

	Final Security Testing Tools Deliverable ID: D5.WP3	Page : 1 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

	Final Security Testing Tools		
	Version: 1.0 Date : 22.05.2013 Pages : 80		
	Editor: Matti Mantere		
	Reviewers: Fredrik Seehusen, Boutheina Chetali, Stephane Maag		
	To: DIAMONDS		
The DIAMONDS Consortium consists of: Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Get IT, Giesecke & Devrient, Grenoble INP,itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, TU Graz, University Oulu, VTT			
Status: [] Draft [] To be reviewed [] Proposal [X] Final / Released		Confidentiality: [X] Public Intended for public use [] Restricted Intended for DIAMONDS consortium only [] Confidential Intended for individual partner only	

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 2 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

Deliverable ID: D5.WP3

Title:

Final Security Testing Tools

Summary / Contents:

This document describes the final security testing tools in DIAMONDS project.

Contributors to the document:

Matti Mantere (VTT), Sanjay Rawat (Grenoble INP), Fabien Duchène (Grenoble INP), Jean-Luc Richier (Grenoble INP), Julien Botella (SMT), Bruno Legeard (SMT), Pramila Mouttappa (IT) Wissam Mallouli (Montimage), Edgardo Montes de Oca (Montimage), Stephan Pietsch (Testing Technologies), Bogdan Stanca-Kaposta (Testing Technologies), Kati Kittilä (Codonomicon), Ari Takanen (Codonomicon), Jürgen Großmann, Martin Schneider, Michael Berger (Fraunhofer FOKUS)



	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 3 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

TABLE OF CONTENTS

1. Introduction.....	8
2. DIAMONDS Security Testing tools	9
2.1 Data Fuzzing Library (FhG FOKUS)	9
2.1.1 Description of the Tool	9
2.1.2 Application to Case Studies	13
2.1.3 Advances during DIAMONDS	14
2.2 Model-based behavioral security testing tool development (Smartesting)	17
2.2.1 Description of the Tool	17
2.2.2 Application to case studies	17
2.2.3 Advances during DIAMONDS	18
2.3 FRAMEwork for active security testing (FSCOM)	24
2.3.1 Description of the Tool	24
2.3.2 Application to Case Studies	27
2.3.3 Advances during DIAMONDS	29
2.4 Static and Dynamic Application Analysis for Vulnerability Detection	31
2.4.1 Description of the Tool	31
2.4.2 Application to Case Studies	40
2.4.3 Advances during DIAMONDS	40
2.5 Codenomicon Defensics	41
2.5.1 Description of the Tool	41
2.5.2 Application to Case Studies	45
2.5.3 Advances during DIAMONDS	46
2.6 Symbolic Passive Testing Tool (TestSym-P) (IT)	47
2.6.1 Description of the Tool	47
2.6.2 Application to Case Studies	51
2.6.3 Advances during DIAMONDS	51
2.7 Montimage Monitoring Tool (Montimage)	52
2.7.1 Description of the Tool	52
2.7.2 Application to Case Studies	53
2.7.3 Advances during DIAMONDS	54
2.8 Malwasm (itrust)	55
2.8.1 Description of the Tool	55
2.8.2 Application to Case Studies	59
2.8.3 Advances during DIAMONDS	59
2.9 TRICK Tester (itrust)	60
2.9.1 Description of the Tool	60
2.9.2 Application to Case Studies	64
2.9.3 Advances during DIAMONDS	64
2.10 TTCN-3 Fuzz testing	65
2.10.1 Description of the Tool	65
2.10.2 Application to Case Studies	67
2.10.3 Advances during DIAMONDS	67
3. Integration platform.....	68
3.1 Tools integration for Security Testing	68
3.1.1 Integration into the Radio Protocol Framework	68
3.1.2 Tools Integration:	68
3.2 Trace Management Platform for Risk-based Security Testing (FhG FOKUS)	69
3.2.1 Description of the Tool	70
3.2.2 Application to Case Studies	73
3.2.3 Advances during DIAMONDS	74
4. Conclusion	78

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 4 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

References..... 79

FIGURES

Figure 1: Excerpt from an XML Request File	12
Figure 2: Excerpt from an XML Response File	12
Figure 3: Internal Architecture of the Data Fuzzing Library	15
Figure 4: The Class ComputableList and its Relationships	16
Figure 5: Smartesting process and tool for model-based security testing	17
Figure 6: Smartesting process applied on the Thales case study	18
Figure 7: Smartesting process applied on the Gemalto TSM case study	18
Figure 8: Smartesting Test Purpose and Keywords editors	20
Figure 9: Smartesting test model import from TSM StateChart plugin	21
Figure 10: Smartesting test model import from TSM StateChart importer	21
Figure 11: UML/OCL model parts generated from a TSM StateChart	22
Figure 12: Generated test sequence from TSM StateChart imported model	22
Figure 13: Gemalto TSM interfaces	24
Figure 14: Security test system architecture	26
Figure 15: Generic abstract protocol tester	26
Figure 16: Execution of the testing framework screenshot	28
Figure 17: Malicious attack which set the HTTP tag Content-Length to 65535	29
Figure 18: LiSTT architecture	31
Figure 19: Generating Arg/Var information from IDA Pro	32
Figure 20: Importing .IDB file into BinNavi	33
Figure 21: Slice Class in LiSTT	33
Figure 22: Various classes that represent dataflow analysis in LiSTT	34
Figure 23: A taint flow slice as computed by LiSTT. TScr= j_fgets; TDst= j_strcat	35
Figure 24: KameleonFuzz Tool Architecture	36
Figure 25: KameleonFuzz: High Level Approach Flow	36
Figure 26: Extract of the Inferred Model for P0wnMe	37
Figure 27: Example of Reflection Annotation (this is an extract of the p0wn model, with only transitions related to the reflection)	38
Figure 28: Genetic Algorithm	39
Figure 29: Extract of the Attack Grammar	39
Figure 30: extract of the found XSS exploit summary	39
Figure 31: Specification-based approach	41
Figure 32: Test-case generation	42
Figure 33: Test target information	42
Figure 34: Step 1. Load PCAP file	43
Figure 35: Step 2. Select protocol elements	43
Figure 36: Model-editing in Defensics	45
Figure 37: Architecture of TestSym-P prototype model.	47
Figure 38: Snapshot of trace parsing (dbo.InputToExcel) table.	48
Figure 39: Snapshot of the trace slicing (dbo.slices) table.	49
Figure 40: Snapshot of the verdicts obtained	49
Figure 41: Snapshot of the guard-conditions table.	50
Figure 42: Snapshot of the Symbolic state details table.	50
Figure 43: MMT-Security Architecture	52
Figure 44: malwasm features	56



	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 5 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

Figure 45: malwasm web-interface	57
Figure 46: malwasm architecture.....	58
Figure 47: Tools integration (tools provider view).....	68
Figure 48: Traceability from risk assessment artefacts to test results	69
Figure 49: Vulnerability coverage by test cases.....	71
Figure 50: Trace Management Framework in Multi-Layer Diagram	71
Figure 51 : Example of the Option Create Trace with the Trace Management Tool	73
Figure 52: Traceability Management embedded in Eclipse (based on CReMa).....	75
Figure 53: Different Domains in Context of Risk-based Security Testing	75
Figure 54: An Example of a Trace Metamodel in Context of Risk-based Security Testing	76
Figure 55: Query Interaction between all Query-related Components	77

TABLES

Table 1: Chosen Fuzzing Heuristics from Selected Fuzzers	10
---	-----------


	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 6 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

HISTORY

Vers.	Date	Author	Description
0.1	2013/3/13	M. Mantere	Document creation
0.5	2013/5/8	M. Mantere	Refactoring the document
0.99	2013/5/22	M. Mantere	Edited based on reviews

APPLICABLE DOCUMENT LIST


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

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 7 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

EXECUTIVE SUMMARY

The DIAMONDS project focuses on model-based security testing of networked systems. Testing is the main method to reliably check that a software-based system meets its requirements with regard to functionality, security and performance. In this document we present the final state of security testing tools touched during DIAMONDS. The tools have been either fully developed during the project or their functionality enhanced.

Model-based Testing is the approach of deriving systematic tests for a system based on an abstract representation thereof called models. These models may describe the behaviour of the system, security constraints (for example access control), the security requirements, or information about possible security threats, faults or attacks. The state of the art in model-based testing tools was given in D1.WP3 [6] and the initial report on testing tools in D3.WP3. In this document D5.WP3 we describe the final model-based security testing tools by the different project partners. This document can be considered a final progress report of each partner developing its testing tool.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 8 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

1. INTRODUCTION

The document comprises of the testing tools that have been created or further developed during the DIAMONDS project. The section 2 comprises of individual tool descriptions, while section 3 gives an overview of the integration platforms.

This document is the final deliverable for work package 3 in the project.

2. DIAMONDS SECURITY TESTING TOOLS

2.1 DATA FUZZING LIBRARY (FHG FOKUS)

2.1.1 Description of the Tool

Today, data fuzzing is a widely accepted and performed technique for security testing. There are many different fuzzers available, some are commercial, many are open-source. The main difference between the commercial and the open source fuzzers is that the latter are often developed for a specific protocol, while the commercial fuzzers address a large number of different protocols. Hence, for testing different systems, several fuzzing tools are needed when considering open source tools. Additionally, Charles Miller came to the conclusion that the more fuzzers are used the better ([22], [24] p. 242) in order to find the most weaknesses. Following that, for performing fuzz testing a lot of fuzzing tools should be used. Furthermore, each tool has to be configured for the specific SUT requiring the tester to become acquainted with each tool. The presented data fuzzing library solves this problem. It combines the core of several open source fuzzing tools – their test data generators – and makes them available for other tools over a single interface. Test data generators, that are fuzzing heuristics, consist of fuzzing generators and fuzzing operators. Fuzzing generators are producing values from a specification while fuzzing operators are modifying valid values.

2.1.1.1 Existing Open Source Fuzzing Tools

Because of the huge number of tools (see [23] for a list of tools available in 2005), a study of open source fuzzing tools was conducted. The study investigated fuzzing tools w.r.t the following aspects:

- **target of fuzzing attacks:** This could be a specific protocol, e.g. SOAP, or for instance Java classes or regular expressions.
- **last source code update:** This information is used to estimate whether the fuzzer is further developed.
- **category of fuzzer:** Random-based, block-based, generation- or mutation-based.
- **existence of a library interface:** If the fuzzer can be used as a library through an interface, its integration would be easier.
- **fuzzing heuristics:** What kind of fuzzing heuristics are implemented. That is the most interesting part because the fuzzing heuristics constitute the functional core of the library.
- **license:** Does the license of the fuzzer allow an integration within the planned library.

After evaluating ten open source fuzzing tools, three were selected for the initial implementation of the fuzzing library:

- **Peach** is an open source smart fuzzer that is under active development since 2006 and also commercially offered by Déjà vu Security. It can perform generation and mutation-based fuzz testing employing a data model and a state model. It has a rich set of generators and operators, e.g. for Unicode string, hostnames, variation and extreme numbers, i.e. going to the edge of an interval.
- **Sulley** is a block-based fuzzer for generation and mutation-based fuzzing. It is under development since 2008, and like Peach, it also has various fuzzing generators and operators. Additionally, Sulley uses many fuzz testing values from SPIKE, a generation-based fuzzing framework.

First, the data fuzzing library provides a uniform interface to access the fuzzing heuristics of the above mentioned tools. This interface and the library itself is open for further extensions. The chosen fuzzing heuristics are shown in Table 1.

Fuzzing Heuristic	Peach	Sulley
Strings		
BadDate	G	
BadIpAdress	G	

Fuzzing Heuristic	Peach	Sulley
BadNumbers	G	
BadTime	G	
Command Injection		G
Delimiter	G	
FilenameMutator	G	
FiniteRandomNumbersMutator	G	
Format String		G
HostnameMutator	G	
LongString		G
PathMutator	G	
SQL Injection		G
String Repetition		O
StringCaseMutator	O	
StringMutator	G	
UnicodeBadUtf8Mutator	G	
UnicodeBomMutator	G	
UnicodeStringsMutator	G	
UnicodeUtf8ThreeCharMutator	G	
Numbers		
Numerical Edge Case Mutator	G	G
Numerical Variance Mutator	O	

Table 1: Chosen Fuzzing Heuristics from Selected Fuzzers

2.1.1.2 Use Cases

Two main use cases for the fuzzing library were identified:


- **Using fuzz testing in SUT specific test execution environments.** Most fuzzing tools integrate the definition of the data format, the test execution and test verdict arbitration. This requires the tester a) to connect the fuzzing tool to the SUT, which could mean a significant effort, especially for embedded systems, and b) to get familiar with the syntax of the data and to specify its format for each fuzzing tool.

The data fuzzing library faces these drawbacks by providing a unified interface for accessing fuzzed values without losing the power of fuzzing heuristics from different fuzzing tools. The test execution environment can request fuzzed values from the library and sent them to the system under test using its specific connection to the interfaces of the SUT.

- **Extending test tools with fuzz testing capabilities.** Fuzz testing is one test method beside others. Other test methods are already supported by test tools implemented in an existing test process. To integrate fuzz testing into an existing test process can lead to substantial effort because new tools have to be adapted and fitted to the whole test process. The fuzzing library enables to extend existing test tools with fuzz testing capabilities acting as a library from which existing tools can request fuzz testing values.

2.1.1.3 Requirements

In order to make a fuzzing library widely adoptable, the following requirements were figured out:

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 11 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

- **platform independence:** The library shall be available on many different platforms. At best on as many platforms as are used for existing test tools and test execution environments.
- **independent from programming languages:** As for platforms, the library shall be accessible from as many programming languages as possible. Hence, the library should not use a language specific interface but should be open for many programming language.
- **largely independent of type representation:** It should be possible to describe and fuzz a wide range of different data types without the limitation of the programming languages used for implementing the library.
- **efficient to use:** The user shall have the possibility to specify which fuzzing heuristics shall be used. This allows adjusting the library to fit specific requirements, e.g. only generate fuzz testing values based on Unicode. Because fuzzing generates a large set of values, the user must have the possibility to request a certain number of fuzz testing values.
- **transparent:** The fuzzing library shall tell its user which fuzzing heuristics were used, so more values only from a certain fuzzing heuristic can be requested. This is useful if a fuzz testing value generated by a certain fuzzing heuristic revealed a weakness in the system under test.
- **repeatability:** Fuzz testing is often a random-driven approach meaning that values are generated in partial randomly. The library shall support the repeatability of such randomly generated values, e.g. for regression testing.
- **extensibility:** Time is going further, and new fuzzing tools with new fuzzing heuristics will come. The library shall be extensible to meet these upcoming developments.


2.1.1.4 Interfaces Provided by the Library

The requirement of language independency has a big impact on the interface provided by the library. It must be accessible from different programming languages. Additionally, the interface must be able to handle many different representation formats. Hence, XML seems to be an appropriate choice for accessing the library and receiving its output.

In order to receive fuzzed values from the fuzzing library, a request must be submitted to the library. Such a request contains the relevant information of a type that shall be fuzzed, e.g. valid lengths and null termination for a string, as shown in Figure 1. Additional information are the number of values to be retrieved (attribute `maxValues`) as well as a name acting as a user-defined identifier (attribute `name`) that can be used for referring this type.

The following types are supported:

- **Strings:** Different kinds of strings, including filenames, hostnames, SQL query parameters.
- **Numbers:** Integers and floats, signed or unsigned with different kinds of precisions.
- **Collections:** Lists and sets. The type of each element is specified by referring one of these four types (strings, numbers, collections, or data structures) using the value of the name attribute.
- **Data structures:** Enables the specification of records with several fields where the type of each field is specified by referring one of these four types (strings, numbers, collections, or data structures) using the value of the name attribute.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 12 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

```
<string name="SimpleStringRequest" maxValues="10">
  <specification type="String" minLength="1" maxLength="5" nullTerminated="true"
    encoding="UTF8" />
  <generator>BadStrings</generator>
  ...
  <validValues>
    <value>ABC</value>
    ...
    <operator>StringCase</operator>
    ...
  </validValues>
</string>
```

Figure 1: Excerpt from an XML Request File


Along with the specification of the data type, it is possible to specify which fuzzing heuristics shall be used and which valid values shall be fuzzed. This is of particular interest if a specific kind of invalid input data is needed, e.g. based on Unicode strings. This allows it to efficiently use the fuzzing library to get certain fuzzed values.

The response of the fuzzing library is also an XML file whereof an excerpt is shown in Figure 2. The response contains the fuzzed values according to the basic request type (in this case string) given by the request. Moreover, there are two new attributes for the tag string. moreValues denotes if further values than the enclosed can be retrieved from the library. Finally, id stands for a UUID that is necessary to retrieve further values after receiving the first response from the library.

The fuzzed values are complemented by information how the fuzzed values were generated by the library. They are grouped by the employed fuzzing generators – for fuzzed values that are generated along the type specification. Moreover, the employed fuzzing operators, and the valid values they were applied to. This makes the generation of fuzzed values transparent to the user of the library, and further request of the fuzzed values generated by specific fuzzing operators if a previous value revealed some abnormal behaviour of the SUT.

```
<string name="SimpleStringRequest" id="ca53abee-0719-43da-a70d-96d61931fb08"
  moreValues="true">
  <generatorBased>
    <generator name="BadStrings">
      <fuzzedValue>+]s}9$# *Y</fuzzedValue>
      <fuzzedValue>0$2)v3D^U1_{X7x,Us\\</fuzzedValue>
      ...
    </generator>
    ...
  </generatorBased>
  <operatorBased>
    <operator name="StringCaseOperator" basedOn="ABC">
      <fuzzedValue>abc</fuzzedValue>
      <fuzzedValue>aBc</fuzzedValue>
      ...
    </operator>
    ...
  </operatorBased>
</string>
```

Figure 2: Excerpt from an XML Response File

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 13 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

The format of the request file as well as the format of the library's response file is specified using an XML schema. The parser and serializer for the XML are generated from those XML schemata using the Eclipse Modelling Framework (EMF).

2.1.2 Application to Case Studies

The presented fuzzing library was applied in the German case studies: the Automotive case study provided by Dornier Consulting, and the Banking case study provided by Giesecke & Devrient.

2.1.2.1 Automotive Case Study


The Automotive case study was provided by Dornier Consulting. It consists of the connection of the car's entertainment system with the driver's mobile phone via Bluetooth. Connections via Bluetooth are open to foreign devices and thus, provide an attack point. In order to check the robustness of the Bluetooth interface, fuzzing is an appropriated testing technique. However, the goals of testing the interface is whether an attack via the Bluetooth interface has an impact on the automotive devices connected to the Bluetooth module within the car. The Bluetooth module and the car's entertainment system communicate between the CAN bus using messages. These messages are specific for each car manufacturer. Therefore, using different tools for different kinds of tests, e.g. functional and security tests, constitutes a significant effort that impedes using fuzzing tools. The fuzzing library helps to reduce this effort by allowing to perform fuzzing with existing functional testing tools. For that purpose, it was integrated in Dornier Consulting's do.Atoms model-based testing tool and provide the fuzz test data. do.Atoms accesses these fuzz test data via the library's interface in order to retrieve invalid host names for Bluetooth devices that are trying to connect to the car's entertainment system.

The case study addresses both use cases of the fuzzing library, the extension of an existing test tool for performing security testing using fuzzing, and by allowing fuzzing for a SUT with a specific interface, in this case Bluetooth and the CAN bus for test verdict arbitration. Full fuzz testing tools needed to be extended in order to decode the CAN message for test verdict arbitration as well as for accessing the Bluetooth interface. The fuzzing library allows for avoiding these effort by the much easier integration of the library with the existing test tool.

2.1.2.2 Banking Case Study

The Banking case study was provided by Giesecke & Devrient. It is a banknote processing system consisting of two machines: a currency processor that is automatically scanning and assessing banknotes and a reconciliation station for manual reassessment of banknotes that were automatically rejected by the currency processor. The main focus of the security tests was on the currency processor. We applied data fuzzing in order to test the robustness of the interface against SQL injection. SQL injection tries to inject SQL commands via the interface of an application in order to bypass authentication mechanisms or to manipulate the database of an application.

Testing for SQL injection vulnerabilities requires less effort and can be easily performed by functional testing tools. The fuzzing library was used to generate such SQL injection strings. Existing functional test cases written in TTCN-3 were used as a starting point. The fuzzing library was integrated with TestingTech's test definition and execution environment TWorkbench. A new TTCN-3 extension for fuzzing developed during the DIAMONDS project facilitated this task by an easier integration with existing templates. SQL injection was achieved by requesting corresponding test data from the fuzzing library and saving this data in variables that were submitted to the SUT where SQL injections seems to be possible. The case study addresses mainly the second use case by allowing security testing with existing testing tools mainly developed for functional testing. On the other hand, the SUT was stimulated via a special-purpose interface that is addressed by the TTCN-3 framework and the test adapter. Thus, the first use case is indirectly addressed by allowing reusing the test adapter by the existing TTCN-3 framework and functional test for security testing.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 14 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.1.3 Advances during DIAMONDS

The fuzzing library was newly developed during the DIAMONDS project. In the following, we describe the architectural decisions that were made in order to meet the requirements. Additionally, we describe how the fuzzing library could be extended with new fuzzing heuristics.

In order to meet the requirement for platform independency, the fuzzing library is developed on top of the Java Platform. Java is supported on many operating systems and seems to be appropriate to achieve the goal of availability on various platforms.

To preserve platform independence as achieved within Java and to minimize dependencies, the fuzzing generators and operators taken from the fuzzing tools (see section 2.1.1.1) are reimplemented in Java. This brings benefits for the performance of the library since no integration of Python code (for Peach and Sulley) is required. The Java integration layer for Python, Jython [25], typically requires some significant time for initialization which might hamper the performance of the data fuzzing library. In addition, the realization of the fuzzing heuristics in pure Java improves the maintainability because only one programming language is used for the implementation of the whole library. The reimplementation of the fuzzing operators seems also to be appropriate because the core functionality of a fuzzing operator is generally quite simple and normally the fuzzing heuristics are hard-wired within the enclosing framework. Additionally, for the Jython integration of Python code into Java, there is additional overhead because a Java interface must be written for each Python class, and that Java interface in turn must be referenced in the Python class.

Fuzzing is often a random-driven approach. Fuzzed values are generated under the influence of randomness without following a certain set of deterministic rules. This impedes regression testing because every time fuzzed values are generated they differ from the previously generated ones. To enable regression testing, the fuzzing library returns a seed that can be used for later requests in order to retrieve the same values. Thus, the requirement for repeatability is fulfilled.

2.1.3.1 Architecture

Figure 3 depicts the architecture of the fuzzing. It consists of three layers:

- **Interface:** The library can be accessed using two interfaces independent from each other. The language independent way is through XML for creating a request and getting the response. As described above, XML schemata were defined for determining the syntax of requests and responses. These schemata were used to generate the proper XML parser for request files and the proper XML serializer for response files. However, the XML schemata were used to define the general structure of XML requests on a high level of abstraction, in order to keep the schemata manageable, and allow the extension of the fuzzing library without the need for changing the XML schema when adding new fuzzing heuristics. The second way to access the library is by directly using the Java interfaces thereby saving the time for parsing the XML request and response. For the direct access via Java there are interfaces for data objects according to the information that can be submitted via XML requests and responses.

Whichever way was selected by the user to access the library, the information about the requests is delivered to the request processing layer. After the requests are processed, the interface receives the results from the request processing layer, creates the XML response (employing the EMF generated XML serializer) and delivers the response from the library to the user.

- **Fuzzing Heuristics:** This layer aggregates the various fuzzing generators and operators reimplemented from the selected fuzzing tools (Peach, Sulley). They are grouped by their types (string, number, collection, data structure). For each type, there is a separate factory for generators and operators that creates instances of them according to the specification given by the request. Thus, only suitable fuzzing heuristics are generating fuzzed values.

- Request Processing:** This layer acts as a broker between the interface and the fuzzing heuristics. A request dispatcher receives a bunch of requests from the interface and passes them to type specific request processors, e.g. to a string request processor. Each type specific request processor handles one request by employing the fuzzing heuristics that match the type specification of the request. The results of the fuzzing heuristics are then returned for building the response to the request. That is, the request dispatcher collects all responses from the different, type specific request processors, and gives the aggregation of all responses to the interface layer.

