

	<p align="center">Initial Model-Based Security Testing Methods</p> <p align="center">Deliverable ID: D3.WP2</p>	Page : 1 of 46
		Version: 1.13
		Date : 06.07.2012
		Status : Final Confid : Public

	Title: Initial Model-Based Security Testing Methods	
	Version: 1.13 Date : 06.07.2012 Pages : 46	
	Editor: Franz Wotawa	
	Reviewers:	
	To: DIAMONDS Consortium	
<p>The DIAMONDS Consortium consists of: Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Giesecke & Devrient, Grenoble INP, IT SudParis,itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, Trusted Labs, TU Graz, University Oulu, VTT.</p>		
Status: <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input type="checkbox"/> Proposal <input checked="" type="checkbox"/> Final / Released	Confidentiality: <input checked="" type="checkbox"/> Public Intended for public use <input type="checkbox"/> Restricted Intended for DIAMONDS consortium only <input type="checkbox"/> Confidential Intended for individual partner only	

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 2 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

Deliverable ID: D3.WP2

Title:

Concepts for Model-Based Security Testing

Summary / Contents:

This document describes the initial model-based security testing methods used within the DIAMONDS project.

Contributors to the document:

Wissam Mallouli (Montimage), Edgardo Montes de Oca (Montimage), Bachar Wehbi (Montimage), Fabien Duchène (Grenoble INP), Roland Groz (Grenoble INP), Laurent Mounier (Grenoble INP), Sanjay Rawat (Grenoble INP), Jean-Luc Richier (Grenoble INP), Josip Bozic (TU Graz), Franz Wotawa (TU Graz), Nikolay Tcholtchev (FOKUS), Sami Noponen (VTT), Anu Phakainen (Ericsson), Mika Rautila (VTT), Ikka Uusitalo (VTT), Felix Jakob (Dornier Consulting), Andreas Schulze (Dornier Consulting), Julien Botella (Smartesting), Julien Botella (Smartesting)



	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 3 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public


TABLE OF CONTENTS

Introduction	7
1. Monitoring and Inspection.....	8
1.1. Multi data source security monitoring approach using MMT	8
1.1.1 Global security monitoring approach using MMT	8
1.1.2 Multi data sources management for security analysis	9
1.1.3 Security properties including “distributed” events	9
1.2 Machine learning approach for Network monitoring in ICS network.....	11
1.2.1 Introduction	11
1.2.2 Machine Learning	12
1.2.3 Industrial Control System Network Specifics	12
1.2.4 Requirements for IDS in ICS Network	14
2. Active Testing.....	15
2.1 Model-Based Security Testing from behavioral models and security-oriented Test Purposes	15
2.2 Vulnerability Directed Input Generation	17
2.2.1 Introduction	17
2.2.2 Model Inference Assisted Evolutionary Fuzzing	17
2.2.3 Static and Dynamic Code Analysis Assisted Fuzzing	21
2.3 Improving the Efficiency of UML Sequence Diagram Fuzzing	25
2.3.1 Status Quo	25
2.3.2 Problem: Explosion of Generated Test Cases	25
2.3.3 Proposed Problem Solution: UMLsec.....	26
2.4 Fuzzing operator in test cases	30
2.4.1 Introduction	30
2.4.2 Applying Fuzzing operators	30
3. Risk Analysis for Risk Based Testing.....	34
3.1 Traceability Methodology	34
3.1.1 General Concept of Software Traceability	34
3.1.2 General Concept of Software Traceability	34
3.1.3 Traceability in Safety	34
3.1.4 The DIAMONDS Traceability Concept	35
3.1.5 Basic off-the-shelf Traceability Model.....	35
3.1.6 Description of the Target Realization of the Trace Management Platform (TMP)	36
3.1.7 Application to the Banking Domain Case Study (G&D).....	39
3.2 Mirroring Risk Analysis Techniques to Telecom Use Case	40
3.2.1 Existing Risk Assessment Technique.....	40
3.2.2 Mirroring techniques to telecom use case	41
3.2.3 Risk modelling techniques	41
3.2.4 Risk assessment based on vulnerability analysis.....	42
3.2.5 Risk related testing techniques.....	42
3.2.6 Conclusions	43
Glossary (adapted from [CNSS 4009])	44
REFERENCES.....	45

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 4 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

FIGURES

Figure 1. MMT global architecture.....	8
Figure 2. MMT security property structure	10
Figure 3. A distant Link Spoofing Attack on OLSR.	11
Figure 4. ICS Network	13
Figure 5. Functional differentiation	14
Figure 6. Smartesting process for model-based security testing	15
Figure 7. Overview of the proposed approach, showing the main three components.....	17
Figure 8. Reflected XSS detection and exploit process	18
Figure 9. An example of the pattern for reflected XSS.....	19
Figure 10. Input Grammar to generate fuzzed inputs (excerpt)	20
Figure 11. Fitness Function for XSS specific input generation	21
Figure 12. Self-Dependency Chains corresponding to Pattern A	22
Figure 13. Extended Self-Dependency Chain corresponding to Pattern B.....	22
Figure 14. Dataflow equation for the VSA analysis	23
Figure 15. Transfer Function for the effect of instructions.....	24
Figure 16. Combining dynamic and static taint flow	25
Figure 17. Proposed NIST Standard for Role-Based Access Control [20]	27
Figure 18. Simple Example of an Activity Diagram with the UMLsec <i>rbac</i>	27
Figure 19. Test Case from the Activity Diagram in Figure 18.	29
Figure 20. The do.ATOMS framework	30
Figure 21. The do.ATOMS framework	31
Figure 22. A fuzzing container applied to the message 'EnterPIN_I4'	32
Figure 23. Concept overview.....	33
Figure 24. The Traceability Meta-Model.....	35
Figure 25. Step 1 - TMP Overview with belonging Tools, Models and Test Notations	37
Figure 26. Step 2 - TMP Overview with Domains of Relevance for Model Based Security Testing	37
Figure 27. TMP Overview regarding the Domains involved in Model Based Security Testing	38
Figure 28. Query Interface Draft.....	39


	<p>Initial Model-Based Security Testing Methods</p> <p>Deliverable ID: D3.WP2</p>	Page : 5 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

HISTORY

Vers.	Date	Author	Description
0.1	2012/05/07	F. Wotawa	Content integrated
0.1	2012/05/07	J. Bozic	Document formating
0.1	2012/05/08	F. Wotawa	Introduction and executive summary added
1.0	2012/05/08	F. Wotawa	Version for review
1.1	2012/07/05	F. Jakob	Version after review (Caption fixed, etc.)

APPLICABLE DOCUMENT LIST

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

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 6 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

EXECUTIVE SUMMARY


DIAMONDS considers the particular issue of security testing of networked systems to validate the dependability of networked systems in face of malice, attack, error or mischance. Testing is still the main method to reliably check the functionality, robustness, performance, scalability, reliability and resilience of systems as it is the only method to derive objectively characteristics of a system in its target environment. A number of approaches have long been around to target specific attacks on systems (e.g. vulnerability scanners), but when we refer here to the more systematic testing of systems with respect to specified policies or security properties, testing a system for its security is a relatively new concern that has started to be addressed in the last few years.

D1.WP2 reviewed the state-of-the-arts methods used in security testing. D2.WP2 described the main concepts dedicated to model-based security testing used by the different project partners. In this document (D3.WP2) more concrete model-based security testing methods are discussed and introduced. This deliverable is intended to be the continuation of previous work and describes some of the advances performed in WP2. We follow the structure of the previous deliverables and divide the content into three parts: Monitoring and inspection, active testing and risk analysis for risk based testing.

Monitoring, which is also named passive testing, is for detecting faults in a system under test during execution time. For this purpose the inputs and outputs are observed. The observed behaviour together with a specification of the expected behaviour is used to distinguish the failing run from a passing run. The classical monitoring approach makes use of traces obtained during system execution. The monitoring approaches explore relevant properties and check them on the execution trace. In Chapter 1 we describe the initial monitoring testing methods to be used by the DIAMONDS partners.

Active testing in contrast to passive testing usually comprises the generation of test cases, i.e., a sequence of inputs, which are used to stimulate the system under test. The output of the system under tests is then compared with the expected outputs also specified in a test case. In most cases the tests are generated automatically from formal models. Most of the works are on testing software control parts and technologies are related to the approaches of model-based testing. Challenges of the active testing method are state explosion, modelling, fault coverage and the minimization of the generated test suite. In Chapter 2 we describe the DIAMONDS approaches to active testing.

The systematic, efficient and goal-oriented identification, selection and execution of tests in order to focus on relevant faults and situation is important. Risk-oriented testing that selects tests based on identified risks is a valuable technique to guide the whole testing process in the right direction. Risk-based testing can be used for the selection and prioritization of test cases. In Chapter 3 we discuss and introduce methods that are related to risk-based testing.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 7 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

INTRODUCTION

Checking the functionality, robustness, performance, scalability, reliability and resilience of systems it is undoubted of greatest importance to ensure trust in the behaviour of the systems. Testing is the main method to objectively derive such systems' characteristics when run in its target environment. Formal verification can hardly serve this purpose because of complexity that comes both from the system and its interaction with its environment. In case of distributed systems where information is shared using different protocols, different programs that run on different computers with different operating systems installed this situation becomes even worse and new issues like security come into the game.


Security testing is for checking security properties like confidentiality, integrity, authentication, authorization, availability and non-repudiation. In case of security testing two streams can be identified. Functional security testing deals with checking the security functionality like authentication directly, whereas security vulnerability testing addresses the discovery of new system vulnerabilities coming from security design flaws. Functional security testing can also be more or less seen as checking the system for ensuring that no known vulnerabilities are present. Therefore, model-based approaches that make use of models of either attacks or security properties to be implemented can be used. Security vulnerability testing may also rely on models. However, deviations of the inputs introduced when using mutations, random testing approaches or negative tests (which might also be obtained from models) have to be used in order to stimulate the system in a way that reveals a new vulnerability.

In this document the initial model-based security testing methods of the DIAMONDS partners are documented. This report is organized as follows:

In Chapter 1 we discuss monitoring approaches. There the authors introduce and describe a security monitoring approach integrating multiple data sources. The approach is based on security properties that are represented as a tree. Beside this approach the authors also introduce the application of machine learning for security testing of industrial control systems. Here the idea is to use machine learning to distinguish ordinary system accesses from intrusions.

Chapter 2 deals with active testing. The chapter comprises methods for security testing based on security properties. Besides discussing the overall active security testing methodology we describe foundations for vulnerability detection based on models and fuzzing. In addition we introduce an approach that improves the efficiency of UML fuzzing. In the last part we discuss the mutation of test cases.

In Chapter 3 we discuss applications to risk analysis and risk based testing. In the first part we tackle the problem of traceability, which is of especial importance in case of security. Hence, we introduce the DIAMONDS traceability concept. In the second part we discuss risk analysis techniques for the telecom use case.

	<p align="center">Initial Model-Based Security Testing Methods</p> <p align="center">Deliverable ID: D3.WP2</p>	Page : 8 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

1. MONITORING AND INSPECTION

1.1. MULTI DATA SOURCE SECURITY MONITORING APPROACH USING MMT

1.1.1 Global security monitoring approach using MMT

The Montimage Monitoring Tool (MMT) [1] is an event-based monitoring solution that allows analysing network traffic according to a set of functional and non-functional properties. It does near real-time QoS and security analysis based on deep packet inspection techniques. With respect to security, it uses model-based description of sequences of events expressed in XML (called security properties) that allow defining security rules (i.e., rules that should be respected) or attacks and misbehaviours.

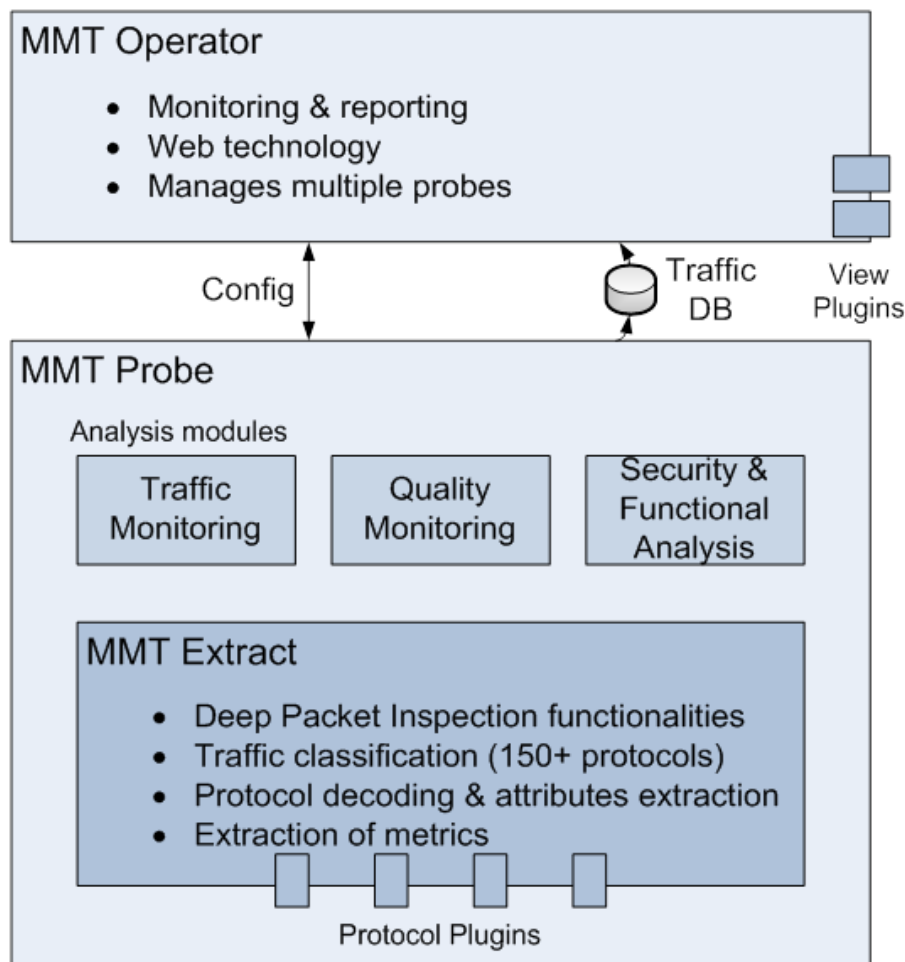



Figure 1. MMT global architecture

MMT can be delivered and installed as (1) a standalone tool that allows the analysis of live or pre-recorded structured data or as (2) a set of two libraries for integration into third party probes. The monitoring solution can be comprised of several probes monitoring different network interfaces with a central application correlating information to obtain a more global view of what is happening in the network. Network traffic analysis for security follows four steps:

- The definition of the monitoring architecture: This architecture depends on the nature of the system under test and its deployment in the network. Capture engines (i.e. probes) are placed at relevant

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 9 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

elements or links in the network to obtain real time visibility of the traffic to be analysed. In the case of a distributed architecture, local traces are correlated (based on event timestamps) to obtain a broader visibility of what is going on in the network.

- The description of the system security goals and attacks based on the MMT security property format: The description specifies the security rules that the studied system has to respect and/or the security attacks that it has to avoid. This task can be done by an expert of the system under test that understands its security requirements in details.
- The security analysis: Based on the security property specification, the passive tester performs security analysis on the captured trace file to deduce verdicts for each property.
- The reaction: In case of a fail verdict, some reactions have to be undertaken in the network, based on previously defined security strategies, e.g. to block any malicious behaviour. The addition of reaction support in MMT is currently under development.

