

	<b>Title: Initial methodologies for model-based security testing and risk-based security testing</b>	
	<b>Version:</b> 1.1 <b>Date :</b> 02.07.2012 <b>Pages :</b> 67	
	<b>Editor: Fredrik Seehusen</b>	
	<b>Reviewers:</b>	
	<b>To:</b> DIAMONDS Consortium	
<p>The DIAMONDS Consortium consists of:          Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Get IT, Giesecke &amp; Devrient, Grenoble INP, itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, TU Graz, University Oulu, VTT</p>		
<b>Status:</b>  <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input type="checkbox"/> Proposal <input checked="" type="checkbox"/> Final / Released	<b>Confidentiality:</b>  <input checked="" type="checkbox"/> Public      Intended for public use <input type="checkbox"/> Restricted      Intended for DIAMONDS consortium only <input type="checkbox"/> Confidential      Intended for individual partner only	
<b>Deliverable ID: D4_3_T2_T3</b>  <b>Title:</b>  <b>Initial methodologies for model-based security testing and risk-based security testing</b>  <b>Summary / Contents:</b>  This document constitutes the second deliverable for task 4.2 and task 4.3 of work package 4 on risk- and model-based security testing methodologies. While the other work packages of the DIAMONDS project describe <i>techniques/methods</i> and <i>tools</i> , work package 4 describes <i>processes/guidelines</i> for applying these tool and techniques in practice.  Contributors:  Nadja Menz, Johannes Viehmann (Fraunhofer FOKUS)  Gencer Erdogan, Yan Li, Fredrik Seehusen, Ketil Stølen (SINTEF)		

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	<p>Page : 2 of 67</p> <hr/> <p>Version: 1.1 Date : 02.07.2012</p> <hr/> <p>Status : Final Confid : Public</p>
---	---	---

## TABLE OF CONTENTS


<b>1. A conceptual framework for model-based security testing and risk analysis .....</b>	<b>7</b>
1.1 Risk Management .....	7
1.2 Risk Analysis .....	9
1.3 Security.....	10
1.4 Testing.....	11
1.5 Security Risk Analysis .....	15
1.6 Security Testing.....	16
1.7 Model.....	16
1.8 Model-based Security Risk Analysis (MSR).....	17
1.9 Model-based Security Testing (MST).....	17
1.10 Test-driven Model-based Security Risk Analysis (TMSR) .....	18
1.11 Risk-driven Model-based Security Testing (RMST).....	20
<b>2. A process for test-driven security risk analysis.....</b>	<b>22</b>
2.1 Step 1: Establish context and target of evaluation .....	24
2.2 Step 2: Risk identification .....	25
2.3 Step 3: Risk estimation.....	25
2.4 Step 4: Test identification .....	25
2.4.1 Identifying testable threat scenarios .....	25
2.4.2 Identifying security tests.....	26
2.5 Step 5: Test execution.....	26
2.5.1 Tools .....	27
2.5.2 Execution .....	28
2.6 Step 6: Risk consolidation and treatment.....	28
<b>3. an Evaluation of a PROCESS FOR Test-driven Security Risk Analysis Based on the norske solutions Case Study .....</b>	<b>28</b>
3.1 Research method and hypotheses.....	29
3.1.1 Risk models .....	29
3.1.2 Difference between risk models.....	31
3.1.3 Hypotheses.....	31
3.2 Overview of process for Test-driven Security Risk Analysis .....	31
3.3 Results.....	33
3.4 Discussion .....	35
3.4.1 Hypothesis 1 .....	35
3.4.2 Hypothesis 2 .....	36
3.4.3 Hypothesis 3 .....	36
3.5 Conclusion.....	37
<b>4. Extension for the CORAS risk analysis method.....</b>	<b>37</b>
4.1 Background .....	38
4.1.1 Information Security Indicators .....	38
4.1.2 Common Criteria for Information Technology Security Evaluation .....	38
4.1.3 Risk analysis with FTA, FME(C)A and probability theory .....	39
4.1.4 Risk analysis with the CORAS method.....	40
4.1.5 CORAS risk analysis complexity and difficulty.....	40
4.2 A Template Library for Model-Based Risk Analysis .....	41
4.2.1 Employing ISI and CC for Model-Based Risk Analysis.....	41
4.2.2 Exemplary Application to two DIAMONDS Case Studies .....	42

	<p><b>Review of security testing tools</b></p> <p>Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 3 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

4.3	Composition of Risk Analysis Artefacts .....	46
4.3.1	Creating reusable threat interfaces for components .....	46
4.3.2	Threat composition diagram .....	47
4.3.3	Composition with external threats and assets .....	55
4.4	Deriving and Comparing Risks .....	59
4.4.1	Comparing the risks of components and architectures .....	63
4.5	Conclusion, Related and Further Work .....	65
<b>5.</b>	<b>References .....</b>	<b>66</b>


## FIGURES

Figure 1	The overall risk management process (adapted from [p.14,[30]]).	8
Figure 2	Conceptual model for Risk Analysis.	10
Figure 3	Conceptual model for Security	11
Figure 4	Overall test process [p.7-p.22,[31]].	12
Figure 5	The testing process used in DIAMONDS, adapted from [31].	13
Figure 6	Conceptual model for Testing.	14
Figure 7	Conceptual model for Security Risk Analysis.	15
Figure 8	Conceptual model for Security Testing.	16
Figure 9	Conceptual model for Model.	17
Figure 10	General model-based testing setting [24].	18
Figure 11	Test-driven Model-based Security Risk Analysis process.	19
Figure 12	Risk-driven Model-based Security Testing process.	21
Figure 13	Overview of the steps in the process.	24
Figure 14	of a threat diagram for Norse Options (without likelihood and consequence values).	25
Figure 15	Test automation.	28
Figure 16	Example of a CORAS risk model.	31
Figure 17	Number of risk model elements before and after testing.	33
Figure 18	Number of risks and threat scenarios tested and updated.	34
Figure 19	Difference between risk models before and after testing.	35
Figure 20	General Class Structure.	39
Figure 21	Template Diagram based on Indicators and SFRs.	42
Figure 22	External Intrusions and Attacks.	45
Figure 23	Malfunctions.	45
Figure 24	Security Critical User Behaviour.	45
Figure 25	CORAS threat diagram.	46
Figure 26	Threat interface.	47
Figure 27	Threat composition diagram with three components.	48
Figure 28	Threat composition diagram with additional base components.	50
Figure 29	Threat composition diagram for power supply.	52
Figure 30	Threat composition diagram with three base services.	54
Figure 31	Difficulties to physically access the server rooms for the different human threats.	55
Figure 32..	Threat composition diagram with coalitions (excerpt 1).	57
Figure 33	Threat composition diagram with coalitions (excerpt 2, containing only the service unavailable top level incident, with probability value results).	58
Figure 34	Threat composition diagram for a single time-stamp service with assets and consequences.	60
Figure 35	Risk diagram for a single time-stamp service.	62
Figure 36	Risk comparison diagram.	64
Figure 37	Tracing Preservation.	65

	<p><b>Review of security testing tools</b></p> <p>Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 4 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final
		Confid : Public

## TABLES

Table 1 Likelihood scale .....	26
Table 2 Overview of the work shops held during the assessment.....	32
Table 5 TD6. <i>Usage of Insecure Protocols/Software</i> .....	43
Table 3 TD7. <i>Insufficient Password Policies/Practices</i> and .....	44
Table 6 Risk Function for Base Incidents .....	61


	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 5 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

## HISTORY

Vers.	Date	Author	Description
0.1	16/04/12	Fredrik Seehusen	Template created
0.2	18/04/12	Nadja Menz	Outline for Chapter 3.1
1.0	01/05/12	Fredrik Seehusen	Document finalised.
1.1	02/07/12	Nadja Menz	Revision of chapter 4

## APPLICABLE DOCUMENT LIST


Ref.	Title, author, source, date, status	DIAMONDS ID
1		

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 6 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

## EXECUTIVE SUMMARY

This document constitutes the second deliverable for task 4.2 and task 4.3 of work package 4 on risk- and model-based security testing methodologies. While the other work packages of the DIAMONDS project describe *techniques/methods* and *tools*, work package 4 describes *processes/guidelines* for applying these tool and techniques in practice.

This deliverable has four sections. First, in Section 1, we describe a conceptual framework defining the main concepts related to model-based security testing risk-based testing and their relationships. The conceptual framework serves a basis for defining methodologies for risk- and model-based security testing. In Section 2, we present an initial process for test-driven security risk assessment which was used in a DIAMONDS case study. This process has been evaluated, and the results of the evaluation are presented in Section 3. Finally, Section 4 presents a method to increase the efficiency of the risk analysis process in the setting of model-based risk assessment.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 7 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

## 1. A CONCEPTUAL FRAMEWORK FOR MODEL-BASED SECURITY TESTING AND RISK ANALYSIS

This chapter documents the conceptual framework clarifying the notions of security testing, risk analysis, and related concepts, as well as defining the relations among them.

The conceptual framework offers a basis for future research in the project by providing a common understanding of the central notions within security testing and security risk analysis. Our approach is to build the conceptual framework upon established concepts from state-of-the-art. However, we also make adjustments of established notions where seen necessary or appropriate, in order to achieve a consistent framework suitable for the particular target of the DIAMONDS project.

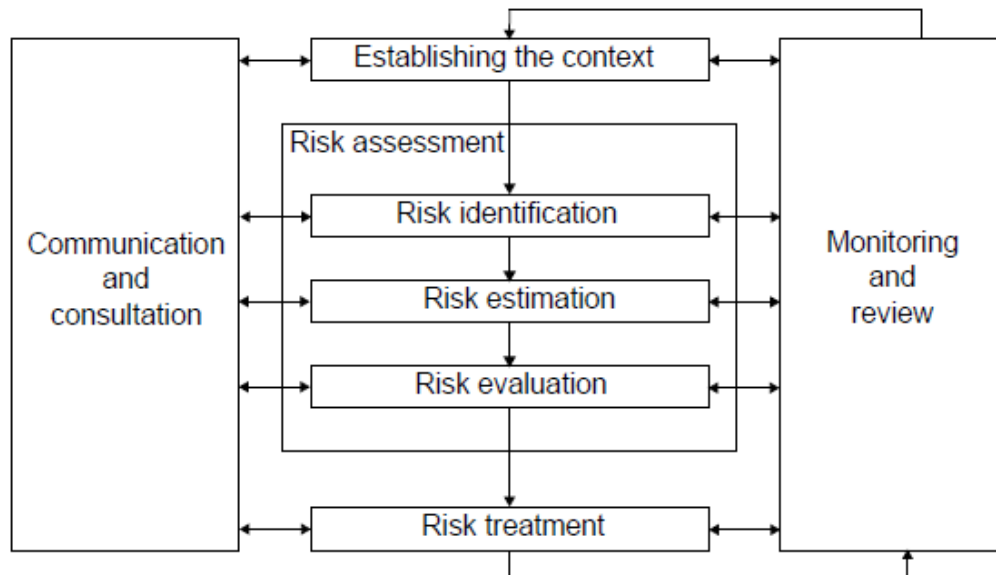
In DIAMONDS, we focus on model-based approaches to security testing and security risk analysis, where models are used as a main artefact both during the testing/analysis process and for documentation purposes. In particular we distinguish between model-based security testing (MST) and model-based security risk analysis (MSR). For combining MST and MSR, there are two main possibilities depending on which approach is taken as the starting point, i.e., the main purpose of the process. We will refer to these as risk-driven model-based security testing (RMST) and test-driven model-based security risk analysis (TMSR).

The remainder of this chapter is organized as follows: We start by introducing the overall risk management process in Section 1.1, before considering the basic notions of risk analysis, security, and testing in Sections 1.2, 1.3, and 1.4, respectively. In Sections 1.5 and 1.6 we build on the previous sections when presenting the definitions for security risk analysis and security testing, respectively. The basic concepts for models are presented in Section 1.7. In Sections 1.2-1.7 we use definitions from standards as much as possible, and additionally provide our interpretation of the relationships between the various concepts. The definitions of model-based security risk analysis (MSR) and model-based security testing (MST) are defined in Sections 1.8 and 1.9, respectively. Finally, we provide our proposal for test-driven model-based security risk analysis (TMSR) in Section 1.10, and for risk-driven model-based security testing (RMST) in Section 1.11.

### 1.1 RISK MANAGEMENT

It is necessary to introduce the overall risk management process before we look closer into risk analysis and other relevant concepts. ISO 31000 Risk management - Principles and guidelines [30] is our main building block in terms of defining and explaining the concept of risk management and the concepts related to risk management. The overall risk management process shown in Figure 1 is taken from [p.14,[30]].

There is however one deviation: Our definition of risk estimation is equivalent to what ISO 31000 refers to as risk analysis. Instead we use the term risk analysis in line with how the term is used in practice to denote the five step process in the middle of Figure 1 starting with Establishing the context and ending with Risk treatment.




**Figure 1 The overall risk management process (adapted from [p.14,[30]]).**

As stated in ISO31000 [p.1,[30]], all activities of an organization may involve risk. Organizations usually manage risk by identifying it, analyzing it and then evaluating it to see whether the risk should be modified by risk treatment in order to satisfy the risk evaluation criteria. Through this process, communication and consultation are carried out with stakeholders to monitor and review the risk as well as controls that are modifying the risk, which ensures no further risk treatment is required. Risk management can be applied to an entire organization, at its many areas and levels, at any time, as well as to specific functions, projects and activities. Although the practice of risk management has been developed over time and within many sectors in order to meet diverse needs, the adoption of consistent processes within a comprehensive framework can help to ensure that risk is managed effectively, efficiently and coherently across an organization.

- **Risk Management**  
Risk management refers to the coordinated activities to direct and control an organization with regard to risk [p.2,[30]].
- **Risk Management Process**  
Risk management process is the systematic application of management policies, procedures and practices to the activities of communicating, consulting, establishing the context, and identifying, analyzing, evaluating, treating, monitoring and reviewing risk [p.3,[30]].
- **Communication and Consultation**  
Communication and consultation refers to the continual and iterative processes that an organization conducts to provide, share or obtain information and to engage in dialog with stakeholders regarding the management of risk [p.3, [30]].
- **Establishing the Context**  
Establishing the context refers to the process of defining the external and internal parameters to be taken into account when managing risk, and setting the scope and risk criteria for the remaining process (adapted from [p.3, [30]]).
- **Risk Assessment**  
Risk assessment is the overall process of risk identification, risk estimation and risk evaluation [p.4, [30]].

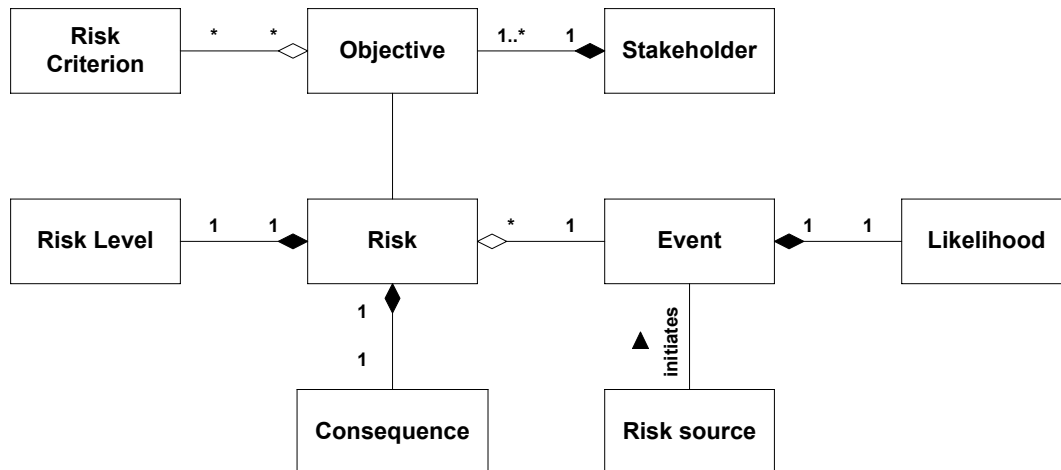


	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 9 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

- **Risk Identification**  
Risk identification is the process of finding, recognizing and describing risks. This involves identifying sources of risk, areas of impacts, events (including changes in circumstances), their causes and their potential consequences. Risk identification can involve historical data, theoretical analysis, informed and expert opinions, and stakeholder's needs [p.4, [30]].
- **Risk Estimation**  
Risk estimation is the process to comprehend the nature of risk and to determine the level of risk. This involves developing an understanding of the risk. Risk estimation provides the basis for risk evaluation and decisions on whether risks need to be treated, and on the most appropriate risk treatment strategies and methods (adapted from [p.5, [30]]).
- **Risk Evaluation**  
Risk evaluation is the process of comparing the results of risk estimation with risk criteria to determine whether the risk and/or its magnitude is acceptable or tolerable. Risk evaluation assists in the decision about risk treatment (adapted from [p.6, [30]]).
- **Risk Treatment**  
Risk treatment is the process to modify risk. This can involve avoiding the risk by deciding not to start or continue with the activity that gives rise to risk , or taking or increasing risk to pursue an opportunity [p.6, [30]].
- **Risk Analysis**  
Risk Analysis is a collective term defining the process consisting of the following steps: Establishing the context, Risk identification, Risk estimation, Risk evaluation and Risk treatment (adapted from [p.14, [30]]).
- **Monitoring**  
Monitoring is the continual checking, supervising, critically observing or determining the status in order to identify change from the performance level required or expected [p.7, [30]].
- **Review**  
Review is the activity undertaken to determine the suitability, adequacy and effectiveness of the subject matter to achieve established objectives [p.7, [30]].

## 1.2 RISK ANALYSIS

The conceptual model and notions defined here are based on the ISO 31000 standard [30]. Figure 2 shows the conceptual model for risk analysis adopted by the DIAMONDS project.




**Figure 2 Conceptual model for Risk Analysis.**

- **Risk**  
Risk is the combination of the consequences of an event with respect to an objective and the associated likelihood of occurrence (adapted from [p.1, [30]]).
- **Objective**  
An objective is something the stakeholder is aiming towards or a strategic position it is working to attain (adapted from [34]).
- **Risk Source**  
Risk source is an element which alone or in combination has the intrinsic potential to give rise to risk [p.4, [30]].
- **Stakeholder**  
Stakeholder is a person or organization that can affect, be affected by, or perceive themselves to be affected by a decision or activity [p.4, [30]].
- **Event**  
Event is the occurrence or change of a particular set of circumstances [p.4, [30]].
- **Likelihood**  
Likelihood is the chance of something happening [p.5, [30]].
- **Consequence**  
Consequence is the outcome of an event affecting objectives [p.5, [30]].
- **Risk Criterion**  
A risk criterion is the term of reference against which the significance of a risk is evaluated [p.5, [30]].
- **Risk Level**  
Risk level is the magnitude of a risk or combination of risks, expressed in terms of the combination of consequences and their likelihood [p.6, [30]].

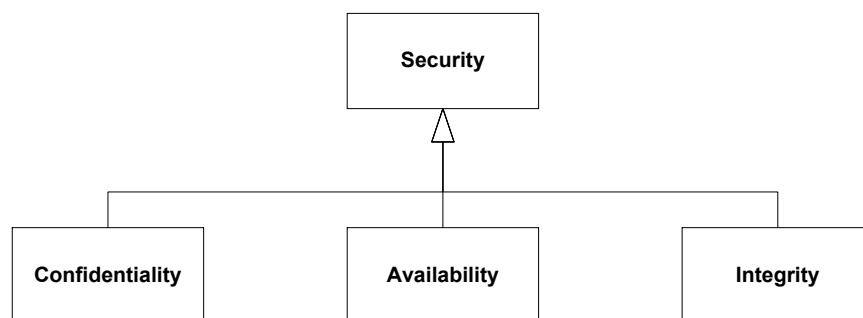
### 1.3 SECURITY

The terms information security, computer security and information assurance are frequently used interchangeably. These fields are often interrelated and share the common goals of protecting the confidentiality, integrity and availability of information; however, there are some subtle differences between them [23]

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 11 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

These differences lie primarily in the approach to the subject, the methodologies used, and the areas of concentration. Information security is mainly concerned with the confidentiality, integrity and availability of data regardless of the form the data may take, such as electronic, print, or other forms. Computer security mainly focuses on ensuring the availability and correct operation of a computer system without concern for the information stored or processed by the computer. Information assurance focuses on the reasons for assurance that information is protected, and is thus reasoning about information security [23].

In the DIAMONDS project we use the term security in the meaning of information security, where the definition of information security has been taken from ISO IEC 27000-Information Security Management System [29].



**Figure 3 Conceptual model for Security**


- **Security**  
Security refers to the preservation of confidentiality, integrity and availability of information (adapted from [p.3,[29]]).
- **Confidentiality**  
Confidentiality is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [p.2,[29]].
- **Availability**  
Availability is the property of information being accessible and usable upon demand by an authorized entity [p.2,[29]].
- **Integrity**  
Integrity is the property of protecting the accuracy and completeness of information (adapted from [p.4,[29]]).

## 1.4 TESTING

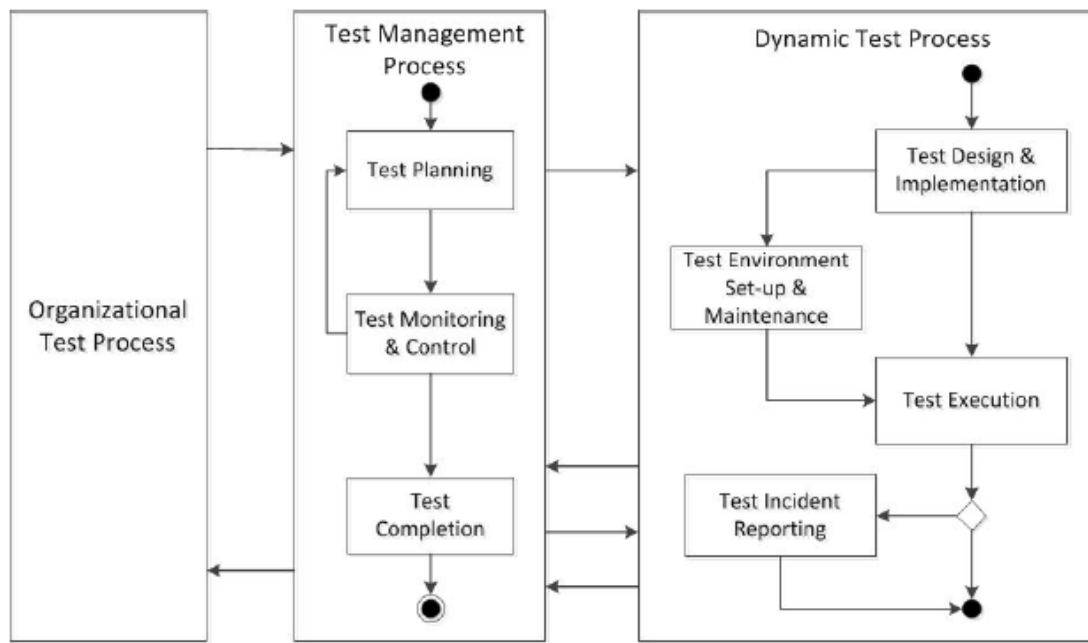
In this section, we particularly focus our conceptual clarification on software testing related to the DIAMONDS project. Our primary source for the notion of software testing and related notions is the upcoming international standard ISO/IEC 29119 Software Testing [31] defined by Software and Systems Engineering Standards Committee of the IEEE Computer Society.

However, ISO/IEC 29119 is still under development, and we only have access to a draft version of Part 2 (ISO/IEC 29119 Draft Part 2-Testing Process) at the time of writing. For related notions not found in Part 2, we use IEEE 829 [28] and BS 7925-1/-2 [33], which are expected to be incorporated into ISO/IEC 29119.