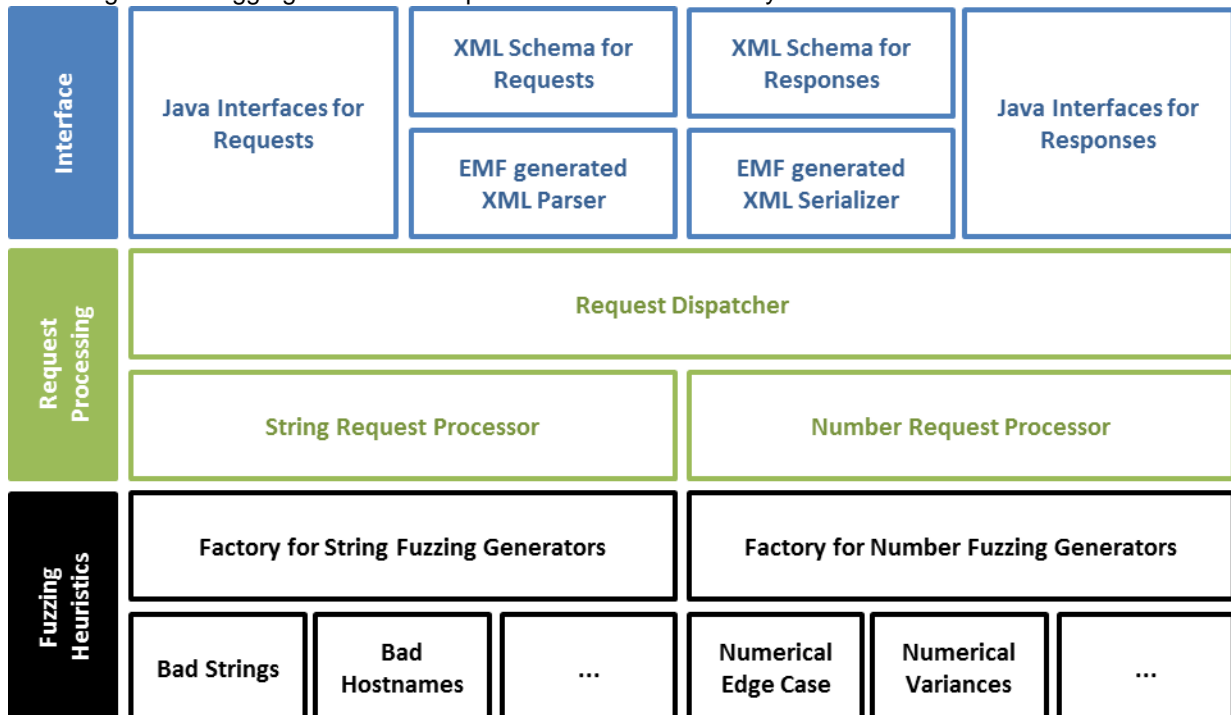


Figure 3: Internal Architecture of the Data Fuzzing Library

Figure 3 depicts the architecture of the fuzzing. It consists of 3 layers:

2.1.3.2 Extending the Library with New Fuzzing Heuristics

As described above, there are two kinds of fuzzing heuristics: **fuzzing generators** that generate fuzzed values based on a specification, and **fuzzing operators** that modify existing, generally valid values. Both kinds of heuristics are supported by the library. Fuzzing generators simply constitute large lists of certain values that are known to have the capability to expose implementation weaknesses. Because of their size, they require a lot of memory at runtime. To overcome this memory issue, two strategies are realized: First, the fuzzed values of the generators are static members of the corresponding classes, in order to avoid duplication during instantiation of a generator class. Secondly, the values are not used as is but generated at request time. Mostly, there is an underlying pattern for the fuzzed values, e.g. many values of the generator Bad String consist of certain characters that are repeated a power of 2 times. Hence, such a value could be very long. In that context, the memory consumption could be limited by generating the fuzzed value when requested. This is not only valid for fuzzing generators, but also for operators, whereas the length of the valid values, fuzzing operators are applied to, is unknown and could be much larger. For that reason, as shown in Figure 4 an interface called `ComputableList` is implemented by an abstract class `ComputableListImpl` that is derived from the `AbstractSequentialList` being part of the Java runtime environment. The abstract class `ComputableListImpl` is the base class of all fuzzing heuristics. Its iterator (`ComputingIterator`) does not iterate a list of values but calls the method `computeElement(int)` in order to obtain a value.

Therefore, a value can be calculated when requested avoiding generating all values when the fuzzing generator or operator is instantiated.

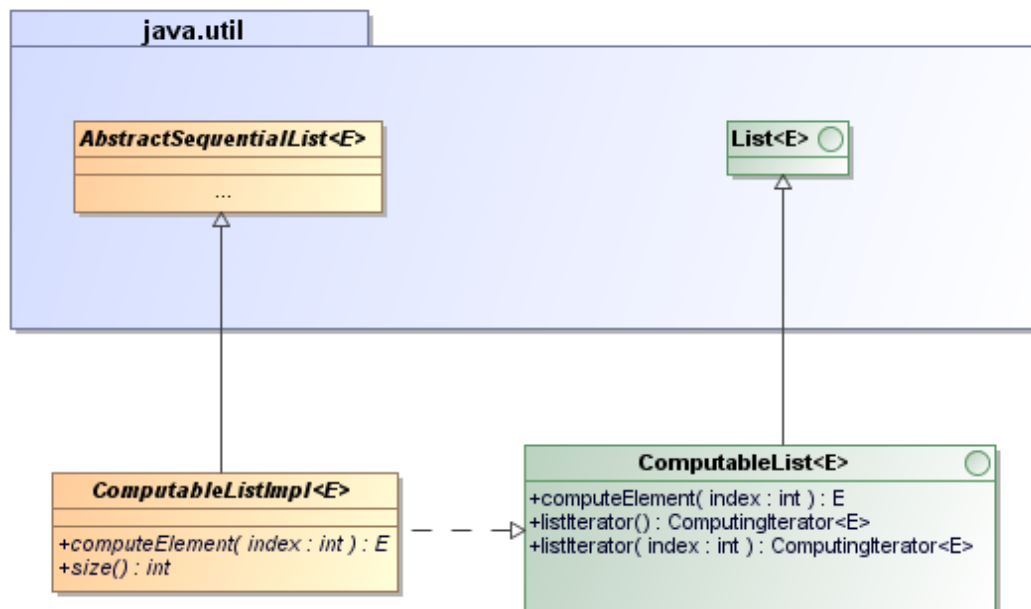



Figure 4: The Class `ComputableList` and its Relationships

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 17 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

2.2 MODEL-BASED BEHAVIORAL SECURITY TESTING TOOL DEVELOPMENT (SMARTESTING)

2.2.1 Description of the Tool

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 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 or she would represent SQL injection operation inside the test generation model. Secondly, the model-based security testing differs in the way test cases are 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 a several) errors cases. In security testing, the goal is really to systematically trying to break (or to bypass) the security functions of the SUT. This search for breaking software security barrier requires to systematically trying possible breakers in a large set of application contexts. The test purpose language goal is to support such test selection criteria.

The Figure 5 presents the Smartesting process and tool for model-based security testing.

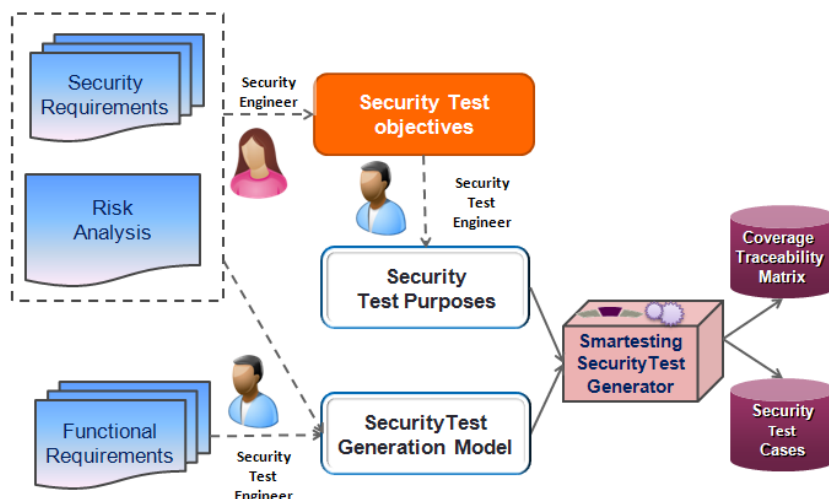



Figure 5: Smartesting process and tool for model-based security testing

2.2.2 Application to case studies

The Smartesting process and tool for model based testing has been used on different case studies during the Diamonds project.

Smartesting worked on the following case studies:

- ltrust LASP case study
- SINTEF/NORSE banking application case study

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 18 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

- Thales HDRM case study
- Gemalto Trusted Service Manager (TSM) case study

On the ltrust case study, only a first functional model has been created, and the generated tests have been executed.

After providing to SINTEF training around the Smartesting process, SINTEF used the solution to generate vulnerability tests around SQL Injection on the NORSE case study.

The more advanced use cases are the Thales and Gemalto use cases, on which the full chain has been applied. The Figure 6 and Figure 7 show the application of the Smartesting process on those use cases. It has been first done on the Thales case study, and then replicated on the Gemalto TSM case study.

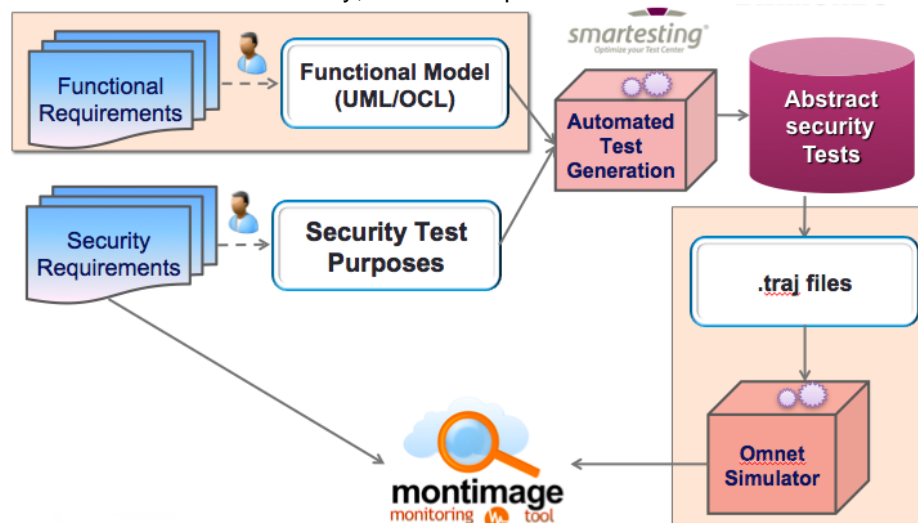


Figure 6: Smartesting process applied on the Thales case study

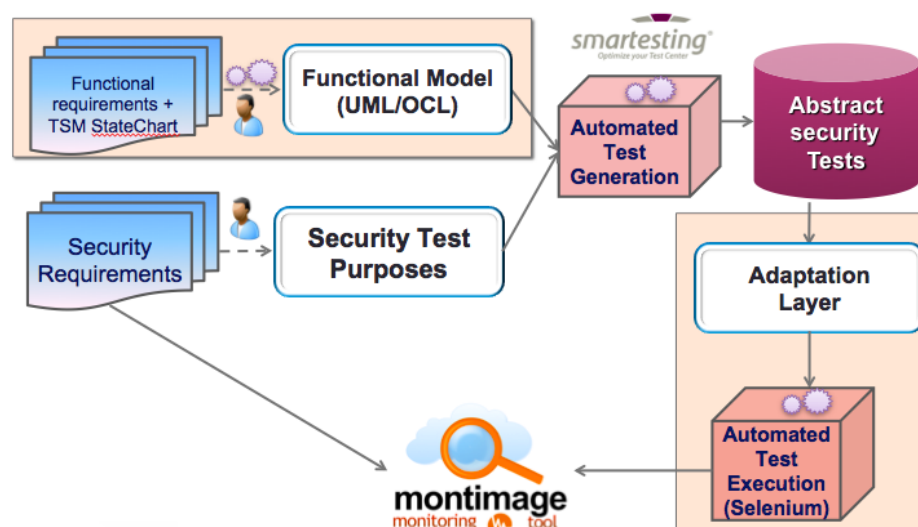



Figure 7: Smartesting process applied on the Gemalto TSM case study

2.2.3 Advances during DIAMONDS

In this section, the key aspects of the Smartesting prototype improvements developed within DIAMONDS for Security Oriented Test Generation are presented. For illustration purpose, we take examples from both

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 19 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

DIAMONDS case studies (Thales HDRM Waveform radio protocols and Gemalto TSM). These elements complete the first presentation done in the D3.WP3 and D4.WP3 deliverables and focus on the improvements done in regards of the first version of the prototype.

We will first present the “keyword editor” added to the Test Purpose editor for usage simplification and reuse purpose, then we will see an addition to the language to increase its expressiveness, and finally a more Gemalto TSM use case specific plugin to generate UML/OCL model parts from specific SUT configuration files.

2.2.3.1 Test Purpose keywords editor

The Test Purpose expresses a « pattern » (incomplete scenario) that the tests will have to cover, in respect to the functional model. It allows using the test generation model artefacts to express chains of specific states of the System Under Test (SUT) and specific behaviours of the SUT.

Those model artefacts, such as enumeration literals, operations, annotations, and even OCL parts of code, are representing different behaviours or states of the modelled SUT. These concepts, during the test purpose definition, can make it difficult for the security tester to express the tests objectives he wants to cover.

For instance, on the Thales HCDR waveform case study, concerning ad-hoc radio protocols, we would like to express a test objective where different stations reach a specific configuration, and then we want to trigger 3 kinds of attacks. The corresponding test purpose would be:

```
for_each operation $ATT from attack1 or attack2 or attack3,
use move any_number_of_times to_reach "self.utilities.areNeighbours(ADDRESSES::ADDRESS_1,
ADDRESSES::ADDRESS_2) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_1, ADDRESSES::ADDRESS_3) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_2, ADDRESSES::ADDRESS_3) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_2, ADDRESSES::ADDRESS_4) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_3, ADDRESSES::ADDRESS_4) and
not(self.utilities.areNeighbours(ADDRESSES::ADDRESS_1, ADDRESSES::ADDRESS_4))" on_instance
waveForm
then use $ATT
```


This makes the test purpose less easy to read and understand. A keyword has been introduced to allow the following test purpose expression:

```
for_each operation $ATT from #attacks,
use move any_number_of_times to_reach #initialNodesConfiguration
then use $ATT
```

The keywords can either be defined immediately, or the mapping with the model can be done afterwards. Here for each keyword we would have the following definition:

attacks: list of operations = attack1 or attack2 or attack3

*initialNodesConfiguration: state on waveform = self.utilities.areNeighbours(ADDRESSES::ADDRESS_1, ADDRESSES::ADDRESS_2) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_1, ADDRESSES::ADDRESS_3) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_2, ADDRESSES::ADDRESS_3) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_2, ADDRESSES::ADDRESS_4) and
self.utilities.areNeighbours(ADDRESSES::ADDRESS_3, ADDRESSES::ADDRESS_4) and
not(self.utilities.areNeighbours(ADDRESSES::ADDRESS_1, ADDRESSES::ADDRESS_4))*

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 20 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

Keywords are defined for a test suite, so the *attacks* or *initialNodeConfiguration* can be used in several test purposes, and be defined one. This render test purpose creation and maintenance easier, as the modification of a keyword definition impacts all the test purposes using it.

The test purpose editor now also embeds the keyword editor, with completion upon model elements.

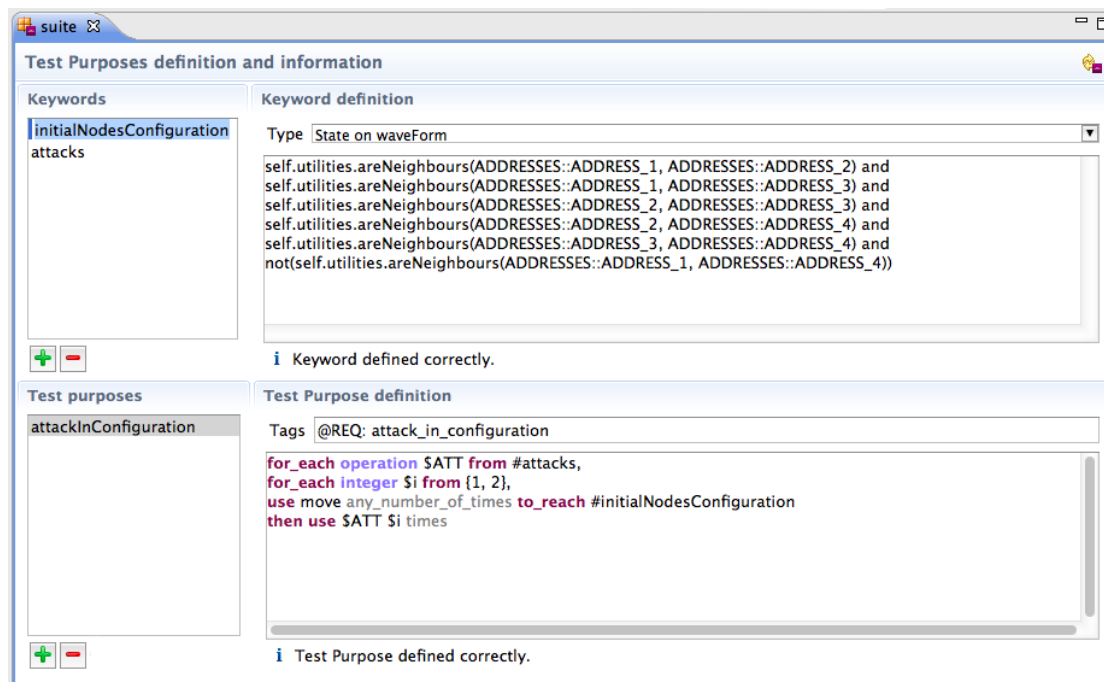


Figure 8: Smartesting Test Purpose and Keywords editors

2.2.3.2 Test Purpose Language improvements

The test purposes are test selection criteria that define the way to generate tests from the test generation model. The language is designed to be expressive enough to create tests from the test generation model.

For the need of several test purposes, the usage of an operation must be restricted in regards of the value of its return parameter. In the initial version of the test purpose language this was not possible. For instance, if an operation has a return parameter that express the success of the failure of that operation, we could express that we only authorize its usage in a success way. Here is the chosen way to express that constraint:


```
for_each operation $ATT from #attacks,
use move with_result SUCCESS any_number_of_times to_reach #initialNodesConfiguration
then use $ATT
```

2.2.3.3 Test Generation Engine improvements

In order to make the test generation more efficient, the generation engine has been modified. A specific algorithm has been developed especially for the need of the test generation from test purposes, as the previous one was dedicated to generate test sequences from behavioural objectives, and was just adapted.

2.2.3.4 Automatic model parts generation

In order to assist the modelling task, around the Gemalto TSM case study, a prototype has been developed. It consists in generating behavioural model parts from existing .xml configuration files for the SUT.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 21 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

During the TSM configuration stage, each .xml file (called StateChart) is imported to define the workflow that must be followed to perform an action on the TSM (service activation, eligibility check...). Each of those file, when imported, is interpreted as a state machine in the model with OCL annotations.

The Smartesting CertifyIt test generation engine is able to use them, in addition to test purposes altering the nominal workflows, to generate test sequences. The Figure 9: Smartesting test model import from TSM StateChart plugin situates this feature in the Smartesting process.

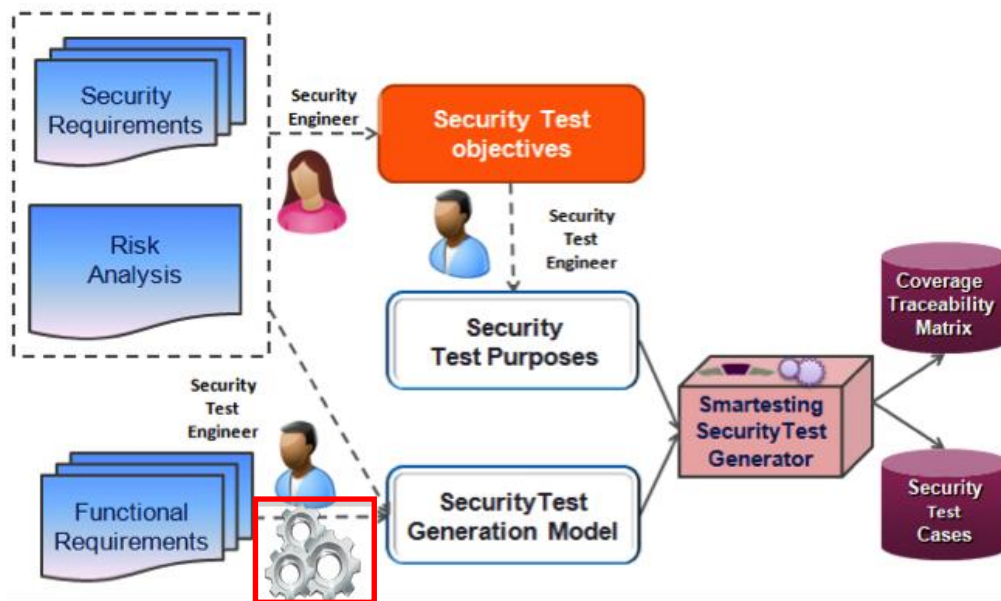


Figure 9: Smartesting test model import from TSM StateChart plugin

This feature has been included in the Smartesting CertifyIt plugin for IBM Rational Software Architect. A menu has been created, and a TSM StateChart can be imported as seen in Figure 10:

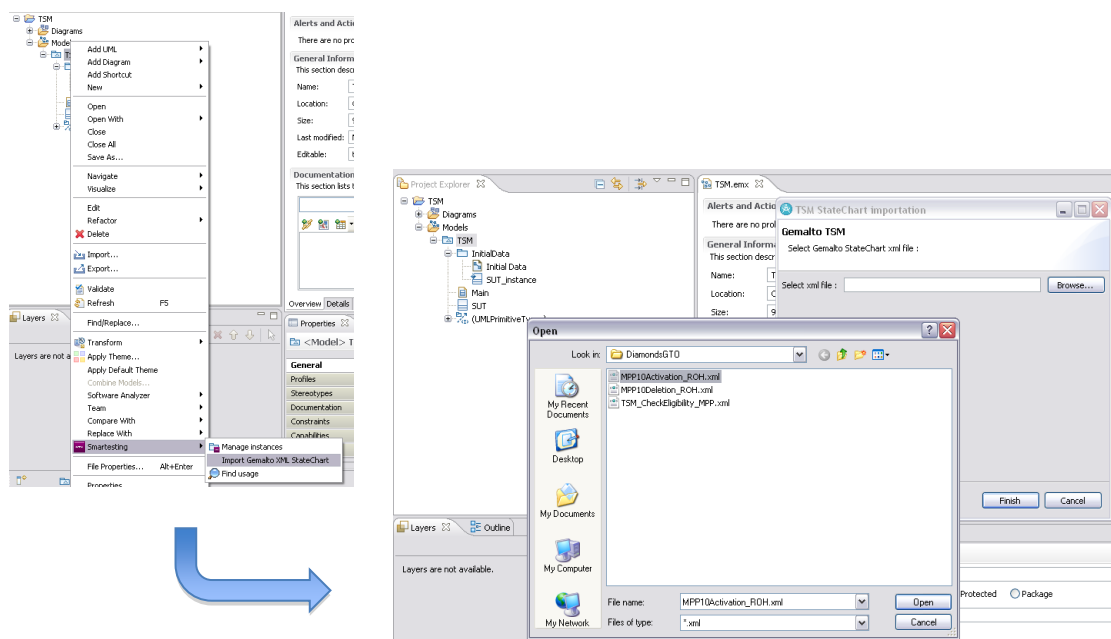


Figure 10: Smartesting test model import from TSM StateChart importer



Final Security Testing Tools

Deliverable ID: D5.WP3

Page : 22 of 80

Version: 1.0

Date : 22.05.2013

Status : Final

Confid : Public

The result of the import is the creation of a UML State Machine, with the transitions having triggers, guards and effects specified in OCL, respecting the imported TSM StateChart. Please refer to Figure 11 and Figure 12 for screenshots for model parts generated from TSM StateChart and test sequence from TSM StateChart imported model.

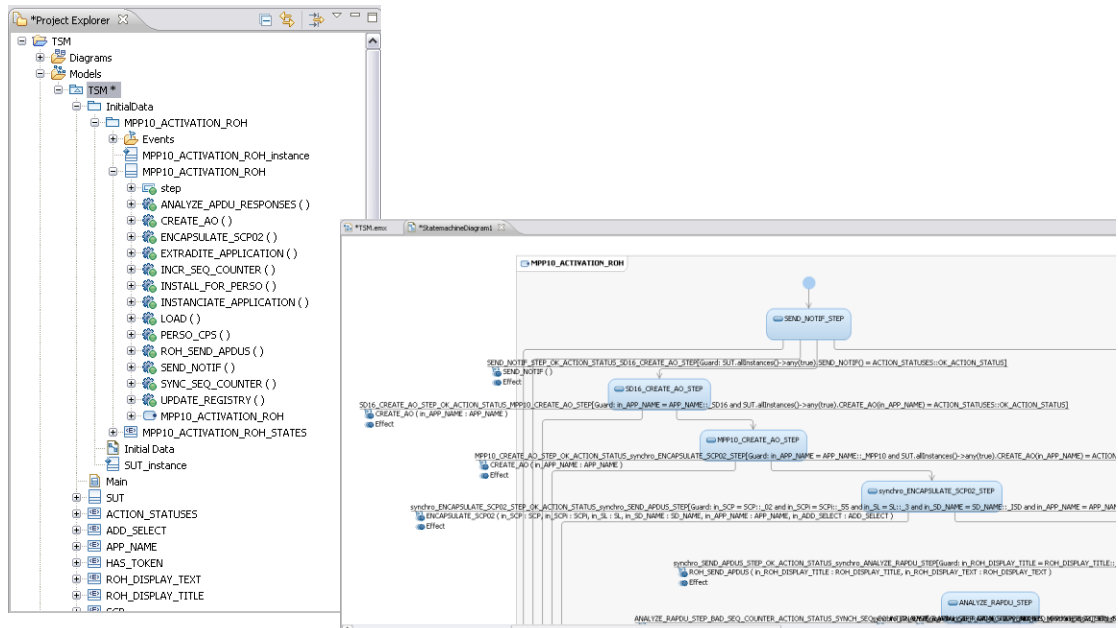


Figure 11: UML/OCL model parts generated from a TSM StateChart

Without any other manipulation, first test sequences can be generated from the model as followed.

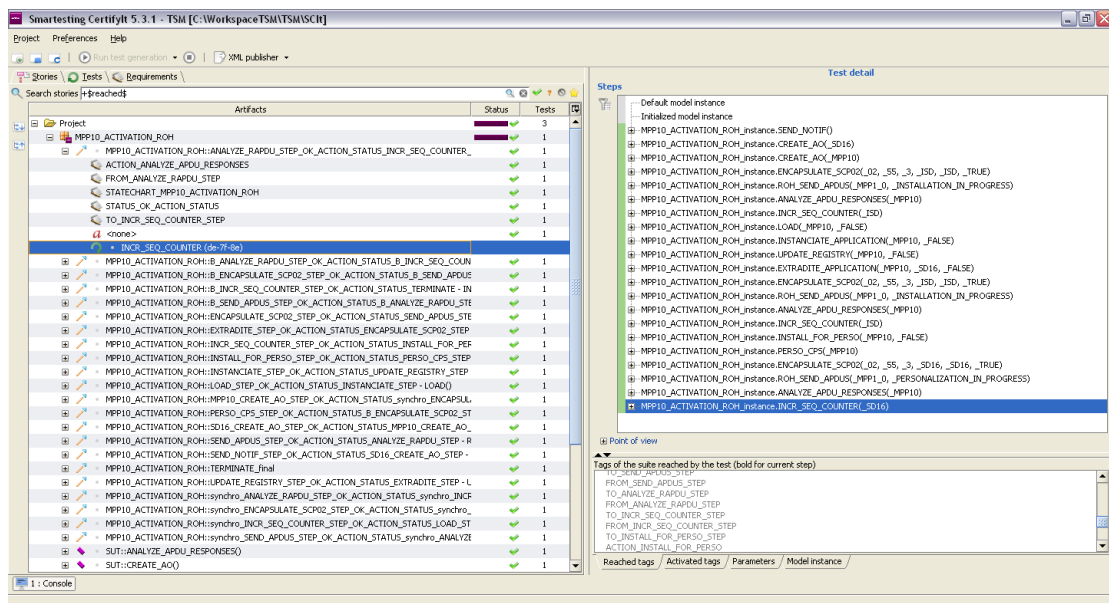




Figure 12: Generated test sequence from TSM StateChart imported model

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 23 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.2.3.5 Conclusion

We have seen DIAMONDS addition to the Smartesting Certifylt Security Oriented test generation prototype. The Test Purpose language has been used on different DIAMONDS use cases, which enabled to improve its expressiveness, like for instance output parameters specification.