Passive monitoring can be coupled with active testing and fuzzing techniques. It can be applied as part of the testing chain, e.g. after the execution of some security-oriented test cases, in order to collect network traffic traces and analyse them based on previously defined MMT security properties.

1.1.2 Multi data sources management for security analysis

In the context of MMT, DPI (Deep Packet Inspection) and DFI (Deep Flow Inspection) are used to help detect and tackle harmful traffic and security threats and to throttle or block undesired behaviours. We define a set of security properties for network traffic, at both control and data levels, to catch interesting events. Indeed, based on the defined security properties, we register the attributes to be extracted from the inspected packets and flows. These attributes are of three types:


- Real attributes: They can be directly extracted from the inspected packet. They correspond to a protocol field value.
- Calculated attributes: They are calculated within a flow. Packets from the same flow are grouped and security/performance indicators are calculated (e.g. delays, jitter, packet loss rate) and made available for the security analysis engine.
- Meta attributes: These attributes are linked to each packet to describe capture information. The time of capture of each packet (timestamp attribute) is the main meta attribute in the current version of MMT.

The extracted attributes needed for security analysis can emanate from different data sources (probes and/or interfaces). This is managed in the MMT monitoring solution during the specification phase of the security properties. Indeed, the data sources identifiers are part of the meta-attributes that can be used in the specification of the relevant events for security analysis. Three architectures are taken into account in MMT:

- Local analysis: the collected traffic is analysed for security purposes in one probe that captures network traffic from one or several interfaces.
- Centralized analysis: the traffic capture is distributed but the security analysis is centralized. All data sources send their collected traffic (filtered or not) to the same master server that correlates the traces (i.e. need to synchronize probes to be able to perform this task).
- Distributed analysis: the traffic capture is distributed and the analysis is performed by all the probes that communicate together to share information. This analysis can be very interesting in some specific case studies like ad hoc networks. The communication between probes is an on-going work for MMT tool.

1.1.3 Security properties including “distributed” events

An MMT-Security property is a “general ordered tree” and can be graphically represented as shown in Figure 2 below:

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 10 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

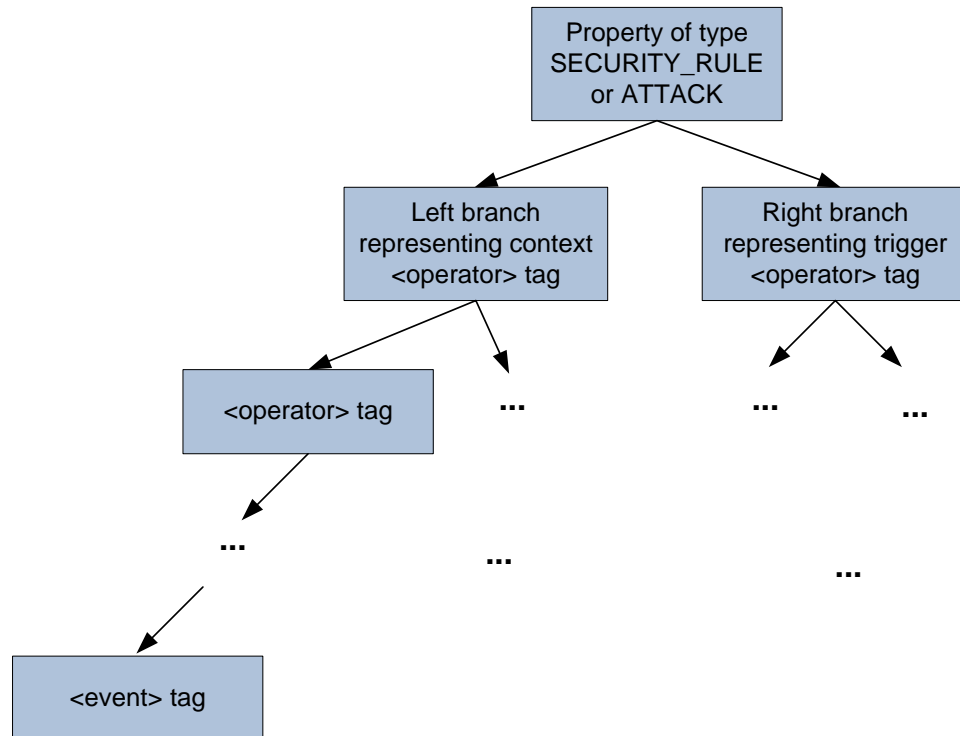


Figure 2. MMT security property structure


The nodes of the property tree are: the **property** node (required), **operator** nodes (optional) and **event** nodes (required). The property node is forcibly the root node and the event nodes are forcibly leaf nodes. The left branch represents the **context** and the right branch represents the **trigger**. This means that the property is found valid when the trigger is found valid; and the trigger is checked only if the context is valid. Properties indicate sequence of events that need to be observed. Events indicate the conditions that need to be verified on a packet for the event to hold. In the following, we specify a security that expresses a distributed network behaviour that allows detecting distant attacks on the OLSR routing protocol (related to Thales radio protocol case study). This detection can only be made through checking the global trace.

Case study:

The Optimized Link State Routing Protocol (OLSR)[2] is an IP routing protocol optimized for mobile ad-hoc networks, which can also be used on other wireless ad-hoc networks. OLSR is a proactive link-state routing protocol, which uses hello and topology control (TC) messages to discover and then disseminate link state information throughout the mobile ad-hoc network. Individual nodes use this topology information to compute next hop destinations for all nodes in the network using shortest hop forwarding paths.

Threat: A distant Link Spoofing:

One of the first properties to check is the correct logical order of HELLO messages exchange. That is a node cannot announce a symmetrical link to any neighbour without having previously received a HELLO message claiming an asymmetric link from that node. In **Figure 3**, the intruder can insert Hello messages claiming a non-existing symmetrical link to C. Consequently, the intruder might be selected as an MPR by A and the traffic from A to C will be disrupted to the intruder.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 11 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

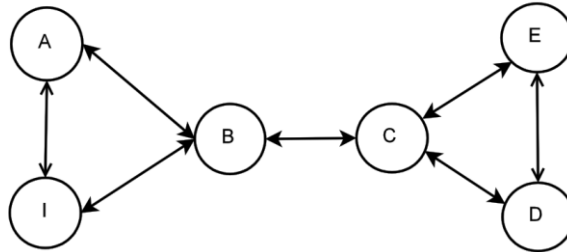


Figure 3. A distant Link Spoofing Attack on OLSR.

The resulting MMT-Security property is the following:

```
<property value="THEN" delay_max="-2" property_id="1" type_property="SECURITY_RULE"
  description="If a node I claims a symmetric link with another node C, that means that C
  sent a Hello message claiming an asymmetric link with I"
>
  <event value="COMPUTE" event_id="1"
    description="A node claims a symmetric link with another node"
    boolean_expression="((OLSR.LINK_TYPE == SYM)&&
    (BASE.DATA_SOURCE != BASE.DATA_SOURCE.2))"/>

  <event value="COMPUTE" event_id="2"
    description=" A node claims an asymmetric link with another node "
    boolean_expression="(OLSR.LINK_TYPE == ASYM)"/>
</property>
```


Events 1 and 2 have different data sources ($\text{BASE.DATA_SOURCE} \neq \text{BASE.DATA_SOURCE.2}$). That means that they are collected from different nodes. We highlight here that the use of the meta attribute `DATA_SOURCE` makes it possible expressing a distributed network behaviour that allows the detection of distant attacks. This detection can only be made through checking the global trace.

1.2 MACHINE LEARNING APPROACH FOR NETWORK MONITORING IN ICS NETWORK

1.2.1 Introduction

Intrusion detection refers to a technique developed to detect malicious network activity in a single host or in an entire network. Currently, Network Intrusion Detection Systems (NIDS) are not widely used in industrial control system (ICS) networks. This leaves a hole in the network defences of industrial sites. Having a NIDS that is able to adapt to the particular environment of an industrial site would reduce the risk of production disruptions caused by anomalous incidents on the network.

ICS networks are restricted networks, not really closed, even if we would want them to be. The firewalls are typically full of holes created for system vendors to grant them maintenance access for the devices they have supplied. The connections for vendors and other personnel requiring exceptional access to systems residing on the control typically do not follow default policies in place for connections. However, the restricted nature of the ICS network does give the NIDS developer some advantages in addition to some specifics that need to be taken into account when designing the system. Documentation should also be on a better level than in the

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 12 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

more open environments with less critical roles. If the documentation is lacking, the situation must be remedied before attempting to customize a NIDS to fit the environment in question.

1.2.2 Machine Learning


Machine learning performs well for network intrusion detection applications in a closed environment, but exposing it to the open world with varying amount of random traffic or noise, adversely affects its usability. 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. Events that cause noise include maintenance of the network, new devices being added to the network, devices that send various configuration and control messages or malfunctioning devices in the network. Depending on the frequency of these types of events, it might be reasonable to require a report of these actions to the personnel responsible for maintaining the network monitoring systems. The network administrators could then either temporarily switch off the system, manipulate its parameters or set it to a learning mode.

We are actively working on a system leveraging machine learning for intrusion detection in ICS network context. Currently we are experimenting using Bro Network Security Monitoring, Rapidminer and Simple Event Correlator with ICS network trace files. In the following section we shall shortly discuss some of the ICS network specifics. The implementation and exact approach shall be further discussed in later milestones of the work package 2.

1.2.3 Industrial Control System Network Specifics

A simplified overview of ICS network is depicted in Figure 4. Traditionally there has been an air gap between various network levels to ensure the separation of office network and production equipment. However, this approach has been phased out in favour of increased connectivity and ease of use. Additionally, commercial off-the-shelf components (COTS) have been gaining ground and found their use in factory settings. This is introducing new threats to the ICS network. One example of this is, that PC-based devices are starting to replace traditional Programmable Logic Controllers, due to cost and flexibility reasons and because their operating systems are becoming suitable for demanding real-time applications. A current trend from the device manufacturers is to add remote access connections to ICS networks which can lead into vulnerabilities too.

The increased usage of wireless connections within ICS networks also lessens the isolation of the network from the outside world. Multiple protocols and techniques are available for wireless ICS operation. System availability is critical for ICS networks. Wireless access seems a natural choice when planning heavily distributed industrial control systems such as power distribution and sensor networks, but as the transmission medium is highly accessible, special care must be put into the isolation of the system. Figure 4 depicts and simplified ICS network differentiated through functional requirements, not the segmentation. Figure 5 depicts the segmentation, with FW standing for Firewall and further for a DMZ (Demilitarized Zone, i.e., secured computer network).

	<p align="center">Initial Model-Based Security Testing Methods</p> <p align="center">Deliverable ID: D3.WP2</p>	Page : 13 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

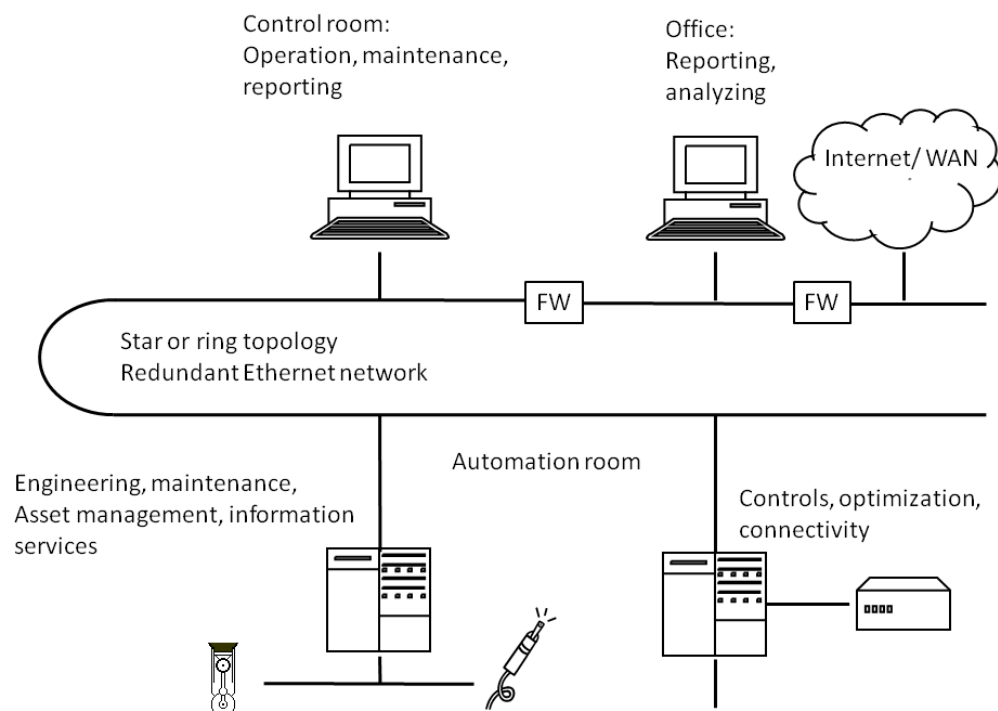


Figure 4. ICS Network

The simplified figure of ICS network depicted in Figure 5 is separated to three different levels. Both threats and attack surfaces differ between the levels of the ICS network. Therefore, the system could possibly use three different models for the different levels of factory communications as proposed by Bertoluzzo et al in [3].

- Device level
- Cell level
- Plant level

These three levels are not defence-in-depth layers. However, defence layers must be taken into account when planning for deployable system.

Device level: At the device level the real-time requirements are very strict. Typical devices doing the communicating are actuators, Programmable Logic Controllers (PLC's), sensors and such. Very little amount of data is transmitted. Data transmitted comes in two forms, periodic and cyclic data transfers that control or measure data between field equipment and alarms that interrupt the normal data flow.

Cell level: The responsibility of the cell level is to coordinate various device controllers that reside in the device level. Communication typically consists of transmissions of tens of kilobytes in structured form in a client-server model.

Plant level: Plant level interconnects the cell controllers to Enterprise Resource Planning (ERP) systems. Most of the plant level communications are performed during non-critical times, as on this level much larger amounts of data are being transmitted compared to the two previous levels.

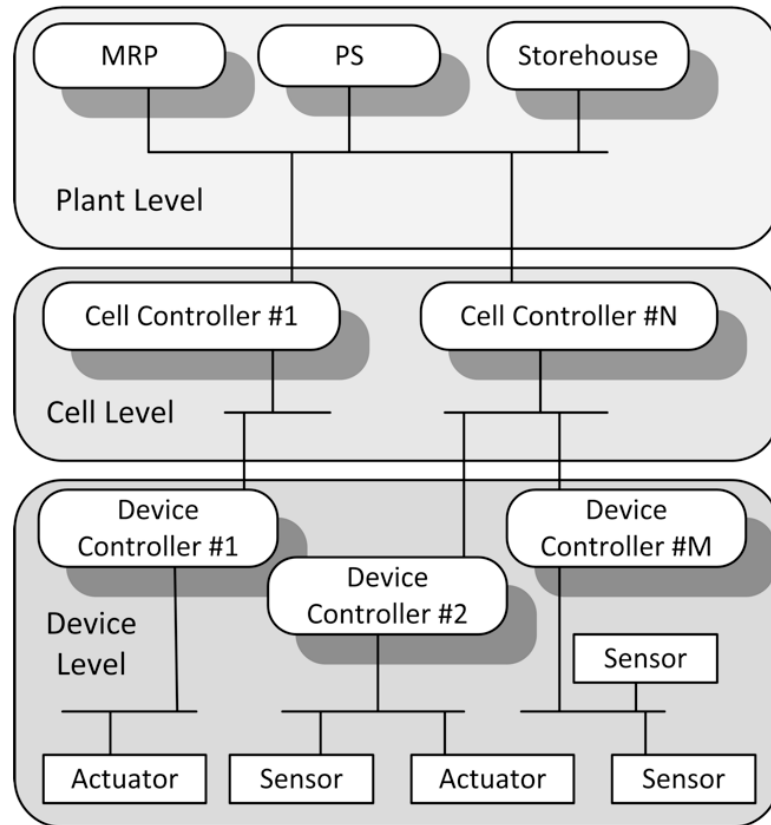



Figure 5. Functional differentiation

1.2.4 Requirements for IDS in ICS Network

The system must be solely detecting the intrusions. Intrusion prevention in ICS environment could have devastating side effects when malfunctioning. When using an Intrusion Detection System (IDS) with an Intrusion Prevention System's (IPS) functionality, all possibilities of the prevention capabilities manifesting themselves must be removed.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 15 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

2. ACTIVE TESTING

2.1 MODEL-BASED SECURITY TESTING FROM BEHAVIORAL MODELS AND SECURITY-ORIENTED TEST PURPOSES

In deliverables D2.WP2 and D2.WP3, we have presented the concepts of automated security test generation based on behavioural models and test purposes. In this deliverable, we detail this approach in terms of process, actors & roles and used artefacts. In D3.WP3, the first version of the associated tool, named Smartesting model-based security testing prototype is presented in detail.