As stated in ISO/IEC TR 19759 [25], testing concepts, strategies, techniques, and measures need to be integrated into a defined and controlled process which is run by people. The testing process provides support for testing activities as well as guidance for testing teams. It provides justified assurance that the test objec-

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 12 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

tives will be met cost-effectively. We introduce the overall testing process (see Figure 4 defined in ISO/IEC 29119 [31] before we define the specific testing process used in the DIAMONDS project.



**Figure 4 Overall test process [p.7-p.22,[31]].**

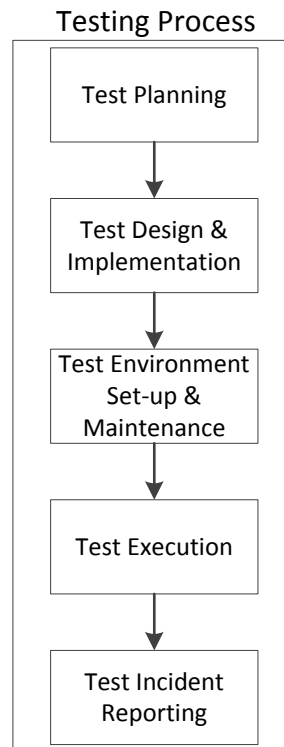
The testing activities that are performed during the life cycle of a software system may be grouped into a three layers test process [p.2,[31]], as shown in Figure 4.

The aim of the organizational test process layer is to define a process for the creation and maintenance of organizational test specifications, such as organizational test policies, strategies, processes, procedures and other assets [p.2,[31]].

The aim of the test management process layer is to define processes that cover the management of testing for a whole test project or any test phase or test type within a test project (e.g. project test management, system test management, performance test management) [p.2,[31]].

The aim of the dynamic test process layer is to define generic processes for performing dynamic testing. Dynamic testing may be performed at a particular phase of testing (e.g. unit, integration, system, and acceptance) or for a particular type of testing (e.g. performance testing, security testing, and functional testing) within a test project [p.2,[31]].

What we refer to at the testing process in the DIAMONDS project is basically what Figure 4 refers to as the dynamic test process. However, as indicated by Figure 5 we also add a test planning step capturing the test planning of the test management process of relevance for one run of the dynamic test process.



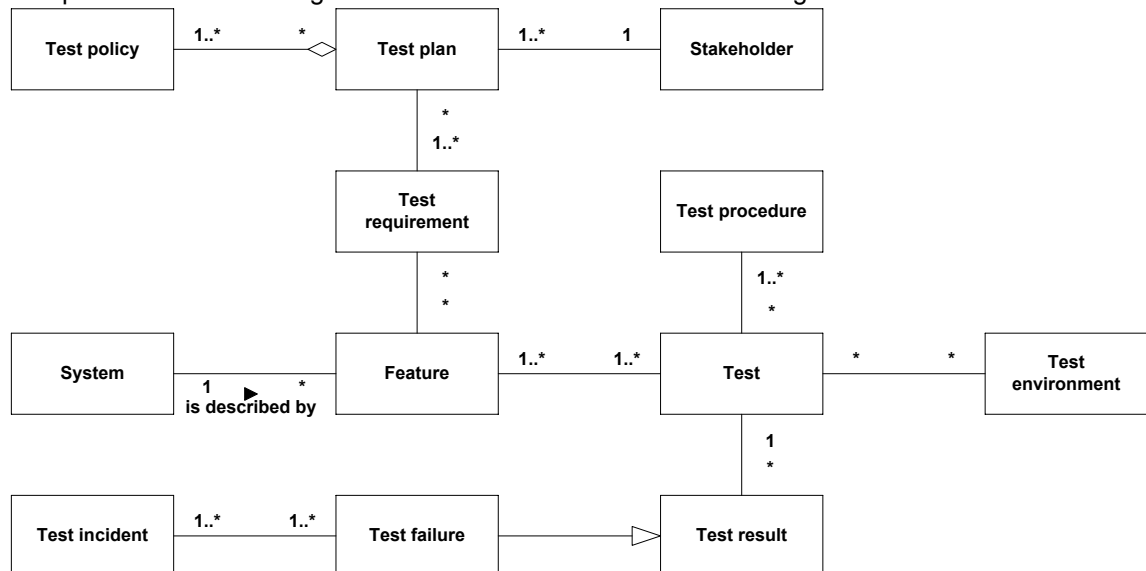
**Figure 5 The testing process used in DIAMONDS, adapted from [31].**

- **Testing**  
Testing is the process of exercising the system to verify that it satisfies specified requirements and to detect errors (adopted from [33]).
- **Test Planning**  
The test planning is the process of developing the test plan. Depending on where in the project this process is implemented this may be a project test plan or a test plan for a specific phase, such as a system test plan, or a test plan for a specific type of testing, such as a performance test plan (adopted from [p.8,[31]]).
- **Test Design and Implementation**  
The test design and implementation is the process of deriving the test cases and test procedures (adopted from [p.23,[31]]).
- **Test Environment Set-up and Maintenance**  
The test environment set-up and maintenance process is the process of establishing and maintaining the environment in which tests are executed (adopted from [p.27,[31]]).
- **Test Execution**  
The test execution is the process of running the test procedure resulting from the test design and implementation process on the test environment established by the test environment set-up and maintenance process. The test execution process may need to be performed a number of times as all the available test procedures may not be executed in a single iteration (adopted from [p.28,[31]]).

- **Test Incident Reporting**

The test incident reporting is the process of managing the test incidents. This process will be entered as a result of the identification of test failures, instances where something unusual or unexpected occurred during test execution, or when a retest passes (adopted from [p.30,[31]]).

The conceptual model for testing used in the DIAMONDS is defined in Figure 6.



**Figure 6 Conceptual model for Testing.**

- **Test Policy**

The test policy is a document that describes the purpose, goals, and overall scope of the testing within the organization (adopted from [p.4,[31]]).

- **Test Plan**

Test plan is a document describing the scope, approach, resources, and schedule of intended test activities [p.11,[28]].

- **Test Requirement**

Test requirement is a capability that must be met or possessed by the system (requirements may be functional or non-functional) - (adopted from [33]).

- **System**

A system is an interacting combination of elements that aim to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements [p.2-3,[25]].


- **Feature**

Feature is a distinguishing characteristic of a system item (includes both functional and nonfunctional attributes such as performance and reusability) [p.9,[28]].

- **Test**

Test<sup>1</sup> is a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement (adopted from [33]).

<sup>1</sup> Note that the term test as defined here is sometimes referred to as test case in other resources.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 15 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

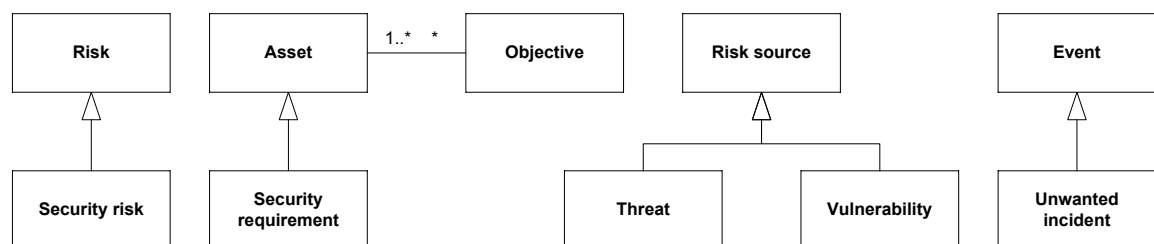
- **Test Procedure**  
Documentation that specifies a sequence of actions for the execution of a test [28].
- **Test Environment**  
Test environment is the description of the hardware and software environment in which the tests will be run, and any other software with which the software under test interacts (adopted from [33]).
- **Test Incident**  
Test incident is an unplanned event occurring during testing that has a bearing on the success of the test. Most commonly raised when a test result fails to meet expectations [33].
- **Test Failure**  
Test failure is the deviation of the software from its expected delivery or service (adopted from [33]).
- **Test Result**  
A test result is an actual outcome or a predicted outcome of a test (adopted from [33]).

## 1.5 SECURITY RISK ANALYSIS

Lund et al. [10] classify risk analysis approaches into two main categories:

- Offensive approaches: Risk analysis concerned with balancing potential gain against risk of investment loss. This kind of risk analysis is more relevant within finance and political strategy making.
- Defensive approaches: Risk analysis concerned with protecting what is already there.


In the context of security, the defensive approach is the one that is relevant.



**Figure 7 Conceptual model for Security Risk Analysis.**

For the definitions of Risk, Objective, Risk source and Event see Section 1.3.

- **Security Risk Analysis**  
Security risk analysis is the process of risk analysis specialized towards security.
- **Asset**  
Asset is anything that has value to the stakeholders (adopted from [29]).
- **Security Requirement**  
Security requirement is a specification of the required security for the system (adopted from [33]).
- **Security Risk**  
Security risk is a risk caused by a threat exploiting a vulnerability and thereby violating a security requirement.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 16 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

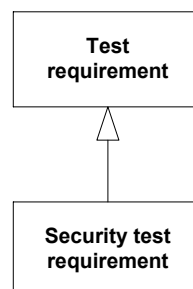
- **Unwanted Incident**  
Unwanted incident is an event representing a security risk.
- **Threat**  
Threat is potential cause of an unwanted incident [29].
- **Vulnerability**  
Vulnerability is weakness of an asset or control that can be exploited by a threat [29].

## 1.6 SECURITY TESTING

Based on the notions of security and testing we define security testing as follows:

- **Security Testing**  
Security testing is the process of testing specialized towards security.

To be more specific, we can understand security testing as the testing and/or evaluation of the management, operational, and technical security controls to determine the extent to which the controls are implemented correctly, operating as intended, and producing the desired outcome with respect to meeting the security requirements for the system or enterprise [27]. The basic security concepts that need to be covered by security testing are: confidentiality, availability, and integrity.



**Figure 8 Conceptual model for Security Testing.**

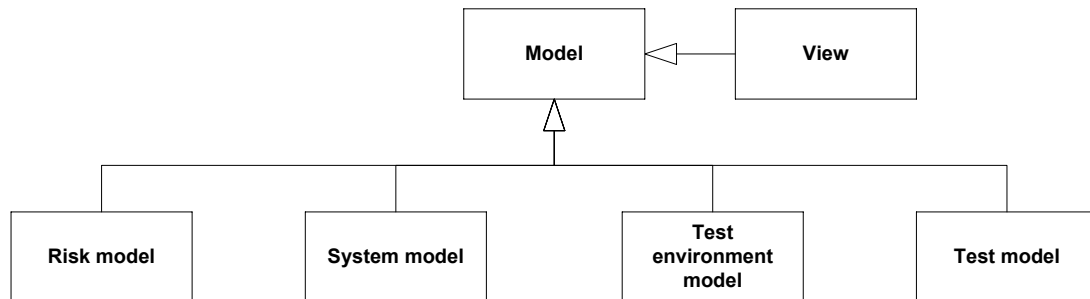
For the definition of Test Requirement see Section 1.4

- **Security Test Requirement**  
A security test requirement is a test requirement specialized towards security.

## 1.7 MODEL

This section clarifies the notion of model (see Figure 9). The concepts described here are based on TOGAF (The Open Group Architecture Framework) [32], System Analysis and Design [26] and SWEBOK (Software Engineering Body of Knowledge) [25].





**Figure 9 Conceptual model for Model.**

- **Model**  
Model is a representation of a subject of interest. A model provides a smaller scale, simplified, and/or abstract representation of the subject matter (adopted from [32]).
- **System Model**  
A system model represents a system.
- **View**  
A view is the representation of a related set of concerns.
- **Risk Model**  
A risk model represents risks.
- **Test Model**  
A test model represents tests.
- **Test Environment Model**  
A test environment model represents the test environment.

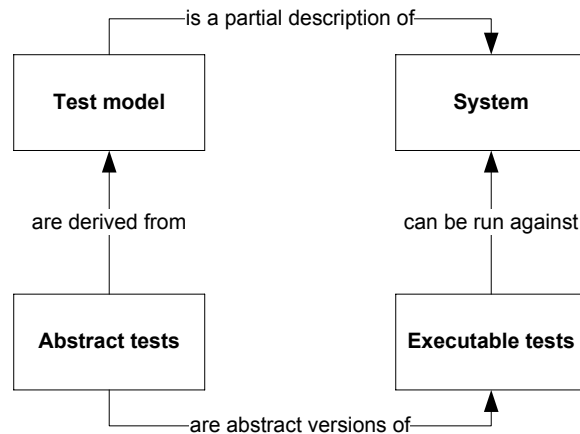
## 1.8 MODEL-BASED SECURITY RISK ANALYSIS (MSR)

Based on the notions of model and security risk analysis we define the term model-based security risk analysis as follows:

- **Model-Based Security Risk Analysis (MSR)**  
Model-based security risk analysis is security risk analysis in which each step of the process includes the construction and analysis of models.

## 1.9 MODEL-BASED SECURITY TESTING (MST)

Model-based testing is a software testing approach that relies on models of a system under test and its environment to derive test cases. Usually, the testing model is derived in whole or in part from a model that describes functional or non-functional aspects (e.g. performance, security, ergonomics) of the system under development [24]. Figure 10 illustrates a general model-based testing setting.




**Figure 10 General model-based testing setting [24].**

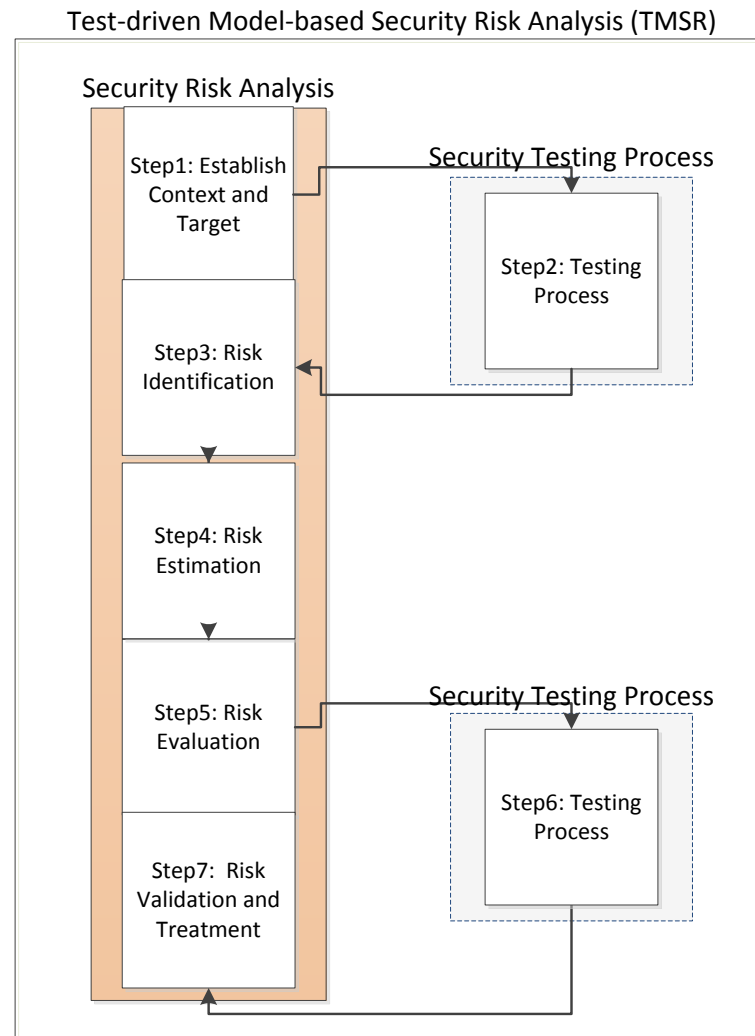
- **Model-based Security Testing**  
Model-based security testing is security testing that involves the construction and analysis of a system model, a test model and a test environment model to derive tests.

### 1.10 TEST-DRIVEN MODEL-BASED SECURITY RISK ANALYSIS (TMSR)

Test-driven Model-based Security Risk Analysis (TMSR) is defined as the combination of security risk analysis and security testing in which security testing is carried out both before and after the security risk assessment process. The first usage (i.e., testing *before* the security risk assessment process) supports the security risk analysis process by identifying potential risks, while the second usage (i.e., testing *after* the security risk assessment process) is used to validate security risk models based on security test results. Figure 11 illustrates the process of TMSR where risk validation is added in Step 7.

- **Test-driven Model-based Security Risk Analysis (TMSR)**  
Test-driven Model-based Security Risk Analysis (TMSR) is model-based security risk analysis that use testing within the risk analysis process.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 19 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public



**Figure 11 Test-driven Model-based Security Risk Analysis process.**

The testing process in Step 2 and Step 6 is the testing process defined in Section 1.4 specialized towards security.


The following describes the steps in the TMSR process including the input and output for each step in the process.

#### **Step 1: Establish Context and Target**

The first step is the initial preparation for risk analysis in which a basic idea about target, scale of analysis and corresponding context are established. This can be performed by an introductory meeting with the customer on the behalf of which the analysis is conducted. The representatives of the customer can present their overall goals of the analysis and the target they wish to have analyzed in structured meetings, in which a common initial understanding of the target for analysis is established. The overall goals of the analysis are defined and the rest of the analysis is planned.

- **Input:** Objective
- **Output:** Risk criteria, System model, Assets that need to be defended

#### **Step 2: Testing Process**

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 20 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

Based on the identified assets, relevant components of the system are identified and considered as test targets. Test models are then built according to the test targets and security requirements, and are further used to generate tests. After tests have been executed the result is reported and directed as an input to Step 3 where it supports the security risk analysis process to identify potential security risks.

- **Input:** Risk criteria, System model, Assets that need to be defended
- **Output:** Test result

### Step 3: Risk Identification

The risk identification involves a systematic identification of threats, unwanted incidents, and vulnerabilities with respect to the identified assets and the security test results from Step 2. Particular focus will be put upon security risks that are related to software functionality and requirements. Taxonomy-based questionnaires or risk checklists may be used by project members in a structured brainstorm meeting.

- **Input:** Risk criteria, System model, Assets that need to be defended, Test result
- **Output:** Incomplete risk model

### Step 4: Risk Estimation

The main goal here is to define the likelihoods and consequences of the risks identified in Step 3. These values in combination indicate the risk level for each of the identified risks. The risk estimation can be conducted as a brainstorming session involving personnel with various backgrounds.

- **Input:** Risk criteria, System model, Assets that need to be defended, Incomplete risk model
- **Output:** Risk model

### Step 5: Risk Evaluation

The purpose of risk evaluation is to assist in making decisions, based on the outcomes of risk estimation, about which risks need treatment and the priority for treatment implementation.

- **Input:** Risk criteria, System model, Assets that need to be defended, Risk model
- **Output:** Risk prioritized with respect to risk criteria

### Step 6: Testing Process

Based on the prioritized risks, relevant system components are analyzed and considered as security test targets. Test models are then built according to test targets and security requirements. Security tests are then created and executed based on the security test models. After tests have been executed the result is reported and directed as an input to Step 7 where it supports the security risk analysis process to validate the security risk models.

- **Input:** Risk criteria, System model, Assets that need to be defended, Risk model, Risk prioritized with respect to risk criteria
- **Output:** Test result


### Step 7: Risk Validation and Treatment

Security testing results from Step 6 are used to validate the risk models. The unacceptable risks will be further evaluated with possible treatment to reduce likelihood and negative consequences, according to risk evaluation criteria.

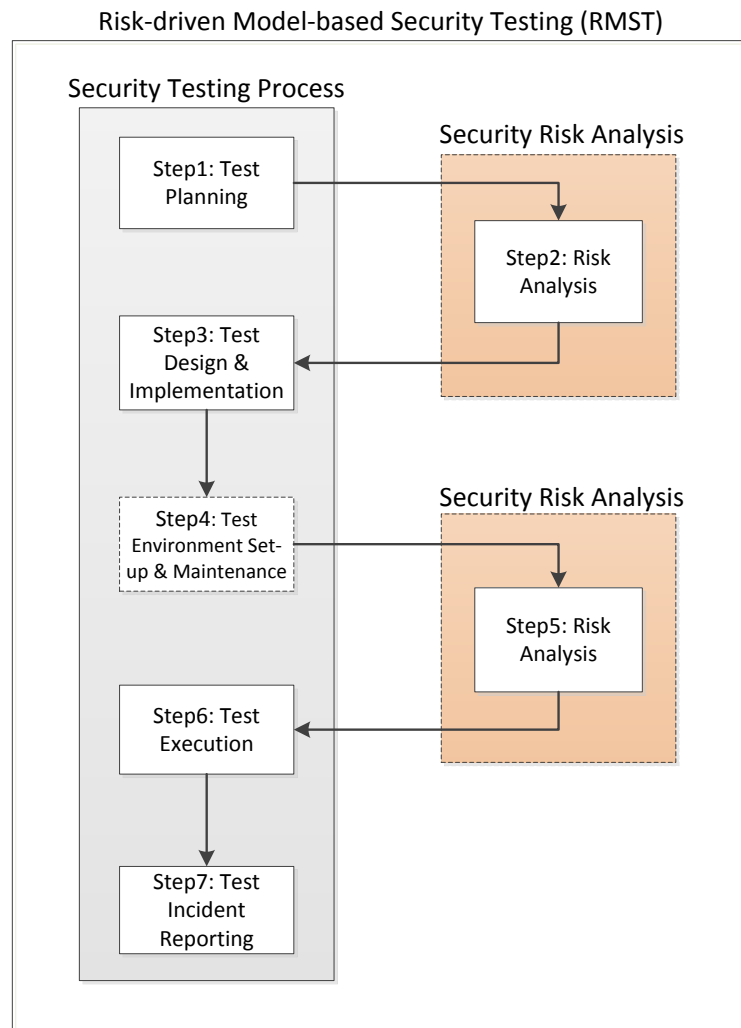
- **Input:** Risk criteria, System model, Assets that need to be defended, Risk model, Risk prioritized with respect to risk criteria, Test result
- **Output:** Updated risk model, Treatment

## 1.11 RISK-DRIVEN MODEL-BASED SECURITY TESTING (RMST)

Risk-driven Model-based Security Testing (RMST) is defined as the combination of security testing and security risk analysis in which security risk assessment is carried out both before and after the Test Design \& Implementation step (Figure 12). The first usage (i.e., security risk assessment *before* the Test Design \& Implementation step) supports the security testing process by identifying the most important parts of the system under test, while the second usage (i.e., security risk assessment *after* the Test Design \& Implemen-

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 21 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

tation step) supports the security testing process by identifying the most important security tests that needs to be executed.



**Figure 12 Risk-driven Model-based Security Testing process.**

The security risk analysis process in Step 2 and Step 5 is equivalent to the security risk analysis process given in Figure 11.


- **Risk-driven Model-based Security Testing (RMST)**  
Risk-driven Model-based Security Testing (RMST) is model-based security testing that use risk assessment within the security testing process.

The following describes the steps in the RMST process including the input and output for each step in the process.

#### **Step 1: Test Planning**

Test planning here can be described as preparation for further security testing activities in which testing conditions (e.g. procedures, scope, resources, administration, effort) are defined according to security requirements. Basically, the test planning is established in terms of "how to test?" and "where to test?".

- **Input:** System model, Test policy
- **Output:** Test plan, Security test requirement

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 22 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

### Step 2: Risk Analysis

The security risk analysis process in this step is carried out in order to identify the most important parts of the system that needs to be tested.

- **Input:** System model, Test policy, Test plan, Security test requirement
- **Final Output:** Risk, Risk criterion

### Step 3: Test Design and Implementation

Based on the risk analysis results from Step 2, the specific test targets are identified. Test models are built and tests are created from these models.

- **Input:** System model, Test plan, Security test requirement, Risk, Risk criterion
- **Output:** Test model, Test, Test procedure

### Step 4: Test Environment Set-up and Maintenance

The environment in which tests are executed is established and maintained in this step. Maintenance of the test environment may involve changes based on the results of previous tests. Where change and configuration management processes exist, changes to the test environments may be managed using these processes [31].