The language manipulation and maintenance have been improved thanks to the keyword editor, and a first prototype has been created around the Gemalto TSM case study to assist the modelling activity. As this last point is still a work in progress, the other features described in this document are now part of a beta feature in the Smartesting Certifylt solution.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 24 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

2.3 FRAMEWORK FOR ACTIVE SECURITY TESTING (FSCOM)

2.3.1 Description of the Tool

FSCOM, member of ETSI and technology partner of Testing Technologies, is focused on active testing targeting communication protocols security and conformance. FSCOM provides solutions based on TTCN-3 [19] standardized and promoted by ETSI. In the second part of this project, FSCOM has developed a prototype for active security testing based on the Gemalto case study [16]. Figure 13: Gemalto TSM interfaces describes the different interfaces provided by the TSM for functionalities such as TSM administration, customer administration etc. The FSCOM prototype is focused on the interface 2", named RoH Proxy.

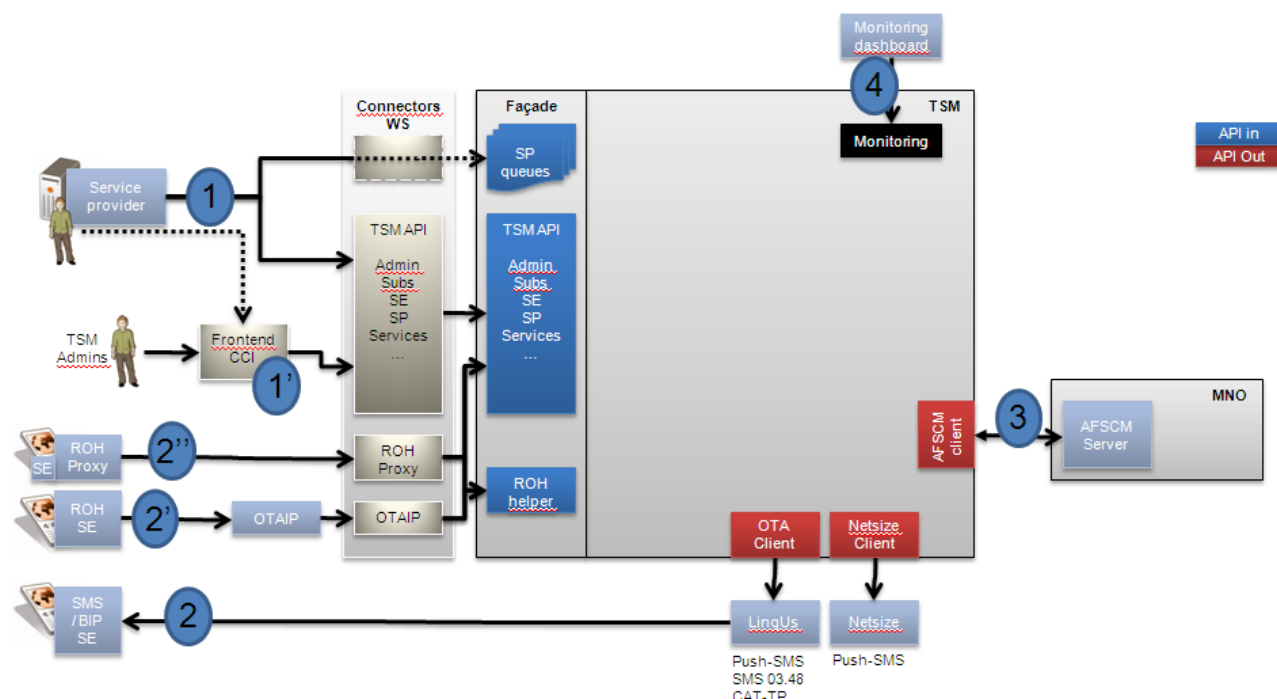


Figure 13: Gemalto TSM interfaces

Typical threat scenarios for the Gemalto use case occur when an attacker tries to use malicious ProxyAgent software hosted by an end user mobile or smart phone. Based on an extended HTTP protocol (Remote Application Management over HTTP [17]), the prototype validates vulnerabilities such as injection of altered protocol messages in the network traffic (network injection), brute-force attacks or denial/destruction threats attacking the interface 2" of the TSM framework [16].


The present framework is close to that used for Thales case study [18] and provides a 'black box' approach with functional test case design based on an analysis of the specification of the interface 2" of the TSM framework [16] and the extended HTTP protocol (Remote Application Management over HTTP [17].)

The FSCOM approach is focused on active attacks applying the strength of the TTCN-3 related test methodologies to new fields, i.e. testing web services security.

2.3.1.1 Introduction to the Security testing framework (FSCOM)

The security testing framework is made up by a set of tools that provide:

- Identification of the candidate implementations under test (IUT) for security testing;

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 25 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

- Definition of the applicable tests, i.e. answering the question "what and how to be tested";
- Development of the resulting test specifications.

For the Gemalto use case the following security testing related topics will be covered in the subsequent clauses:

- Identification of candidate EUTs/IUTs.
- Identification of test scenarios.
- Definition of test bed architecture.
- Identification of test bed interfaces.

2.3.1.2 Security testing

The following clauses provide security testing methodologies on which FSCOM based the active security test framework.

2.3.1.2.1 Candidate EUTs/IUTs

For security testing, both "Implementation Under Test" (IUT) and "Equipment Under Test" (EUT) are considered. An EUT is a physical implementation of one or more network layers (IUT), which interact with one or several other EUTs via one or more reference points (RPs).

2.3.1.2.2 Test scenarios

In security, a large number of use cases are already identified. In a specific implementation of an IUT, very likely only a sub-set of these use cases is supported. In order to perform the tests, IUTs supporting the same use cases are required.

2.3.1.2.3 Test bed architecture

The "System Under Test" (SUT) contains (refer to [19], [20] and [21]):

- The "Implementation Under Test" (IUT), this is the Gemalto TSM, focused on interface 2".
- The "Upper tester application" enables TSM actions such as 'eligibility check' or 'workflow activation'.
- The "Lower tester" enables to establish a proper connection to the system under test (SUT) over an Ethernet physical link.
- The "Upper tester transport" enables the test system to communicate with the upper tester application. Then the upper tester can be controlled by a TTCN-3 test component as part of the test process.

The "Security test system" contains (refer to [19], [20] and [21]):

- The "TTCN-3 test components" are processes providing the test behaviour. The test behaviour may be provided as one single process or may require several independent processes.
- The "Codec" is a functional part of the test system to encode and decode messages between the TTCN-3 internal data representation and the format required by the related base protocol standard.
- The "Test Control" enables the management of the TTCN-3 test execution (parameter input, logs, test selection, etc.).
- The "Test adapter" (TA) realizes the interface between the TTCN-3 ports using TTCN-3 messages, and the physical interfaces provided by the IUT.

Figure 14 describes the functional architecture of the security test bed:

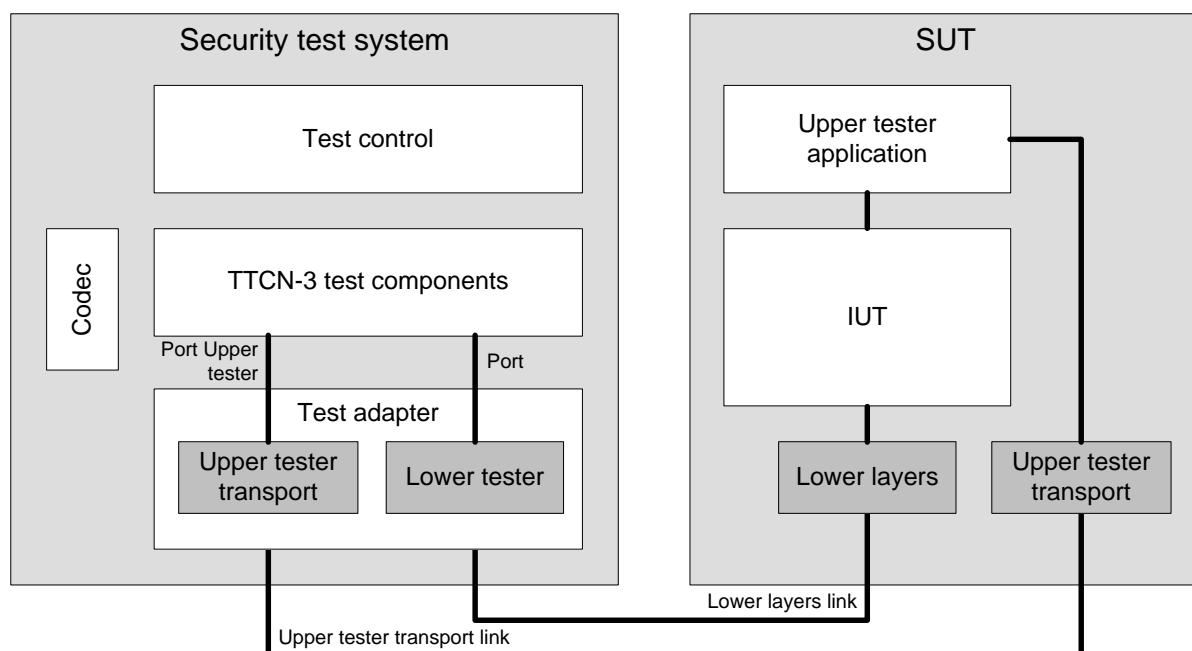


Figure 14: Security test system architecture

2.3.1.3 Identification of abstract test method

An abstract protocol tester presented in Figure 15 below is a process providing the test behaviour for testing an IUT. Consequently it will emulate a peer IUT of the same layer/the same entity. This type of test architecture provides a situation of communication which is equivalent to real operation between real devices. The security test system will simulate valid and invalid protocol behaviour, and will analyse the reaction of the IUT. Then the test verdict, e.g. pass or fail, will depend on the result of this analysis. Thus this type of test architecture enables to focus the test objective on the IUT behaviour only.

In order to access an IUT, the corresponding abstract protocol tester needs to use lower layers to establish a proper connection to the system under test (SUT) over a physical link (Lower layers link).

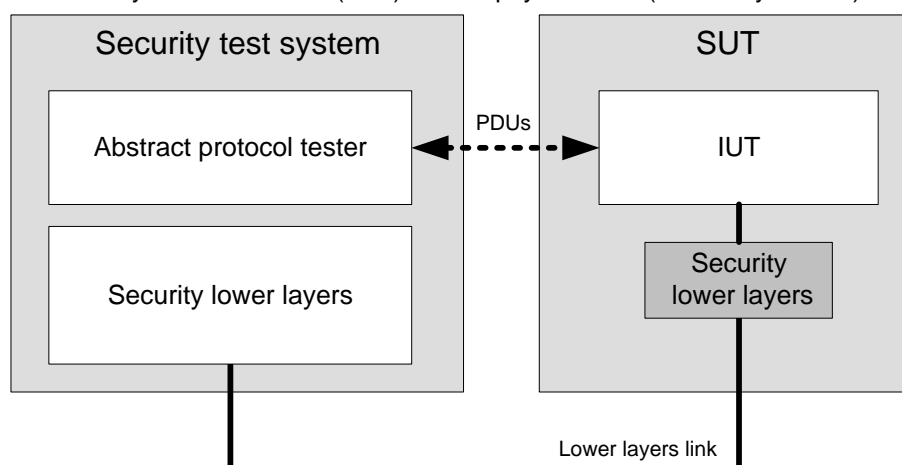



Figure 15: Generic abstract protocol tester

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 27 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

The "Protocol Data Units" (PDUs) are the messages exchanged between the IUT and the abstract protocol tester as specified in the base standard of the IUT. These PDUs are used to trigger the IUT and to analyse the reaction from the IUT on a trigger. Comparison of the result of the analysis with the requirements specified in the base standard allows assigning the test verdict.

2.3.1.4 Protocol description

Remote Application Management over HTTP protocol is used by the TSM for remote communication between the service provider and the End User Secure Element such as a USIM card [17].

In addition of tags already defined by these standards, Gemalto TSM uses an extended RAM over HTTP provided additional tags such as display information or error messages.

2.3.2 Application to Case Studies

2.3.2.1 Thales case study (final results)

In this second run, we integrated MNT tool to retrieve network traffic generated by the radio stack and we implemented active testing concept.

2.3.2.1.1 Active testing

Active testing is used to trigger different kinds of attack on the radio stack.

Three triggering methods were proposed:

- Attack is triggered when a random delay after starting the test case is reached;
- Attack is triggered when a specified timeslot is used;
- Attack is triggered when a specific delay is elapsed.

2.3.2.1.2 Integration with MNT

In parallel with MNT traffic network analysis the MNT tool forwards network traffic messages to our framework.

2.3.2.1.3 Results

The screenshot below in Figure 16 shows an execution of the testing framework applied to the Thales case study:

1. This framework is based on:
 - a. Test cases developed in TTCN-3 language which:
 - i. Trigger attacks on the IUT (bottom-right DOS windows),
 - ii. Check the IUT behaviour according to the rules as defined by MNT,
 - b. A TTCN-3 tool which executes these test cases (background window),
 - c. The MNT tools (top-right DOS window) which:
 - i. Collect the network traffic,
 - ii. Check security rules as defined by MNT
 - iii. Forward the logs to the TTCN-3 tool
 - d. The Thales radio stack, the IUT (top-left DOS window)

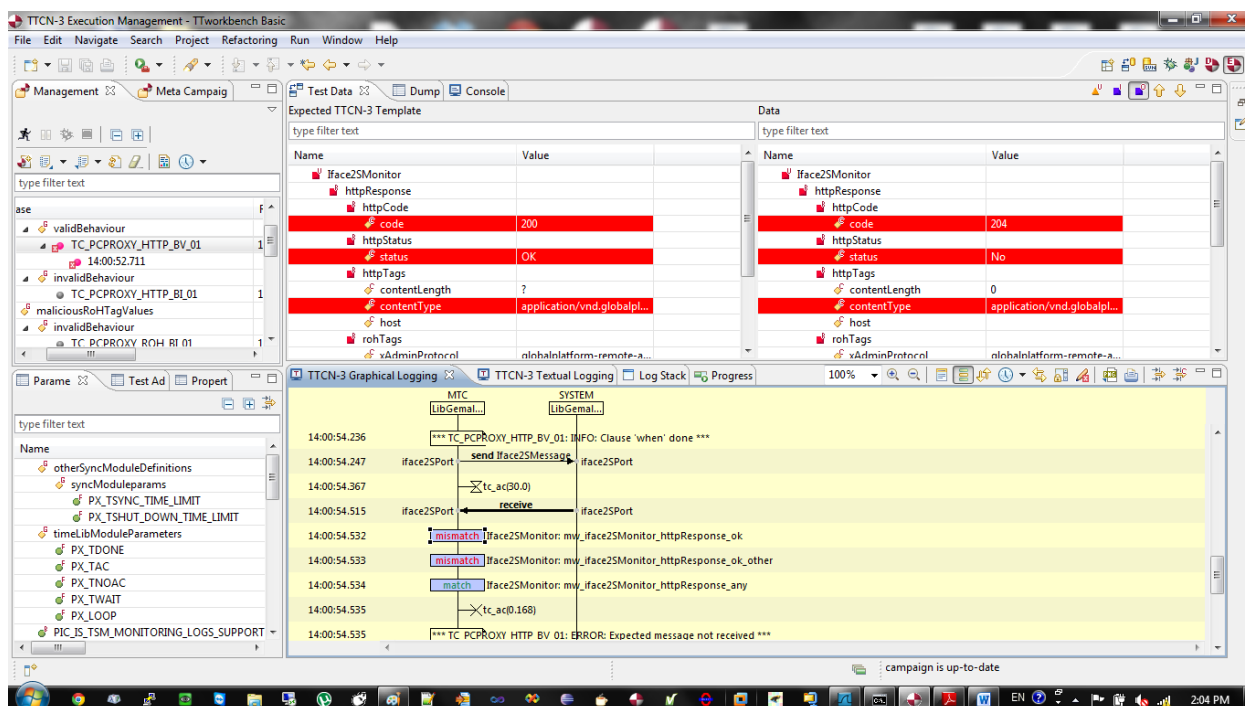


Figure 17: Malicious attack which set the HTTP tag Content-Length to 65535

2.3.3 Advances during DIAMONDS

2.3.3.1 Tools to convert Montimage security rules into ETSI TPLan language

2.3.3.1.1 Synoptic

ETSI has introduced TPLan in 2009 [31]. TPLan is defined with a minimal set of test-oriented keywords but owns the capability that permits users to define extensions to the notation. The benefits of using TPLan are:


- Consistency in test purpose descriptions - less room for misinterpretation;
- Clear identification of the TP pre-conditions, test body, and verdict criteria;
- Automatic syntax checking and syntax highlighting in text editors;
- A basis for a TP transfers format and representation in tools.

FSCOM developed a tool, as part of our security test framework to translate security rules introduced by MNT into TPLan [31]. This tool is based on XSL technology to translate XML tags describing a security property as introduced by MNT into a TPLan description.

This tool is truly innovative because a TPLan description could be used for producing test purpose description as introduced by ETSI methodologies and also for producing the test cases codes.

2.3.3.1.2 Results

The pseudo-code [Code 2] shows the result of the translation of a MNT security rules into TPLan. In ETSI methodologies, the TPLan pseudo-code is very important to produce test purposes documents and to develop test cases.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 30 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

```

<property value="THEN" delay_min="-1" property_id="1" type_property="SECURITY_RULE"
  description="DataUMAC within SPHY_DATA_REQ message with a slot type equal to SCH must be
the same transmitted within the previous UMAC_GA_DATA_REQ message">
  <event value="COMPUTE" event_id="1"
    description="SPHY_DATA_REQ message with a slot type equal to SCH"
    boolean_expression="((BASE.PROTO == 1)&&&((MSG_SPHY_DATA_REQ.SLOT_TYPE ==
0)&&&(BASE.TIME_SLOT
BASE.TIME_SLOT.2))&&&(MSG_UMAC_GA_DATA_REQ.RLC_Q_DATA.2
MSG_SPHY_DATA_REQ.SDATA_UMAC))"/>

  <event value="COMPUTE" event_id="2"
    description="UMAC_GA_DATA_REQ message with the same transmitted Data"
    boolean_expression="(BASE.PROTO == 801)"/>
</property>

```


Code 1: MNT security rule #1

```

/**
 * @desc (property_id #1) DataUMAC within SPHY_DATA_REQ message with a slot type
equal to SCH must be the same transmitted within the previous UMAC_GA_DATA_REQ message
 * Pics Selection: none
 * Config Id: CF02
 * <pre>
 * Initial conditions:
 * with {
 *   Not applicable
 * }
 * Expected behaviour:
 * ensure that {
 *   when {
 *     SPHY_DATA_REQ message with a slot type equal to SCH, and
 *     UMAC_GA_DATA_REQ message with the same transmitted Data
 *   }
 *   then {
 *     ((BASE.PROTO == 1) and ((MSG_SPHY_DATA_REQ.SLOT_TYPE == 0) and
(BASE.TIME_SLOT == BASE.TIME_SLOT.2)) and (MSG_UMAC_GA_DATA_REQ.RLC_Q_DATA.2 ==
MSG_SPHY_DATA_REQ.SDATA_UMAC)), and
 *     (BASE.PROTO == 801)
 *   }
 * }
 * </pre>
 *
 * @version 0.0.1
 */

```

Code 2: TPLan pseudo-code generated from MNT security rule #1

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 31 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.4 STATIC AND DYNAMIC APPLICATION ANALYSIS FOR VULNERABILITY DETECTION

2.4.1 Description of the Tool

Grenoble INP has been working in two directions for vulnerability detection, namely grey-box based static security analysis of executables for vulnerability detection and black-box based model inference assisted evolutionary fuzzing for vulnerability detection. Both of the approaches are explained in WP2 [1]. In the following sections, we describe the prototype implementation of two tools based on the approaches.

2.4.1.1 Grey-box based Static Security Analysis of Executables for Vulnerability Detection

Based on the technique described in [1], a prototype named Light-weight Static Taint Tracer (LiSTT) is implemented. Figure 18 provides a high level view of the tool.

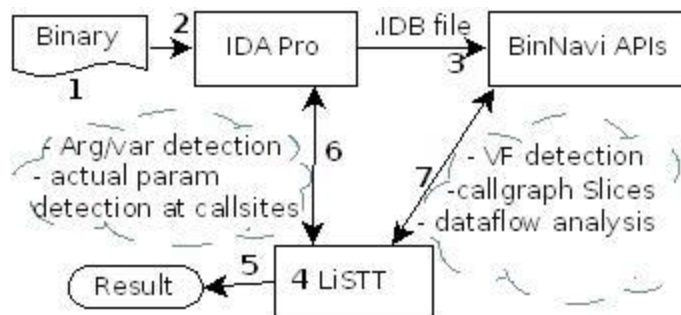


Figure 18: LiSTT architecture

First, IDA Pro (2) takes as input a binary file (1) and produces an .idb file which is loaded into the BinNavi framework (3). LiSTT (4) interacts both with BinNavi API and IDAPython API (6) to perform intra- and inter-procedural dataflow analysis (7). The produced result (5) is the set of vulnerable paths that have been detected with respect to an input source IS (provided as input).

In order to analyse a binary, there are three major steps as described below:

2.4.1.2 Getting the Disassembly of the Binary Executable

In order to analyse a binary, it is first disassembled into the corresponding assembly code. LiSTT handles x86 code and therefore, the binary is converted to x86 assembly code by using IDA Pro [3]. As LiSTT performs dataflow analysis on the binary, we need to know all the useful memory locations within the binary. The useful memory locations are arguments and local variables for functions. LiSTT has python scripts to extract this information. After IDA Pro finishes analysing the binary, we execute a python script within IDA Pro to extract above mentioned information. This script saves the extracted information in a file (.inst file), which is supplied as an input (see Figure 19). The saved results have the following information. For each function in the binary, we extract:

- Arguments and local variables for the function
- All the functions called by the function
- All the **PUSHed** parameters for each of the called functions.

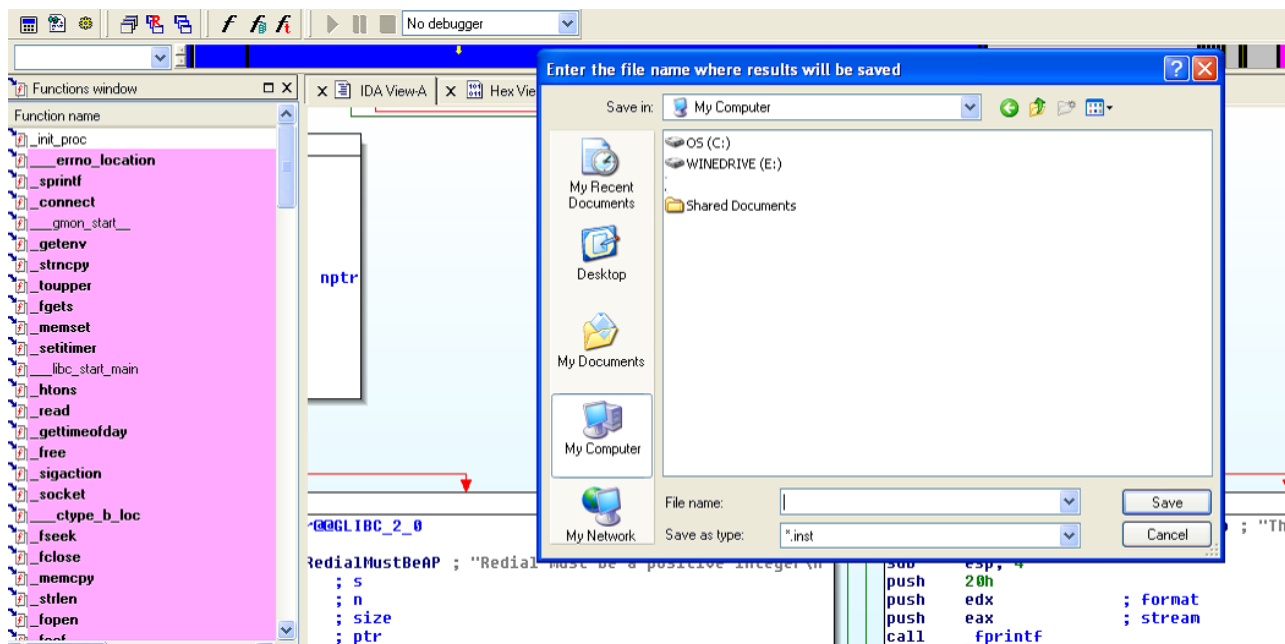


Figure 19: Generating Arg/Var information from IDA Pro

2.4.1.3 Transforming Native Assembly Code into REIL- A Higher Level Intermediate Language

IDA Pro creates an .idb file that contains the result of analysis performed by IDA Pro e.g. control flow graph of the functions and call graph of the application. However, all such results are applicable to assembly code and assembly code is still very low-level to be used for other complex analysis such as dataflow. Therefore, the native assembly code is needed to be transformed into some other intermediate language with less complex syntax.

BinNavi [4] is a framework for analysing assembly code. In order to perform more complex dataflow analysis, it also provides an intermediate language called REIL (reverse engineering intermediate language) [5]. REIL consists of only 17 instructions with *three address code* (TAC) format. Every native assembly instruction is converted to equivalent REIL instructions and the dataflow analysis is performed on this REIL code. In order to obtain the REIL code, we need to import the IDA Pro generated .idb file into BinNavi. BinNavi provides a GUI to do so (see Figure 20).

Once the file is imported into BinNavi, we can access various information about the binary by using APIs provided by BinNavi. These APIs are available via Jython programming language. As a result, all of the LISTT is written in Python/Jython.

