ETSI TR 101 582 V1.1.1 (2014-06)

Methods for Testing and Specification (MTS);
Security Testing;
Case Study Experiences

Reference

DTR/MTS-101582 SecTestCase

Keywords

analysis, security, testing

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**may not**", "**need**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document reports on the application of model-based security testing in different industrial domain. Relevant case studies and their results are described in terms of system under test, applied tool chain, together with an overview of the technical requirements. The case studies were conducted as part of ITEA2 DIAMONDS project (http://www.itea2-diamonds.org/index.html) and SPaCIoS project (http://www.spacios.eu/). The document concentrates on the results and conclusions from this work, giving an insight into how applicable such methods are today for testing and indicating the current strengths and weaknesses.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] AVANTSSAR Deliverable 2.3 (update): "ASLan++ specification and tutorial", 2011.

NOTE: Available at http://www.avantssar.eu.

[i.2] ITEA2 DIAMONDS Deliverable D5.WP2: "Final Security-Testing Techniques", 2013.

[i.3] ITEA2 DIAMONDS Deliverable D5.WP3: "Final Security Testing Tools", 2013.

[i.4] ITEA2 DIAMONDS Deliverable D5.WP4: "DIAMONDS Security Testing Methodology", 2013.

[i.5] SPaCIoS Deliverable 3.3: "SPaCIoS Methodology and technology for vulnerability-driven security testing", 2013.

[i.6] SPaCIoS Deliverable 5.1: "Proof of Concept and Tool Assessment v.1", 2011.

[i.7] SPaCIoS Deliverable 5.2: "Proof of Concept and Tool Assessment v.2", 2012.

[i.8] SPaCIoS Deliverable 5.4: "Final Tool Assessment", 2013.

[i.9] A. Ulrich, E.-H. Alikacem, H. Hallal, and S. Boroday: From scenarios to test implementations via promela: "Testing Software and Systems", pages 236-249, 2010.

[i.10] J. Oudinet, A. Calvi, and M. Büchler: "Evaluation of ASLan mutation operators". In Proceedings of the 7th International Conference on Tests and Proofs. Springer, June 2013. 20 pages.

[i.11] OWASP Cross-Site Request Forgery, 2013.

NOTE: Available at https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[i.12] Erik van Veenendaal: "Test Maturity Model integration".

NOTE: Available at http://www.tmmi.org/pdf/TMMi.Framework.pdf.

[i.13] T. Koomen, M. Pool: "Test process improvement - A practical step-by-step guide to structured testing", Adison Wesley, 1999.

[i.14] Rik Marselis & Ralf van der Ven: "TPI NEXT CLUSTERS FOR CMMI", 2009.

NOTE: Available at http://www.tmap.net/sites/tmap.net/files/attachments/TPI___NEXT_clusters_for_CMMi_0.pdf.

[i.15] ISO 27000:2009(E): "Information technology - Security techniques - Information security management systems - Overview and vocabulary", 2009.

[i.16] ISO 31000:2009(E): "Risk management - Principles and guidelines", 2009.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

asset: anything that has value to the stakeholders (adopted from [i.15])

behavioural fuzzing: security testing technique that creates test procedures by changing a pre-known valid sequence of messages to an invalid sequence by rearranging messages, repeating and dropping them or just changing the type of message

consequence: outcome of an event affecting objectives [i.16]

likelihood: chance of something happening [i.16]

model-based behavioral fuzzing: test technique that combines behavioural fuzzing with model-based testing in that sense, that the pre-known valid sequence valid sequence of messages are given by behavioural models and the test generation is driven by these models

model-based security risk assessment: security risk assessment technique that is conducted with a formalized language for documenting assessment results and a clearly defined process for conducting the assessment

model-based security testing: security testing technique that uses models (e.g. threat models, behavioural models) to automatically or semi-automatically generate accurate and precise security tests

random data fuzzing: test technique that generates input data randomly without any dedicated knowledge on the SUT's protocols

risk: combination of the consequences of an event with respect to an objective and the associated likelihood of occurrence (adapted from [i.16])

risk criterion: term of reference against which the significance of a risk is evaluated [i.16]

risk level: magnitude of a risk or combination of risks, expressed in terms of the combination of consequences and their likelihood [i.16]

security passive testing/ security monitoring: technique of detecting errors, vulnerabilities and security flaws in a system under test (SUT) by observing its behavior (input/output) without interfering with its normal operations (no external stimulations)

security requirement: specification of the required security for the system

security risk: risk caused by a threat exploiting a vulnerability and thereby violating a security requirement

security risk assessment: process for identifying security risks consisting of the following steps: establishing context, security risk identification, security risk estimation, security risk evaluation, and security risk treatment

security risk model: formal or semi-formal specification of threats, vulnerabilities, unwanted incidents as well as their likelihood and consequences

security test case: set preconditions, inputs (including actions, where applicable), and expected results, developed to determine whether the security features of a *test item* have been implemented correctly or to determine whether or not the covered part of the *test item* has vulnerabilities that may harm the availability, confidentiality and integrity of the test item

security test pattern: Collection of best practices/solution for a known security testing problem. It assembles reusable parts of a test plan e.g. the security test design techniques and corresponding test completion criteria, a test coverage item description, applicable test and coverage metrics, estimation on the necessary testing efforts and estimation of test effectiveness with respect to the given problem. Additionally it may contain also test data and specification and assumptions on the test environment as well as testing tool requirements.

