	Title: State of the art on methodologies for risk- and model-based security testing
	Version: 1.0 **Date :** 12.12.2011 **Pages :** 34
	Editor: Fredrik Seehusen (SINTEF)
	Reviewers: Edguardo Montes de Oca, Stephan Schulz, Ari Tarkanen
	To: DIAMONDS Consortium

The DIAMONDS Consortium consists of:
Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Get IT, Giesecke & Devrient, Grenoble INP, itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, TU Graz, University Oulu, VTT

Status:		**Confidentiality:**			
[ ]	Draft	[X]	Public	Intended for public use	
[ ]	To be reviewed	[ ]	Restricted	Intended for DIAMONDS consortium only	
[ ]	Proposal	[ ]	Confidential	Intended for individual partner only	
[x]	Final / Released				

Deliverable ID: D1.WP4

Title:
State of the art on methodologies for risk- and model-based security testing

Summary / Contents:

Contributors:

Menz, Rennoch, Großmann, Schieferdecker (Fraunhofer FOKUS)

Maag, Cavalli (Institut Telecom SudParis)

Erdogan, Li, Seehusen, Stølen (SINTEF)

Noponen (VTT)

Weiser (Oulu)

© Copyright DIAMONDS Consortium

TABLE OF CONTENTS

FIGURES

TABLES

HISTORY

Vers.	Date	Author	Description
0.1	22/0811	Fredrik Seehusen	Template created
0.1	26/10/11	Stephane Maag	Section 5 added
0.2	11/11/11	Menz, Rennoch, Großmann	New Secton 6
0-5	15/11/11	Fredrik Seehusen	Integrated contributions and edited document
1	12/12/11	Fredrik Seehusen	Feedback from reviewers implemented

APPLICABLE DOCUMENT LIST

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

EXECUTIVE SUMMARY

This document constitutes the first deliverable of work package 4 on risk- and model-based security testing methodologies. While the other work packages of the DIAMONDS project describe *techniques/methods* and *tools,* work package 4 describes *processes/guidelines* for applying these tool and techniques in practice. Thus, the state-of-the-art topics of this deliverable are related to techniques in WP2 that should be supported by methodologies and whose state-of-the-art survey has not already been covered by deliverable D1.WP2.

1. INTRODUCTION

This document constitutes the first deliverable of work package 4 of the DIAMONDS project. Whereas the other work packages focus on techniques (work package 2) and tools (work package 3), work package 4 focuses on methodology, i.e. processes and guidelines for how to apply the techniques and tools in practice.

The deliverable addresses two main research areas: *model-based security testing* and *risk-based security testing.* The first topic is related to task 4.2 of work package 4, and is documented in Section 2. The second topic is related to task 4.3 of work package 4 and documented in Section 3.

2. STATE-OF-THE-ART ON METHODOLOGY FOR MODEL-BASED SECURITY TESTING

This section gives a state-of-the-art on methodologies for model-based security testing. The topics are related techniques developed in tasks 2.1 – 2.3 in work package 2 that should be supported by methodologies and whose state-of-the-art has not been already covered in deliverable D1.WP2. In particular, Section 2.1, model-driven security, is related to task 2.2 and 2.3. Section 2.2, security testing methodologies based on formal languages, is related to task 2.1. Sections 2.3 and 2.4, network monitoring using machine learning techniques and security testing with GCC extensions, are related to techniques developed in task 2.2. Finally, Section 2.5, model inference assisted fuzzing is related to task 2.1.

2.1 MODEL-DRIVEN SECURITY

Model-Driven Security (MDS) advocates a methodological approach in which (1) security requirements to be formulated and tested at high-levels of abstraction in the early phases of system development, and (2) security analysis results to be maintained by transformations to lower levels of abstraction. As far as we know, the term MDS was first coined in [16]. Most of the papers addressing MDS consider the specification of access control requirements [16][17][18][20][23][27][36][5]. Perhaps the most notable in this area is the work related to SecureUML [5][16][17][18]. SecureUML is an extension of UML for modeling platform independent access control requirements. It is intended that SecureUML be used together with a system design language (e.g., UML class diagrams or UML statecharts). The SecureUML access control requirements can be transformed into platform specific models. In [17], three platforms are considered: Enterprise JavaBeans (EJB), Microsoft Enterprise Services for .NET, and Java Servlets. An advantage of this approach is that these platforms already have access control enforcement mechanisms. However, it is difficult to precisely characterize what it means that a system satisfies the enforcement mechanisms. For instance, although [17] formalizes what it means that a system adheres to an access control model on the platform independent level, they are unable to do so for the platform specific level. No other approach to MDS, that we are aware of, gives a precise description of what it means that a specification is secure (even for the platform independent models). Consequently, these approaches do not exploit the full potential of MDS.

Fernandez-Medina et. al. [23][24][25][26][35][36] show how platform independent access control requirements for a conceptual data base model can be expressed in an extension of UML and OCL. However, only an informal description of how to interpret the security constructs of the extension is given. A transformation from the conceptual data base model and its associated access control requirements to an XML database schema is also discussed without being precisely defined in the papers. Similarly, [20] proposes a platform independent model for access control and discuss issues that must be considered in order to transform this into a platform specific model, but no precise characterization of transformation or adherence is given. Other approaches to MDS that are not specifically related to access-control requirements are presented in [19][29][30].

Heldal and Hultin [30] presents an approach in which UML diagrams can be annotated by security (in the sense of confidentiality) requirements. The work also discusses the possibility of transforming annotated UML diagrams into Java code that can be validated with respect to confidentiality constraints by the language-based checker Jif (Java Information Flow).

Nakamura et. al. [32] present a tool framework for web service security. Three levels of abstraction are considered: the operation level, the execution level, and the deployment level. At the operation level, UML models can be annotated with security primitives such as “integrity” or “confidentiality”. Security requirements at the execution level are assumed to be written in so-called deployment descriptors of J2EE. At the deployment level, security requirements of the execution level are bound to specific security infrastructures. Transformation rules between the abstraction levels are discussed, but no precise characterization of adherence or transformations is given. Haftner et. al. [19][29] define the abstract syntax of a domain specific language for the design of inter-organizational workflows. The language supports various categories of security patterns. A distinction of platform-independent and platform-specific models is made, and a transformation from the PIM to PSM is discussed. The PSM considered is the abstract syntax of the eXtensible Access Control Markup Language (XACML), a standard supporting the specification of authorization policies to access Web services.

Other works that consider security in a model-driven setting (without addressing transformations) are the access-control related works [14][15][21][22][2][31] and the more general high level security requirements works [13][28][37]. Of these, only [13][14][21][2][31] offer a formal foundation which allows the full potential of MDA to be realized.

Abie et. al. [2] integrate a language for specifying high level security requirements – the Security Requirement Language (SRL) – with UML sequence diagrams. SRL is based on first-order logic extended with a small set of modal operators. Thus a formal foundation for precise security analysis is offered. Integration of SRL and UML sequence diagrams is achieved by annotating the sequence diagrams by so-called *tagged values* that contain SRL macros defining security requirements.

Alghathbar et. al. [14] present authUML, a framework for analyzing access control requirements of UML use cases. The formal foundation of authUML is based on Prolog logic programming rules. AuthUML has three main phases. In the first phase, a set of access control requirements is transformed into so-called access predicates. In the second phase, all accesses predicates are ensured to be consistent, complete, and conflict-free without considering the operations used to describe their functionality. Finally, in the third phase, authUML analyzes the access control requirements in operations.

Doan et. al. [21] integrate access control requirements into UML use case, class, and sequence diagrams. They propose a number of *security assurance rules* (SAR's), which can be used to enforce access control requirements for UML. They also define an algorithm with which to check that UML diagrams adhere to the SAR's.

