Network of Excellence

Deliverable D10.3

Prototype Tools to Support Risk and Cost Analysis
Abstract

The main objective of NESSoS WP10 is to develop a framework to facilitate a risk and cost aware software development life cycle (SDLC). The NESSoS engineering of secure software services is based on the principle of addressing security concerns from the very beginning in system analysis and design. The framework of WP10 should support the various engineering activities of the SDLC in identifying and assessing security risks, identifying options for security risk mitigation, and assessing the implied costs and the expected return on investment in security. This deliverable builds on the overall process for a risk and cost aware SDLC previously developed in the context of WP10. Specifically, the deliverable extends the WP10 portfolio of artifacts to support the tasks and activities that need to be conducted to assess security risk and cost. These artifacts include methods, techniques, modeling languages and tools. The work that is presented targets many of the research challenges that are imposed by the nature and characteristics of Future Internet (FI) software services and systems. Such challenges include the need for a continuous and iterative risk analysis process that is accommodated to the iterative SDLCs, the need for techniques to handle possible changes and evolutions with potential impact on security risks and costs, the need for handling the highly dynamic and heterogeneous nature of FI services and systems, as well as the need for adequate tool support.
Keyword List

*Future Internet, security, risk, cost, software development life cycle, return on investment in security*
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Type of Change</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Table of Contents</td>
<td>Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ketil Stølen (SINTEF)</td>
</tr>
<tr>
<td>1.0</td>
<td>First full draft</td>
<td>Jorge Cuellar (SIEMENS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marina Gonzalez (ATOS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Olav S. Ligaarden (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jan Stijohann (SIEMENS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ketil Stølen (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Le Minh Sang Tran (UNITN)</td>
</tr>
<tr>
<td>2.0</td>
<td>Review ready</td>
<td>Marina Gonzalez (ATOS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jan Stijohann (SIEMENS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Le Minh Sang Tran (UNITN)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Artsiom Yautsiukhin (CNR)</td>
</tr>
<tr>
<td>2.1</td>
<td>Revision after review</td>
<td>Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jan Stijohann (SIEMENS)</td>
</tr>
<tr>
<td>3.0</td>
<td>Final version</td>
<td>Bjørnar Solhaug (SINTEF)</td>
</tr>
<tr>
<td>3.1</td>
<td>Approved by the internal reviewers</td>
<td>Christophe Huygens (KUL)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marinella Petrocchi (CNR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Karsten Radatz (UDE)</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table of Contents</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>Acronyms</strong></td>
<td>11</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td>13</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>15</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>17</td>
</tr>
<tr>
<td><strong>2 Risk and Cost Aware SDLC</strong></td>
<td>19</td>
</tr>
<tr>
<td>2.1 Overall Process for the Risk and Cost Aware SDLC</td>
<td>19</td>
</tr>
<tr>
<td>2.2 Instantiation of the NESSoS Methods in the RC-SDLC Process</td>
<td>21</td>
</tr>
<tr>
<td><strong>3 Tool-Supported Software Application Risk Assessment</strong></td>
<td>25</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>25</td>
</tr>
<tr>
<td>3.1.1 Procedure Summary</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2 Instantiation in the Risk and Cost Aware SDLC</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Step 1: Business Risk Assessment</td>
<td>26</td>
</tr>
<tr>
<td>3.3 Step 2: Attack Surface Analysis and Quantification of Security Relevant Information</td>
<td>26</td>
</tr>
<tr>
<td>3.3.1 Desired Properties of a Security Overview</td>
<td>26</td>
</tr>
<tr>
<td>3.3.2 Information to Include in the Security Overview</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Step 3: Adapted Threat Modeling</td>
<td>27</td>
</tr>
<tr>
<td>3.5 Step 4: Risk Estimation</td>
<td>27</td>
</tr>
<tr>
<td>3.5.1 Technical Likelihood</td>
<td>27</td>
</tr>
<tr>
<td>3.5.2 Risk Estimation</td>
<td>28</td>
</tr>
<tr>
<td>3.6 Step 5: Selecting Appropriate Security Tests – Vulnerability Indicators</td>
<td>28</td>
</tr>
<tr>
<td>3.6.1 Vulnerability Indicators</td>
<td>29</td>
</tr>
<tr>
<td>3.7 Step 6: Security Testing</td>
<td>29</td>
</tr>
<tr>
<td>3.7.1 Testing Library</td>
<td>29</td>
</tr>
<tr>
<td>3.8 Summary</td>
<td>30</td>
</tr>
<tr>
<td><strong>4 Tool-Supported Risk Analysis and Management</strong></td>
<td>31</td>
</tr>
<tr>
<td>4.1 Overview of the RIGER Tool</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Benefits of using RIGER</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Objective</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Functionalities</td>
<td>32</td>
</tr>
<tr>
<td>4.5 User Roles</td>
<td>33</td>
</tr>
<tr>
<td>4.6 Process of Data Acquisition</td>
<td>33</td>
</tr>
<tr>
<td>4.7 Phases of Decision Making and Type of Decision Support</td>
<td>34</td>
</tr>
<tr>
<td>4.8 Extensions</td>
<td>34</td>
</tr>
<tr>
<td><strong>5 Tool-Supported Risk Analysis of Evolving Systems</strong></td>
<td>35</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Background</td>
<td>36</td>
</tr>
<tr>
<td>5.3 Tool Support</td>
<td>37</td>
</tr>
<tr>
<td>5.3.1 Features</td>
<td>37</td>
</tr>
</tbody>
</table>

NESSoS - 256980
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BWCS</td>
<td>Business Worst Case Scenario</td>
</tr>
<tr>
<td>DFD</td>
<td>Data Flow Diagram</td>
</tr>
<tr>
<td>EHR</td>
<td>Electronic Health Record</td>
</tr>
<tr>
<td>FI</td>
<td>Future Internet</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>ISMS</td>
<td>Information Security Management System</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>RC-SDLC</td>
<td>Risk and Cost aware Software Development Life Cycle</td>
</tr>
<tr>
<td>ROI</td>
<td>Return on Investment</td>
</tr>
<tr>
<td>ROISI</td>
<td>Return on Information Security Investment</td>
</tr>
<tr>
<td>ROSI</td>
<td>Return on Security Investment</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>SotA</td>
<td>State of the Art</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1: Risk management process ......................................................... 20
Figure 2.2: ISMS process ................................................................. 20
Figure 2.3: Instantiation of RC-SDLC in RUP ........................................... 21
Figure 5.1: Patient monitoring scenario .................................................. 35
Figure 5.2: Elements of CORAS threat diagrams ..................................... 36
Figure 5.3: Model artifacts ................................................................. 37
Figure 5.4: Detecting changes .............................................................. 38
Figure 5.5: Detecting inconsistencies ...................................................... 39
Figure 6.1: Conceptual model ............................................................... 42
Figure 6.2: Risk graph ....................................................................... 43
Figure 6.3: Countermeasure with treats relation ...................................... 43
Figure 6.4: Effect dependency relation .................................................. 43
Figure 6.5: CORAS instantiation .......................................................... 44
Figure 6.6: Decision diagram ............................................................... 45
Figure 7.1: Cost-Effective Enforcement of Access Control ....................... 49
Figure 7.2: Cost-Effective Enforcement of Usage Control ....................... 51
Figure 8.1: Monitoring artifacts .......................................................... 54
Figure 8.2: Models/descriptions ........................................................... 54
Figure 8.3: Overview of ValidKI ............................................................ 55
Figure A.1: Risk assessment in iterative SDLCS ..................................... 68
Figure A.2: Overview of the risk assessment process ............................... 69
Figure A.3: Data flow diagram ............................................................. 73
Figure A.4: Ontology of risk factors and their relationships ..................... 75
Figure A.5: Examples for all introduced terms ....................................... 79
List of Tables

Table 6.1: Treatment analysis process. ................................................................. 42

Table A.1: Vulnerability indicators ................................................................. 77
1 Introduction

The main objective of WP10 is to incept a risk and cost aware software development life cycle (SDLC) that enables practitioners to instantiate and exploit an engineering process that optimizes value-for-money in terms of minimizing effective risk while keeping cost low and justified. To this end WP10 is developing a risk and cost analysis process that is supported by a portfolio of methods, techniques and tools. Performing the analysis process should support practitioners in identifying and selecting appropriate security solutions to mitigate unacceptable risk. Moreover, the analysis should demonstrate the value of the solutions in terms of return on investment (ROI) in security from a business oriented perspective.

In previous WP10 work [10] we defined the overall process of the risk and cost aware software development life cycle (RC-SDLC). The process was defined to accommodate to an iterative SDLC and to conform to established standards on risk management [20] and information security management [23]. We moreover presented a number of more specific methods and techniques that can be utilized to support different phases or tasks in this overall process.

This deliverable builds on the previous WP10 work as well as promising existing methods identified in our state of the art and gap-analysis [53]. In particular, this deliverable elaborates on and extends the portfolio of methods to support the RC-SDLC presented in D10.2 [10]. Moreover, the deliverable describes various prototype tools to support the methods that are developed in WP10. The main focus is on integrating existing promising tools and accommodating them to the setting of the SDLC of Future Internet (FI) services and systems.

In Chapter 2 we briefly recapitulate the overall process for the RC-SDLC presented in the previous WP10 deliverable [10] and explain how the WP10 artifacts support this process. Some of these artifacts are methods, techniques and tools that are presented in the subsequent chapters. In Chapter 3 we present a process for continuous security risk assessment and mitigation during iterative software development. The chapter also discusses automation and required tool support. Chapter 4 presents a tool-supported approach to security risk analysis and management. The approach comes with a management suite to provide and maintain an Information Security Management System (ISMS) as defined in ISO/IEC 27001 [23]. Chapter 5 presents a prototype tool to support the modeling and analysis of evolving systems where changes are systematically dealt with by automated traceability from system models to risk models. In Chapter 6 we present a method to support the identification of cost-effective risk treatments. The method comes with language support for the modeling of risk treatments and their cost and effect, as well as a calculus with rules for reasoning about cost and effect. Chapter 7 presents a risk and cost-based approach to ensure cost-effective run-time enforcement of usage control policies. In Chapter 8 we present an approach to dynamic security risk assessment by indicator monitoring, including a method for designing and assessing the validity of such indicators. In Chapter 9 we give an overview of ongoing and planned interactions with other NESSoS WPs. The chapter also summarizes the most important WP10 dissemination of foreground, and gives an overview of the coverage of the WP10 tasks as described in the NESSoS DoW. Finally, in Chapter 10 we summarize and conclude.

Note that this deliverable only gives an overview of the artifacts and research results of WP10. For the full presentation of the methods, techniques and tools the reader is throughout the report referred to the relevant publications. For Chapter 3, a more comprehensive presentation is given in Appendix A.
2 Risk and Cost Aware SDLC

In this chapter we briefly recapitulate on the overall process for a risk and cost aware software development life cycle (RC-SDLC). The reader is referred to NESSoS deliverable D10.2 [10] for further details. We moreover give an overview of the WP10 methods, techniques and tools and explain how they can be instantiated in the overall RC-SDLC process to support its various tasks and phases. We also explain how these WP10 artifacts relate to each other, and how they can be orchestrated or integrated to leverage each other in conducting the overall process.

2.1 Overall Process for the Risk and Cost Aware SDLC

The aim of WP10 is to provide a RC-SDLC process that is generally applicable, that gives guidance on information security management, and that is adapted to an iterative SDLC.

To ensure general applicability, the RC-SDLC process is based on the ISO 31000 risk management standard [20]. Figure 2.1 is adapted from the standard and illustrates the seven activities of the risk management process. The five activities in the middle constitute the core activities of a risk analysis, and result in the documentation of risks, as well as the documentation of treatment options for unacceptable risks. The remaining two activities are continuous activities of the risk management process. ISO 31000 is quite general and does not prescribe any specific risk assessment methods or techniques to support conducting the activities. End-users, i.e. risk analysts, are hence free to use any existing risk analysis method that implements the standard (such as [2, 8, 37, 54]). For further support the ISO 31000 standard refers to ISO/IEC 31010 [24] which gives a comprehensive overview of existing risk assessment techniques, each of which supports various phases and tasks of the ISO 31000 process.

To ensure that the RC-SDLC comes with appropriate guidance on information security management we have also based the overall process on the ISO/IEC 27001 standard [23]. The standard defines an Information Security Management System (ISMS) as “the part of the overall management system, based on a business risk approach, to establish, implement, operate, monitor, review, maintain and improve information security”. The standard describes a Plan-Do-Check-Act process model for ISMS processes as illustrated in Figure 2.2. The ISO/IEC 27001 process can be understood as a specialization of the ISO 31000 risk management process tailored towards security risk analysis.

With respect to the SDLC we do not assume a specific service and system development process or method, but rather make the general assumption that the development lifecycle is iterative and comprising the typical activities from requirements and design, via implementation and testing, to deployment. Such iterative development processes can be categorized into three classes ranging from heavy to light, namely the spiral model, the unified process and agile methods. The reader is referred to Appendix A of NESSoS Deliverable D10.2 [10] for an overview.

With these standards and the iterative SDLC as background, WP10 defines the RC-SDLC as a software and service development lifecycle in which security risk and cost analysis is an integrated activity. The security risk and cost assessments will be drivers in the development process and support each of the other activities to various extents in the different iterations.

To give a concrete example of a RC-SDLC, Figure 2.3 shows its instantiation in the Rational Unified Process (RUP) [31]. RUP is structured around two dimensions as depicted in the figure. One dimension progresses along the lifespan of the system development and is divided into the four phases of inception, elaboration, construction and transition. The other dimension is activities, or disciplines, such as requirements, design, implementation and test, that run across all the phases. The process is highly iterative, with the possibility of several iterations within each of the phases. As can be seen from the figure, these iterations are iterations of the actual development, so each iteration includes all cross-cutting activities, however with different emphasis in different iterations. In Figure 2.3 we have added risk and cost analysis to the set of activities and indicated how this activity may be emphasized to various degrees over the development lifespan.

To summarize, the WP10 RC-SDLC is based on the ISO 31000 risk management standard and assumes an iterative SDLC. This means that end-users such as risk analysts and software developers have a wide flexibility with respect to selecting specific methods that instantiate or support this life cycle. Moreover, the RC-SDLC uses information security guidelines from ISO 27001 to specialize the risk analysis
Figure 2.1: Risk management process

Figure 2.2: ISMS process
The NESSoS project aims to develop novel artifacts to provide a portfolio of methods, techniques, and tools that support the RC-SDLC. The objective of WP10 is to develop risk assessment techniques that can be integrated into the SDLC of FI services and systems. The project addresses several research challenges that need to be addressed in order to adequately support a systematic security risk and cost analysis in the setting of FI software services and systems. The objective of WP10 is to develop novel artifacts to provide a portfolio of methods, techniques, and tools that support the RC-SDLC. In the following, we give an overview of the current status of this portfolio, summarizing the work and results of WP10 so far.

### 2.2 Instantiation of the NESSoS Methods in the RC-SDLC Process

Our main conclusions from the gap analysis are structured into the five following research areas, each of which with a potential to contribute to the overall goal of WP10.

**Risk methodology integration:** For analysts to consider security risk and cost from the early phases of the SDLC of FI services and systems, they often need to combine several methods and techniques. The aim of WP10 in this respect is to identify promising existing SotA frameworks, as well as developing novel methods and techniques, and combine and orchestrate these to support the RC-SDLC.

**Risk versus cost:** WP10 aims for a RC-SDLC in which the value of the identified security solutions can be demonstrated from a business oriented perspective. For this purpose, we build on existing techniques for estimating the return on security investment (ROSI). A challenge to be addressed is that while return on investment (ROI) techniques can be applied on investment in security, it is still not apparent how to measure the benefit of security solutions. As part of the solution, WP10 is developing cost assessment techniques that are combined with security risk assessment techniques.