Model-based security testing from behavioural models and test purposes is an extension of functional model-based testing (MBT):

- The model for test generation captures the expected behaviour of the system under test (SUT). This model is dedicated for automated generation of security tests and generally formalizes the security functions of the SUT but also the possible stimuli of an attacker as well as the expected answer of the SUT.
- The test purposes are test selection criteria that define the way to generate tests from the test generation model.

The main difference with “classical” functional MBT (see [4]) for a detailed introduction of functional MBT) is firstly that the behavioural model may represent stimulations that are not defined in the specification of the SUT. For example, if a security test engineer wants to generate SQL injection test, he/she would represent SQL injection operation inside the test generation model. Secondly, the model-based security testing differs in the way test cases are not directly selected from the behavioural models. In functional MBT, the way is often based on a structural coverage of the model, mixing the coverage of expected behaviour and logical test data. The tests are in general “positive”, aiming to test all the nominal cases and few (or several) error cases. In security testing, the goal is really to systematically try to break (or to bypass) the security functions of the SUT. This search for breaking software security barrier requires to systematically try possible breakers in a large set of context. This is the goal of the test purpose language to support such a test selection criteria.

The following figure presents the Smartesting process and tool for model-based security testing.

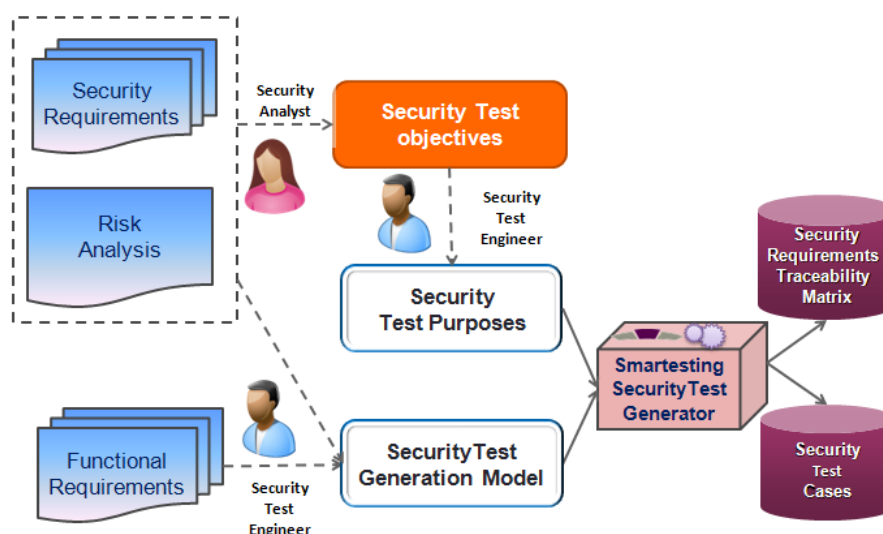



Figure 6. Smartesting process for model-based security testing

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 16 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

In this process, the following artifacts are considered:


- **Input artifacts:**
 - The *Security Requirements* and *Risk analysis* are the inputs to define the *Security Test Objectives*.
 - The *Functional Requirements*, but also the *Security Requirements* and *Risk analysis*, are the inputs to design the *Test Generation Model*.
- **Working artifacts:**
 - The *Security Test Objectives* define informally the tests to be generated to mitigate the security risks and to cover the security requirements that have been elicited. These artifacts are produced and maintained by a joint work of the Security Analyst (who knows the business risks) and the security test engineer (who knows the security testing methods).
 - The *Test Generation Model* is a formal artifact prepared and maintained by the security test engineer.
 - The *Test Purposes* are formal artifacts that define the test selection criteria to be applied on the model to generate the security tests. The Test Purposes formalize the *Security Test Objectives* in an adequate test selection language. They are developed by the Security Test Engineer.
- **Output artifacts:**
 - The *Generated Test Cases* are the result of the automated test generation phase. The *Generated Test Cases* are in general automatically translated into *Test Scripts* for test execution automation.
 - The *Security Requirements Traceability Matrix* is automatically generated and updated to exhibit the coverage of the Security Test Objectives by the Generated Test Cases.

In this process, there are two main actors: the Security Analyst and the Security Test Engineer. The following table defines the respective missions of these two actors in the Smartesting model-based security testing process.

Role	Actions	Responsibilities
Security Analyst	<ul style="list-style-type: none"> ▪ Drive the Security Requirements elicitation and Risk Analysis ▪ Pilot security testing by the risks 	<ul style="list-style-type: none"> ➤ In charge of <i>Security Requirements</i> and <i>Risk Analysis</i> ➤ Design the <i>Security Test Objectives</i>
Security Test Engineer	<ul style="list-style-type: none"> ▪ Realize security testing using model-based security testing approach 	<ul style="list-style-type: none"> ➤ Contribute to the Security Test Objectives definition ➤ In charge of the <i>Test Generation Model</i> and the <i>Test Purposes</i> creation and maintenance

This process is supported by a tool-set called Smartesting model-based security testing prototype, developed within the Diamonds process on the top of the Smartesting test generation technology. This tool-set consists of the following features:

- Support for behavioral modeling activities;
- Support for test purposes design activities;
- Automated test generator;

	Initial Model-Based Security Testing Methods Deliverable ID: D3.WP2	Page : 17 of 46 Version: 1.13 Date : 06.07.2012 Status : Final Confid : Public
----------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------

- Traceability support between test cases and security test objectives.

This tool-set and the way to use it, is presented in Diamonds deliverable D3.WP3.

2.2 VULNERABILITY DIRECTED INPUT GENERATION

In the Section 2.4 of the previous deliverable D2.WP2, entitled “Combining Model-Recovery and Evolutionary Fuzzing”, we proposed our line of work in the direction of finding vulnerabilities in software by applying the concepts of *model recovery and evolutionary fuzzing*. In this section, we unfold the terms and concepts that we over-viewed in the last deliverable, giving details on two instantiations of the concepts that we have implemented.

2.2.1 Introduction

We recall from the previous deliverable that the objective of this work is to use testing techniques to exhibit security vulnerabilities on a System Under Test (SUT) by combining three main steps:

1. model recovery, which consists in building (or retrieving) a (partial and abstract) behavioural model of the SUT;
2. vulnerability detection, by defining patterns for a particular vulnerability and finding a path to reach that vulnerability in the SUT;
3. evolutionary fuzzing, whose goal is to concretize those abstract test sequences following a mutation-based input generation guided by a fitness function.

Each of these steps is explained in the deliverable D2.WP2. In the present work, we address the problem of generating test inputs for security vulnerabilities from two different perspectives that encompass two widely applied techniques for security testing viz. black-box and white-/ gray-box testing. Each of these testing methods has its own pros and cons, depending upon the context in which it is applied. Therefore, depending on the context, one method might be more suitable than the other. From this standpoint, we use a black-box approach for web-based vulnerabilities in general and cross site scripting (XSS) in particular; and a gray-box approach for x86 executables for low level vulnerabilities, e.g. buffer overflow (BoF).

In the following sections, we explain both of the approaches in detail for the particular instances of the methods that we address in Diamonds.

2.2.2 Model Inference Assisted Evolutionary Fuzzing

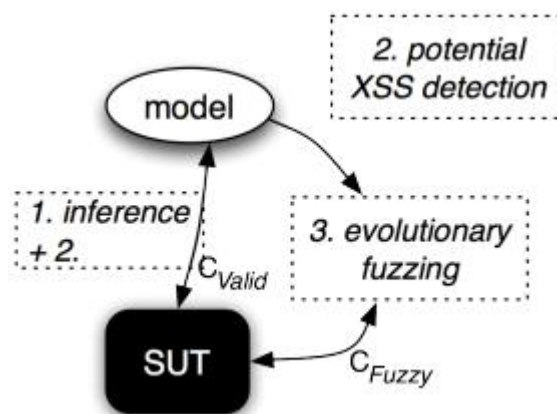



Figure 7. Overview of the proposed approach, showing the main three components

Given the complexity of modern web based applications, naive black-box fuzzing approaches may not be sufficient to detect deeply nested vulnerabilities. To address aforementioned problem, we propose to build a model of the SUT by using model inference techniques and guiding the fuzzing process by Evolutionary

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 18 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

Algorithm. **Fehler! Verweisquelle konnte nicht gefunden werden.**⁷ illustrates a high level overview of the approach. As can be seen in the figure, there are three main steps involved in this approach.

2.2.2.1 Inference

As mentioned before, the vulnerabilities that exist deep inside the application in terms of data and control flow are difficult to detect by means of input generation. One possible workaround for this problem is to have some idea about how an application consumes its inputs. In other words, what is the state transition of the application in terms of its input/output pairs? Therefore, we make use of model inference techniques to derive the state transition automaton of the SUT. We make use of a particular implementation that is developed in the SPaCloS project [14]. The technique serves our purpose well as it denotes the automaton as *Mealy machines* which represent a relationship between input/state and output. This relationship is important for us and is explained in the next section.

2.2.2.2 Vulnerability Pattern Detection:

XSS vulnerability allows an attacker to steal sensitive information from a user in web based environment. There are mainly two kinds of XSS- reflected and stored/persistent. In this work, we focus on reflected XSS. There may exist a reflected XSS if part of the request (input) is reflected in the response (output) from the web server. **Fehler! Verweisquelle konnte nicht gefunden werden.** illustrates mechanism of XSS exploit.

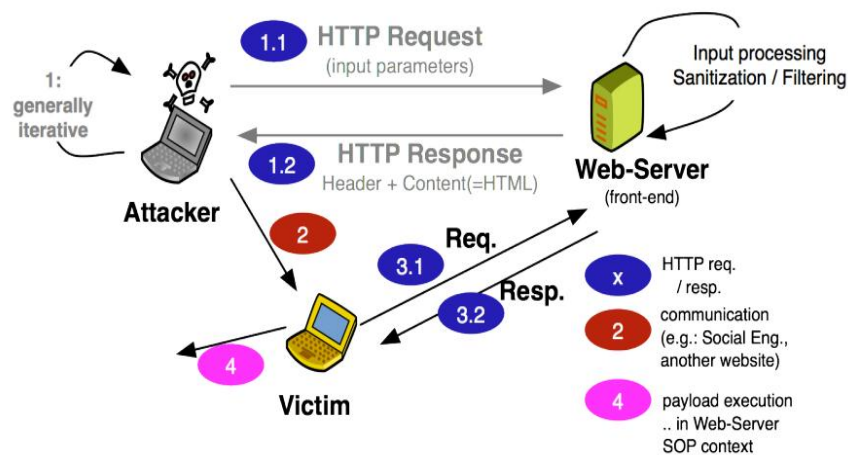



Figure 8. Reflected XSS detection and exploit process

An attacker sends a HTTP request to a web-server (1.1) and observes its output (response, 1.2). If some part of the input is present in the output, there may be a possibility of XSS. As if an attacker inserts some code that is reflected in output, the browser will try to execute the code. After this step, the attacker can send a URL with malicious code as part of the input to the victim (2). If the victim clicks on that URL, the malicious part will be reflected back to the victim's browser (3.1, 3.2) that will execute it. In this way, the attacker can execute JavaScript code to get access to sensitive data on a victim's machine (4). Therefore, we can observe that one prerequisite of reflected XSS vulnerability is that the input should be reflected back in the output. This is where the above learnt model of the SUT helps us in detecting the possibility of XSS. As the automaton is represented as a Mealy machine, we can immediately know if some part of the input is present in the next output. In this way, we can focus on the transition for which this behavior is observed. **Fehler! Verweisquelle konnte nicht gefunden werden.**⁹ shows this behavior for the transition $S2 \rightarrow S3$.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 19 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

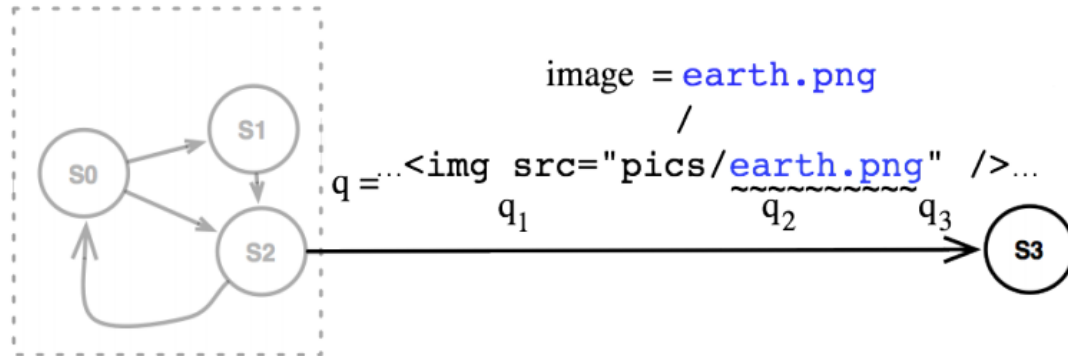


Figure 9. An example of the pattern for reflected XSS

Here the image name `earth.png` is reflected in the output at state S2.

However, it should be noted that there may be some sanitization in place at the server side which makes it impossible to insert any malicious code. Therefore, in order to validate the presence of XSS vulnerability, we are required to generate inputs with various malicious cases that trigger the vulnerability. In other words, we need to generate malicious inputs to be executed on the victim's side, including the case when filters are also present to sanitize the inputs at the server side. This is where we introduce the notion of evolutionary fuzzing, which is explained in the next section.

2.2.2.3 Evolutionary Fuzzing:


Evolutionary fuzzing is a form of fuzzing with ideas taken from evolutionary testing which, in turn, is inspired from the Nature's evolution process [16]. Genetic algorithm (GA) is one among the most widely used evolutionary algorithms that have been used in software testing by generating inputs for the desired properties [1]. We also make use of GA to detect XSS. A typical GA has the following pseudo-code:

1. Generate the initial population P
2. REPEAT
 - a. Run the SUT with P and calculate the fitness.
 - b. Sort the input in P.
 - c. Do CROSSOVER
 - d. Do MUTATION
 - e. Generate new population from children (from step 3 and 4) and set it as P.
3. UNTIL <some condition>
4. Return P

In the following, we explain each part in details.