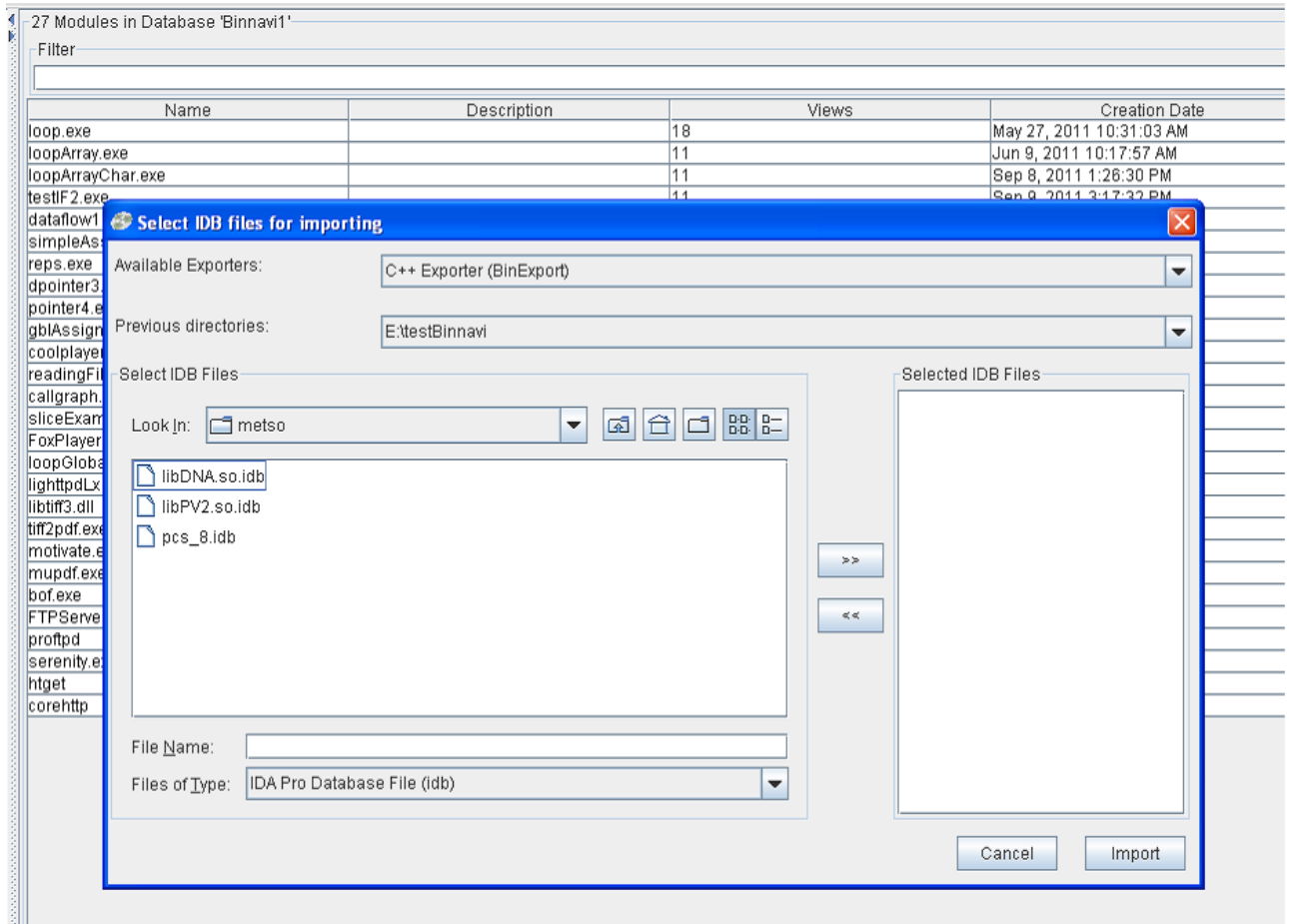


Figure 20: Importing .IDB file into BinNavi

2.4.1.4 LiSTT Overview and Usage


The main module LiSTT (component 4 in Figure 18) consists of several python files. The main file that computes the taint flow is "taint-analysis.py". The computation begins with computing the call graph slice for a given pair (Tsrc, Tdst) of taint source and destination functions. The following class represents a slice (Figure 21).

```

class CSlice
    __init__(self, SliceSrc, SliceDst, allEdges, sNode, dNode, commonRoot)
    getTSrcNode(self)
    getTDstNode(self)
    getDOTStr(self)
    getCommonRoot(self)
    getChildren(self, node)
    getParents(self, node)
  
```

Figure 21: Slice Class in LiSTT

Once all the class slices are calculated, LiSTT starts dataflow calculation using abstract interpretation based framework. There are three different dataflow analysis takes place within LiSTT. A typical diagram of various components of such analysis is illustrated in Figure 22.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 34 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

SkeletonLattice(Lattice) combine(self,states)	SkeletonTransformationProvider(TransformationProvider) init__(self,gVarUsed=None) transform(self,node,currentState,influencingState)	SkeletonLatticeElement(LatticeElement) __init__(self) equals(self,rhs) lessThan(self,rhs)	Summary(object) __init__(self,name) get_arg_dep(self,num) get_mem_dep(self,num) get_ret(self) get_name(self)
---	---	---	--

Figure 22: Various classes that represent dataflow analysis in LiSTT

The **Summary** class is used to represent the “*function summary*” that is used while performing **interprocedural** dataflow analysis.

There is a configuration file called “configTaint.py” wherein we can set various parameters for performing a particular analysis. The most important ones are:

```
# set the logging level
1. loglevel = DEBUG, INFO, WARNING, ERROR, CRITICAL
# function name that is used for reading tainted data
2. taintFuncN = 'Tsrc'
# which argument of taint function introduces taint
3. taintArgN = 1
# the taint may be introduced by the return value of the taint src function
4. taintIsReturn = False
# this value will be set to true for returned taint
```

The LiSTT also has a module to calculate buffer overflow prone (BOP) functions [2]. This module is launched from command-line by typing the following command:

```
C:\> jython BOPFunctionRecognition.py
```

The output of the above command is a list of functions that are prone to buffer overflow. The list is saved as “**pickle**” object with .pkl extension. This file is used as one of the parameters in main taint flow computation.

The LiSTT's taint flow computation is launched from command-line with the following parameters:

1. name of the .inst file (as described in section 2.X.1.1),
2. name of the pickle file containing BOP functions,
3. name of the folder where slices are saved (e.g. **result**).

The main command to launch the LiSTT taint flow analysis is:

```
C:\> jython taint-analysis.py
```

The final results are saved in the **result** folder.

A typical example of the call graph based slice computed by LiSTT is shown in Figure 23.

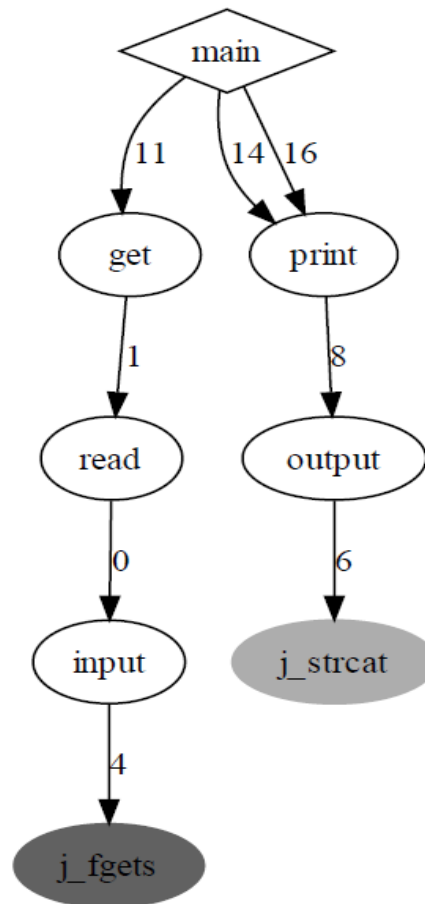


Figure 23: A taint flow slice as computed by LiSTT. TScr= j_fgets; TDst= j_strcat

The LiSTT also provides the fine-grained information about “How and where the taint flow occurred. For the above example, LiSTT provides the following information:

```
Taint Src: j_fgets -> common root: sub_401164 -> Taint sink:
j_strcat
```

```
At 0x4011fe03L by:
```

```
*V_1
```

```
V_4
```

```
*V_2
```

```
eax
```

```
At 0x4011d603L by:
```

```
V_4
```

```
*V_2
```

```
eax
```

2.4.1.5 Black-Box Based Model Inference Assisted Evolutionary Fuzzing For Vulnerability Detection

The approach described in WP2 [1], is implemented in the form of a tool named *KameleonFuzz* (KF). KameleonFuzz automatically detects Type-1 and Type-2 Cross Site Scripting (XSS) in Web Application. Figure 24 illustrates the main architecture of KF.

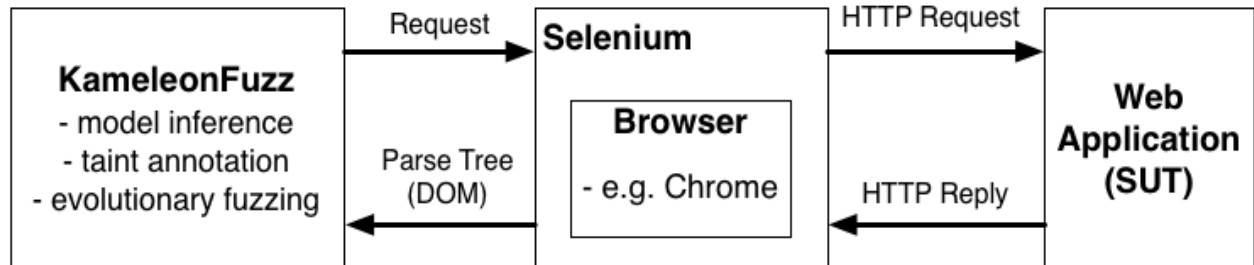


Figure 24: KameleonFuzz Tool Architecture

KF is implemented as collection of many python modules for implementing various functionalities as described in [1]. At a high level, KF send request to SUT through browser. In order to parse the HTTP response, it makes use of selenium binding for Python [6]. In this way, KF can generate the DOM tree of the HTTP response (output from the SUT). After getting DOM tree, KF invokes the modules to compute other information like, string matching, URI and forms recognition (for model inference), taint flow etc. It is this DOM parse tree that is used for fuzzing various parameters by traversing its nodes. In the following sections, we provide details on these major components of KF.

2.4.1.5.1 High Level View of KF's Main Components

As show in Figure 25, KF first infers a model representing the observable control flow of the application. This model is then annotated for approximate observable data flows (potential reflections). Finally, the ability of attackers to exert control on such reflections is explored via Evolutionary Fuzzing.

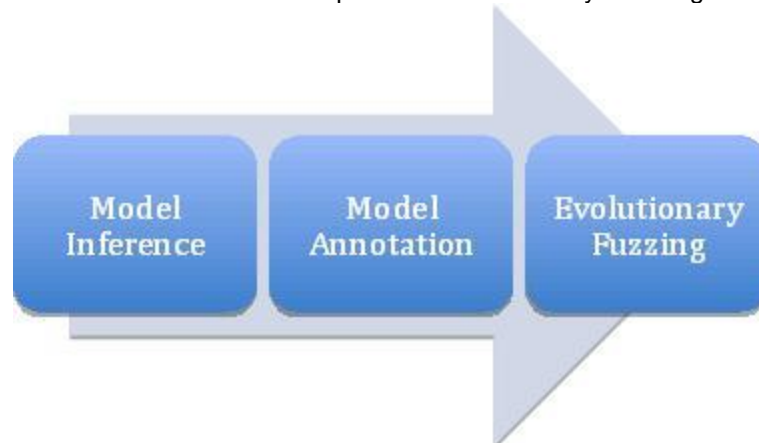


Figure 25: KameleonFuzz: High Level Approach Flow


Configuration

Two files control the configuration of KF:

- **main config file:** ./config/webgoat/config.xml
 - SUT interfaces
 - Authentication credentials
 - Reset method
 - Limits for Inference, Annotation and Fuzzing
 - Parser to use (Browser in the XSS case)
- **attack grammar file:** ./KameleonFuzz/Grammar/HTTP/XSS/attack_grammar.kfgrammar

Reset Script

It is specific to each system. For WebGoat, killing the process and restarting it is sufficient. For Gruyere, it is also necessary to restore the database in its initial state.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 37 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

Running KameleonFuzz

For each step, the command to use is the same:

```
$ python3.2 ./KameleonFuzz.py --config webgoat
```

2.4.1.5.2 Model Inference

In order to learn the model of the SUT, KF implements model inference step in the form of a **state-aware crawler**. As aforementioned, after getting the DOM tree, KF extracts the URI/Forms from this tree to find other reachable pages. In this way, KF generates a FSM with input/output as observed while accessing each newly discovered page.

The tester has to identify which parameters are *Nonces* (i.e., they have two different values for the same input sequence submitted to the application). This is the case for cookies, anti-CSRF token, view-states... The names of such parameters have to be entered in the config.xml file.

At the end of this step, KF saves the inferred model in a pickle file, in order to reuse it later, and in an SVG image (see Figure 26), for a human tester to analyse it. In fact, we state that this output is of great help for a human penetration tester.

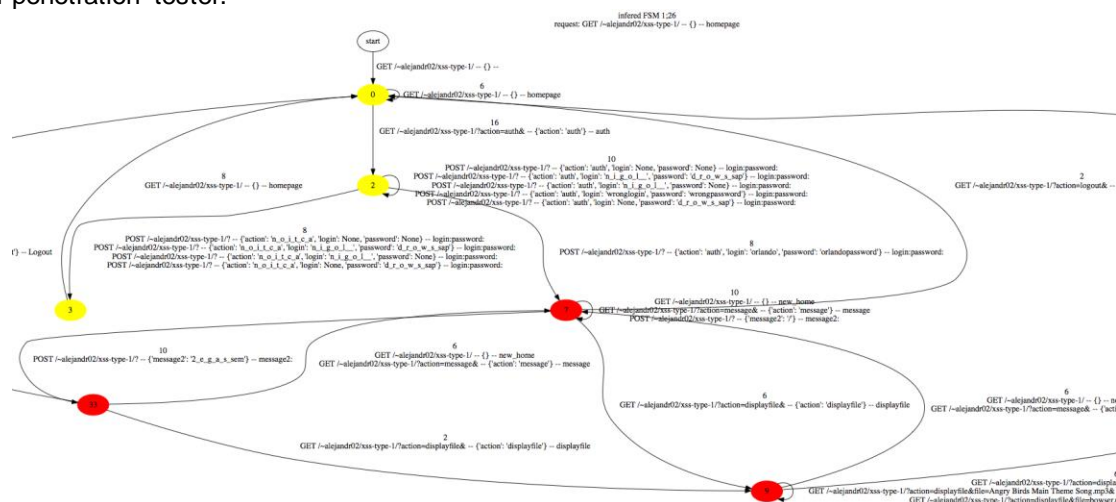


Figure 26: Extract of the Inferred Model for P0wnMe

2.4.1.5.3 Approximate Data Flow Annotation

The inferred model is fed to the approximate data flow annotation module. This module will infer the taint between previously sent input parameters value and the current output.

Since many pattern matching computations are performed, this process has to be efficient. Thus we make use of a naïve substring matching algorithm, because we assume that input values will not significantly be transformed. The tester has to indicate what is the minimal length of such substrings to be considered. In many of our experiments, we arbitrarily choose 6 characters.

The annotated model is also saved to a pickle file, to avoid recomputing this when willing to perform several fuzzing sessions afterwards. Each reflection is also saved to a separate image file, such as the one displayed in Figure 27.

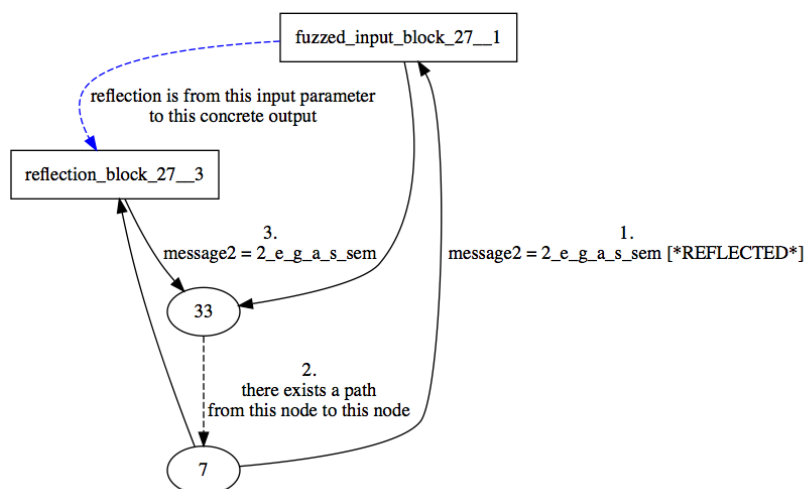


Figure 27: Example of Reflection Annotation (this is an extract of the p0wn model, with only transitions related to the reflection)

2.4.1.5.4 Evolutionary Fuzzing

Genetic Algorithm

The Evolutionary Fuzzing step is driven by a Genetic Algorithm (see Figure 28).


In the config.xml configuration file, various GA parameters can be controlled:

- Population size
- mutation rate
- elitism
- stopping conditions: number of generations, number of found faults, fitness to obtain...

Attack Grammar

Malicious inputs sent during the Fuzzing step are built using the attack grammar. It is vulnerability name specific (that is necessary to write another attack grammar to target SQL injection instead of XSS).

In Figure 29, we list an extract of the attack grammar for XSS Fuzzing.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 39 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

```

1                                > Creating the first generation: n individuals
2 for l ∈ [1..n] do
3   Population[l] ← generateIndividual(Model,AttackGrammar)
4 end for
5                                > Evolving the population
6 repeat
7   for all I in Population do
8     SUT Reset
9     Submit I to SUT
10                                > precise taint data flow inference
11     Compute FITNESS(I,Model,0) + VERDICT(I, 0, TreePatterns)
12   end for
13   CROSSOVER: f fittest individuals according to attack grammar to
    produce m Children
14   for all C in Children do
15     MUTATE(C,AttackGrammar)
16   end for
17   Population ← (n - m) fittest parents + m Children
18 until stopCondition

```

Figure 28: Genetic Algorithm

```

-----
xss_in_structure = (xss_outside_tag | xss_in_attribute_value |
xss_in_css)
xss_outside_tag = (js_script | img_onerror)
img_onerror = "<img src=x onerror=" [0:1](js_quote) js_payload
[0:1](js_quote) " >"
js_quote = ("\" | "'")
js_payload = js_pay_4
js_pay_4 = "alert(" [3:6](text_simple_number) ")"

```

Figure 29: Extract of the Attack Grammar

XSS Exploits


When fuzzing stops, KF outputs to its log file one exploit for each distinct found XSS. Below we illustrate an extract of such an output for the WebGoat application.

```

-----
- source: {'node_to__id': 14, 'node_from__id': 28,
'fuzzed_param': 'startDate'}
- dest: {'node_to__id': 14, 'node_from__id': 28, 'fuzzed_param':
'startDate'}
- param_name: startDate
- individual fuzzed value: <img src=x onerror=alert(663) >

```

Figure 30: extract of the found XSS exploit summary

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 40 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

2.4.2 Application to Case Studies

2.4.2.1 Metso DNA Application Analysis

INP Grenoble analysed Metso NDA software binaries by using its LiSTT tool. There are 3 binaries provided by Metso, each have more than 1000 functions. LiSTT found buffer overflow prone functions and reported them to Metso. INP Grenoble also analysed the binaries to detect vulnerable paths that could lead to buffer overflow vulnerabilities. With the limited analysis information, LiSTT found no vulnerable paths in the Metso **main** binary of the application.

2.4.2.2 Gemalto TSM Analysis

INP Grenoble performed a black-box analysis (for the existence of cross site scripting XSS vulnerability) on the Gemalto TSM application by using evolutionary fuzzing approach. Due to the limited accessibility to the application, full code coverage was not achieved. Therefore, within the scope of analysed portion of the application, no XSS vulnerability was found. It was also noted that the application was written by taking security into the consideration.

2.4.3 Advances during DIAMONDS

The whole of the LiSTT tool was developed during DIAMONDS project. The underlying techniques were advanced and extended during the project. Similarly, the KameleonFuzz tool came to its full development during the project. The underlying techniques got matured during the project.

2.5 CODENOMICON DEFENSICS

2.5.1 Description of the Tool

Since 2001, Codenomicon DEFENSICS™ test platform has been applying a wide range of fuzzing techniques to provide preemptive security testing for network equipment manufacturers, operators, consumer electronics companies, enterprises and governmental organizations. The latest release of the test generation engine is Defensics 10, launched in November 2011.

Codemicon Defensics performs fuzzing, or fuzz testing. In fuzzing, an automated testing tool has to be first capable of creating valid message structures and message sequences. Using these behavioural models it then alters the behaviour to generate millions and millions of nearly-valid messages that systematically anomalise some parts of the information exchange to test the target system for robustness.

The Figure 31 shows the specification-based approach, where the model is built from protocol interface specifications. The model is pre-built by Codenomicon, but the user can edit both message sequences and the actual messages sent and received by the tool.

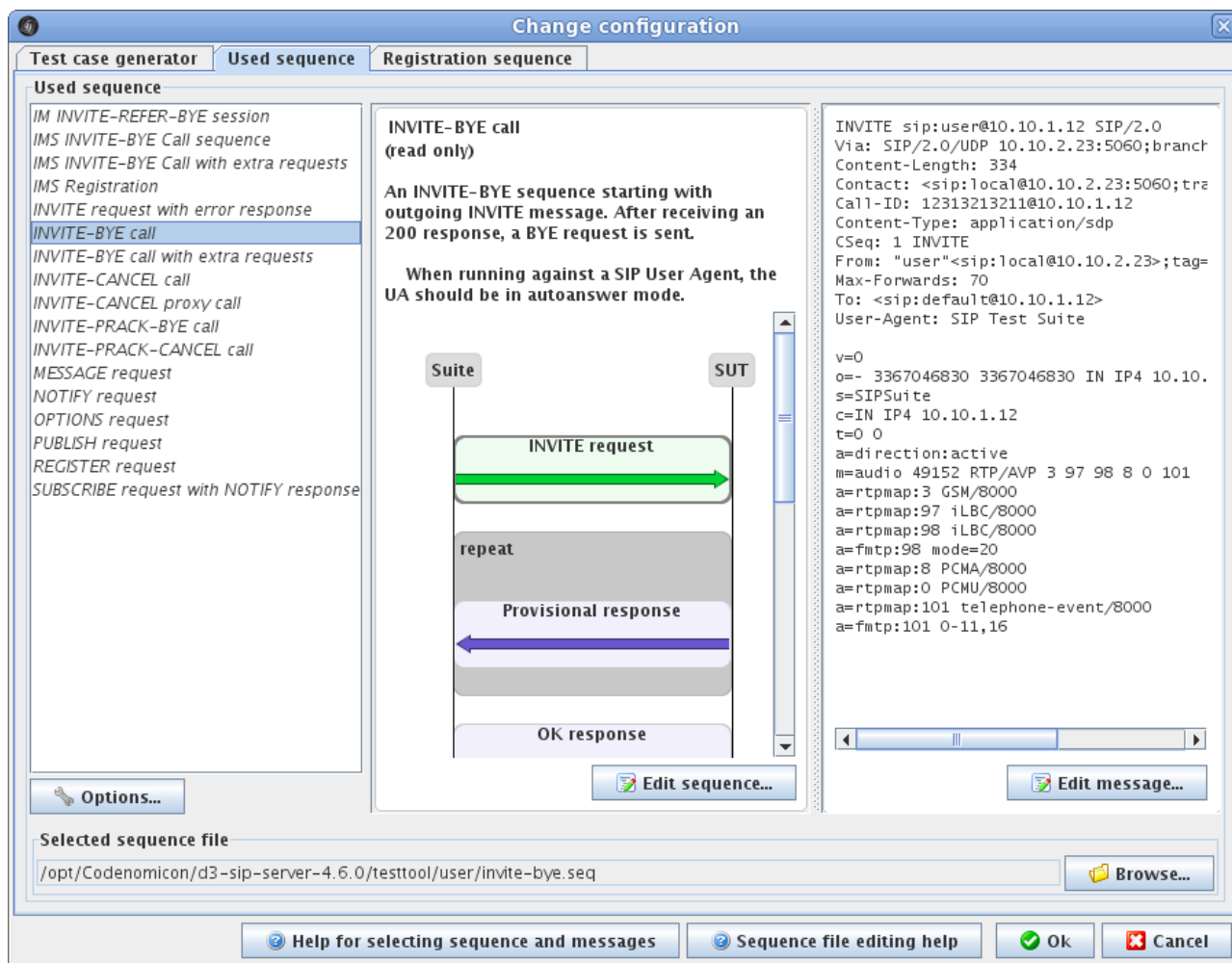


Figure 31. Specification-based approach

After model parameters are chosen, the test generator creates hundreds of thousands of test cases:

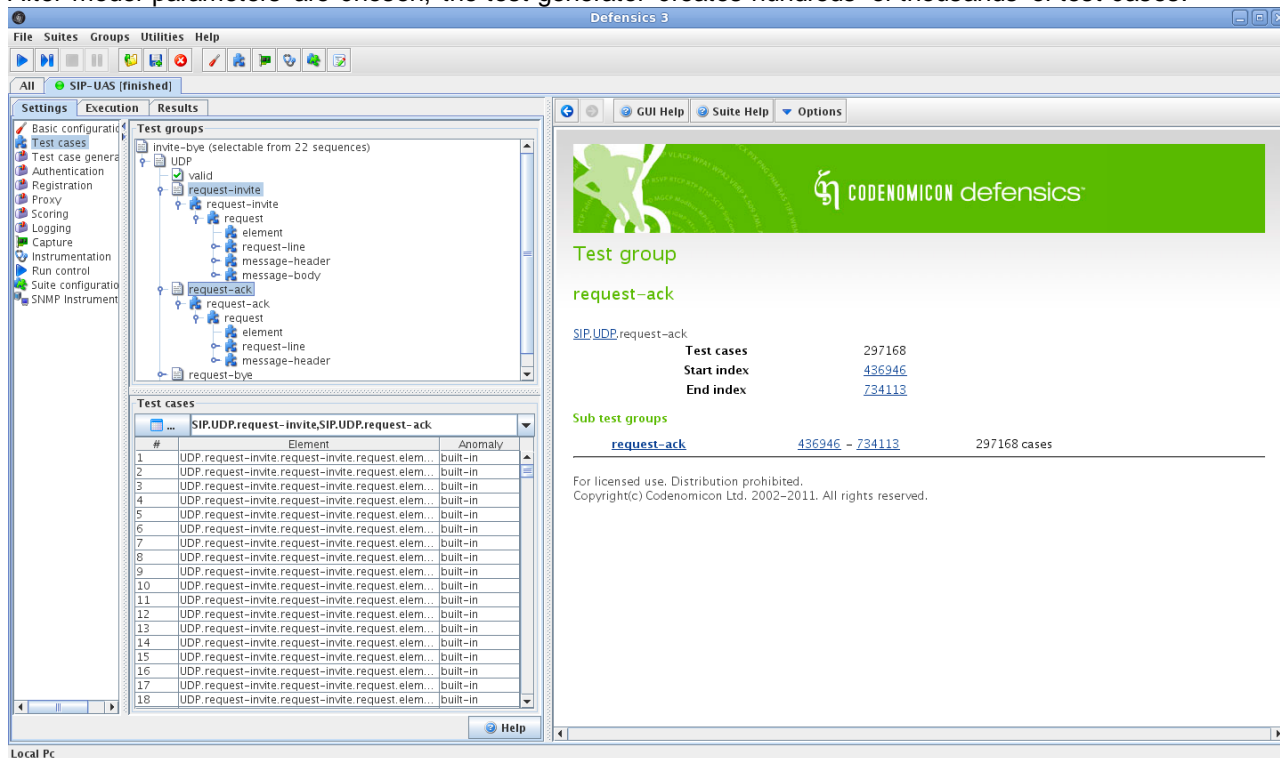


Figure 32. Test-case generation

In specification-based fuzzing, the tool user only needs to configure the required information about test target, as shown in the Figure 33:

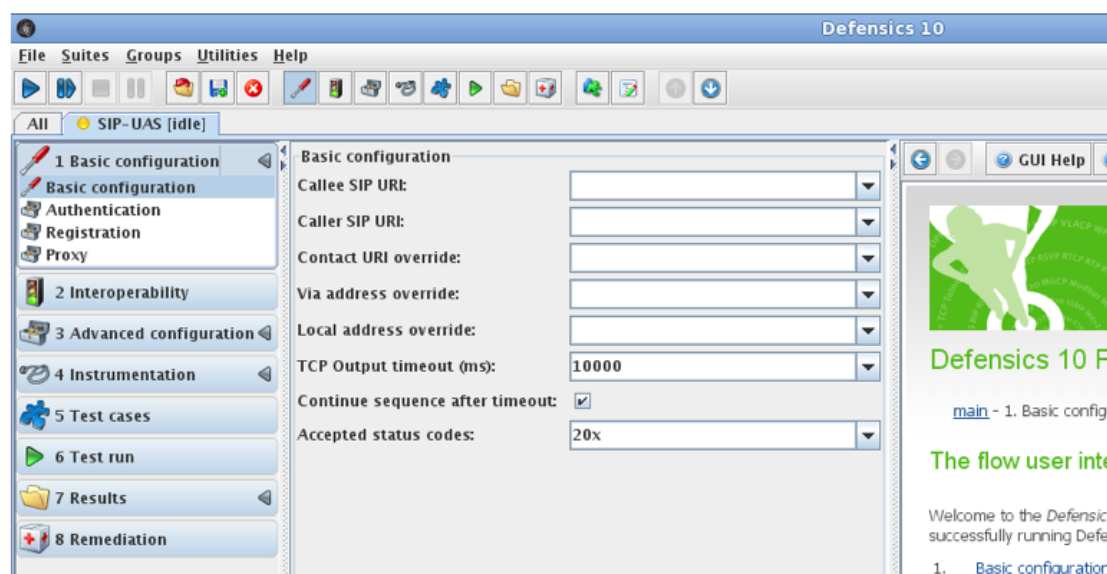


Figure 33. Test target information

The Figure 34 shows the steps of using Defensics to create a protocol model from captured network traffic. This is called template based fuzzing, as the model is created from a template file or network capture.

Step 1: Load PCAP file with network traffic and select the protocol you want to test:

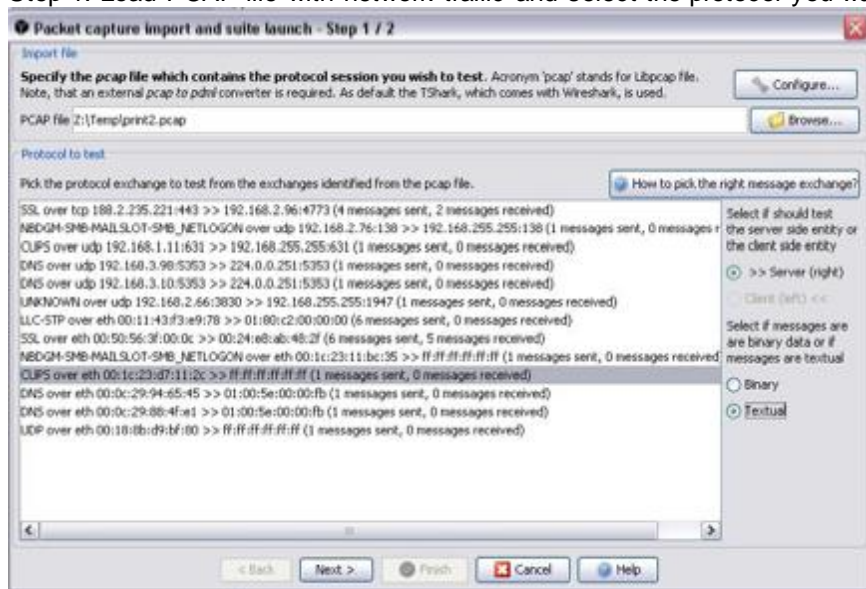


Figure 34. Step 1. Load PCAP file

Step 2: Protocol model and thousands of tests are automatically created, and the user only needs to select what elements from the protocol he wants to test:

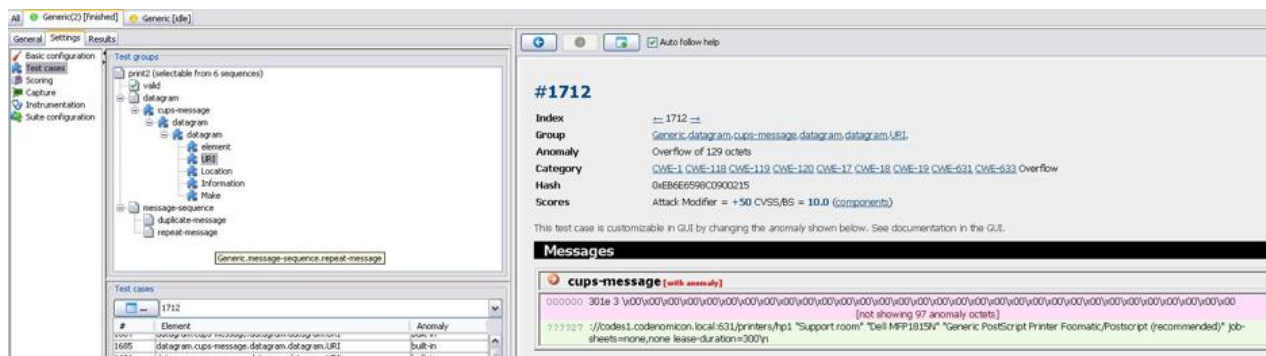



Figure 35. Step 2. Select protocol elements

Test execution and reporting work the same way as with specification-based fuzzers.

Defensics uses Extended Backus-Naur form with Java extensions as the core notation for the behavioural model. It is a proprietary extended BNF variant that allows for message exchange descriptions.

An example of Codenomicon proprietary Extended BNF (EBNF):

```
TLS ClientHello:
client-hello = Handshake: {
  msg_type: { HandshakeType.client_hello },
```

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 44 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

```

body: { @sid-len @cs-len @cm-len (
!hs:client-version protocol-version
!hs:client-random Random
.session_id_length: !sid-len:target uint8
.session_id: !sid-len:source (!hs:client-session-id SessionID)
.cipher_suites_length: !cs-len:target uint16
!cs-len:source !hs:client-cipher-suite !cipher-suite:cipher-type cipher-suites
.compression_methods_length: !cm-len:target uint8
!cm-len:source compression-methods
# RFC4366: TLS Extensions
.extensions: ()
) }
}

```

The Extended BNF is used to model the syntax of messages in both binary and textual protocols. Message sequence-level behaviour is also modelled using BNF.

The way Extended BNF works on the message sequence level is that it uses rules to append the model with callbacks to Java code. Rules are used to:

- Perform I/O operations like connect, send, and receive, on message sequence level.
- Calculate fields like lengths, checksums, and sequence numbers, inside protocol messages.

To provide the end user with lots of ready-made test cases - that is, anomalous messages and message sequences - the users of the Codenomicon test tools may also specify the used message sequences and/or message content themselves. For this purpose a proprietary XML based sequence file is used. An example sequence file for Session Initiation Protocol (SIP) based interfaces may look like the following:

```

<sequences>
  <!-- SIP dialog definitions -->
  <sequence name="used-dialog" setting="user-sequence-file">
    <description name="PUBLISH request">A sequence sending a PUBLISH request
      and expecting a success response.</description>

    <send name="publish-request" type="sip-message" description="PUBLISH request">
      <content file="sip-publish.txx" format="text"/>
      ...
      <store attribute="call-id">...</store>
      ...
    </send>

    <recv name="publish-response-ok" type="sip-message" description="OK response">
      ...
      <match attribute="call-id">!sip:callId $publish-request:call-id</match>
      ...
    </recv>
  </sequence>
</sequences>

```

The actual content of the send message "publish-request" is specified in a separate file as raw data. The sequence file specifies that the "call-id" must match the corresponding header line in the received message. Note that the real sequence specifications contain a lot more details than included here, as the SIP protocol is rather complex.

As seen in a tool screenshot below, the users can launch editors to edit both the sequence models and the message structures. The models are edited as text.

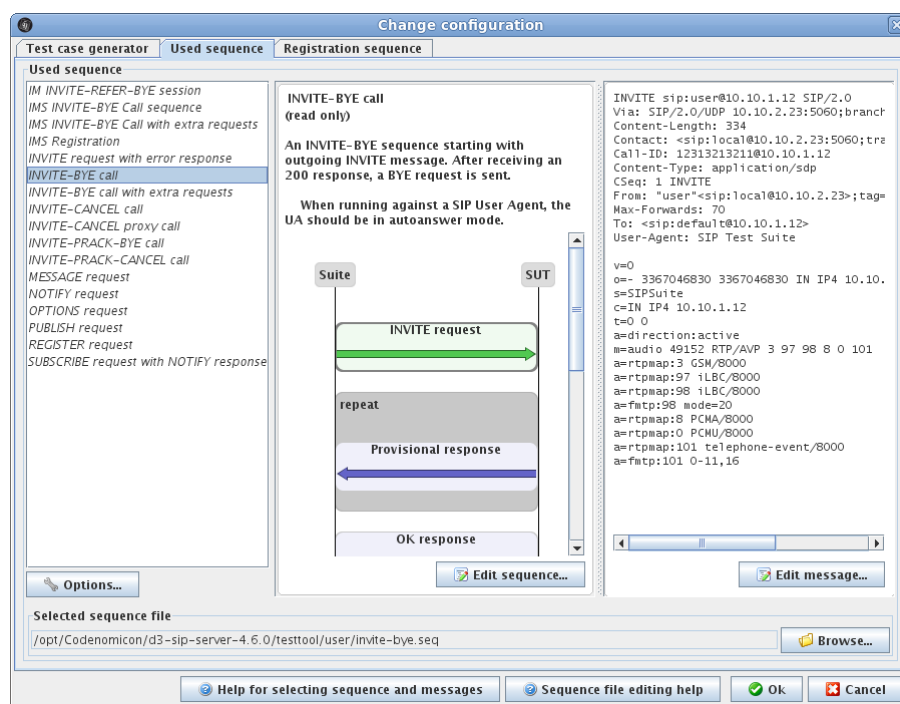


Figure 36. Model-editing in Defensics

2.5.2 Application to Case Studies

2.5.2.1 Industrial Automation Case Study

Codenomicon conducted testing with Defensics Traffic Capture Fuzzer against the Metso setup at OUSPG. The focus was to compare the functionality of Defensics and Network Hoover(tm). The same identical traffic sample was presented to both fuzzers as seed for the test case creation.


The first test runs with Hoover had resulted a system crash in Metso setup. We imported the same traffic in un-anomalised form to Defensics TCF to see if anomalised test cases from another fuzzer would catch the same vulnerability and cause the same failure mode in the target system.

The test results concluded that Defensics TCF had formed a test case from the same seed input material that caused a crash, and the same byte string was found from the output of both fuzzers.

While this comparison was limited to one vulnerability and two fuzzers only, the conclusions are not conclusive. However, based on the results from our proof-of-concept test run we can say that it is possible to trigger same vulnerabilities with two different non-modelling fuzzers, neither of which are aware of the protocol structure or traffic type at all.

2.5.2.2 Telecommunication Case Study

The work with Ericsson was done under an NDA. Results can be found from Ericsson's case study report.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 46 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.5.3 Advances during DIAMONDS

Codenomicon Defensics 10 was implemented during Diamonds, including improved monitoring capabilities with SCTP protocol. Defensics 10 included also improved UI that supports better the actual test flow. Traffic Capture Fuzzer (TCF) was developed and used in the case studies. TCF SDK development was also done within Diamonds. TCF SDK improves usability of plain SDK by adding possibility for tester to add own code to the test to improve interoperability. Defensics engine was also improved and it is now able to generate more test cases which leads to the better test coverage.

Codenomicon also implemented Web Application Test Suite tool to enable web application testing. Web Application Test Suite supports multiple formats over HTTP: HTML, URL encoded POSTs, JSON and multipart mime. It does not replace HTTP Server Suite, but complements it. Use HTTP Server Suite to tests HTTP server and Web Application Test Suite to ensure the robustness of the web application running behind the HTTP server.

2.6 SYMBOLIC PASSIVE TESTING TOOL (TESTSYM-P) (IT)

TestSym-P [26] is a prototype model implemented by Institut Telecom (IT) that aims at passively testing an Implementation under test (IUT) to verify if it respects the protocol requirements represented as IOSTS property. The prototype model is conceived to check whether the IUT collected traces respect the protocol requirements and if it does not satisfy we check if any attack patterns are satisfied based on that a verdict Pass/Fail/Inconclusive/Attack-Pass [23] is given.

2.6.1 Description of the Tool

Figure 37 illustrates the components of the TestSym-P prototype model. It has three different inputs (coloured blue):

- The real communication traces represented in .txt format (captured using Wireshark or any trace analyser for instance).
- The Guard conditions associated with each state of the symbolic executions.
- The symbolic traces collected from the symbolic execution of the IOSTS property (as defined in D5.WP2), Traces(SE(M)) (i.e. the property/attack that is to be verified on the IUT traces) and the number of states involved.

The prototype model is written in SQL. SQL is used to process the huge amount of data contained in the captured traces. We specifically used its efficiency to perform our trace slicing approach. The brief description of each module in the model is described below:

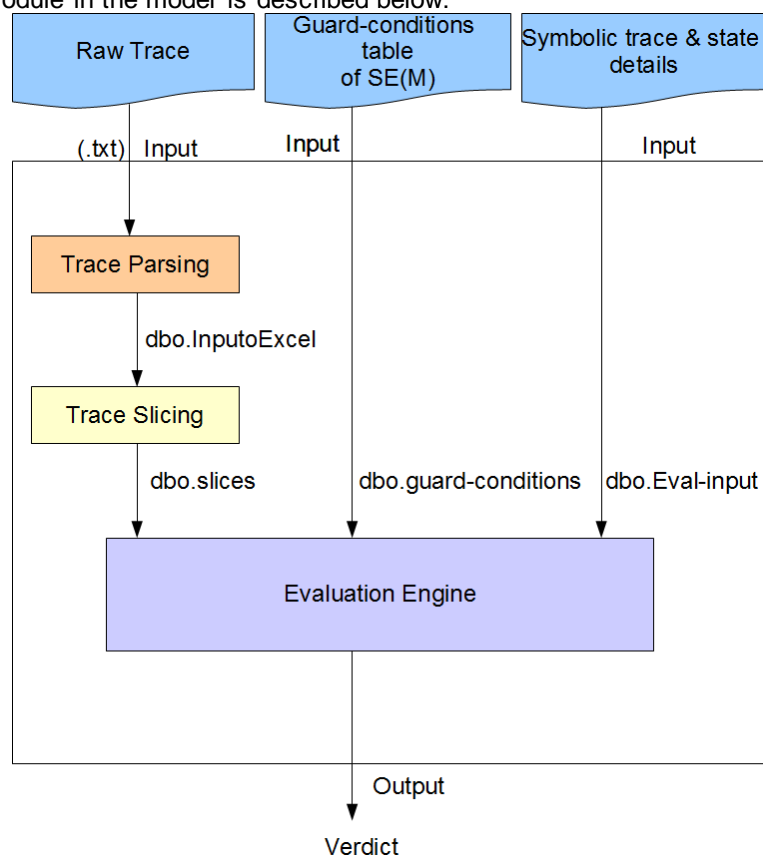
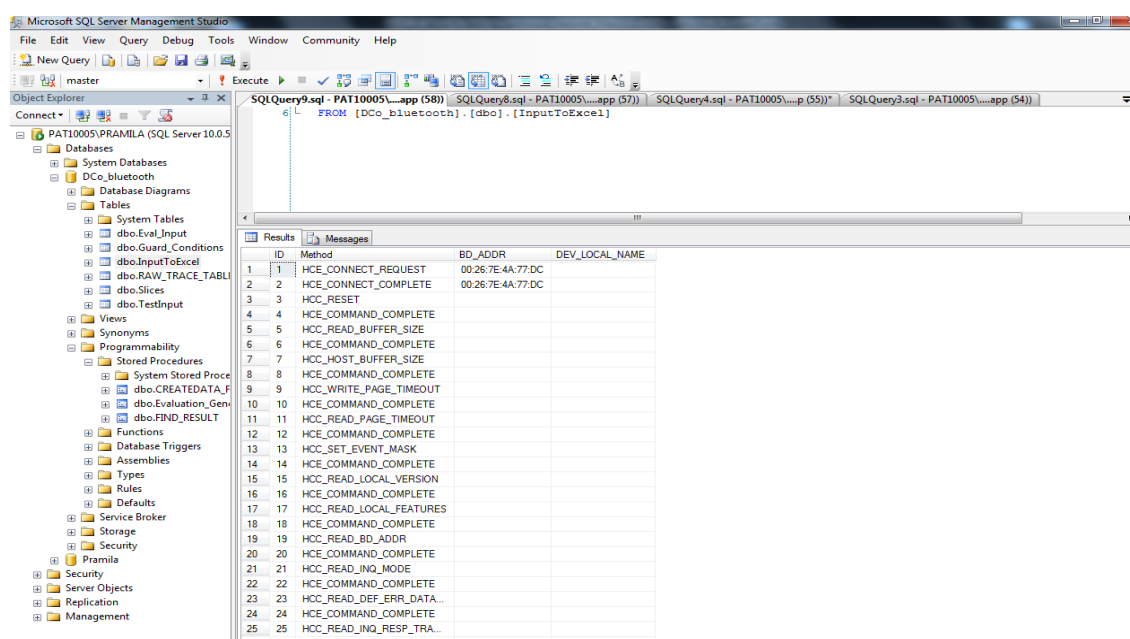


Figure 37: Architecture of TestSym-P prototype model.

A. Input 1: Raw trace input (Wireshark collected traces or from any trace analyser)

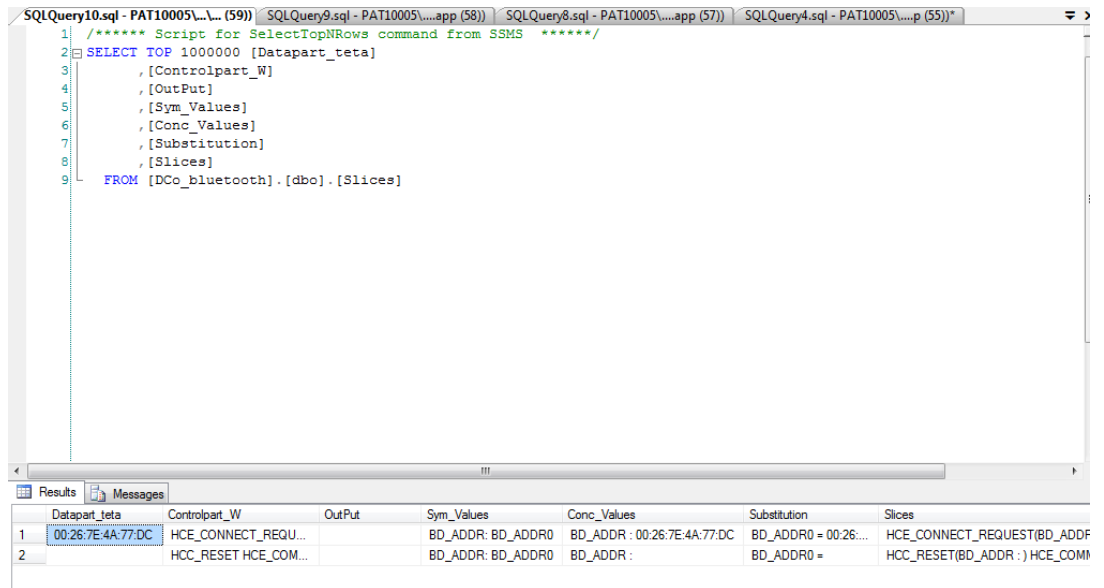
- (i) **Trace parsing:** The raw text file (.txt) obtained from the real Bluetooth framework is given as input to our tool. The tool converts the text file format .txt to a tabular file format, dbo.InputToExcel with the required Bluetooth specific fields. This table is the database table for SQL as shown in Figure 38.



ID	Method	BD_ADDR	DEV_LOCAL_NAME
1	HCE_CONNECT_REQUEST	00:26:7E:4A:77:DC	
2	HCE_CONNECT_COMPLETE	00:26:7E:4A:77:DC	
3	HCE_RESET		
4	HCE_COMMAND_COMPLETE		
5	HCE_READ_BUFFER_SIZE		
6	HCE_COMMAND_COMPLETE		
7	HCE_HOST_BUFFER_SIZE		
8	HCE_COMMAND_COMPLETE		
9	HCE_WRITE_PAGE_TIMEOUT		
10	HCE_COMMAND_COMPLETE		
11	HCE_READ_PAGE_TIMEOUT		
12	HCE_COMMAND_COMPLETE		
13	HCE_SET_EVENT_MASK		
14	HCE_COMMAND_COMPLETE		
15	HCE_READ_LOCAL_VERSION		
16	HCE_COMMAND_COMPLETE		
17	HCE_READ_LOCAL_FEATURES		
18	HCE_COMMAND_COMPLETE		
19	HCE_READ_BD_ADDR		
20	HCE_COMMAND_COMPLETE		
21	HCE_READ_INQ_MODE		
22	HCE_COMMAND_COMPLETE		
23	HCE_READ_DEF_ERR_DATA...		
24	HCE_COMMAND_COMPLETE		
25	HCE_READ_INQ_RESP_TRA...		

Figure 38: Snapshot of trace parsing (dbo.InputToExcel) table.

- (ii) **Trace Slicing:** This module uses the SQL table, dbo.InputToExcel as input and updates another table dbo.slices. A stored procedure dbo.Find-Result is written to populate the dbo.slices using dbo.InputToExcel table. The trace slicing algorithm according to our approach (as defined in D5.WP2) is implemented in this SQL procedure. The given trace file is sliced based on certain parameters of the Bluetooth protocol. This slice put together constitutes the real trace. Figure 39 shows the snapshot of the trace slicing table output obtained.



The screenshot shows a SQL query window with the following text:

```

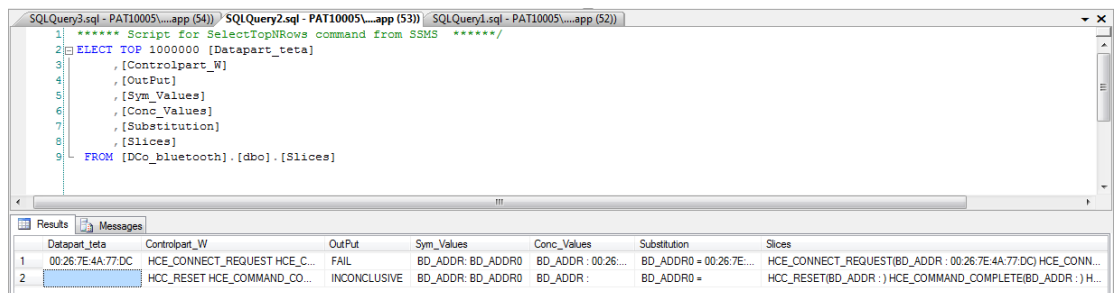
1  /***** Script for SelectTopNRows command from SSMS *****/
2  SELECT TOP 1000000 [Datapart_teta]
3    , [Controlpart_W]
4    , [OutPut]
5    , [Sym_Values]
6    , [Conc_Values]
7    , [Substitution]
8    , [Slices]
9  FROM [Dco_bluetooth].[dbo].[Slices]
  
```

The results window shows the following data:

	Datapart_teta	Controlpart_W	OutPut	Sym_Values	Conc_Values	Substitution	Slices
1	00:26:7E:4A:77:DC	HCE_CONNECT_REQU...		BD_ADDR: BD_ADDR0	BD_ADDR : 00:26:7E:4A:77:DC	BD_ADDR0 = 00:26:...	HCE_CONNECT_REQUEST(BD_ADDR...
2		HCC_RESET HCE_COM...		BD_ADDR: BD_ADDR0	BD_ADDR :	BD_ADDR0 =	HCC_RESET(BD_ADDR :) HCE_COM...

Figure 39: Snapshot of the trace slicing (dbo.slices) table.