- **Input:** System model, Test plan, Test model, Test, Test procedure
- **Output:** Test environment model

### Step 5: Risk Analysis

The security risk analysis process in this step is carried out in order to identify and prioritize the most important security tests that need to be executed.

- **Input:** Test plan, Test model, Test, Security test requirement
- **Final Output:** Risk, Risk criterion

### Step 6: Test Execution

Security tests are prioritized and executed based on the results from the risk analysis in Step 5. The test execution may be iterative according to the complexity, scope, and attribute of the test targets.

- **Input:** Test plan, Test model, Test, Test procedure, Test environment model, Risk, Risk criterion
- **Output:** Test result

### Step 7: Test Incidents Reporting


Security testing results are analyzed and validated according to system security test requirements. The evaluated security performance of targeting system provides data and input required for further development and measurements. If test execution is iterative, the test incidents reporting may also be iteratively performed.

- **Input:** Test result
- **Output:** Test result analysis

## 2. A PROCESS FOR TEST-DRIVEN SECURITY RISK ANALYSIS

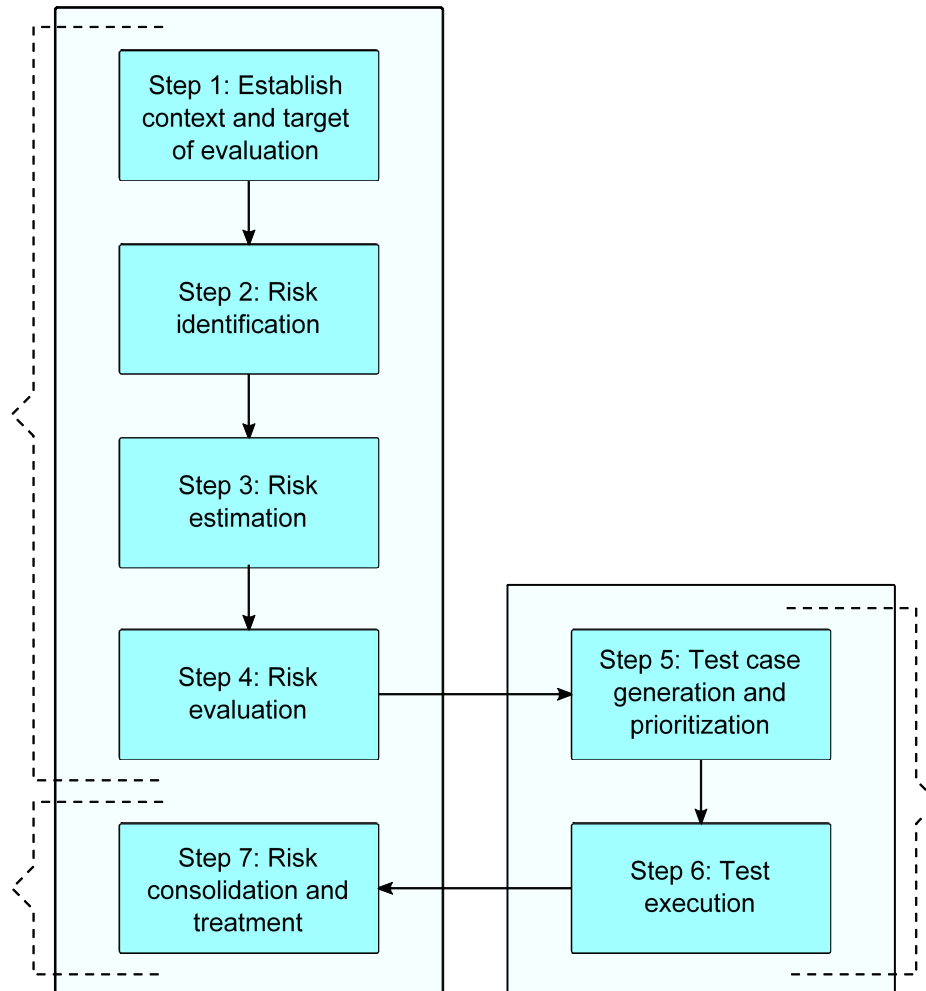
In this section, we describe a process for test-driven security that has been followed in one of the DIAMONDS case studies (the Norse Solutions case study).

Our TSR process is divided into three phases. The goal of **Phase 1** is first to establish the context and target of evaluation, and then conduct a security risk assessment of the target of evaluation. This includes defining the scope of the assessment, identifying security risks w.r.t. the target of evaluation, estimating and evaluating the security risks based on likelihood and consequence values. Having discovered security risks in Phase 1, the analysis proceeds to **Phase 2** in which security tests are identified and executed in order to explore the security risks. Finally, **Phase 3** completes the analysis by validating and updating the risk models based on the security testing results obtained in Phase 2. Additionally, treatments are suggested in order to mitigate the vulnerabilities identified during Phase 2. The phases are further decomposed into the following seven consecutive steps:

	<p><b>Review of security testing tools</b></p> <p>Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 23 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

- **Phase 1** Establish context and target of evaluation, and carry out security risk assessment of the target of evaluation.
  - **Step 1** Establish context and target of evaluation.
  - **Step 2** Risk identification.
  - **Step 3** Risk estimation.
  - **Step 4** Risk evaluation.
- **Phase 2** Generate and execute security tests that explore the risks identified during the security risk assessment.
  - Step 5 Test case generation and prioritization.
  - Step 6 Test execution.
- **Phase 3** Validate and update the risk model based on the security test results.
  - Step 7 Risk consolidation and treatment.

As indicated by Figure 19, the TSR approach is two-folded in the sense that it addresses both security risk analysis and security testing. Security risk analysis was conducted using the CORAS approach [10]. CORAS consists of a language, a tool and a method which are collectively referred to as the CORAS approach. The CORAS language is a customized diagrammatic language for risk modelling. The CORAS tool is a graphical editor for making CORAS diagrams using the CORAS language. The CORAS method is an eight-step method for asset-driven defensive risk analysis in which the tool-supported risk modelling language is tightly interwoven. It is beyond the scope of this section to give a full description of the CORAS approach. We therefore refer to [10] for a detailed and stepwise explanation. Security testing was carried out in a structured manner by (1) identifying and prioritizing *testable* threat scenarios from the total list of threat scenarios identified during Phase 1, (2) identifying security test cases that address the prioritized testable threat scenarios, and (3) executing the identified security test cases. Sections 2.1-2.6 gives an explanation of the six consecutive steps in the TSR approach.




**Figure 13 Overview of the steps in the process.**

## 2.1 STEP 1: ESTABLISH CONTEXT AND TARGET OF EVALUATION

Step 1 was carried out by performing the first four steps in the CORAS method:

- **CORAS Step 1**, preparation for the analysis, aims to make the necessary preparations for the actual analysis tasks based on a basic understanding of the target.
- **CORAS Step 2**, customer presentation of the target, aims to get the representatives of the customer to present their overall goals of the analysis, the target they wish to have analyzed, and the focus and scope of the analysis.
- **CORAS Step 3**, refining the target description using asset diagrams, aims to ensure a common understanding of the target of analysis by having the analysis team present their understanding of the target, including its focus, scope and main assets.
- **CORAS Step 4**, approval of target description, aims to ensure that the background documentation for the rest of the analysis, including the target, focus and scope is correct and complete as seen by the customer.



	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 25 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

## 2.2 STEP 2: RISK IDENTIFICATION

Step 2 was carried out by performing the fifth step in the CORAS method:

- **CORAS Step 5**, risk identification using threat diagrams, aims to systematically identify threats, unwanted incidents, threat scenarios and vulnerabilities with respect to the identified assets.

## 2.3 STEP 3: RISK ESTIMATION

Step 3 was carried out by performing the sixth and seventh step of the CORAS method:

- **CORAS Step 6**, risk estimation using threat diagrams, aims to determine the risk level of the risks that are represented by the identified unwanted incidents (discovered in CORAS step 5).
- **CORAS Step 7**, risk evaluation using risk diagrams, aims to clarify which of the identified risks are acceptable, and which of the risks must be further evaluated for possible treatment.

## 2.4 STEP 4: TEST IDENTIFICATION

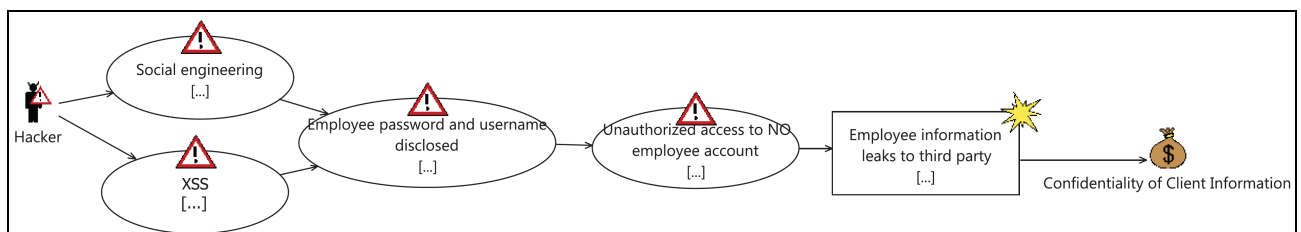
The goal of Step 4 is to identify security tests based on the risk analysis results obtained in Step 3. This was carried out by first *identifying threat scenarios that are testable*, and thereby *identifying security tests for the testable threat scenarios*:

### 2.4.1 Identifying testable threat scenarios

The threat scenarios that lead up to a specific risk were traced back from the risk until the threat source was reached (i.e., accidental human threat, deliberate human threat or non-human threat). For each threat scenario that was encountered in each path, it was evaluated whether the threat scenario was testable or not. By *testable* threat scenarios, we mean threat scenarios that are testable at the implementation level. Furthermore, this process of identifying testable threat scenarios was strictly constrained by and limited to the underlying risk analysis results. I.e., it was not explored for other potential threat scenarios.

The following is an example for how this was done. Figure 14 shows a portion of a threat diagram for Norse Options which has the following elements:

- A deliberate human threat: Hacker.
- Threat scenario 1: Social engineering.
- Threat scenario 2: Cross site scripting (XSS) attack.
- Threat scenario 3: Employee password and username disclosed.
- Threat scenario 4: Unauthorized access to Norse Options employee account.
- An unwanted incident (the risk): Employee information leaks to third party.
- An asset: Confidentiality of client information.



**Figure 14 of a threat diagram for Norse Options (without likelihood and consequence values).**

By starting with the risk and following the path to the hacker we see two different trace-paths:

- Trace-path 1: The risk → Threat scenario 4 → Threat scenario 3 → Threat scenario 1 → Hacker.
- Trace-path 2: The risk → Threat scenario 4 → Threat scenario 3 → Threat scenario 2 → Hacker.

By considering the threat diagram given in Figure 14, the following can be deduced:

- Threat scenario 4: This threat scenario is quite general in sense that there are many ways to realize it. It is therefore not directly testable. We continue further in the trace-path.
- Threat scenario 3: One way of realizing threat scenario 4 is if an employee's password and username is disclosed by a hacker. Again, there are many ways to realize this threat scenario and it is therefore not directly testable. We therefore continue further in the trace-path.
- Threat scenario 1: One way of getting an employee's login credentials is by social engineering. This is not testable because of the threat scenario's nature. It is a threat scenario that is not related directly to the software, but rather a threat scenario that addresses potential vulnerabilities at the organizational level (policies for handling sensitive information, the employees' security knowledge, etc.).
- Threat scenario 2: Another way of getting an employee's login credentials is by performing a successful cross site scripting attack. This threat scenario is testable by performing XSS tests.

From the example above, we have identified threat scenario 2 (XSS) as a testable threat scenario.

## 2.4.2 Identifying security tests

1. All the identified testable threat scenarios were prioritized based on their likelihood values. However, it is emphasized that the likelihood values were given with a certain degree of uncertainty. The degree of uncertainty was not defined in terms of scales (such as the likelihood and consequence scales), but was rather considered on-the-fly in qualitative terms.
2. A scale was defined in order to determine which of the prioritized testable threat scenarios were to be perceived as highest priority. The following is an example. Consider the scale of likelihood values in Table 1:

**Table 1 Likelihood scale**

Likelihood	Description	
Certain	Five times or more per year	[5,infinity>:1y
Likely	Two to five times per year	[2,5>:1y
Possible	Once per year	[1,4>:2y
Unlikely	Less than once each two years	[1,5>:10y
Rare	Less than once per ten years	[0,1>:10y

Based on the likelihood values in Table 1, one can for example define that every testable threat scenario that has a likelihood value in the range of *possible* to *certain* must be perceived as highest priority. Using this perception, one can elicit the highest priority testable threat scenarios for further exploration.


3. Based on the highest priority-scale, the testable threat scenarios were elicited.
4. Security tests that address each of the elicited testable threat scenarios were created. This was done by considering the nature of the testable threat scenario and then creating the tests based on information collected from sources like the Open Web Application Security Project<sup>2</sup> and MITRE's list of Common Vulnerabilities and Exposures<sup>3</sup> (CVE). E.g., XSS tests were created for threat scenario 2 in **Error! Reference source not found..**

## 2.5 STEP 5: TEST EXECUTION

The security tests were carried out automatically, semi-automatically and manually. Before explaining how this was done it is necessary to give a short explanation of the tools that were used:

<sup>2</sup> [https://www.owasp.org/index.php/Main\\_Page](https://www.owasp.org/index.php/Main_Page)

<sup>3</sup> <http://cve.mitre.org/>

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 27 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

### 2.5.1 Tools

- **IBM Rational Software Architect:** IBM Rational Software Architect is a modelling and development environment that uses the Unified Modelling Language (UML) for designing architecture for C++ and Java 2 Enterprise Edition (J2EE) applications and web services. Rational Software Architect is built on the Eclipse open-source software framework and includes capabilities focused on architectural code analysis, C++, and model-driven development (MDD) with the UML.
- **Smartesting CertifyIt:** Smartesting CertifyIt is a test design automation tool that creates tests based on system models (i.e., model based testing). It has built-in integrations with Micro Focus, HP and IBM solutions.
- **Selenium:** Selenium is a suite of tools specifically for testing web applications. The ones used in this case were Selenium IDE, Selenium Server and Selenium Client Drivers. Selenium IDE is a Firefox plug-in that does record-and-playback of interactions with the browser. The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium WebDriver scripts. The Selenium Client Driver is necessary in order to create scripts that interact with the Selenium Server or create local Selenium WebDriver scripts (e.g., in order to run the scripts directly from Eclipse).
- **OWASP WebScarab:** WebScarab is a framework for analysing applications that communicate using the HTTP and HTTPS protocols. WebScarab has several modes of operation, implemented by a number of plugins. In its most common usage, WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser. WebScarab is able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have passed through WebScarab.
- **Eclipse:** Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.
- **Wireshark:** A tool for capturing and analyzing network traffic supporting numerous communication protocols.

#### Tool justification

The approach to risk based security testing in the case study presented in this section is model-based. It was therefore necessary to utilize tools that have the ability to model parts of the system under test (SUT) and create the tests from the model. **IBM Rational Software Architect (RSA)** has the ability to model the system under test and **Smartesting CertifyIt** has the ability to generate JUnit interfaces (which can further be implemented as functional tests) based on the model. Neither IBM RSA nor Smartesting CertifyIt is a security specific tool, but the combination of these tools makes it possible to create functional tests based on a model of the SUT. Furthermore, by implementing security specific properties in the tests that are generated by Smartesting CertifyIt (e.g., security specific test-data), it is possible to create some security specific tests. However, it is emphasized that these types of security tests are limited to traditional functional tests in nature – i.e., the only difference between a traditional functional test and a security test that is created with the help of IBM RSA and Smartesting CertifyIt is that security specific properties are added to the functional tests. It is, e.g., not possible to perform security tests that intercept with the http/https requests and responses, which is an essential tool property when performing security tests on web-based applications.

The tests that were generated by Smartesting CertifyIt were automated by using the **Selenium** API in the tests that were implemented. This made it possible to automatically execute the tests via a Web Browser from Eclipse. **Eclipse** was used to implement and execute the tests.

**OWASP WebScarab** has many functions for executing semi-automatic tests. It was used to carry out the tests that could not be fully automated. In particular, it was used to carry out the tests that required interception of the http/https requests and responses.

**Wireshark** was used to execute the tests that required network traffic analysis for other protocols than HTTP and HTTPS. In line with WebScarab, Wireshark made it possible to carry out the security tests that could not be fully automated.

## 2.5.2 Execution

1. Security tests that could be automated were carried out in the following way (see Figure 15):
  - a. A model was created of the part of the system under test that was addressed by a given security test. The model was created using IBM Rational Software Architect.
  - b. JUnit tests were created (in means of Java interfaces) based on the model using Smartesting CertifyIt.
  - c. The tester implemented the security tests using the JUnit test interfaces. The Selenium Web Application Testing System's API was also used in the tests that were implemented. This API was necessary in order to execute the tests automatically via a Web Browser (in this case, Firefox).
  - d. The tests were executed directly from Eclipse. At this stage, the tests were automated and could be run by "one-click" from Eclipse.
2. Security tests that could not be fully automated, but could be executed with the help of tools, were carried out using OWASP WebScarab and Wireshark.
3. Security tests that could neither be automated nor semi-automated were explored and carried out manually using a Web Browser.

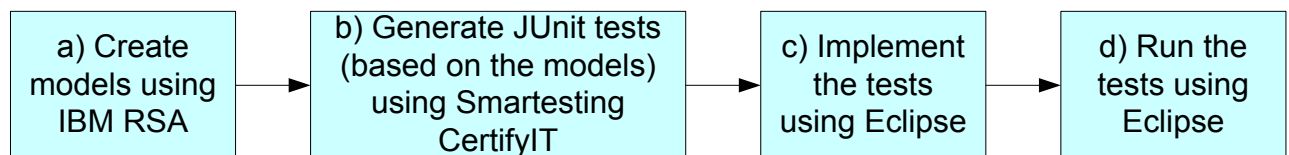


Figure 15 Test automation.

## 2.6 STEP 6: RISK CONSOLIDATION AND TREATMENT

Step 6 was carried out partly in line with the eighth and final step of the CORAS method.


- **CORAS Step 8**, risk treatment using treatment diagrams, aims to identify and analyze possible treatments for the unwanted incidents that have emerged. Treatments are assessed with respect to their cost-benefit evaluation, before a final treatment plan is made.

More specifically, the following was carried out:

1. Based on the security test results, the likelihood values of each testable threat scenario were updated. Consequently, this changed the initial risk picture, and thereby the values of the risks.
2. Risks that contributed to the same overall risk were combined.
3. Treatments were identified for each testable threat scenario that had successfully been carried out (i.e., the threat scenarios that were actually realized). The treatments were suggested based on:
  - a. The testers' knowledge for how to mitigate a given vulnerability that was addressed by a given threat scenario.
  - b. Recommendations collected from sources like OWASP and CVE.
4. The treatments were given in form of a discussion in the final meeting (see **Error! Reference source not found.**), and are given in writing in this section.

## 3. AN EVALUATION OF A PROCESS FOR TEST-DRIVEN SECURITY RISK ANALYSIS BASED ON THE NORSE SOLUTIONS CASE STUDY

Security risk assessment (SRA) is a process that is carried out in order to identify and assess security specific risks. Traditional risk assessments often strongly rely on expert judgment for the identification of risks and their causes and the estimation of their likelihood and consequence. The outcome of these kinds of risk assessments are therefore strongly dependent on SRA participant's background, experience, and knowledge, which in turn reflects an uncertainty in the correctness of the SRA results.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 29 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

In order to validate the correctness of the SRA results, the SRA process can be complemented by other ways of gathering information of relevance to the assessment other than relying on expert judgement by the SRA participants. One such approach is to combine risk assessment with security testing following the steps described

- 1) Establish context and target of evaluation, and carry out security risk assessment of the target of evaluation relying mostly on expert judgement from the SRA participants.
- 2) Generate and execute security tests that explore the risks identified during the security risk assessment.
- 3) Validate and update the risk model based on the security test results.

We refer to this approach as Test-driven Security Risk Assessments (TSR).

We present an evaluation of a TSR process based on the experiences from case study that was carried out in a period of four months, between March 2011 and July 2011. The target system analyzed is a multilingual Web-based e-business application. The system serves as the backbone for the system owner's business goals and is used by a large number of users every day. The system owner, which is also the client that commissioned the case study, required full confidentiality. The results that are presented in this section are therefore limited to the experiences from applying the TSR approach.

The objective of the evaluation is to assess how useful testing is for gaining confidence in the correctness the risk models produced in the risk assessment (phase 1 above). To make the evaluation precise, we have specifically focused the degree to which the testing yielded information that cause us to change the risk model. Our overall hypothesis is that

*The risk model created before testing (in step 1 above) is equal to the risk model after testing (step 3) above.*

If the hypothesis is false, then this means that new information was obtained in the testing step that resulted in the risk model having to be updated/corrected. Our underlying assumption is that this would indicate that process of performing the tests was useful. If the hypothesis is true however, then we cannot, on the basis of this fact alone, conclude that the testing was or was not useful.

Our evaluation suggests that the hypothesis is false. In the case study, the risk model had to be updated after testing. In particular many of the likelihood values of the threat scenarios and risk had to be changed. Moreover, the testing uncovered vulnerabilities that would never have been uncovered in the risk assessment phase (phase 1 above), regardless of how much effort we would have spent in this phase. We therefore believe that the combination of risk assessment and testing is useful.


The rest of the section is structured as follows: Section 3.1 describes the research method of the evaluation and our hypotheses. Section 3.2 gives an overview of the TSR process used in the case study. Section 3.3 describes the case study results. Section 3.4 provides an evaluation of the experiences and results, with respect to the identified hypotheses. Finally, Section 3.5 concludes the evaluation.

### 3.1 RESEARCH METHOD AND HYPOTHESES

The basis of our evaluation is to compare the risk models produced before and after testing. In order to do this, we first make precise what we mean by a risk model, and what we mean by risk models being different.

#### 3.1.1 Risk models

Risk models are created during the risk assessment phase. These models contain the identified risks as well as other information that is relevant for the assessment such as the cause of risks and how likely it is that these causes will occur and so on.

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 30 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

The kind of risk models that were produced in the case study were CORAS risk models. These were constructed on-the-fly during a series of work shops. All the information there was based on expert judgement, mostly from the custom on whose behalf the analysis was conducted.

As illustrated in by the example in Figure 16, a CORAS risk model is a directed acyclic graph where every node is of one of the following kinds:

- **Threat** A potential cause of an unwanted incident.
- **Threat scenario** A chain or series of events that is initiated by a threat and that may lead to an unwanted incident.
- **Unwanted incident** An event that harms or reduces the value of an asset.
- **Asset** Something to which a party assigns value and hence for which the party requires protection.

Note that risks can also be represented in a CORAS risk model, but these correspond to pair of unwanted incidents and assets. If an unwanted incident harms exactly one asset, as is the case in Figure 16, then this unwanted incident will represent a single risk. In the Norse case study, all unwanted incidents that were identified each harmed exactly one asset, thus every unwanted incident corresponded to one risk. Throughout this section, we will therefore use the terms unwanted incident and risk interchangeably.

