setup changes. For example, software components and products can typically be installed on different logical and physical systems.

Tags can also be used to influence which elements should currently be shown. In that way, tags can be used like layers in common graphic programs. This might be helpful if graphs get large and complex. Also it is possible to let all set tag values for a certain category be displayed with different colors in order to get a good overview.

4.2 The Scope of Faults and Unwanted Incidents

Just having information about the different categories of isolation for the risk assessment artifacts is not enough to start any sound risk value aggregation. Incidents or faults might spread and affect other components. Instead of modeling in detail what could eventually be affected, it is much simpler to specify how far the consequences might reach.

The RACOMAT tool allows users to specify for any incident and for each tag if any other element having the same tag category and the same tag value might also be affected whenever the incident occurs.

For example, the entire logical system on which a certain component runs could be influenced by an incident *“Execute unauthorized code”* occurring on that component. Then any other faults and unwanted incidents of other components or programs running on the same logical system could also be triggered by that incident.

If dependencies had to be modeled manually within the risk graph, then eventually lots of new relations would have to be added. Figure 8 shows a risk graph with an explicitly modeled dependency between two components. Note that modeling the dependencies requires a concept like gates known from Fault Tree Analysis [3] to express how the triggering might work. In the example, the “or” gate (≥ 1) expresses that one of the threat scenarios is enough to trigger the incident “Leaks data”.

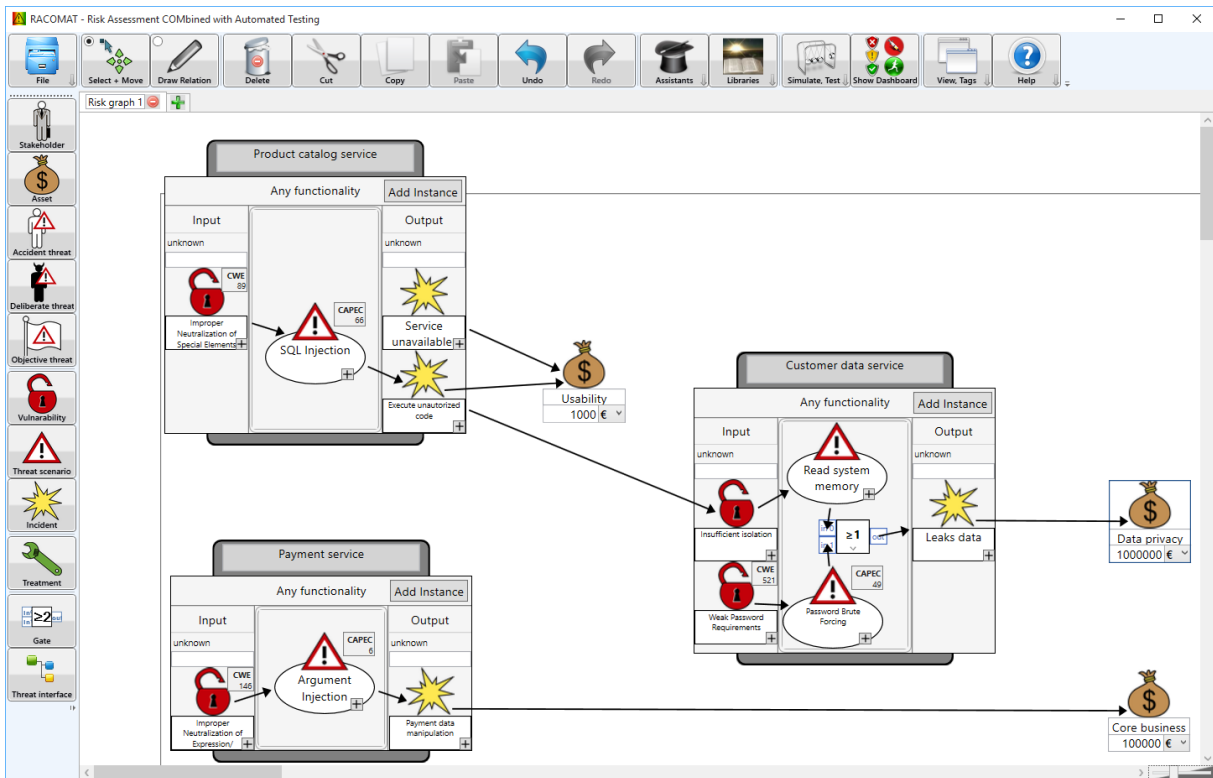


Figure 8 – Explicitly modeled dependencies

However, if all components are tagged, there is no need to model the dependencies in detail. Instead, as shown in the example, just the scope for the unwanted incident *“Execute unauthorized code”* can be specified to affect the entire logical system. Then all other faults or incidents having the same tag value “Main server” in the tag category “Logical system” will be treated as if a dependency was explicitly modeled. Hence, the tagged risk graph shown in Figure 9 is equivalent to the risk graph shown in Figure 8 and produces identical results in RACOMAT.