2.2.2.3.1 Initial Population

Initial population means the initial inputs that will be used as seeds to generate new inputs. During the inference step, we know the sequence of inputs that will lead us to the state wherein the reflection occurs. Therefore, we take that input sequence as it is, called as prefix sequence. Here, we want to elaborate the structure of an individual input. Each input is a HTTP request that consists of several parts, consisting of parameters. When the input is at least partly reflected in the output, these parts may be used to form a valid HTML code. Therefore, it is important to supply valid parts with respect to HTML grammar. It should be noted that during inference step, we use only valid (expected) inputs, whereas during fuzzing, we may not adhere to this assumption. Therefore, we define many input classes that we use to generate inputs.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 20 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

C0: HTML Spaces: | \r | \t | \n
C1: HTML Attribute delimiter: " | ' | `
C2: HTML Tag delimiter: < | > | />
C3: HTML Equal sign: =
C4: JavaScript code: (|) | ; | { | }
C5: URL related: / | : | ? | &
C6: Escaping character: \
C7: HTML_TEXT_SIMPLE: [a-z 0-9]

During inference, we use a class $C_{\text{valid}} = C0 + C7$ at the lowest level of the input structure and which we then fuzz by replacing lowest level classes with C0 through C7.

Therefore, each input in the initial population is a sequence of inputs, consisting of the prefix sequence and fuzzed input for the state wherein the reflection is observed. This fuzzed input is derived from all the aforementioned classes.

Before we go explaining crossover and mutation, we explain an important feature that will be used in crossover and mutation.

2.2.2.3.2 Attack Input Grammar (AIG):

AIG helps to generate inputs that manifest a typical attack behavior. Therefore, while generating populations in GA, we try to adhere to AIG to generate inputs that are close to attacker's behavior. An excerpt of such an AIG is given in **Fehler! Verweisquelle konnte nicht gefunden werden.10**.

```

HTML_XSS_FIELD ::= HTML_TEXT_SIMPLE HTML_TAG_QUOTE
                  HTML_TAG_SPACE HTML_TAG_EVENT HTML_TAG_EQUAL
                  HTML_TAG_QUOTE JS_PAYLOAD
HTML_TAG_QUOTE ::= ' | "
HTML_TAG_SPACE ::= \n | \t | \r | _
HTML_TAG_EQUAL ::= =
HTML_TAG_EVENT ::= onabort | ... | onclick | ... | onwaiting

```

Figure 10. Input Grammar to generate fuzzed inputs (excerpt)


Based on this grammar, each input is represented as its parse tree, which helps while doing crossover/mutation.

2.2.2.3.3 Crossover

As each input in GA is a set of inputs that follow a particular state transition, crossover can be performed at two levels: 1) Between the sets of inputs; 2) between the individual inputs. In the former case, few individual inputs are exchanged, while in the later case, we change the parts of individual inputs. In the current implementation, we only use case 2. We use two-point crossover, which means that we select a point in both of the parents and exchange parts before and after that point. As we are using parse tree for individual, we choose the crossover point in such a way that the resulting children remain valid strings of the AIG.

2.2.2.3.4 Mutation

Mutation is performed at individual input level, i.e. it is performed on the parameters of an input. As inputs are strings from AIG, our mutation involves three operations to mutate the string: add, replace and delete. These three operations are performed on a particular non-terminal of AIG. The resulting new input is again a valid string of AIG.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 21 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

2.2.2.3.5 Fitness Function

The role of the fitness function is to assign higher scores to inputs that take execution closure to the vulnerable state. We devise a fitness function, which is specific to XSS. Intuitively, a good fitness function should assign higher score to:

1. inputs that go to deeper state (code coverage)
2. Inputs that contain “problematic” characters from XSS standpoint.
3. Inputs satisfying 2 and 3 and are valid as per the HTML grammar.

In view of the above points, we define the following fitness function (**Fehler! Verweisquelle konnte nicht gefunden werden.**¹¹)

$$Fit(I, SUT) = \frac{S(I, SUT)}{S_{total, SUT}} + \frac{C_{injected}(I)}{C_{total}(I)} + W_{ell}(I) + N(I)$$

Figure 11. Fitness Function for XSS specific input generation

where:

- I is the input being evaluated,
- $C_{injected}$ is the number of character classes successfully injected in the output q_i during fuzzing,
- C_{total} is the number of character classes that are present in the sent input parameters,
- W_{ell} is 0 or 1 (weighted) depending on whether the input is well-formed with respect to the output grammar and
- N is the number of newly added nodes in the parse tree of the inputs.

This is in relation to the notion of *valid syntactic form*, defined in [15]. Actually each term of this sum is implicitly assigned a weight to tune its impact on the fitness.

[Note: A prototype implementation is underway on the concepts defined above. This work was presented in SecTesT 2012 [10].]

2.2.3 Static and Dynamic Code Analysis Assisted Fuzzing

We recall from the previous deliverable D2.WP2 that our proposed work has three main steps:

- defining and finding patterns in the executable code for buffer overflow (BoF) vulnerability;
- doing a taint analysis by performing value-set analysis for memory accesses at intraprocedural level and generating tainted paths at interprocedural level;
- generating inputs to traverse those tainted paths to trigger the vulnerability.

In the following sections, we explain the progress that we made in each of the above mentioned steps. It is worth mentioning here that BoF, in spite of being one of the oldest vulnerabilities, has managed to get 3rd rank in Sans's most dangerous errors list [13]. *In fact, very recently, Mozilla released its Security Advisory 2012-21 on April 24, 2012 [11] and reported many memory errors in FreeType font library that is used by Mozilla products e.g. Firefox and our reported work is detecting many of those vulnerable functions.*

2.2.3.1 Patterns for Buffer Overflow Vulnerability

Currently, we focus on BoF vulnerability detection in the binary. A general practice to find BoF is to look for the presence of certain functions that manipulate string buffers, like the C/C++ **strcpy** family. A more general approach would be to define patterns to detect similar functions, which could equally be dangerous. We describe such patterns in terms of buffer overflow inducing loop (Boil) - a loop with memory write within the loop such that the written memory is changing within the loop. We identify two patterns that cover this behavior.



Initial Model-Based Security Testing Methods

Deliverable ID: **D3.WP2**

Page : 22 of 46

Version: 1.13
Date : 06.07.2012

Status : Final
Confid : Public

Pattern A: In this, the memory is accessed via a register (which happens outside the loop) and changing the memory address is a matter of incrementing the register value. Thus, it involves only one memory write within the loop.

Pattern B: In this, the memory is accessed directly within the loop. Each time, the address of the memory is changed and the corresponding value is read to be written into the other memory address that is also changed in the similar way. Thus there are three memory writes within the loop.

In order to detect such patterns, we make use of *dependency chains* (similar to classical *use-def chains*). We further generalize this notion to define *backward dependency chains*. Thereafter, we proceed to define *self-dependency chains*. The notion of self-dependency captures the fact that a variable v (or register/memory location) changes within the loop because its current value is obtained by something which is based on its previous value. A more detailed description can be found in [12]. **Pattern A** can be captured by detecting self-dependency chains. An illustration in Figure 12 makes this relationship more visible. In the figure, value of $v1$ depends on itself (i) and on $v3$, which in turn depends on itself (ii).

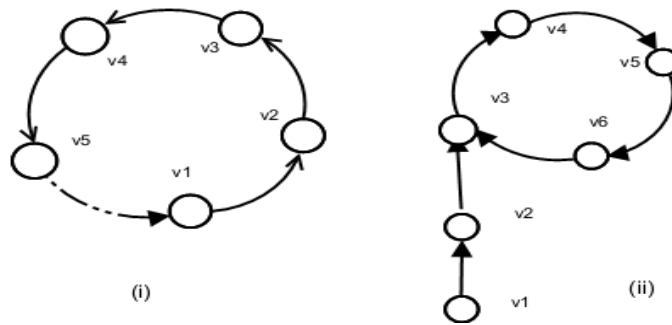


Figure 12. Self-Dependency Chains corresponding to Pattern A

In order to detect Pattern B, we again extend the notion of self-dependency chain to define *extended self-dependency chain* that consists of two or more self-dependency chains with some link. Figure 13 illustrates this relationship.

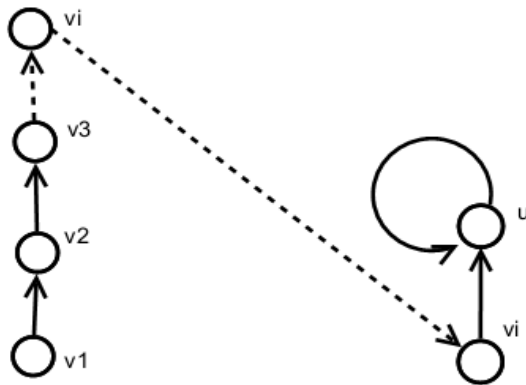



Figure 13. Extended Self-Dependency Chain corresponding to Pattern B

Intuitively, it says that value $v1$ depends on vi under a dependency chain and vi has a self-dependency chain starting from it, which means that a location depends on a location which in turn depends on itself, but indirectly under a different instruction sequence.

On the basis of above formalism, we implemented a tool to detect **Buffer Overflow Prone (BOP)** functions. The tool is able to analyze real world applications. We experimented with many applications (results are published in [12]), e.g. kernel32.dll, ntdll.dll, OpenSSL, mpg123 etc. and established empirically that our approach is practical in identifying buffer overflow vulnerabilities.

Such BOP functions are the destination of the tainted path that we describe below.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 23 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

[Note: This part of our work is completed and we shall be analyzing the binary code from Metso, Finland. This work is presented in SERE 2012 [12].]

2.2.3.2 Value-Set Analysis and Taint Data Flow

Taint based information flow tracking is a widely used approach to effectively detect many vulnerabilities. Taint analysis is a specialized form of dataflow analysis, wherein the dataflow is calculated from a tainted source to some sensitive sink. As it is dataflow analysis, we need to be careful for *aliases*. Static alias analysis is a hard problem, even at source code level. At binary level, this is much more complicated as we lose any high level of information regarding the variables. In order to do an accurate dataflow analysis, we develop a **value set analysis** (vsa)[7][8][9] that is used to detect aliases to do accurate dataflow analysis.

Our algorithm will compute *abstract addresses of memory locations*, meaning that:

1. An abstract address will be defined as offsets with respect to some reference value (the value of an extended stack pointer (**ESP**), the value returned by a given call to malloc(), etc.) ;
2. An abstract address may represent a set of possible real addresses (over-approximation) ;
3. Only a finite set of distinct offsets will be considered for a given reference value (widening to a special address meaning any possible address).

2.2.3.2.1 Value computation as a forward data:

The technique we used can be viewed as generalization of a constant propagation algorithm implemented as a dataflow analysis along the Control Flow Graph (CFG) of the program. It computes a set of abstract values (where each abstract value is a superset of some concrete values) for each register and each memory location. The algorithm computes the more precise state vectors associated to each node compatible with the program execution. To do so, the program semantics is abstracted by a transfer functions F describing how the execution of each instruction i may transform a state vector. Abstract values coming from different execution paths are combined using a merge operator, defined as the least upper bound in the lattice. This analysis is performed *intraprocedurally*.

The dataflow equations are defined in Figure 14.

$$S_{in}^n = \begin{cases} S0_{in} & \text{if } n \text{ is the entry point of the CFG} \\ \bigsqcup_{n' \in \text{Pred}(n)} S_{out}^{n'} & \text{otherwise .} \end{cases}$$

$$S_{out}^n = \mathcal{F}(S_{in}^n)$$

Figure 14. Dataflow equation for the VSA analysis

S_{in}^n (resp. S_{out}^n) is state vector while entering (resp. exiting) the n th instruction. Transfer function F is defined as per the semantic of the instruction. The table depicted in Figure 15 illustrates the transfer function for such operations.



Initial Model-Based Security Testing Methods

Deliverable ID: **D3.WP2**

Page : 24 of 46

Version: 1.13
Date : 06.07.2012

Status : Final
Confid : Public

Instruction (I)	Abstract Loc $\cdot (\mathcal{A}')$
$R := c$	$Upd(\mathcal{A}, [R \mapsto [c, c]])$
$R := R'$	$Upd(\mathcal{A}, [R \mapsto \mathcal{A}[R']])$
$R := *(R')$	$Upd(\mathcal{A}, [R \mapsto \bigcup_{x \in \mathcal{A}[R']} \mathcal{A}[x]])$
$*(R) := R'$	$Upd(\mathcal{A}, [x \mapsto \mathcal{A}[R']]), \text{ if } \mathcal{A}[R] = \{x\}$ $Upd(\dots (Upd(\mathcal{A}, [x_1 \mapsto \mathcal{A}[x_1] \cup \mathcal{A}[R']]) \dots$ $\dots), [x_n \mapsto \mathcal{A}[x_n] \cup \mathcal{A}[R']])$ $\text{if } \mathcal{A}[R] = \{x_1, \dots, x_n\}, n > 1$
$R := R_1 + R_2$	$Upd(\mathcal{A}, [R \mapsto \bigcup_{r_1 \in \mathcal{A}[R_1], r_2 \in \mathcal{A}[R_2]} r_1 \oplus r_2])$

Figure 15. Transfer Function for the effect of instructions

An abstract location (a-loc) is a representation of a memory zone accessed at program execution (like a variable name at the source level). An abstract loc is expressed as a pair $\langle B; X \rangle$ where:

B is an (abstract) base value, which can be either:

- an instruction (address) ik
- an element of the set $\{\text{Empty}, \text{NoVAL}, \text{InitEsp}, \text{Any}\}$

and X is a finite set of integers.

2.2.3.2.2 Calculating the Function Summaries for Inter-procedural Analysis

After performing the VSA, we proceed to calculate taint dataflow at intra-procedural level. This is required to record the effect of individual instructions on the flow of tainted data. We model taint dataflow as classical *copy propagation* analysis. More precisely, we propagate the copy of function's arguments, memory pointed to by arguments and global variables. In other words, after performing the copy propagation analysis, we know precisely if there is some dependence among function's arguments, memory pointed to be those arguments, global variable and function *return*. The overall use of this step is to obtain the effect of a procedure call. This effect is denoted by the *function summary* of the corresponding function.

Function summary is a function parameterized by the arguments, memory pointed to by the arguments (denoted as MEM(args)) and global variables.

The output of this function is the indication of (taint) data dependency of return value, arguments, MEM(args) and global variables on the parameters.


For any instruction, its operand is initialized when it is assigned with either arguments, local variables or global variable. We assume that arguments and local variables are accessed via offsets with respect to the extended base pointer (EBP).

Function Summary returns four dictionaries:

- Dependencies for the arguments ARGS.
- Dependencies for the MEM(args) MEMARGS.
- Dependency for the return (last eax used) RET.
- Dependencies for the global variable GLOBALS.

Once we calculate summaries for all the *required* functions, we can perform a module-wise taint analysis by using the summaries. This *whole system analysis* provides us with a set of tainted paths that could lead to vulnerable functions when executed with tainted data.

However, it should be noted that mere the presence of tainted paths does not make the vulnerability exploitable as there may exist various filters on tainted data that prevent any possible exploits. Therefore, in order to verify the exploitability, we need to perform a dynamic approach by generating real malicious inputs. This is covered in the next section.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 25 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

2.2.3.3 Tainted Path Based Input Generation

A tainted path is a sequence of function calls such that source of the tainted data (functions that accept tainted input, e.g. ReadFile, socket etc.) starts this sequence and the sequence ends at some vulnerable function. In essence, a tainted path captures the process of data consumption by the application. Therefore, by generating inputs that traverse a given tainted path i.e. driving the execution in a particular direction, we should be able to show the exploitability of the vulnerability. In this work, we focus on file processing software e.g. PDF viewers, image file viewers. Such applications read files and display it to the users. One can observe that even though, we have a tainted path from file reading function to some vulnerable code, we still need to know which part (or offsets) are used to execute the vulnerable functions. As our tainted path generation is static, it is very difficult to know this information a priori. We, therefore, propose to perform a dynamic analysis to know where in the memory a file is copied and how the application accesses that memory buffer later on. We do not perform this dynamic analysis for the whole duration of the application run because such a monitoring makes the application very slow. Rather we do it to the access level from where we can get a static path, with the required offsets. Figure 16 illustrates the process on a higher level.