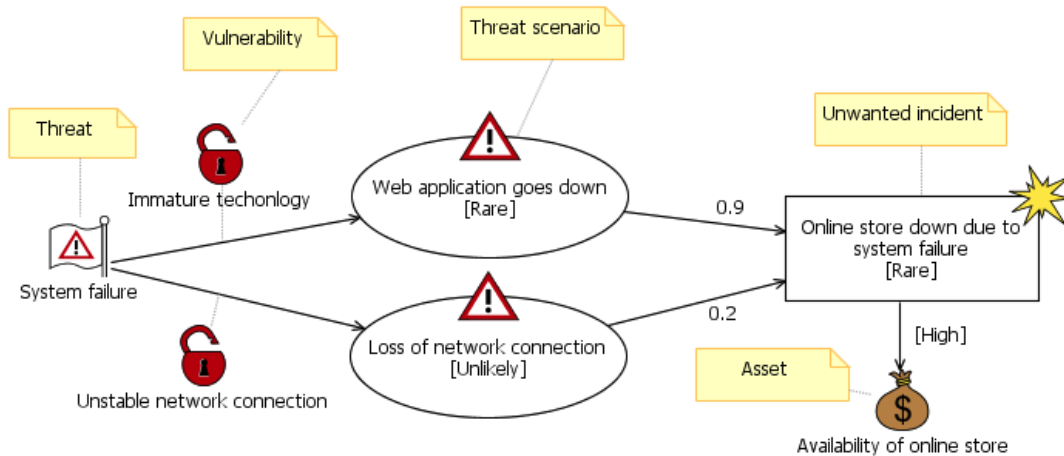
A relation in a CORAS model may be of one of the following kinds:

- **Initiates relation** going from a threat *A* to a threat scenario or unwanted incident *B*, meaning that *A* initiates *B*.
- **Leads to relation** going from a threat scenario or unwanted incident *A* to a threat scenario or unwanted incident *B*, meaning that *A* leads to *B*.
- **Harms relation** going from an unwanted incident *A* to an asset *B*, meaning that *A* harms *B*.

Relations and nodes may have assignments, in particular

- **Likelihood values** may be assigned to a threat scenario and unwanted incident *A*, estimating the likelihood of *A* occurring.
- **Conditional probabilities** may be assigned leads to relations going from *A* to *B*, estimating the probability that *B* occurs given that *A* has occurred.
- **Consequence values** may be assigned to harms relations going from *A* to *B*, estimation the consequence the occurrence of *A* has on *B*.
- **Vulnerabilities** may be assigned to leads to relations going from *A* to *B*, describing a weakness, flaw or deficiency that opens for *A* leading to *B*.





**Figure 16 Example of a CORAS risk model.**

### 3.1.2 Difference between risk models

Two CORAS risk models are equal if they contain the same nodes, relations, and annotations. Otherwise they are not equal. Let *RMB* be the risk model before testing, and *RMA* be the risk model after testing, then we distinguish between 3 different kinds of changes

- **Add** A node or reference in *RMA* has been added if it is not in *RMB*
- **Delete** A node or reference in *RMB* has been deleted if it is not in *RMA*
- **Edit** A node or reference in both *RMB* and *RMA* has been edited if its assignment in *RMB* or *RMA* is different.

### 3.1.3 Hypotheses

Our overall hypothesis is that the risk models before and after testing are equivalent. Having made precise what we mean by risk model difference, we break this hypothesis down accordingly:


- **H1** No risk elements have been added after testing.
- **H2** No risk elements have been deleted after testing.
- **H3** No risk elements have been edited after testing.

## 3.2 OVERVIEW OF PROCESS FOR TEST-DRIVEN SECURITY RISK ANALYSIS

This section presents the process in which the case study was carried out. We give a chronological outline of each meeting that took place in terms of date, participation, preparation, content and the time spent.

In total six meetings took place. The two first meetings primarily focused on motivating the analysis by defining the goals that were to be achieved by the case study, the context and target of analysis, and the focus and scope of analysis. The third meeting focused on concretizing the scope of analysis, the assets that were to be addressed and their belonging risk evaluation criteria. The fourth meeting was dedicated to identify risks and estimate the risk values using the predefined risk evaluation criteria. The fifth meeting focused on identifying security tests. The security tests were identified based on the risk analysis results from the fourth meeting. Finally, in the sixth meeting, the analyst presented the security test results (the security tests were executed between the fifth and the sixth meeting) in addition to the treatments for each security test that had failed. The security tests that had failed led to an update of the initial risk analysis results. The updated risk analysis results were also presented in the sixth meeting.

outlines the process of the analysis. The first column specifies the sequence number for each meeting. The second column shows the dates in which the meetings were carried out. The third column lists the partici-

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 32 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

pants (C denotes participants from the customer organization, while A denotes participants from the analysis team). The fourth column describes the preparations that were carried out prior to each meeting. The fifth column describes the contents and achievements in each meeting. Finally, the sixth column shows the approximate time spent (in man-hours) for each meeting. The letter ``T" in the sixth column denotes the total number of hours spent by all participants.

**Table 2 Overview of the work shops held during the assessment**

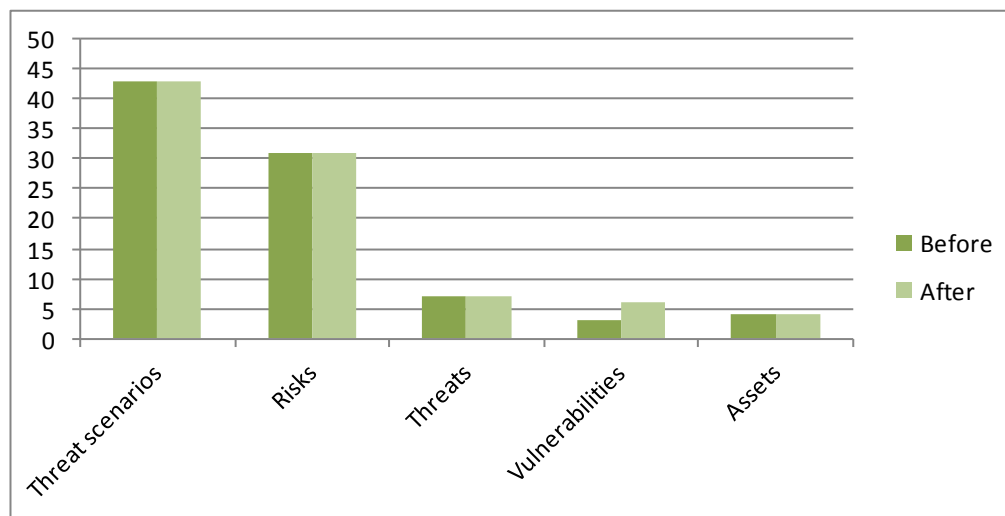
Me etin g	Date	Participants	Preparation	Contents	Hour s
1	28 March 2011	<b>C:</b> One domain expert. <b>A:</b> The analyst. Two domain experts.	The analyst prepared an initial suggestion of the TSR method.	The customer presented the needs and challenges regarding security threats and the assets that needed special attention during the TSR. A brief presentation of the TSR method was given by the analyst, followed by a discussion of how it could be conducted in the case study. The forthcoming meetings were planned. Formalities regarding communication channels and information exchange were clarified.	T:2
2	12 April 2011	<b>C:</b> One manager. One domain expert. One developer. <b>A:</b> The analyst. The secretary. Two domain experts.	The analyst received the input requested: system documentation and user documentation. The documentation was reviewed prior to the meeting.	The customer presented the target and the overall goals they wished to achieve from the TSR. The scope of the target was specified together with the analysis team. The analyst and the secretary gathered information during the meeting for further study.	T:3
3	09 May 2011	<b>C:</b> One domain expert. One developer. <b>A:</b> The analyst. The secretary. One domain expert.	A high-level model of the target of analysis was modelled. A preliminary set of assets and their belonging risk evaluation criteria was documented.	The analyst presented the high-level model of the target of analysis, and the preliminary set of assets and their belonging risk evaluation criteria. The customer provided with corrections and adjustments. The secretary noted the corrections and the adjustments given by the customer.	T:3
4	20 May 2011	<b>C:</b> One domain expert. <b>A:</b> The analyst. The secretary.	The target of evaluation, scope, scales and risk evaluation criteria were updated based on the outcomes of the previous meeting.	The customer approved the focus, scope, scales and risk evaluation criteria. The analyst identified risks and estimated the risk values together with the customer.	T:6
5	27 May 2011	<b>C:</b> One domain expert. <b>A:</b> The analyst. The	A preliminary list of evaluated risks was documented based on the outcomes of the previous meet-	The preliminary list of risks was reviewed and reevaluated (if it was necessary) together with the customer. Security tests were identified based on the evaluated risks.	T:6



		secretary.	ing.		
6	07 July 2011	<b>C:</b> One domain expert. One developer. <b>A:</b> The analyst. The secretary.	The identified security tests were carried out and the results were documented. Treatments were documented for each security test that had failed. The risk analysis results were updated according to the security test results.	The analyst presented the security test results in addition to the treatments for each security test that had failed. The updated risk analysis results were presented and reviewed together with the customer.	T:2

### 3.3 RESULTS

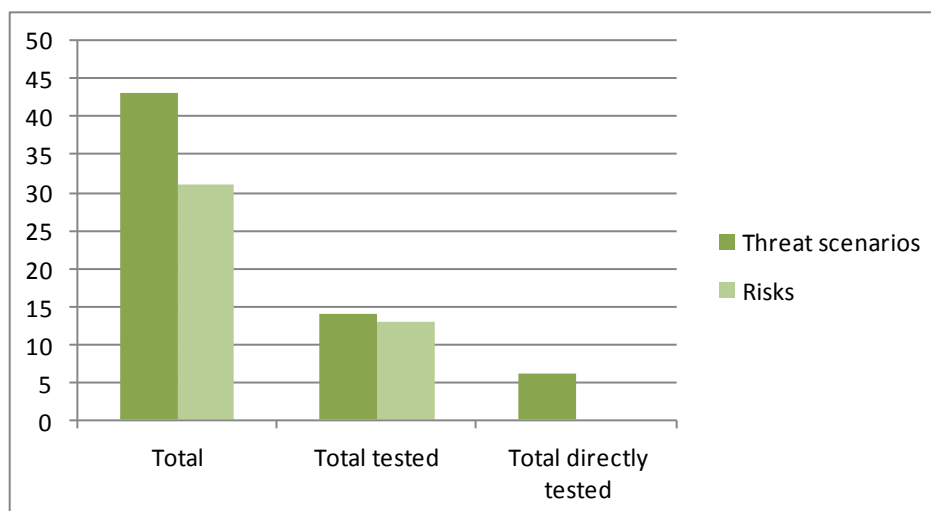
In this section, we describe the differences between the risk models before and after testing.



**Figure 17 Number of risk model elements before and after testing.**

Figure 17 Number of risk model elements before and after testing shows the number of risk model elements in the risk models before and after testing. Only one element was deleted after testing (a vulnerability), hence the figure shows that four new vulnerabilities were added after testing, but no new threats, threat scenarios, unwanted incidents, or assets were added.

Figure 18 shows the number of threat scenarios and risks that were tested. As can be deduced from the figure, 33% of the threat scenarios were tested, and 42% of the risks were tested. Note however, that we have distinguished between those model elements that were directly tested from those that were not. We say that a threat scenario  $T$  was directly tested if  $T$  was used as a basis for deriving tests. A threat scenario or a risk  $TR$  is indirectly tested if there is a threat scenario or a risk leading up to  $TR$  that was directly or indirectly tested. From the figure we can see 14% of the threat scenarios were directly tested, and that none of the risks were directly tested.

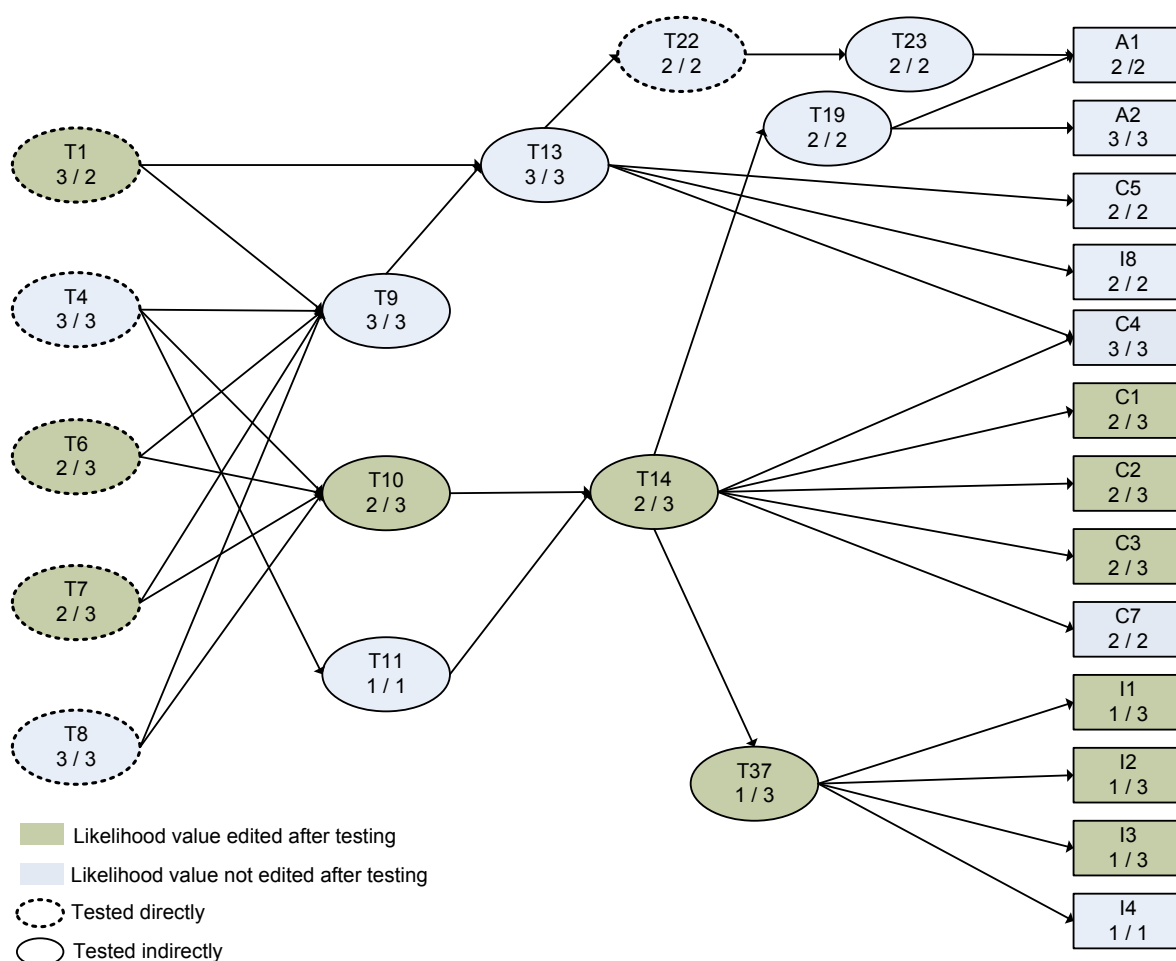


**Figure 18 Number of risks and threat scenarios tested and updated.**

In Figure 19 shows the difference between the threat scenarios and risks that *were tested* before and after testing. In the figure, each threat scenario and risk *TR* has a label of the form *i / j* which means that *TR* had a likelihood value of *i* before testing, and *j* after testing. The likelihood scale that was used in the case study can be mapped to a number between 1 and 5 where 1 represents the most unlikely value and 5 represents the most likely value.

All the threat scenarios and risks whose likelihood values were edited after testing are in the Figure 19 given a darker colour than those threat scenarios and risks that were not edited. Note that all except one risk element whose likelihood values were edited after testing were estimated to be more likely after testing than before testing.

In Figure 19 the threat scenarios that were directly tested are represented by ellipses with a dotted outline; all the other elements of the diagram are indirectly tested. It can be noted that the level of indirection from the directly tested threat scenarios to the risks is quite large.



**Figure 19 Difference between risk models before and after testing.**


## 3.4 DISCUSSION

### 3.4.1 Hypothesis 1

Based on the discussion in Section 3.3 and the numbers in Figure 17, we know that new vulnerabilities were added to the risk model after testing, and that no other kinds of risk elements were added.

Why did the testing only yield new information about the vulnerabilities? The main reason for this is that the tests were designed from the threat scenarios. The threat scenario would typically describe some kind of security attack and the purpose of the tests were to investigate whether the system had some vulnerability that could be exploited by the attack. In other words, the tests were designed to uncover vulnerabilities; not unknown assets, threats, threat scenarios, or risks. These elements were instead part of the context in which the testing was performed.

Recall that an asset is something that is of value for the party, and that can be harmed by a risk. If a party has no assets, then there is no reason to conduct a risk assessment. For this reason, assets are always identified in the beginning of the risk assessment, before the risks are identified. In our experience, the process of identifying the risks has never led to the identification of new assets because the assets are then part of the context of the risk identification. The same is also true for the testing.

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 36 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

The argument is similar regarding threat. A threat is a potential cause of an unwanted incident such as a hacker, an insider or a virus, and the testing is performed *with regards to* the identified threats. It therefore seems unlikely that the testing would uncover additional threats.

In principle, we cannot rule out that it would be possible that the test results could yield information that would lead to the identification of new threat scenarios or risks. For instance, it might be the case that a threat scenario may be refined (i.e. split up into more than one threat scenarios) after testing, or lead to the identification of an unwanted incident that had not been previously thought of. However, as long as the tests are designed to uncover vulnerabilities, we believe that this would be unlikely.

Would it make sense to design tests that would uncover new risks or threat scenarios instead of vulnerabilities? For automated or semi-automated security testing, we do not believe that it would. Since threat scenarios correspond attacks, this would imply performing security testing to uncover new attacks that were previously unknown. This process be possible in theory, but in practise we believe that it would be hard to automate.

In summary, hypothesis H1 is false, since new vulnerabilities were added to the risk model after testing. However, no other risk element kinds were added after testing, and we believe that this will be the case for most assessments following our approach.

It is worth noting that vulnerabilities uncovered by testing in the case study could never have been uncovered if we had performed a risk assessment alone (without doing the testing), regardless of how much effort we would have spent. This is because the testing uncovered issues which only appeared in extremely specific circumstances which could not have been reproduced without execution the system under analysis.

### 3.4.2 Hypothesis 2

Based on the discussion in Sect. 3.3, we know that the testing resulted in the deletion of exactly one risk element - a vulnerability. Furthermore, we believe that this result is generalizable. That is, in general, threats, threat scenarios, risks and assets are unlikely to be deleted after testing, whereas vulnerabilities may be deleted.

The reason why we deleted a vulnerability after testing was that the testing provided evidence that a potential vulnerability identified in the risk assessment phase was actually not present in the system. This led us to remove the vulnerability from the risk model. We also believe that in general, testing can result in the deletion of vulnerabilities, since the tests can be designed to check whether a vulnerability is actually present in the system or not.

The reason why threats and assets are unlikely to be deleted after testing is the same as for hypothesis H1. That is, the assets and threats are part of the context in which the testing is performed, and the testing is therefore unlikely to yield information about this context.


As for threat scenarios and unwanted incidents, these are assigned likelihood values. Therefore, there will never be a need to delete these from the risk model after testing. Instead of deleting them from the risk model, we would instead assign to them a low likelihood value.

In summary, hypothesis H2 is strictly speaking false, since one vulnerability had to be deleted.

### 3.4.3 Hypothesis 3

As documented in Figure 19, 11% of the threat scenarios and 13% of the risks were edited after testing. Moreover, only likelihood values were edited after testing.

For all risk elements that were edited (with the exception of one), the likelihood value was increased after testing, i.e. the risk element was believed to be more likely after testing than before testing. The reason for

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 37 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

this was that the testing uncovered vulnerabilities that were previously unknown, and that led to the belief that certain threat scenarios were more likely to occur than believed before testing.

For one of the threat scenarios, the likelihood values were decreased after testing as a result of one vulnerability being deleted.

In general, we believe that testing will uncover information that may cause the likelihood values of threat scenarios and unwanted incidents to be edited after testing.

The testing did not result in the consequence values that unwanted incidents has on assets being edited. The reason for this is that all the tests were designed to uncover information about vulnerabilities that would increase or decrease the likelihood of a successful attack; the consequence of a successful attack was already known in advance. Is this result generalizable? We believe it is. As long as all the risks have been identified in before testing, their consequences can be estimated before testing, and it is unlikely that the testing will uncover information which will cause the consequence values to change.

In summary, we can conclude that hypothesis H3 is false because the likelihood values of several threat scenarios and unwanted incidents had to be edited after testing. Furthermore, we believe that it is unlikely that testing will uncover information that results in the consequence values of risks to be edited.

### 3.5 CONCLUSION

We have described an evaluation of a process for test-driven security risk assessment (TSR) based on our experiences from applying this process in a case study. The objective of the evaluation was to evaluate how useful testing is in gaining confidence in the correctness of the risk models produced in the risk assessment phase of the TSR process. To make the evaluation precise, we analysed the difference between the risk model produced before testing and the risk model produced after testing. Our overall hypothesis was

*The risk model created before testing is equal to the risk model created after testing.*

Based on the results of our evaluation, we conclude that the hypothesis is false. The process of testing yielded information which led to a change in the risk model created before testing. Specifically, four vulnerabilities were added to the risk model, one vulnerability was deleted, and the likelihood values of 10% of the threat scenarios and 19% of the risks were edited after testing.


We believe that the testing was useful in the sense that it yielded a more accurate risk model. But more than this, the testing uncovered vulnerabilities that would never have been uncovered in the risk assessment phase, regardless of how much effort we would have spent. In other words, if the risk assessment phase had been extended with the effort spent on testing, we would not have uncovered the vulnerabilities that were uncovered in the testing phase.

## 4. EXTENSION FOR THE CORAS RISK ANALYSIS METHOD

This chapter proposes a method to increase the efficiency of the risk analysis process and outlines how the results of the CORAS risk analysis can be reused and combined. It introduces a well-defined template library as a starting point for the risk analysis as well as new models, diagram types and procedures as an extension of the CORAS method.

Taking risk analysis artefacts generated for the individual base components as input, probability values for unwanted incidents of complex systems can be calculated if the relations between these artefacts are modelled correctly. This extension is predestined for analysing large scale systems consisting of heterogeneous components, which no single analyst team could handle.

In many applications in varying market sectors, including eCommerce, eGovernment, and eHealth, perfect security cannot be achieved. Trust allows people to use such applications though there are remaining risks. Before taking risks, it is reasonable to carefully analyse the chances, the potential benefits and the potential losses. Those offering security critical applications or services can use risk analyses to treat potential weak-

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 38 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

nesses in their products. Communicating the identified remaining risks honestly can be important to create trust. However, risk analysis might be difficult and expensive. This chapter therefore introduces new concepts to reuse and combine results of the CORAS method for risk analysis as well as make the whole risk analysis process more efficient.

## 4.1 BACKGROUND

### 4.1.1 Information Security Indicators

The Information Security Indicators (ISI, [20]) specification is currently being produced by an ETSI Industry Specification Group<sup>4</sup>. The standard aims to assess the security posture of an organization.

In order to measure the effectiveness of an organization's security policy, the monitoring of nearly 100 operational security indicators of both internal and external origin is proposed. For each indicator, means and tools for the detection of relevant events are given, as well as the respective detectability level ranging from 1 (very difficult) to 3 (relatively easy). As additional information, the state-of-the-art detection ratio is expressed as the monthly frequency of the occurrence of an event or as the percentage against a common basis.

The 94 indicators are grouped into the following four categories and subcategories:

- Indicators with Security Incidents
  - External intrusions and attacks
  - Malfunctions
  - Usurpation of internal rights or identity
  - Other abnormal internal behaviours (general or specific)
  - All incident categories
- Indicators with Vulnerabilities of a Behavioural and Technical Kind
  - Behavioural vulnerabilities
  - Software vulnerabilities
  - Configuration vulnerabilities
  - Global security framework technical vulnerabilities
- Indicators with Vulnerabilities Regarding some Key Processes
- Indicators as Regards Impact Measurement

The ISI indicators were employed for the template library as they provide a catalogue of well-defined vulnerabilities and incidents that can be used as a starting point for risk analysis.

### 4.1.2 Common Criteria for Information Technology Security Evaluation

The Common Criteria for Information Technology Security Evaluation (CC, [21]) are a common framework for the certification of IT-security products. The ISO standard (ISO/IEC 15408) originated out of well-known international standards like Orange Book (1985), ITSEC (1991) and the Canadian Criteria (1993). The current version is Version 3.1, rev. 3 from July 2009.