**Risk from a run-time perspective:** Traditional security risk assessments are intended to be conducted from time to time, but the more and more dynamic FI services, systems, and environments require
faster re-evaluation of risks. In other words, real-time risk assessment is necessary for timely reaction to changes in the operational environment. While leveraging on SotA approaches to real-time security risk assessment, WP10 is developing techniques for run-time risk monitoring, dynamic risk and cost assessment, as well as run-time risk- and cost-based policy enforcement.

**Risk assessment for web applications:** For FI software applications in general and web applications in particular there is a need for specialized methods to integrate security risk assessment during development. A challenge is that such applications can be used for many different purposes to support different services and systems. WP10 develops techniques for automatic generation of risk models based on threat and vulnerability databases, as well as systematic use of static analysis and testing.

**Risk from a formal methods perspective:** A formal foundation for risk analysis facilitates the assessment by providing a rigorous basis for analyzing and reasoning about risks. WP10 builds on existing risk modeling notations to develop support for formal reasoning about both security risk and cost during the RC-SDLC. The research challenges include support for modular reasoning about risk, support for reasoning about security risk, support for reasoning about evolving risks and support for reasoning about dynamic risks at run-time.

The methodology targeted by WP10 will enrich the whole SDLC and has the potential of governing the development process with security risk and cost as dominant drivers. The goal is ambitious and obviously needs to be attacked from several different angles; no single approach, method or technology is likely to have the capacity to provide solutions to all the challenges of upholding a risk and cost aware SDLC. Moreover, instead of aiming at one holistic and integrated approach—which likely would be quite rigid and inflexible—we are developing a portfolio of complementary artifacts supporting different phases, activities or tasks of the overall process. The artifacts can leverage each other and be orchestrated in different ways throughout security risk and cost analysis process. In the following we give an overview of the most important artifacts developed in NESSoS WP10 so far, referring to both this deliverable and to the previous WP10 deliverable, i.e. D10.2 [10].

**Method for balancing risk and cost:** The method for balancing risk and cost presented in D10.2 targets the final phase of risk treatment in the risk management process of the ISO 31000 standard. Given the documentation of the identified risks and the treatment options, the method uses techniques for identifying, categorizing and estimating costs and benefits associated with the risk treatments within a specified time frame. This method complements the approach to the identification of cost effective risk treatments presented in Chapter 6 of this deliverable. The latter provides modeling techniques, analysis techniques and decision support for selecting optimal security risk treatments and can make use of the cost and benefit estimates of the former as input.

**Method for continuous assessment of risk and cost:** In D10.2 we presented a method for continuous assessment of risk and cost at run-time, designed to support access and usage decisions when there may be uncertainties regarding satisfaction or violations of the relevant policies. Chapter 7 continues this work, where different models for decision making are compared, using risk and cost as main criteria for comparison. The approach presented here also considers the problem of cost-effective attribute retrieval. A different approach to continuous security risk assessment is presented in Chapter 8, where risks are monitored and updated by means of indicator monitoring. The method supports both the identification of relevant indicators and the evaluation of their validity with respect to the specified objective. In the context of the ISO 31000 process, these approaches fit mostly with the monitor and review activity, although they also involve risk identification, estimation and evaluation.

Relevant in this setting is also the problem of security risk assessment of changing and evolving systems, where the risk assessment models and results need to be updated in order to maintain their validity. In previous work we have developed method and modeling support for handing change, and in Chapter 5 of this deliverable we present tool support for automation of several of the assessment tasks, in particular the traceability of changes from the target system models to the risk models. The tool is developed to support the whole risk analysis process from context establishment, through risk assessment, to risk treatment.
Method for risk and cost assessment of web and software applications: As part of the WP10 portfolio we are developing methods to continuously identify and assess applications’ security risks during an iterative software development process. In D10.2 we presented an approach to capability-based threat modeling focusing on web applications. The approach makes use of repositories of identified vulnerabilities as well as capabilities of attackers to automatically generate risk models during risk identification. A similar approach is presented in Chapter 3, addressing FI software applications in general. The latter approach covers the whole risk assessment process in a highly iterative way, and with the use of security tests to support the risk estimation. While the former approach particularly covers the initial steps of the risk analysis process, the latter covers all three risk assessment phases as defined by ISO 31000.

Formal foundation: In D10.2 we presented a formal foundation for risk assessment, aiming to provide a rigorous basis for analyzing and reasoning about risk in the risk management process. The formalization included a formal syntax and semantics of risk models, as well as different calculi to support reasoning about risks. In Chapter 6 we build on these calculi, extending them to provide support for the precise reasoning about treatment options for unacceptable security risks, including their cost and effect.

Tool support: The WP10 portfolio also comes with several tools to support different risk assessment tasks. Throughout the deliverable we discuss tool support while the various artifacts are introduced and presented. However, two chapters worth mentioning in this respect are Chapter 4 and Chapter 5 as both are solely for tool presentation. The former presents the RIGER tool, which is a management suite to support organizations in maintaining an ISMS. The latter presents the tool support for automating the change management discussed above.
3 Tool-Supported Software Application Risk Assessment

In this chapter we present a tool-supported process for security risk identification and mitigation during the software development process. The chapter gives a shortened overview of the more comprehensive presentation in Appendix A.

3.1 Introduction

The goal of our research activities is to develop a process for continuous security risk assessment and risk reduction in highly iterative software development processes [30]. The process should leverage existing collections of threats and vulnerabilities, as well as methods and tools used in static and dynamic software analysis. We do not intend to replace human judgment with a fully automated process based on mathematical formulas. The process should rather describe and support (with tools) the information retrieval and the involved activities. It will still be the risk analysts’ task to evaluate what information is most important and to do the final interpretation. The process will, however, improve the quality of risk analysts’ judgments and make them more comprehensive and more transparent for all involved parties including developers, security testers, architects, and managers.

3.1.1 Procedure Summary

The process combines manual tasks with tool support whenever possible. It can be summarized as follows:

1. The application business owner identifies security requirements, deduces corresponding business worst case scenarios and estimates the impact of the latter (step 1). Business worst case scenarios (BWCS) are short, informal, and non-technical descriptions of business consequences that may result from the violations of the security requirements. In addition, a security analyst determines non-technical likelihood factors such as the expected attacker type and the attractiveness of the target. Next, the analyst creates a security overview reflecting the application’s attack surface and security architecture (step 2). The required information is obtained via personal interviews with the developers but also through a tool-supported static and dynamic analysis of the application. The security overview, illustrated in form of an annotated data flow diagram (DFD), serves as a basis for an adapted threat modeling (step 3). During this activity, the analyst examines which abstract threats such as information disclosure, denial of service or escalation of privilege of what application parts can lead to the previously defined BWCS. The identified technical threats are then prioritized according to their technical likelihood which is initially based on the information captured in the security overview. At this point, the analyst can perform an initial risk estimation for each BWCS (step 4) considering the business impact and the technical and non-technical likelihood of all associated technical threats. If the risk estimation yields unacceptable results, the analyst can modify the application to reduce the attack surface. Furthermore, he can select (step 5) and conduct (step 6) suitable security tests to better estimate the real likelihood of the concerned technical threats. Based on the kind of testing activities and their results, the analyst can re-estimate the technical likelihood of the corresponding technical threats. Further tests or modifications of the application may be performed until an acceptable risk level for all BWCS is achieved. Steps 2 to 6 may be redone at the end of a development iteration.

Figure A.1 in Appendix A shows how the process is integrated into an iterative development life cycle. The steps annotated with (T) can be tool supported, whereas the (M) indicates purely manual steps. Figure A.2 presents a graphical overview of the relations between the involved activities and artifacts. Figure A.5 in the appendix gives concrete examples for all introduced terms. The following sections describe each step in more detail.

3.1.2 Instantiation in the Risk and Cost Aware SDLC

Our risk and cost aware software development life cycle (RC-SDLC) is partly based on the ISO 31000 risk management process depicted in Figure 2.1. The process proposed in this chapter contributes to the
three first phases as follows:

**Establish the context:** Step 1 and 2 cover the identification of assets and existing controls, respectively.

**Identify risks:** The identification of threats is done when defining the expected attacker type in step 1, and, on a more technical level, during the adapted threat modeling in step 3. Vulnerabilities are identified via practical security tests in step 6.

**Estimate risks:** The identification of consequences is done by the definition of the BWCS in step 1. Both assessment of incident likelihood and estimation of risk level are part of step 4.

The risk evaluation and risk treatment phases are also implicitly covered, as step 4 includes the option to mitigate or further analyze the system if the risk is not yet acceptable. A graphical overview of the process and how it covers the activities of the ISO 31000 is given in Figure A.2.

### 3.2 Step 1: Business Risk Assessment

The business risk assessment is conducted by the application business owner. It is supposed to furnish the security requirements, the business worst case scenarios with associated impacts, and the non-technical likelihood factors. Section A.2 in the appendix contains examples for the first two terms in the context of the WP11 eHealth use case described in D11.2 [9].

**Security Requirements**

A *security requirement* is a non-technical asset together with a security property.

**Business Worst Case Scenarios (BWCS)**

*Business worst case scenarios* are short, informal, and non-technical descriptions of business consequences that may result from the violations of the security requirements. Every BWCS has an associated estimated impact which is one of the primary factors used for the later risk estimation. Section 3.5 lists all factors considered for the final risk estimation.

**Non-Technical Likelihood Factors**

The likelihood of a BWCS is composed of technical and non-technical factors. The first are determined for each technical threat, see Section 3.4 for a definition, while the latter are mostly the same for a BWCS or even all BWCS. The non-technical likelihood includes the expected attacker’s motivation, the resulting attractiveness of the target, and the attacker’s skill level. Attackers are categorized in three different categories according to their skill level and resources: opportunity rider, advanced and professional attacker.

### 3.3 Step 2: Attack Surface Analysis and Quantification of Security Relevant Information

The goal of this step is to capture, structure, and comprehend the security relevant technical aspects of the software system under test. The resulting *security overview* is a crucial point for the proposed process as it connects the BWCS with technical threats, and the latter with concrete practical testing activities. This is the foundation for a transparent risk analysis and efficient risk reduction.

#### 3.3.1 Desired Properties of a Security Overview

A security overview should have the following properties to be of greatest utility for the proposed process.

- Documented to ensure that all involved parties have the same conception of the software system
- Created by the risk analysts, as (design) documents are often inappropriate, if they exist at all
• Generated with tool support to shorten a time consuming and tedious manual creation
• Based on static and dynamic binary analysis since binaries are a uniform, machine readable, and reliable source of information
• Syntactically standardized as it helps recognizing recurring problems and facilitates transformation, for example to a data flow diagram used for the adapted threat modeling as described in Section 3.4

Section A.3.1 in the appendix discusses the properties and their advantages in more detail. Section A.3.2 presents first ideas for the practical, automated extraction of security relevant information.

### 3.3.2 Information to Include in the Security Overview

The security relevant information falls basically into three main categories:

- **Data flow related aspects** such as processes, data flows, data stores, and local or trust boundaries
- **Security controls** for example encryption, integrity checks, input validation, or data sanitization
- **Design decisions** including protocols, data formats, file permissions and process privileges

Section A.3.3 in the appendix contains a more extensive list of relevant information items to extract.

### 3.4 Step 3: Adapted Threat Modeling

In the context of the proposed risk analysis process, *adapted threat modeling* refers to the analysis of a system's architecture and design. The goal is to find out which threats, such as denial of service, disclosure of information, escalation of privilege, of what parts of the application can lead to the previously defined BWCS. Obviously, this mapping, or, "translation" of BWCS to technical threats is only possible for software which is built for a defined business purpose or whose area of application is known to the risk analyst. This constitutes the adaptation of the original Microsoft STRIDE\(^1\) threat modeling [17] which does not consider concrete business use cases. As STRIDE threat modeling, the adapted version is based on a data flow diagram (DFD). Thus a *technical threat* can be defined as a tuple of one or more DFD elements and a STRIDE threat. Figure A.3 in the appendix presents a simplified DFD for the example medical system mentioned in step 1. A DFD consists of four main symbols: Circles represent logical processes; rectangles represent external entities; double horizontal lines represent passive data stores; and arrows represent data flows. The special characters are not part of the standard notation. They represent examples for possible annotations. Here they mean "not encrypted" (-), "encrypted" (+), "user privileges" (\(\sim\)) and "elevated privileges" (*). The DFD can be manually derived from the security overview created in step 2, or, if the overview exists in a standardized form, it could be generated with tool-support. There are two approaches to the mapping of BWCS to technical threats:

**Top-down:** For every BWCS, examine which technical threats could lead to the BWCS.

**Bottom-up:** For each DFD model element, brainstorm if any technical threat could pose a security problem which could lead to a, possibly not yet identified, BWCS.

Practical aspects such as a possible slimming down of the adapted threat modeling and the advantages of DFDs are discussed in Section A.4.1 in the appendix.

### 3.5 Step 4: Risk Estimation

#### 3.5.1 Technical Likelihood

The overall risk estimation is based on two principal factors: impact and likelihood. The latter is composed of technical and non-technical factors. While risk, impact, and non-technical likelihood are estimated per

\(^1\)STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.
BWCS (see step 1, Section 3.2) the technical likelihood is estimated per technical threat. Figure A.4 in the appendix shows an ontology which contains all considered risk factors and their relationships with each other. The most significant non-technical likelihood factors are:

- Identified vulnerability indicators, see Section 3.6
- The sophistication of security tests, if any have been performed, that cover the technical threats
- Potentially discovered vulnerabilities under consideration of the chance for occurrence, detection and exploitation; the required technical skill for exploitation; and requirements and mitigation factors such as physical access restrictions, restricted LANs or required user accounts

The technical likelihood allows the analyst to determine the technical threats with the highest "risk reduction potential". This means the architect can focus the development and testing effort on the corresponding application parts and thus achieve the most efficient risk reduction possible.

3.5.2 Risk Estimation

The risk estimation for BWCS can be (re-)done as soon as a modification has manifested itself in the application binaries. It involves the following steps, from which all but the first may be repeated after every modification:

1. Estimate impact and non-technical likelihood. This is done in step 1 of the overall process, described in Section 3.2.

2. For each technical threat associated with the currently selected BWCS: Estimate the technical likelihood by evaluating relevant vulnerability indicators and considering eventual, previously performed practical tests. Note that these might have become obsolete due to modification of the application.

3. Aggregate the technical likelihood of all associated technical threats and merge the result with the threat-independent non-technical likelihood factors.

4. Estimate the risk for the BWCS based on impact and likelihood determined in the previous steps.

5. If the risk is too high, there are two options:
   (a) Reduce the attack surface by modifying the application to have less technical threats or vulnerability indicators.
   (b) Perform more, or more sophisticated, practical security tests to increase the precision of the estimation. Therefore:
      i. Order all technical threats according to their impact on the overall risk and select the highest ranked.
      ii. Perform a not yet conducted security test covering this threat. The test may be taken from a relevant vulnerability indicator, from the test library introduced in Section 3.7, or it can be a freestyle test.
      iii. Evaluate the practical results and re-estimate the technical likelihood. Identified vulnerabilities are evaluated according to the properties listed in Section 3.5.1. Consider especially the security assurance level (see Section 3.7.1) of the performed tests in regard to the expected attacker’s skill level.
      iv. Go back to step 3 of the risk estimation.