static security testing: security testing technique that analyses an application without executing it. One of the main components is code analysis. The code could be source code (in higher languages like C/C++/Java™, etc.) or compiled binary code (in x86 assembly code or Java bytecode, for example)

threat: potential cause of an unwanted incident [i.15]

unwanted incident: event representing a security risk

vulnerability: weakness of an asset or control that can be exploited by a threat [i.15]

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AAT	Abstract Attack Traces
AP	Access Point
API	Application Program Interface
ATM	Asynchronous Transfer Mode
BAM	Business Activity Monitoring
BWCS	Business Worst Case Scenario
CAN	Controller Area Network
CSRF	Cross-Site Request Forgery
DBMS	DataBase Management Systems
DFD	Data-Flow Diagram
DY	Dolev-Yao
EH	eHealth server
EMF	Eclipse Modelling Framework
GSM	General System for Mobile communications
GUI	Graphical User Interface
HCI	Host Controller Interface
HMAC	Hash based Message Authentication Code
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IDE	Integrated Development Environment
IDR	Infobase Document Repository
IDS	Intrusion Detection System
IOSTS	Input-Output Symbolic Transition System
IP	Internet Protocol
ITEA	Information Technology for European Advancement
JSON	JavaScript Object Notation
JSP	Java™ Server Pages
LAN	Local Area Network

LHS	Left-Hand Side
LTL	Linear Temporal Logic
MAC	Media Access Control
MBBF	Model-Based Behavioural Fuzzing
MBST	Model Based Security Testing
MBT	Model-Based Testing
MDD	Model Driven Development
MSC	Message Sequence Charts
NDA	Non Disclosure Agreement
OEM	Original Equipment Manufacturer
OS	Operating System
OSI	Open Service Interconnection
OWASP	Open Web Application Security Project
PCAP	Packet CAPture
PDU	Protocol Data Unit
PIN	Personal Identification Number
PMR	Private Mobile Radio
RLC	Radio Link Control
RSA	Restricted Stock Awards
RSN	Radio Service Network
SAL	Security Attestation Level
SATMC	SAT-based Model-Checker
SCM	Service Control Module
SDU	Service Data Unit
Selenium RC	Selenium Remote Control
SFR	Security Functional Requirement
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SQL	Structured Query Language
STIP	Security Testing Improvement Profile
SUT	System Under Test
TCI	TTCN-3 Control Interface
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TPI	Test Process Improvement^{TM}
TRI	TTCN-3 Runtime Interface
TTCN	Testing and Test Control Notation
TTCS	Testing & Test Control Sequence
TTS	Technical Threat Scenario
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
URL	Unified Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network
XML	eXtended Markup Language
XSS	Cross-Site Scripting

4 Overview on case studies

The present document will provide an overview of the case studies and the final test results from the DIAMONDS project and the SPaCIoS project.

DIAMONDS: The security of a software-intensive system is directly related to the quality of its software? In particular, over 90 % of software security incidents are caused by attackers exploiting known software defects. DIAMONDS addresses this increasing need for systematic security testing methods by developing techniques and tools that can efficiently be used to secure networked applications in different domains. By developing its model-based security testing approaches, extending exiting fuzz testing methodologies introducing the security testing pattern catalogue and a platform for security testing tools, DIAMONDS is building base technologies to offer security tests as a service.

SPaCIoS: State-of-the-art security validation technologies, when used in isolation, do not provide automated support to the discovery of important vulnerabilities and associated exploits that are already plaguing complex web-based security-sensitive applications, and thus severely affect the development of the IoS. Moreover, security validation should be applied not only at production time but also when services are deployed and consumed. Tackling these challenges is the main objective of the SPaCIoS project, which has been laying the technological foundations for a new generation of analysers for automated security validation at service provision and consumption time, thereby significantly improving the security of the IoS. This is being achieved by developing and combining state-of-the-art technologies for penetration testing, security testing, model checking and automatic learning. These are all being integrated into the SPaCIoS Tool, which applies a proof of concept on a set of security testing problem cases drawn from industrial and open-source IoS application scenarios. This will pave the way to transfer project results successfully in industrial practice.

The present document aims to provide insight on these different aspects drawn from experiences in testing within the case studies:

- Different testing techniques
- Initial results
- Metrics, Comparisons
- Contribution
- Exploitation of Case Study results
- Value of DIAMONDS for the case study users

The project results are evaluated in form of Security Testing Improvement Profiles (STIP).

5 Banknote processing case study results

5.1 Case study characterization

This clause provides the revised case study description and requirements from the Giesecke & Devrient case study in the banking sector. It presents the applied security testing approaches as well as results achieved. The case study consists of a banknote processing system that counts and sorts banknotes depending on their currency, denomination, condition and authenticity.

5.1.1 Background

Banknote processing machines are used in central, large and medium banks and also in CITs (cash in transport) and other organizations that handle large amounts of banknotes. These machines are usually configured to work in a network as shown in figure 1. Currency processors, reconciliation stations, vault management systems and control centres are connected on a network either on a LAN or WAN.