The professed goal of the CC is to obtain a comparability of evaluation results. This is achieved via the standardised Common Methodology for Information Technology Security Evaluation (ISO/IEC 18045, [22]) which was passed as part of the International Recognition of National Evaluation Results through the Common Criteria Recognition Arrangement (CCRA). The main document as part of a CC-evaluation is the Security Target, containing threats, security objectives and Security Functional Requirements (SFRs). All key activities during an evaluation are based on this document which outlines the scope of the security functionality. The activities most relevant to our proposed approach are threat analysis, functional testing and vulnerability assessment.

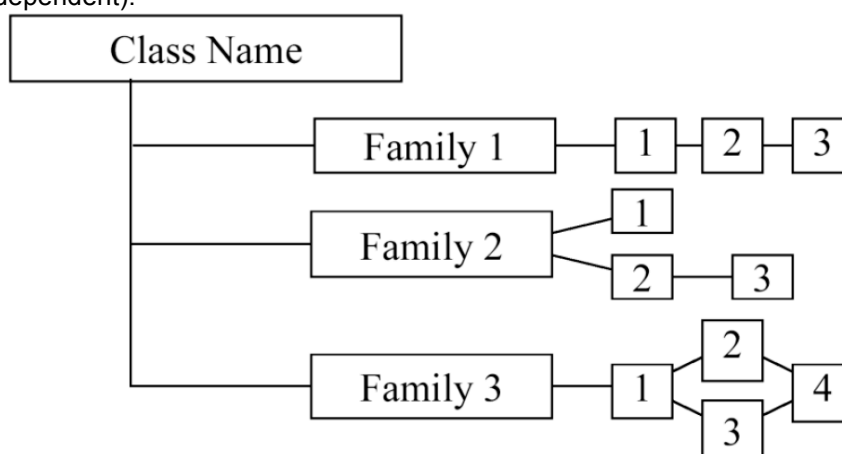
The SFRs are to be selected from a generic list of security requirements as defined in CC Part 2. The SFRs are grouped into eleven classes:

- FAU: Security Audit
- FCO: Communication
- FCS: Cryptographic Support
- FDP: User Data Protection

<sup>4</sup> At the time of writing, the Group Specification still has the status *draft*.

FIA: Identification and Authorization  
 FMT: Security Management  
 FPR: Privacy  
 FPT: Protection of the TSF  
 FRU: Resource Utilization  
 FTA: TOE Access  
 FTP: Trusted Path / Channel

The classes consist of several families which in turn consist of components. Figure 20 outlines this general class-family-component structure and illustrates both dependencies between components (e.g. Component 2 of Family 1 is hierarchical to Component 1) as well as independency (e.g. Component 1 and Component 2 of Family 2 are independent).



**Figure 20 General Class Structure.**

The Security Functional Requirements as defined by the Common Criteria standard were employed for the template library as they provide well-defined functional requirements that can be utilized as treatments during risk analysis.

### 4.1.3 Risk analysis with FTA, FME(C)A and probability theory


Fault tree analysis (FTA) [14] is widely used in the process of risk analysis for critical systems like airplanes or nuclear power plants and hence well-studied [6]. It is a deductive, top-down approach to study how faults can be triggered by sets of other incidents (faults). FTA considering temporal effects is called dynamic FTA. Analysing potential failure paths, FTA makes it possible to determine the probability that a single top level fault occurs. All possible paths have to be taken into consideration by the analysts. Starting at the top level fault, it might be very difficult to recognize all initiating faults that could somehow cause the top level fault. In contrast, failure modes and effects (and criticality) analysis FME(C)A [4] is commonly used as an inductive, bottom-up approach. FME(C)A is better for identifying initial failures than FTA, but not for getting a complete analysis of a complex failure. It can be beneficial to do both, FME(C)A and FTA because they have complementary strengths. In [1], a combination of both is suggested as "Bouncing Failure Analysis (BFA): The Unified FTA-FMEA Methodology".

Based upon [12], for calculating probability values, the following notations and equations/formulas will be used in this chapter: The probability of some incident  $X$  is noted as  $P(X)$ . The conditional probability for incident  $X$  given that it is known that incident  $Y$  occurs is noted as  $P(X | Y)$ . The probability  $P(X \cap Y)$  that both incidents  $X$  and  $Y$  occur can be calculated with  $P(Y)$  and  $P(X | Y)$ :

$$P(X \cap Y) = P(X | Y) * P(Y) \quad (1)$$

The probability  $P(X \cup Y)$  that at least one of two incidents  $X, Y$  occurs is:



	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 40 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

$$P(X \cup Y) = P(X) + P(Y) - P(X \cap Y) \quad (2)$$

If  $X$  and  $Y$  are statistically independent (i.e.  $P(X | Y) = P(X)$  and  $P(Y | X) = P(Y)$ ), then:

$$P(X \cap Y) = P(X) * P(Y) \quad (3)$$

$$P(X \cup Y) = P(X) + P(Y) - P(X) * P(Y) \quad (4)$$

If  $P(X | Y) = 1$  and  $P(Y | X) = 1$ , then:

$$P(X \cap Y) = P(X \cup Y) = P(X) = P(Y) \quad (5)$$

The probability value  $V$  for an incident that has to be triggered by at least *threshold*  $\Psi$  of  $n$  statistically independent incidents each having the probability  $p$  can be calculated using the following binomial formula [13]:

$$V = \sum_{k=\Psi}^n \binom{n}{k} * p^k * (1-p)^{n-k} \quad (6)$$

There are multiple algorithms known for calculations in fault trees – including the binary decision diagram based (BDD) algorithm presented in [11] and DIFtree [7] using BDD and additionally Markov chains. Various software tools support FTA, e.g. Galileo [5].

#### 4.1.4 Risk analysis with the CORAS method

In contrast to the pure failure analytic methods FTA/ FME(C)A, the model based CORAS method [8] supports the entire process of risk analysis “from asset identification to risk treatment” [10].

The CORAS method consists of eight steps. Following this guided step by step procedure, it is possible to identify, analyse and evaluate assets, threats, risks and possible treatments. During that process, different types of diagrams with intuitively understandable graphic symbols are generated as results. CORAS diagrams can be translated to English paragraphs [3]. Besides the completeness, the easy comprehensibility of the CORAS artefacts makes the CORAS method a good choice for analysing the risks of the *S-Network* because communicating the risks is essential for creating trust in the *S-Network*.

In this chapter, the CORAS terminology will be used. Threat diagrams and risk diagrams will be used and extended.


#### 4.1.5 CORAS risk analysis complexity and difficulty

Many computer programs and services are composed of different components – developed, produced and operated by different entities. There is no need to reinvent the wheel or recreate things that already exist. Often, the internals of existing components do not have to be studied in detail to be able to utilize them. It should be enough to look at the public interfaces and their documentation, for example.

Each individual component might eventually have certain risks. For the risk analysis of an entire complex system, to identify the risks inherited from the components it consists of, it would be necessary to get a deep understanding about the internals of these base components. Probably only the producers or operators of each base component will have the required knowledge. Additionally, it would not be efficient to analyse the same base component that is used in many systems over and over again for each system containing that component.

If risk analysis results for individual components were reusable and if they could be composed along with the components to get the risks of complex systems consisting of these components, there would be no need to analyse them again and again. In [2] “Dependent CORAS Diagrams” are suggested to deal with dependencies of different components. But these diagrams are only appropriate to hide some complexity from the “context scenario”. Hence, in [10] chapter 16, “Dependent CORAS” is only mentioned for dealing with assumptions about the environment, which could then be replaced with risk analysis results about the environment. There is not yet a satisfying solution for composing CORAS risk analysis results in not trivial ways.



	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 41 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

## 4.2 A TEMPLATE LIBRARY FOR MODEL-BASED RISK ANALYSIS

In addition to the approach described in the following Chapters 4.3 and 4.4, a generic template library for risk analysis based on CORAS was developed. The goal of this library was to make available to risk analysts a predefined set of CORAS diagrams to facilitate their own risk analysis. Since every system has risks that are inherently system-specific, the template library is not intended to cover every aspect of a systems risk analysis. The predefined template diagrams nevertheless have the claim of playing an important part during risk analysis as they on the one hand contain well-defined risks relevant for most of today's systems and on the other hand propose actions to treat them. These treatments can in turn be associated with Test Patterns, which help increase the level of test automation. A first concept for tracing the CORAS treatments to Test Patterns is described in chapter 4.5. An in-depth description of Test Patterns is given in WP4.1.

The following two sections illustrate how Indicators for Security Incidents (ISIs) and Security Functional Requirements (SFRs) were employed as a starting point for the definition of a generic CORAS-Model to simplify model-based risk analysis (4.2.1) and the exemplary application of the resulting template library to two DIAMOND case studies (4.2.2).

### 4.2.1 Employing ISI and CC for Model-Based Risk Analysis

In order to develop a generic template library applicable to varied domains (automotive, banking, etc.), catalogues of well-defined vulnerabilities, threat scenarios and treatments were needed. For the first two, the Information Security Indicators were chosen, for the latter the Security Functional Requirements as defined in the Common Criteria were selected. Unwanted incidents and assets were deemed too company-dependent to allow any useful suggestions. Following the categorization of indicators proposed by the ISI group specification and adding some of our own, the following seven template diagrams were defined for the template library:

- TD1. External Intrusions and Attacks,
- TD2. Malfunctions,
- TD3. Usurpation of Internal Rights or Identity,
- TD4. Malicious Behaviour of Users and Administrators,
- TD5. Security Critical User Behaviour,
- TD6. Usage of Insecure Protocols/Software and
- TD7. Insufficient Password Policies/Practices.

The definition of the diagrams followed a four-step process:

**Selection:** The vulnerabilities and incidents best suited for risk analysis were selected from the complete list of 92 proposed security indicators. One such indicator is the incident INC21 *Downtime or malfunction of the trace production function* in the category *Malfunctions*.


**Mapping:** Both were then mapped to CORAS vulnerabilities and threat scenarios. As can be seen from , this is not a 1:1 mapping. ISI vulnerabilities were mapped to CORAS vulnerabilities, ISI incidents to CORAS threat scenarios and vice versa. This is due to the definition of both differentiating slightly between the group specification and CORAS and will be discussed in a future meeting of the ISI group.

**Enrichment:** Whenever possible, relevant threat(s) were identified from the indicator description. At times, the description text of incidents also named specific vulnerabilities that were added to the diagram, e.g. *Unlawful Voluntary Stoppage* which could lead to the incident INC21 (see ).

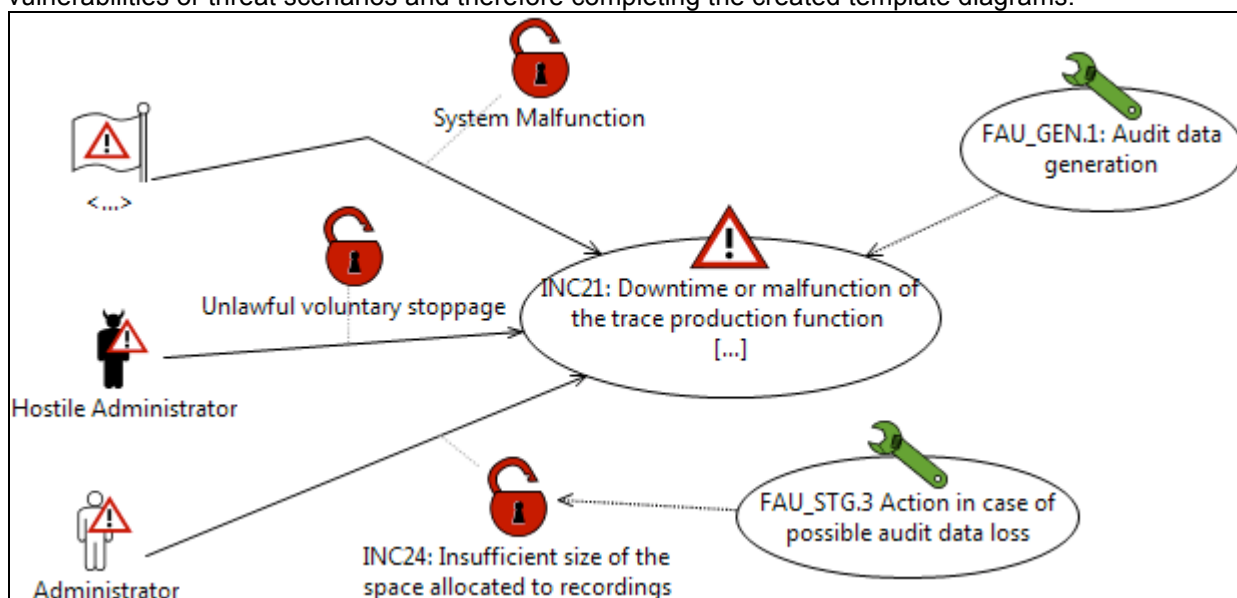
**Completion:** The resulting threat diagrams were enhanced by manually mapping the vulnerabilities and threat scenarios to Security Functional Requirements from the Common Criteria catalogue that were best suited to treat them. One such requirement is FAU\_STG.3 with which one can specify actions to be taken if a threshold on the audit trail is exceeded:

*The TSF<sup>5</sup> shall [assignment: actions to be taken in case of possible audit storage failure] if the audit trail exceeds [assignment: pre-defined limit].*

<sup>5</sup> TSF: TOE security functionality – Combined functionality of all hardware, software, and firmware of a TOE (target of evaluation) that must be relied upon for the correct enforcement of the SFRs.

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 42 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

The result of this process is depicted in in the form of an excerpt from the *Malfunctions* diagram. The three defined threats are *IT-Infrastructure* (non-human threat), *Hostile Administrator* (deliberate threat) and *Administrator* (accidental threat). Together with the three vulnerabilities *System Malfunction*, *Unlawful Voluntary Stoppage* and *Insufficient Size of the Space Allocated to Recordings* they lead to the threat scenario *Downtime or Malfunction of the Trace Production Function*. Up to this point, all information in the diagram were taken from the ISI specification, either in the form of an explicit indicator (INC21, INC24) or derived from their description. Wherever possible, SFRs from the Common Criteria catalogue were assigned as treatments to either vulnerabilities or threat scenarios and therefore completing the created template diagrams.




**Figure 21 Template Diagram based on Indicators and SFRs.**

## 4.2.2 Exemplary Application to two DIAMONDS Case Studies

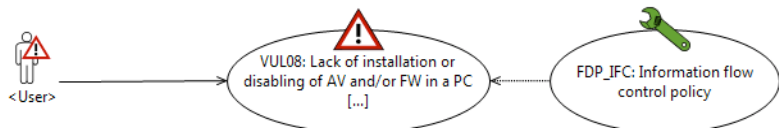
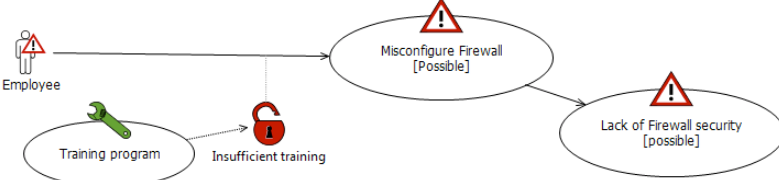
The applicability of the template library was examined in the scope of the two case studies by Giesecke & Devrient anditrust. In order to avoid influencing the preparation of the template library and inadvertently making it applicable only to the systems from the two case studies domains, the performed examination was carried out after completion of the library.

### Giesecke & Devrient (G&D)

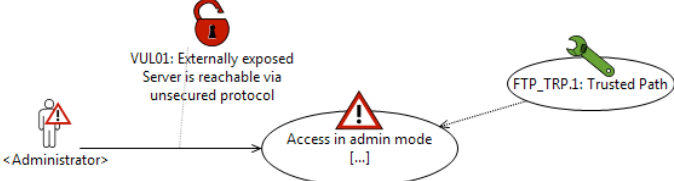
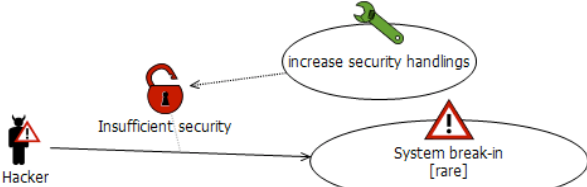
The G&D case study was chosen because of the already available CORAS treatment diagrams. Table 3 to Table 6 illustrate how vulnerabilities, threat scenarios and treatments from the G&D CORAS diagram can be matched to the template library. This supports our argument that employing the library during risk analysis will lead to a swifter yet similar risk assessment basis.


	<p><b>Review of security testing tools</b></p> <p>Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 43 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

**Table 3 TD5. Security Critical User Behaviour**

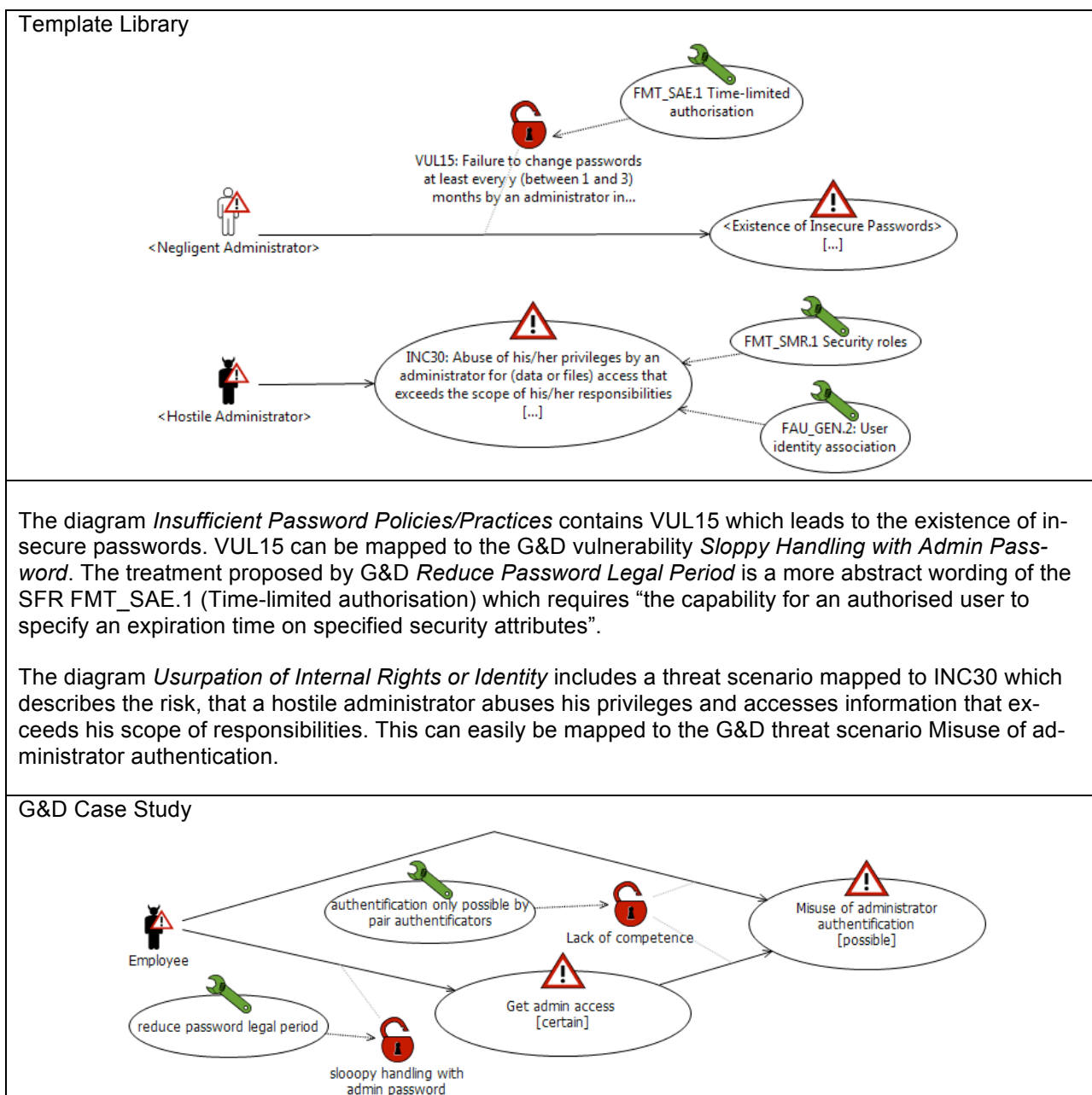
<p>Template Library</p> 
<p>The G&amp;D threat scenario <i>Lack of Firewall Security</i> is simply a more general wording of VUL08 as taken from the ISI group specification. While the training program suggested by G&amp;D treats the vulnerability <i>Misconfigure Firewall</i>, the information flow control policy required by FDP_IFC can aid in defining a global firewall policy that is obligatory for every user.</p>
<p>G&amp;D Case Study</p> 

**Table 4 TD6. Usage of Insecure Protocols/Software**

<p>Template Library</p> 
<p>The vulnerability and threat scenario by G&amp;D comprises the more specific risk of a server that is reachable externally via an unsecured protocol. One way to increase the security handling is to define a locally distinct communication path between the server and external entities as required by FTP_TRP.1.</p>
<p>G&amp;D Case Study</p> 

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 44 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public


**Table 5 TD7. *Insufficient Password Policies/Practices and Usurpation of Internal Rights or Identity***



### itrust

The itrust case study was chosen because it provided two Protection Profiles (PP) that follow the requirements of the Common Criteria standard. While neither PP has been certified at the time of writing, the versions used for our application examination were the final versions scheduled to be send to a certification body for certification.

In the CC, the security problem definition – i.e. the assessment of threats – is axiomatic, meaning that the determination of threat agents, adverse actions and assets falls outside of the scope of the CC. This is the second promising application of the proposed template library: provide a well-defined risk analysis process

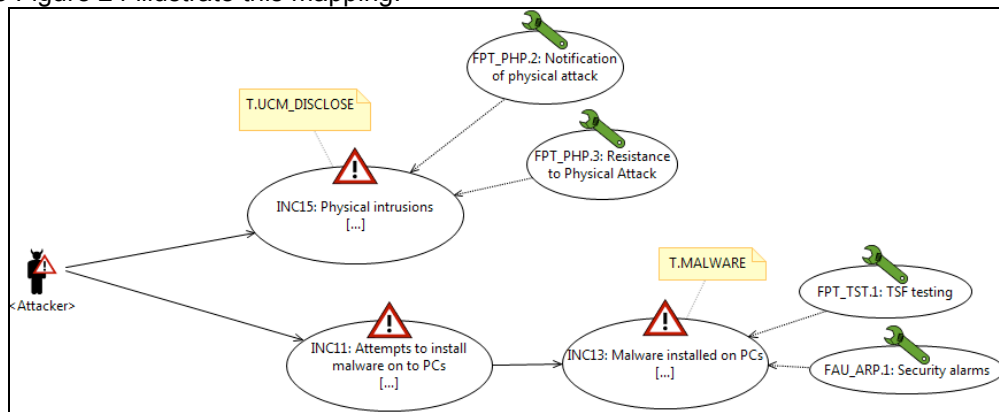
	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 45 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

with automatic derivation of SFRs – a cornerstone of the security assessment as defined by the Common Criteria.