3.6 Step 5: Selecting Appropriate Security Tests – Vulnerability Indicators

As described in Section 3.5, security tests are a means for more precise risk estimation. This, however, requires a proper mapping of technical threats to appropriate tests. To make this step more systematic, and ideally tool-supported, we introduce the concept of vulnerability indicators.
3.6.1 Vulnerability Indicators

A vulnerability indicator is a pattern in a security overview (see Section 3.3) that points to a possible vulnerability, together with a suggested practical test, or family of tests, to find out if the indicated vulnerability indeed exists. The vulnerability indicator approach is therefore similar to that of the "threat library" proposed by the EMC Corporation [12]. The EMC threat library links DFD patterns with "threats" that basically correspond to our tests. In both concepts, most patterns do not describe actual vulnerabilities, but conditions signifying an increased likelihood that a certain vulnerability is present. While the EMC threat library entries are based on patterns in manually created DFDs, vulnerability indicators comprise patterns in a security overview. This means that vulnerability indicators can not only be based on manually created design information but also on implementation properties extracted via static/dynamic binary analysis. Good candidates are properties that are either a prerequisite for, or frequently occur in the context of certain vulnerabilities.

To our knowledge, the idea of vulnerability indicators as described above has not yet been presented in literature. Related research fields such as "vulnerability prediction" and "security metrics" also deal with the quantification of security relevant information, but their goals and results are less technical, less specific, and in most cases not based on binary analysis.

The proposed process suggests setting up and maintaining a collection of vulnerability indicators that the risk analyst can re-use for different applications. The indicators listed in Table A.1 are only examples to demonstrate the general idea. In Section A.6.1 we outline how to systematically extend the collection of vulnerability indicators and thus obtain a more complete set.

3.7 Step 6: Security Testing

One possibility to cope with identified technical threats is to cover them with security tests to check the practical system resistance against them. The proposed process supports the selection of appropriate security tests, as explained in Section 3.5, with the help of vulnerability indicators. The integration of a systematic collection of practical security tests, in form of a testing library, completes the support for an efficient use of testing resources.

3.7.1 Testing Library

Format and Security Assurance Levels (SAL)

The library entries are grouped in blocks such as "web applications", "network traffic analysis", "reverse engineering / binary analysis" or "fuzzing". The creation of such blocks supports the bottom-up usage explained below and eases maintenance of the library. In addition, each library entry is labeled with a so-called Security Assurance Level (SAL) which reflects the level of confidence in the results of a test. It is determined by estimating the required skill level that an attacker needs in order to perform the test. The levels range from 1 (very easy) to 3 (very complex). The SAL / attacker-type ratio is an important factor for the estimation of the technical likelihood. Given the case, for example, that an SAL 2 test unveils a vulnerability. An advanced or professional attacker is much more likely to detect and exploit it than an opportunity rider. Section A.7.1 in the appendix contains further desired characteristics of test entries.

Usage

The primary library use is in a "top-down" manner, that is to cover technical threats with practical tests, as exemplary shown in Figure A.5. The figure also explains all other introduced terms using the example of a medical system introduced in Section A.2. P1, DF1, and DF2 refer to a process and data flows being part of the simplified DFD in Figure A.3, which is also part of the medical system example. From a practical point of view, the analyst should also proceed bottom-up by studying the test library and checking for possibly forgotten technical threats.
3.8 Summary

In this chapter we have outlined a process for continuous, transparent risk assessment and efficient risk reduction during iterative software development. The process consists of six steps: 1) The definition of business worst case scenarios, 2) the creation of a security overview, 3) an adapted threat modeling, 4) the actual risk estimation, 5) the selection of appropriate practical security tests and 6) the actual testing. For each of these steps, we discuss possible tool support and sketch the first ideas for its implementation. Moreover, we introduced three key concepts: The creation of a security overview based on static and dynamic binary analysis; the use of vulnerability indicators that point to possible vulnerabilities and suggest appropriate practical test; and finally the set up and utilization of a test library with entries categorized according to Security Assurance Levels. Used in combination, these concepts allow a transparent risk assessment and efficient risk reduction.

As described in Section 3.1.2 and depicted in Figure A.2 of the appendix, the process supports all tasks of the risk management process defined by the ISO 31000 risk management standard on which our RC-SDLC is based. The process is moreover tailored for security risk assessment, which is a main concern in secure engineering of software services.

The process is developed to be conducted in close interaction with the mainstream (security) engineering life cycle, and is particularly suitable for highly iterative SDLCs. The process has the potential to support all engineering activities of the SDLC, but mostly requirements, architecture and implementation. Security cost assessment is not part of the security risk assessment process defined in this chapter, but we envisage the complementary use of the cost assessment methods developed in WP10, both in this deliverable and in D10.2.
4 Tool-Supported Risk Analysis and Management

This chapter gives a presentation of the RIGER tool, a management suite which provides an Information Security Management System (ISMS) that allows customers to deploy a solution aligned to their maturity level and focused on Asset Risk Analysis and Management. It is a fully customizable tool built to support the Magerit methodology [7]. The tool is available since 2009, and is currently being extended to adapt it more closely to the NESSoS risk and cost aware SDLC. The tool is already compatible with the ISO 31000 risk management process, and is moreover based on the ISO/IEC 27001 standard, but further support is required for cost-benefit analyses to ensure and demonstrate return on investment in security.

4.1 Overview of the RIGER Tool

The tool aims at controlling the risk associated to any kind of assets (human, physical objects and/or services) that represent the values of an organization. It is a fully customizable tool suitable for any kind of scenarios offering an updated and centralized asset, vulnerabilities and safeguards inventory. It also includes Risk Management, in compliance with legal requirements and Continuous Improvement in a unified process following the ISO/IEC 27001 framework [23]. Recall that, in deliverable D10.2, we decided to base our guidelines in a subset of the requirements given in the ISO/IEC 270001 standard. Therefore this tool is specially suited to support the activities and the development life cycle proposed in this work package.

The methodologies Magerit, PILAR, and PILAR BASIC [52] offer similar capabilities to what the RIGER tool supports. In fact, it is compatible with EAR family tools (PILAR, PILAR Basic and µPILAR and it is able to upload vulnerabilities from NESSUS [55]).

The usual inputs for the tool are a set of assets, possibly with dependencies between them, valuation dimensions, threats, and safeguards.

As the main output, the tool displays a security planning and control panel with values allowing a comparison considering indicators and metrics based on current and historical information.

4.2 Benefits of using RIGER

RIGER offers an updated and centralized asset inventory. It includes Risk Management, Compliance with legal requirements, and support to continuous improvement in a unified process following IEC/ISO 27001 Framework. RIGER helps in maintaining and/or improving all type of security measures, technical or organizational, implanted or to implant. The tool is modular and it may be customized, the company could include new security frameworks by means of developing new security libraries (Knowledge Base). By using RIGER, the implementation and the maintenance of an ISMS will be more efficient, and effective Risk Analysis and Management method phases will be better supported.

Regarding risk identification, the tool supports the following:

- Assets: identification, classification, dependencies between assets, and value.
- Threats: identification of relationships with assets and evaluation of vulnerability.
- Safeguards: identification and evaluation.

Regarding Risk assessment, the tool supports the following:

- Accumulated impact and risk,
- Deflected impact and risk, and
- Risk evaluation.

Regarding Risk analysis, the tool supports the following:

- Risk assessment: (See above)
• Risk treatment: Support of scenarios: phases, what if, security projects, long-term objectives.

• Risk acceptance: Security indicators

• Risk communication: Definition of reports containing the findings and conclusions from a risk analysis and management project: value model, risk map, safeguard evaluation, risk status, deficiencies report and security plan.

Its main steps are the following:

• Determine the relevant assets for the organization, their inter-relationships and their value i.e. what prejudice (cost) would be caused by their degradation.

• Determine the vulnerabilities to which those assets are exposed

• Determine the threats related to the vulnerabilities

• Determine what safeguards are available and how effective they are against the risk.

• Estimate the impact, defined as the damage to the asset arising from the appearance of the threat.

• Estimate the risk, defined as the weighted impact on the rate of occurrence (or the expectation of appearance) of the threat. Once a full cycle has finished there is a revision to determine the residual impact and risk following Deming Cycle: Plan, Do, Check, Act (PDCA)

4.3 Objective

The tool was created as a framework for the Magerit [7] methodology and shares with it the same objectives, which are:

1. To make those responsible for security aware of the existence of risks and of the need to treat them in time.

2. To offer a systematic method for analysing these risks.

3. To help in describing and planning the appropriate measures for keeping the risks under control.

4. Manage security issues to ensure continuous improvement using the Deming Cycle: Plan, Do, Check, Act (PDCA)

4.4 Functionalities

RIGER main task is to help to maintain and/or improve all type of security measures, technical or organizational, deployed or to be deployed. It is fully compatible with Magerit Risk Analysis and Management (Assets, Threads, Safeguards, Impact and Risk) and uses Deming Cycle: Plan, Do, Check, Act (PDCA) to ensure continuous improvement.

Regarding architecture, the tool is very adaptive and its architecture can be easily upgraded. Regarding the architecture of the tool, some of its components are a User Interface, a Database server, an Application server, an Active Directory, and an Enterprise Content Manager. ¹

RIGER main functionalities are:

• resources internal management,

• management of documents of the of ISMS policy,

• management of assets,

• management of vulnerabilities,

¹For more details, please, contact Atos directly since it is confidential information.
• management of controls (safeguards),
• management of threats and risks,
• continuous improvement,
• security planning, and
• balanced scoreboard.

The main approach of this methodology is qualitative, although it also provides a quantitative approach for some functionalities. It allows to deal with availability, confidentiality, integrity, and authenticity.

The tool includes Risk Management, in compliance with legal requirements and Continuous Improvement in a unified process following IEC/ISO 27001 Framework. It is also in compliance with the Spanish law LOPD (Ley Orgánica de Protección de Datos- Law to protect data).

4.5 User Roles

RIGER is a customizable, multi-user and multi-role application. By default there are five roles:

• ISMS Supervisor: in charge of supervision of correct operation of the ISMS (For example: CISO Chief Information Security Officer, Security Board . . .)

• ISMS Manager: in charge of managing the operation of the ISMS (For example: CISO Chief Information Security Officer, Security Board . . .)

• Risks Operator: in charge of operating functions of the ISMS as far as risk are concerned, as well as inventory assets.

• Administrator: in charge of administration of the RIGER application like user management, and importing / exporting of files.

• Compliance Operator: in charge of operating functions of the ISMS about compliance.

These default roles are customizable; they can be changed, and new roles may be added. Moreover, they have different levels of access granted to the information depending on the activities they are supposed to perform using the tool (RBAC policies).

4.6 Process of Data Acquisition

First thing you need to get data is to know very well the organization/s you want to protect and analyse in terms of risk management. This is needed to be able to identify the assets properly. Interviews, workshops and meetings could be very useful.

The main difficulty to acquire data is actually the lack of knowledge of the organization and the scenarios that affect it so as to define the assets, vulnerabilities, threats and safeguards. On the other hand, if the user has the knowledge he must be able to properly sort out in the categories desired.

After the analysis, the tool displays a security planning and control panel with values allowing a comparison considering indicators and metrics based on current and historical information. It shows a balanced scoreboard of the following: value model (assets), risk map, safeguard evaluation, risk status and deficiencies reports.

The user can define a catalogue of threats and safeguards according to each scenario. Also assets can be defined with the required features suitable for each situation.
4.7 Phases of Decision Making and Type of Decision Support

After the identification of the assets, their vulnerabilities are identified as well. Once this has been done, risk managers realize what are the problems of the organization in terms of security that must be solved.

The tool displays a security planning and control panel with values allowing a comparison considering indicators and metrics based on current and historical information. It shows a balanced scoreboard of the following: value model (assets), risk map, safeguard evaluation, risk status and deficiencies reports.

The typical damage categories that RIGER relates to are economic, e.g., devaluation of assets, and those regarding critical infrastructure, like Critical Information Infrastructure.

RIGER deals with the average incident probability, particularly with low probabilities that the user may introduce in forms of the expected probability. These values are taken into account in the tool core algorithms for the reports and outputs. However, the tool does not deal with uncertainty.

RIGER can report on the effects of a security incident or measure, e.g., information systems, regulatory frameworks, etc..

The tool is owned by Atos, and it is being used in the EU project SEMIRAMIS. On top of that it is being used internally in Atos and by some of its clients, mainly in the telecommunications area, including a major client of that field.

The tool consists of several forms used to set necessary values and submit required information, but nevertheless they are clear and user friendly. Overall with some basic risk management knowledge RIGER is easy to use. The amount of resources needed to use the tool depend on the size and complexity of the domain/s, the level of the decomposition of assets, vulnerabilities, threats, safeguards you want to deal with. Once all the information needed is gathered, a small team may perform the analysis and management. Usually, around 1 PM would be enough but it could be more if the catalogues are very big and you take into account several domains at the same time.

Regarding the competence level required to handle the tool, because of the amount of functionalities of the tool, it is compulsory to read some parts of the user manual. Other parts may be only consulted whenever needed. With the knowledge of the basis for security and risks you may follow it so as to be able to apply it later. On top of that, people responsible for the analysis should have knowledge about the main concepts of risk security about the specific case (e.g. which kind of safeguards can face certain threats).

4.8 Extensions

Within ATOS Research & Innovation Department, we are currently extending this prototype towards offering support not only to perform a cost analysis but a cost-benefit analysis that would put this tool in position to deal with a risk and cost aware SDLC as it is being developed in NESSoS. In particular, RIGER is compatible with the NESSoS overall process for a risk and cost aware SDLC as both are based on the ISO/IEC 27001 standard for security risk management. As to the risk management process depicted in Figure 2.1 on page 20, RIGER offers support for all phases from context establishment, through risk assessment, to risk treatment. In the near future we plan to perform a risk analysis assessment using RIGER to check its functionality on the eHealth scenario that is proposed by Siemens within NESSoS. This scenario is focused on Electronic Health Records Management (it is detailed in deliverable D11.3).

During current period, ATOS has been collaborating with WP6 and WP11 towards ensuring a successful eRISE contest (task mostly within WP6 and WP11). In particular, since most of the development methodologies applied to the Smart Grid scenario contained or were focused on performing risk analysis, e.g. CORAS, SREP or LINDDUN, we think that these cross-WP activities’ results are also of relevance for WP10 since they help to validate the methodologies proposed in D10.2, e.g. CORAS.
5 Tool-Supported Risk Analysis of Evolving Systems

Future Internet (FI) services are characterized by a heterogeneous, modular and highly dynamic nature. Appropriately understanding, documenting and assessing the security risks therefore requires methods and techniques that are accommodated to this nature. The problem we address in this chapter is how to maintain the validity of risk assessment results and keep the risk picture up to date while services and systems evolve. This is relevant also during development to maintain consistency with the mainstream system and service engineering process.

5.1 Introduction

As an example consider the patient monitoring scenario described in NESSoS deliverable D11.2 [9] and illustrated in Figure 5.1. This electronic health (eHealth) care scenario is related to both Internet of Things and to mobile health (mHealth); the patients are monitored by wearable diagnostic sensors that transmit data to an eHealth server via a mobile device. Both patients and general practitioners (GPs) can access the server via their browsers, and the server is in turn connected to the electronic health record (EHR) database. An identity provider (idP) authenticates the user and his/her devices.

Identifying and assessing the security risks in this setting require risk assessment of all components, their use and environment, the underlying infrastructure, and so forth. In the FI setting, we could moreover envisage even further services provided to the GP and the patient, for example from fitness centers, pharmacies and hospitals, each of which affecting the security risks, such as risks with respect to confidentiality of EHR and privacy of the patient. The challenge is that a change in only one component or service may have impact on the risk picture as a whole, and therefore requires the security risk assessment to be conducted from scratch.

The problem of maintaining consistency between the target of analysis and the security risk assessment is evident also during system and service development. Integrating risk assessment in the secure system engineering process means that the risk assessment makes active use of documentation resulting from other phases of the development process as input, both from the mainstream system engineering process and from the security engineering process. Conversely, activities of the other processes, such as (security) requirements engineering and system (security) design use the risk assessment results as input. In this chapter we assume a model-driven approach to secure system engineering in which model artifacts serve as both input to and output from the various activities. The timely challenge we investigate is how to orchestrate risk assessment with mainstream system engineering, and—in particular—how to maintain consistency between the domains and their models.