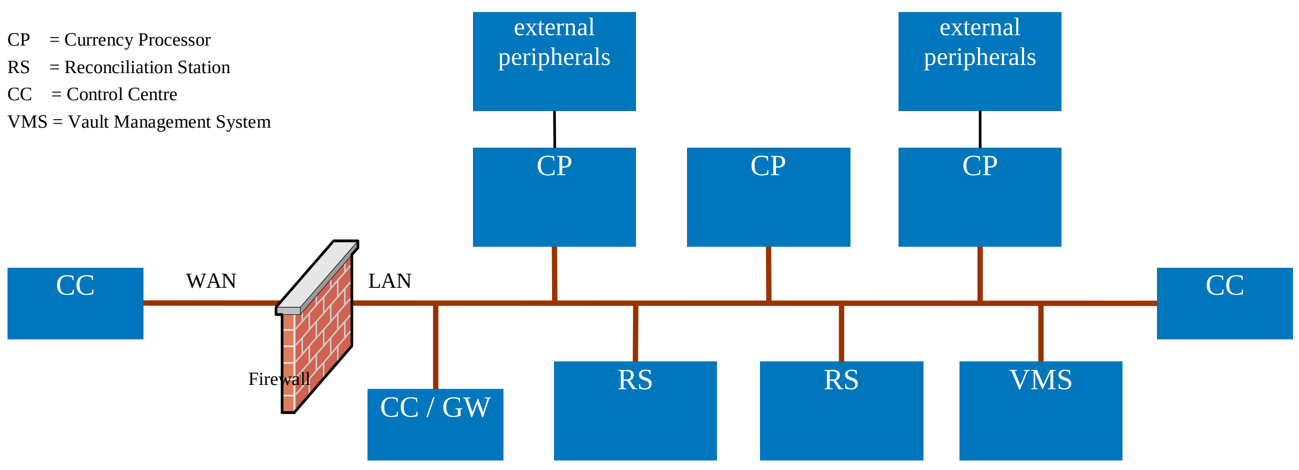


Figure 1: Banknote processing network overview

Different type of information is transferred between network entities. In figure 2 we can see that deposit information is sent to the vault management from the currency processor.

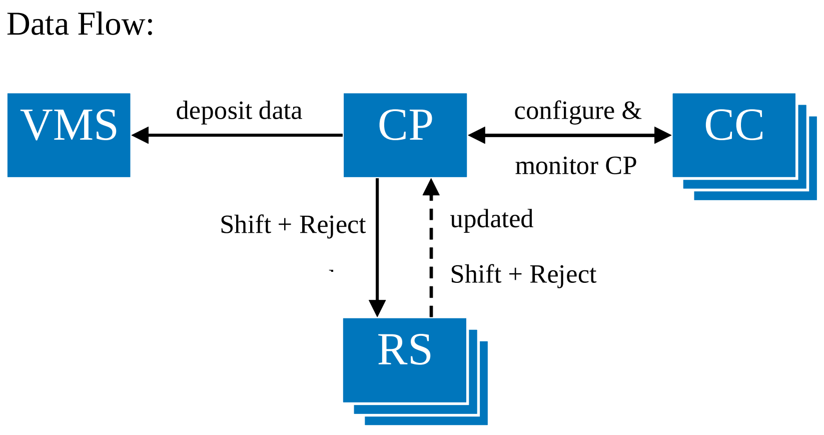


Figure 2: Data flow in processing network

Configuration and monitoring information is exchanged between the currency processor and the control centre. The type of information exchanged requires a high degree of security. Table 1 summarizes the requirements imposed by the Giesecke & Devrient case study.

Table 1: Requirements for banknote processing case study

Req. no	Requirement Type	Description
1	Operating system for test generator (if specific requirements)	Windows XP™/Windows 7
2	Operating system for monitoring tools (if specific requirements)	Windows XP™/Windows 7
3	Operating system for test controller framework (if specific requirements)	Windows XP™/Windows 7
4	Operating system (and platform) for the SUT	Windows XP™/Windows 7
5	List of "physical" interfaces for testing (keyboard, usb, wireless, MAC/Ethernet, ATM, Serial/Parallel and/or communication bus such as TTF/CAN/MOST)	Keyboard and USB provided by the VM abstraction layer, .Net Remoting over Ethernet
6	List of network interfaces/protocols	TCP/IP
7	List of API interfaces/protocols (C, C#, XML/SOAP/REST, SQL, etc.)	.Net remoting over TCP/IP, TTCN-3
8	Programming language used in SUT	C/C++/C# .Net 4.0
9	Existing system/protocol models (languages)	.Net Remoting
10	Requirements for test controller and/or tool interconnection/integration	Test execution should be based on existing TTCN-3 test framework or integrated to work with TTCN-3
11	Requirements for risk modelling	Risk models should enable the communication about threats with non-technical stake holders as well as provide the basis for test
12	Requirements on security testing approaches, such as hacking tools (if available), functional test scripts/plans or fuzzing or other type of negative testing (or other)	Any tool has to provide a TTCN-3 interface, including types, functions, and TCI/TRI implementations
13	Requirements for monitoring techniques such as process/memory monitors, network monitors, security incident monitors or fault detection monitors (or other)	Monitoring tools have not to interfere with the operation of the SUT especially in regards to performance
14	Test environment exists (yes/no)	Yes. A TTCN-3 framework is available
15	Physical access to the test environment is possible to arrange (yes/no)	Possible to arrange
16	Remote access (VPN) to the test environment exists (yes/no)	No
17	Local copy (virtual setup or similar) is available of the test environment exists (yes/no)	Yes
18	NDA required from partners to access the test environment (yes/no)	Yes

5.1.2 System under test

While the banknote processing system consists of several components as depicted in figure 1, the focus of security tests is on the currency processor and the reconciliation station. The currency processor as well as the reconciliation station were provided as virtual machines, where external interfaces are replaced by simulation and were supplemented with snapshots. That allows creating a consistent state of the SUT before executing a test case and is necessary for batch execution of test cases. The test bed at Fraunhofer FOKUS is depicted in figure 3 and consists of the two virtual machines, one for the currency processor and another for the reconciliation station. Windows 7-based host system runs the virtual machines. The main focus of security tests will be the components inside the virtual machines. The available interfaces are the Message Router (.Net Remoting implementation) over LAN, as well as keyboard, USB and other peripherals through the hardware abstraction layer of the virtual machine. There is a database running inside the virtual machine.