- (iii) **Final Evaluation:** The evaluation scheme defined in D5.WP2 is implemented in the evaluation engine module. Inputs to this module are: the trace slice table, dbo.Slices, a table comprising the symbolic traces, Trace(SE(M)), total number of states involved in the symbolic execution of the IOSTS and a table with the set of associated guard-conditions G for each state of the symbolic execution. Then, the verdicts Pass/Fail/Inconclusive/Attack - Pass/Attack-Fail are obtained for each trace slices. Based on these verdicts the final verdict of the tested property on the trace evaluated. Figure 40 shows the snapshot of the output obtained.



The screenshot shows a SQL query window with the following text:

```

1  /***** Script for SelectTopNRows command from SSMS *****/
2  SELECT TOP 1000000 [Datapart_teta]
3    , [Controlpart_W]
4    , [OutPut]
5    , [Sym_Values]
6    , [Conc_Values]
7    , [Substitution]
8    , [Slices]
9  FROM [Dco_bluetooth].[dbo].[Slices]
  
```

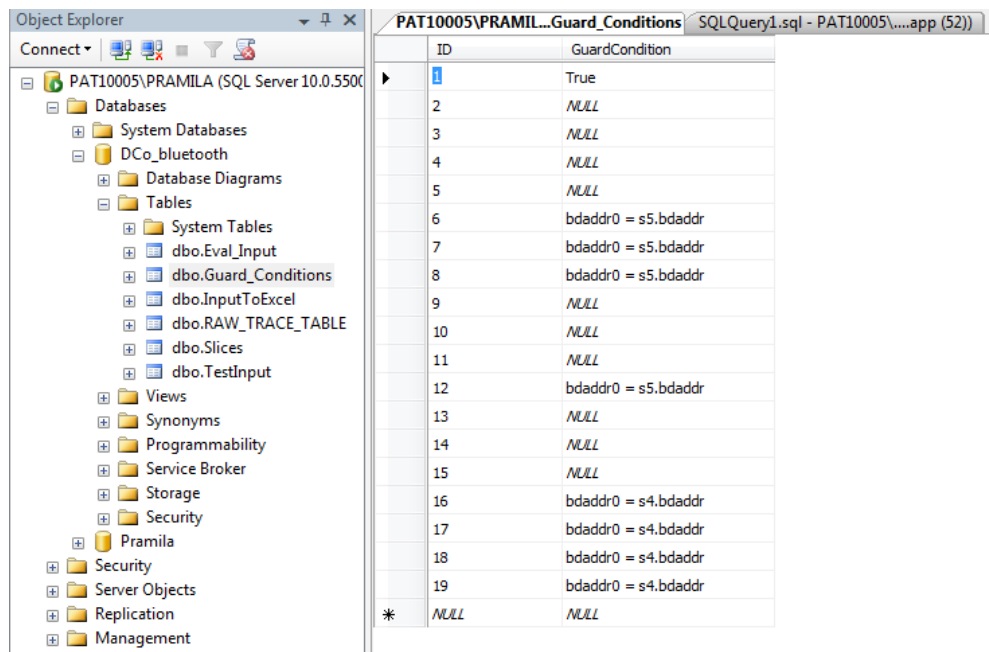
The results window shows the following data:

	Datapart_teta	Controlpart_W	OutPut	Sym_Values	Conc_Values	Substitution	Slices
1	00:26:7E:4A:77:DC	HCE_CONNECT_REQUEST HCE_C...	FAIL	BD_ADDR: BD_ADDR0	BD_ADDR : 00:26:...	BD_ADDR0 = 00:26:7E:...	HCE_CONNECT_REQUEST(BD_ADDR : 00:26:7E:4A:77:DC) HCE_CONN...
2		HCC_RESET HCE_COMMAND_CO...	INCONCLUSIVE	BD_ADDR: BD_ADDR0	BD_ADDR :	BD_ADDR0 =	HCC_RESET(BD_ADDR :) HCE_COMMAND_COMPLETE(BD_ADDR :) H...

Figure 40: Snapshot of the verdicts obtained.

B. Input 2: Guard-Conditions table of SE(M)

The guard conditions associated with each state of SE(M) are created as a table in SQL, dbo.Guard-conditions. Sample of the guard-conditions table in SE(M) (defined in D5.WP2) is shown in Figure 41.



ID	GuardCondition
1	True
2	NULL
3	NULL
4	NULL
5	NULL
6	bdaddr0 = s5.bdaddr
7	bdaddr0 = s5.bdaddr
8	bdaddr0 = s5.bdaddr
9	NULL
10	NULL
11	NULL
12	bdaddr0 = s5.bdaddr
13	NULL
14	NULL
15	NULL
16	bdaddr0 = s4.bdaddr
17	bdaddr0 = s4.bdaddr
18	bdaddr0 = s4.bdaddr
19	bdaddr0 = s4.bdaddr
*	NULL

Figure 41: Snapshot of the guard-conditions table.


C. Input 3: Symbolic trace and state details

The symbolic execution state details like Symbolic trace sequence, number of states, guard-condition number, type (property or attack) are provided as one of the input to the prototype tool for evaluation. A snapshot of the table inputs is shown in Figure 42.



ID	Seqs	GuardCondRowNr	NrOfStates	AttackSeq	SeqType
1	hcc_chng_local_name() hcc_inquiry() hcc_inquiry_complete() hcc_create_connection(...	1,2,3,4,5,16,17,18,9,10,2,19,14,15	15	0	P1
2	hcc_chng_local_name() hcc_inquiry() hcc_chng_local_name() hcc_inquiry_complete() ...	1,2,3,4,4,5,6,7,8,9,10,11,12,13,14,15	17	1	A1
*	NULL	NULL	NULL	NULL	NULL

Figure 42: Snapshot of the Symbolic state details table.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 51 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.6.1.1 Scalability of TestSym-P

The symbolic passive testing tool was also applied to monitor several SIP (Session Initiation Protocol) properties and attack patterns. For the experiments, SIP traces were obtained from two different sources:

(A) Traces were provided by Alcatel-Lucent and extracted from the interfaces of the PoC server on top of the Application Server [24]. (B) Traces were also obtained from SIPp [25]. SIPp, provided by the Hewlett-Packard Company, is an Open Source SIP implementation of a test system conforming to the IMS as well as a testing tool and traffic generator for the SIP protocol.

We actually experimented our approach to large traces ($> 10^6$ packets) to study the scalability of our approach. Promising results have already been obtained and will be used to compare our work with other data-centric approaches.

2.6.2 Application to Case Studies

IT applied the presented technique in Automotive case study provided by the German partners (DCo). In this case study, the connection of the car's entertainment system with the driver's mobile phone via Bluetooth was monitored. The traces obtained were collected and monitored using the symbolic passive testing approach (defined in D5.WP2). Connections via Bluetooth are open to foreign devices and thus, provide an attack point. Hence in this case study, IT applied the passive testing technique to monitor the functional behaviour and the attack scenario.

2.6.3 Advances during DIAMONDS

The TestSym-P tool has been fully implemented during the Diamonds project. This tool can be used to passively test any message-based protocol like SIP, HTTP, etc.

2.6.3.1 Conclusions and Future work

Currently, the TestSym-P tool is efficient enough to monitor behavioural properties and attack patterns for the Bluetooth protocol. Our prototype and the sample files used for the experiments can be found at http://www-public.it-sudparis.eu/_mouttapp/TestSym.html. Though the current work deals with offline monitoring, as future works it would be interesting to support online monitoring or runtime monitoring. The approach can be applied to monitor any type of message-based protocols.

2.7 MONTIMAGE MONITORING TOOL (MONTIMAGE)

2.7.1 Description of the Tool

MMT (Montimage Monitoring Tool) is a monitoring solution that combines: data capture; filtering and storage; events extraction and statistics collection; and, traffic analysis and reporting. It provides network, application, flow and user level visibility. Through its real-time and historical views, MMT facilitates network security and performance monitoring and operation troubleshooting. MMT's rules engine can correlate network and application events in order to detect operational, security and performance incidents.

In the context of the DIAMONDS project, Montimage developed MMT-Security that is a functional and security analysis tool (part of MMT) that verifies application or protocol network traffic traces against a set of MMT-Security properties. MMT-Security properties can be either "Security rules" or "Attacks" as described by the following:

- A Security rule describes the expected functional or security behaviour of the application or protocol under-test. The non-respect of the MMT-Security property indicates an abnormal behaviour.
- An Attack describes a malicious behaviour whether it is an attack model, a vulnerability or a misbehaviour. Here, the respect of the MMT-Security property indicates the detection of an abnormal behaviour that might imply the occurrence of an attack.

MMT-Security can be executed against a pre-recorded trace file or live on a network interface using real time analysis.

2.7.1.1 MMT-Security architecture

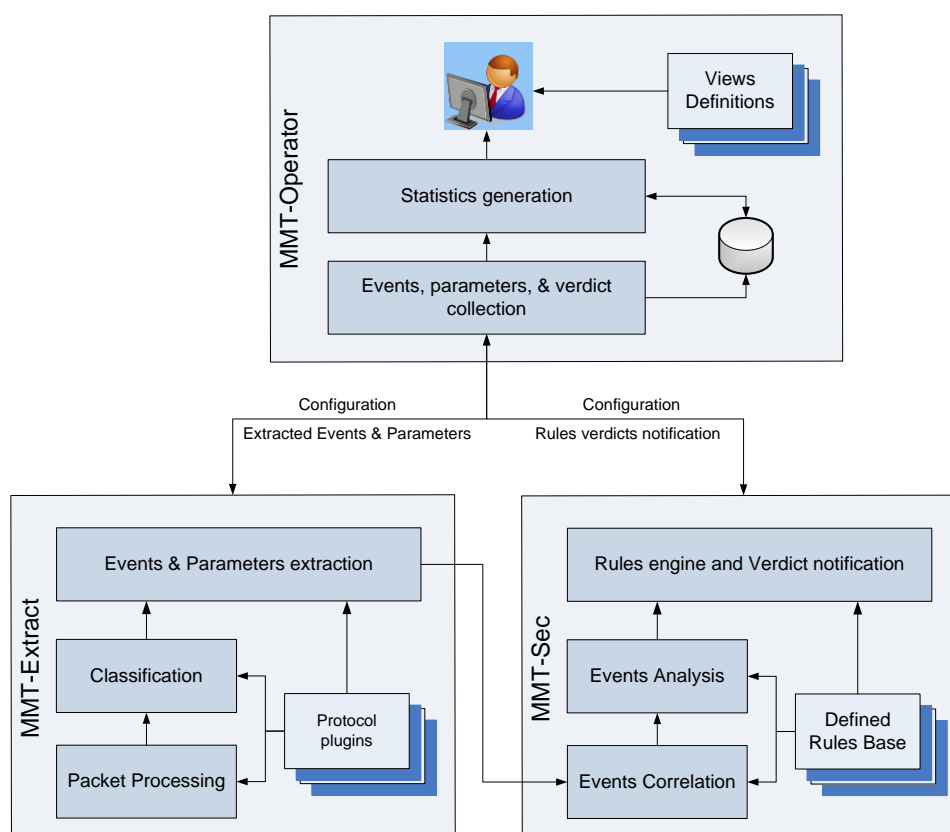



Figure 43. MMT-Security Architecture

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 53 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

MMT-Security is composed of three complementary, but independent, modules as depicted in Figure 43.

- **MMT-Extract** is the core packet processing module. It is a C library that analyses network traffic using Deep Packet/Flow Inspection (DPI/DFI) techniques in order to extract hundreds of network and application based events, including: protocols field values, network and application QoS parameters and KPIs. MMT-Extract incorporates a plugin architecture for the addition of new protocols and a public API for integrating third party probes.
- **MMT-Security** is a security analysis engine based on MMT-Security properties described in D2.WP2 section 1.4.1. MMT-Security analyses and correlates network and application events to detect operational and security incidents. For each occurrence of a security property, MMT-Security allows to detect whether it was respected or violated.
- **MMT-Operator** is a visualization application for MMT-Security. It allows collecting and aggregating security incidents, and presents them via a graphical user interface (e.g., report tables). MMT-Operator is customizable: the user is able to define new views or customize the large list of predefined ones. With its generic connector, MMT-Operator can be integrated with third party traffic probes.

2.7.1.2 MMT-Security features

Granular traffic analysis capabilities: MMT allows parsing a wide range of protocols and applications and to extract various network and application based traffic performance indicators. The extraction is powered by a plugin architecture for the addition of new protocols and applications.

Application classification: prior to extracting protocol or application attributes, MMT uses DPI techniques for application identification and classification. This is essential when analysing applications that use non-standard port numbers (e.g. P2P, Skype).

Rule engine: that allows the detection of complex sequence of events that conventional monitoring does not detect. This is used to monitor: i) access control policies (e.g. authorized users are authenticated prior to using a critical business application); ii) anomaly or attacks (e.g. excessive login attempts on the application server); iii) performance (e.g. identification of VoIP calls with QoS parameters exceeding acceptable quality thresholds); etc.

Configurable reports: MMT traffic reports and charts can be configured by the user. The user can edit pre-configured reports and create new ones. Different chart types and graphs can be used including: pie, histograms, time charts, stacked area charts, sequence charts, tables, hierarchical tables, etc.).


Multi-platform solution: MMT is available on Windows and Linux based distributions. It can be installed as software on commodity hardware or optimized for integration with dedicated probes.

Modular solution: MMT is a modular solution composed of three components: MMT-Extract for the traffic processing and data decoding; MMT-Sec for rules analysis; and MMT-Operator for data aggregation, correlation and reporting. It is possible to integrate MMT-Extract and MMT-Sec in third party traffic probes and to connect MMT-Operator with existing systems.

2.7.2 Application to Case Studies

The MMT tool has been applied to three case studies:

- Radio protocols case study provided by Thales Communication & Security
- Smartcards and the mobile NFC ecosystem case study provided by Gemalto
- Automotive case study provided by Dornier Consulting


	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 54 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

Case study	Offline/Online	Number of designed properties	Number of analysed traces (including traffic or server logs)	Results
Radio protocols case study	Both	19	17	No security flaws detected + All attacks implemented and applied are detected
Smartcards and the mobile NFC ecosystem case study	Offline	9	3	No security flaws detected
Automotive case study	Offline	3	2	No security flaws detected

More details are presented in D5.WP1 deliverable.

2.7.3 Advances during DIAMONDS

The MMT tool has been fully implemented in the context of Diamonds project. It is the result of previous research work in the network monitoring field and relies on the multi-domain security requirements identified in the context of Diamonds case studies. The main idea of MMT is that its main core tool can be easily extended (based on a plugin architecture) to fit any new constraints of any communicating system under test.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 55 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.8 MALWASM (ITRUST)

2.8.1 Description of the Tool

malwasm is an open source tool designed byitrust consulting with the intention to help reverse engineers understand what a binary does.

Presently we can identify 2 different types of malware analysis.

- Static analysis;
- Dynamic analysis.

Static analysis consists of obtaining the assembly code of an application in order to understand how the application works. However today, the majority of malware use obfuscation to hide their activities. For example, the windows registries name modified by a malware can be hidden using compression or encryption.

The dynamic analysis consists of executing a program step-by-step, instruction-by-instruction. In this case, the analysis can take a long time. If the analyst goes too far, he has no way to go back and the only solution is to restart the analysis from the beginning.

The malwasm tool, designed byitrust consulting, incorporates both static and dynamic analysis; enabling reverse engineers to perform a rigorous malware analysis.

An online demo of the tool is available at <http://malwasm.com>.

2.8.1.1 How it works

The following sub-chapters describes in steps, how the tool works.

Step 1

malwasm starts a virtual machine using VirtualBox;

Step 2

The second step involves executing a malware sample in order to obtain a database of information. The sample is executed in a dedicated virtual environment in order to record all interaction with the logical or physical parts of the machine where it is executed. As the execution of the sample could lead to a system crash, the sample is executed in an isolated sandbox, which records system behaviour in real-time and in a secure manner.

Step 3


A PinTool application logs all the activity (and partial activity) of the binary.

Step 4

All the activity of the binary is stored in a postgres database.

Step 5

The recorded data (as a raw report of the sample behaviour) is displayed in a smart web interface which allows step by step monitoring of the sample behaviour and its interaction with the system environment (data flows, input/output analysis).

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 56 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.8.1.2 Features

- Offline program debugging;
- Timeline monitoring of the malware execution;
- Register and flag states;
- Stack/heap/data values;
- Multiple options following the dump;
- Supports x86 architecture;
- Supports multithreads.

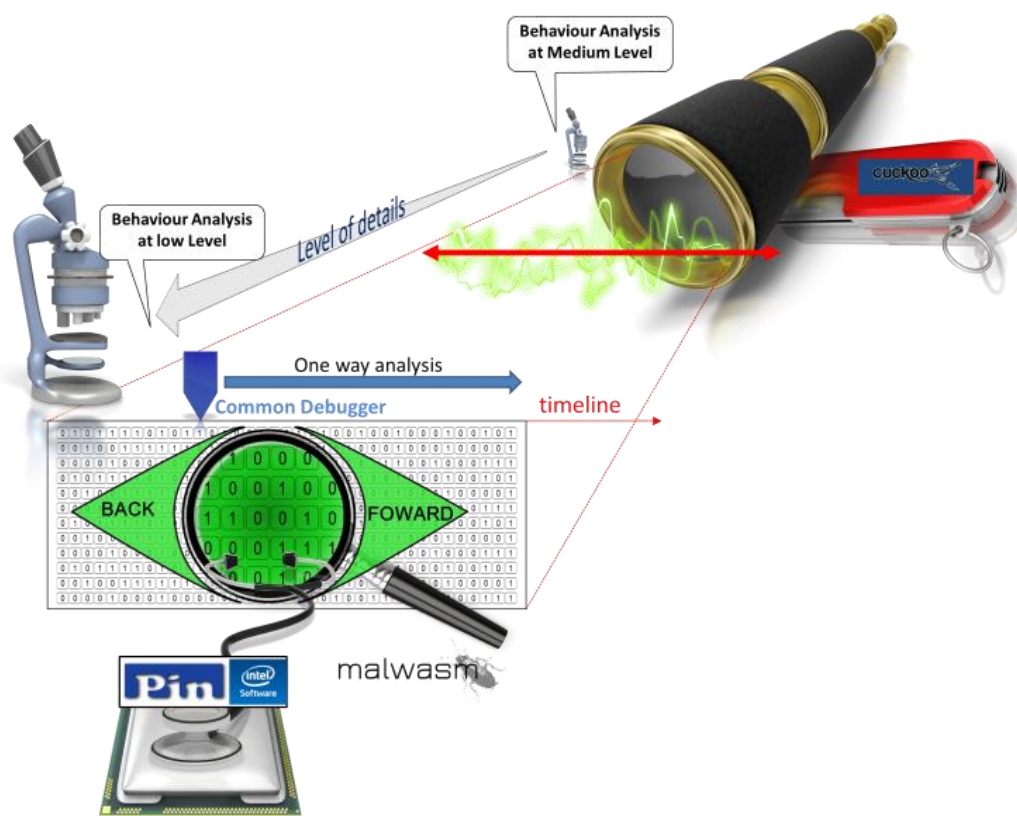
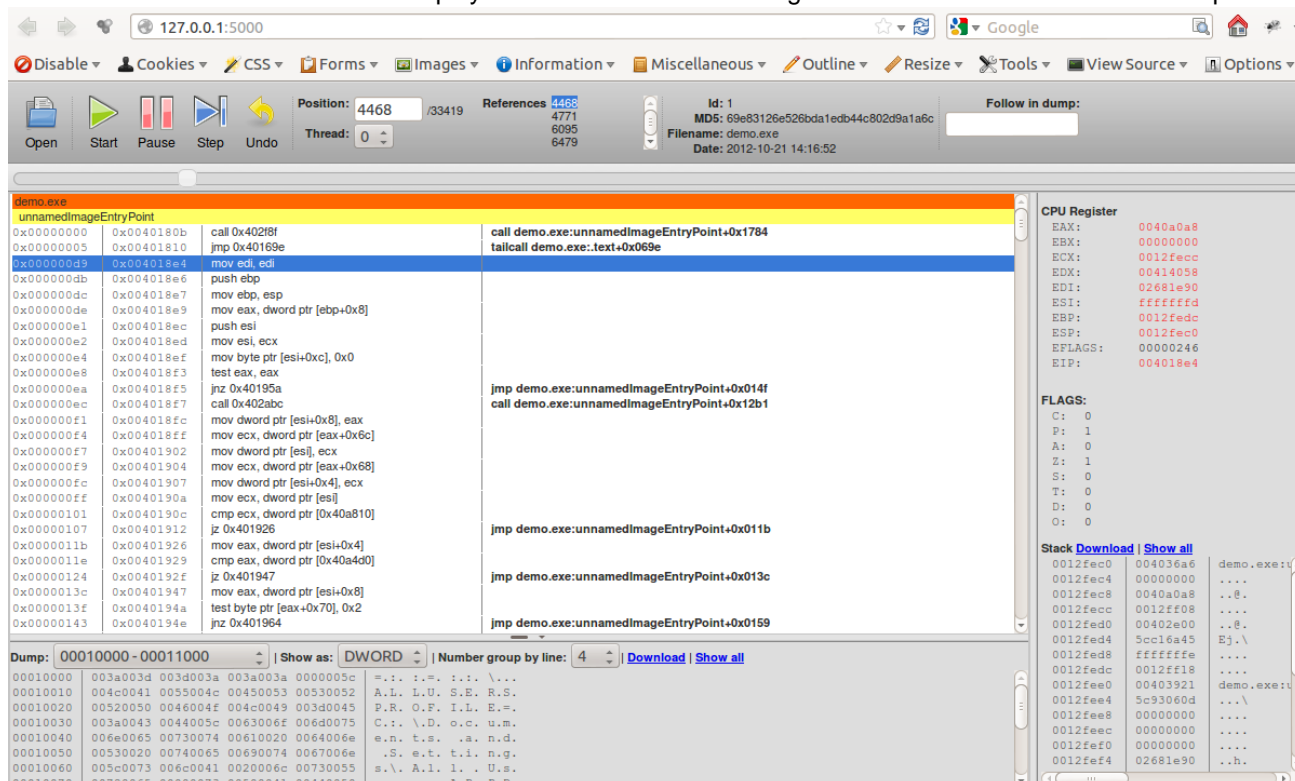


Figure 44: malwasm features


2.8.1.3 Interface

malwasm uses a web browser to display the information concerning the execution of the malware sample.



The screenshot displays the malwasm web interface in a web browser. The address bar shows '127.0.0.1:5000'. The interface includes a toolbar with buttons for Open, Start, Pause, Step, and Undo. Below the toolbar, there are controls for Position (4468), Thread (0), and References (4468). The main area shows assembly code for 'demo.exe' with columns for address, disassembly, and comments. The CPU Register section on the right shows the state of various registers (EAX, EBX, ECX, etc.) and flags. The Stack section at the bottom shows memory addresses and their contents.

Figure 45: malwasm web-interface

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 58 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.8.1.4 Innovation

The malwasm tool can be classified as a reverse engineering tool and is based on known products such as cuckoo or PinTool. However, malwasm provides at least 2 major innovations compared to similar tools. Figure 46 below shows the general architecture of malwasm.

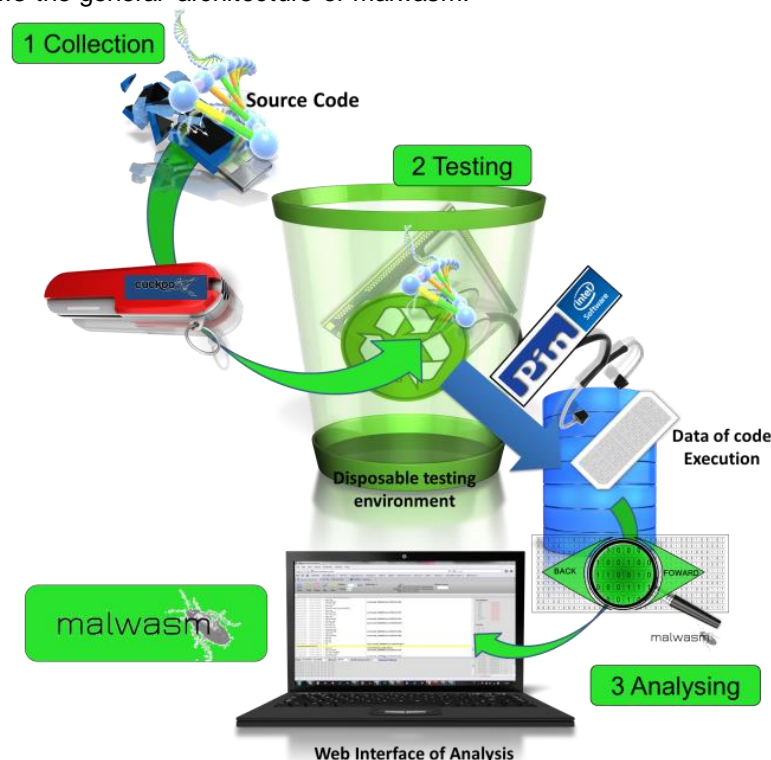


Figure 46: malwasm architecture

malwasm uses Cuckoo Toolbox features to run the code sample in a dedicated and disposable environment. In this environment, malwasm uses PinTool (Intel software) to record the behaviour of the sample execution into a dedicated database. The web interface of malwasm allows replaying the sample execution step by step, forwards and backwards, and monitoring system activities during the execution of the sample.


The main innovation of malwasm is to bring together the advantages provided by the Cuckoo ToolBox, i.e. the disposable environment, the collection of data, the advantages of the low level analysis provided by PinTool, and also to avoid the limits of, on one hand, a common debugger which can only work one way (forward); and on the other hand the limits of Cuckoo ToolBox which provides only an overview of the system behaviour.

Additionally, the malwasm tool provides a web interface to replay step by step the sample execution and to monitor the main parameters of system behaviour according to displayed sample instructions (displayed in byte, word or dword).

2.8.1.5 Exploitation

The tool is used during the creation of malware analysis articles, which itrust publicly shares on the website malware.lu. The articles can be used by reverse engineers, students, security researchers, etc. in order to better understand how malware works.

The website also contains a repository of malware, which authorised members can download and analyse, using the malwasm tool.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 59 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.8.1.6 Support


malwasm is an open source tool, developed and maintained by the ethical hacking team ofitrust consulting. It can be downloaded at <http://code.google.com/p/malwasm/>. An installation guide can be found at <http://code.google.com/p/malwasm/wiki/README>. It contains instructions on how to install malwasm and on how to install support software such as Cuckoo or PinTool.