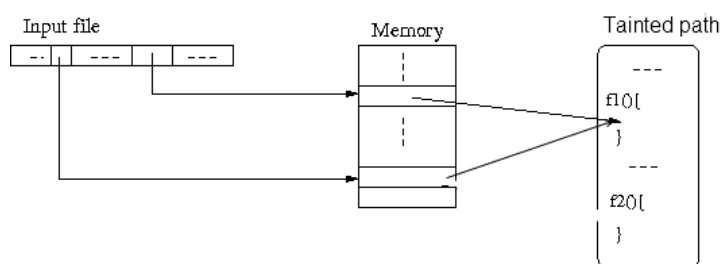


Figure 16. Combining dynamic and static taint flow

Once we know the offsets, we can generate inputs by focusing on those offsets only. The intuition behind this approach is that by fuzzing data at only those offsets, we do not disturb the control flow of the application. We plan to go beyond this approach by not assuming the format knowledge of the input file and applying evolutionary fuzzing concepts to generate inputs that fuzz data while traversing the tainted paths.


2.3 IMPROVING THE EFFICIENCY OF UML SEQUENCE DIAGRAM FUZZING

2.3.1 Status Quo

In D2.WP2 [17], a novel fuzzing approach – behavioural fuzzing – was introduced along UML sequence diagrams. The novelty of the approach, in contrast to traditional fuzzing that focuses on data, is that invalid behaviour is used while stimulating the SUT. That means that the focus of behavioural fuzzing is not on input data but on generating invalid sequences of messages to stimulate the SUT. The goal is to find security related weaknesses in the realization of the SUT. Invalid sequences of messages are generated out of valid ones represented by UML sequence diagrams by applying behavioural fuzzing operators. A behavioural fuzzing operator is a new kind of fuzzing operator that modifies the sequence in a distinct manner. Examples of behavioural fuzzing operators are *Move Message* that moves a certain message from its original position and inserts it at another position. A list of behavioural fuzzing operators is shown in D2.WP2 [17]. A behavioural fuzz test case is generated by applying one or more behavioural fuzzing operators to a sequence diagram, e.g. representing a functional test case.

2.3.2 Problem: Explosion of Generated Test Cases

A behavioural fuzzing operator can be applied to a sequence diagram in many different ways. The number of possibilities how to apply a fuzzing operator to a UML sequence diagram depends at least on the number of elements that are of the type the fuzzing operator is applied to. That is the case for the fuzzing operator

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 26 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

Remove Message that simply deletes a message. However, further parameters may have influence. For instance, the fuzzing operator *Move Message* has two parameters: the message that shall be moved and the position where the message can be moved to. Therefore, the maximum number of possible applications of *Move Message* is constituted by the number of messages multiplied by the number of possible positions where a message can be moved to. The number of possible positions is given by the number of messages in the sequence minus 1. The whole can be approximated by the square number of elements within a sequence diagram.

As discussed above, a test case is generated by applying different fuzzing operators to a sequence diagram, thereby respecting the order of how fuzzing operators are applied, where each fuzzing operator can be applied to a sequence diagram only once.

If o' is the number of possibilities how all fuzzing operators can be applied to a certain sequence diagram, then the number of test cases that can be generated is approximated by the following formula:

$$o \left(\sum_{i=1}^n \frac{o'!}{(o' - i)} \right)$$

o' is calculated by the following formula, where o is the number of behavioural fuzzing operators that can be applied to a sequence diagram and e is the number of elements enclosed in the sequence diagram. In addition, k is the maximal number of parameters the application of a fuzzing operator depends on:

$$o' = o \cdot e^k$$

The variable i is running from the minimal (usually 1 in order to generate semi-valid test cases) up to the maximal number of fuzzing operators to be applied to a sequence diagram in order to generate a single test case.


As discussed above, for the fuzzing operator *Move Message*, k equals 2, since the square number of elements is involved. For a simple sequence diagram consisting of 5 messages and a set of 3 behavioural fuzzing operators, applying up to 3 fuzzing operators in order to generate a single test case can lead to generating over 400,000 test cases. Applying only the behavioural fuzzing operators regarding single messages (in [17]) and setting k to 1, for the small model of 5 messages, about 2 million test cases can be generated. That explosion of generated test cases leads to the problem that there is not enough time to run all these test cases. Hence, the goal must be to reduce the number of test cases to a reasonable amount fulfilling two properties: find these test cases that have the capabilities to find weaknesses in the SUT's implementation (maintaining the effectiveness of the test cases while improving the efficiency) and from that set, eliminate these test case that are addressing the same weaknesses (detection of duplicates). Because fuzzing is a test method to find zero-day vulnerabilities, this seems to be an infeasible job. However, we believe that this can be achieved with support from the system engineer. The basic idea is to give the test case generator hints by augmenting the model with security-relevant stereotypes.

2.3.3 Proposed Problem Solution: UMLsec

There are some security related UML profiles that allow for augmenting the system model with security-relevant information that can be used by the test case generator. One of them is UMLsec which aims at providing a rich set of stereotypes where some of them seem to be suitable to guide the test generation process of behavioural fuzzing. We discuss how that could work along the stereotype *rbac* meaning role-based access control (RBAC).

2.3.3.1 Discussion on the UMLsec RBAC Stereotype

Role-based access control is a way to assign permissions to roles and roles in turn to concrete users in order to a simplify permission management [19]. The corresponding data model that is part of a proposed standard of the National Institute of Standards and Technology (NIST) is shown in Figure 17 where PRMS are permissions on objects (OBS) and operations (OPS).

	Initial Model-Based Security Testing Methods Deliverable ID: D3.WP2	Page : 27 of 46 Version: 1.13 Date : 06.07.2012 Status : Final Confid : Public
----------------------------------------------------------------------------------	------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------

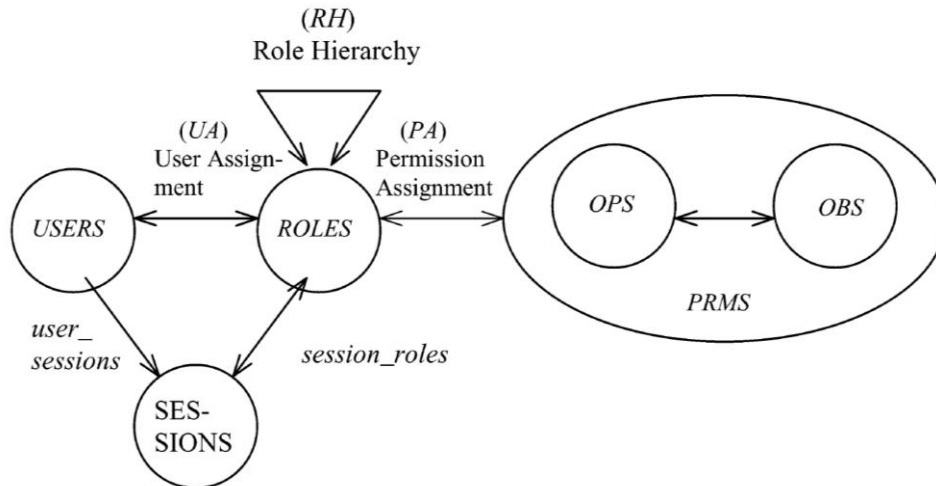


Figure 17. Proposed NIST Standard for Role-Based Access Control [20]

The UMLsec profile supports role-based access control via the stereotype *rbac* that is annotated to an activity diagram. In addition, there is a set of associated tags:

- **protected** whose values contains the protected resources that are states of the *rbac* annotated activity diagram. The values of this tag correspond to *operations* (OPS) on *objects* (OBS) in Figure 17.
- **role** contains tuples of an actor (of the annotated activity diagram) and a role where the actor is assigned to the role. The tuples of this tag correspond to the association *User Assignment* (UA) in Figure 17. The users and roles are implicitly defined by the tuples.
- **right** describes which roles have permissions to access a protected resource by corresponding tuples. This corresponds to the association *Permission Assignment* (PA) in Figure 17.

A simple example is given in Figure 18. depicting a simplified use of a banknote processing machine. Although the stereotype *rbac* is not useful for such a small example, it is sufficient to show the principles of UMLsec *rbac*. Before the machine can be used, a user has to login with valid login data. If the login was successful, he is logged in as an operator and may configure the banknote processing machine in order to count money and at the end the operator logs out. The actions *configure* and *count money* are protected as required by the values of the tag *protected*. The operator is taking the role of the money counter (tag *role*) and may access the protected actions *configure* and *count money*.

Counting money «rbac»

```
{role=(operator, money counter)}
{right=(money counter, configure), (money counter,
count money)}
{protected="configure", "count money", "logout"}
```

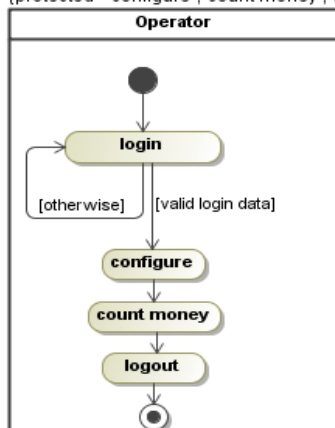



Figure 18. Simple Example of an Activity Diagram with the UMLsec *rbac*

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 28 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

2.3.3.2 Adoption of UMLsec RBAC for Improving Behaviour Fuzzing of Sequence Diagrams

In order to reduce the number of test cases to a manageable set, a model augmented with stereotypes regarding role-based access control is helpful. It allows identifying a subset of test cases that are able to find weaknesses regarding authentication. To achieve that goal, it is necessary to enhance the UMLsec rbac mechanism to mark such messages that change the authentication state and to allow rbac to be applied to sequence diagrams. Those messages generally are login and logout messages. For the sake of simplicity, the terms login and logout are used instead of messages that increase respectively decrease the authentication state.

Having the piece of information what messages are login and logout messages, the number of messages considered by behavioural fuzzing operators as well as their number of applications can be reduced:

- The fuzzing operator *Move Message* can now only move the login and logout messages. Login messages can be moved stepwise closer to the logout message to test if the messages appearing after the login can be successfully executed without authentication. Accordingly, the logout message can be moved stepwise closer to the login message to test if the logout is successful and no operations can be executed after a logout.
- *Remove Message* may consider only the login message in order to test if messages that need authentication can be performed without.
- *Repeat Message* may only repeat the login and logout message in order to check if the authentication state remains unchanged by the repeated message

2.3.3.3 Application to an Example

When considering the example depicted in Figure 18., a corresponding test case would look like the one in Figure 19. where the information about protected resources, roles and rights are copied from the activity diagram. Additionally, there is one more tag *authentication* with a tuple whose first element contains the information which message performs authentication and which performs a de-authentication.



Initial Model-Based Security Testing Methods

Deliverable ID: **D3.WP2**

Page : 29 of 46

Version: 1.13
Date : 06.07.2012

Status : Final
Confid : Public

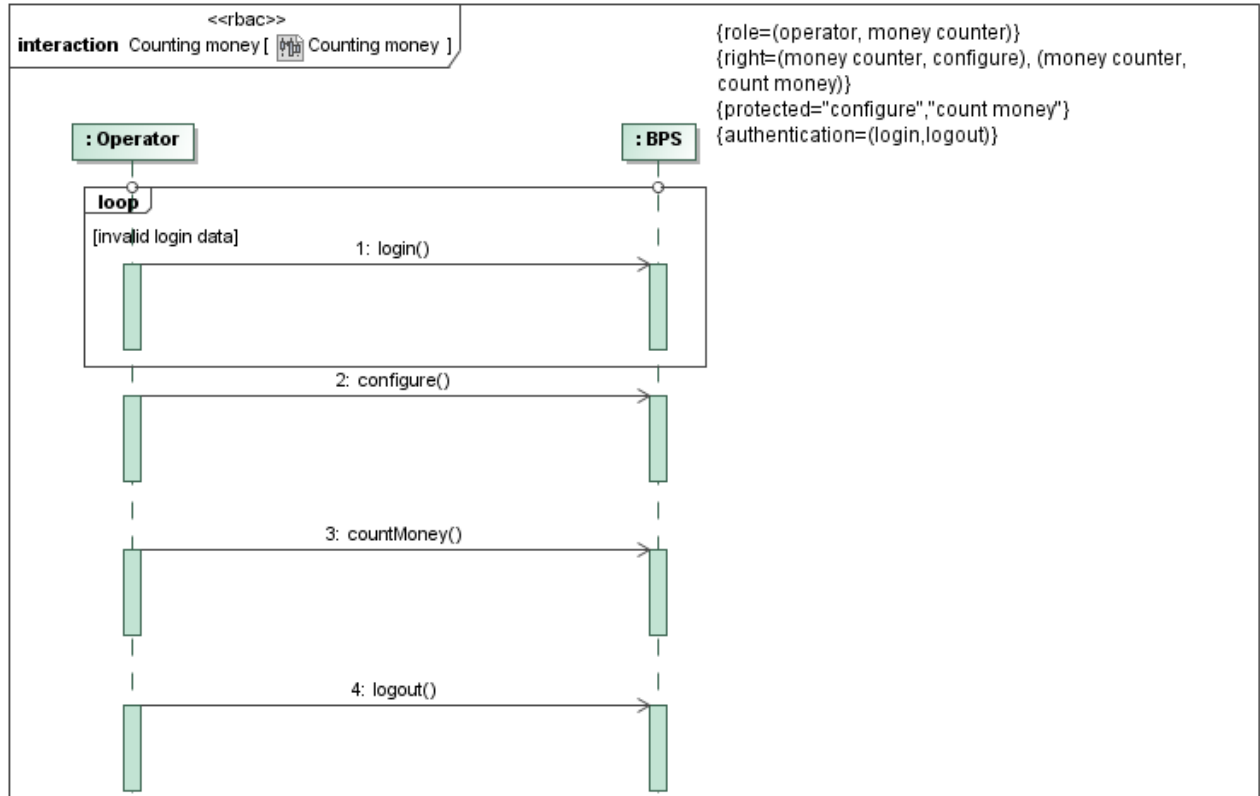


Figure 19. Test Case derived from the Activity Diagram in Figure 18.

Considering the three behavioural fuzzing operators *Move Message*, *Remove Message* and *Repeat Message* and the equation in section 2.3.2, setting o to 3, k to 2, e to 4 and n to 1 (meaning that a test case is generated by applying only one fuzzing operator), the maximal approximated number of test cases that can be generated is 48. The exact number is 20 (see approximation in section 2.3.2). By respecting the UMLsec RBAC stereotype, the number of relevant test cases can be reduced to 8.

2.3.3.4 Number of Generated Test Cases Revisited

The number of possibilities to apply a fuzzing operator with respect to RBAC is reduced and can be approximated by:


$$o'_{rbac} = o \cdot 2 \cdot (e - 2)$$

This is because each fuzzing operator is applied only to two messages - the login and the logout message - and can be applied at most as the number of the messages excluding login and logout. Obviously, o'_{rbac} is much smaller than o' because only one factor depends on the number of elements within the sequence diagram.