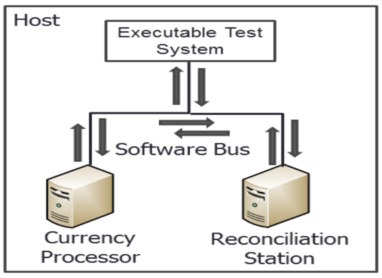


Figure 3: Test bed setup for batch execution

Additionally, the executable test system runs on the host system. It is responsible for executing the test cases, starting the virtual machines with a dedicated snapshot and sending and receiving messages from and to the system under test. The test framework is written in TTCN-3 (Testing and Test Control Notation version 3) and is executed at Fraunhofer FOKUS using a test development and execution environment. In order to run the TTCN-3 test cases using this environment, adapters for encoding and decoding messages were necessary and were adapted from another TTCN-3 test execution environment. By this adaptation, the existing TTCN-3 test framework provided by Giesecke & Devrient was used for performing security tests.

5.1.3 Security risk assessment

The currency processor is exposed to threats which compromise the accounting accuracy. The following high level treatments against the threats were identified:

- **Restricted access to functions:** The access to security functions is restricted to authorized users.
- **Operation system access restriction:** The access to the operation system, i.e. file system, or process monitor is restricted to authorized users.
- **Prevent Admin Hijacking:** Hijacking an administrator account is used to get the privileges of an administrator account as a user that is not assigned to the administrator group.
- **Prevent infiltration/manipulation of software:** Software manipulation can be used to fake data or to provoke errors on the currency processor application.
- **Prevent manipulation of application configuration:** The configuration of the machine should be secured to prevent manipulation otherwise it could be possible to change the classification of banknotes.

The underlying threats were used as starting point for the security risk assessment. A security risk assessment following the CORAS approach was performed and the potential vulnerabilities as well as the consequences of the threats were analysed.

CORAS is a model-based security risk assessment method developed by SINTEF. It provides several kinds of diagrams for different phases of the analysis. E.g. threat diagrams are used to analyse threats to a system by determining potential attackers and vulnerabilities that may be exploited to reach a threat scenario. A threat scenario is a description of how a threat may lead to an unwanted incident by exploiting vulnerabilities. An unwanted incident is the result of reaching one or more threat scenarios by exploiting vulnerabilities and has an impact on an organization. This impact is denoted by assets that are connected with unwanted incidents. Treatment diagrams are the result of analysis of possible mitigations against the analysed vulnerabilities.

A threat to prevent is a manipulation of the configuration of the SUT that may lead to shedding of banknotes which should not be shed. It may result from exploiting an authentication bypass vulnerability. The corresponding risk diagram is depicted in figure 4.

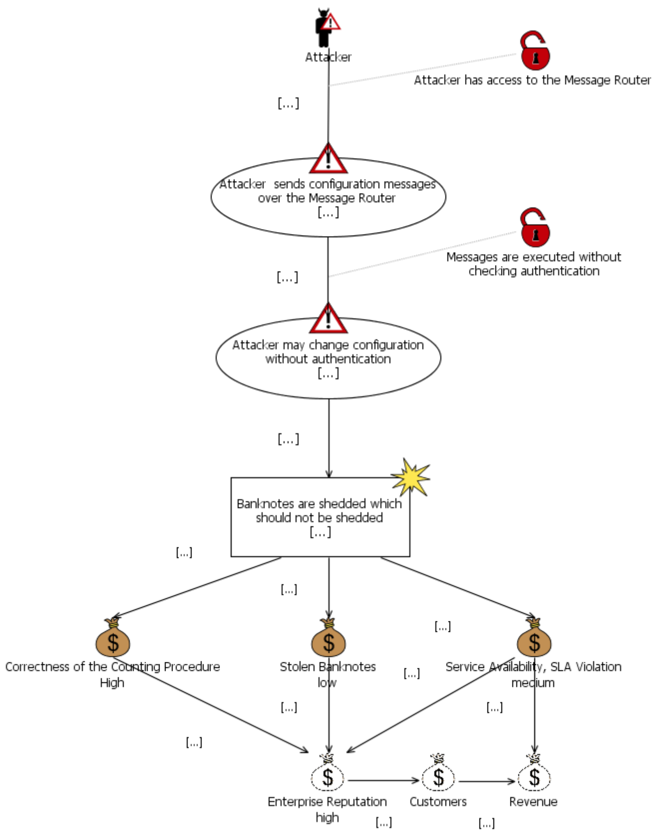

Attacker
Attacker has access to the Message Router
[...]
Attacker sends configuration messages over the Message Router [...]
Messages are executed without checking authentication
[...]
Attacker may change configuration without authentication [...]
[...]
Banknotes are shedded which should not be shedded [...]
[...]
Correctness of the Counting Procedure High
Stolen Banknotes low
Service Availability, SLA Violation medium
Enterprise Reputation high
Customers
Revenue

Figure 4: Risk diagram for authentication bypass

5.2 Security testing approaches

As a result of the security risk assessment, several vulnerabilities were considered that should be tested whether they actually exists within the SUT. In order to generate appropriate tests for these vulnerabilities, security test patterns provide a suitable way to select test generation techniques or test procedures. Those security test patterns constitute the link between security risk assessment and security testing. Two security test patterns are fitting to the results of the security risk assessment.

5.2.1 Detection of vulnerability to injection attacks

The security test pattern is described by table 1a.