We present a risk analysis tool that supports the modeling and analysis of changing and evolving risks. The tool supports traceability of changes from system models to risk models, as well as the explicit
modeling and assessment of the impact of the changes on the overall risk picture.

The chapter is organized as follows. In Section 5.2 we give the background to the tool-supported approach. In Section 5.3 we give an overview of the features of the tool and how they are supported. We moreover give some small examples with screen shots to show a few aspects of the use of the tool. Finally, we conclude in Section 5.4. For a more detailed description of the tool, as well as an evaluation, the reader is referred to the full paper [50]. The paper also discusses related work and gives references to reports on validation activities.

5.2 Background

According to the ISO 31000 standard [20] risk analysis should be regularly conducted for the purpose of assessing and mitigating risks. In order to handle change and maintain consistency in a systematic and methodical way, each of the risk assessment activities (as depicted in Figure 2.1 on page 20) must be supported by specialized guidelines and techniques. In previous work we have generalized the ISO process to include such guidelines throughout the whole process, and in turn instantiated this generalization in the CORAS method [38]. At the same time the CORAS language is extended and generalized to offer risk modeling support for the traceability techniques, for maintaining consistency, and for assessing changes to risks. The reader is referred to [38, 50] for the details about the generalized method and language. Focusing on the risk assessment, the methodological guidelines are summarized as follows.

1. Identify and document risks by using as input the target description before changes have been taken into account.
2. Establish and document the traceability between the target description before change and the risk documentation resulting from the previous step.
3. Based on the traceability and the description of the changed target, identify the parts of the risk documentation that are persistent under change.
4. Conduct the risk identification of the changed target only with respect to the parts of the target and the risks that are affected by the change.

In conducting these activities we make active use of three model artifacts, namely the system model, the risk model and the trace model. The system model needs to capture the relevant aspects of the target of analysis and can be built using any suitable notation, such as the UML [42]. For risk modeling CORAS threat diagrams are used. Threat diagrams document risks by describing how threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents. A risk is the likelihood of an unwanted incident and its consequence for a specific asset. Risks are estimated by annotating each identified unwanted incident with a likelihood and a consequence. The trace model is the specification of the traceability links between elements of the system model and the risk model. The graphical language constructs are shown in Figure 5.2. In the threat diagrams, these elements are related to describe how unwanted incidents may arise. The target segment is connected to any threat diagram element to visualize the traceability links to the system model elements.

The tool to support the activities has been developed as a plug-in to Eclipse such that it can be easily extended with new features and easily integrated with other Eclipse-based tools. To ensure that the
models that are created using the tool are stored in a standard format, it is based on Eclipse Modeling Framework (EMF).

### 5.3 Tool Support

In this section we first give an overview of the underlying features of the tool and how they are supported. Next we give some small examples with screen shots to demonstrate its use.

#### 5.3.1 Features

The system model $S$ is created in a separate tool and can be of an arbitrary language, provided that the meta-model $M$ is an (Eclipse) Ecore meta-model. Based on $S$, risks are identified and modeled using our tool to create the risk model $R$. To maintain consistency between $S$ and $R$ during the system engineering process in which $S$ may evolve, the trace model is created in the tool to enable traceability of changes from $S$ to $R$. Basically, the trace model is a set of pairs of one element from $S$ and one element from $R$. To enable the specification of the trace model in the tool, the system model $S$ is first imported to the tool. To this end the system model must conform to a generic system meta-model $M_C$ that is used by the tool to represent any model. It is the use of the generic meta-model $M_C$ that allows any language to be used for system modeling in the mainstream engineering process, but this requires the user to define a mapping from $M$ to $M_C$ as illustrated in Figure 5.3. This is a one-off task for a chosen system modeling language, but requires some expertise in language design. The mapping is defined by a set $TR$ of transformation rules that apply to elements, element attributes and element references. As shown by Figure 5.3, the tool uses $TR$ to transform $S$ to the representation $S_C$ in the tool. The latter is a set of uniquely indexed tuples that is stored in a table format (which is not intended for viewing or editing the system model as this is conducted externally in a designated tool). In Figure 5.3 we moreover see that the trace model $T$ is specified on the risk model $R$ and the tool representation $S_C$ of the system model. Note that it is only the risk analyst that makes use of all the model artifacts depicted in Figure 5.3; other stakeholders, such as engineers and executives, need only relate to the graphical models $S$ and $R$. As for the trace model $T$, this is visualized in $R$ as icons with references to $S$.

For a valid risk model $R$ that is consistent with $S$, and in which the trace model $T$ is complete, the tool is ready to support the handling of system changes that may occur. First, $S$ is updated to $S'$ to document the changes. Next, the tool imports $S'$ and transforms it according to $TR$. To capture the changes the tool automatically generates the delta (difference) between $S$ and $S'$ and creates an updated representation $S_C'$. Importantly, whereas $S$ and $S'$ are two separate models, $S_C'$ is a single representation in which the changes are explicitly specified. This is conducted automatically by the tool based on the delta; for each element it is indicated whether it has been added, deleted or modified. Based on $S_C'$ and the detected system changes the tool now uses the trace model $T$ to detect the parts of the risk model $R$ that may be affected. Any such part is then flagged to support the risk analysts in reassessing only what is necessary to restore the validity of the risk assessment and the consistency with the mainstream engineering process. Once the risk model is updated and consistency restored, the trace model $T$ must be updated to prepare for any further future system changes.

The approach handles changes in batches, typically after each round of an iterative development process, where each batch is captured by the updated system model $S$. We often refer to this as the before-after perspective on change [36], which particularly supports the risk assessment of planned system changes before they are implemented. In fact, the generalized approach presented in [38] also comes...
The target element related to this risk element has been changed.

Figure 5.4: Detecting changes

with language support for explicitly modeling the risk changes in it to assess and document how risks may emerge, disappear or evolve as a consequence of planned system changes. Using this approach for more dynamic changes at runtime may be less adequate. Another option from the CORAS tool portfolio is then the risk monitor [32] for continuously updating risk estimates by the monitoring of key indicators. An approach to this is presented in Chapter 8. However, the risk monitor requires all threats, threat scenarios, vulnerabilities, assets, etc. to be modeled in advance. Any system changes that yield inconsistency with these predefined risk models would require reassessments of the kind discussed in this chapter.

5.3.2 Examples

Figure 5.4 shows a screen shot of the tool with the drawing canvas in the middle, the palette to the left and the index table to represent the system model. Note that the index table is used only to store a representation of the system model in the tool; to view and edit the system model, a separate and designated tool is used. The CORAS threat model that is depicted shows results of the risk identification conducted on the eHealth case discussed in Section 5.1 and in deliverable D11.2 [9]. One threat scenario is that GP stores data on irregular media, such as his/her private smart phone or a USB memory stick, which in turn could lead to accidental leakage of EHRs and a confidentiality breach.

This threat scenario is related to the target segment GP desktop application. This means that a mapping rule has been specified in the trace model to link the threat scenario GP stores data on irregular media in the risk model to the GP desktop application of the system model to convey that the application does not prevent sensitive data to be stored outside the application. The mapping rule made use of index i1 shown at the bottom of the screen.

In this example, some system changes are implemented, including replacing the GP desktop application with a GP web application. In Figure 5.4, this change has already been implemented in the system model and imported to the tool; to see this, notice that GP desktop application (index i1) has mode before, whereas GP web application (index i5) has mode after. In other words, the former has been removed from
If the source has mode before, then the target cannot have mode after.

If the source has mode before, then the target cannot have mode after.

Figure 5.5: Detecting inconsistencies

and the latter has been introduced to the system.

This detection and specification of system changes is conducted automatically by the tool. Also automated is the flagging of changes. Based on the modes of the system model index and the trace model, the tool identifies the parts of the risk model that may be affected by change. For example, a warning has popped up on the mentioned threat scenario.

Figure 5.5 shows the initial step of updating this threat diagram to take changes into account. In this case the mode of the mentioned threat scenario has been set to before to specify that it has become obsolete after the introduction of the web application. Also the related target segment has been assigned mode before. This yields an inconsistency with the modes of preceding threat GP and the following threat scenario EHRs are accidentally disclosed to 3rd party. Such inconsistencies are automatically detected and flagged by the tool, as shown in Figure 5.5. As explained and exemplified in more details in the full paper [50], the tool supports a systematic resolution of inconsistencies by a stepwise reassessment of the affected risks. The tool moreover offers different views on the risk models to display the risk picture before and after the system changes, and to display the changes to the risks.

5.4 Conclusion

Orchestrating risk assessment with mainstream system engineering requires the respective models to be kept up to date and mutually consistent. This is particularly challenging for systems and system models that frequently change during the SDLC. In this chapter we have presented a tool that facilitates such orchestration by automated traceability from system models to risk models. The tool can support all the phases of our risk and cost aware SDLC, in particular to cope with the several iterations of the development life cycle.

To ensure general applicability, the tool is designed to accept any language or notation for the modeling of the target of analysis. As a model-driven approach to security risk analysis, the method and tool is of course most suitable for model-driven approaches to secure service engineering. In the paper [50] we demonstrate the integration with security requirements engineering using SI* as system models. Considering the activities of the RC-SDL depicted in Figure 2.3, the tool is perhaps most suitable to support the business modeling, the requirements engineering and the analysis & design. However, improving security by means of risk assessment in any activity of the iterative SDLC should obviously improve the security considerations overall.
6 Identification of Cost-Effective Risk Treatments

In this chapter we present an approach to identify and select cost-efficient treatments for unacceptable risks. The approach consists of a process for conducting the analysis of treatment cost and effect, and is facilitated by modeling support and a calculus with rules for reasoning about cost and effect.

6.1 Introduction

Risk management and security risk assessment are conducted by organizations and stakeholders to increase the awareness of potential risks and to keep risks at an acceptable level. As part of the risk management process, treatments to mitigate risks are identified, the implementation of which should ensure a tolerable residual risk level. However, the decision of whether to implement possible treatments is not only a matter of their mitigating effect; the cost needs to be justifiable and demonstrable.

The approach targets the risk treatment phase of the risk management process depicted in Figure 2.1 on page 20. In particular, the process takes as input risk models that document identified risks, their likelihoods, and their consequences with respect to identified assets. The approach moreover makes use of available methods and techniques for the estimation of the costs and benefits of risk treatments, i.e. their quantification from a business-oriented perspective. Such a method is presented in Chapter 5 of NESSoS deliverable D10.2 [10].

Our approach to the identification of cost-effective risk treatments builds on the formal foundation for a risk and cost aware SDLC presented in Chapter 8 of D10.2. In particular, the modeling and reasoning about treatments is formalized by means of risk graphs. A risk graph can be understood as a common abstraction of several established risk modeling techniques [6], such as fault trees [18], event trees [19], attack trees [49], cause-consequence diagrams [40], Bayesian networks [4] and CORAS threat diagrams [37]. Hence, our techniques for modeling and assessing treatment cost and effect comply with these and can be instantiated by them. In this chapter we demonstrate this by exemplifying the instantiation in CORAS [37]. The example is based on a case study from the eHealth domain and as described in NESSoS WP11 deliverable D11.2 [9].

The structure of this chapter is as follows. In Section 6.2 we give an overview of the process, the underlying concepts and the model artifacts that are used. In Section 6.3 we present the modeling support, and in Section 6.4 we present the calculus for reasoning about the treatments and their cost and effect. In Section 6.5 we give a concrete example by instantiating the approach in CORAS, before concluding in Section 6.6. The reader is referred to the technical report [56] for the full details of the contribution, including the full calculus and discussion of related work.

6.2 Process

In this section we give a brief overview of the process and the activities that are conducted in using the approach. The process involves three steps as shown in Table 6.1. The input to the process is a risk model documenting unwanted incidents together with their causes and consequences. The output is an overview of recommended treatment alternatives, where each alternative is a subset of the identified treatments documented with its accumulated cost and mitigation effect.

Figure 6.1 shows the underlying concepts on which our approach is based. During Step 1 of the process, the risk model is annotated with the information that is needed to document and reason about identified treatments. A countermeasure is a treatment, i.e. a means to mitigate risk by reducing the likelihood and/or consequence of one or more risks. The cost includes the cost of countermeasure implementation and the loss due to residual risks. The reduction effect is the estimate of the mitigating effect of each identified countermeasure in terms of reduction of likelihood and/or consequence. The effect dependency captures the possible countervailing effect among countermeasures that must be taken into account in order to understand the combined effect of identified treatments. The calculus is a set of rules for reasoning about the annotated risk model, and is defined as an extension of the calculus presented in deliverable D10.2 [10] as part of our formal foundation for the risk and cost aware SDLC. Using the calculus we can perform cost-benefit analyses of implementing selected subsets of the identified countermeasures. The
### Table 6.1: Treatment analysis process

<table>
<thead>
<tr>
<th>Input</th>
<th>Risk model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Annotate risk model</td>
<td>Document treatments, treatment cost and effect and countervailing effects among treatments by annotating the risk model</td>
</tr>
<tr>
<td>Step 2: Conduct cost-benefit analysis</td>
<td>Identify treatment alternatives and evaluate the cost-benefit for each of them by applying the calculus</td>
</tr>
<tr>
<td>Step 3: Conduct synergy analysis</td>
<td>Analyze synergies for selected risks based on decision diagrams to identify recommended treatment alternatives</td>
</tr>
<tr>
<td>Output: Treatment alternatives</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 6.1: Conceptual model

A decision diagram facilitates the decision making process of managers or other stakeholders, and is based on the cost-benefit analysis.

#### 6.3 Modeling Support

As explained above, the process takes a risk model resulting from a risk analysis as input. To facilitate generality, i.e. to ensure that our approach is compatible with several risk modeling techniques, we assume that the risk model can be understood as an instantiation of a risk graph.

A risk graph, as exemplified in Figure 6.2, consists of a finite set of vertices and a finite set of relations between them. Each vertex represents a threat scenario or an unwanted incident, and can be assigned a likelihood, \( lh \), and—if it is an unwanted incident—a consequence, \( co \). A leads-to relation from \( v_1 \) to \( v_2 \) means that the former may lead to the latter. Likelihoods on the relations are conditional likelihoods indicating the likelihood of the former leading to the latter when the former occurs.

To support the annotation of risk graphs during Step 1 with the necessary information, we have extended the risk graph notation as defined in deliverable D10.2. As shown in Figure 6.3, countermeasures, \( cm \), are represented by rectangles annotated with their expenditure, \( e \). The expenditure is the estimated cost for implementing and maintaining the countermeasure over a defined time period. The treats relations from a countermeasure to threat scenarios specify which scenarios are mitigated by the countermeasure. The mitigating effect of a countermeasure on a threat scenario is captured by the annotation \( lr \) and \( cr \) for likelihood and consequence reduction, respectively. Each of them are expressed in relative percentage value. Note that in some cases these values can be negative as countermeasures may have a negative effect on some scenarios or incidents. Figure 6.4 shows the specification of effect dependencies to capture possible combined effects of several countermeasures. Such a dependency is modeled as an effects relation from one countermeasure to a treats relation of another. The relation is annotated with a pair of effect on likelihood reduction \( elr \) and effect on consequence reduction \( clr \). Both are expressed in terms of
relative percentage value, and can be positive or negative.

6.4 Calculus

The cost-benefit analysis during Step 2 uses the annotated risk model as input and is supported by the calculus. The analysis is conducted for each individual risk to evaluate the cost-benefit balance of selected countermeasure alternatives. Each such alternative is a subset of the identified countermeasures. Eventually, the result of the different alternatives is visualized in a decision diagram.