2.8.2 Application to Case Studies

2.8.2.1 Gemalto case study

For the creation ofitrust's smartcards case study, malwasm was used to debug our developments in order to gain a greater understanding of how smartcards work. The malwasm tool alloweditrust's reverse engineers to develop a piece of malware which successfully exploited weaknesses in the smartcard's design.

2.8.3 Advances during DIAMONDS

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 60 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.9 TRICK TESTER (ITRUST)

2.9.1 Description of the Tool

TRICK Tester is a platform used for penetration (intrusion) testing. It contains all the main software tools to test web applications and network systems like Web Servers.

The involvement of itrust in research and development projects allows the company to gain a deep understanding of the software tools and optimise their usage. Additionally, the organisation's participation in research and development projects has led itrust to develop its own innovative security testing tools which are often used during the vulnerability tests.

2.9.1.1 TRICK Tester support


TRICK tester consists of an ISO image based on the Linux distribution Ubuntu. This ISO image can be burnt on DVD or can be copied on a USB flash drive (this solution provides a more reactive operating system) in order to be launched as a live operating system. In this case, hard disks can be easily analysed without data to be written on the running system. The tool can also be installed permanently on a hard disk which allows more reactivity of the operating system and storage of data during the tests.

New releases of TRICK Tester are only provided as a new ISO image and not through the Linux update manager. It is recommended to have a different location to store data apart from the operating system partition. This ensures that data is not lost when installing a new version of TRICK Tester.

2.9.1.2 Launching TRICK Tester

After a successful login, documentation such as manuals and guides are available on the Desktop. The documentation contains:

- The penetration methodology;
- The different intrusion attacks (e.g. SQL injections, XSS) explanations and examples;
- How to use the different intrusion software tools that are installed.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 61 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.9.1.3 Security criteria

TRICK tester methodology aims at detecting security issues on the three main criteria of information security:

- **Confidentiality:** TRICK tester enables to check that data transmitted by or stored on web applications is correctly protected against divulgation. For example, sensitive data of a web application accessed by a user must be transmitted through a protected protocol (e.g. HTTPS);
- **Integrity:** TRICK tester helps to check that data stored by web application cannot be modified by unauthorised users. This can be done for example by verifying that it is not possible to modify data by SQL injection on the database;
- **Availability:** TRICK tester includes software to test Denial of Service.

2.9.1.4 Test results


Results found during intrusion tests have to be checked on their level of impact (rating). TRICK Tester outputs (e.g. results of web application scanning) enable auditors to list the vulnerabilities to be corrected. This list is sorted considering the level of impact of the vulnerability. Four levels exist:

Level	Description
● ● ● ●	Recommendations requiring an immediate action, for example in case of a known vulnerability, to avoid an unacceptable risk.
● ● ●	Recommendations requiring a prompt action to avoid a high risk or to prevent a certification
● ●	Recommendations requiring a dedicated action to re-establish best practices, to reduce a medium risk or to increase compliance
●	Recommendations requiring a dedicated action useful for improving security and to reduce rather low risk

2.9.1.5 Exploitation and Dissemination


TRICK tester can be sold to customers wanting to automate security testing for example when making internal audit. TRICK tester could then be sold as following, providing:

- Training to customers for the use of the Live DVD and its tools;
- Support as making new releases new version of the Live DVD, including new software;
- Help to choose and apply the relevant security checks on the customer's information system.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 62 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

2.9.1.6 List of TRICK Tester tools

Tool	Type	Description
Nexpose	Application vulnerability scanner	Capable to scan a system for known vulnerabilities
OpenVAS	Application vulnerability scanner	Tool to scan a system for vulnerabilities
Fast-track	Application vulnerability scanner and exploiter	This tool contains multiple tools with the target to check vulnerabilities on a network and to exploit the vulnerabilities
Metasploit	Application vulnerability scanner and exploiter	a tool for developing and executing exploit code against a remote target machine
WeBcoo	Backdoor	Web Backdoor Cookie Script-kit
Mantra	Browser	Browser developed by OWASP especially for web application security testing
Dpscan	CMS scanner	Scans the Drupal CMS for vulnerabilities
JoomlaScan	CMS scanner	Scans Joomla CMS for vulnerabilities
Plecost	CMS Scanner	Scans the WordPress CMS for finger information
WPScan	CMS Scanner	WordPress CMS vulnerability scanner
InstantClient	Database connector	Client software that connects to Oracle databases
dirbuster	Folder structure scanner	Tool that brut forces file and folder names of a web server using a given list of file names
findmyhash.py	Hash cracker	Python script that tries to crack different types of hashes using free online services
HttpPrint	Information gathering	A web server fingerprinting tool that retrieves data of a web server.
P0f	Information gathering	Passive finger information gathering of an operating system
Maltego	Information gathering	Capable to sniff relations between entities on the network
Wireshark	Information gathering	Tool to capture communications received and sent over a network interface
Volatility / Volatilitux	Memory analysis	Dumps the volatile memory of a system to analyse it
SIPVicious	Network audit	Tool to audit SIP based VOIP systems
Hostmap	Network discovery	Hostname and virtual host discovery tool
NetBScanner	Network scanner	Scans computers and IP's of a given range
Scapy	Packet manipulator	Tool capable of modifying OSI level 2 Ethernet packets, to alter requests and responses

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 63 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

Tool	Type	Description
Rainbowcrack	Password Cracker	Break hash using rainbow tables
Pdfid.py / pdf-parser.py	PDF file parser	Parses PDF files to scan for malicious code that represents a malware
nmap / Zenmap	Port Scan	Tools used to scan ports on a target IP or ranges of IP's, can be extended with plugins to gather other data.
Burp	Proxy	Tool with multiple possibilities: web application scanner, spider, intruder, proxy...
spkproxy	Proxy	Spike Proxy
WebScarab	Proxy	Tool used as proxy capable of intercepting and altering HTTP and HTTPS requests and responses of browsers (and spider, intruder...)
binwalk	Reverse engineering	Tool to analyse firmware images and other binary blobs
Edbdebugger	Reverse engineering	Cross platform application debugger and code analyser
Firmware Mod Kit	Reverse engineering	It allows easy deconstruction and reconstruction (debugging) of firmware images
Flare	Reverse engineering	Freeware Action Script decompiler
IDA	Reverse engineering	Disassembler and debugger tool
Jdd	Reverse engineering	Java code decompiler
Metasm	Reverse engineering	Metasm is a cross-architecture assembler, disassembler, compiler, linker and debugger
Sydbbox	Sandbox	Tool to run programs in a sandbox based on ptrace
SET	Simulator	Social Engineer Toolkit, capable to simulate common human reactions to requests
SQLBrute	SQL injection	A tool for brute forcing data out of databases using blind SQL injection vulnerabilities
sqlmap	SQL injection	Automatic SQL injection and database takeover tool
sqlninja	SQL injection	Automatic SQL injection and database takeover tool
The mole	SQL injection	Command line tool for SQL injection exploitations
sslscan	SSL scanner	Extract security relevant details of a SSL
Hachoir	Stream parser	Allows to view and edit a binary stream
Pinktrace	Tracer	Traces data of a system process


Tool	Type	Description
wfuzz	Web application vulnerability scanner	a tool designed to brute force web applications
Grendel-Scan	Web application vulnerability scanner	Tool used to scan a web application for vulnerabilities
Nikto	Web application vulnerability scanner	Tool used to scan a web application for vulnerabilities
Paros	Web application vulnerability scanner	Tool used to scan a web application for vulnerabilities
w3af	Web application vulnerability scanner	Tool used to scan a web application for vulnerabilities
GNUradio	Wireless signal processor	Capable of capturing and analysing wireless signals
Aircrack-ng	Wireless WPA-PSK cracker	Tool capable to audit wireless networks and password cracker for WEP/WPA-PSK secured wireless networks

2.9.2 Application to Case Studies

We applied TRICK tester for the security testing of the LASP use case. The LASP use case contains web technologies (web services) and a web server, which were the target of the security testing. The results of the application of TRICK tester on the LASP use case can be found in the document RAP025_Intrusion_LAP_v1.0.docx.

This security testing report contains one recommendation considered as requiring a prompt action to avoid a high risk, and 9 recommendations requiring a dedicated action from relevant staff to avoid inconsistencies with documentation and non-compliance with what is expected, which represents a rather low risk.

2.9.3 Advances during DIAMONDS

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 65 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

2.10 TTCN-3 FUZZ TESTING

2.10.1 Description of the Tool

TTworkbench is a highly integrated test development and execution platform for a wide range of industry domains, including telecommunications, automotive and financial domains, produced by Testing Technologies. TTworkbench supports the entire lifecycle of TTCN-3 based tests with textual and graphical editors, a TTCN-3 to Java compiler, and a test execution management environment composed of graphical tracing, debugging and reporting facilities for centralized and distributed test components. TTworkbench supports, additionally to TTCN-3, different system modelling languages such as ASN.1, Google Protocol Buffers, IDL, WSDL and XML.

The Testing and Test Control Notation version 3 (TTCN-3) is a standardized test specification language created 10 years ago by the European Telecommunications Standards Institute (ETSI) that is becoming more and more popular. Having its root in the testing of telecommunications protocols, TTCN-3 spreads now throughout a large number of domain such as Mobile Telecommunications: 3G, 3GPP LTE, WiMAX, GSM, Internet protocols: SIP, IMS, IPv6, Intelligent transport systems (ITS), IOT – Internet Of Things (building automation, smart metering, etc.), Automotive (AUTOSAR conformance and acceptance testing) and Smart Cards.

These domains have integrated TTCN-3 into their own methods and processes. Security Testing, a rather new domain for TTCN-3, provides a test method called Fuzzing. Fuzzing or Fuzz Testing is a testing technique that monitors a system for exceptional behaviour (such as crashes, memory leaks) while stimulating it with random, invalid or unexpected input data.

In order to be able to apply this method with using TTCN-3, there was a need to extend the standardized language to support fuzz testing. Data Fuzzing with TTCN-3 is a combined effort of FOKUS Fraunhofer and Testing Technologies in the context of the DIAMONDS project. For this reason the following extension of the TTCN-3 standard was discussed.

2.10.1.1 *TTCN-3 Core Language Extensions*

TTCN-3 packages are intended to define additional TTCN-3 concepts, which are not mandatory as concepts in the TTCN-3 core language or in its interfaces TRI and TCI, but which are optional as part of a package which is suited for dedicated applications and/or usages of TTCN-3. The package presented is identified by the package tag: "TTCN-3:2013 Security Testing" - to be used with modules complying with it.


This package defines the TTCN-3 means to define fuzz functions. Fuzzing operations are defined on basis of the TTCN-3 type system and formally specified by special fuzz functions. The fuzzing itself (i.e. the generation of fuzzed data) is done implicitly during the call of the send operation, or explicitly by calling `valueof`. The repeated application of data fuzzing, i.e. generation of multiple variants to be sent, can be realized via loop constructs. To allow deterministic test cases and to support repeatability we are using pseudo randomness specified on basis of a constant seed. While simple dump random fuzzing often causes poor results, intelligent application/protocol based fuzzing is much more powerful. To support application/protocol based fuzz generators fuzz functions can also specified as external functions.

2.10.1.2 *The fuzz function*

Syntactical Structure

```
external fuzz function ExtFunctionIdentifier "(" [ { FormalValuePar [","]
} ] ")" return Type
```

```
fuzz function FunctionIdentifier "(" [ { FormalValuePar [","] } ] ")" [
runs on ComponentType ] return Type StatementBlock Semantic Description
```

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 66 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

The concept of a `fuzz` function is similar to the present ordinary functions, but their evaluation is delayed until a specific value is selected via `send` or `valueof` operation (lazy evaluation). Fuzz functions may declare formal (in) parameters, and must declare a return type. Apart from the time of evaluation fuzz functions are treated as usual function resp. external functions, hence no extension of the runtime interfaces is required.

EXAMPLES:

```
external fuzz function fxz_UnicodeUtf8ThreeCharMutator(
    in charstring p_param1) return charstring;

fuzz function fz_RandomSelect(
    in RecordOfInteger list) return integer {
    return list[float2int(rnd(getseed())) *
        int2float(lengthof(list))];
}
```

Generally, `fuzz` function instances are used to replace values of single template fields or to replace even the entire contents of a `template`. Fuzz function instances may also be used in-line. A `fuzz` function instance represents an element of a set of values (the function range), and can only occur in value templates used like a native matching mechanism “instead of values” to define a list of values or templates.

EXAMPLES:

```
template myType mw_myData := {
    field1 := fxz_UnicodeUtf8ThreeCharMutator("abc"),
    field2 := '12AB'O,
    field3 := fz_RandomSelect({ 1, 2, 3 })
}
```

A single value will be selected in the event of a sending operation or of the invocation of a `valueof` operation.

EXAMPLES:

```
myPort.send(mw_myData);
myPort.send(fxz_UnicodeUtf8ThreeCharMutator("abc"));
var myType v_myVar := valueof(mw_myData);
```

Storing the selected value of a `fuzz` function for later use is possible within the sending operation using the `'-> value'` notation and via explicit invocation of the `valueof` operation.

EXAMPLES:


```
myPort.send(mw_myData) -> value v_myVar;
myPort.send(fxz_UnicodeUtf8ThreeCharMutator("abc")) -> value v_myVar;
var myType v_myVar := valueof(mw_myData);
```

Restrictions:

- Fuzz functions shall not be used as incoming templates.
- All formal parameters must have direction in.
- Fuzz functions must have a return type.

2.10.1.3 The seed

To allow repeatability of fuzzed test cases, an optional seed shall be used. There will be one seed per test component. Two predefined functions will be introduced to set the seed and to read the current seed value of the component calling the function. This value exposes the seed used by the predefined `rnd` function. When

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 67 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

`rnd` is called, this always causes the seed to be implicitly set to the result of the `rnd` call. When `rnd` is called with a seed parameter, this is equivalent to calling `setseed` with the same seed parameter before the `rnd` call.

Syntactical Structure

```
setseed "(" in float initialSeed ")"
getseed "(" ")" return float
```

Semantic Description

The `setseed` predefined function allows setting of a special seed value per test component. With `getseed` this value can be read out. Without a previous initialization a value calculated randomly will be used as seed.

Upon creation of a new test component a new seed will be created implicitly, if no seed is set via `setseed`.

EXAMPLES:


```
setseed(1.0);
var float v_f := getseed();
```

2.10.2 Application to Case Studies

The tool is used solely for the banking case study. More information on the case study related information can be found in the corresponding DIAMONDS deliverable considering the use cases.

2.10.3 Advances during DIAMONDS

The tool has been fully developed within the DIAMONDS project.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 68 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

3. INTEGRATION PLATFORM

3.1 TOOLS INTEGRATION FOR SECURITY TESTING

3.1.1 Integration into the Radio Protocol Framework

The following figure presents the security provider tools integration on the Radio Protocol Case Study. This framework meets all the features expressed in the previous section.

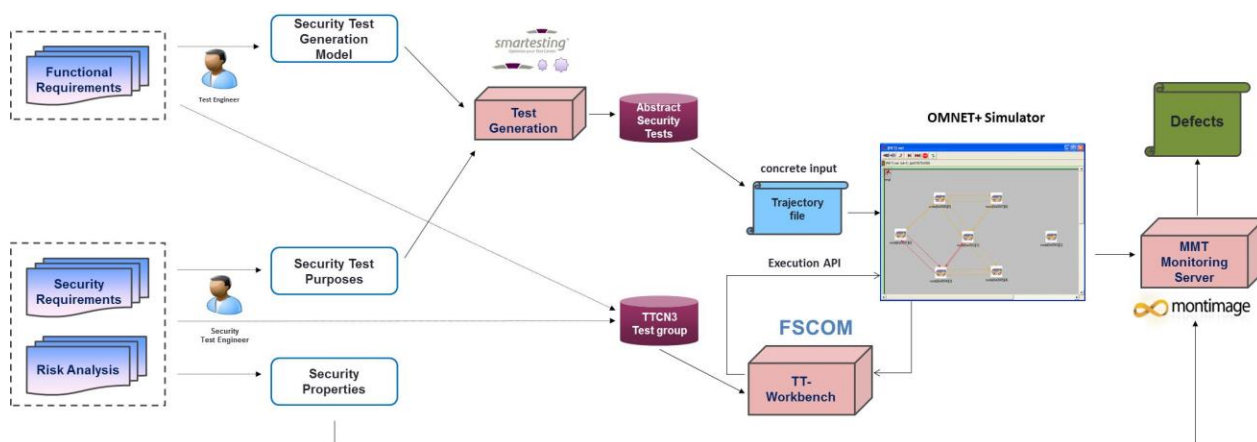



Figure 47: Tools integration (tools provider view)

3.1.2 Tools Integration:

- ❖ **MONTIMAGE:** Montimage worked on the integration of the monitoring tools toolset called MMT (for Montimage Monitoring Tool) into the THALES validation framework. In addition of improvements of the monitoring kernel engine, Montimage contribution on this case study is manifold:
 - Parsing of the fields of the specific THALES protocol PDUS.
 - Addition of online analysis of a previous offline analysis.
 - Improvement of the expressiveness of the security properties notation.
 - Verification of properties. Close collaboration with THALES team on the diagnosis of the results.
- ❖ **SMARTESTING:** In addition to the improvement of the automatic test generation kernel engine, Smartesting worked on the modelling of the system under test UML/OCL specification and scenarios generation of nodes instantiations, movements and traffic set up, including intrusive nodes, and directives information provided to monitoring tools.
- ❖ **FSCOM :** FSCOM worked on the TTCN specification of the application from behavioural information provided by THALES on the protocol behaviour. FSCOM also modelled intrusive behaviours. FSCOM worked with THALES Communications to define and integrate the control of intrusive nodes within the simulation. Moreover, FSCOM monitored security properties described as TTCS charts, to assume the matching between the TTCN specification and execution traces.
- ❖ **Institut Telecom :** The contribution of Institut telecom is apart from the work of tools interconnection and focused on the specification and design at routing level of a distributed IDS on routing protocol.

Further information on results of application of this test for security framework integration may be found in the D5.WP5 deliverable related to the Radio protocol case Study.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 69 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

3.2 TRACE MANAGEMENT PLATFORM FOR RISK-BASED SECURITY TESTING (FHG FOKUS)

Today, increasingly complex systems are developed. Several developers create different parts of a model that represents the system under development. Each developer has specific views on the system with respect to his role in the development process. For instance, the requirement engineer develops the requirement model while the tester creates a test model on basis of the requirement and the system model. There are specific constraints for creating and visualizing these models that are realized by different tools. However, complex systems are not just a bundle of loose models without relationships. For instance, the purpose of test cases in the test model is to test system components in the system model and to validate system requirements. To handle the diverse relationships between these models the concept of traceability has been developed [32] [8]. It was originally established for requirements engineering and is mostly used in safety [9]. Traceability defines relationships between different models. Such a relationship consists of at least a tuple of model elements and is called *trace*. For example, a trace can refer to a test case in a test model and a requirement in a requirement model, meaning that the test case “validates” the realization of the requirement. All traces constitute the trace model that can be used by analytic tools to evaluate such “validate” relationships. Such analytic tools can traverse not only a single model but several models that are connected via traces. In Diamonds we have introduced the idea of traceability to support risk-based security testing. The idea behind risk-based security testing is to use artefacts from the risk-assessment to support the security testing process. Thus we are interested in establishing traces from the risk assessment to the testing artefacts. These traces need to be persistent and operational so that we can navigate along the traces and use the traces for calculating the coverage of risk assessment artefacts by testing. Figure 48 provides an overview over the classes of artefacts that are to be related. The traceability links are given in red and the tools (and their roles in the testing process) are outlined by the bubbles with dotted lines.

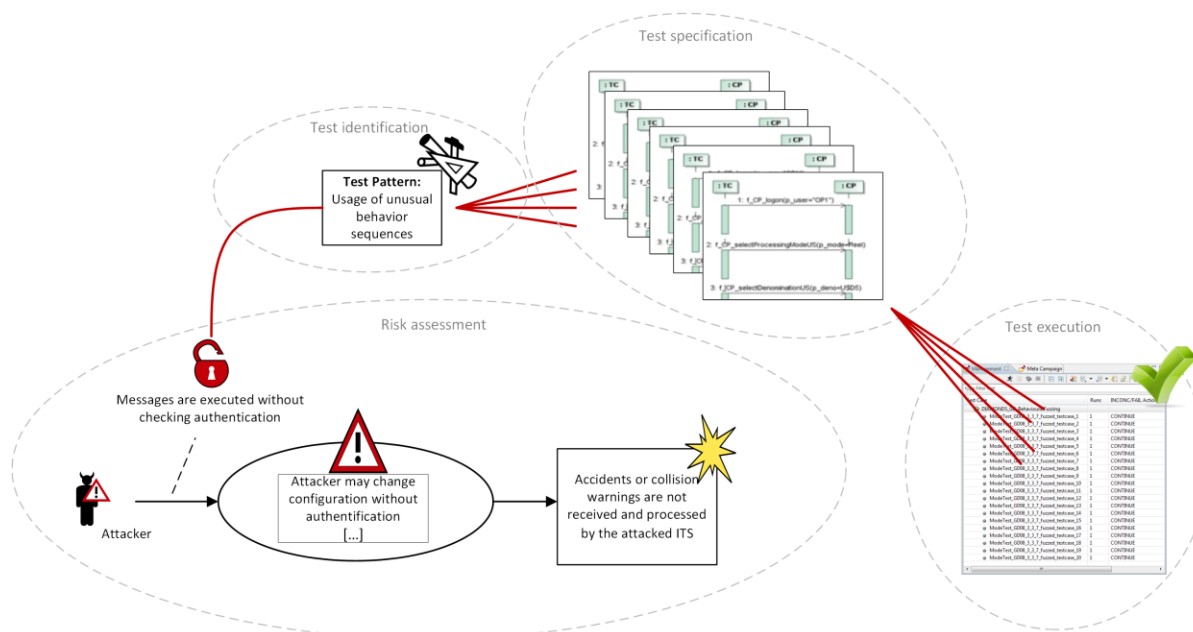



Figure 48: Traceability from risk assessment artefacts to test results

The simplest kind of traceability management is the management of *untyped tracing*. A trace may be created between every kind of element in arbitrary models. It allows easily navigating between models and model elements. It is not suitable for more advanced approaches of analytic tools because of the lack of semantic information about the links and the corresponding traces. A more reasonable kind of tracing is *typed tracing*. That requires the definition of a trace metamodel to restrict the traces to specific element types. This allows analytic tools to use the semantic information given by the type of a trace for evaluating certain aspects of the models and traces between them.

	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 70 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

The information of such a trace metamodel can be used by a lot of services. For example, a coverage analysing service can collect all tested components in a system model by selecting test cases in the test model. Queries on the traced models may be more complex. For example, a likelihood analysing service should mark all possible threats in a risk diagram affecting system elements with a specified likelihood. A traceability supporting tool must meet a set of requirements in order to enable the efficient use by developers. For that purpose, traceability management functions have to be integrated smoothly in the accustomed tool landscape of the developer. It should allow facilely adding new services and also convenient usage of such services.

3.2.1 Description of the Tool

The RiskTest trace management platform is based on a provisional version of the trace management tool CReMa¹. It is integrated in the desktop development environment of the Eclipse workbench and runs with the modelling tools Eclipse configuration in the versions JUNO and INDIGO. The trace management capabilities, i.e. the creation of trace links, the navigation of trace links and the evaluation trace links, are restricted to a set of integrated tools. These tools are the risk modelling tool CORAS², the Eclipse UML modelling editor Papyrus³, the requirement modelling tool ProR, based on the ReqIf model, and the test managing tool TWorkbench.

- CORAS has been used for security risk assessment,
- ProR, has been used for security requirements engineering and as a data base for the security test pattern catalogue,
- Papyrus, has been used for security test specification and modelling
- TWorkbench. has been used for security test execution

3.2.1.1 Basic RiskTest Use Cases

The RiskTest trace management platform has been used to realize two high level uses cases. During the security test development RiskTest has been used to create and maintain trace links between risk assessment artefacts (i.e. vulnerabilities, threat scenarios, and treatment scenarios), test pattern and test specification. The test developer starts with the risk assessment tools CORAS and identifies security test objectives and security testing approaches by relating test pattern from the test pattern library in ProR to risk assessment results in CORAS. Based on these initial assignments the test developer starts specifying the test cases in Papyrus following the ideas given by the test pattern. Each of the test models are again linked to the corresponding test pattern, so that we get a transitive trace link to our initial test basis (i.e. risk assessment results). As DIAMONDS provides model-based testing approaches we normally use test generators to generate the test cases from the test models. The test generator has been integrated in that way, that it adds and updates traceability links from the test models to the generated test cases. Thus, during the whole security test development process the test developer has full control over all dependencies that are made persistent by means of the trace management platform. He can manually navigate along the links and actively switch between the different models, artefacts and perspectives. He can easily control the current status of the test development process by analysing the coverage of the risk assessment elements with test pattern, test models and test cases.

¹ <http://www.guersoy.net/knowledge/crema>

² Lund, M., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach, Springer-Verlag, ISBN 978-3-642-12322-1, 1st Edition 2011

³ <http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=55&L=EN&ITEMID=2>

Tracing Explorer

Weakness Explorer

Model Explorer

Name of Vulnerability	#Testcases	#Pass	#Fail	#None	#Inconclusive	#Error
SQL Injection	5	5	0	0	0	0
Attacker has access to the Router	4	4	0	0	0	0
Messages are executed without checking authentication	28	28	0	0	0	0

Figure 49: Vulnerability coverage by test cases

The second use case addresses the test documentation and test result aggregation. The test results from the test execution are automatically linked with the corresponding test cases in the test specification and modelling tools. Thus at the final end we can provide traceability links that relate test results with the initial risk assessment results. Based on the traces we are able to calculate the coverage of e.g. vulnerabilities with the successful and unsuccessful test from the test execution. Figure 49 shows the aggregated test results for a set of vulnerabilities from the risk analysis.

3.2.1.2 General Concepts and High Level Architecture

The tool can be divided in three layers: The *service layer* contains all services that operate on the models. The *traceability layer* constitutes the core of the trace management tool and implements query handling. The *domain layer* contains all domains and editor tools including their metamodels (see Figure 50). In the following we describe the concept of our trace management framework layer by layer and component by component.