Table 1a: Test Pattern "Detection of Vulnerability to Injection Attacks"

Pattern name	Detection of Vulnerability to Injection Attacks
Context	Test pattern kind: Data Testing Approach(es): Prevention
Problem/Goal	Injection attacks (CAPEC 152) represent one of the most frequent security threat scenarios on information systems. They basically consist in an attacker being able to control or disrupt the behaviour of a target through crafted input data submitted using an interface functioning to process data input. To achieve that purpose, the attacker adds elements to the input that are interpreted by the system, causing it to perform unintended and potentially security threatening steps or to enter an unstable state. Although it could never be exhaustive, testing information systems resilience to injection attacks is essential to increase their security confidence level. This pattern addresses methods for achieving that goal.
Solution	Test procedure template: 1) Identify all interfaces of the system under test used to get input with the external world, including the kind of data potentially exchanged through those interfaces. 2) For each of the identified interfaces create an input element that includes code snippets likely to be interpreted by the SUT. For example, if the SUT is web-based, programming languages and other notations frequently used in that domain (JavaScript, JAVA™, etc.) will be used. Similarly, if the SUT involves interaction with a database, notations such as SQL may be used. The additional code snippets should be written in such a way that their interpretation by the SUT would trigger events that could easily be observed (automatically) by the test system. Example of such events include: – Visual events: e.g. a pop-up window on the screen – Recorded events: e.g. an entry in a logging file or similar – Call-back events: e.g. an operation call on an interface provided by the test system, including some details as parameters 3) Use each of the input elements created at step 2 as input on the appropriate SUT interface, and for each of those. – Check that none of the observable events associated to an interpretation of the injected code is triggered
Known uses	
Discussion	The level of test automation for this pattern will mainly depend on the mechanism for submitting input to the SUT and for evaluating potential events triggered by an interpretation of the added probe code.
Related patterns (optional)	• CAPEC 152
References	

The application of this security test pattern leads to data fuzzing in order to generate injection attack strings that may be able to as discussed in the following.

5.2.1.1 Data Fuzzing with TTCN-3

In order to test for the above mentioned vulnerabilities identified during security risk assessment, both well established and new developed methods were applied to the system. Data fuzzing approaches for SQL injection were applied by a new developed fuzz testing extension for TTCN-3. Data fuzzing sends a large number of invalid values to the system under test at certain points within a test case. At these points, the values for fuzzing should be retrieved, for instance by an external function. TTCN-3 external functions retrieve a value from an external function once, buffer this value and use it each time the external function is called. This is not appropriate for fuzzing where another value has to be retrieved and sent to the SUT for each invocation. The fuzz testing extension for TTCN-3 complies with this requirement by requesting values from external fuzz functions each time a value is requested via TTCN-3 `valueof` or `send`. It has been submitted for standardization at ETSI. The fuzzing extension was implemented in the test development and execution tool.

5.2.1.2 TTCN-3

In order to be able to apply this method with TTCN-3, there was a need to extend the standardized language to support fuzz testing. Generally, matching mechanisms are used to replace values of single template fields or to replace even the entire contents of a template. Matching mechanisms may also be used in-line. A new special construct called a `fuzz function` instance can be used like a normal matching mechanism "instead of values" to define the application of a fuzz operator working on a value or a list of values or templates. The definition of such a function is similar to the existing TTCN-3 concept of `external function` with the difference that the call is not processed immediately but is delayed until a specific value is selected via the fuzz operator. For fuzz testing, such function instances can only occur in value templates.

The `fuzz function` instance denotes a set of values from which a single value will be selected in the event of sending or invoking the `valueof()` operation on a template containing that instance. The `fuzz function` may declare formal parameters and should declare a return type. Since the execution time cannot be predicted, only formal `in` parameters are allowed (e.g. no `out` or `inout`). For sending purposes or when used with `valueof()`, `fuzz functions` will return a value.

EXAMPLE 1:

```
fuzz function zf_UnicodeUtf8ThreeCharMutator(
 in template charstring param1) return charstring;

fuzz function zf_RandomSelect(
 in template integer param1) return integer;

template myType myData := {
 field1 := zf_UnicodeUtf8ThreeCharMutator(?),
 field2 := '12AB'O,
 field3 := zf_RandomSelect((1, 2, 3))
}
```

The `fuzz function` instance may also be used instead of an inline template.

EXAMPLE 2:

```
myPort.send(zf_FiniteRandomNumbersMutator(?));
```

To get one concrete value instance out of a fuzzed template the `valueof()` operation can be used. At this time the fuzz function is called and the selected value is stored in the variable `myVar`.

EXAMPLE 3:

```
var myType myVar := valueof(myData)
```

To allow repeatability of fuzzed test cases, an optional seed for the generation of random numbers used to determine random selection will be used. There will be one seed per test component. Two predefined functions will be introduced in TTCN-3 to set the seed and to read the current seed value (which will progress each time a fuzz function instance is evaluated).

EXAMPLE 4:

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

Without a previous initialization a random value will be used as initial seed.

The above declared fuzz function is implemented as a runtime extension and will be triggered by the TTCN-3 Test Control Interface (TCI) instead of (TRI), as external functions, in order to accelerate the generation by avoiding the encoding of the parameters and return values.

More information about the TTCN-3 extension for data fuzzing can be found in the DIAMONDS project deliverable D5.WP3 [i.3].