Koche et. al. [31] present an approach in which access-control requirements are integrated into the software development process. In their approach, access control requirements are specified in the so-called Abstract Security Model (ASM) which is a graph-based security framework that provides a theoretical basis for verifying security constraints. The ASM model may be integrated with UML use cases, class, and sequence diagrams to obtain a so-called concrete security model that can be checked for consistency.

Jurjens [4][2] presents an approach in which UML diagrams can be labeled with confidentiality, integrity, and secure information flow constraints. He also shows that these kinds of requirements are preserved under refinement. The semantics is based on so-called UML machines which are a kind of state machines.

2.2 SECURITY TESTING METHODOLOGIES BASED ON FORMAL LANGUAGES

As mentioned in the deliverable WP2.D1, several formal languages are provided in order to express security models, properties and policies. Most of these languages are dedicated to formally define different kind of security aspects that can be applied through diverse contextual security testing architectures. Based on these formalisms, some methodologies have been defined, implemented and some of them are now tooled. We present in this section the security testing methodologies that take as basis some of the languages defined in the deliverable WP2.D1. We will mainly focus on the Timed Extended Finite State Machine (TEFSM) [51], Nomad [54], Or-BAC (Organizational Based Access Control) [52], OCL4ST (OCL for Smart Testing) [55] and SDL (Specification and Description Language) [53].

2.2.1 Security testing methodology with TEFSM and Nomad

The integration of security rules into a TEFSM model describing the behavioral aspects of a system leads to a TEFSM specification that takes the security policy into account: it is called *secure functional specification*. The integration process is twofold. At first, the algorithm searches for the rules to be applied on each transition of the TEFSM specification.

Then, it adds some states, transitions or updates the guard of the related transition. These modifications depend on the nature of the rule (prohibition, permission or obligation) and its syntax format. To integrate security rules into a TEFSM specification, the following assumption has to be made: the security rules to be integrated are consistent. We assume that the security rules do not contain any incoherent rules [6]. Besides, the TEFSM may be non deterministic if for instance two outgoing transitions have their guard true at the same time. In that case, one of them, chosen randomly, is fired.
According to the Nomad syntax, there are several possible forms for security rules. It would obviously be tedious to deal separately with each of these forms. Consequently, we classify the Nomad security rules in two main classes described as follows:

1. Basic security rules: in this class we consider security rules of the form $R(start(A)|O^{[<]-d}done(B))$ where A and B are actions, $R \in \{F;O;P\}$ and ($d > 0$).
2. General security rules: a general security rule denotes any rule that does not fit into the first class. This means that the rule may contain several contextual and/or timed operators and/or logical connectors.

2.2.1.1 *Formal testing of security rules*

The methodology takes the following three elements as input:

- a TEFSM functional description of the system,
- a set of security rules described using Nomad language,
- the existing implementation of the system.

The objective is to check whether the existing implementation verifies the security rules. It proceeds in four steps as shown in Figure 1.

1. The security rules are integrated into the TEFSM specification according to the different algorithms above mentioned.
2. Abstract test cases are automatically generated from the secured TEFSM specification obtained in the first step. We use TestGenIF tool that implements a formal test case generation approach based on the Hit-or-Jump algorithm [8]. This tool accepts a TEFSM specification encoded in the IF textual formalism [9].
3. The abstract test cases are transformed into an executable script capable of communicating via http (or https) with the implementation under test.
4. The concrete test cases obtained from the instantiation are executed on the implementation under test to check whether it verifies the security rules.

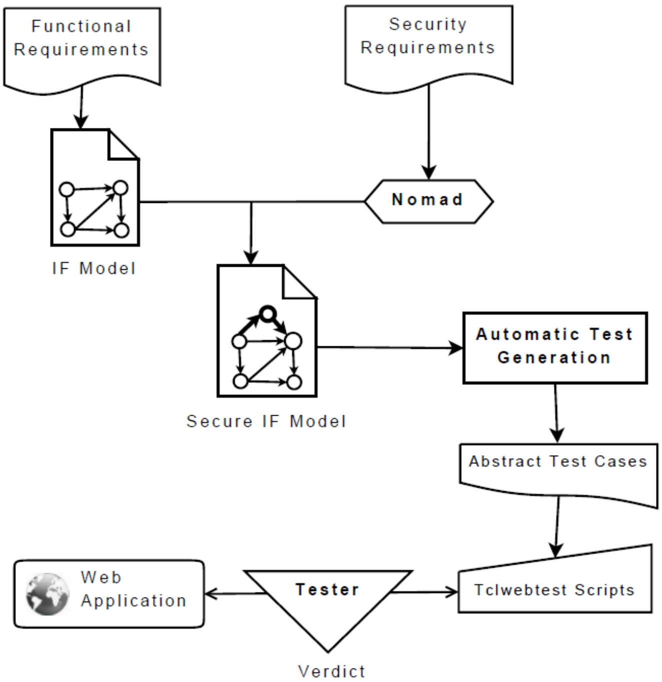

Figure 1 - Security Testing methodology overview

2.2.2 Security testing methodology with ocl4st

2.2.2.1 UML4ST

UML4ST (UML for Security Testing) is a subset of the UML notation as presented in [10]. UML offers a set of diagrams to represent the functional and security aspects of a system under test from static and dynamic points of view. This subset UML4ST is composed of three diagrams: class diagram, object diagram and state-transitions diagrams (Figure 2).

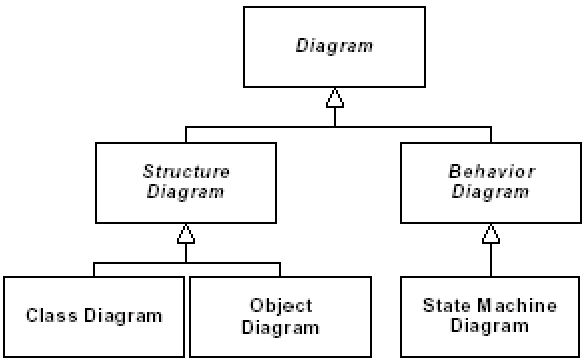

Figure 2 - UML meta-model of UML4ST

The class diagrams are used to represent the data of the system, their types and links. The state-transitions diagram (or state machine) contains on one hand the formal design of the functional behaviors of the SUT and allows on the other hand to specify the dynamicity of the security properties to test. In UML4ST, the execution of a transition of a state machine respects the semantic run-to-completion.

An event emitted on the state machine is selected among a pool of events recognized by the entity carrying the state machine. For each execution step, an event is selected among this pool and triggered. The behaviors depending of this trigger are executed. During the execution of this step (eventually composed of 'sub-steps'), no other pool event can be triggered till its end (or completion). The execution starts with the event triggering in a stable configuration and also ends in a stable configuration. A stable configuration is characterized by the impossibility to trigger a transition of completion with no events. A run-to-completion step can be seen as a complex transition between two stable configurations of the state machine.

The triggering of a transition $t\ (evt_t, grd_t, efft_t)$ between a state A and a state B depends on:

- The triggering of the event evt_t
- The respect of the transition grd_t

The execution order of the behaviors associated to the triggering of the transition *t* is given by the sequence:

1. Execution of the output behaviors from state A
2. Execution of the behaviors of the effect eff_t
3. Execution of the input behaviors in state B.

2.2.2.2 Methodology with OCL4ST

In order to precise the behaviors of the SUT, UML4ST is accompanied by a subset of the OCL language (Object Constraint Language). OCL is a language of constraints expression de facto dedicated to UML (and used by the OMG to define the meta-model UML). OCL allows to formally express the constraints on the UML entities. It allows to specify constraints on the states of an object or a set of objects.