The calculus consists of a set of rules for calculating the effects of countermeasures, and how the effects are propagated through the risk diagrams. We refer to the report [56] for the full calculus, and give only the following example to explain how it is used.

Rule 6.1 (Countermeasure) If there is a treats relation from countermeasure \( cm \) to vertex \( v(lh, co) \) with likelihood reduction \( lr \) and consequence reduction \( cr \), we have:

\[
\frac{cm \rightarrow lr, cr \cdot v(lh, co)}{v(lh \cdot lr, co \cdot cr)}
\]

Rule 6.1 applies to countermeasures as depicted in Figure 6.3. The likelihood reduction \( lr \) on the likelihood \( lh \) of the scenario means that \( lh \) is reduced by \( lr \in [0, 1] \). Hence, \( lh \) is multiplied by \( lr = 1 - lr \). Likewise for the consequence reduction. For each scenario \( v \), the rule is applied for each countermeasure that applies to \( v \) to yield the eventual result. The calculus also comes with rules for resolving effect dependencies as captured by Figure 6.4. When combined with the other rules of our formal foundation (see D10.2), we can propagate the countermeasure effects through the risk diagrams.

6.5 Example – CORAS Instantiation

In this section we exemplify our approach in the CORAS instantiation by using a concrete scenario. The scenario we consider is on patient monitoring as described in NESSoS deliverable D11.2 [9]. The reader is also referred to Figure 5.1 in Chapter 5 where the same use case is described. The diagram we show is only a small snip of a larger case study. For further details and examples, see also the report [56].

Figure 6.5 shows a CORAS treatment diagram depicting some of the results of Step 1 of our process, namely the annotation of the risk model with countermeasures (referred to as treatments in CORAS).
Figure 6.5: CORAS instantiation

costs, reduction effects and effect dependencies. To enable these annotations we have extended the CORAS language. For a brief introduction to the CORAS notation, the reader is referred to Chapter 5.

The diagram in Figure 6.5 shows results of the risk identification and risk estimation. The likelihoods and consequences are hence before the treatments are taken into account. For example, the threat scenario Transmission of monitored data is interrupted occurs with a frequency of 33 per 10 years. The risk Loss of monitored data occurs over 26 times per 10 years with an estimated consequence of $500 for each occurrence.

Consider now the threat scenario Network connection goes down. The current likelihood of this scenario is 30 : 10y. There are two treatments to mitigate this scenario (and hence the risk Loss of monitored data), namely Implement redundant network connection and Ensure sufficient QoS from network provider. The former has an estimated cost of $5000 per 10 years and the latter the cost $15000. Both have the likelihood reduction effect of 0.7. However, when both treatments are implemented, it is estimated that the latter will reduce the effect of the former by 0.2.

By applying the effect dependency rule, the effect of the former is first updated to 0.5. We can then apply the countermeasure rule, i.e. Rule 6.1, for each treatment in turn. The former treatment yields the updated likelihood 15 : 10y for the threat scenario, and the latter in turn yields 4.5 : 10y. Proceeding from here we can calculate the treatment effects for the whole CORAS diagram by using the full calculus and by considering different subsets of the identified treatments.

We refer to each such subset of treatments as a treatment alternative. During Step 2 of our process all such alternatives are enumerated, and the cost-benefit balance for each of them is evaluated to understand the different treatment options and their combined cost and effect.

We use decision diagrams to facilitate the documentation and evaluation of the different treatment options. The decision diagram for the risk Loss of monitored data is depicted in Figure 6.6. Each node represents the risk with one treatment alternative implemented. The node S0 is the initial situation, i.e. before any treatment is considered. What the diagram shows is that treatment alternative S2 and S3 are the most favorable. The former is the alternative of using only the treatment Implement redundant network connection, where as the latter is to additionally implement the treatment Keep available backup device. The decision diagram also shows that it is not recommended to implement all the identified treatments together (S7), due to the high total cost and the countervailing effects. In fact, it is only S2 and S3 that are more beneficial than not implementing any treatment at all (S0).

Due to space constraints we will not exemplify Step 3 of the analysis here, which is the synergy analysis. Whereas the objective of Step 2 is to evaluate the cost-benefit with respect to individual risks, Step 3 aggregates the results for all risks in combination to yield the final outcome of a recommended countermeasure alternative for the whole risk analysis. For this purpose a global decision diagram, similar to the one depicted in Figure 6.6, is created for showing the different treatment alternatives for all risks together.
6.6 Conclusion

In this chapter we have presented an approach to conduct a systematic identification of security risk treatments with support for assessing and analyzing cost and benefit. The approach is based on our formal foundation for a risk and cost aware SDLC as presented in D10.2, and comes with modeling support and a calculus for reasoning about the treatments. The process facilitates the final treatment activity of the risk analysis process defined by the ISO 31000 standard, and the objective is to support stakeholders and managers in selecting optimal treatment alternatives for unacceptable risks.

Because our approach extends the standard risk analysis process, it has the potential to support cost effective security risk treatment in all phases of the SLDC that already uses risk assessment as a basis for secure software and service development. For example, during the business modeling and security requirements engineering, the identification of risk treatments that are justifiable from a business oriented perspective will substantiate the ROI in security already during these initial phases. At the later security design phases where services are specified at a more detailed and low-level, new iterations of risk and cost assessment can give better estimates and a more solid justification. The approach is also useful and adequate during deployment to maintain an ISMS by reviewing, maintaining and improving information security based on a business risk approach [23].
7 Cost-Effective Run-Time Enforcement

In this chapter we consider cost-effective run-time enforcement. Our goal is to consider the problems of decision making in dynamic systems, when enforcement of security policies is affected by the mutable nature of required parameters. Since the Future Internet becomes more dynamic and distributed (Web Service, Clouds, Grid, sensor networks, etc.) the considered problems become more and more important. Thus, we show how risk could help to improve the quality of protection at the run-time phase of SDLC, when the implemented system is deployed and managed by users. In this work we focus on decision making in access and usage control systems.

Access control aims to ensure that only trusted principals are granted access to a resource [1]. Usage control [57] is responsible for guaranteeing that principals also remain trusted when the access is in progress, i.e. when these principals use the resource. The reference monitor evaluates an access decision on the basis of the principal's attributes. The attributes are issued by the attribute provider and characterize subjects and objects participating in access and usage control [47, 48].

Some security attributes (e.g., the requester's reputation and location) are remote, i.e. the attributes reside outside the control of the reference monitor, and can be only observed. The system usually allows only the current attribute value to be received, and, as a result, some attribute changes between adjacent queries might be missed. Worse still, these unnoticed changes may violate security policies. For example, if a security policy grants access rights to the users residing in a certain location, there is no evidence that mobile users did not leave the location in-between checks [11].

In addition, system failures, delays occurring during attribute delivery due to network latency, as well as malicious activities (e.g., a man-in-the-middle, eavesdropping and impersonating of data by the attribute provider) contribute to the problem of correct policy enforcement. The impact of uncertainties associated with observed attributes should be mitigated by the reference monitor [29, 41].

Our current work extends the results obtained in D10.2 [10] and proposes the cost-effective enforcement models of UCON AC [57] security policies. In our model, an uncertainty-aware reference monitor should adjust its decisions according to the information about the relevant uncertainty. We propose to consider cost-effectiveness (risk) as the main criterion for making such a decision. In the current work we compare two models for decision making and compare them using risk as the main criteria. Moreover, we consider a new problem of cost-effective attribute retrieval, which is specific for UCON. We refer the reader interested in formal description of the problems, proofs and details of experiments to our paper [28].

7.1 Policy Enforcement Under Uncertainties

In the model we consider there are two main actors: a reference monitor (RM), which collects required values of attribute and makes a decision, and an attribute provider (AP), which provides the values. Naturally, an attribute provider has real values of attribute while a reference monitor has only the values received from the attribute provider, called observed attributes. The observed attributes differ from real attributes because of a number of existing uncertainties:

**Freshness I (non-continuous checks)** corresponds to the scenarios where only a part of attribute changes can be detected because the checks are carried out through some time interval.

**Freshness II (delays in processing)** implies that there are inevitable time delays in delivery of an attribute value and decision making.

**Freshness III (pending updates)** corresponds to scenarios where the current attribute value is uncertain since some update queries are pending.

**Correctness** is affected by additive noises that usually exist in case of non-accurate measurements.

**Trustworthiness** appears as a result of the attribute provider altering attributes or as a result of attacks during attribute delivery, storage, etc.
Correct enforcement is not feasible in the presence of uncertainties since the reference monitor is unable to show that real attribute values satisfy a policy. The basic idea of the policy enforcement of access control under uncertainties is:

1. The reference monitor evaluates the policy with respect to observed attribute values.
2. If the observed values satisfy the policy, the reference monitor runs an experiment which estimates to what extent the observed attributes vary from the real ones. If this difference is negligible, the experiment succeeds and the reference monitor allows the access.

Exploiting this idea we specify two possible models of behavior for reference monitor. Both models assume that the reference monitor is able to compute the probability that the real value violates the policy using the latest possible observed value ($Pr_{RM}$).

The reference monitor computes $Pr_{RM}$ using the following data:

1. Observed values of the attribute.
2. Parameters of a stochastic process that models a real behavior of an attribute.
3. A list of uncertainties presented in the system.

Possible combinations of the last two factors produce a variety of techniques on how to compute $Pr_{RM}$. As an example, we refer the reader to [26, 25] where the behavior of an attribute is modeled as a Markov chain and freshness uncertainties exist in the system. Another example given in [5] studies a static attribute (i.e. the attribute does not change its value over time) in the presence of the trustworthiness uncertainty. In our running example for access control we compute $Pr_{RM}$ considering only freshness III uncertainty and model the attribute behavior as a discrete-time Markov chain. The probability for correctness uncertainty may be computed using the information about error distributions available for the concrete method of measurement. For trustworthiness case, approaches which consider trust as a probability that an interaction will succeed or fail can be used for analysis the probability for static attributes, i.e., the fact which can be either true or false (e.g., [51, 13]).

**Definition 7.1** Threshold Enforcement of Access Control. The reference monitor computes $Pr_{RM}$ and grants access if: 1) The observed value satisfies the policy; 2) The computed probability is greater than a threshold.

**Definition 7.2** Flip Coin Enforcement of Access Control. The reference monitor computes $Pr_{RM}$ and grants access if: 1) The observed value satisfies the policy; 2) The random experiment grants access with the probability $Pr_{RM}$.

We would now like to estimate the cost-effectiveness of the proposed enforcement methods. Our goal is to find the expected profit $\langle C \rangle$ for enforcement of access control. We have four scenarios (events) of how the reference monitor acts under uncertainties. Every scenario results in gains or losses for reference monitor and has its probability in every experiment ($Pr$):

- **True positive**: grant access when policy holds ($C_{tp}$).
- **False negative**: grant access when policy is violated ($C_{fn}$).
- **False positive**: deny access when policy holds ($C_{fp}$).
- **True negative**: deny access when policy is violated ($C_{tn}$).

Finally, let $C_a$ be the cost to push/pull (observe) an attribute value. Now we are able to define the formulas for average cost for the proposed models. The average cost for threshold model is computed aggregating possible benefits and risks:

$$\langle C \rangle_{th} = C_a + \begin{cases} Pr_{RM} \cdot (C_{tp} - C_{fn}) + C_{fn} & \text{if } Pr_{RM} \geq th \\ Pr_{RM} \cdot C_{fp} + C_{tn} \cdot (1 - Pr_{RM}) & \text{otherwise} \end{cases}$$ (7.1)
Moreover, we are able to find the optimal threshold value, which always allows us to make the most profitable, i.e., less risky, decision. The threshold probability is:

$$\text{th} = \frac{C_{fn} - C_{tn}}{C_{fp} + C_{fn} - C_{tn} - C_{tp}}$$  \hfill (7.2)

The average cost for flip coin model is:

$$\langle C \rangle_{\text{flip}} = C_{tp} \cdot \Pr_{RM}^2 + (C_{fp} + C_{fn}) \cdot \Pr_{RM} \cdot (1 - \Pr_{RM}) + C_{tn} \cdot (1 - \Pr_{RM})^2 + C_a$$  \hfill (7.3)

Now we are able to prove [28] that threshold model is always better than the flip coin one.

**Proposition 1** Threshold strategy is more cost-effective than flip-coin, except the points $\Pr_{RM} = 0$, $\Pr_{RM} = 1$, and $\Pr_{RM} = \text{th}$, where the strategies are equal: $\langle C \rangle_{\text{th}} \geq \langle C \rangle_{\text{flip}}$.

Our theoretical findings are confirmed by our simulations. We created a model for simulation of reputation change, which has four values: “general”, “normal”, “suspicious”, and “malicious”. Only the last value violates the policy. We used Discrete Markov Chain to model possible changes of the reputation. The observed value for the reference monitor was “normal” (state 2). The following costs were selected: $C_{tp} = 10, C_{fn} = -15, C_{fp} = -1, C_{tn} = 0$ and to query an attribute we pay $C_a = -2$. $C_{tp}$ is a positive cost, since we do a correct decision, while $C_{fn}$ and $C_{fp}$ are negative, since the decision is incorrect. We put $C_{tn}$ to 0 since catching a violation usually does not result in gain or loss.

Figure 7.1 shows the results obtained. Since we use discrete model for time passed after getting new value the x axis is also discrete. The average profit per access request for the correct enforcement is always higher. The decline of the correct curve occurs because while the delay increases the probability that the received value would fail the policy also increases. Since the attribute cannot get a bad value in $m = 0$ or $m = 1$ steps (starting from state 2) all three curves have the same maximal value in these cases. The flip coin enforcement shows the worse results with respect to the threshold enforcement which tallies with our theoretical findings.

### 7.2 Attribute Retrieval Under Uncertainties

Now we consider another problem provoked by Freshness I uncertainty, which is more relevant to UCON, when the decisions must be made continuously during the usage of data. The main problem we would like to solve now is when we have to perform next check in order to maximize our benefits.
Cost of Usage Session in Case of Periodic Checks We start with a cost gained from the enforcement of a particular usage session. There are three costs for usage control: (i) \( c_{tp} \) - the gain per atomic interval of time when all changes of real attributes satisfy the policy; (ii) \( c_{fn} \) - the cost per atomic interval of time when the policy fails; and (iii) \( C_u \) - the cost paid for the attribute retrieval and the re-evaluation of access decision.

Let \( q \) be a number of attribute changes on a check. A cost \( C_s \) of a particular usage session depends on the time \( \tau_q \) when an attribute satisfies a policy, on the time \( \tau_b \) when the attribute violates the policy, and a number of checks \( n \):

\[
C_s = c_{tp} \cdot \tau_q + c_{fn} \cdot \tau_b + C_u \cdot (n + 1)
\]  

(7.4)

Let \( \theta(x) \) be a function such that

\[
\theta(x) = \begin{cases} 
1 & \text{if attribute } A_x \text{ violates the policy} \\
0 & \text{otherwise}
\end{cases}
\]

Let \( cl_{AP}(i) \) be a function which returns the time when attribute \( A_i \) was received by the attribute provider. Similarly, \( cl_{RM}(i) \) returns the time when attribute \( A_i \) was received by the reference monitor.

Then, \( \tau_q \) and \( \tau_b \) are given by

\[
\tau_q = \sum_{j=0}^{l-1} (cl_{AP}(j + 1) - cl_{AP}(j)) \cdot \theta(j)
\]

(7.5)

\[
\tau_b = cl_{RM}(n) - cl_{RM}(0) - \tau_q
\]

(7.6)

Consider the following example. An attribute provider observes six changes for a session: at \( cl_{AP}(0) = 0 \) \( A_0 = normal \), at \( cl_{AP}(1) = 5 \) \( A_1 = suspicious \), at \( cl_{AP}(2) = 8 \) \( A_2 = malicious \), at \( cl_{AP}(3) = 13 \) \( A_3 = suspicious \), at \( cl_{AP}(4) = 14 \) \( A_4 = normal \), at \( cl_{AP}(5) = 22 \) \( A_5 = general \), at \( cl_{AP}(6) = 25 \) \( A_6 = normal \).