5.2.1.3 Data Fuzzing Library

In order to retrieve a valuable set of fuzzed values, a fuzzing library was implemented. It provides fuzz testing values from well-established fuzzers. These tools work standalone and thus, cannot be integrated in the existing test execution environment. So the fuzzing library was developed which allows integration in the test execution environment by using XML interface provided by it or by accessing the Java™ code directly. The integration of the fuzzing library to the test development and execution tool was done by implementing external fuzz functions according to the TTCN-3 fuzz testing extension. These external functions are then used within test cases to retrieve fuzz testing values from the library and submit them to the system under test.

To preserve platform independence as achieved within Java™ and to minimize dependencies, the fuzzing operators taken from the fuzzing tools are re-implemented in Java™. This brings benefits for the performance of the library, e.g. since no integration of Python code is required. 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.

In order to receive fuzzed values from the fuzzing library, a request will be submitted to the library. Such a request specifies a type that will be fuzzed, e.g. valid lengths and null termination for a string, as shown in figure 5. 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 request.

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.

```xml
<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 5: Excerpt from an XML request file

Along with the specification of the data type, it is possible to specify which fuzzing heuristics will be used and which valid values will 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 fuzzing library replies to such a request with a response file containing fuzzed values. These values are complemented by information on how they were generated. They are grouped by the employed fuzzing generators for fuzzed values that are generated along the type specification, as well as 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 allows further requests of fuzzed values generated by specific fuzzing operators if a previously generated value revealed some abnormal behaviour of the SUT.

```xml
<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 6: Excerpt from an XML response file

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).

More information on the fuzzing library can be found in the DIAMONDS project deliverable D5.WP3 [i.3].

5.2.2 Usage of unusual behaviour sequences

The vulnerability from the security risk assessment "Messages are executed without checking authentication" constitutes a message sequence that is unusual with respect to normal use of the SUT. Therefore, the security test pattern "Usage of Unusual Behaviour Sequences" is an appropriate starting point for generating test cases that test for this vulnerability.

Table 1b: The test pattern "Usage of Unusual Behaviour Sequences"

Pattern name	Usage of Unusual Behaviour Sequences
Context	Test pattern kind: Behaviour Testing Approach(es): Prevention
Problem/Goal	Security of information systems is ensured in many cases by a strict and clear definition of what constitutes valid behaviour sequences from the security perspective on those systems. For example, in many systems access to secured data is pre-conditioned by a sequence consisting of identification, then authentication and finally access. However, based on vulnerabilities in the implementation of software systems (e.g. in the case of a product requiring authentication, but providing an alternate path that does not require authentication – CWE 288), some attacks (e.g. *Authentication bypass*, CAPEC 115) may be possible by subjecting the system to a behaviour sequence that is different from what would be normally expected. In certain cases, the system may be so confused by the unusual sequence of events that it would crash. Thus potentially making it vulnerable to code injection attacks. Therefore uncovering such vulnerabilities is essential for any system exposed to security threats. This pattern describes how this could be achieved through automated testing
Solution	Test procedure template: 1) Use a specification of the system to clearly identify the normal behaviour sequence it expects in interacting with an external party. If possible, model this behaviour sequence using a notation such as UML, which provides different means for expressing sequenced behaviour, e.g. sequence diagrams or activity diagrams. 2) Run the normal behaviour sequence (from step 1) on the system and check that it meets its basic requirements. 3) From the sequence of step 1, derive a series of new sequences whereby the ordering of events would each time differ from the initial one. 4) Subject the system to each of the new behaviour sequences and for each of those. – Check that the system does not show exceptional behaviour (no live-/deadlock, no crashing, etc.) – Check that no invalid behaviour sequence is successfully executed on the system (e.g. access to secure data without authentication) – Check that the system records any execution of an invalid events sequence (optional)
Known uses	Model-based Behavioural fuzzing of sequence diagrams is an application of this pattern
Discussion	
Related patterns (optional)	
References	CWE 288, CAPEC 115

The application of this security test pattern leads to behavioural fuzzing in order to generate attacks based on invalid message sequences as discussed in the following.

5.2.2.1 Behavioural fuzzing of UML sequence diagrams

A new fuzzing approach was developed for testing against the vulnerability of an authentication bypass. It consists of creating invalid message sequences instead of invalid input data by modifying functional test cases. While existing fuzzing approaches focus on data generation, a few approaches also implicitly or explicitly perform behavioural fuzzing. These approaches generally use context-free grammars or state machines. The behavioural fuzzing approach developed in DIAMONDS uses UML sequence diagrams and modifies these. This allows reusing functional test cases for non-functional security testing. For that purpose, a functional test case from the case study, written in TTCN-3, was modelled as UML sequence diagram and then used for test case generation. The generated test cases aim at revealing authentication bypass vulnerabilities by submitting messages for configuring the banknote processing system before or without authentication.

The fuzzed sequence diagrams are generated as follows: In a first step only one model element at once leading to a fuzzed sequence diagram representing a test case. For instance, an interaction constraint of a combined fragment of kind *alternatives* is negated. This is done for the different model elements and the possibilities to fuzz their behaviour.

In a second step, fuzzing different model elements is combined resulting in fuzzed sequence diagrams each containing at least two fuzzed model elements. For instance if a sequence diagram is fuzzed on the one hand by negating the interaction constraint of an *alternatives* combined fragment and on the other hand by repeating a single message, in the second step a fuzzed sequence diagram is created by combining these two fuzzed model element in a single fuzzed sequence diagram. This is done due to the fact that an invalid sequence containing only one invalid element does not necessarily reveal a vulnerability as showed for data fuzzing.

The third step consists of fuzzing three model elements at once, for example negating the interaction constraint of an *alternative* combined fragment and repeating a message within the first interaction operand. This is done for the same reason as in step 2. The second and the third step are repeated increasing the number of fuzzed model elements in each iteration.