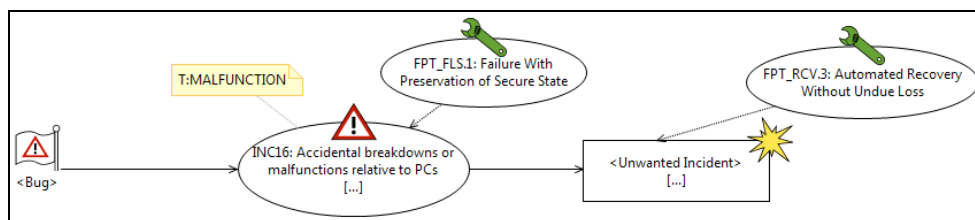
The case study defines the following threats for the target of evaluation:

- T.CRYPTANALYSIS: Attack on the cryptographic algorithms and protocols
- T.INSECURE\_INIT: Insecure initialisation of the TOE (see Figure 24)
- T.MALFUNCTION: Bad operation of TOE component (see Figure 23)
- T.MALWARE: Malicious software on the TOE (see Figure 22)
- T.MAN\_MIDDLE: Man-in-the-middle between User Device and Localisation Assurance Provider
- T.UCM\_ADMIN: Misuse of management
- T.UCM\_DISCLOSE: Physical or Logical Disclosing All or Part of the User Communication Module (see Figure 22)

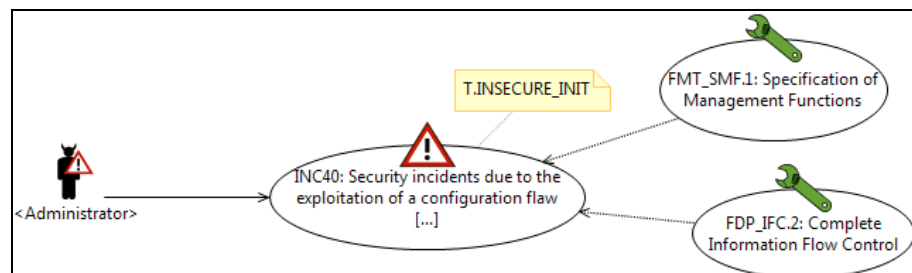
Out of these seven threats, four were easily mapped to threat scenarios contained in the template library. Figure 22 to Figure 24 illustrate this mapping.



**Figure 22 External Intrusions and Attacks.**




**Figure 23 Malfunctions.**



**Figure 24 Security Critical User Behaviour.**

Following the verification of the applicability of the threat scenarios defined in the template library, the suggested treatments were examined. In order to check whether a risk analysis based on the template library would have resulted in the selection of the same SFRs, the tracing from threats to security objectives to SFRs as well as the respective rationales were examined. After thorough investigation, it was concluded that

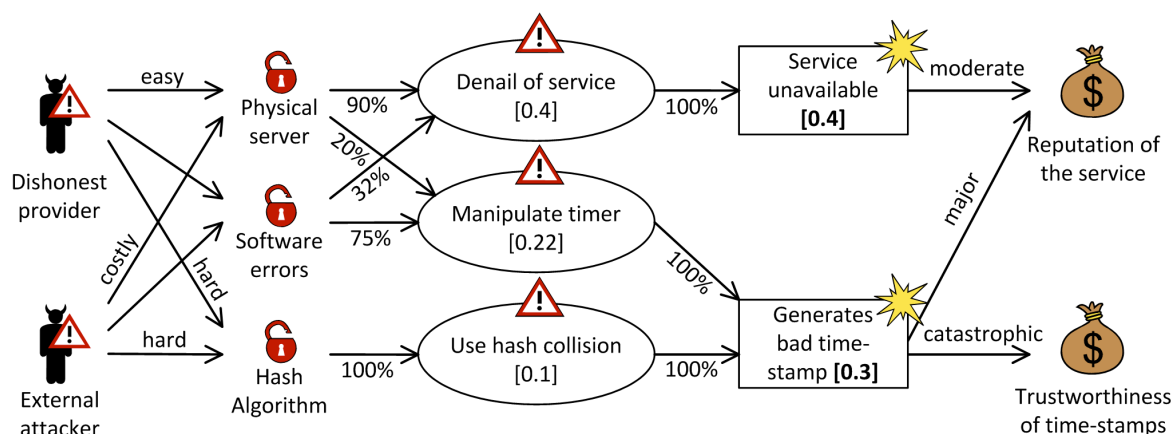
	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 46 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

all treatments from the library were also listed in the PPs and could be traced back to the same threats. Yet, as several SFRs from the PPs were not covered by the template library, the point made earlier in this chapter is emphasised again, namely, that the template library is not a completed risk analysis but merely a starting point for an in-depth analysis of the system-specific risks.

### 4.3 COMPOSITION OF RISK ANALYSIS ARTEFACTS

The idea presented here to make the risk analysis for complex systems more feasible is to use the conventional CORAS method only for the relatively small individual components the system consists of. Composing the resulting artefacts of such analysis along with the combination of the components should allow to detect and to evaluate the risks of the complex system. Combining components, their risks could be reduced; increased or even new risks might arise.

In the scope of this chapter, a service for generating time-stamps is analysed in the scope of this chapter as a compact example. It is a relatively small, but for many applications important and security critical component. The conventional CORAS risk analysis process will not be presented in detail. Instead, just some results are given. The risk analysis artefacts shown here are exemplary excerpts – they are not meant to be complete. Figure 25 shows a *threat diagram* for the exemplary time-stamp service that will be used as the base for all further risk analysis throughout this chapter.



**Figure 25 CORAS threat diagram.**


In step four of the CORAS method the scales for expressing likelihoods, consequences and the functions to calculate risk values are defined by those who do the risk analysis. This freedom makes it eventually difficult to reuse results of the risk analysis for different components if they use different scales and risk functions. Eventually, it might be necessary to define and apply proper conversion functions. In this example, absolute likelihoods of threat scenarios and unwanted incidents to occur within a time period of ten years are noted within square brackets as probability values according to the Kolmogorov axioms [9], i.e. as real numbers between zero (will not happen) and one (will definitely happen). Relative likelihoods on relations are instead noted as percentage values. For example if someone exploits “software errors”, Figure 25 indicates that there is a relative likelihood of 32% that this is a denial of service attack and there is a relative likelihood of 75% that this attack manipulates the timer. Note that attackers can try to do both at the same time, so the sum of these relative likelihoods may be above 100%.

#### 4.3.1 Creating reusable threat interfaces for components

A *threat interface* describes how an individual component could be influenced by the unwanted incidents of other components and how it could itself affect the security of other components or the entire system. It should hide internal details. But it must be detailed enough to model and to evaluate the threats of a complex system composed of multiple components.

A *threat interface* consists of a descriptive name of the component and three lists: The first list contains vulnerabilities that are exposed to other components. The second list contains the unwanted incidents that might be a threat for other components or for the entire system. The third list contains directed relations,



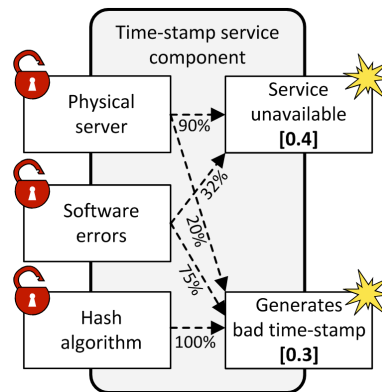
	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 47 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

each having a vulnerability from the first list as starting point and an unwanted incident from the second list as end point.

The threat diagrams created in step five and six of the conventional CORAS method contain all the information required to define a *threat interface*: These diagrams give a very detailed picture by distinguishing between vulnerabilities, threat scenarios and unwanted incidents.

The vulnerabilities and the unwanted incidents from the threat diagram can directly be used within the *threat interface*. The threat scenarios are somehow internal. They are hidden in the *threat interface*: Any relation path in the threat diagram leading from a vulnerability to a threat scenario and further to an unwanted incident is replaced in the *threat interface* with a direct relation between the vulnerability and the unwanted incidents. The relative likelihood values of the replaced relations are multiplied to get the relative likelihood for each new direct relation.

*Threat interfaces* for components have a graphic representation as a box with vulnerabilities on the left hand side and the unwanted incidents on the right hand side. Arrows with dashed lines represent the relations. Relative likelihood values are written under the arrows. The *threat interface* for the time-stamp service is shown in Figure 26.



**Figure 26 Threat interface.**

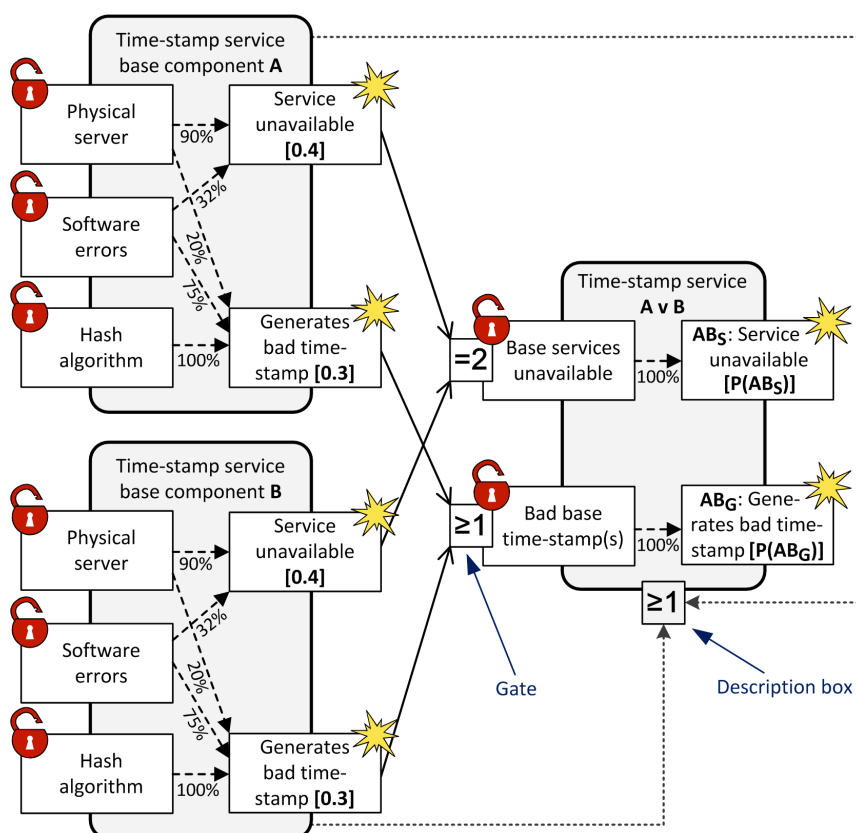
### 4.3.2 Threat composition diagram

In the example, it is more likely that the time-stamp service becomes unavailable, but the threat diagram (Figure 25) shows that if the service generates a bad (i.e. wrong or weak) time-stamp, the consequences are expected to be more serious. In step seven and eight of the conventional CORAS method, both unwanted incidents are therefore evaluated as high risks and both should be treated.

One possible treatment to improve the availability of time-stamp generation is to use multiple time-stamp services. If there are two alternative time-stamp services  $\{A, B\}$  and if it is enough that just one of them is accessible, then the combined service  $A \vee B$  would still be accessible even if one base service becomes unavailable. A client application can immediately contact one of the two services  $\{A, B\}$ . Hence, the combined service  $A \vee B$  does not have to be implemented. It can be just a logical service. Instead of doing a complete conventional CORAS risk analysis for the logical service  $A \vee B$ , the idea is to make a *threat composition* with the *threat interfaces* for the base services. Therefore, the *threat composition diagram* is introduced:

The *threat composition diagram* consists of two layers. The *component layer* contains information about how the base components themselves are combined to a complex component. The second layer contains information about the vulnerabilities and unwanted incidents identified for each individual component and about how these could affect one another. That layer is called the *directed graph of consequences*.

In a *threat composition diagram*, each individual component is represented by its *threat interface*. The relations between the components are modelled on the *component layer* as relations between the entire *threat interfaces* using arrows with dotted lines. If a simple arrow is not enough to make the relation understandable, *description boxes* may be used to informally explain relations. For the relations between the components in the *threat composition diagram* in Figure 27 there is a *description box* on the side of the *threat interface* for time-stamp service  $A \vee B$  having the value " $\geq 1$ ". This means that the new combined time-stamp service  $A \vee B$  relies on the output of at least one of its base services  $\{A, B\}$ .



**Figure 27 Threat composition diagram with three components.**


For each component, the *threat interface* is generated from a threat diagram produced in a conventional CORAS risk analysis processes. If a component is composed of other base components, that analysis should not go into the details of the base components. Instead, vulnerabilities corresponding to the unwanted incidents of the base components are identified. Numeric values for the probability of unwanted incidents which could be triggered by unwanted incidents of the base components do not have to be estimated in the conventional CORAS analysis process. These values can be calculated using the *directed graph of consequences*.

While the *threat interfaces* themselves become a part of the *component layer* in a *threat composition diagram*, their vulnerabilities and unwanted incidents become nodes in the *directed graph of consequences*. The internal relations of threat interfaces become edges in the *directed graph of consequences*. Additionally, if an unwanted incident of some *threat interface* could affect a vulnerability of another *threat interface*, that relation is modelled as another edge in the *directed graph of consequences*. In the *threat composition diagrams*, the graphic representation for such a *trigger relation* is an arrow having a continuous line.

*Gates* can be used to express complex *trigger relations*: For example, multiple unwanted incidents might be required to actually affect a specific vulnerability. Graphically, a *gate* is represented by a small square with a label representing its function. In the example (Figure 27), the combined time-stamp service  $A \vee B$  becomes only unavailable if service A and service B are unavailable. That is why the *gate* on the side of the “base services unavailable” vulnerability has the label “=2”. But if just one of base service A or base service B produces a bad time-stamp, this will cause the combined service  $A \vee B$  to produce a bad time-stamp, too. Therefore, the *gate* on the side of the “bad base time stamp(s)” vulnerability has the label “ $\geq 1$ ”.

The *directed graph of consequences* with its *gates* is similar to a fault tree and it will allow doing similar calculations of probability values. The unwanted incidents correspond to faults in a fault tree. But in contrast to a fault tree, the *directed graph of consequences* does not have to be a tree. It can have multiple top level incidents, for example. With the vulnerabilities, the *directed graph of consequences* contains significantly more information than a fault tree. Furthermore, it is always integrated in a *threat composition diagram*, which contains information about the components and their combination.



	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 49 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

In particular, vulnerabilities are important for identifying potential statistical dependencies not yet modelled in a *threat composition diagram*. Knowing the statistical dependencies is essential to calculate probability values accurately.

If two unwanted incidents can be triggered by the same unwanted incident (i.e. they have a common trigger unwanted incident in the *directed graph of consequences*) then they are definitely somehow statistically dependent. In the *threat composition diagram* shown in Figure 27, there are no such obvious dependencies. Unfortunately, if there is no common trigger in the *threat composition diagram*, this does not necessarily mean that the unwanted incidents are statistically independent.

For statistical independency, there are no such simple criteria. Eventually, the *threat composition diagram* is not fine grained enough and a more detailed analysis is required. Can there possibly be some common trigger incidents that were not yet modelled? Looking at the incidents or faults without knowing further details about the components, it is impossible to answer that question. Vulnerabilities are exactly the missing information: they tell the analysts directly how a component could be affected. A careful look at the vulnerabilities especially of the base components is crucial. In Figure 27 the two base services have vulnerabilities with identical labels. That is a clear indicator that probably there could be some dependencies.

In that case, additional *threat interfaces* for more base components have to be added and their relations must be modelled. Thereby, the *threat composition diagram* gets finer grained.

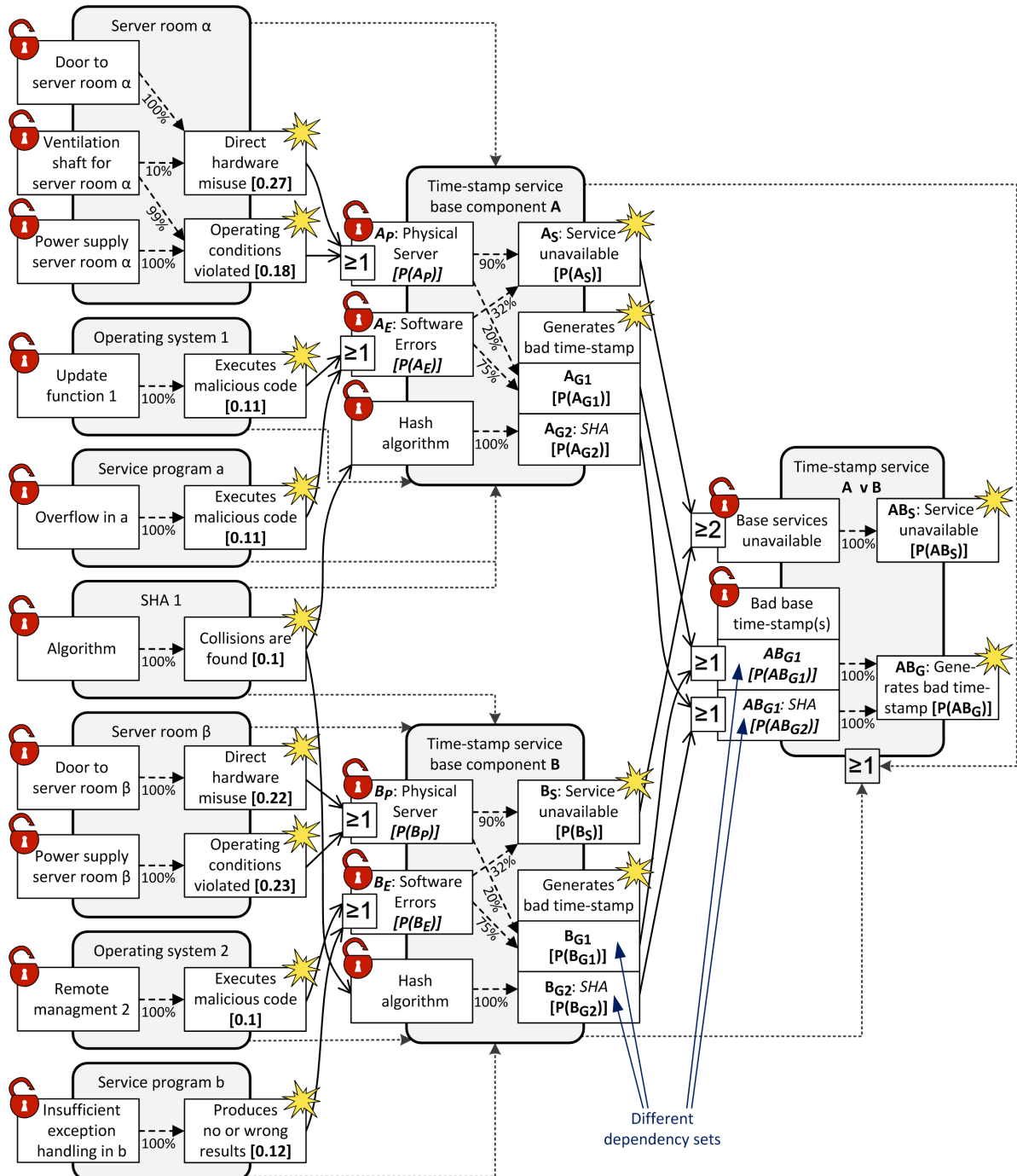



Figure 28 Threat composition diagram with additional base components.

For the time-stamp service base components A and B, there are several base components that will be taken into consideration here: the server room, the operating system, the service program and a hash algorithm. The *threat composition diagram* with the *threat interfaces* for these components is shown in Figure 28.

In the example, both base time-stamp services rely on the SHA-1 algorithm. If that algorithm is broken and collisions can be found easily, both services will produce weak time-stamps. In the *directed graph of consequences* the “generates bad time-stamp” incident of service A and the “generates bad time-stamp” incident of service B have a common trigger node – they can both be triggered by the same unwanted incident “collisions are found” of the “SHA-1” component. Clearly, they are statistically dependent. Note that in a fault tree,

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 51 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

it would be necessary to model two separate nodes having the same name for the two triggers. Else, it would not be a tree. The *directed graph of consequences* can model dependencies more directly and intuitively.

The “generates bad time-stamp” incidents of the base services can also be triggered by other incidents obviously not having the same dependency. The probability values for unwanted incidents caused by triggers of different dependencies must be kept separately to make a correct probability value calculation possible. Therefore, an unwanted incident can have multiple different *dependency sets*, each representing only those incidents caused by triggers that have the same dependency throughout the entire *directed graph of consequences*. In the *threat composition diagram*, the dependency sets are represented as rows in the box representing the incident. *Dependency sets* should have a description indicating the cause of dependency if there is any dependency. In the example, the description for the *dependency set* triggered by the “collisions are found” incident is *SHA*.

Vulnerabilities can be affected by unwanted incidents having multiple different dependencies. In order to support the correct calculation of probability values for the possible consequences, information about different *dependency sets* of the “input” unwanted incidents must be preserved in vulnerabilities. Therefore, the vulnerabilities in the *threat composition diagram* can have multiple rows representing different *dependency sets*, too. Relations between an unwanted incident and a vulnerability (or vice versa) both having the same *dependency sets* are modelled directly between the dependency sets. That way, probability values for different *dependency sets* can be propagated through the *directed graph of consequences* without mixing them up. In the example, the “bad base time-stamp(s)” vulnerability of time-stamp service  $A \vee B$  preserves the *dependency sets* of the incidents which can affect it.

For the *dependency set* named *SHA*, the kind of dependency is visible in the *directed graph of consequences* in the common trigger “collisions are found”. For the other *dependency set*, Figure 28 does not show any common triggers. Does that mean that these parts of the “generates bad time-stamp” incidents of the two base services are statistically independent? The “service unavailable” unwanted incidents of base time-stamp service *A* and of base time-stamp service *B* do not have a common trigger, too. The graph shows no dependency between them. Is the diagram fine grained enough to decide whether they are statistically independent?

The only way to figure that out is to look carefully at the vulnerabilities. For all the vulnerabilities that could eventually have a common trigger, a closer look at a finer grained *threat composition diagram* is required.

One of the vulnerabilities by which both unwanted incidents of time-stamp service *A* could be affected is called “power supply server room  $\alpha$ ”. For time-stamp service *B*, there is a similar vulnerability called “power supply server room  $\beta$ ”. If both server rooms are connected to the same electricity network with the same power plants, then these can definitely be affected by the same unwanted incidents. A closer look with more fine grained components and risk interfaces is required to decide about statistical independency. Figure 29 shows a detailed *threat composition diagram* excerpt just for the two power supplies.

The two incidents *X* (EG1 fails to produce power) and *Y* (EG2 fails to produce power) are statistically dependent. They are an example for mutual dependency, they can affect each other. If both electric generators EG1 and EG2 work within normal parameters, each has to produce 10 KW. If one of these generators fails, then the other generator has to produce up to 20 KW (the maximum capacity). A generator which has to produce 20 KW needs more cooling. It becomes more likely that it will overheat. Though the cooling systems themselves are independent – their vulnerabilities will not be affected by the same incident – the failure of one generator increases the likelihood that the other generator will overheat.