The following table shows the number of test cases generated without and with respect to RBAC:

k=2, e=5	behavioural fuzzing so far		with respect to RBAC	
	o'	number of test cases	o'_{rbac}	number of test cases
o=3	75	410,775	18	5,220
o=5	125	1,922,125	30	25,260

That numbers show that the number of test cases can be dramatically reduced by focusing on a certain security aspect while performing behavioural fuzzing. This security aspect can be integrated in the model by employing a UML profile, for example UMLsec.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 30 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

2.4 FUZZING OPERATOR IN TEST CASES

2.4.1 Introduction

The following chapter discusses the integration of the fuzzing operators to the Dornier Consulting case study. As explained in D3.WP3 the system model gets analyzed by one of the framework's model parsers and is then transformed into a do.ATOMS internal representation, the so called workflows (based on the workflow foundation of Microsoft's .NET framework) as shown in Figure 20. These workflows in turn can also be considered as an independent test model, which are - in this case - automatically derived from the system model (therefore this step is also called "model to model transformation"). Each workflow consists of several workflow activities that interact with the SUT (e.g. via sending a CAN or Bluetooth message) or that monitor the state of it (verification of the presence of certain CAN messages). The flow order of the activities can also be altered in various ways using workflow flow control constructs (like if/else branches or loops). All test cases follow the uniform approach that do.ATOMS simulates the Bluetooth end user device to test the Bluetooth connectivity features of the SCM.

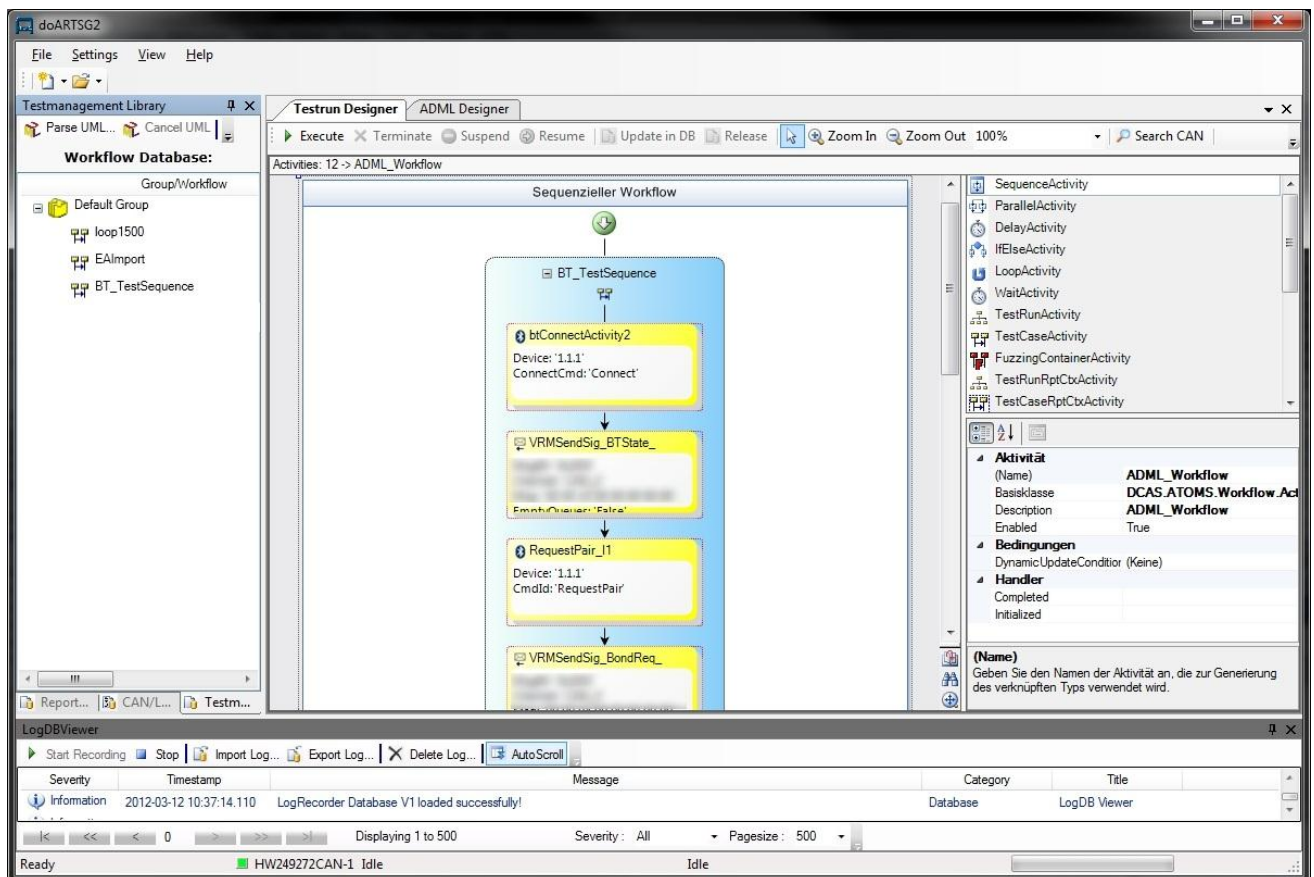



Figure 20. A workflow diagram within the do.ATOMS framework

During the transformation phase security relevant information (like the security score metadata) that is part of the system model is also transferred and is still present in the do.ATOMS workflow model. This allows the application of security testing techniques like data fuzzing, which is explained in the following section.

2.4.2 Applying Fuzzing operators

Based on the security score and the occurrence the priorities of the used messages are calculated. This priority tells do.ATOMS, which message is relevant in respect to security. For normal functional testing

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 31 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

purpose the sequence diagrams of the system model are transformed to a workflow diagram within do.ATOMS. In the next step the user has the option to generate fuzzed test cases based on the previously generated functional test as shown in Figure 21 **Fehler! Verweisquelle konnte nicht gefunden werden.**below.

In the first implementation of the approach the calculated security score is used to choose the best message for a fuzzing modification. In the near future a setting dialog will be available to choose from different techniques. During the transformation phase security relevant information (like the security score metadata) that is part of the system model is also transferred and is still present in the do.ATOMS workflow model. This allows the application of security testing techniques like data fuzzing.

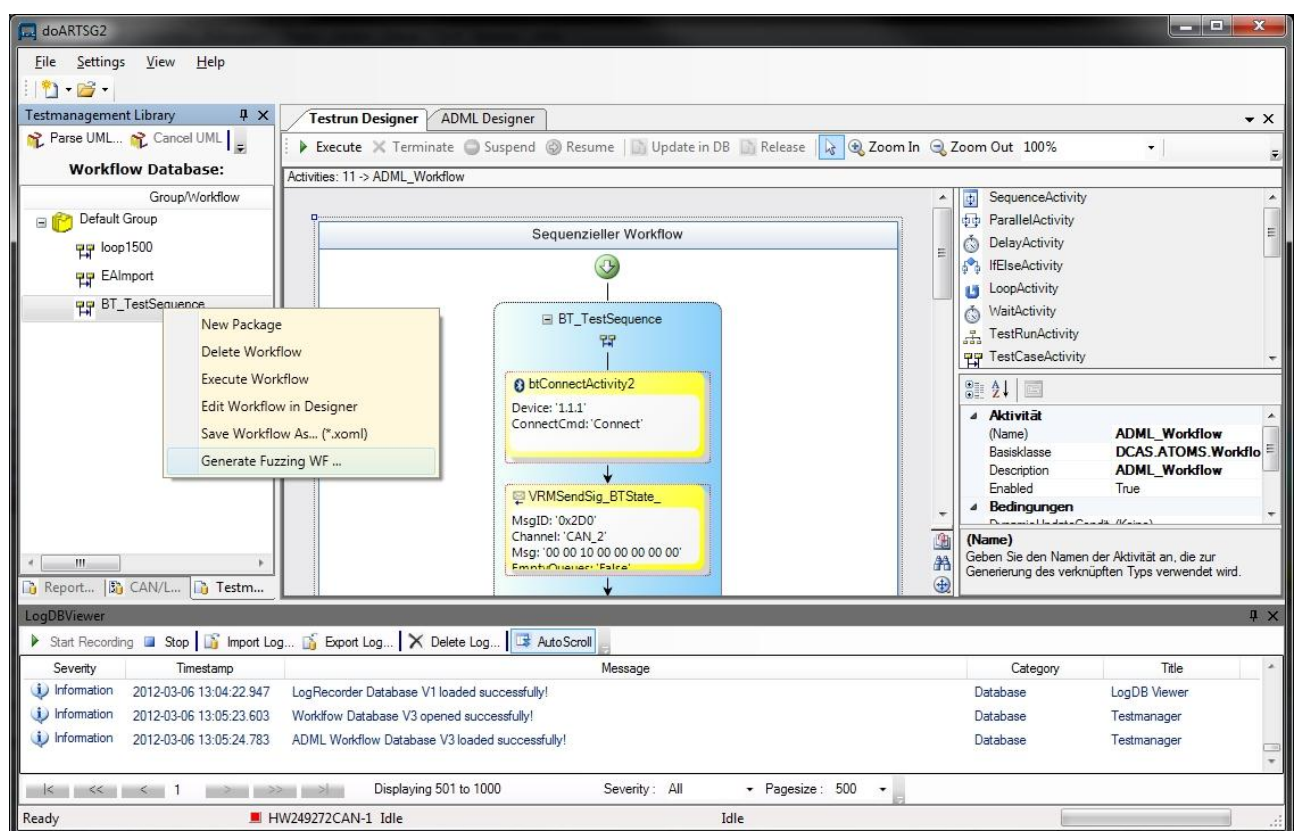


Figure 21. The do.ATOMS framework

Each derived test case that is transferred into the test case library of the do.ATOMS test case designer can be transformed into a fuzzing test case. A fuzzing test case can be seen as a “malicious” counterpart of a test case that represents the normal system behavior. As mentioned before, the workflow activities are responsible for sending stimuli to the SUT. For finding entry points for data fuzzing in these activities the following approach was chosen:

- Each activity of the “normal” test case is examined and its security score information is evaluated.
- If the security score is high enough, the activity is enclosed by a so-called fuzzing container. This element is responsible for injecting the fuzzing functionality by manipulating its child element (at design time when the workflow is created or at runtime when the workflow gets executed).

After selecting the message with the highest score a fuzzing container is set around the message. This will change the original message without changing the message itself. Figure 22 below shows the message ‘EnterPIN_I4’ applied with a red fuzzing container.

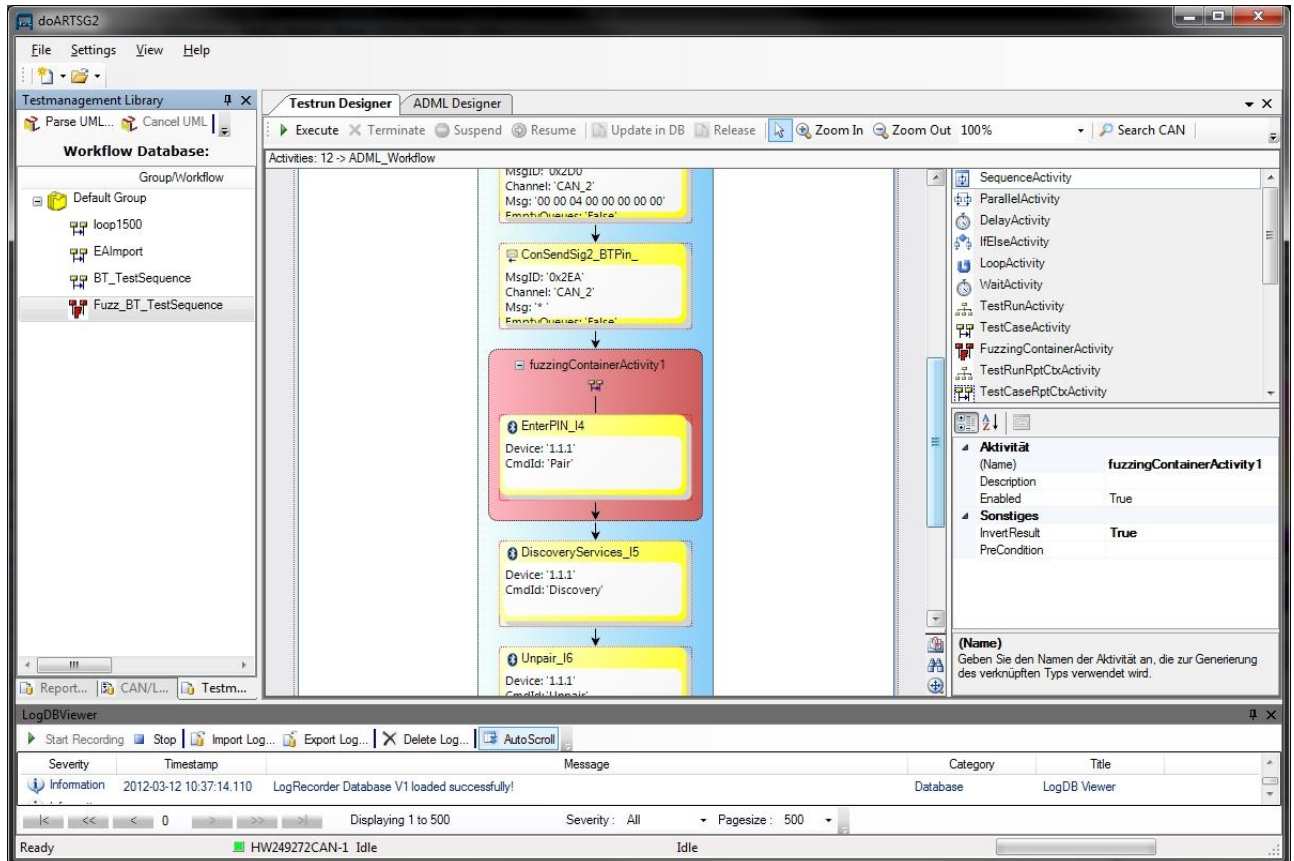


Figure 22. A fuzzing container applied to the message 'EnterPIN_I4'

To sum up, the following figures shows the complete conceptual approach for test case generation that should be realized with do.ATOMS:



Initial Model-Based Security Testing Methods

Deliverable ID: **D3.WP2**

Page : 33 of 46

Version: 1.13
Date : 06.07.2012

Status : Final
Confid : Public

Concept overview:

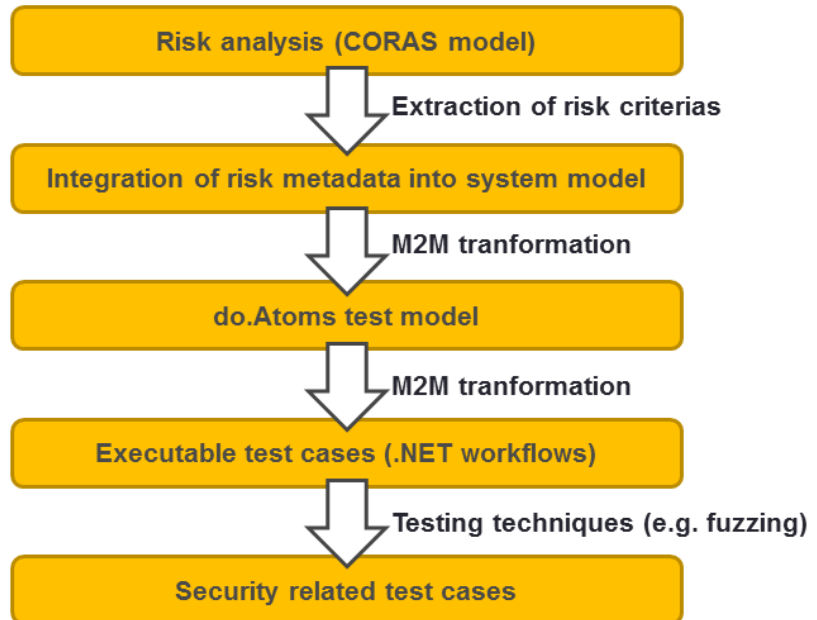



Figure 23. Concept overview

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 34 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

3. RISK ANALYSIS FOR RISK BASED TESTING

3.1 TRACEABILITY METHODOLOGY

3.1.1 General Concept of Software Traceability

The following sections describe the current status of the development of a traceability methodology within the DIAMONDS project. This methodology brings a number of benefits for the tasks of Model Based Security Testing. For example, a traceability framework would enable to put in relationship the various models and artefacts involved in security testing.