The number of iterations can be stopped for several reasons depending on the capabilities to get feedback from the SUT. The modification of elements of UML sequence diagrams is done by a set of fuzzing operators. Each fuzzing operators performs a single modification of an element in order to generate an invalid message sequence. In the DIAMONDS project, a set of fuzzing operators for messages, combined fragments, their interaction operands and guards as well as for state/duration invariants were developed, e.g. Remove Message, Repeat Message or Move Message, Change Bounds of Loop Combined Fragment, and Negate Guard of an Interaction Operand.

How the approach can be used for testing for an authentication bypass vulnerability is described using a simple example as given in figure 7. Before the machine can be used, a user has to login with valid login data. If the login was successful, he is logged in as an operator and may configure the banknote processing machine in order to count money and at the end the operator logs out. The actions *configure* and *count money* are protected as required by the values of the tag *protected*. The operator is taking the role of the money counter (tag role) and may access the protected actions *configure* and *count money*.

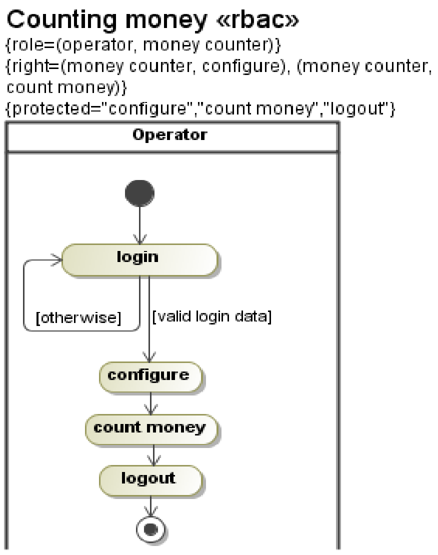

Counting money «rbac»
{role=(operator, money counter)}
{right=(money counter, configure), (money counter, count money)}
{protected="configure","count money","logout"}
Operator
login
[otherwise]
[valid login data]
configure
count money
logout

Figure 7: Simple example of an Activity Diagram with the UMLsec *rbac*

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

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

- The fuzzing operator *Move Message* can now only move the login and logout messages. Login messages can be moved stepwise closer to the logout message to test if the messages appearing after the login can be successfully executed without authentication. Accordingly, the logout message can be moved stepwise closer to the login message to test if the logout is successful and no operations can be executed after a logout.

- *Remove Message* may consider only the login message in order to test if messages that need authentication can be performed without.

- *Repeat Message* may only repeat the login and logout message in order to check if the authentication state remains unchanged by the repeated message.

When considering the example depicted in figure 7, a corresponding test case would look like the one in figure 8 where the information about protected resources, roles and rights are copied from the activity diagram. Additionally, there is one more tag *authentication* with a tuple whose first element contains the information which message performs authentication and which performs a de-authentication.