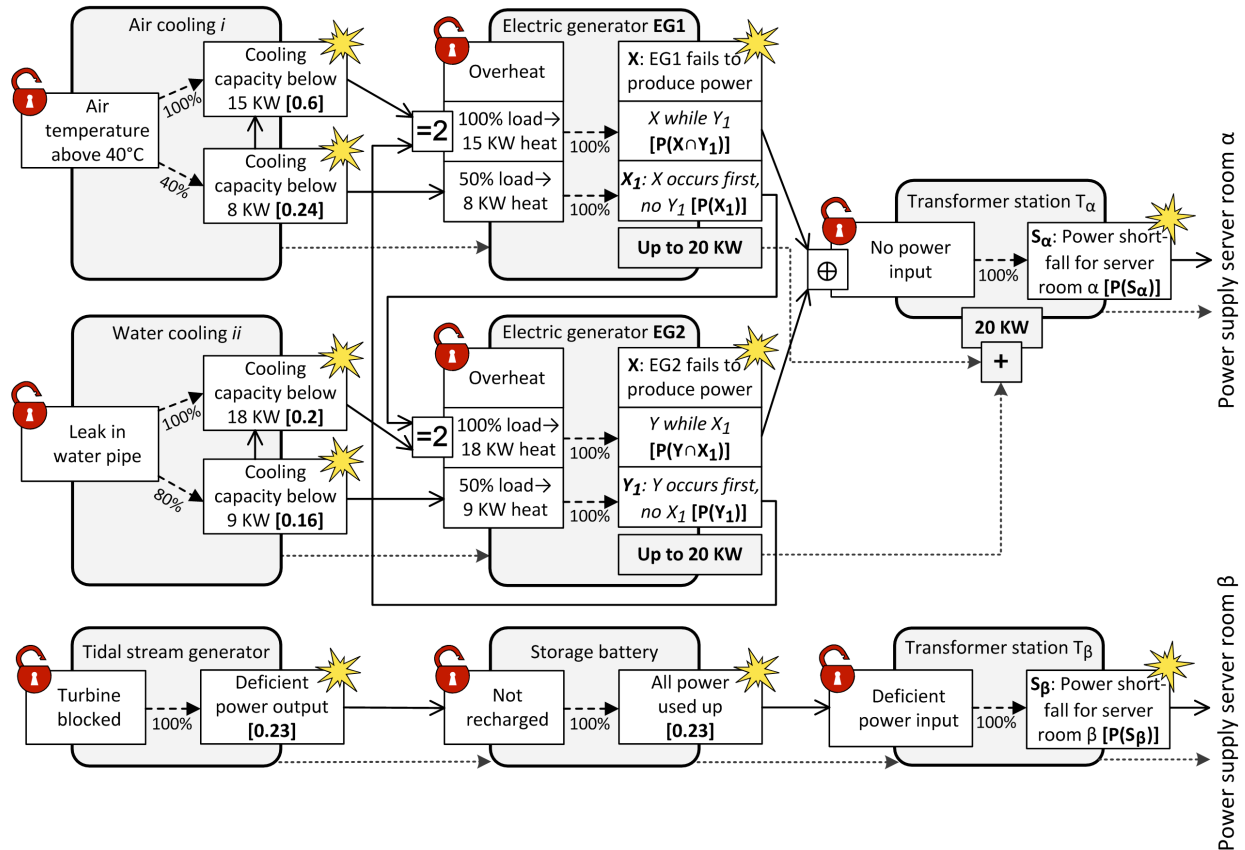


Figure 29 Threat composition diagram for power supply.

As long as EG2 works, EG1 needs to produce only 10 KW and for air cooling *i* a cooling capacity of 8 KW is sufficient. The probability  $P(X_1)$  that EG1 will fail under such conditions is 0.24. The conditional probability  $P(Y | X_1)$  that EG2 will fail if EG1 has already failed is 0.2 because that is the probability that the cooling capacity of “water cooling ii” will drop below 18 KW. Once EG2 has to produce the entire 20 KW, it needs at least that cooling capacity.


With the formula from equation 0, it is possible to calculate the probability that both electric generators will fail if EG1 fails first:  $P(Y \cap X_1) = 0.048$ .

The probability  $P(X \cap Y_1)$  that both electric generators will fail if EG2 fails first can be calculated the same way.  $P(Y_1)$  is 0.16,  $P(X | Y_1)$  is 0.6 and  $P(X \cap Y_1) = P(X | Y_1) * P(Y_1) = 0.096$ .

Only if both generators fail at the same time, there will not be enough power for server room  $\alpha$ . Each of the two incidents  $X \cap Y_1$  and  $Y \cap X_1$  alone can trigger the “power shortfall for server room  $\alpha$ ” incident  $S_\alpha$ . 0 indicates that if  $X_1$  occurs,  $Y_1$  will not occur and vice versa.  $X_1$  and  $Y_1$  are mutually exclusive. Therefore,  $X \cap Y_1$  and  $Y \cap X_1$  are mutually exclusive, too, i.e.  $P((X \cap Y_1) \cap (Y \cap X_1)) = 0$ . The probability  $P(S_\alpha)$  that the power supply for server room  $\alpha$  fails can be calculated using the formula from equation 0:  $P(S_\alpha) = 0.144$ .

This dependency affects transformer station  $T_\alpha$  and server room  $\alpha$ , but neither transformer station  $T_\beta$  nor server room  $\beta$ . The *threat composition diagram* in Figure 29 shows no dependencies between the two transformer stations  $T_\alpha$  and  $T_\beta$ . It is detailed enough to see that the base vulnerabilities do not have any common trigger incidents with a relevant likelihood. The two “power shortfall” incidents are statistically independent. The power supply for server room  $\alpha$  is indeed completely separated from the power supply for server room  $\beta$ . There are no statistical dependencies between the two power supplies that have to be taken into consideration in order to calculate probability values correctly for the *threat composition diagram* given in Figure 28.

Once this top-down analysis is completed and the *threat composition diagram* is detailed enough to decide about statistical independencies, a bottom-up analysis is required to propagate any identified new dependencies with the help of *dependency sets* throughout the entire *directed graph of consequences*. Having finer grained components, analysts doing the bottom-up analysis will eventually identify some vulnerabilities and incidents in higher level components that have been overlooked before. Eventually, further bouncing analysis

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 53 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

going multiple times top-down and bottom-up might be necessary to get a complete picture. Because the *directed graph of consequences* can have multiple top level incidents, it is possible to do this bouncing analysis without changing the model. In a fault tree, this would not be possible.

For the time-stamp service example, in the scope of this chapter, details of the finer grained analysis for other components than the power supplies are omitted. Instead, just the result is given: The fine grained component analysis reveals no additional statistical dependencies between the unwanted incidents of the two base time-stamp services.

Having a *threat composition diagram* with complete information about the dependencies and absolute probability values at least for all initial unwanted incidents, it becomes possible to calculate the missing probability values. For each top level incident, this calculation works like in a fault tree.

Figure 28 shows, that the incident  $AB_S$  (i.e. the combined time-stamp service  $A \vee B$  becomes unavailable) occurs only if both “service unavailable” incidents  $\{A_S, B_S\}$  of the two base services  $\{A, B\}$  occur, i.e.  $P(AB_S) = P(A_S \cap B_S)$ . The analysis shows that  $\{A_S, B_S\}$  are statistically independent. Hence, it is possible to apply the formula from equation 0 and  $P(AB_S)$  is simply  $P(A_S) * P(B_S)$ .

The incident  $A_S$  can be triggered by unwanted incidents that affect the vulnerability physical server ( $A_P$ ) or by unwanted incidents that affect the vulnerability software errors ( $A_E$ ). There is no statistical dependency between any incident affecting  $A_P$  and any incident affecting  $A_E$ . If  $A_P$  is affected by some incident, this will trigger in 90% of all cases  $A_S$ . If  $A_E$  is affected by some incident, this will trigger in 32% of all cases  $A_S$ . Therefore:

$$P(A_S) = 90\% * P(A_P) + 32\% * P(A_E) - 90\% * P(A_P) * 32\% * P(A_E)$$

Both  $P(A_P)$  and  $P(A_E)$  can be calculated using the formula from equation 0 and the absolute probability values of the incidents that affect them as parameters. The results are:  $P(A_P)=0.4$ ,  $P(A_E)=0.21$ ,  $P(A_S)=0.4$ .

$P(B_S)$  can be calculated the same way and has a numeric value of 0.4, too. Finally, it is possible to calculate  $P(AB_S)$ , which is 0.16.

The combined time-stamp service  $A \vee B$  will produce bad time-stamps if at least one of the two base services produces a bad time-stamp. For the unwanted incident  $AB_G$  there are trigger incidents belonging to two different dependency sets.  $AB_{G1}$  represents the vulnerability bad base time-stamp(s) being affected by statistically independent incidents.  $AB_{G2}$  represents the vulnerability bad base time-stamp(s) being affected by incidents depending on the SHA-1 collisions found incident. Any incident that could affect  $AB_{G1}$  is statistical independent from any incident that could affect  $AB_{G2}$ . To calculate  $P(AB_G)$ , it is possible to apply the formula from equation 0 with  $P(AB_{G1})$  and  $P(AB_{G2})$  as parameters. Using the same formula several times and applying the relative likelihoods correctly, it is possible to calculate  $P(AB_{G1})$ .  $P(AB_{G2})$  can be trivially calculated using the formula from equation 0. The numeric values are:  $P(AB_{G1}) = 0.39$ ,  $P(AB_{G2}) = 0.1$ ,  $P(AB_G) = 0.45$ .

The probability that a bad time stamp will be generated has been increased by taking one result of two different base services. To improve both, the availability and the correctness, it would probably be a good idea to use three different base services and to require at least two of them to confirm the same time-stamp value.





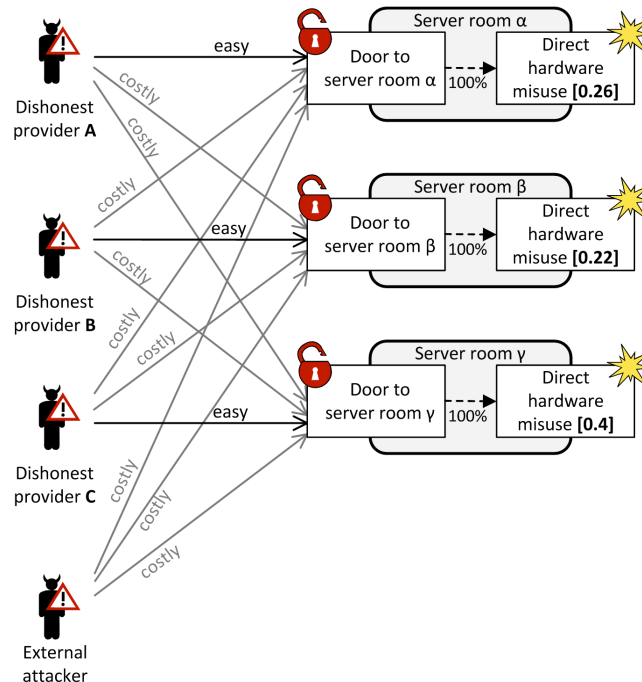
results are:  $P(ABC_S) = 0.352$  and  $P(ABC_G) = 0.212$ . For the logical time-stamp service  $A \wedge (B \vee C) \vee (B \wedge C)$ , both the availability and the correctness are improved.

### 4.3.3 Composition with external threats and assets

Just looking at the *threat interfaces* of the components might eventually not be enough. External threats (especially human threats) identified for different components in separate threat diagrams could probably interact with one another. There could be new combined threats, resulting in different dependencies of unwanted incidents. Hence, the probability values can only be calculated correctly if the potential combinations of threats are modelled and composed correctly.

In the example threat diagram for a single time-stamp service (Figure 25), there is a human threat “dishonest provider”. A single time-stamp service will have a provider who is responsible for the operation of the time-stamp service. The provider owns the server room and he has the key offering easy access to the physical server his service runs on. A dishonest provider could use his privileged access to manipulate the service he is responsible for.


If the time-stamp service is composed of multiple base time-stamp services, no single entity should provide more than one of the base services. Consequently, no single entity would have easy access to more than one of the servers used for a base time-stamp services (Figure 31). Manipulating just one base service would then not be enough to manipulate the logical combined service  $A \wedge (B \vee C) \vee (B \wedge C)$ . The easy access to a single server of a base service becomes less critical.



**Figure 31 Difficulties to physically access the server rooms for the different human threats.**

However, this does not mean that having multiple services with different providers is automatically more secure: Two providers could agree to cooperate with one another to cheat successfully. For those who collaborate, it does not matter that each of them only has the key (and therefore easy access) to exactly one single server room: Two or more allied providers working together do have at least two keys and therefore easy access to at least two different server rooms. Doing manipulations in two different server rooms is enough to manipulate the combined service successfully.

In the *threat composition diagram*, potentially manipulative coalitions can be represented by *threat interfaces* as shown in Figure 32. Each manipulative collaboration incident of these interfaces can trigger direct hardware misuse incidents in two or more different server rooms. Incidents triggered by the same initial manipulative collaboration incident are statistically dependent. The different dependencies have to be modelled as separate dependency sets. In the example, each direct hardware misuse incident has four different depend-

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 56 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

ency sets. These different dependencies have to be propagated forward through the directed graph of consequences as shown in Figure 33.



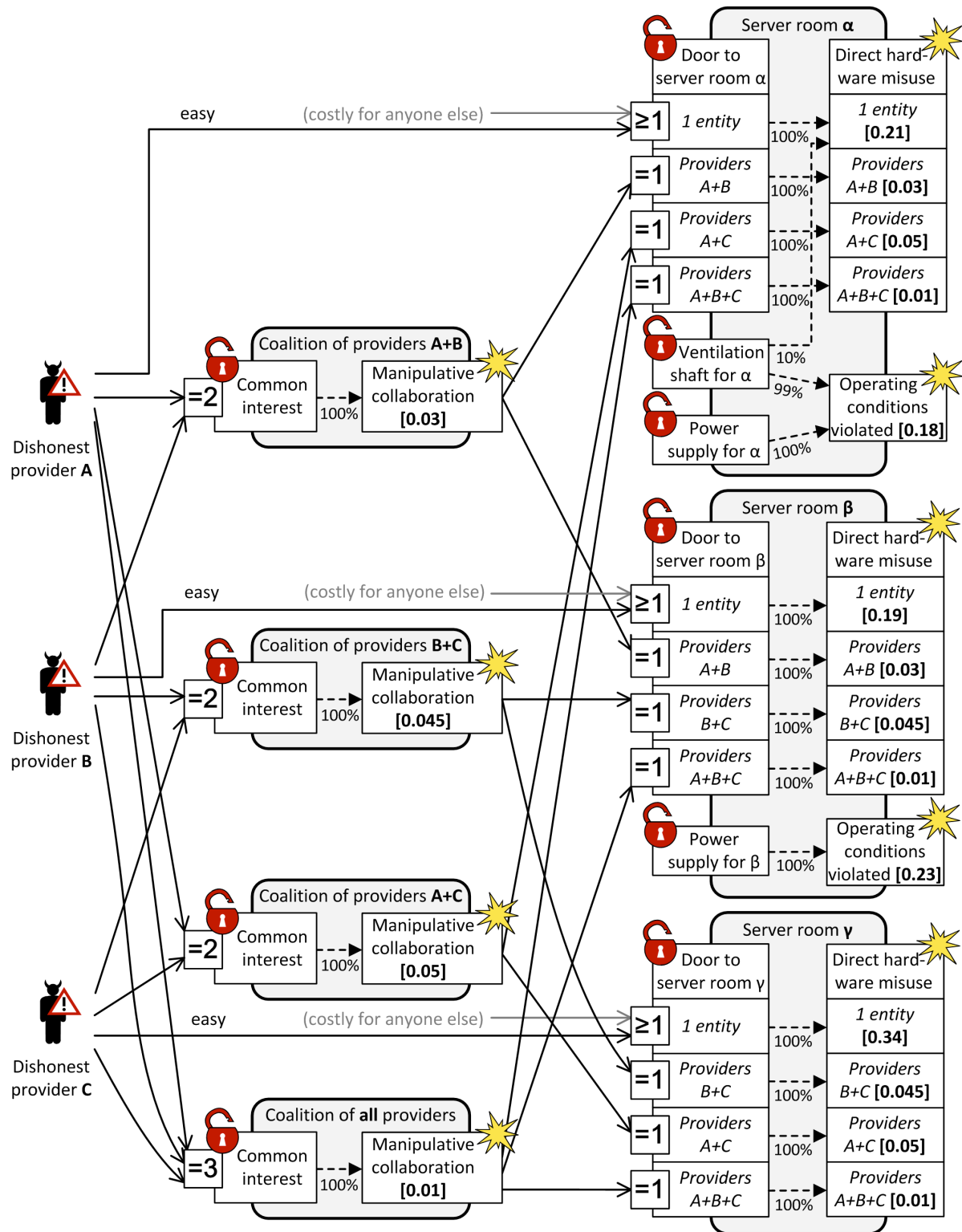
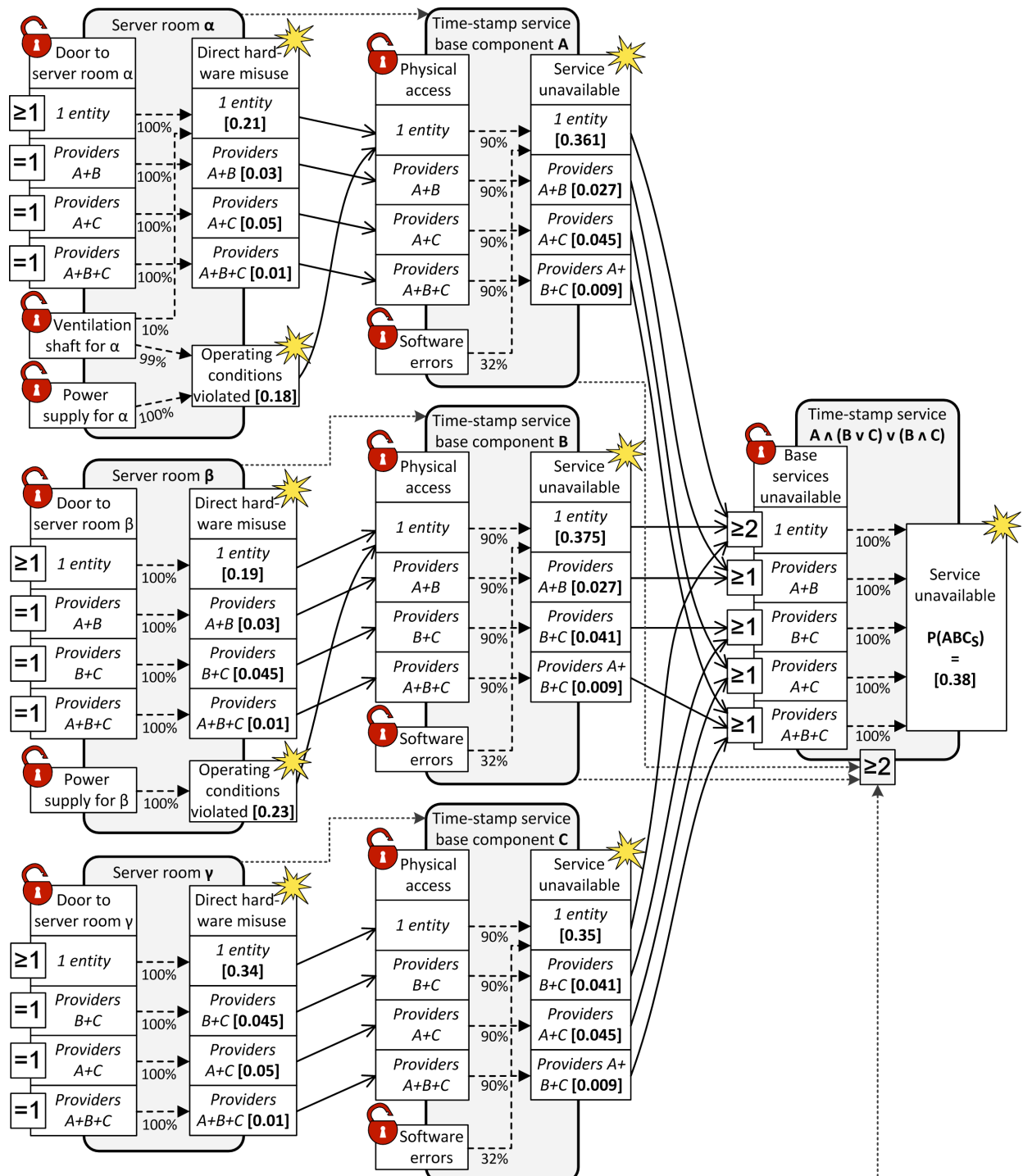



Figure 32.. Threat composition diagram with coalitions (excerpt 1).



**Figure 33 Threat composition diagram with coalitions (excerpt 2, containing only the service unavailable top level incident, with probability value results).**

In general, all threats identified in the threat diagram for any individual involved component have to be added to the *threat composition diagram*. Each threat should appear only once. For each relation from a threat  $K'$  to some vulnerability  $M'$  in the threat diagram of an individual component it is necessary to make sure that there is a relation between the corresponding threat  $K$  and the vulnerability  $M$  in the *threat composition diagram*, too. Eventually, it is not necessary to insert a direct relation: If there is a relation leading from the threat  $K$  to

	<p style="text-align: center;"><b>Review of security testing tools</b></p> <p style="text-align: center;">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 59 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

another vulnerability  $N$ , and if there is a path between  $N$  and  $M$  in the *directed graph of consequences* having a relative likelihood of 100%, then this indirect relation is sufficient.

For the actual composition analysis process of external threats (e.g. the process of finding potentially harmful coalitions of human threats), there is no simple algorithm. Those threats that can affect some of the involved components, but not all of them (at least not in the same way) are candidates for composition analysis. If there can be any interaction between these threats which could affect the new composed system, these interactions and the resulting dependencies must be modelled in the *threat composition diagram*.

The *threat composition diagram* is not yet complete until the consequences of unwanted incidents for the assets are taken into consideration, too. All consequences and assets identified in the threat diagrams for individual components have to be included in the *threat composition diagram*. Let  $T'$  be an asset identified in the threat diagram for the component  $D'$ . If there is not yet an asset  $T$  corresponding to  $T'$  in the *threat composition diagram*, then  $T$  must be added.

For each unwanted incident  $E'$  identified for component  $D'$  that has the consequence  $Q'$  for  $T'$ , it is necessary to make sure that this consequence is also modelled correctly as a consequence relation  $Q$  between  $E$  (i.e. the incident corresponding to  $E'$  in the *threat interface* for  $D'$ ) and  $T$  in the *threat composition diagram*. The consequence value of  $Q'$  is assigned to  $Q$ .

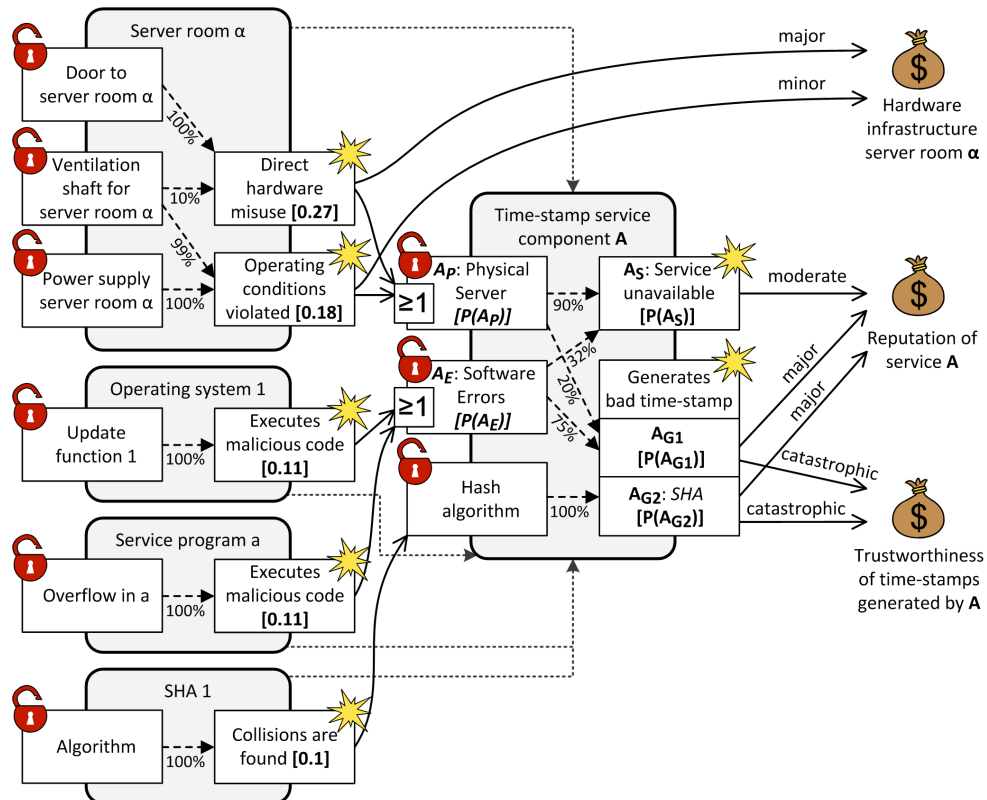
Threats and their influence relations are added to the *threat composition diagram* to support the analysis of dependencies and to enable the correct calculation of probability values. Consequences and assets are basically added to the *threat composition diagram* because these are required for the further steps in the risk analysis process.