In UML4ST, OCL is used to formally design the behaviors of an SUT. Concretely, a behavior is an action on the system under test, that can be activated in a particular context (i.e. the activation condition of the action). However, OCL is not an action language (or procedural), but a declarative language. It only allows to describe constraints on events composing a UML model. In order to overcome this concern, UML4ST disposes of its own OCL interpretation. The subset OCL supported by UML4ST is called OCL4ST for OCL for Security Testing.

All the object and class diagrams, completed by the state-transitions diagrams used to design the functional behavior of the system under test is called the *test model*: it represents the SUT behaviors we want to validate. It has the particularity to be executable or interpretable in order to produce executable test sequences (test cases obtained from the test model). Moreover, it contains the test oracle allowing to establish the verdict of each test when executed on the SUT.

Each of the other state-transitions diagrams used to design the security properties to be tested is called *secure diagram*. A secure diagram contains the abstract execution traces that are interpretable on the corresponding test model. This type of diagram is strongly dependant of the test model. The security scenarios are thus defined allowing to express the specific test objectives which complete the functional objectives automatically computed from the test model.

Figure 3 illustrates the entire generation approach from secure diagrams (red) and indicates its complementarity with the common test sequences generation approach.

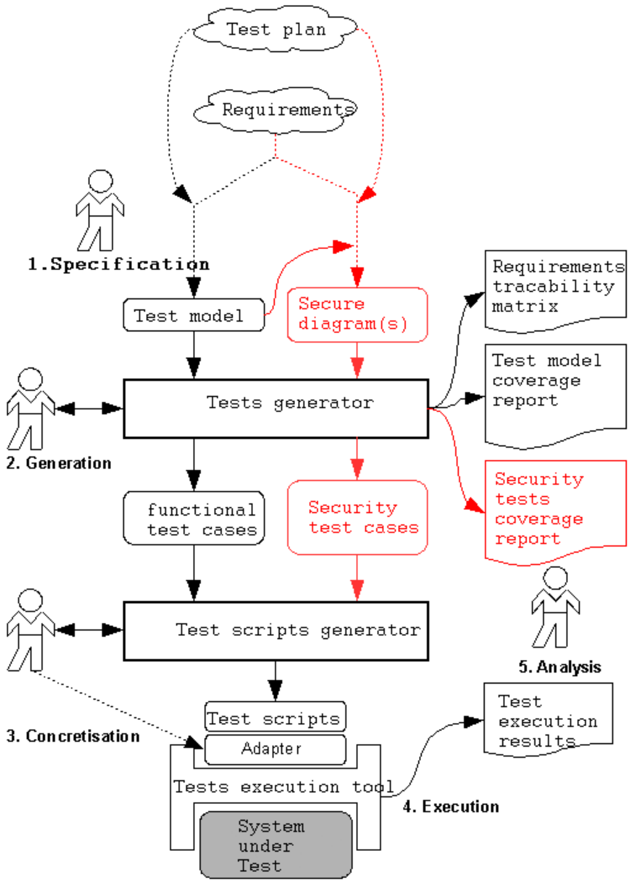

Figure 3 - Generation methodology from secure diagrams

The security scenarios are written by the testing engineer expert from the testing requirements (test plan and/or functional and security requirements) and the operation/events of the test model. The basis of the functional test cases generated from the test model is then enriched by the security test cases concretizing the defined security scenarios. A specific report may then be generated to deliver the testing coverage rate.

The security scenarios described by the dedicated state machines are some abstract traces whose the concretization is executable on the system under test. In this methodology, such a trace is composed of an operation sequence and/or events described in the test model.

We may establish a hierarchy of the scenario expressiveness with regard to the abstraction level of the sequence defined by the testing expert. Four expressiveness levels of a scenario are then proposed:

- Type 1: completely valuated trace: an operation sequence whose the input parameters are all valuated.
- Type 2: trace on an operation sequence with free valuation: a type 1 trace in which the operation parameters can also be free.

- Type 3 : trace on partial operation sequence: a type 2 trace whose the operation sequence can be parameterized (e.g. in specifying the optional presence of an operation in the trace).
- Type 4 : trace constrained by regular expression: a type 3 trace augmented by constraints (guards) on states reached by the generated test(s). The constraints are expressed in OCL and deal with the state variable of the system.

2.2.3 From the Or-BAC security rules and the EFSM-based SUT

To achieve its active testing methodology, security experts need to edit the set of security rules that system under test (SUT) has to respect. These rules can be specified in Or-BAC or Nomad models and the SUT is also specified from its functional point of view based on the formal language SDL (or IF2.0).

2.2.3.1 From SDL representing the SUT

To describe a communicating system based on the SDL language, several tools are available. But, whatever the tool chosen by the testing expert in order to edit, verify and test and SUT, the methodologies rather follow the same steps. The communicating system is specified by means of states and transitions (based on the Extended Finite State Machine formalism). One can also verify the specification syntactically and semantically. The syntactic analysis ensures that the specification complies with the syntactic rules of SDL, whereas the semantic verification ensures the consistency of the specification. This step is carried out not only by static analysis but also by an automatic exhaustive exploration of the specification. This is performed by testing all possible ways of system execution, with a certain number of rules and the cases of violations such as deadlocks, loops etc. During verification, the main properties analyzed are:

- Safety (absence of deadlock, unspecified reception, blocking cycles, etc). Deadlock takes place when a state of the system, reachable from the initial state cannot trigger a transition anymore.
- Promptness (livelock). A state is known as alive if it can be reached starting from all the states of the global system.

2.2.3.2 From Or-BAC

The Or-BAC model introduced in the deliverable WP2.D1 has been integrated in a testing security policies methodology. Besides, an associated tool called MotOrBAC [[1]] has been developed to help designing and implementing security policies using the Or-BAC formalism. First, the security policies are designed, uploaded and stored. They are later simulated to verify their consistencies. While the Nomad-based security rules are currently edited textually (no specific tool exists to perform these tasks yet), the contexts evaluation through the security policies are managed through APIs. The following steps are then followed.

[1] http://orbac.org/index.php?lang=en&page=motorbac

- Create/edit the Or-BAC security policies (XML RDF files can be performed). The editorial operations must take into account the administration policy associated if this latter is activated.
- Contexts evaluation. The contexts defined in the Or-BAC model are evaluated, which are the temporal contexts declared by the user and the contexts expressing a condition on the attributes of the concrete entities. The composition of the contexts may also be processed if required.
- Checking of the concrete security policy and its administration policy.

2.3 NETWORK MONITORING USING MACHINE LEARNING TECHNIQUES

Detecting network anomalies and intrusions in a network environment is possible with the help of machine learning techniques. However, the network environment should be stable and somewhat isolated for the best efficiency. Networks in industrial control systems meet these requirements. The main method of our study is to exploit the traffic captures from a real network. The method we are using is machine learning combined with passive monitoring and a priori knowledge of protocols used. Passive monitoring is required due to the nature of the environment. It is important that no measuring device or monitoring system interferes with the ICS environment under scrutiny.