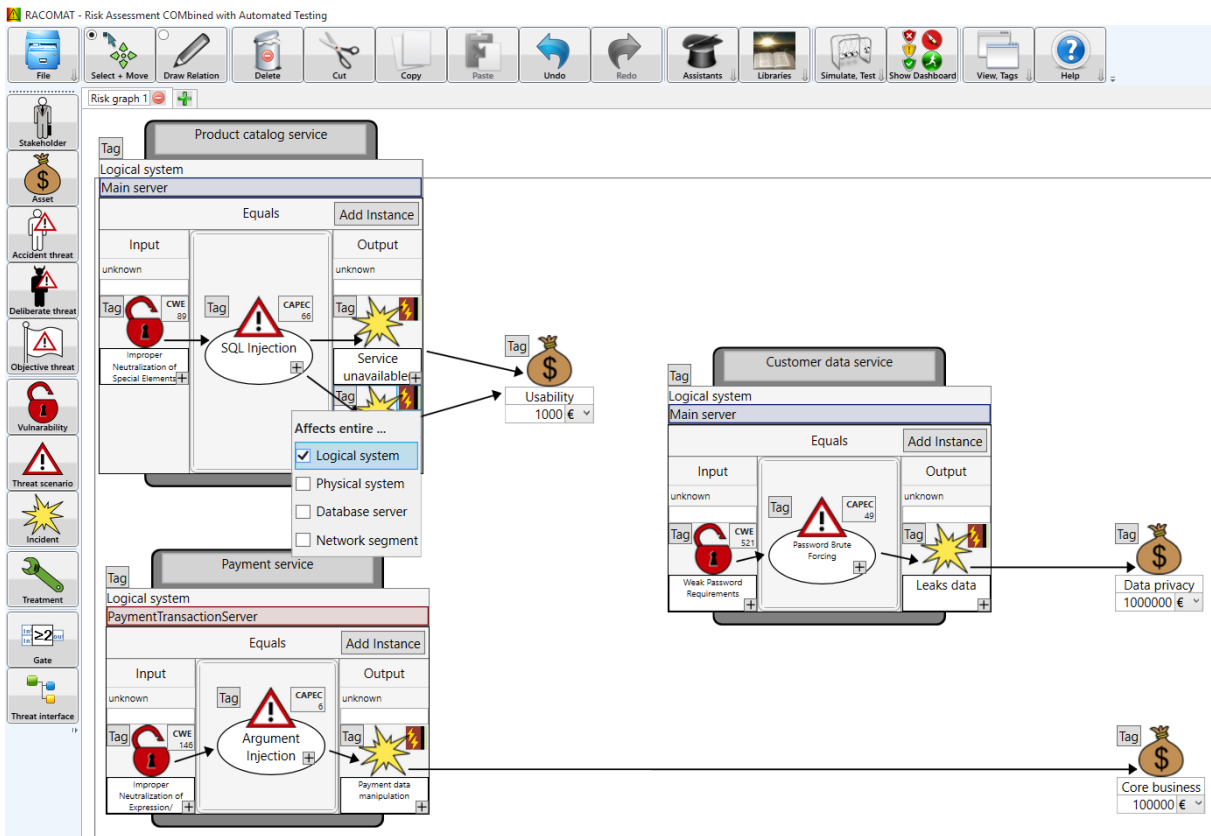


Figure 9 – Using tags and scopes to model dependencies

Ideally, there should be a library for the most common faults or unwanted incidents that specifies how far they typically might spread and affect other components. Such information could be added to an extended attack pattern library. Hence, it would become reusable and the amount of manual work would be further reduced.

4.3 Risk Aggregation Using Tags and Scopes

When aggregating the risk analysis results, the RACOMAT tool executes Monte Carlo Simulations in order to calculate likelihood values for dependent faults or unwanted incidents. This is a common approach, as described for example in [1][2].

The RACOMAT tool uses any available information to calculate how far consequences of some incident might spread and affect other incidents. Dependencies manually modeled with the help of relations in the risk graph have the highest priority. For example, manual modeling is currently needed to express redundant usage of components.

Wherever there are no dependencies explicitly modeled, before starting any simulations, the RACOMAT tool internally creates relations which follow from the isolation tags and the influence scope notations. Hence, the actual calculation of likelihood values is based on a fully modeled risk graph even if some relations are auto generated from tag and scope data. Therefore, the Monte Carlo Simulation always works as described in [8].

However, the complexity that might be introduced by automatically completing the model stays hidden. Users do not have to bother with expanded complex risk graphs. Changing configurations is possible at any point in time with limited amount of manual work required. Typically only some tags have to be altered.