Reference monitor gets values every 3 steps: \( cl_{RM}(0) = cl_{RM}(1), cl_{RM}(2) = cl_{RM}(1), cl_{RM}(3) = cl_{RM}(1) \).

Now, applying Equations 7.5 and 7.6 we get \( \tau_q = 5 + 3 + 1 + 8 + 3 = 20 \) and \( \tau_b = 25 - 0 - 20 = 5 \).

In fact, \( s \) is a random event and let \( Pr[s] \) denote the probability of \( s \) occurs. Thus, the average cost of usage control enforcement will be a sum over every possible cost weighted by the probability of \( s \):

\[
\langle C \rangle_q = \sum_{s \in S} Pr[s] \cdot C_s
\]

(7.7)

where \( S \) contains all possible sample sequences associated with usage sessions enforced under uncertainties. \( Pr[s] \) could be found in different ways, depending on the model used for computation. In particular, using a Discrete Markov Model of an attribute, we can compute the probability using the transition matrix and taking into account that all states after \( q \) steps must be good states, while only the last step violates the policy.

Cost-effective enforcement implies that the reference monitor should choose such \( q \) that maximizes profit: \( \text{arg max}_q\langle C \rangle_q \).

Cost of Usage Session in Case of Aperiodic Checks In case of aperiodic checks, a number of attribute changes occurred on each check is different. There is a set \( Q = \{q_1, q_2, ..., q_n\} \) and each \( q_i \) tells how many attribute changes happened on the \( i \)-th check. All formulas given for periodic checks are valid for aperiodic. Only a number of attribute changes is different. We also use \( \langle C \rangle_Q \) to denote the average cost of the usage control enforcement under aperiodic checks.

Proposition 2 Aperiodic checks are at least as good as periodic checks in terms of cost-effectiveness:

\[
\langle C \rangle_Q \geq \langle C \rangle_q
\]

In our example, for aperiodic checks the computations of \( Pr[s] \) are similar to periodic ones above. However, since the reputation is modeled as a Markov chain, the probabilistic behavior of the reputation significantly depends on the current state of the random process. Therefore, we compute \( q \) for every state in the Markov chain.
We performed several simulations to check the values provided by our theoretical equation. To evaluate the aperiodic checks we carried out an exhaustive search of the optimal lengths of intervals between checks and found the values $q_1 = 7$, $q_2 = 4$, and $q_3 = 1$ if the current observed value is “general”, “normal”, and “suspicious” respectively. The computations of $q_i$ are only required ones the policy is deployed in the system.

The results of the simulations are shown in Figure 7.2. Since there is no single interval for aperiodic checks, we display aperiodic checks as a straight line.

First, both periodic and aperiodic checks are close enough to the theoretical curves. Second, the simulations illustrate our proposition regarding the fact that aperiodic checks are at least as cost-effective as periodic ones. In our example, aperiodic checks are about 15% more cost-effective than periodic checks. Third, the analysis of the periodic checks shows that the average cost of the session has the maximum value when the interval between checks is 4. The smaller interval is ineffective because we pay more for requesting an attribute. The bigger intervals are ineffective, because the system misses more policy violations.

### 7.3 Conclusion

The methods proposed in this chapter were developed in order to address new challenges raised by the dynamic and distributed Future Internet. The first method allows making decisions even when attribute values at hands are uncertain by taking into account risk of making wrong decisions. The second method accepts that FI tends to have long-lived sessions and proposes a solution for selection of the most cost-effective attribute retrieval policy. The proposed methods show how risk can be used during the run-time part of SDLC, when software/service is implemented and has to make cost-effective decisions in the changing environment. Finally, our next step is to implement the proposed methods in a real UCON system.
8 Dynamic Risk Assessment by Indicator Monitoring

In this chapter we present a method for designing indicators to support the monitoring and dynamic assessment of security risks. The method supports both the identification of relevant indicators and the evaluation of their validity with respect to the specified objective.

8.1 Introduction

Future Internet (FI) software services and systems facilitate business environments in which companies, enterprises and organizations can cooperate across organizational borders. Such cooperation may involve the sharing or outsourcing of ICT-supported business processes, as well as the provisioning and consumption of different services by different actors. In this setting, the information that is stored, processed and exchanged by the systems is an asset in its own that requires appropriate and adequate protection so as to establish and maintain information security [23]. Security risk assessments may be conducted on a regular basis to determine whether information assets are sufficiently protected. However, due to the dynamic nature of FI services and systems, also security risks are highly dynamic. As thorough risk analyses are costly processes, conducting these at a very frequent basis is rarely a viable option. Instead, companies can benefit from business process intelligence applications monitoring and analyzing different aspects of a business, information system or service system and its underlying processes. For this to be possible and for the monitoring to be adequate, the monitored indicators must be valid, meaning that they correctly indicate or measure the degree to which a specific objective is fulfilled.

In this chapter we present a method called ValidKI (Valid Key Indicators) for designing indicators to monitor the fulfillment of business objectives with particular focus on information security risks and ICT-supported indicator monitoring. The reader is referred to the journal paper [34] for the full details of the contribution. The paper also presents a large example from the health domain and the use of electronic health records (EHRs) to illustrate the method.

The structure of this chapter is as follows. In Section 8.2 we introduce the underlying terminology and give their definitions, and in Section 8.3 we give an overview of the ValidKI method, the process that is conducted and the model artifacts that are produced. In Section 8.4 we conclude.

8.2 Basic Terminology and Definitions

An indicator may be defined “something that provides a clue to a matter of larger significance or makes perceptible a trend or phenomenon that is not immediately detectable” [16]. Indicators are closely related to metrics, which may be defined as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute,” while an attribute is “the specific characteristic of the entity being measured” [22]. For example, an unexpected rise in the traffic load of a web server may signal a denial of service attack in progress. In this case availability is an example of an attribute, and an availability metric may serve as an indicator for denial of service attacks. Indicators can be used to measure to what extent a company or an organization fulfills its business objectives, in which case we speak of key indicators. Some business objectives may focus on business performance, while others may focus on risk or compliance with laws and regulations.

The UML class diagram in Figure 8.1 relates the main artifacts used by ValidKI. One or more key indicators are used to measure to what extent a business objective is fulfilled with respect to a relevant part of the business. Each key indicator is calculated based on data provided by one or more sensors. The sensors gather data from the relevant part of the business. A sensor may gather data for more than one key indicator.

As illustrated in Figure 8.2, performing the steps of ValidKI results in nine different models/descriptions, each of which describes one of the artifacts of Figure 8.1 from a certain perspective. Business objectives are typically expressed at an enterprise level and in such a way that they can easily be understood by,
for example, executives, board members, partners, etc. It is therefore often not completely clear what it means to fulfill them. This motivates the need to capture each business objective more precisely.

The fulfillment of a precise business objective may be affected by a number of risks. We therefore conduct a risk analysis to capture risks related to the fulfillment of the precise business objective. To evaluate which risks that are acceptable and not acceptable with respect to the fulfillment of the precise business objective, we specify risk acceptance criteria; it is the risks that are not acceptable that we need to monitor.

The degree of fulfillment of a precise business objective is measured by a set of key indicators. To measure its degree of fulfillment there is a need to express each precise business objective in terms of key indicators. We refer to this reformulation as the reformulated precise business objective. Moreover, the correctness of key indicators may be impaired if they are not implemented correctly. This may again lead to new unacceptable risks that affect the fulfillment of the precise business objective. Since the reformulated precise business objective is the precise business objective expressed in terms of key indicators, we need to analyze risks to the correctness of the reformulated precise business objective.

The computation of key indicators relies on different kinds of data. To collect the data, sensors need to be deployed in the relevant part of business. Thus, there is a need to specify the deployment of different sensors. For each key indicator we distinguish between two specifications, namely the key indicator requirements specification and the key indicator design specification. The former captures requirements to a key indicator with respect to the sensor deployment specifications, while the latter defines how the key indicator should be calculated.

Considering the ValidKI artifacts and the models to describe them, there is still the issue of validity. Validation may be defined as “confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled” [21]. Since an indicator is basically a metric that can be compared to a baseline/expected result, the field of metric validation is highly relevant. Following [39] we define a set of key indicators to be valid with respect to a business objective if it is valid in the following two ways. 1) Internal validity: The precise business objective expressed in terms of the key indicators correctly measures the degree to which the business objective is fulfilled; 2) Construct validity: The gathering of the sensor measurements of each key indicator is suitable with respect to its requirements specification.

8.3 The ValidKI Method

Figure 8.3 gives an overview of the process that is conducted when using the ValidKI method. The process takes as input a business objective and delivers a set of key indicators together with a report substantiating its validity with respect to the business objective. Note that in practice there will typically be
Input: A business objective

<table>
<thead>
<tr>
<th>Step 1 Establish the target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1.1 Express the business objective more precisely</td>
</tr>
<tr>
<td>Step 1.2 Describe the relevant parts of the business</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2 Identify risks to fulfillment of the business objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2.1 Specify the risk acceptance criteria</td>
</tr>
<tr>
<td>Step 2.2 Risk identification and estimation</td>
</tr>
<tr>
<td>Step 2.3 Risk evaluation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3 Identify key indicators to monitor risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 3.1 Deploy sensors to monitor risks</td>
</tr>
<tr>
<td>Step 3.2 Specify requirements to key indicators wrt deployed sensors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 4 Evaluate internal validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 4.1 Express business objective in terms of key indicators</td>
</tr>
<tr>
<td>Step 4.2 Evaluate criteria for internal validity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 5 Specify key indicator designs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Step 6 Evaluate construct validity</th>
</tr>
</thead>
</table>

Output: A set of key indicators and a report arguing its validity with respect to the business objective received as input

Figure 8.3: Overview of ValidKI

A set of business objectives as input, but we describe the process with respect to only one. The process is conducted in an iterative manner where one or more of the steps may be conducted more than once. In the following we describe the steps in more detail.

**Step 1 – Establish target:** The first step can be understood as a specialization of the first step of the risk analysis process as depicted in Figure 2.1 on page 20. The concern is to understand exactly what the business objective means and acquiring the necessary understanding of the relevant part of business for which the business objective has been formulated. We distinguish between two sub-steps. In the first sub-step we characterize the business objective more precisely by formulating constraints that need to be fulfilled. In the second sub-step we specify the relevant part of the business.

**Step 2 – Identify risks to fulfillment of the business objective:** The second step can be understood as a specialization of step two, three and four of the risk analysis process depicted in Figure 2.1. The step is concerned with conducting a risk analysis to identify risks with respect to the fulfillment of the business objective. We distinguish between three sub-steps. In the first sub-step the risk acceptance criteria are specified. The criteria classify a risk as either acceptable or unacceptable based on its likelihood and consequence. In the second sub-step we identify how threats may initiate risks. We also identify vulnerabilities and threat scenarios leading up to the risks, and we estimate likelihood and consequence. In the third sub-step we evaluate the identified risks with respect to the specified risk acceptance criteria.

**Step 3 – Identify key indicators to monitor risks:** The third step of ValidKI is concerned with identifying key indicators to monitor the unacceptable risks identified in the previous step. We distinguish between two sub-steps. In the first sub-step we specify how sensors should be deployed in the relevant part of business. The key indicators that we identify are to be calculated based on data gathered by the sensors. In the second sub-step we specify our requirements to the key indicators with respect to the deployed sensors. The two sub-steps are typically conducted in parallel.

**Step 4 – Evaluate internal validity:** The fourth step of ValidKI is concerned with evaluating whether the set of key indicators is internally valid with respect to the business objective. We distinguish between two sub-steps. In the first sub-step we reformulate the precise business objective by expressing it in terms of the identified key indicators. This step serves as an introductory step in the evaluation of internal validity. In the second sub-step we evaluate whether the set of key indicators is internally valid.
valid by showing that the reformulated precise business objective from Step 4.1 correctly measures
the fulfillment of the precise business objective from Step 1.1. If the set is not internally valid, we
iterate by redoing Step 3.

Internal validity may be decomposed into a broad category of criteria [39]. The reader is referred to
[34] for the list of criteria that is used in ValidKI.

**Step 5 – Specify key indicator designs:** In the fifth step of ValidKI we specify the designs of the identi-
fied key indicators. Each design specifies how the key indicator should be calculated. The design
also shows how sensors, actors, and different components interact.

**Step 6 – Evaluate construct validity:** In the sixth step of ValidKI we evaluate whether the set of key
indicators has construct validity with respect to the business objective. As with internal validity,
construct validity may be decomposed into a broad category of criteria [39]. Again we refer to [34]
for the list of criteria that is used in ValidKI.

To evaluate the different criteria, we re-do the risk analysis from Step 2.2 with the precise business
objective replaced by the reformulated precise business objective, which is the precise business
objective expressed in terms of key indicators. For each key indicator we identify risks towards the
correctness of the reformulated precise business objective that are the result of threats to criteria for
construct validity that the key indicator needs to fulfill. If the risk analysis does not result in any new
unacceptable risks, we have established construct validity for each key indicator. If the set does not
have construct validity, we iterate. We will most likely be redoing Step 5, but it may also be the case
that we need to come up with new key indicators and new sensors. In that case, we redo Step 3. If
the set of key indicators is both internally valid and has construct validity with respect to the business
objective, then we have established that the set is valid.

As indicated in the description of the ValidKI process, the method supports the risk management ac-
tivities of context establishment and risk assessment, i.e. the first four phases of the risk management
process as defined by the ISO 31000 standard [20] and depicted in Figure 2.1 on page 20. The ValidKI
method is not limited to security risks, but it is clearly applicable also in the security domain by the iden-
tification of indicators to monitor security attributes such as availability and confidentiality. Considering
the risk and cost aware SDLC described in Chapter 2, ValidKI particularly covers the monitor and review
activities of security risk management in addition to the security risk assessment activities. The ValidKI
example case described in detail in the full paper [34] addresses security in an eHealth scenario, the
relevance of which in FI systems is discussed in NESSoS deliverable D11.2 [9].

### 8.4 Conclusion

In this chapter we have presented a method for designing indicators to support the monitoring and dynamic
assessment of security risks. Risk monitoring is particularly relevant for highly dynamic systems, such as
FI software services, in which security risks are also dynamic. Tool support for risk monitoring using
indicators is described in [32] where an architectural pattern for monitoring tools is presented. The paper
moreover gives a presentation of the CORAS risk monitor tool, which implements the pattern. The tool is
also described in the NESSoS CKB.

Considering the SDLC, risk assessment by indicator monitoring is most adequate during deployment.
However, the identification of relevant indicators can leverage security metrics identified and used for
security assurance for services (cf. WP9) during the development lifecycle.

An important part of the ValidKI method is the assessment of the validity of the key indicators that are
designed. To the best of our knowledge, there exists no other method for the design of valid key indicators
to monitor the fulfillment of business objectives. For related work on risk monitoring and metric validation,
the reader is referred to the full paper [34].
9 Interactions and Task Coverage

In this chapter we give an overview of ongoing and planned WP10 interactions and collaborations, the dissemination of foreground, and the coverage of the WP10 tasks as described in the NESSoS DoW.

9.1 Interactions

The methodology to support a risk and cost aware SDLC that is developed in W10 spans the orthogonal activities of WP6, WP7 and WP8. The risk and cost assessments will drive the SDLC by identifying potential security risks and by identifying options for risk mitigation that will ensure and demonstrate return on investment in security.