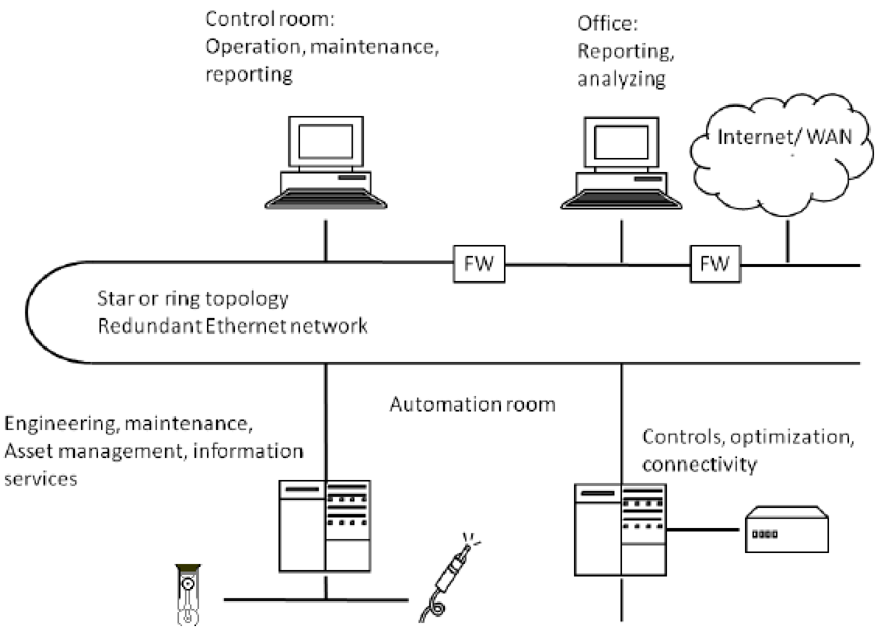

Figure ICS Environment

2.3.1 Traffic analysis

The main method of the study is to exploit the traffic captures available to us. In the initial testing and feasibility studies we have so far used tools such as NetAI, NetMate and WEKA 3. NetAI and NetMate are used as provided by the NetMate-flowcalc bundle [56], [57], [58], [59].

2.3.2 The machine learning approach

We argue that the factory networks for ICS that are functioning properly and without serious design flaws can be defined as nearly closed environments. When the factory network would be in a normal status without serious incidents there is typically very little noise. Again, if the network architecture is well defined and implemented, most of the network traffic on the ICS level of the network should be more deterministic than that of open networks such as the office networks.

The most challenging aspect in the initial phase of using machine learning is feature extraction and selection. We have been identifying possibilities which include usual features used by many IDS's. Properly done feature extraction is one of the most important steps in machine learning, producing a classifier with higher generalization capability by excluding redundant or irrelevant attributes. The feature selection process benefits from extensive testing of the recorded live data.

The usage of payload form and payload data has some significant challenges, subject to future work. However, they could be used to very accurately monitor the sanity of the system and conformance to security policies. Some features studied for possible use include: *throughput, IP address and port pairs in a flow, average size of the packets, timing, flow direction, Average duration of flow between endpoints, Payload form, payload data, MAC to IP mapping, networking protocol, protocol settings and connectivity number.*

For the model of the ICS network environment all attributes would have to take into account the possible periodic nature of the traffic. Depending on the system being monitored, there might be variations caused by maintenance, periodic processes or environmental fluctuations.

The main argument for using machine learning approaches is the more closed nature of ICS networks. Any benign changes in the traffic are likely caused by actions that can be informed to the system a priori. Malign or anomalous activity, on the other hand, is any new form of traffic that has not been taught or programmed to the monitoring system. We are planning to develop a proof of concept of the traffic analysis and machine learning functionality on top of our existing system. The system already implements support for a vector machine and data pre-processing. We plan to implement new algorithms when needed, or use open source implementations when possible. To be able to implement the planned system, a number of steps must be taken before the implementation can be started. For the next few steps, a study will be completed regarding the specific algorithms available and their strengths and weaknesses in ICS network environment. In addition to this, a deeper analysis of the different aspects of the network layers present in ICS environments must be accomplished.

2.4 SECURITY TESTING WITH GCC EXTENSIONS

GNU Compiler Collection [60] is a very widely used compiler. GCC is open source and it compiles C, C++, Java, Fortran, Ada and Objective-C. GCC versions 4.5 and later support plug-in modules. Plug-ins have full control over the program's presentation and utilize all the information extracted and generated by the earlier compiler passes. Plug-ins make possible that the program analysis and instrumentation can be added and removed relatively easily. We use plugins to extract information about the program structure and execution. VTT has built a set of libraries that can be used to build external tools that utilize this information

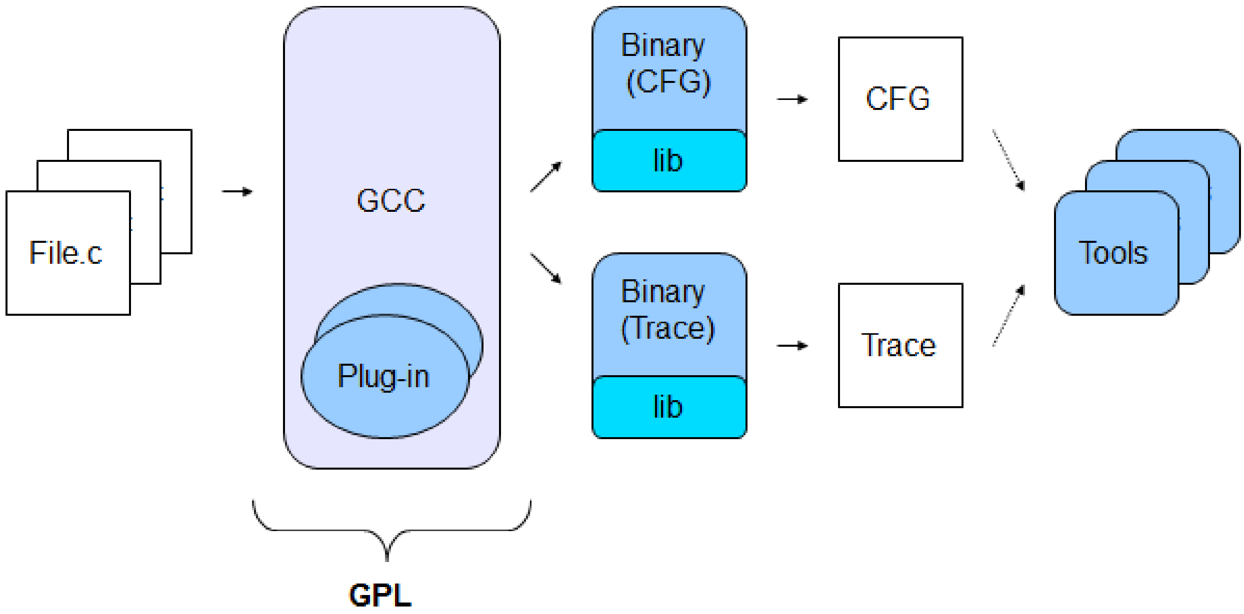

Figure CFG and Trace-plugins

CFG (Control Flow Graph) plugin is used to instrument the program to output representation of its global control flow graph and some additional link time information when it is executed. The GFC contains more information than what is generally available at compile time. CFG plugin offers much lower level and easier to use representation than the original source code.
Trace is a sequence of function calls, returns and visited basic blocks that describes the program's execution path. This information can be combined with the CFG to reconstruct a specific instance of program's execution.

The plugin-modules can be used in various ways:

- Automatic test case generation - searching test case inputs based on execution traces.
- Automatic test case selection – based on changes in the program structure and test case's execution trace.
- Model checking – e.g. correct use of API calls
- Fault detection – Static or dynamic checks to detect errors early
- Fault injection – for generating "hard to detect"-failures for testing purposes
- Program analysis – simplify debugging and gaining information about the program structure.

2.5 MODEL INTERFERENCE ASSISTED FUZZING

One example of model based security testing is fuzzing (or fuzz testing), which exposes tested software with malformed input. The aim is to evaluate if weaknesses in the software can be found, for example denial-of-service conditions, which can be further exploited to a full compromise of the software. Fuzz testing is negative testing – we do not aim to demonstrate presence of functionality, but would like to see about the absence of specific vulnerabilities.