We start with a general description of traceability and proceed with state-of-the-art from the domain of safety. Afterwards, the specific traceability notion realized within the DIAMONDS project is presented. Finally, different concepts, models and realization aspects are described that lay the ground for an implementation of traceability and its application to industrial case studies within the DIAMONDS project.

3.1.2 General Concept of Software Traceability

Traceability is a concept that has been around for some time in the domain of software and systems engineering. According to [22] traceability stands for *“the ability to follow the life of a requirement, in both a backward and forward direction”*. The software traceability concept, also known as requirement traceability, was developed in safety engineering. A requirement engineer defines relationships between safety requirements and system elements and/or test cases.

Traceability is a mechanism for relating artefacts and is vital for ensuring the completeness of the specification and for the system itself [21]. A system with trace relations can respond accordingly on requirement changes and system restructurings.


3.1.3 Traceability in Safety

Traceability has been especially used as a key concept in the domain of safety. Thereby, it plays an essential part in the development of safety critical systems. Traceability is traditionally based on a Traceability Information Model (TIM) that allows understanding the relationships around various Safety Requirements. This model enables for the realization of different types of traces [26]:

- **structural traces:** generalization and aggregation, traces contained inside the models that have to be traced
- **explicit traces:** traces that are manually created and connected between models
- **implied traces:** transitive associations, indirect associations

In the development of safety critical systems, traceability and the belonging TIM are widely used for Program Slicing. Program Slicing stands for the computation of a set of programs statements (the program slice) which may affect the values at some point of interest, referred to as a slicing criterion. Program Slicing can be used in the course of debugging, in order to easily localize sources of errors. Other useful applications of Program Slicing include software maintenance and optimization, program analysis and information flow control. Obviously, a Traceability Information Model can play a key role in the creation and identification of program slices, as well as in tracing the relation between Safety Requirements and particular program slices.

In addition to Program Slicing, another use case within safety critical systems development is to comprehend software and requirement changing effects for the system.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 35 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

To conclude, traceability within the development of safety critical systems is a mechanism for relating artefacts and is vital for ensuring the completeness of the specification and for the system itself [21]. The next section proceeds with a description of the DIAMONDS concept.

3.1.4 The DIAMONDS Traceability Concept

Different realization concepts are known. One approach is to define the traces directly in the model. This is only possible if the traced elements are in same model domain. For example if the system model and the test model system are modelled in UML. The advantage is no needed trace mechanism. The traces are handled equally as all other relations in the model, but in scenario with more than one model domain it is a useless concept. Additionally the model has to be extended with a new relation that is not really necessary for the model itself.

A second approach will be used in case of handling with elements in more than one model domains. A new model domain will be defined that contains the logical structure of all related models. The benefit is to extract the trace mechanism out from the target model. But the disadvantage is the redundancy because the model information will be handled twice and an update mechanism has to be implemented.

Another approach defines a new model directly for the trace elements. The trace model contains only the trace information and no other redundancy information. This approach will be used in DIAMONDS. The disadvantage of this approach is a complex handling in case of handling transitive relations through the models. This problem will be handled in chapter 3.1.6 for the defined model domain adapters and the query language concept.

3.1.5 Basic off-the-shelf Traceability Model

The following paragraphs describe the trace model that was adopted for realizing traceability within the DIAMONDS project.

3.1.5.1 Ecore Based Model

After evaluating freely available traceability tools, we identified the CreMA [27] framework as the most suitable one. It is quite mature in its realization and is based on Eclipse and Ecore, which allows to easily integrate Eclipse based tools and corresponding models within the traces.

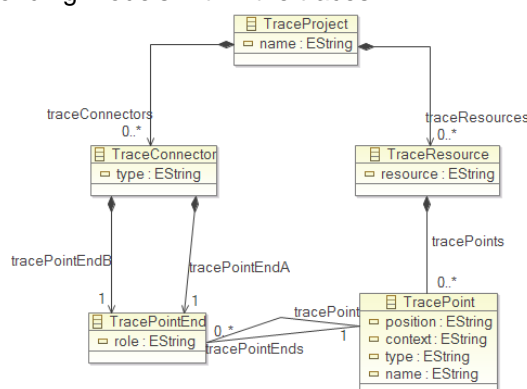



Figure 24. The Traceability Meta-Model

The Traceability Meta-Model used in the course of DIAMONDS is illustrated in Figure 24. The traces are contained in a *TraceProject*-element and stored as *TraceConnector*-elements. A *TraceConnector*-element uses two references to *TracePointEnd*-elements. Each *TracePointEnd*-element is connected with a *TracePoint*-element, representing an element in the traced model. The resource information with the origin of

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 36 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

the model source (normally the file name) for each *TracePoint* is directly linked with the *TraceProject*-element. This simple but powerful traceability model contains all the basic concepts which are required for establishing traces between elements of models.

3.1.5.2 Elements in Detail

Next, each part of the model in Figure 24 is elaborated in turn.

3.1.5.2.1 Trace Element

The basic trace element in our model is denoted as Trace Connector. It constitutes the core of a user defined trace between two elements from the traced models. Furthermore, the type of the traced elements is explicitly involved within a trace. In DIAMONDS the type will be extended to restrict different trace types to specified traced element types. The types will be defined in our project in two steps: In the first step the user have to select the type for a trace from a defined list. In the second step the types will be automatically set from an additionally developed information model.

3.1.5.2.2 Trace Point Element


The Trace Point element represents an element integrated within a traced model, thereby holding the information for the traced element. This information contains:

- the name of the element or other kind of **unique identification**
- a type for a **simple type identification** as string
- an **unique Ecore UID** of the traced model for identifying
- the **file name** for the location of the traced model
- a unique information string for the **specific model type** of the representation tool (like CORAS, PAPYRUS or ProR)

The file name for the location is indirectly stored in the TracePoint-element. It is kept in the TraceResource-element linked to the TracePoint-element.

3.1.6 Description of the Target Realization of the Trace Management Platform (TMP)

The goal of the traceability concept in the DIAMONDS project is to provide a framework that enables the definition of relations between models for the security assessment. Models are security analysing, system development, requirement specification, test case definition and, perhaps other model types that may play a role in the future. In the first step the target is to provide a framework for creating traces between elements of different models, which are used within some of the project relevant tools. This framework is denoted as the Trace Management Platform (TMP). TMP is initially illustrated in Figure 25. For security and risk analysis, we integrated the CORAS-tool based on the CORAS meta-model. For the system development we integrated Papyrus in order to use the SysML meta-model. For requirement specification, we attached the ProR tool with the ReqIF meta-model and for the test case generation and execution we integrated TTCN3 related artefacts and tools.

	<p align="center">Initial Model-Based Security Testing Methods</p> <p align="center">Deliverable ID: D3.WP2</p>	Page : 37 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

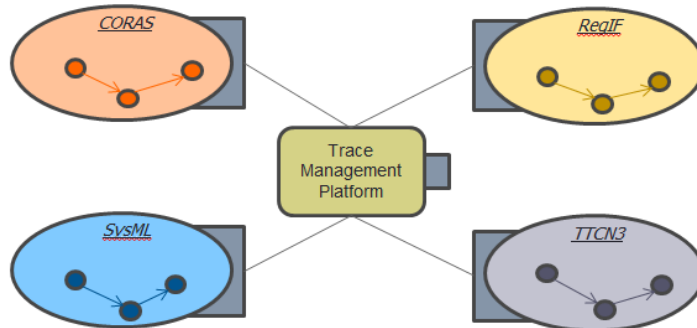


Figure 25. Step 1 - TMP Overview with belonging Tools, Models and Test Notations

For the rest of the DIAMONDS project and beyond, other types of models and tools can be linked with TMP in the near future. Hence, a generalization (abstraction) from what is shown in Figure 25 is required. This generalization naturally leads to the concept of domains and domain models, which are integrated over the TMP. In that line of thoughts, the generalization of Figure 25, leads to the domains and domain models illustrated in Figure 26. Hence, we see the application of TMP for Model Based Security Testing as an integration platform between the following domains and their belonging domain models: risk model domain, system development domain, requirement specification domain and test model domain.

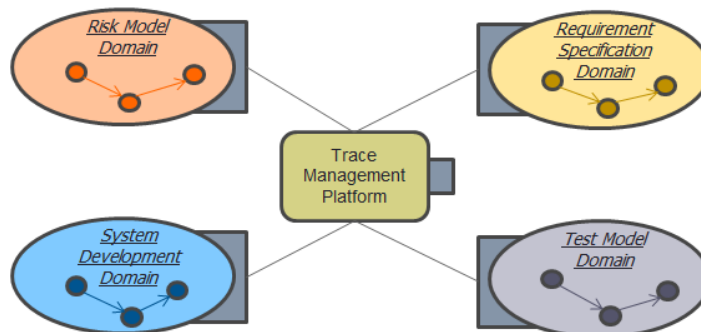



Figure 26. Step 2 - TMP Overview with Domains of Relevance for Model Based Security Testing

Based on this abstraction, TMP can be designed in a way as to be independent of the particular tools and models from the domains. This can be achieved by defining generic meta-models for the domains in question and providing an infrastructure such that each tool (that belongs to a certain domain) can be integrated with TMP over an adapter that translates between the TMP domain model and the tool specific model. That way, TMP does not need to be aware of the specifics of a tool, but simply relates to the domain model in question while communicating with a tool.

Figure 26 illustrates the domains involved in Model Based Security Testing including the belonging domain models and adapters that translate between a domain model and a tool specific model. In that context, every request from/to a tool attached to TMP results in queries from/to the underlying domain model. As illustrated in Figure 27, TMP supports two types of interfaces: the domain adapter interface and the Trace Management Service Interface (TMSI). TMSI is exposed to all services that are implemented and run on top of TMP.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 38 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

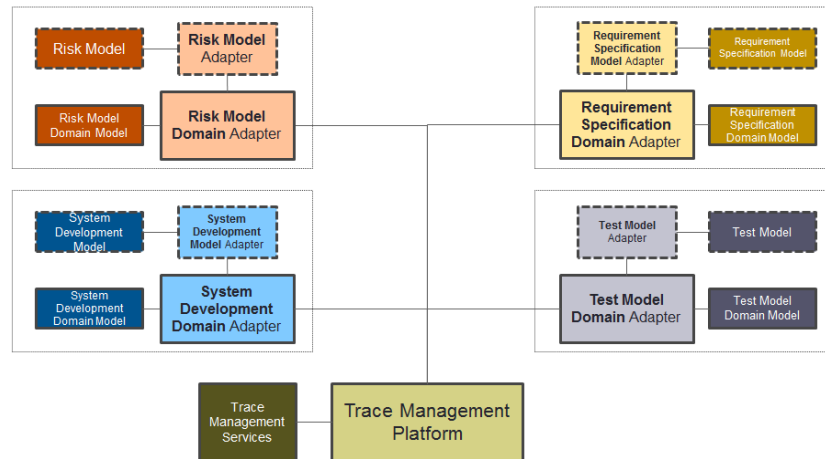


Figure 27. TMP Overview regarding the Domains involved in Model Based Security Testing

3.1.6.1 Domain Specific Models

We define two kinds of traced models. The first kind is the domain model, an abstract model for a specific domain. In our project we are currently supporting four domain model types:

- the **risk model domain** - to model the system risks using concepts such as threats, vulnerabilities, risks, assets and treatments,
- the **system development model domain** - to model the system components, the signals and the behaviours,
- the **requirement specification model domain** - to specify all security and test specific requirements for the system,
- the **test model domain** - to manage test cases and test suites.


Naturally, for every domain there is a separate domain model. For the risk model domain, we use an abstract domain model that is aligned to the concepts and understanding within the CORAS methodology. For the system development domain, SysML is adopted as domain model. The requirement specification domain model is realized by Req4f. Finally, for the test model domain we use a meta-model that is aligned with the concepts around TTCN-3.

In addition, we have the models which are brought in by the different tools from the domains of interest. These models will be connected over specific domain adapters that meaningfully align the elements of a proprietary tool model to the selected domain model, e.g. CORAS for the risk domain.

3.1.6.2 Service Adapters on Top of TMP

The TMP allows for easily implementing services that utilize the traces between the models. Examples for TMP services are given by:

- Service for **Trace Navigation**.
- Service for **Management of Traces** including tasks like creation, deletion and copying traces
- **Trace Information Collector Service** for relating different aspects of Model Based Security Testing such as relations between requirements and test results
- **Change Management Service** for tracking changes between related artefacts and enabling **change impact analysis** on the models involved in Model Based Security Testing

	<p align="center">Initial Model-Based Security Testing Methods</p> <p align="center">Deliverable ID: D3.WP2</p>	Page : 39 of 46
		Version: 1.13
		Date : 06.07.2012
		Status : Final Confid : Public

- **Automatic Test Case Generation Service** for realizing the automated generation of test code based on the relations captured in traces

In order to handle all elements in the connected models, the platform needs a mechanism to reach the corresponding models and their elements.

3.1.6.3 Concept of a Query Language

In order to utilize TMP, services running on top of it should be provided with the possibility to reach every single element in all connected models. Thereby two kinds of traces should be considered: 1) the **implicit relationship** inside a model, such as the “harm” relation in CORAS that connects an unwanted incident to an asset in a risk model and 2) the **explicit specified trace** between elements in different model domains. The first one is contained in each model instance as structural element. The second type of traces is kept and managed by the TMP. For services running on top of TMP, the interface to reach the elements has to be very abstract and lightweight but powerful and flexible at the same time. Hence, a query language is the better choice than providing simple getter/setter type of APIs (Application Programming Interface). For the research prototype, we considered a proprietary concept for the queries within TMP. Our query interface specifying the input and output types as well as constraint methods for a query is shown in Figure 28. These constraint methods can be used by the service to specify which elements should be included in a query result and which not. Furthermore, these constraints can influence the depth of transitivity thereby limiting the query results. In addition, the services can define white/black lists that explicitly include/exclude elements from the query. This model is evolving in the course of applying it to the case studies of the DIAMONDS project.

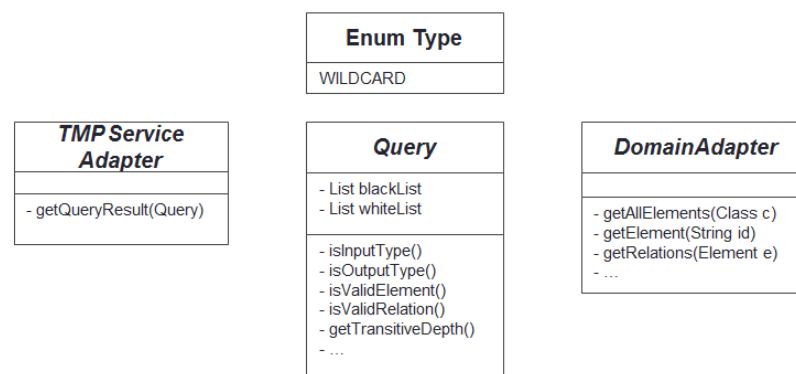



Figure 28. Query Interface Draft

3.1.7 Application to the Banking Domain Case Study (G&D)

As previously mentioned, TMP provides APIs that enable the implementation of services running on top of it. These services utilize the traceability platform in order to automate different aspects of the test management process in a particular environment. We believe that in diverse environments and contexts, specific services should be developed that target relevant important aspects of the model based security testing process in this concrete field of application. Hence, TMP only provides the platform that in turn can be utilized in a particular scope by a service that uses the relations between the model artefacts. Thereby, the service employs the above described query language and underlying trace models. Here, we give a short example of how such a test case generation service can be developed for the banknote processing machine case study (Giesecke & Devrient) of the DIAMONDS project.