Calculating likelihood values for some configuration is only a first step in the risk aggregation process. Typically, managers want high level results for example for entire products or even for complete products families.

The RACOMAT tool allows to aggregate risk values for any tag category. In RACOMAT risk is generally expressed always in the same unit – as the financial loss per time period. It is possible to calculate expected loss values, but it is also possible to calculate realistic and worst case scenarios. This can be very important since expected values will typically only occur in the average with large numbers. If the risk assessment is done for a single system for a single period of time, then there will be probably no loss at all – or a very high (eventually the maximal) possible loss if something goes wrong. Whatever happens, the expected value will never be reached. Where there are catastrophic high worst case losses, but low expected loss values, insurance might be a good idea to mitigate the risk, though an insurance will be more expensive than the expected loss value, for sure.

No matter if expected loss values, realistic or worst case losses are calculated, all risks are actually modeled as natural numbers having the same unit (i.e. currency per time span). Hence, it is possible to simply sum up all the risks having the same tag.

High level risks are displayed in the internal dashboard view by the RACOMAT tool and they can be exported so that they can be viewed in simple web browsers. Within the RACOMAT dashboard view, it is possible to switch between different tag categories that should be evaluated.

5 Conclusion

In this deliverable we have reported on the main results of RASEN WP3 tasks from the third and final year of the project. The results show our progress within all of the R&D tasks of WP3 of compositional risk assessment techniques, techniques for test-based security risk assessment and techniques for continuous security risk assessment. The presented techniques come with relevant modeling support, and they are moreover supported by the prototype tools of deliverable D3.3.3 that are integrated into the RASEN tool-box.

References

- [1] W. Gleißner, T. Berger: Auf nach Monte Carlo: Simulationsverfahren zur Risiko-Aggregation. RISKNEWS, Volume 1, Issue 1, pp. 30–37. Wiley (2004)
- [2] S. Greenland: Sensitivity Analysis, Monte Carlo Risk Analysis, and Bayesian Uncertainty Assessment. Risk Analysis 21(4), 579-584 (2001)
- [3] International Electrotechnical Commission: IEC 61025 Fault Tree Analysis (FTA) (2006)
- [4] MITRE: Common Weakness Enumeration (CWE) [ONLINE] Available at: <https://cwe.mitre.org/> [Accessed 22 September 2015]
- [5] MITRE: Common Weakness Scoring System (CWSS) [Online] Available at: <http://cwe.mitre.org/cwss/> [Accessed 22 September 2015]
- [6] J. Viehmann: Reusing risk Analysis results - An extension for the CORAS risk analysis method. In Proc. 4th International Conference on Information Privacy, Security, Risk and Trust (PASSAT'12), pp. 742-751. IEEE (2012)
- [7] J. Viehmann, F. Werner: Risk assessment and security testing of large scale networked systems with RACOMAT. In Proc. third International Workshop on Risk Assessment and Risk-Driven Testing (RISK'15). To appear.
- [8] J. Viehmann: Towards Integration of Compositional Risk Analysis Using Monte Carlo Simulation and Security Testing. In Proc. first International Workshop on Risk Assessment and Risk-Driven Testing (RISK'13). LNCS, vol. 8418, pp. 109-119. Springer (2014)

Appendix A: Import/Export Format for ARIS in XSD

The following XSD specifies the import/export Format in ARIS Business Architect to export the models for automated testing, and to import the weaknesses back into the graphical model.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="subComponentType">
    <xs:sequence maxOccurs="unbounded" minOccurs="0">
      <xs:element name="subComponentName" type="xs:string"/>
      <xs:element name="subComponent" type="subComponentType"/>
      <xs:element maxOccurs="unbound" minOccurs="0" name="cwe">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="1000"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="vignetteEntryType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="9"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="productName" type="xs:string"/>
  <xs:element name="vignette">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="HA" type="vignetteEntryType"/>
        <xs:element name="RC" type="vignetteEntryType"/>
        <xs:element name="MD" type="vignetteEntryType"/>
        <xs:element name="UE" type="vignetteEntryType"/>
        <xs:element name="GP" type="vignetteEntryType"/>
        <xs:element name="EA" type="vignetteEntryType"/>
        <xs:element name="BP" type="vignetteEntryType"/>
        <xs:element name="RD" type="vignetteEntryType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element maxOccurs="unbounded" minOccurs="0" name="component">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="componentName" type="xs:string"/>
        <xs:element name="subComponent" type="subComponentType"/>
        <xs:element maxOccurs="unbound" minOccurs="0" name="cwe">
          <xs:simpleType>
            <xs:restriction base="xs:integer">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="1000"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```



```
</xs:element>  
</xs:schema>
```