Miller et al. have shown in their work that even very simplistically fuzzing models are able to disclose a large amount of parsing errors [38][39]. To systematize work, the Oulu University Secure Programming group had started in 1999 the PROTOS project [40] and has developed an approach to systematically test implementations of protocols in a black-box fashion. Several successful test suites have been released and this research led to a spin-off company: Codenomicon Ltd.

Our previous work in robustness testing of protocol implementations has shown that manually designed structural mutations and exceptional element values are an efficient way to expose errors in software. Unfortunately, while powerful, manual test design has some bottlenecks: i) it requires some kind of format specification as a basis; and, ii) poorly documented formats must be reverse-engineered before test designers can write a model-based test suite for the format. The human factor also brings in the danger of tunnel vision, as the power of manually designed cases is largely dependent on the expertise and imagination of the designer. On the other hand, blind random fuzzing has a considerably lower entry barrier, but is hindered by the impossibility of efficiently addressing a virtually infinite input space in finite time.

In a subsequent direction, the aim was to automatize the test case generation. Radamsa [41] is an example of this kind of black-box fuzzing tool. As an input it requires valid data stream samples, either in file format or as network data. Radamsa inferes a model from this data and then generates – based on predefined heuristics –test cases. As these test cases are not required to trigger predefined situations, Radamsa has leeway in generating such model inference.

The results have been promising: more than 28 findings are reported in the Common Vulnerabilities and Exposures (CVE), a database of computer-security vulnerabilities. Open and closed source software has been affected.

3. STATE-OF-THE-ART ON METHODOLOGY FOR RISK-BASED SECURITY TESTING

The second topic, risk-based security testing, is related to task 4.2 of work package 4, and is in this document presented in Sections 3.1-3.2. These sections are related to task 2.4 of work package 2. Section 3.1 presents standards supporting risk-based security analysis while Section 3.2 presents a literature survey of risk-based testing.

3.1 STANDARDS SUPPORTING RISK-BASED SECURITY ANALYSIS

The evaluation and certification of a product is the advantageous method of attesting its quality through a standardized process. Since the certification is generally carried out by an independent certification body, the customer is now provided with a meaningful testament of the products quality. At the same time the developer benefits from the certification process by increasing their market potential and gaining additional confidence in the product due to the expertise from evaluation and certification labs.

The Common Criteria for Information Technology Security Evaluation is an international standard (ISO/IEC 15408) for the certification of IT-security products. The process conducted by the certification body is internationally agreed upon and the certificates internationally accepted. The evaluation process is highly driven by developer documentation and focuses on product development, security testing and vulnerability assessment. Besides creating trust in the product's quality, the evaluation results also allow the customer to compare the security functionality of similar products.

The ETSI TVRA method [12] developed by ETSI benefit from the CC work, e.g. by using a generic catalogue of security functional requirements (SFRs). TVRA is characterized by the following concepts and approaches:

- Threat types: Interception, manipulation, Denial of Service, repudiation of sending, repudiation of receiving
- Security objective types: Confidentiality, integrity, availability, authenticity, accountability
- attack potential is defined by the openness of a system to attack, attackers expertise and resources, systems availability.
- UML is used to model relationships within systems.
- Methods to analyze/evaluate system security including: threats, risks, vulnerabilities
- Step-wise approaches: TOE identification – objectives – functional security requirements – assets - vulnerability
- Calculation of attack potential
- TVRA uses CC taxonomy (family – class – component): Security requirements taxonomy (SFR)

- Lacks on details about generation of security tests.

The following table provides a comparison between CC and TVRA. The most important items have been marked with yellow.

Table 1: Comparison between CC and TVRA.

	CC	**TVRA**
Field of application	IT-Security products	Telecommunications system
Purpose	- To answer the **question if the TOE fulfills certain security needs**. - Making two TOEs comparable.	**Determine** how open to attack a system is.
Focus Point	*Resistance* to attack of the system	*Impact* of an attack on the system
Target of Evaluation	IT-Product	System under Standardization
Output	IT-Security certificate	Quantified measure of the risks to the assets and a set of detailed security requirements that will *minimize the risks*
EAL	Evaluation is based on a single evaluation level	Evaluation level can be expressed as a *range*: EAL3 – EAL5
Terminology - **Objectives**	*must* or *shall*	*should*
Countermeasures	IT countermeasures (firewalls, smart cards) **and *non-IT countermeasures*** (guards and procedures)	only *technical* security countermeasures are considered

CC and TVRA are both lacking on details on how to derive security tests from the TOE description. The following illustration presents the relationship between the SUT/TOE, its requirements, interfaces and security tests.

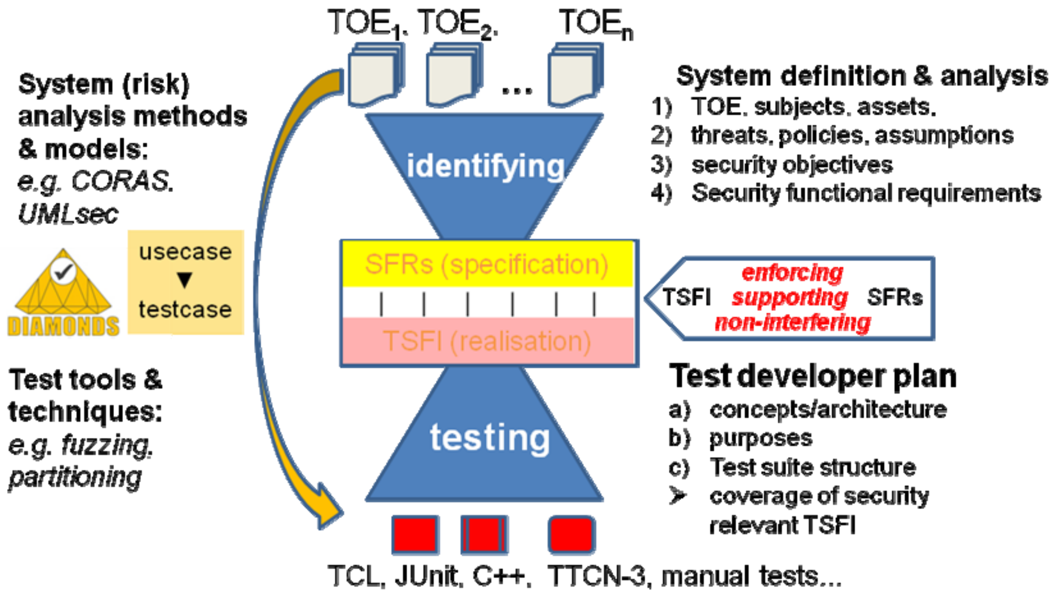

Figure 2: Diamonds contribution to standardization

The security analysis as part of the Common Criteria evaluation is performed during the vulnerability assessment aspect AVA and is closely linked to the aspects development (ADV) and guidance documents (AGD). The analysis is performed by the external evaluation body and supported by testing. Its goal is to determine whether exploitable flaws and weaknesses exist in the system, i.e. if the target of evaluation is resistant to penetration attacks. Publicly available information about known weakness for the specific product and product type serve as the basis for the analysis.

Following the principal of evaluation assurance levels (EAL) with an increasing evaluation depth, the vulnerability analysis during a particular evaluation only considers flaws and weaknesses that can be exploited with the attack potential relevant for the assurance level chosen for this evaluation. These different levels are shown in

Vulnerability Component	TOE resistant to attacker with attack potential of:	Residual vulnerabilities only exploitable by attacker with attack potential of:
VAN.5	High	Beyond High
VAN.4	Moderate	High
VAN.3	Enhanced-Basic	Moderate
VAN.2	Basic	Enhanced-Basic
VAN.1	Basic	Enhanced-Basic

Figure 4 - Attack Potential [11]

The attack potential score is calculated based on

- the time it takes to identify and exploit a vulnerability,
- the required expertise of the attacker,

- the needed knowledge of the systems design and operation,
- the needed window of opportunity as well as
- hard- and software required for the exploitation.