Firstly, in the course of the banknote processing machine case study, a risk model was developed that captures potential threats, vulnerabilities and threat scenarios. These artefacts can be linked and

	<p>Initial Model-Based Security Testing Methods</p> <p>Deliverable ID: D3.WP2</p>	Page : 40 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

correspondingly traced to existing security functional requirements (SFR). These are in turn linked to security test patterns (refer to relevant activities in WP4). The security test patterns recommend an approach to testing SFRs and correspondingly (over the transitive link) vulnerabilities from the risk model. Given that all these artefacts where linked (and can be traced in-between) using the above described concepts, a service can follow the traces and can reach to the approach which would also offer a method for generating tests in order to test a particular vulnerability. In the scope of the banknote processing machine, the risk model and the belonging SFRs indicated that some special testing effort should be dedicated to verify that the authentication process works correctly. The corresponding security test pattern recommends a fuzzing approach (the belonging trace has been set up by the test manager). Hence, the service would follow all this traces until it reaches the fuzzing approach and would trigger it to generate a number of fuzzed test cases to execute against the test target.

3.2 MIRRORING RISK ANALYSIS TECHNIQUES TO TELECOM USE CASE


3.2.1 Existing Risk Assessment Technique

In general for any telecom use case (e.g. a node, a product, a solution), product related risk assessment is run at Ericsson. This risk assessment is aligned with well known standards and best industry practices e.g. ISO/IEC 17799, ISO/IEC 27001, ISO/IEC 27011, COBIT and ISO 13335.

The initial high-level risk evaluation is done at an early stage to enable early impact on the specification of the product. For a new release of an existing product, risk assessments made for the preceding release can be built upon, thereby focusing on the impact of the incremental changes in the new release. In Agile product development risk assessment as well as security testing is a continuous process during development.

The following activities are usually included in the risk assessment process:

- Planning and Scoping, including quick system analysis
- Actual risk assessment workshop addressing the following items
 - Identify Assets
 - Identify Threats and Vulnerabilities
 - Review Existing Security Controls
 - Evaluate Risks
- Propose Risk Treatments and Report Results
- Mitigation Plan

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 41 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

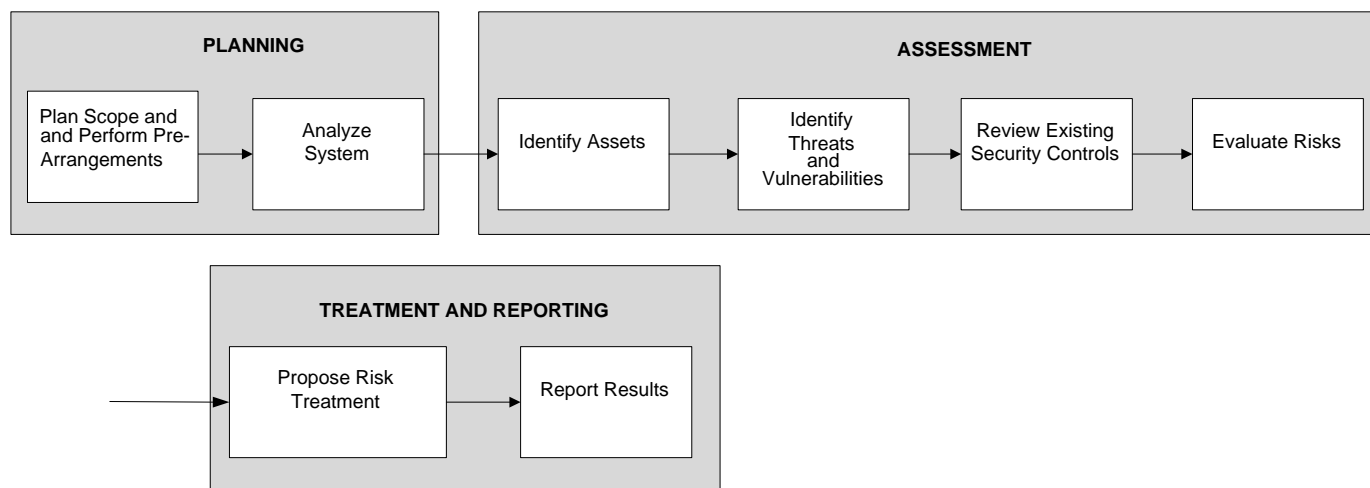


Figure 29. Activities of a risk assessment process

The results of the Risk Assessment are used as input to planning of Vulnerability Analysis, which takes place later in the development phase.

3.2.2 Mirroring techniques to telecom use case

In order to enhance both the existing risk assessment process and the actual vulnerability testing activities, risk techniques from other consortium partners have been studied and mirrored to the telecom use case.

The following sections describe considerations on this mirroring activity specifically for risk modelling techniques, risk assessment based on vulnerability analysis and risk related testing techniques. It is evaluated whether the specific technique is already in use or if it should be considered to be experimented with.

3.2.3 Risk modelling techniques

3.2.3.1 Bayesian Networks

A Bayesian network modelling technique may help to understand the causes and their interrelations for a risk (or vulnerability) to realize, as it categorizes causes into organizational factors, human factors and technical factors.


This could bring in another type of systematic approach looking into different factors and their interrelations and help to shift focus from pure technical factors to other areas as well. It may reveal also that testing for some risks/vulnerabilities can be down prioritized due to factors how they are being realized.

Bayesian Networks is a candidate for further experimentation within the telecom use case.

3.2.3.2 CORAS Risk Modelling

CORAS threat diagrams are an interesting way to visualize how different threats exploit vulnerabilities and can be seen as an interesting tool to be introduced to risk analysis performed for telecom nodes. The benefit is seen on the intuitiveness and readability of the charts and hence worth trying in actual risk assessment.

CORAS Risk Modelling is a candidate for further experimentation within the telecom use case, e.g. trying to combine it with the current practice of using mindmaps as visualization tool.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 42 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

3.2.3.3 Markov Diagrams

As Markov models are more suitable for showing the operation modes of a system that may transit between states, it might be suitable for planning test cases for telecom nodes (these nodes usually have lot of different states) and a valid tool for threat modelling provided it is an approach not too academic. Markov Diagrams could be a candidate for further experimentation with regards to specific threats that trigger the target system into different states.

3.2.3.4 Trick-Light

From Trick-Light model the most interesting part is the Risk Reduction Factor (RRF) calculation associated identified risks. However, the current Risk Assessment methodology in use has already well proven risk rating schema.

Trick-Light could be a candidate for experimenting with different risk rating schemas and to compare its results with the currently used risk rating methodology.

3.2.4 Risk assessment based on vulnerability analysis

3.2.4.1 Attack vectors and Attack surface analysis

Attack surface analysis would be applicable to test scenarios where there are more targets to be tested than just one node, e.g. a network of interrelated nodes and either simulated or real traffic running between those nodes. In an environment where usually a single node is tested in standalone configuration, this is not feasible approach.

Attack surface analysis is of general interest, but not feasible for a standalone telecom use case in Diamonds context.

3.2.4.2 CVSS vulnerability metrics

CVSS provides an open industry standard framework for vulnerability metrics that certainly makes conceptually sense. However, there is certain discrepancies between the three rating categories CVSS is consisting of which may result in misleading actual scoring. Statistically the most common CVSS score is 7-8 resulting in priority 1 vulnerabilities. Some consideration shall be given to actual scoring, but it nevertheless provides widely used, additional tool for categorizing findings from a vulnerability analysis.

CVSS is a candidate for rating vulnerabilities, taking into considerations generic well-known issues with this metrics.

3.2.4.3 Risk based planning for a technical security test/audit

The purpose of risk based planning for attack surface analysis is to identify the most critical interfaces and to help in prioritizing the testing effort.


This is already the current practice used for telecom use cases.

3.2.5 Risk related testing techniques

3.2.5.1 Risk-based testing process

The purpose of risk-based testing is to cover most risky areas and focus on areas with highest risks.

This is covered by the existing practices and processes already where risk analysis results are used as input to vulnerability analysis planning.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 43 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

3.2.5.2 Test specification and modelling strategies

Model based testing and test specification could be enhanced by experimenting with new risk modelling techniques from 2.1. Partial test automation for the telecom use case might be possible to achieve.

3.2.5.3 Fuzz test generation strategies

Fuzzing is amongst the existing test methods for telecom use cases. As part of the scope for telecom use case new interfaces are being fuzzed.


3.2.5.4 Common criteria evaluation

Common criteria is used for certain type of equipment, e.g. firewalls and routers. However, for telecom equipment (e.g. base stations etc.) there are no Protection Profiles defined at all.

Common criteria is not relevant for telecom use cases within DIAMONDS project due to lack of existing protection profiles.


3.2.6 Conclusions

In practice the objective is to try to embed some of the new techniques, specifically risk modelling techniques, to the existing risk assessment process and test planning. The expected output is to have a separate section in a risk treatment plan that provides direct input to security testing planning.

	<p>Initial Model-Based Security Testing Methods</p> <p>Deliverable ID: D3.WP2</p>	Page : 44 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public


GLOSSARY (ADAPTED FROM [CNSS 4009])

Risk Analysis	The process of identifying, prioritizing and estimating risks. This includes determining the extent to which adverse circumstances or events could impact an enterprise. Uses the results of threat and vulnerability assessments to identify risk to organizational operations and evaluates those risks in terms of likelihood of occurrence and impacts if they occur. The product of a risk assessment is a list of estimated, potential impacts and unmitigated vulnerabilities.
Security Requirements	Requirements levied on an information system that are derived from applicable laws, executive orders, directives, policies, standards, instructions, regulations and organizational mission/business case needs to ensure the confidentiality, integrity and availability of the information being processed, stored or transmitted.
Security Requirements Traceability Matrix	Matrix that captures all security requirements linked to potential risks and addresses all applicable security requirements. It is, therefore, a correlation statement of a system's security features and compliance methods for each security requirement.
Security Test Generation Model	A behavioural model that defines the expected behaviour of a System Under Test, specifically designed for security test generation.
Security Test Objectives	The definition of security test objectives that are derived from risk analysis and security requirements.
Security Test Purposes	Formal artefacts that act as test selection criteria in the model-based security test generation process. Security Test Purposes are related to a model for test generation.

	<p style="text-align: center;">Initial Model-Based Security Testing Methods</p> <p style="text-align: center;">Deliverable ID: D3.WP2</p>	Page : 45 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

REFERENCES

- [1] Bachar Wehbi, Edgardo Montes de Oca and Michel Bourdellès. *Events-Based Security Monitoring Using MMT Tool*. In the Third International Workshop on Security Testing affiliated with ICST 2012. Montreal, Quebec, Canada. April 21, 2012.
- [2] W. Afzal, R. T. Clausen, P. J. (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot. Optimized link state routing protocol (OLSR). RFC 3626, October 2003. Network Working Group.
- [3] M. Bertoluzzo, G. Buja, and S. Vitturi, "Ethernet networks for factory automation," in Industrial Electronics, 2002. ISIE 2002. Proceedings of the 2002 IEEE International Symposium on, vol. 1, 2002, pp. 175 – 180 vol. 1.
- [4] M. Utting and B. Legeard, *Practical Model-Based Testing – A Tools Approach*, Morgan&Kauffmann, 2007.
- [5] CNSS 4009 - Committee on National Security Systems – Instruction n°4009, National Information Assurance Glossary, 26 April 2010. http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf
- [6] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties", Information and Software Technology, vol. 51, pp. 957–976, 2009.
- [7] Wolfram Amme, Peter Braun, Eberhard Zehendner, and Francois Thomasset, "Data Dependence Analysis of Assembly Code", Research Report RR-3764, INRIA, 1999.
- [8] Gogul Balakrishnan and Thomas Reps, "Analyzing memory accesses in x86 executables", In Proc. Int. Conf. on Compiler Construction, pages 5-23. Springer-Verlag, 2004.
- [9] Saumya Debray and Robert Muth, "Alias analysis of executable code", In POPL, pages 12-24, 1998.
- [10] Fabien Duchene, Roland Groz, Sanjay Rawat, Jean-Luc Richier, "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing", In proc. of the Third International Workshop on Security Testing (SECTEST), in association with ICST 2012, Montreal, IEEE CS press, April 2012
- [11] Mozilla Foundation Security Advisory 2012-21. "Multiple security flaws fixed in FreeType v2.4.9", <http://www.mozilla.org/security/announce/2012/mfsa2012-21.html>
- [12] Sanjay Rawat and Laurent Mounier, "Finding Buffer Overflow Inducing Loops in Binary Executables". To appear: In Proc. of the IEEE International Conference on Software Security and Reliability (SERE) 2012, June 2012, Washington DC, USA.
- [13] Sans, "2011 CWE/SANS Top 25 most dangerous software errors," <http://cwe.mitre.org/top25/>
- [14] "SPaCloS Project no. 257876, FP7-ICT-2009-5. <http://www.spacios.eu>
- [15] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications", in Symposium on Principles of Programming Languages, 2006, pp. 372–382.
- [16] Takanen, Ari, Jared DeMott, and Charles Miller (2008), "Fuzzing for software security testing and quality assurance", Artech House Publishers.
- [17] Mallouli, W., Diamonds deliverable D2.WP2: "Concepts for Model-Based Security Testing"
- [18] Jürjens, J.: Secure Systems Development with UML, Springer, 2005, ISBN 3540007016
- [19] Sandhu, R. Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control Models. IEEE Computer February 1996; 38-47

	<div>Initial Model-Based Security Testing Methods</div> <div>Deliverable ID: D3.WP2</div>	Page : 46 of 46
		Version: 1.13 Date : 06.07.2012
		Status : Final Confid : Public

- [20] Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, R. Chandramouli, R.: Proposed NIST standard for role-based access control. In: ACM Transactions on Information and System Security (TISSEC) 3 (4), ACM, New York (August 2001)
- [21] V. Katta, T. Stålhane: A Conceptual Model of Traceability for Safety Systems, proceedings of Springer-Verlag
- [22] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem", Proc. of the IEEE International Conference on Requirements Engineering (ICRE), 1994.
- [23] Federal Aviation Administration: System Safety Handbook.
http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/, as of date 4.5.2012
- [24] Joint Software System Safety Committee: SOFTWARE SYSTEM SAFETY HANDBOOK. A Technical & Managerial Team Approach
http://www.system-safety.org/Documents/Software_System_Safety_Handbook.pdf, as of date 4.5.2012
- [25] John Joseph Chilenski: Software Development under DO-178B
<http://www.opengroup.org/rtforum/jan2002/slides/safety-critical/chilenski.pdf>, as of date 4.5.2012
- [26] Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, Thierry Coq: A SysML-Based Approach to Traceability Management and Design Slicing in Support of Safety Certification: Framework, Tool Support, and Case Studies
http://simula.no/publications/Simula.simula.193/simula_pdf_file, as of date 4.5.2012
- [27] The CreMA tool (itemis): <http://www.guersoy.net/knowledge/crema>, as of date 4.5.2012