Considering the Security Requirements for Services of WP6, the outcomes from security requirements analysis can be used as input to the identification of several threat and risk factors that later need to be managed during the SDLC. Moreover, during a security requirements analysis, one needs to consider the level of risk and cost that a system is exposed to, as well as cost efficient options for mitigating unacceptable risks. One ongoing collaboration with WP6 involves the orchestration of security requirements engineering and security risk assessment in the SDLC. In the orchestrated process, the two activities leverage each other in a model-driven approach; changes are propagated between the respective model artifacts to trigger activities and maintain their mutual consistency. In the paper [50] that gives the more comprehensive presentation of the tool introduced in Chapter 5 we show tool support for automatically propagating changes from SI* security requirement models to CORAS risk models; with a tighter orchestration and mapping between the respective models we envisage a stronger and bi-directional automation of the propagation.

With respect to the Secure Service Architecture and Design of WP7 we plan to explore the potential of using the pattern-based cloud analysis described in D7.3 to support the security risk assessment as prescribed by the ISO 27001 standard. In particular, the WP7 prototype will facilitate the initial context description that is needed to create an ISMS for cloud computing systems. The subsequent security risk assessment can benefit strongly from this by the identification of relevant information assets. Related to this is also a planned collaboration to use CORAS to support not only the security risk assessment to maintain an ISMS, but also the creation of documents that are required for ISO 27001 certification. A potential link to WP6 that we may explore is the complementary use of security requirements engineering to support ISMS as investigated in [3].

As to the Programming Environments for Secure and Composable Services of WP8, the run-time monitoring middleware is related to the support for risk monitoring developed in WP10. For example, if the monitoring and enforcement infrastructure for continuous usage control is only semi-trusted, risk monitoring can be integrated to use risk assessment as part of the basis for decision. On the other hand, the monitoring infrastructure of WP8 can be utilized to strengthen the risk assessment by providing risk metrics and indicators, and by providing a stronger basis for estimating probabilities of unwanted incidents.

The plan for interaction with Security Assurance for Services of WP9 includes the integration of the approach to cost-effective run-time enforcement of WP10 with the approach to run-time verification in WP9. The resulting framework for usage control enforcement should take into account existing work for adapting the XACML language and architecture for usage control, and also incorporate risk-based methods for making decisions under uncertainty. We moreover plan to utilize WP9 security metrics to support the reference monitor in making assessment as to whether the received attribute values are genuine or not.

In addition to the interactions between WP10 and the other technical WPs of NESSoS, WP10 is interacting with WP2—Integration of methodologies and tools in the WorkBench—and WP5—Integrated body of Knowledge—to integrate its methods and tools in the NESSoS workbench and knowledge body. Risk assessment tools currently integrated in the SDE include the CORAS tool and the PRRS tool. The former was integrated during Y1 as reported in NESSoS D2.2, whereas the latter was integrated during Y2 as reported in D2.3. A usage example of the former is presented in D2.3. These tools are moreover described in the NESSoS CBK. The CBK objects of relevance to WP10 also include the RIGER tool and the CORAS Risk Monitor, the CORAS method and risk modeling language, as well as the techniques for risk-aware
usage control. Further relevant knowledge objects can be explored by browsing the CBK entries within the knowledge area Risk and cost aware SDLC.

As already shown in this deliverable, WP10 makes extensive use of the Future Internet application scenarios developed in WP11. As reported in D11.3 and fully documented in a separate publication [44], the CORAS method, language and tool was applied to the patient monitoring scenario of the eHealth use case. CORAS was also one of the methods that was evaluated in the eRISE challenge by applying it to both the Smart Grid use case and the eHealth use case. The results of the eRISE challenge are reported in D11.3. Further plans for using WP10 methods and tools in the context of WP11 include applying both the framework for usage control enforcement and the RIGER tool to selected scenarios from the Smart Grid and eHealth use cases.

9.2 Dissemination of Foreground

The different methods, techniques and tools of the NESSoS WP10 portfolio support different phases and tasks of the risk and cost aware SDLC presented in Chapter 2. The work leverages and further develops existing approaches, including CORAS. As mentioned above, CORAS was applied to the patient monitoring scenario and the results are reported in in a separate publication [44]. Based on the evaluation and experiences, we are currently improving the approach by building a framework for identifying and assessing required quality characteristics [21] for patient monitoring services, focusing on security. In this work we will make use of our method for model-based assessment of system quality [45, 46], which was part of PhD research [43] partially supported by NESSoS.

In handling the dynamic and evolving nature of Future Internet services and systems we have developed specialized methodological and modeling support for risk assessment [38]. This method is in turn supported by the tool presented in Chapter 5. A more detailed presentation of this tool is given in a paper [50], and currently we are working on a journal paper to give a more comprehensive presentation and evaluation of this tool. The full report [56] on our method and techniques for identifying cost-effective risk treatments is currently under finalization, as is the paper that will be published.

Several recent publications are related to our work on risk aware usage control and cost-effective runtime enforcement. In [27] the problem of decision making under uncertainty is explicated and formalized, and in [26] the risk awareness is supported. In [25] the influence of attribute freshness is taken into account, and we show how usage control decisions can be made when several policy rules are considered at the same time. The journal paper [28] gives an overview of this approach by summarizing the ideas presented, and by providing a new method for solving the attribute retrieval problem. We moreover propose an architecture for the access and usage control enforcer.

Also our work on dynamic risk assessment by indicator monitoring addresses the problem of evolution and run-time assessment. The method for designing valid indicators presented in Chapter 8 summarizes the full presentation given in our journal paper [34], while our methods for indicator-based risk monitoring are presented in other papers [33, 35]. An architectural pattern for monitoring tools is presented in [32], where the CORAS Risk Monitor serves as a specific implementation to demonstrate its use.

A further NESSoS WP10 research strand that we will intensify over the next years is on risk-based security testing and test-based security risk analysis. Recent publications include two workshop papers [14, 15].

To give a concise overview of the publications discussed above, we list them in the following.


9.3 Task Coverage

As described in the DoW, there are three WP10 tasks for which research activities are planned for the second year of NESSoS:

• Task 10.2: Methodology for risk and cost aware SDLC

• Task 10.3: Run-time risk management

• Task 10.4: Integration of the risk and cost aware methodology
The methods and tools presented in this deliverable most heavily cover Task 10.2 and Task 10.3. In particular, Chapter 2 through Chapter 6 offer methods and tools to tackle challenges described under Task 10.2, both in terms of the overall process to support the risk and cost aware SDLC and in terms of specialized techniques and tools that are needed in the setting of secure development of Future Internet services. Chapter 7 and Chapter 8 mostly address Task 10.3 and the problems of run-time risk assessment based on monitoring. The contributions of the different chapters are, however, not strictly bisected with respect to these tasks; instead WP10 aims to offer a portfolio of artifacts that can be applied in different combinations to support different tasks and activities in the risk and cost aware SDLC.

Task 10.4 involves the integration of the methodology for managing risk and cost awareness within the full SDLC, leveraging on the assurance techniques and security metrics of WP9. The progress on this task is currently preliminary and rather implicit because this task was initiated only at M18. The ongoing work is described above, and will be intensified during year three of the project.
10 Conclusions

For enterprises, organizations and industry, risk management should be an integrated part of the business and organization processes to ensure and maintain a justified balance between realizing opportunities and minimizing vulnerabilities and losses [20]. Risk management is a continuous activity to manage the analysis, plan, implementation, control and monitoring of implemented measurements and policies. Traditionally, risk analysis—as a core part of the overall risk management—is in contrast conducted only at a regular basis with some time interval between. Hence, the risk analysis results and documentation only gives a static and temporary picture of the risks. For the information society of today, and with the emerging Future Internet software services and systems, this traditional approach to risk analysis and documentation is often not adequate. FI services and systems and their environments are highly compositional and dynamic, and the stakeholders are multiple and heterogeneous. Approaches to risk analysis need to accommodate to these characteristics, and they need to be supported by adequate methods, techniques and tools.

Furthermore, the risk management of FI services and systems needs to cope with possible threats and vulnerabilities with respect to information security. As discussed in the context of the application case studies of NESSoS WP11, FI systems involve a number of different information assets. Moreover, users and stakeholders are highly heterogeneous, and security requirements are diverse. Security properties are also very diverse, typically including confidentiality, integrity, availability, privacy and data protection, accountability, and more. There is therefore a need for industry and stakeholders to embed in the overall risk management a systematic approach to establish, implement and maintain information security and mitigating security risks [23].

NESSoS is targeting these challenges by aiming for the engineering of secure software services by addressing security concerns from the very beginning of the analysis and design process. To this end, WP10 is developing a risk and cost aware SDLC where security risk and cost assessments are part of the overall, iterative engineering process. The overall process for this SDLC is based on established standards and recommendations for security risk management and system development, and defines the different activities that need to be conducted. This process is in turn facilitated by the portfolio of artifacts that are being developed in the context of WP10, some of which are presented in this deliverable. These artifacts include methods, techniques, languages and tools to cope with the challenges imposed by a continuous and iterative development process in which services and environments are evolving, dynamic and compositional. The different artifacts target different challenges and support different activities of the overall process; they complement each other, and leverage each other during the SDLC.

Security risk analyses may, depending on the target of analysis and the level of abstraction, be quite technical, and so may the proposed security mechanisms and controls to mitigate the risks. For managers and end-users that often need to bear the costs, there is a need to always demonstrate the value of security solutions from a business-oriented perspective. WP10 is therefore developing methods for assessing the cost and benefit of security solutions so as to ensure return on investment in security. Engineering secure software services and systems by addressing security concerns from the very beginning, as envisioned in NESSoS, is beneficial from a security risk perspective, but it comes with a cost. In WP10 we deliver the means to assess this balance and to identify the security solutions that pay off.
References


A Tool-Supported Software Application Risk Assessment

Jan Stijohann  Jorge Cuellar

Keywords: Risk assessment, security testing, threat modeling, data flow diagrams, static analysis, dynamic analysis, binary analysis, software vulnerabilities, design recovery

A.1 Introduction

The goal of our research activities is to develop a process for continuous security risk assessment and risk reduction in highly iterative software development processes [8]. The process should leverage existing collections of threats and vulnerabilities, as well as methods and tools used in static and dynamic software analysis. We do not intend to replace human judgment with a fully automated process based on mathematical formulas. The process should rather describe and (tool-) support the information retrieval and the involved activities. It will still be the risk analysts’ task to evaluate what information is most important and to do the final interpretation. The process will, however, improve the quality of this judgment and make it more comprehensive and more transparent for all involved parties including developers, security testers, architects, and managers.

A.1.1 Procedure Summary

The process combines manual tasks with tool support whenever possible. It can be summarized as follows:

In a first step, the application business owner identifies security requirements, deduces corresponding business worst case scenarios and estimates the impact of the latter (step 1). Business worst case scenarios (BWCS) are short, informal, and non-technical descriptions of business consequences that may result from the violations of the security requirements. In addition, a security analyst determines non-technical likelihood factors such as the expected attacker type and the attractiveness of the target. Next, the analyst creates a security overview reflecting the application’s attack surface and security architecture (step 2). The required information is obtained via personal interviews with the developers but also through a tool-supported static and dynamic analysis of the application. The security overview, illustrated in form of an annotated data flow diagram (DFD), serves as a basis for an adapted threat modeling (step 3).

During this activity, the analyst examines which abstract threats such as information disclosure, denial of service or escalation of privilege of what application parts can lead to the previously defined BWCS. The identified technical threats are then prioritized according to their technical likelihood which is initially based on the information captured in the security overview. At this point, the analyst can perform an initial risk estimation for each BWCS (step 4) considering the business impact and the technical and non-technical likelihood of all associated technical threats. If the risk estimation yields unacceptable results, the analyst can modify the application to reduce the attack surface. Furthermore, he can select (step 5) and conduct (step 6) suitable security tests to better estimate the real likelihood of the concerned technical threats. Based on the kind of testing activities and their results, the analyst can re-estimate the technical likelihood of the corresponding technical threats. Further tests or modifications of the application may be performed until an acceptable risk level for all BWCS is achieved. Steps 2 to 6 may be redone at the end of a development iteration.

Figure A.1 summarizes how the process is integrated into an iterative development life cycle. Figure A.2 presents a graphical overview of the relations between the involved activities and artifacts. Figure A.5 gives concrete examples for all introduced terms. The following sections describe each step in more detail.
- Determine security requirements & business worst case scenarios (M)
- Develop, compile & run
- Capture security overview of the application (T)
- Generate annotated DFD (T)
- Identify vulnerability indicators (T)
- Translate business worst case scenarios to technical threats (M)
- (Re-) estimate technical likelihood of technical threats and prioritize them (M)
- Estimate risk for each business worst case scenario (M)
- If risk is too high:
  1.) Reduce attack surface (M) or
  2.) Improve estimation by choosing (T) and performing adequate tests.

Figure A.1: Tool-supported risk assessment in iterative SDLCs.
Figure A.2: Overview of the risk assessment process.
A.1.2 Integration into NESSoS Work Package 10

The risk and cost aware software development life cycle (RC-SDLC) developed in NESSoS work package 10 is based on the ISO 31000 risk management standard. It therefore supports various tasks and phases, illustrated in Figure 2.1. Our proposed process contributes to all three risk assessment phases as defined by the ISO 31000:

Establish the context  Step 1 and 2 cover the identification of assets and existing controls, respectively.

Identify risks  The identification of threats is done when defining the expected attacker type in step 1, and, on a more technical level, during the adapted threat modeling in step 3. Vulnerabilities are identified via practical security tests in step 6.

Estimate risks  The identification of consequences is done through the definition of the BWCS in step 1. Both assessment of incident likelihood and level of risk estimation are part step 4.

The risk evaluation is also implicitly covered, as step 4 includes the option to mitigate or further analyze the system if the risk is not yet acceptable.

A.2 Step 1: Business Risk Assessment

The business risk assessment is conducted by the application business owner. It is supposed to furnish the security requirements, the business worst case scenarios with associated impacts, and the non-technical likelihood factors.

Security Requirements

A security requirement is a non technical asset together with a security property. For example, consider the eHealth use case described in deliverable 11.2 and a radiology software system in a hospital. Reasonable security requirements could be:

- (Patient data, confidentiality)
- (MRT configuration, integrity)
- (MRT emergency stop, availability)

Business Worst Case Scenarios (BWCS)

Business worst case scenarios are short, informal, and non-technical descriptions of business consequences that may result from the violations of the security requirements. In the previously mentioned example, BWCS would be:

- Personal damage due to modified patient data
- Damage of the MRT device due to a corrupted configuration
- Personal damage due to a non-functioning emergency stop

Every BWCS has an associated estimated impact which is one of the primary factors used for the later risk estimation. Section A.5 lists all factors considered for the final risk estimation.

Non-Technical Likelihood Factors

The likelihood of a BWCS is composed of technical and non-technical factors. The first are determined for each technical threat, see Section A.4 for a definition, while the latter are mostly the same for a BWCS or even all BWCS. The non-technical likelihood includes the expected attacker's motivation, the resulting attractiveness of the target, and the attacker's skill level. Attackers are categorized in three different categories according to their skill level and resources: opportunity rider, advanced and professional attacker.
A.3 Step 2: Attack Surface Analysis and Quantification of Security Relevant Information

The goal of this step is to capture, structure, and comprehend the security relevant technical aspects of the software system under test. The resulting security overview is a crucial point for the proposed process as it connects the BWCS with technical threats, and the latter with concrete practical testing activities. This is the foundation for a transparent risk analysis and efficient risk reduction.

A.3.1 Desired Properties of a Security Overview

The next subsections discuss the desired properties that a security overview should have to be of greatest utility for the proposed process.

Documented

Business owners, system architects, developers, and security testers often have different conceptions of the same software system. A clearly documented security overview ensures that all involved parties have the same correct understanding. Furthermore, a documented security overview eases organization and realization of the risk analysis and the included security tests. Examples for such organizational activities are budgeting of resources, creation of separated tasks, and the estimation of progress.