Figure 5**Fehler! Verweisquelle konnte nicht gefunden werden.** is taken from the Common Methodology for Information Technology Security Evaluation and associates those five factors with numeric values to allow for the calculation of a total score for the evaluation of the products resistance to vulnerabilities.

Factor	Value
Elapsed Time	
<= one day	0
<= one week	1
<= two weeks	2
<= one month	4
<= two months	7
<= three months	10
<= four months	13
<= five months	15
<= six months	17
> six months	19
Expertise	
Layman	0
Proficient	3*(1)
Expert	6
Multiple experts	8
Knowledge of TOE	
Public	0
Restricted	3
Sensitive	7
Critical	11
Window of Opportunity	
Unnecessary / unlimited access	0
Easy	1
Moderate	4
Difficult	10
None	**(2)
Equipment	
Standard	0
Specialised	4(3)
Bespoke	7
Multiple bespoke	9

Figure 5 - Calculation of attack potential [11]

3.2 A LITERATURE SURVEY OF RISK-BASED TESTING

In this section, we review state-of-the-art approaches to risk-based testing. Each approach is classified in Table 1. This table has the following categories:

- *Security specific*: Indicates whether or not the approach is specifically aimed at security
- *Black box/white box*: Indicates whether the approach is based on black-box or white-box testing (white-box testing is typically used under system development)
- *Degree of structure*: Indicates how structured the approach is. Three values are possible: low, medium, and high.
- *System specification language*: Indicates whether the approach is supported by any particular system specification languages.
- *Risk specification language*: Indicates whether the approach is supported by any particular risk analysis languages.
- *Test specification language*: Indicates whether the approach is supported by any particular test specification languages.
- *Tool support*: Indicates whether the approach is supported by a tool.
- *System to risk*: Indicates whether the approach is based on starting with the risk and identifying system parts that needs to be tested, or based on starting with the systems parts and indentifying risks for those.
- *Test case derivation*: Indicates whether or not the approach describes how test cases can be derived from a risk model.

Table 1 Risk-based testing approaches

Method/attributes	**Security specific**	**Black-box/white box**	**Degree of structure**	**System spec. language**	**Risk spec language**	**Test spec language**	**Tool support**	**System to risk**	**Test case derivation**
Prisma	No	Both	Medium	No	Tables	No	Yes (for prioritizing)	Yes	No
Redmill	No	White box/under development	Low	No	No	No	No	Yes	No
Amland	No	White box /under development	Medium	No	Tables	No	No	Yes	No
Bach	No	Both	Low	No	Tables	No	No	Both	No
RiteDAP	No	White box /under development	Low (for risk analysis part)	No	Yes	Yes	No	Yes	Yes

Rosen-berg et al	No	White box /under de-velopment	Low	No	No	No	No	Yes	No
Murthy et al.	Yes	White box/under dev	Medium	No	No	No	No	No	No
Zach	Yes	White box	Low	No	No	No	No	No	No

Based on the state-of-the-art analysis, we conclude that risk-based testing is still an immature research area with a many opportunities for further development. In particular,

- Most approaches to risk-based testing target functional testing, only 2 out of 8 consider security.
- Few approaches to risk-based to risk-based testing are sufficiently documented and structured to serve as an easy to use industrial-setting process. For instance, many of the approaches read more like a general discussion on the topic risk-based testing than a detailed description of steps involved in carrying out the process/approach.
- Only 1 out of 8 approaches are supported by tools.
- Few of the approaches are supported by any particular languages for specifying systems, tests, or risk-analysis results.

In the following we give a summary of the approaches.

3.2.1 The Prisma method

The Prisma method is presented in a white paper [48]. The method suggests prioritizing based on the most important areas of a product, and the part of the product which is likely to have the most defects. Areas with most defects are likely to be: complex areas (>200 complexity measures exist), changed areas, areas with new technology or methods, areas developed by inexperienced people involved, areas developed under time pressure and high defect history.

The prisma method has 5 steps:

- *Planning:* Gather input documents, identify risk items (i.e. parts of the systems that may impact risks) through interview and reading documents, determine impact and likelihood factors, define a weight for each factor, select stakeholders.
- *Individual preparation:* Each participant assigns values to factors per risk item. The participants score by selecting (the description of) the value that fits (supported by Excel sheets)
- *Gather individual scores:* Do an initial check to see of the scores are OK, then process individual scores
- *Define a differentiated test approach:* Prioritize risk items than need to be tested based on location in risk matrix.

3.2.2 Redmill's appraoch

Redmill has written two papers on risk-based testing. In the first paper [43], Redmill argues that risk-based testing is not consistently defined nor supported by literature on either theory or practice, and proceeds with an informal discussion of risk-based testing. In his second paper [44], Redmill presents three approaches to risk-based testing

- *Single-factor analysis: Consequence*. This approach is based on estimating consequence for risks values only (not likellihood). The steps of the approach are:
  - o Consequence identification: For each service and each stakeholder, determine consequence of failure.
  - o Consequence analysis 1: Use Hazop guidewords to distinguish between the different ways a service may fail.
  - o Consequence analysis 2: Distinguish between services, the functions that provide the services, and the software items of which the functions are composed. Find the relationship between these types of entities to determine the relationship between services and software items. Identify potential causal links between software items and various types of service failure.
  - o Assessment on the basis of consequence: Determine how the consequence value is translated into test plans for software items. Assign a category to each software item and then use this category to define an appropriate test programme.
- *Single-factor analysis: Probability.* This approach is based on estimating the likelihood of risks. The steps of the approach are:
  - o Relevant factors and their attributes: For each software item and each relevant quality attribute (such as complexity, structure, comments), assign a quality level.
  - o Using the information: Each software item's quality factor is used to inform decisions on what risk-management action to take. One approach is for quality factors to be equated with test programmes.
- *Two-factor analysis.* This approach is based on based on the combination of the previously mentioned approaches in which software items are placed in a matrix of intergrity level and quality factors
  - o Combine the two previous mentioned approaches and place software items in a matrix of integrity levels and quality factors.

3.2.3 Amland – Risk-based testing

Åmland describes a six step process which has been applied to a financial application case study [50]. The steps of the method are the following:

- Planning (risk identification/risk strategy): Define the test item tree, i.e. a hierarchical breakdown of the functions and features in the system to be tested. Establish test plan and overall risk strategy.
- Identify risk indicators (risk assessment): Define a set of indicators (e.g. size of function, number of changes since previous release, complexity) that can be used to assess the probability of failure of a function. Then for each function, and each indicator, assign an indicator value. Then use this information to calculate the probability of failure for each function.
- Identify cost of fault (risk assessment): For each function, estimate the consequence of failure.
- Identify risk elements (risk assessment): Calculate a risk expose for each function based on estimated probability and consequence of failure. Use this to prioritize the functions.
- Test execution (risk mitigation): Start testing based on the prioritized list of functions.

3.2.4 Back – Heuristic Risk-Based Testing

Bach defines risk based testing as the following process [42]:

- Make a prioritized list of risks.
- Perform testing that explores each risk.
- As risks evaporate and new one emerge, adjust your test effort to stay focused on the current crop.