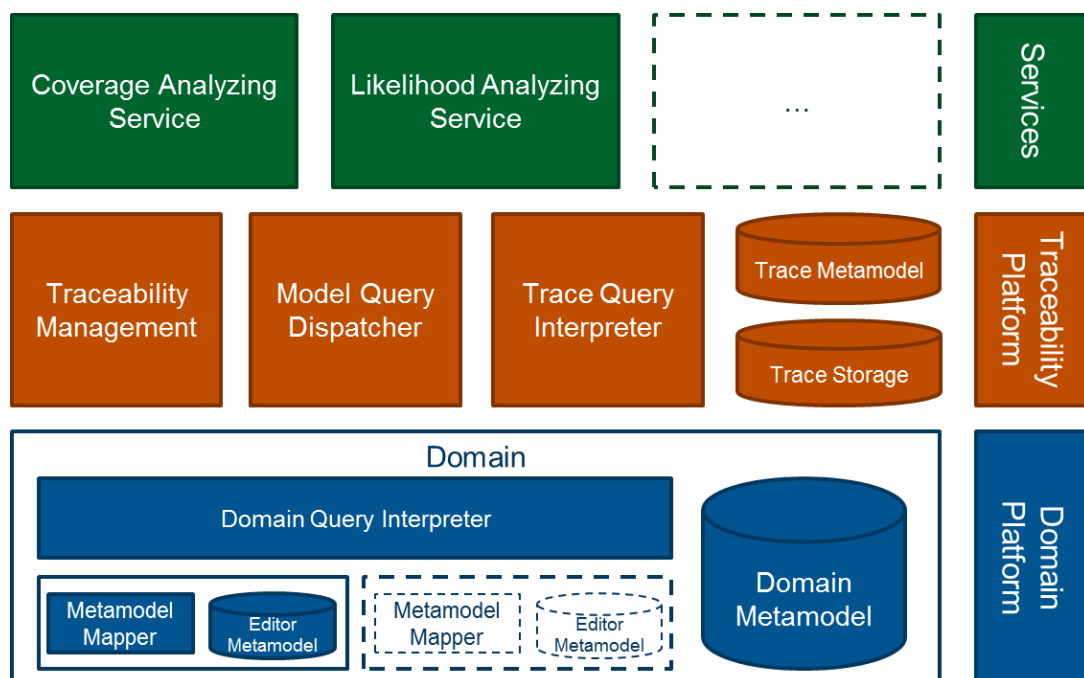



Figure 50: Trace Management Framework in Multi-Layer Diagram

- **Traceability Platform Layer:** All trace handling components are specified in the central layer. These components manage the creation and modification of traces, use the information of the trace metamodel and process query requests.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 72 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

- **Domain Platform Layer:** In order to enable the development of services for the traceability managing tool, it is necessary to provide a unified metamodel of each domain. This unified domain metamodel allows services to access elements from editors that uses different metamodels. Therefore, it is necessary that the editor dependent metamodel is mapped to the unified domain metamodel. This is done with an editor-specific extension to the domain platform layer.
- **Service Layer:** The whole trace management framework is useless without the services. Each service uses parts of information from the models. By using the domain metamodels and the trace metamodel, queries will be defined to get this information. The queries are sent from the services to the query dispatcher and answered with a set of elements and relations. A service is triggered by traceability management component e.g. when a user invokes corresponding functionality within an editor. For example, a service has to analyse the coverage of test cases and the related system components. The results may then be highlighted within the editor.

The Traceability Platform Layer consists of a number of components that each serve to the creation and analysis of trace links.

- **Traceability Management:** All user interactions for editing, viewing and deleting traces are implemented in the trace management component. The traceability editing functions consists of a *trace editor* and the *trace explorer*. The trace editor enables setting the trace type, assigning elements to a trace selected within a model editor, and obtaining other trace information like source model or name of each involved element.
- **Trace Metamodel:** The trace metamodel specifies the different types of traces and between which kind of elements these traces can be created. The traceability management uses the trace metamodel in order to constrain the creation of traces. It prohibits creating traces between elements a trace type is not defined for in the trace metamodel. The trace types as well as domain-specific elements, their attributes and relationships within a domain can be referenced within a query. The query dispatcher uses the trace metamodel to manage the query processing. In order to support new services, it is easy to extend the trace metamodel.
- **Trace Storage:** The trace storage is used to store all traces. This store may be located on a local disk. While this is sufficient in a single-user environment, teams require to access and modify traces from different workstations. For that purpose, the trace metamodel can be stored in a network repository and version control system. The project EMFStore⁴ provides such a distributed storage.
- **Model Query Dispatcher:** This service has to collect information from the models depending on the querying service. A query is solved by stepwise evaluating the constraints by distributing the (partially solved) query to the trace query interpreter and the domain-specific query interpreters. The task of each interpreter is to solve a partial query (called sub-query) by traversing its model using the constraints defined in the query.
- **Trace Query Interpreter:** The query dispatcher sends the sub-queries to the belonging interpreter. The task of the trace query interpreter is to solve the submitted by accessing the actual trace model and interprets the query of the dispatcher in that context. The trace query interpreter can only resolve requests specified for the trace model but cannot process queries specified on elements without a trace relation in the trace meta-model.

The Domain Platform Layer allows services to access elements from the individual tools or editors. This layer adapts the unified domain metamodel to the individual models in of the tools.

- **Domain Metamodel:** The domain metamodel is an abstract model of a set of models belonging to a certain domain. A domain metamodel has two conflicting requirements: While it should be very small to just fit the specific use case it is selected for, it has to provide a set of elements that is large enough to enable a reasonable mapping of an editor's metamodel to the domain metamodel. This can be a difficult task because some editor metamodels are very complex and hence, a mapping to a simple domain metamodel is not trivial. Another problem is to support enough use cases in order to meet the requirements different services developed for the traceability managing platform have. Each service needs other elements from a metamodel. A new service may need a relationship,

⁴ <http://www.eclipse.org/emfstore/>

element that not considered previously in the domain metamodel. For that purpose, the domain metamodel has to be changed. That change may result in ambiguity of relationships that relate to an element that is split by such a change. As a further consequence, the model mapping as well as queries used by a certain service has to be adapted to fit the changed domain metamodel. The obviously contradicting requirements of a small but flexible domain metamodel can be solved by respecting only a certain set of use cases. In the context of the DIAMONDS project, we are developing the trace management framework use case of risk-based security testing.

- **Domain Query Interpreter:** The domain query interpreter is similar to the trace query interpreter. It queries a metamodel for certain elements. The main difference is that a domain metamodel instead of the trace metamodel is used for processing a query. The domain query interpreter interprets the data on the domain metamodel and the mapper can associate the related elements to the editor model. That reduces the complexity of queries by working on domain metamodels that are less complex than editor metamodels because they are designed to fit only certain use cases.
- **Editor Metamodel:** The editor metamodel is the metamodel the editor tool is using. For example, Papyrus⁵ belongs to the system domain and uses UML2 [12] as metamodel. The requirements domain belonging editor ProR uses ReqIF, and the risk model editor CORAS [10] uses the CORAS metamodel. Such models will not be queried directly but by using the mapping to the domain metamodel.
- **Model Mapper:** The model mapper defines a mapping between the domain metamodel and a metamodel of a certain editor.

3.2.2 Application to Case Studies

The trace management framework was developed in the context of the Giesecke & Devrient (G&D) and the Dornier Consulting case studies. In the context of G&D we had defined risk scenarios for the G&D banknote processing machine with the risk assessment tool CORAS, like the authentication bypass of the Message Router or the SQL injection into the database. The feasible vulnerabilities in the risk scenario are related with test pattern and test cases. The trace management framework supports the user with an easy way to create such relationships. The test pattern can be selected from the test pattern catalogue, modelled with the requirement management tool ProR, and can create the relationship to the selected vulnerability. In the same way, a relationship between the tests, like fuzzing test cases generated by the fuzzing test case generator, and the vulnerability or the test pattern can be created.

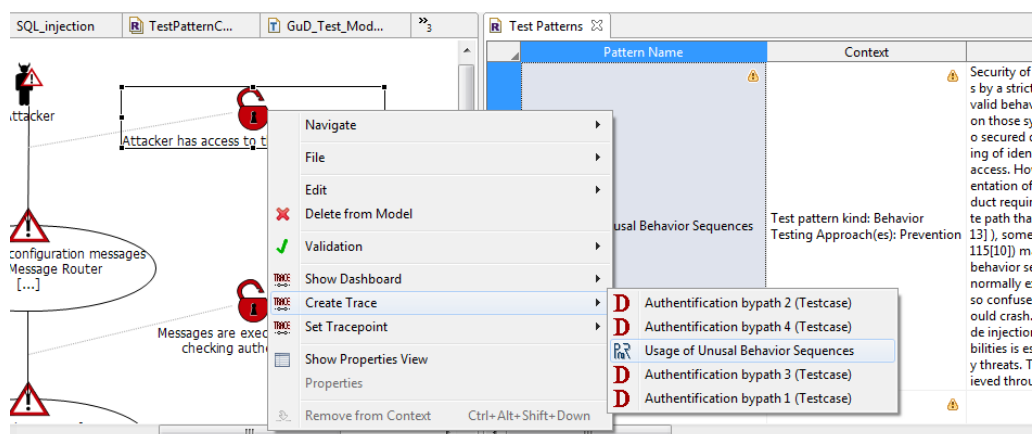


Figure 51 : Example of the Option Create Trace with the Trace Management Tool

In the context of G&D and the Dornier Consulting case study we developed a view for vulnerability coverage. Each linked vulnerability is shown in the Weakness Explorer of the trace management framework (See

⁵ <http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=55&L=EN&ITEMID=2>


	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 74 of 80
		Version: 1.0
		Date : 22.05.2013
		Status : Final Confid : Public

Figure 49). For this, all traced vulnerabilities with directly or transitive relationships to test cases are listed with the results of these test cases.

3.2.3 Advances during DIAMONDS

Starting with a general trace tool concept of CReMa without restrictions and necessary trace information and unspecified tool integration we developed a model based trace platform for risk based security testing. The following improvements have been introduced in Diamonds:

- Dedicated support for risk-based security testing
- Improved user interaction directly from within the tools
- Domain model abstraction layer
- Query interface

3.2.3.1 Dedicated support for risk-based security testing

In the process of risk-based security testing a tool for risk assessment is necessary. We decided to integrate the tool of the SINTEF ICT Company in Norway, CORAS. CORAS based on a model-driven risk analysis method. Focus on the risk assessment, based on eclipse and open source were important criteria for the decision. For the testing part we integrated the test development and execution tool TTworbench of the Testing Tech Company in Germany. The test description language is based on the IEEE standard test language TTCN-3 and the tool also based on eclipse.

In DIAMONDS we developed the test pattern approach to define test categories with examples, instructions and use cases for test cases. The most vulnerability testing test cases can assign to a test category respectively the test case can defined by follow the line of one test pattern. To collect the dedicated test pattern we use the requirement engineering tool ProR, developed by the Formal Mind Company in Germany. The tool is based on the IEEE standardized requirement model ReqIf, open source and integrated in the eclipse workbench.

To evaluate the test results we developed a test aggregation view where all vulnerabilities with related test cases are listed. The results of the test results are listed in a verdict state matrix and can be used to calculate the coverage of vulnerabilities by test cases.

3.2.3.2 Improved user interaction directly from within the tools

To support the developer with an easy interaction we integrated the trace administration directly in the model developing editor views. The user can:

- Create new traces between selected elements in all supported editors.
- Navigate to one traced element from a selected element.
- Delete a trace between the selected element and one of traced element.
- Edit one of the traces of a selected element.

The main advantage to other traceability is the neat integration of the interaction triggers in the user interfaces of original tools, so that the user can develop the model and define traces with the same tool interface.

All traced elements can be administrated with the trace explorer. The explorer enables navigating through the trace model and focusing on traced elements using the corresponding editor view. Also a filter mechanism is enabled to define a filter for hiding non-relevant elements types. An example is depicted in Figure 52, there is showing a screenshot of our current implementation. The traceability management implementation is based on the tool CReMa developed by Itemis in the research project VERDE⁶.

⁶ <http://www.guersoy.net/knowledge/crema>

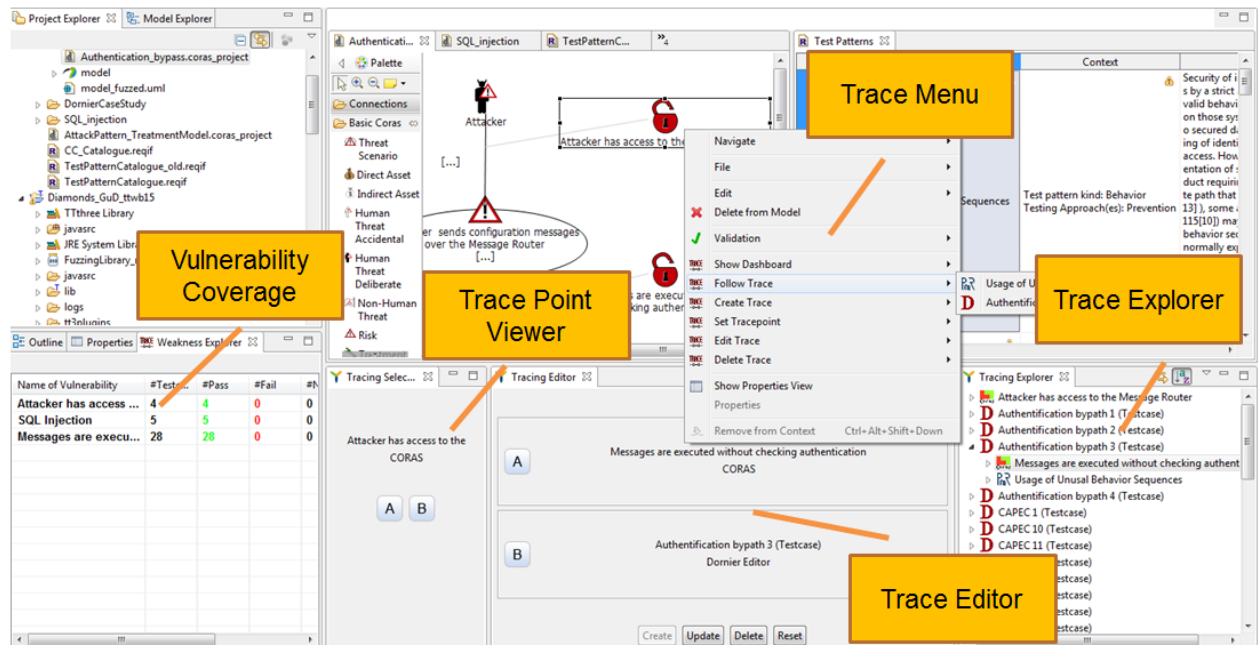


Figure 52: Traceability Management embedded in Eclipse (based on CReMa)

3.2.3.3 Domain model abstraction layer

For a risk driven traceability tool the bulk of tools are the most important part of interest. Each tool has to be integrated into the tool landscape and the interaction between such tools has to be specified. We need tools for risk assessment, for test definition and also for system modelling. We do not want to create a single tool for all modelling parts but to use specific tools for each part of risk driven development to get the benefit of each of specified tool. We defined different domains the tools are related to: The risk domain for developing risk assessment models, the test domain for test cases simulating attacks on detected vulnerabilities, the system domain for specify system components and interface interactions between them, and as the 4th domain the requirement domain to specify requirements for test cases and for a test pattern catalogue.

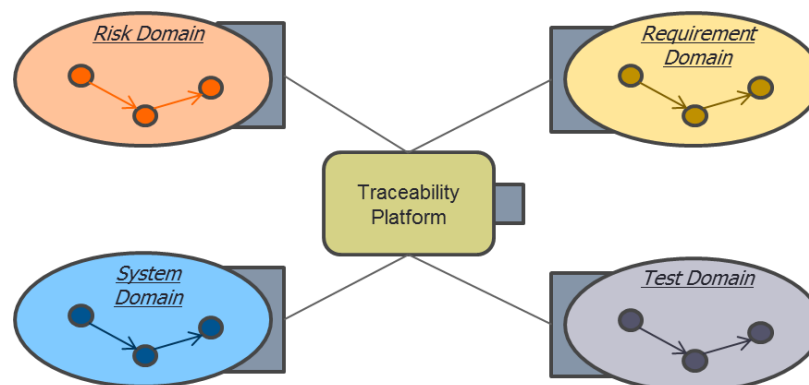



Figure 53: Different Domains in Context of Risk-based Security Testing

To synchronize each tool in a domain, a domain-metamodel is specified.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 76 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

The interactions of the different domains are specified with the trace-metamodel (see Figure 54) and handled by the trace management framework. The trace-metamodel is not only for restrict trace setting, but also to connect the different domains in an information model and to use the metamodel for resolve queries on the whole modelled system analyse. Each tool is integrated in such a specified domain and has only to define a mapping from his owned tool model to the domain model.

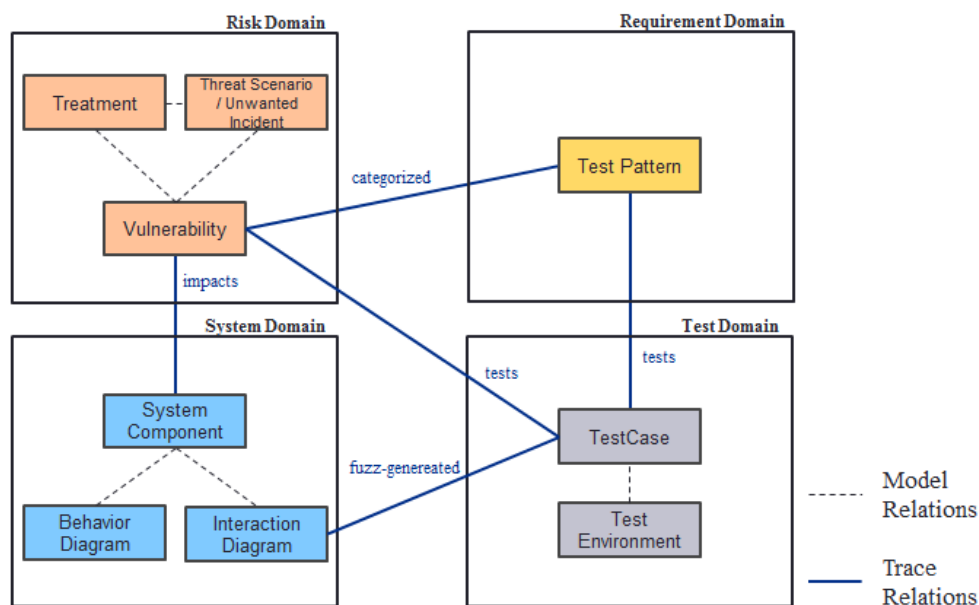


Figure 54: An Example of a Trace Metamodel in Context of Risk-based Security Testing

3.2.3.4 Query interface

To have a way with the set of domains and the traces we had to develop an idea of an interaction mechanism. The mechanism will be complex and needs much effort for optimizing and is only in a draft stage.

The idea is to create an interface for solve queries on the whole model. The service point has to define a query and a query dispatcher has to solve it in a distributed way. That means each domain has to specify an own interpreter for the elements of the owning model. For example only the test domain query interpreter can handle test domain elements and relationships between them.

Additionally we need the trace query interpreter to handle the relationships between the different domains. These relationships are all traces defined by the user.

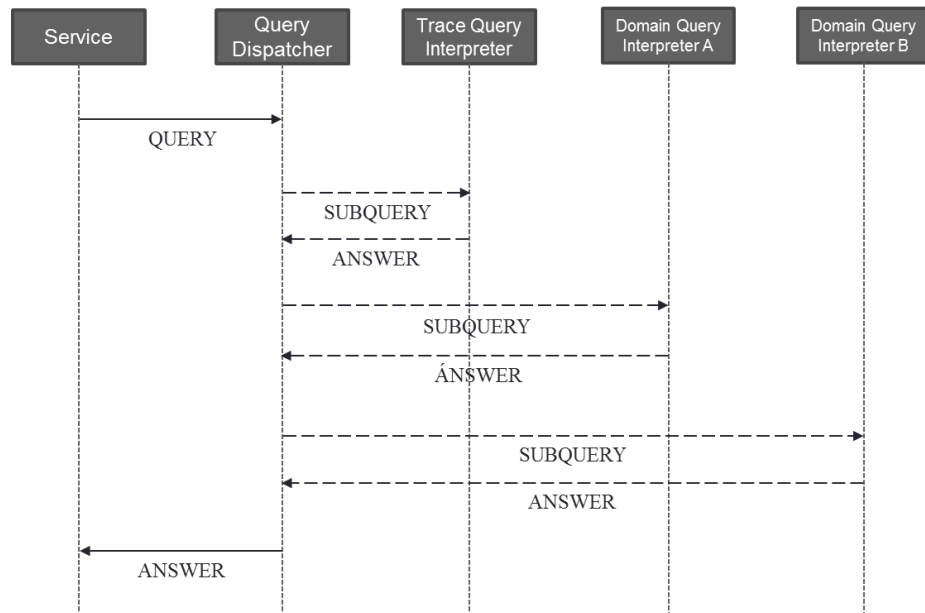



Figure 55: Query Interaction between all Query-related Components


	<p align="center">Final Security Testing Tools</p> <p align="center">Deliverable ID: D5.WP3</p>	Page : 78 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

4. CONCLUSION

In this document the tools that were designed or further developed during DIAMONDS project were described. In addition, the advances particular to each tool were explained along with how they were applied to the use cases of the project. In addition to the ten separate tools described two different integration platforms were described in Section 3.


Advances and tool designs were typically based on advances made in the DIAMONDS work package 2. The advances in methods that were made during the project on the methods are described in the deliverable of that work package.

This deliverable is the final deliverable of the work package 3 of the DIAMONDS project.

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 79 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

REFERENCES

- [1] Diamonds Consortium, section C - Active Security Testing Techniques". In: Final Security Testing Techniques, deliverable D5.WP2, 2013
- [2] Diamonds Consortium, section 2.2 "Vulnerability Directed Input Generation", Active Testing, deliverable D3.WP2, 2012.
- [3] Hex-Rays, "Ida Pro disassembler and debugger", <http://www.hexrays.com/products/ida/index.shtml>
- [4] Zynamics, "BinNavi - binary code reverse engineering tool", <http://www.zynamics.com/binnavi.html>
- [5] "Reil language specification," http://www.zynamics.com/binnavi/manual/html/reil_language.htm
- [6] Selenium - <http://docs.seleniumhq.org/>
- [7] Altheide F., Dörr H., Schürr A.: Requirements to a Framework for sustainable Integration of System Development Tools, in: Stoewer, Garnier (eds.), Proc. of the 3rd European Systems Engineering Conference, AFIS (2002), 53-57
- [8] Freude R, Königs A.: Tool integration with consistency relations and their visualization. In: Proceedings of the ESEC/FSE 2003 workshop on tool integration in system development, Helsinki, September 2003
- [9] Katta, V., Stålhane, T.: A Conceptual Model of Traceability for Safety Systems, proceedings of Springer-Verlag
- [10] Lund, M., Solhaug, B, Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach, Springer-Verlag, ISBN 978-3-642-12322-1, 1st Edition 2011
- [11] Maletic, J., Collard, M.: TQL: A Query Language to Support Traceability, TEFSE '09. ICSE Workshop on, 18. May 2009
- [12] OMG Unified Modeling Language (OMG UML), Superstructure v2.4.1, formal/2011-08-06. OMG specification, OMG (2011)
- [13] Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability, IEEE Transactions on Software Engineering, Vol. 27, No. 1, January 2001
- [14] Spanoudakis, G., Zisman, A.: Software Traceability: A Roadmap, Handbook of Software Engineering and Knowledge Engineering, Vol. III: Recent Advancements, (ed) Chang S. K., World Scientific Publishing Co., ISBN 981-256-273-7, 2005
- [15] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF Eclipse Modeling Framework, Person, ISBN-13: 978-0-321-33188-5, 2nd Edition 2008-12-16
- [16] D1.WP1.FINAL.v11.use-case-overviews-and-requirements.pdf: "SMART CARDS AND THE MOBILE NFC ECOSYSTEM".
- [17] GlobalPlatform Card: "Remote Application Management over HTTP Card Specification v2.2 – Amendment B".
- [18] Diamonds_FPP_v1_3.pdf: " DIAMONDS: Development and Industrial Application of Multi-Domain Security Testing Technologies".
- [19] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".
- [20] ETS 300 406: "Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".
- [21] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".
- [22] Charles Miller: Fuzz By Number. CanSecWest 08. <http://cansecwest.com/csw08/csw08-miller.pdf>
- [23] Jack Koziol: Fuzzers - The ultimate list. InfoSec Institute. <http://www.infosecinstitute.com/blog/2005/12/fuzzers-ultimate-list.html>
- [24] C DeMott, J., Miller. C.: Fuzzing for Software Security Testing and Quality Assurance. Artech House, Boston (2008)
- [25] The Jython Project. <http://www.jython.org/>

	<p style="text-align: center;">Final Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D5.WP3</p>	Page : 80 of 80
		Version: 1.0 Date : 22.05.2013
		Status : Final Confid : Public

- [26] ETSI TS 102 226: "Technical Specification Smart Cards; Remote APDU structure for UICC based applications"
- [27] P. Mouttappa, S. Maag, A. Cavalli, Monitoring based on IOSTS for testing functional and security properties. Application to an Automotive case study, in: 37th International Conference on Computer Software and Applications (COMPSAC'13), 2013.
- [28] F. Lalanne, X. Che, S. Maag, Data-centric property formulation for passive testing of communication protocols, in: Proceedings of the Applied Computing Conference (ACC11), 2011, pp. 176--181.
- [29] Hewlett-Packard, SIPP, Website, [http://sipp.sourceforge.net/\(2004\)](http://sipp.sourceforge.net/(2004)).
- [30] P. Mouttappa, Telecom SudParis, http://www-public.it-sudparis.eu/_mouttapp/TestSym.html
- [31] ETSI ES 202 553: Methods for Testing and Specification (MTS);TPLan: A notation for expressing Test Purposes
- [32] Altheide F., Dörr H., Schürr A.: Requirements to a Framework for sustainable Integration of System Development Tools, in: Stoewer, Garnier (eds.), Proc. of the 3rd European Systems Engineering Conference, AFIS (2002), 53-57