Created by Risk Analysis Team

It is often not possible to use existing (design) documents for the security overview since these tend to be overwhelming, out-dated, incorrect, incomplete or non-existent. Most likely though, they miss security relevant information. It is therefore judicious let the risk analysis team create its own suitable security overview and extract, or "recover", the required information from more reliable sources. Reasonable alternatives are interviews with developers and architects, but first and foremost, tool-supported methods based on static and dynamic analysis of the application binaries.

Generated with Tool Support

Practical experience shows that it is time consuming and tedious to manually create a correct and sufficiently detailed security overview of a target application. This holds true especially for larger projects with numerous developers. Not only are these kinds of applications constantly changing, but often nobody has a complete security overview of the system. An automated approach can help to obtain and keep the overview. Moreover, the automatically gathered information could be used to discover discrepancies with existing design documents and the interview results.

Based on Static and Dynamic Binary Analysis

Data containing security relevant information can exist in many different formats such as UML, sketches, text, or source code. However, a tool-supported generation of the security overview requires machine readable and preferably uniform input data. Executable binary files meet both requirements: they are by nature machine readable, and all native, that is non-interpreted, applications running on the same operating system use the same format\(^1\). Further advantages are:

- All executable software has to exist in binary form - at one time or another\(^2\).
- Binaries will most likely be available for the risk analysis team. This is often not true for design documents or source code. Experience shows that especially the latter is often kept secret - even from security and risk analysts.

\(^1\)For example Portable Executable (PE) on Windows and Executable and Linkable Format (ELF) on Linux and most Unix.
\(^2\)Interpreted languages such as Java or .NET use "Just-in-time" compilation.
• Binaries are a very valuable and also reliable source as they represent the actual system as it is deployed and attacked.

First steps towards an automated system model generation for security purposes based on limited static analysis of binary files have been made in 2006 by Wang and Torr [1]. From a practical point of view, an additional dynamic analysis of the application seems to be a reasonable complement. It does not only require little effort but can also identify aspects that a purely static analysis could miss, for example, dynamic API calls or the impact on, and interactions with external modules and the operating system.

Syntactically Standardized

Application binaries provide a machine readable and uniform source of information for the security overview. Syntactically standardizing this overview has also several advantages:

1. It can be used for consistent complexity estimation. Possible metrics extracted from the security overview could be, for example, the number of all information items, the number of interfaces, or the number of data flows from source to sink.

2. The security overview can act as intermediate language that helps recognizing recurring problems in different applications.

3. The overview can be processed for further use. The adapted threat modeling described in Section A.4, for example, requires a Data Flow Diagram (DFD) of the application which can be generated from the security overview.

A.3.2 First Ideas for the Practical, Automated Information Extraction

There exist several kinds of tools which support the automated extraction of security relevant information. We intent to evaluate:

• Scriptable debuggers such as pydbg³, IDA Pro⁴ or Immunity Debugger⁵
• Out of the box dynamic analysis tools such as Procmon⁶, ProcessExplorer⁷ or WinApiOverride32⁸
• Operating system tools such as strace, ps, syslog, and windows event log

A.3.3 Information to Include in the Security Overview

The following list contains examples of security relevant information that a tool-supported preliminary analysis of the target software may capture.

• Data flow related aspects
  – (OS) Processes
  – Used libraries
  – Persistent data
  – Non persistent data
  – Data Flows
  – Interfaces
  – Trust Boundaries
  – Local Boundaries

  – Sensitive information (encryption keys, authentication credentials, log files, configuration files)

• Security controls
  – Encryption
  – Integrity checks
  – Input validation
  – Data sanitization
  – Obfuscation

³https://github.com/OpenRCE/pydbg
⁵http://debugger.immunityinc.com/
⁸http://jacquelin.potier.free.fr/winapioverride32/
• Relevant design properties
  – Protocols or data flow types, for example HTTP, LDAP, SQL, API call, command invocation, file I/O
  – Data formats
  – Authenticated, encrypted or signed data flows and data stores
  – Encodings
  – Programming languages, especially if managed or native
  – File Permissions
  – Process privileges
  – Custom developed components
  – 3rd party components

A.4 Step 3: Adapted Threat Modeling

In the context of the proposed risk analysis process, adapted threat modeling refers to the analysis of a system's architecture and design. The goal is to find out which threats, such as denial of service, disclosure of information, escalation of privilege, of what parts of the application can lead to the previously defined business worst cases scenarios. Obviously, this mapping, or, "translation" of BWCS to technical threats is only possible for software which is built for a defined business purpose or whose area of application is known to the risk analyst. This constitutes the adaptation of the original Microsoft STRIDE\(^9\) threat modeling [7] which does not consider concrete business use cases. As STRIDE threat modeling, the adapted version is based on a data flow diagram (DFD). Thus a technical threat can be defined as a tuple of one or more DFD elements and a STRIDE threat. Figure A.3 presents a simplified DFD for the example medical system. A DFD consists of four main symbols: Circles represent logical processes; rectangles represent external entities; double horizontal lines represent passive data stores; and arrows represent data flows. The special characters are not part of the standard notation. They represent examples for possible annotations. Here they mean "not encrypted" (-), "encrypted" (+), "user privileges" (~) and "elevated privileges" (*). The DFD can be manually derived from the security overview created in step 2, or, if the overview exists in a standardized form, it could be generated with tool-supported.

There are two approaches to the mapping of BWCS to technical threats:

**Top-down:** For every BWCS, examine which technical threats could lead to the BWCS.

**Bottom-up:** For each DFD model element, brainstorm if any technical threat could pose a security problem which could lead to a, possibly not yet identified, BWCS.

\(^9\)STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.
A.4.1 Practical Aspects

Slimming Down the Adapted Threat Modeling

Practical experience [2] suggests that analyzing each model element in a DFD is often time-consuming and leads to overlapping results for different elements. Alternatively one can instead examine entire data flows from source to sink or system interactions with external actors.

Data Flow Diagrams (DFDs)

Using DFDs to model the target application and perform threat modeling has several advantages:

• A DFD is easy to understand for security experts, software developers and business people.

• DFDs are suitable for security and risk analysis as they represent all interaction points that an attacker may leverage and also shows how data, often the target of attacks, moves through the system.

• Annotations may enrich the diagram and add security relevant information taken from the security overview. Examples are file permissions, process privileges, or security properties of data (flows) such as confidentiality or integrity.

• DFDs scale well since they are hierarchical and allow to scope and analyze the system at different decomposition levels.

A.5 Step 4: Risk Estimation

A.5.1 Technical Likelihood

The overall risk estimation is based on two principal factors: impact and likelihood. The latter is composed of technical and non-technical factors. While risk, impact, and non-technical likelihood are estimated per BWCS, see step 1 Section A.2, the technical likelihood is estimated per technical threat. Figure A.4 shows an ontology which contains all considered risk factors and their relationships with each other. For simplicity, impact, likelihood, attacker skill, and assurance level have all three levels. However, this metric can be adapted without affecting the overall process. The most significant non-technical likelihood factors are:

• Identified vulnerability indicators, see Section A.6.

• The sophistication of security tests, if any have been performed, to cover the technical threats.

• Potentially discovered vulnerabilities under consideration of the chance for occurrence, detection and exploitation; the required technical skill for exploitation; and requirements and mitigation factors such as physical access restrictions, restricted LANs or required user accounts.

The technical likelihood allows analysts to determine the technical threats with the highest "risk reduction potential". This means the architect can focus the development and testing effort on the corresponding application parts and thus achieve the most efficient risk reduction possible.
Figure A.4: Ontology of risk factors and their relationships
A.5.2 Risk Estimation

The risk estimation for BWCS can be (re-)done as soon as a modification has manifested itself in the application binaries. It involves the following steps, from which all but the first may be repeated after every modification:

1. Estimate impact and non-technical likelihood. This is done in step 1 of the overall process, described in Section A.2.

2. For each technical threat associated with the currently selected BWCS: Estimate the technical likelihood by evaluating relevant vulnerability indicators and considering eventual, previously performed practical tests. Note that these might have become obsolete due to modification of the application.

3. Aggregate the technical likelihood of all associated technical threats and merge the result with the threat-independent non-technical likelihood factors.

4. Estimate the risk for the BWCS based on impact and likelihood determined in the previous steps.

5. If the risk is too high, there are two options:
   (a) Reduce the attack surface by modifying the application to have less technical threats or vulnerability indicators.
   (b) Perform more, or more sophisticated, practical security tests to increase the precision of the estimation. Therefore:
      i. Order all technical threats according to their impact on the overall risk and select the highest ranked threat.
      ii. Perform a not yet conducted security test covering this threat. The test may be taken from a relevant vulnerability indicator, from the test library introduced in Section A.7, or it can be a freestyle test.
      iii. Evaluate the practical results and re-estimate the technical likelihood. Identified vulnerabilities are evaluated according to the properties listed in Section A.5.1. Consider especially the security assurance level (see Section A.7.1) of the performed tests in regard to the expected attacker’s skill level.
      iv. Go back to step 3 of the risk estimation.

A.6 Step 5: Selecting Appropriate Security Tests - Vulnerability Indicators

As described in Section A.5, security tests are a mean for more precise risk estimation. This however requires a proper mapping of technical threats to appropriate tests. To make this step more systematic, and ideally tool-supported, we introduce the concept of vulnerability indicators.

A.6.1 Vulnerability Indicators

A vulnerability indicator is a pattern in a security overview (see Section A.3) that points to a possible vulnerability, together with a suggested practical test, or family of tests, to find out if the indicated vulnerability indeed exists. The vulnerability indicator approach is therefore similar to that of the “threat library” proposed by EMC [2]. The EMC threat library links DFD patterns with “threats” that basically correspond to our tests. In both concepts, most patterns do not describe actual vulnerabilities, but conditions signifying an increased likelihood that a certain vulnerability is present. While the EMC threat library entries are based on patterns in manually created DFDs, vulnerability indicators comprise patterns in a security overview. This means that vulnerability indicators can not only be based on manually created design information but also on implementation properties extracted via static/dynamic binary analysis. Good candidates are properties that are either a prerequisite for, or frequently occur in the context of certain vulnerabilities.
To our knowledge, the idea of vulnerability indicators as described above has not yet been presented in literature. Related research fields such as "vulnerability prediction" and "security metrics" also deal with the quantification of security relevant information, but their goals and results are less technical, less specific, and in most cases not based on binary analysis.

Besides the suggestion of appropriate tests, vulnerability indicators serve an additional purpose: They improve early risk estimation by indicating vulnerabilities that have not yet manifested themselves in the source code or binary. Moreover, they allow a prioritization of technical threats (see Section A.4) before any practical tests are performed. Inspired by the EMC threat library, table A.1 lists exemplary vulnerability indicators.

<table>
<thead>
<tr>
<th>Pattern in security overview</th>
<th>Example test activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication process with hard-coded credentials</td>
<td>Reverse engineering</td>
</tr>
<tr>
<td>Process writing to a data store</td>
<td>Path traversal attack</td>
</tr>
<tr>
<td>Process invoking OS commands</td>
<td>OS command injection</td>
</tr>
<tr>
<td>Unauthenticated network data flow</td>
<td>Network spoofing</td>
</tr>
<tr>
<td>Unencrypted network data flow</td>
<td>Network sniffing</td>
</tr>
<tr>
<td>Unsigned network data flow</td>
<td>Network traffic tampering</td>
</tr>
<tr>
<td>Interface with data flow to a SQL database</td>
<td>SQL injection</td>
</tr>
<tr>
<td>Interface with data flow to a process written in C/C++</td>
<td>Fuzzing</td>
</tr>
<tr>
<td>Data store that stores passwords</td>
<td>Offline password cracking</td>
</tr>
</tbody>
</table>

Table A.1: Vulnerability indicators consist of two parts: The pattern in a security overview that indicates a possible vulnerability and suggested corresponding security test activities.

Desired Properties of Vulnerability Indicators

A vulnerability indicator should be specific and "sound" enough, so that it does not produce too many false positives. But it should also be generic to keep the collection of indicators manageable. Vulnerability indicators extend the concept of a security overview as they are interpretations of the gathered information. This means that the information on which a pattern is based should be easy to include in security overviews and ideally be automatically extractable from the application binaries.

Extending the Collection of Vulnerability Indicators

The proposed process suggests setting up and maintaining a collection of vulnerability indicators that the risk analyst can re-use for different applications. The indicators listed in Table A.1 are only examples to demonstrate the general idea. We propose the following three steps to create new indicator and thus obtain a more complete set:

1. Vulnerability research
   The first step is to identify relevant classes of vulnerabilities that should be indicated. Appropriate
sources are collections of threats and vulnerabilities such as CAPEC\(^{10}\) and CWE\(^{11}\), and secure software development and security testing literature, for example [5], [6], [3], and [4].

2. Vulnerability context analysis
   Once a certain vulnerability class has been chosen to cover, the next step is to analyze the context and identify critical, static environment properties. The goal is to determine a reliable pattern that indicates the vulnerability.

3. Practical realization
   Finally, practical solutions to extract the context information from a binary and add it to a security overview need to be found. The concrete approach may vary for different indicators. The list in Section A.3.2 presents tools which could be useful.

A.7 Step 6: Security Testing

One possibility to cope with identified technical threats is to cover them with security tests to check the practical system resistance against them. The proposed process supports the selection of appropriate security tests, as explained in Section A.5, with the help of vulnerability indicators. The integration of a systematic collection of practical security tests, in form of a testing library, completes the support for an efficient use of testing resources.

A.7.1 Testing Library

Format and Security Assurance Levels (SAL)

The library entries are grouped in blocks such as "web applications", "network traffic analysis", "reverse engineering / binary analysis" or "fuzzing". The creation of such blocks supports the bottom-up usage explained below and eases maintenance of the library. In addition, each library entry is labeled with a so-called Security Assurance Level (SAL) which reflects the level of confidence in the results of a test. It is determined by estimating the required skill level that an attacker needs in order to perform the test. The levels range from 1 (very easy) to 3 (very complex). The SAL / attacker-type ratio is an important factor for the estimation of the technical likelihood. Given the case, for example, that an SAL 2 tests unveils a vulnerability. An advanced or professional attacker is much more likely to detect and exploit it than an opportunity rider.

Usage

The primary library use is a "top-down", that is to cover technical threats to practical tests, as exemplary shown in Figure A.5. The figure also explains all other introduced terms using the example of a medical system introduced in Section A.2. P1, DF1, and DF2 refer to a process and data flows being part of the simplified DFD in Figure A.3 which is also part of the medical system example. From a practical point of view, the analyst should also proceed bottom-up by studying the test library and checking for possibly forgotten technical threats.

Desired Characteristics of Test Entries

In general, the description of the tests should to be specific enough to be of practical value for the analyst. On the other hand, it should be as generic as possible to keep the number of entries reasonable and make them applicable on different applications.

\(^{10}\)http://capec.mitre.org/
\(^{11}\)http://cwe.mitre.org/
### A.8 Summary

We outlined a process for continuous, transparent risk assessment and efficient risk reduction during iterative software development. It consists of six steps: The definition of business worst case scenarios, the creation of a security overview, an adapted threat modeling, the actual risk estimation, the selection of appropriate practical security tests and the actual testing. For each of these steps, we mention where tool support is possible and sketch first ideas for its implementation. Moreover, we introduced three key concepts: The creation of a security overview based on static and dynamic binary analysis; the use of vulnerability indicators that point to possible vulnerabilities and suggest appropriate practical tests; and finally the set up and utilization of a test library with entries categorized according to Security Assurance Levels. Used in combination, these concepts allow a transparent risk assessment and efficient risk reduction.

### References