Back then proceeds by presenting two different approaches to risk-based testing. One approach is based on starting with the system and then identifying the risks based on an identification of find vulnerabilities, threats, and victims. The other approach is based on starting with the risks to which parts of the system they apply to. Bach suggest that the latter approach could be aided by three kinds of lists: A list of quality criteria categories, a generic risk list, and a risk catalog (containing risks that belong to particular domains)

3.2.5 RiteDAP

RiteDAP [46] is an approach to risk-based testing that allows for the automatic derivation of system test cases from activity diagrams as well as their prioritization based on risk. The RiteDAP process has the following steps:

- Specify activity diagrams which can be used as test models.
- Annotate the activity diagrams with risks.
- Derive a set of unordered test case scenarios form the test model.
- Order the test scenarios based on the risk information in the test model.

The last two steps can be automated.

3.2.6 Rosenberg et.al. Risk-based Object Oriented Testing

Rosenberg et. al. [45] proposes measurement criteria that can be used to estimate the complexity of object-oriented code. It is suggested that a high-level of complexity could imply a high likelihood of failure. Consequence analysis is not considered.

3.2.7 Murthy et al. – Leveraging Risk Based Testing in Enterprise Systems Security Validation

Murphy et. al. [47] proposes an iterative process to risk based testing that consists of the activities:

- Risk analysis: In this activity, risks are identified and evaluated according to the Risk Analysis method proposed by NIST. The output of the activity is a documented list of high level risks.
- Threat modeling: In this activity, risks are detailed by modeling threats to the system under evaluation according to Microsoft's Threat Modeling process. The system is then decomposed into sub-systems to identify assets, entry points, and trust levels. The output of the activity is a detailed list of categorized risks.
- Test design: In this activity, misuse cases are used to identify security test scenarios. Also, any Security Controls suggested as part of the application design will automatically translate into a test scenario.
- Test execution: In this activity, test scenarios are translated into more detailed test cases which are categorized an prioritized according to the risk of categorized risks.
- Reporting: In this activity, the output of the test execution is captured that details the vulnerabilities found along with their severity level.

3.2.8 Zech – Risk-Bases Security Testing in Cloud Computing Environments

Zach proposes a model-driven methodology for the security testing of cloud environments [49]. The main steps of the method are:

- Step 1: Perform a risk analysis of the Cloud Under Test (CUT), possibly with the help of a vulnerability repository.
- Step 2: Transform the risk model (generated in step I) into a set of negative requirements (more or less textual descriptions) using a model to model (M2M) transformation.
- Step 3: Transform the negative requirements into misuse cases using an M2M transformation.
- Step 4: Automatically transform the misuse cases into test cases using an M2M transformation

The transformations mentioned in the steps are not presented in detail; it merely suggests that such transformations could be used.

4. CONCLUSIONS

In this document, we have presented state-of-the-art related to methodologies for risk- and model-based security testing. The state-of-the-art is related to techniques developed in WP2 that should be supported by methodologies and whose state-of-the-art survey has not already been covered by deliverable D1.WP2.

5. REFERENCES

[1] H. Götz, M. Nickolaus, T. Roßner, K. Salomon, “Model Based Testing - Modelling and generation of tests - basics, criteria for tool use, tools in the overview” (in German), iX Studie, 01/2009

[2] [67] Jürjens, J.: *Secure Systems Development with UML*, Springer, 2005

[3] Jürjens, J.; Schreck, J. & Yu, Y.: *Automated Analysis of Permission-Based Security Using UMLsec; Fundamental Approaches to Software Engineering*, 11th International Conference (FASE), Springer, 2008, 4961, 292-295

[4] [65]Jürjens, J. Jézéquel, J.-M.; Hussmann, H. & Cook, S. (Eds.) *UMLsec: Extending UML for Secure Systems Development*; The Unified Modeling Language, Springer Berlin / Heidelberg, 2002, 2460, 1-9

[5] [77] Lodderstedt, T.; Basin, D. A. & Doser, J. Jézéquel, J.-M.; Hußmann, H. & Cook, S. (Eds.) *SecureUML: A UML-Based Modeling Language for Model-Driven Security*; The Unified Modeling Language, 5th International Conference, Springer, 2002, 2460, 426-441

[6] L. Cholvy, F. Cuppens, Analyzing consistency of security policies, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 1997, pp. 103-112.

[7] Amel Mammar, Wissam Mallouli, and Ana Cavalli. *A systematic approach to integrate common timed security rules within a TEFSM-based system specification*. Publiched in Information and Software Technology Journal. ISSN = 0950-5849, Elsevier Editor, August 2011

[8] A. Cavalli, D. Lee, C. Rinderknecht, F. Zaidi, Hit-or-jump: An algorithm for embedded testing with applications to in services, in: J. Wu, S. Chanson, Q. Gao (Eds.), Formal Methods for Protocol Engineering and Distributed Systems(FORTE), Vol. 156 of IFIP Conference Proceedings, Kluwer, 1999, pp. 41-56.

[9] M. Bozga, J. Fernandez, L. Ghirvu, S. Graf, J. Krimm, L. Mounier, J. Sifakis, IF: An intermediate representation for SDL and its applications, in: Proceedings of SDL Forum, Elsevier, 1999.

[10] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting, “A subset of precise UML for model-based testing,” in *3rd int. Workshop on Advances in Model Based Testing*, 2007, pp. 95–104.

[11] The Common Criteria Recognition Arrangement. Part 3: Common Criteria Evaluation Methodology http://www.commoncriteriaportal.org/ccra/

[12] TVRA ETSI TS 102 165-1 v4.2.3 (2011-03) TISPAN Methods and Protocols; Part 1 Meth-

od and Proforma for Threat, Vulnerability Analysis; eTVRA: http://portal.etsi.org/eTVRA/

[13] H. Abie, D. B. Aredo, T. Kristoffersen, S. Mazaher, and T. Raguin. Integrating a security requirement language with UML. In Proc. of the 7th International Conference on The Unified Modelling Language: Modelling Languages and Applications (UML’04), volume 3273 of Lecture Notes in Computer Science, pages 350–364. Springer, 2004.

[14] K. Alghathbar and D. Wijesekera. authUML: a three-phased framework to analyze access control specifications in use cases. In Proc. of the 2003 ACM workshop on Formal methods in security engineering (FMSE’03), pages 77–86. ACM, 2003.

[15] K. Alghathbar and D.Wijesekera. Consistent and complete access control policies in use cases. In Proc. of the 6th International Conference on The Unified Modeling Language, Modeling Languages and Applications (UML’03), volume 2863 of Lecture Notes in Computer Science, pages 373–387. Springer, 2003.

[16] D. Basin, J. Doser, and T. Lodderstedt. Model driven security for processoriented systems. In Proc. of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT’03), pages 100–109. ACM, 2003.

[17] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. ACM Transactions on Software Engineering Methodologies, 15(1):39–91, 2006.

[18] D. A. Basin, M. Clavel, J. Doser, and M. Egea. A metamodel-based approach for analyzing security-design models. In Proc. of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS’07), volume 4735 of Lecture Notes in Computer Science, pages 420–435. Springer, 2007.

[19] R. Breu, M. Hafner, B. Weber, and A. Novak. Model driven security for interorganizational workflows in e-government. In Proc. of the 2005 International Conference on E-Government: Towards Electronic Democracy (TCGOV’05), volume 3416 of Lecture Notes in Computer Science, pages 122–133. Springer, 2005.

[20] C. C. Burt, B. R. Bryant, R. R. Raje, A. M. Olson, and M. Auguston. Model driven security: unification of authorization models for fine-grain access control. In Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC’03), pages 159–173. IEEE Computer Society, 2003.

[21] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. MAC and UML for secure software design. In Proc. of the 2004 ACM workshop on Formal methods in security engineering (FMSE’04), pages 75–85. ACM, 2004.

[22] P. Epstein and R. Sandhu. Towards a UML based approach to role engineering. In Proc. of the 4th ACM workshop on Role-based access control (RBAC’99), pages 135–143. ACM, 1999.