#### 4.4 DERIVING AND COMPARING RISKS

For identifying and evaluating risks, it would be possible to define another composition process. But there is no need to do the composition twice. The differentiation between vulnerabilities and unwanted incidents is probably more helpful for the composition than just having risks. For that reason, composition should be done only at threat analysis level.

A *threat composition diagram* with assets and consequence estimations can be used as the base to immediately identify and evaluate the risks without further need for component based composition. Hence, it is possible to create a conventional CORAS risk diagram for the entire system – without worrying about individual components anymore.

Just like in a conventional threat diagram, in a *threat composition diagram* each consequence relation  $Q$  leading from an unwanted incident  $E$  to an asset  $T$  is a risk  $R$ . The probability value of that unwanted incident  $E$  and the consequence value of  $Q$  are the parameters for the risk function, which is used to calculate the risk value for  $R$ . The risk value is necessary for applying the risk evaluation criteria. Risk functions, risk values and evaluation criteria are defined by the risk analysts in step 4 of the conventional CORAS method.



**Figure 34 Threat composition diagram for a single time-stamp service with assets and consequences.**

Typically, risks should be identified at a certain level of abstraction. For example, in the *threat composition diagram* shown in Figure 34, it would be possible to identify the risks at the high level of time-stamp component A or it would be possible to identify them at the low level of the base components. Only the unwanted incidents of the risk interfaces representing the components at the chosen level of abstraction are translated to risks for all possible consequences.

A consequence relation of some unwanted incident  $E$  can be indirect: If there is a path in the *directed graph of consequences* leading from  $E$  to some unwanted incident  $W$  having the consequence  $Q$  for asset  $T$ , then incident  $E$  can have the consequence  $Q$  for  $T$ , too. Hence, a risk can be identified for  $(E, W, Q, T)$  and the risk value can be calculated using the product of the absolute probability value for  $E$  and the relative likelihood for the path between  $E$  and  $W$  (i.e.  $P(E) * P(W|E)$ ) as the first parameter and the consequence value of  $Q$  as the second parameter for the risk function.

For example, the unwanted incident “executes malicious code” of the “service program a” component in the example *threat composition diagram* shown in Figure 34 does not have direct consequences for any identified asset. But there are paths in the *directed graph of consequences* indicating that the incident can indirectly affect assets: There is one moderate consequence that the “executes malicious code” incident will have if it triggers the “service unavailable” incident and there are two consequences (one major, the other catastrophic) that it will have if it triggers the “generates bad time-stamp” incident.

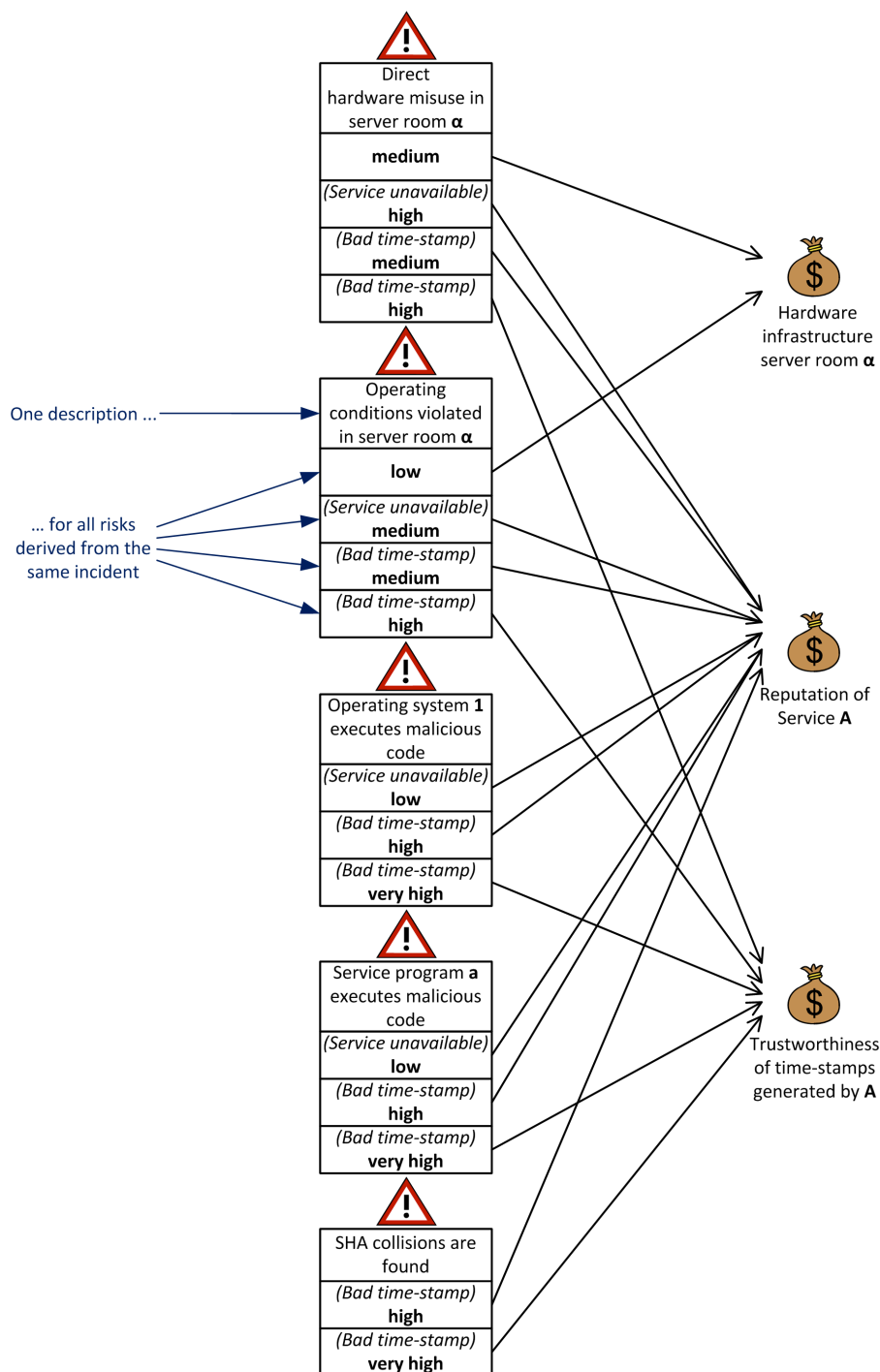
Each of these indirect consequence relations leading from an incident to an asset is identified as an individual risk.

**Table 6 Risk Function for Base Incidents**

		Consequences			
		<i>Minor</i>	<i>moderate</i>	<i>major</i>	<i>catastrophic</i>
Likelihood	$< 0.03$	very low	very low	low	medium
	$[0.03-0.06[$	very low	low	medium	high
	$[0.06-0.16[$	Low	medium	high	very high
	$\geq 0.16$	Medium	high	very high	very high


A common risk function defined for all base components in step 4 of the conventional CORAS risk analysis process is given in Table 6. While the consequence value for a risk can just be read from the graph, the likelihood value for a risk has to be calculated along the path in the directed graph of consequences. The probability that the “executes malicious code” incident of component “service program a” occurs is 0.11, but only 32% of these incidents lead to the “service unavailable” incident. Therefore the probability for the risk “service program a executes malicious code” (“service unavailable”) is 0.0352. Having a “moderate” consequence, this is a “low” risk.

Identifying and determining the risks in that way, it is possible to construct a flat conventional CORAS risk diagram using a *threat composition diagram* as input. It probably makes sense to summarize all the risks derived from the same unwanted incident in a compact structure as shown in Figure 35.



**Figure 35 Risk diagram for a single time-stamp service.**

If only the unwanted incidents of risk interfaces for higher level components should be analysed for identifying risks, caution is required if some base component incidents have consequences for assets that have not been identified for the higher level components. In the example shown in Figure 34, there are no consequences from the unwanted incidents of the “time-stamp service A” component for the “hardware infrastructure” asset. But these unwanted incidents can be triggered by the “direct hardware misuse” incident or by the “operating conditions violated” incident of the *risk interface* for the “server room α” component, which both have consequences for the “hardware infrastructure” asset. If these consequences and assets do not matter

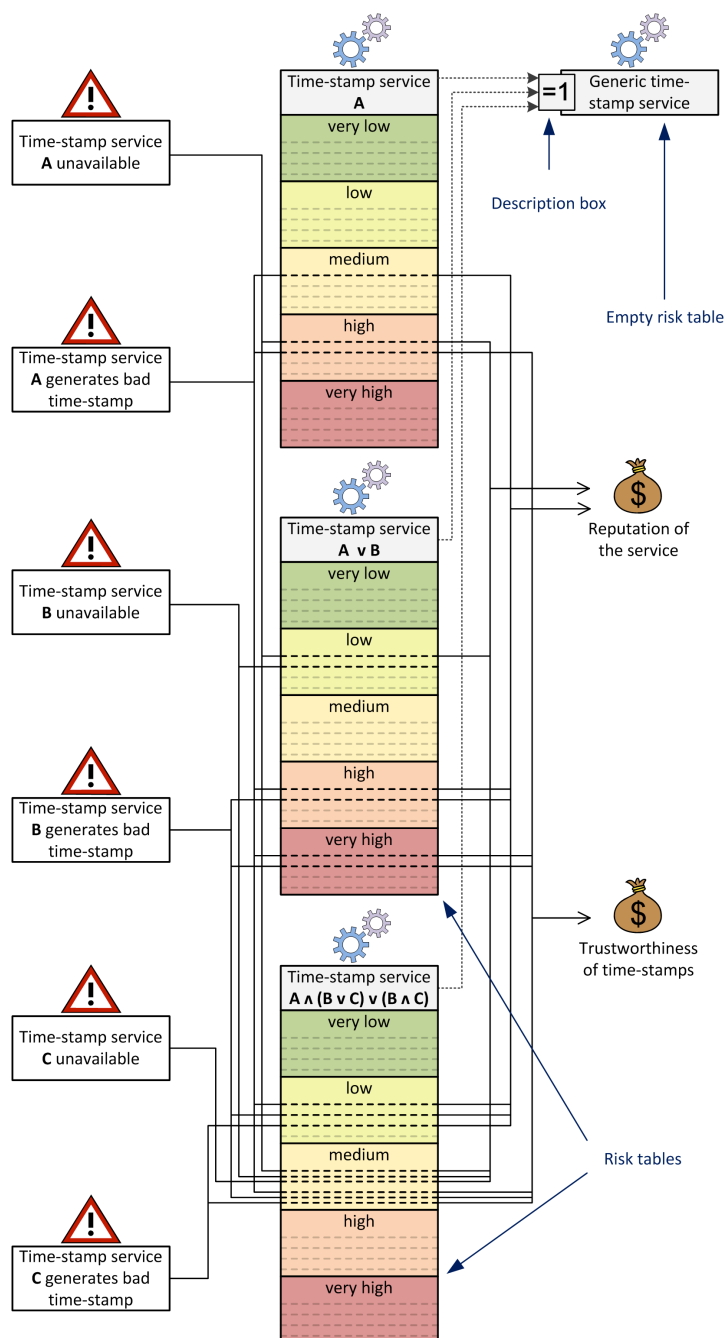
	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 63 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

in the higher level context they may be ignored. Otherwise, the risk analysis for the higher level components was probably not complete and must therefore be repeated taking more assets into consideration.

#### **4.4.1 Comparing the risks of components and architectures**


Though it is possible to get completely rid of all the component and composition information when deriving risks from a *threat composition diagram*, it might also offer some benefits to create a diagram that keeps some information about the components. The idea is to make components or complex combinations of components comparable in terms of risks. Identifying the most critical components allows focusing treatment efforts. Typically, for a complex system, there is not only one single possible configuration. The system could probably be build using another combination of components or using completely other base components, too. It should be possible to choose the architecture with the fewest risks. Therefore, the *risk comparison diagram* is introduced here.





**Figure 36 Risk comparison diagram.**

In a *risk comparison diagram*, each component is modelled as a *risk table*. Each *risk table* has a row for the component name and rows for all the risk values that have been defined during the risk analysis for that component. Relations between the components can be modelled between the *risk tables* with arrows having dashed lines and description boxes. A *risk comparison diagram* contains the risks and assets that can be identified for all involved components. In contrast to a risk diagram, the risk value is not written down for each risk. Instead, the consequence relations from risk *R* to asset *T* are made through the *risk table* representing the component the risk was identified for. More precisely, the relations have to pass through the row representing the risk value of the risk *R*. That way, all risks of a component and their values are summarized in a *risk table* at a glance. Figure 36 shows a *risk comparison diagram* for three alternative time-stamp service designs.

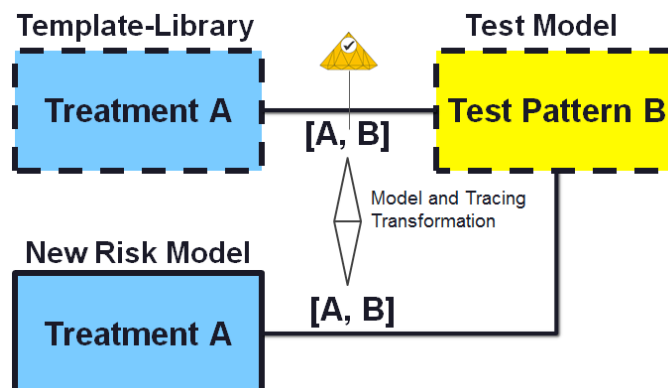
	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 65 of 67
		Version: 1.1
		Date : 02.07.2012
		Status : Final Confid : Public

For a complex system, there are many difficult design decisions, e.g. which technologies and implementations should be used for the individual components to minimize the total risks. *Risk comparison diagrams* have proven to be a valuable tool for making such decisions for the S-Network and they are helpful to communicate such decisions graphically.

#### 4.5 CONCLUSION, RELATED AND FURTHER WORK

Chapter 4 presents a homogeneous approach for risk-driven security testing which introduces multiple advantages for the risk analysis process. With the extension presented in Chapters 4.3 and 4.4, the CORAS method becomes practicable for the risk analysis of large scale systems consisting of many different components. Modelling the relations between risk analyses artefacts generated for individual components, the probability values of unwanted incidents for the complex system can be calculated. In addition, an approach to make the risk analysis process more efficient as a whole is presented in Chapter 4.2. In combination, the template library and composition diagrams allow for a well-structured, efficient risk analysis process that is well suited for large systems normally deemed too complex for risk analysis.


The template library described in this deliverable will deploy its full potential when being integrated with the Test Pattern catalogue described in D3.WP4.T1. Both catalogues can be linked together through the use of SFRs and therefore offer a predefined tracing from the risk analysis phase to the test phase and back. In order to preserve the predefined tracing between the CORAS template library and the test pattern catalogue, a model transformation as shown in Figure 37 will be implemented. The relevant risk paths from the template library can then be selected and transformed into a new, standalone CORAS risk model, while the tracing between treatments and test pattern is preserved.



**Figure 37 Tracing Preservation.**

The *directed graph of consequences* in *threat composition diagrams* is similar to fault trees. It contains gates, which can express relations that conventional CORAS diagrams cannot model. But in contrast to a fault tree, the *directed graph of consequences* does not have to be a tree: there can be multiple top level incidents. A single *directed graph of consequences* can represent multiple fault trees. Nodes in the *directed graph of consequences* modelling incidents can have relations leading to more than a single consequence incident. Therefore, dependencies can be modelled directly as common trigger nodes. In a fault tree, a fault triggering  $n$  other faults must be represented by  $n$  nodes having the same name but no graphical connection – which is less intuitive. Even more important, using the *directed graph of consequences*, bouncing analysis becomes feasible, going top-down and bottom-up with the same model. Using FTA, bouncing analysis is only possible in combination with other risk analysis methods like FMCA, which work on other models than fault trees. Transitions between different methods can cause problems and might be too difficult.

Containing the vulnerabilities, the external threats, the consequences and the assets, the *directed graph of consequences* offers the analyst more useful information than a fault tree. As a part of the *threat composition diagram*, the *directed graph of consequences* is always integrated in a model for the combination and interaction of the components themselves – represented by their *threat interfaces*. Having such a complete picture can help the analyst to identify all relevant risks. However, diagrams can also get large and complex. High level CORAS is suggested to hide details in conventional CORAS diagrams [10]. A similar approach could be used for hiding parts in the new diagram types suggested here. A Software tool like the CORAS

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 66 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public


tool for the conventional CORAS method ([http://coras.sourceforge.net/coras\\_tool.html](http://coras.sourceforge.net/coras_tool.html)) can help the analysts to deal with complex diagrams. Despite modelling and visualizing, a software tool could also support the computation of probability values in the *directed graph of consequences*.

Conventional risk diagrams can be created directly from the *threat composition diagram*. The least risky components and designs can be chosen using the *risk comparison diagram*.

Further research could try to add some information about the life cycle of unwanted incidents to the extended CORAS method. For how long does an unwanted incident last? Is the unwanted incident detected? Will the unwanted incident be repaired within a certain time-period once it was detected? Such information is essential to calculate more precise probability values. Established in other analysis methods like dynamic FTA, these aspects should be captured by CORAS, too.

## 5. REFERENCES

- [1] Zigmund Bluvband, Rafi Polak, Pavel Grabov: Bouncing Failure Analysis (BFA): The Unified FTA-FMEA Methodology, ALD Reliability Engineering, Tel-Aviv 2005, <http://www.aldservice.com/en/articles/bouncing-failure-analysis-bfa-the-unified-fta-fmea-method.html> (2012-04-15)
- [2] Gyrð Brændeland, Heidi E. I. Dahl, Iselin Engan, Ketil Stølen: Using dependent CORAs diagrams to analyse mutual dependency, Lecture Notes in Computer Science 5141, Second International Workshop on Critical Information Infrastructures Security (CRITIS'07) pp. 135-148, Springer 2008
- [3] Heidi E. I. Dahl, Ida Hogganvik, Ketil Stølen: Structured semantics for the CORAS security risk modelling language, 2nd International Workshop on Interoperability solutions on Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ'07). Report B-2007-3 pp. 72-92, University of Helsinki 2007
- [4] Department of Defense: Proceedings for Performing a Failure Mode, Effects and Criticality Analysis, MIL-STD-1629, Washington 1949/1980, <http://www.fmea-fmeca.com/milstd1629.pdf> (2012-04-15)
- [5] Joanne Bechta Dugan, Kevin J. Sullivan, David Coppit: Developing a low-cost high-quality software tool for dynamic fault-tree analysis, IEEE Transactions on Reliability 2000-03 pp. 49-59, IEEE Computer Society 2000, ISSN: 0018-9529, Digital Object Identifier: 10.1109/24.855536
- [6] Clifton A. Ericson II: Fault Tree Analysis - A History, in Proceedings of the 17th International System Safety Conference, System Safety Society, Unionville 1999, <http://www.fault-tree.net/papers/ericson-fta-history.pdf> (2012-04-15)
- [7] Rohit Gulati, Joanne Bechta Dugan: A Modular Approach for Analyzing Static and Dynamic Fault Trees, Proceedings of the 1997 Reliability and Maintainability Symposium in Philadelphia, PA pp. 57-63, IEEE Computer Society 1997, Print ISBN: 0-7803-3783-2
- [8] Ida Hogganvik, Ketil Stølen: A Graphical Approach to Risk Identification, Motivated by Empirical Investigations, 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), Lecture Notes in Computer Science 4199 pp. 574-588, Springer Berlin Heidelberg 2006, DOI: 10.1007/11880240\_40
- [9] Andrei Kolmogorov: Grundbegriffe der Wahrscheinlichkeitsrechnung, Springer Verlag Berlin 1933,
- [10] Mass Soldal Lund, Bjørnar Solhaug, Ketil Stølen: Model-Driven Risk Analysis, The CORAS Approach, Springer Verlag Berlin Heidelberg 2011, ISBN: 978-3-642-12322-1
- [11] Antoine Rauzy: New algorithms for fault trees analysis, Reliability Engineering and System Safety 40 (1993) pp. 203-211, Elsevier Science Publishers 1993
- [12] Michael Stamatelatos, Joanne Dugan, Joseph Fragola, Joseph Minarick, Jan Railsback: Fault Tree Handbook with Aerospace Applications, NASA, Washington 2002, <http://www.hq.nasa.gov/office/codeq/doctree/ftthb.pdf> (2012-04-15)
- [13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haas: Fault Tree Handbook, U.S. Nuclear Regulatory Commission, Washington 1981, <http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/sr0492.pdf> (2012-04-15)
- [14] H. A. Watson: Launch Control Safety Study, Section VII, Vol 1, Bell Laboratories, Murray Hill 1961
- [15] H. Götz, M. Nickolaus, T. Roßner, K. Salomon, "Model Based Testing - Modelling and generation of tests - basics, criteria for tool use, tools in the overview" (in German), iX Studie, 01/2009
- [16] Jürjens, J.: Secure Systems Development with UML, Springer, 2005

	<p align="center"><b>Review of security testing tools</b></p> <p align="center">Deliverable ID: <b>D4_3_T2_T3</b></p>	Page : 67 of 67
		Version: 1.1 Date : 02.07.2012
		Status : Final Confid : Public

- [17] Jürjens, J.; Schreck, J. & Yu, Y.: Automated Analysis of Permission-Based Security Using UMLsec; Fundamental Approaches to Software Engineering, 11th International Conference (FASE), Springer, 2008, 4961, 292-295
- [18] Jürjens, J. Jézéquel, J.-M.; Hussmann, H. & Cook, S. (Eds.) UMLsec: Extending UML for Secure Systems Development; The Unified Modeling Language, Springer Berlin / Heidelberg, 2002, 2460, 1-9
- [19] Lodderstedt, T.; Basin, D. A. & Doser, J. Jézéquel, J.-M.; Hußmann, H. & Cook, S. (Eds.) SecureUML: A UML-Based Modeling Language for Model-Driven Security; The Unified Modeling Language, 5th International Conference, Springer, 2002, 2460, 426-441
- [20] Information Security Indicators (ISI), Part 1: A full set of operational indicators for organizations to use to benchmark their security posture, Version 0.0.2, February 2012.
- [21] Common Criteria for Information Technology Security Evaluation, Version 3.1R3, July 2009
- [22] Common Methodology for Information Technology Security Evaluation, Version 3.1R3, July 2009
- [23] [1] Information security-wikipedia. [http://en.wikipedia.org/wiki/Information\\_security](http://en.wikipedia.org/wiki/Information_security), last date accessed 17.09.2011.
- [24] [2] Model based testing-wikipedia. [http://en.wikipedia.org/wiki/Model-based\\_testing](http://en.wikipedia.org/wiki/Model-based_testing), last date accessed 17.09.2011.
- [25] [3] P. Bourque and R. Dupuis. Guide to the software engineering body of knowledge 2004 version. Technical report 19759, IEEE Computer Society, 2004.
- [26] [4] William S. Chao. System Analysis and Design: SBC Software Architecture in Practice. Lambert Academic Publishing, 2009.
- [27] [5] The Committee on National Security Systems. National Information Assurance (IA) Glossary, CNSS Instruction No. 4009, 2010.
- [28] [6] IEEE Computer Society. IEEE 829 - Standard for Software and System Test Documentation, 2008.
- [29] [7] International Standards Organization. ISO 27000:2009(E), Information technology - Security techniques - Information security management systems - Overview and vocabulary, 2009.
- [30] [8] International Standards Organization. ISO 31000:2009(E), Risk management – Principles and guidelines, 2009.
- [31] [9] International Standards Organization. ISO 29119 Software and system engineering - Software Testing-Part 2 : Test process (draft), 2012.
- [32] [11] The Open Group. The Open Group Architecture Framework Version 9.1, 2011.
- [33] [12] Testing Standards Working Party. BS 7925-1 Vocabulary of terms in software testing. 1998.
- [34] [13] F. John Reh. [www.about.com](http://management.about.com/cs/generalmanagement/g/objective.htm).  
<http://management.about.com/cs/generalmanagement/g/objective.htm>, last date accessed 19.04.2012.