[23] E. Fernandez-Medina and M. Piattini. Extending OCL for secure database development. In Proc. of the 7th International Conference on The Unified Modelling Language: Modelling Languages and Applications (UML’04), volume 3273 of Lecture Notes in Computer Science, pages 380–394. Springer, 2004.

[24] E. Fern´andez-Medina and M. Piattini. Designing secure databases. Information & Software Technology, 47(7):463–477, 2005.

[25] E. Fern´andez-Medina, J. Trujillo, R. Villarroel, and M. Piattini. Extending UML for designing secure data warehouses. In Proc. of the 23rd International Conference on Conceptual

Modeling (ER'04), volume 3288 of Lecture Notes in Computer Science, pages 217–230. Springer, 2004.

[26] E. Fern´andez-Medina, J. Trujillo, R. Villarroel, and M. Piattini. Developing secure data warehouses with a UML extension. Information Systems, 32(6):826–856, 2007.

[27] T. Fink, M. Koch, and K. Pauls. An MDA approach to Access Control Specifications Using MOF and UML Profiles. Electronic Notes in Theoretical Computer Science, 142:161–179, 2006.

[28] P. Giorgini, F. Massacci, and J. Mylopoulos. Requirement engineering meets security: a case study on modelling secure electronic transactions by VISA and mastercard. In Proc. of the 22nd International Conference on Conceptual Modeling (ER'03), volume 2813 of Lecture Notes in Computer Science, pages 263–276. Springer, 2003.

[29] M. Hafner, M. Alam, and R. Breu. Towards a MOF/QVT-Based Domain Architecture for Model Driven Security. In Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06), volume 4199 of Lecture Notes in Computer Science, pages 275–290. Springer, 2006.

[30] R. Heldal and F. Hultin. Bridging Model-Based and Language-Based Security. In Proc. of the 8th European Symposium (ESORICS'03), volume 2808 of Lecture Notes in Computer Science, pages 235–252. Springer, 2003.

[31] M. Koch and F. Parisi-Presicce. Formal access control analysis in the software development process. In Proc. of the 2003 ACM workshop on Formal methods in security engineering (FMSE'03), pages 67–76. ACM, 2003.

[32] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono. Model-Driven Security Based on a Web Services Security Architecture. In Proc. of the 2005 IEEE International Conference on Services Computing (SCC'05), pages 7–15. IEEE Computer Society, 2005.

[33] A. Poniszewska-Maranda, G. Goncalves, and F. Hemery. Representation of extended RBAC model using UML language. In Proc. of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05), volume 3381 of Lecture Notes in Computer Science, pages 413–417. Springer, 2005.

[34] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In Proc. of the 9th ACM symposium on Access control models and technologies (SACMAT'04), pages 115–124. ACM, 2004.

[35] E. Soler, J. Trujillo, E. Fern´andez-Medina, and M. Piattini. A Framework for the Development of Secure Data Warehouses based on MDA and QVT. In Proc. of the International Conference on Availability, Reliability and Security (ARES'07), pages 294–300. IEEE Computer Society, 2007.

[36] B. Vela, E. Fern´andez-Medina, E. Marcos, and M. Piattini. Model driven development of secure XML databases. SIGMOD Record, 35(3):22–27, 2006.

[37] J. L. Vivas, J. A. Montenegro, and J. Lopez. Towards a business process-driven framework for security engineering with the UML. In Proc. of the 6th International Conference on Information Security (ISC'03), volume 2851 of Lecture Notes in Computer Science, pages 381–395. Springer, 2003.

[38] MILLER, B., KOSKI, D., LEE, C. P., MAGANTY, V., MURTHY, R., NATARAJAN, A., AND STEIDL, J. Fuzz revisited: A reexamination of the reliability of UNIX utilities and services. Tech. rep., 1995.

[39] MILLER, B. P., FREDRIKSEN, L., AND SO, B. An empirical study of the reliability of UNIX utilities. Communications of the Association for Computing Machinery 33, 12 (1990), 32–44.

[40] OULU UNIVERSITY SECURE PROGRAMMING GROUP. PROTOS. http://www.ee.oulu.fi/research/ouspg/protos/.

[41] [4] OULU UNIVERSITY SECURE PROGRAMMING GROUP. RADAMSA. http://code.google.com/p/ouspg/wiki/Radamsa

[42] J. Bach, Heuristic Risk-Based Testing, Software Testing and Quality Engineering Maga-zine, No-vember 1999, pp. 96-98.

[43] F. Redmill, Exploring Risk-based Testing and Its Implications 1. Software Testing, Verification & Reliability, 14(1), 2004

[44] F. Redmill, Theory and practice of risk-based testing. Software Testing, Verification & Reliability, 15(1), 2005

[45] L. H. Rosenberg, R. Stapko, and A. Gallo. Risk-based object oriented testing. In: Proceed-ings of the 24th annual Software Engineering Workshop, NASA, Software Engineering La-boratory, 1999.

[46] H. Stallbaum, A. Metzger, and K. Pohl. 2008. An automated technique for risk-based test case generation and prioritization. In Proceedings of the 3rd international workshop on Au-tomation of software test (AST '08). Pp. 67-70.

[47] K.K. Murthy, K.R. Thakkar, S. Laxminarayan. Leveraging Risk Based Testing in Enterprise Sys-tems Security Validation, First International Conference on Emerging Network Intelli-gence, pp.111-116, 11-16 Oct. 2009

[48] E. van Veendendaal, Practical Risk-Based Testing – Product RISk Management: the PRISMA me-thod, 2009, http://www.erikvanveenendaal.nl/NL/files/PRISMA%20white%20paper%201.4.pdf

[49] P. Zech, Risk-Based Security Testing in Cloud Computing Environments, 2011 IEEE Fourth Inter-national Conference on Software Testing, Verification and Validation (ICST), pp.411-414, 2011

[50] S. Åmland: Risk-based testing: Risk analysis fundamentals and metrics for software testing includ-ing a financial application case study. Journal of Systems and Software 53(3): 287-295 2000

[51] Wissam Mallouli, Bachar Wehbi, Ana Cavalli and Stéphane Maag, Formal Supervision of Mobile Ad hoc Networks for Security Flaws Detection. Book chapter in "Security Engineering Techniques and Solutions for Information Systems: Management and Implementation", Editors Pr. Noureddine Boudriga and Pr. Mohamed Hamdi. Publisher: Information Science Reference - IGI Global. ISBN: 9781615208036. May 2011.

[52] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, Gilles Trouessin, Organization based access control, Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, 2003

[53] International Society for Optical Engineering, Citeseer, 2006, 6242, 114-129, Telecommunication Union. Recommendation Z.100 — Specification and description language (SDL), 1999

[54] F.Cuppens, N.Cuppens, T.Sans, Nomad: A Security Model with Non Atomic Actions and

Deadlines, Proceedings of the 18th IEEE workshop on Computer Security Foundations, 2005

[55] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet and M. Utting. A subset of precise UML for model-based testing. In A-MOST'07: Proceedings of the 3rd international workshop on Advances in model-based testing, pages 95-104. 2007

[56] "Weka 3: Data mining software in java," (accessed 9/29/2011). [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

[57] "Network measurement and accounting system," (accessed 9/29/2011).[Online]. Available: http://www.ip-measurement.org/tools/netmate/

[58] "The network traffic based application identification,"(accessed 9/29/2011). [Online]. Available: http://caia.swin.edu.au/urp/dstc/netai/netai.html

[59] "Netmate-flowcalc," (accessed 9/29/2011). [Online]. Available: http://web.cs.dal.ca/darndt/projects/netmate-flowcalc/

[60] "GCC - the GNU Compiler Collection", (accessed 9/29/2011) [Online]. http://gcc.gnu.org/