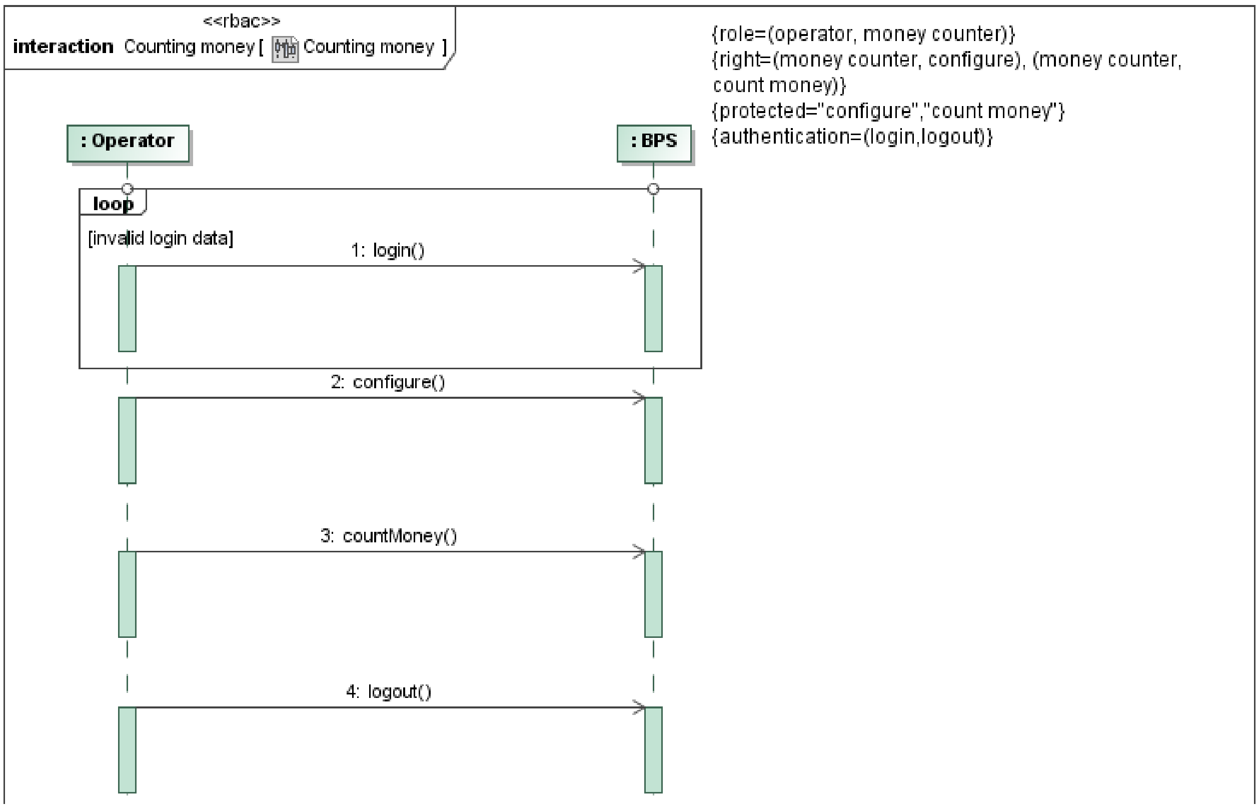

<<rbac>>
interaction Counting money [Counting money]
{role=(operator, money counter)}
{right=(money counter, configure), (money counter, count money)}
{protected="configure","count money"}
{authentication=(login,logout)}
: Operator
: BPS
loop
[invalid login data]
1: login()
2: configure()
3: countMoney()
4: logout()

Figure 8: Test Case derived from the Activity Diagram in figure 7

More information about model-based behavioural fuzzing can be found in the DIAMONDS project deliverable D5.WP2 [i.2].

5.2.2.2 Online model-based behavioural fuzzing

Execution of a single test case takes very long time due to start-up times of the virtual machines and initializing them with a snapshot in order to achieve a consistent state. This step takes several minutes. Because fuzzing approaches generally result in a large number of test cases, this is a serious impediment. To overcome it, a concept called online model-based behavioural fuzzing was conceived that improves runtime efficiency by reducing the number of restarts and initialization of the virtual machines and increases the number of tests executed while the SUT is healthy. Figure 9 illustrates the approach. The current test setup is amended by an online test generator.

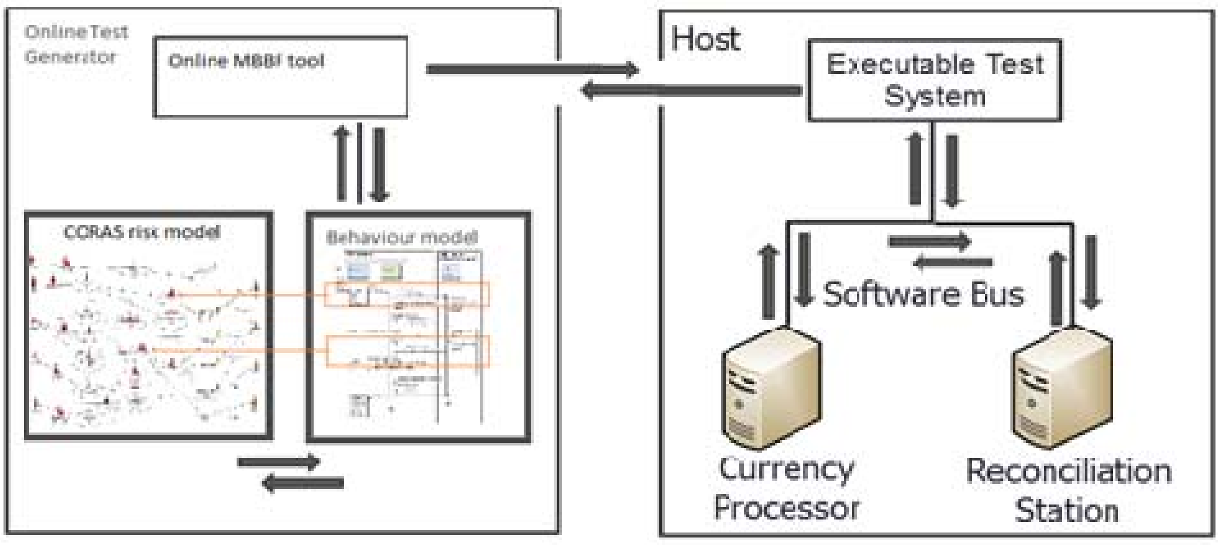


Figure 9: Online Model-based Behaviour Fuzzing Approach

This approach is driven by the desire to apply more fuzzing to interesting behaviour and simultaneously use the test execution time efficiently. The interesting areas in the behaviour model are identified from the riskmodel thus reducing fuzzing to areas where a vulnerability might be located. At the same time more fuzzing operators can be applied while the SUT is healthy. This approach has been implemented and tested using the case study. The test framework needed to be adapted to be able to deal with incorrect sequences which where correctly rejected. The results are very promising because even though no new vulnerabilities were discovered the number of fuzzing operations per test time has increased and heightened the confidence in the implementation of the SUT.

Online model-based behavioural fuzzing is an approach to make the test execution for behavioural fuzz testing more efficient by:

- generating test cases at runtime instead of before execution;
- focusing on interesting regions of a message sequence based on a previously conducted security risk assessment; and
- reducing the test space by integrating already retrieved test results in the test generation process.

More information about model-based behavioural fuzzing can be found in the DIAMONDS project deliverable D5.WP2.

5.3 Results

5.3.1 Requirements coverage

The existing TTCN-3 framework including their test adapters were customized for the test development and execution tool. This was necessary because of subtle differences in the interpretation of the TTCN-3 specification by the different TTCN-3 test execution environments. For this step, test adapters were reused and adapted (applies to requirements 10 and 14 in table 1). The TTCN-3 test framework provides also simple monitoring of the SUT by observing the timing behaviour of and the messages received from the SUT. Thus, it does not interfere with the operation of the SUT (requirement 13).

For enabling fuzzing approaches, a fuzz testing extension for TTCN-3 was developed and implemented for the test development and execution tool. It allows integrating fuzz data generators from the Fuzzing Library with the test development and execution tool and use of them in the TTCN-3 code (requirement 12).

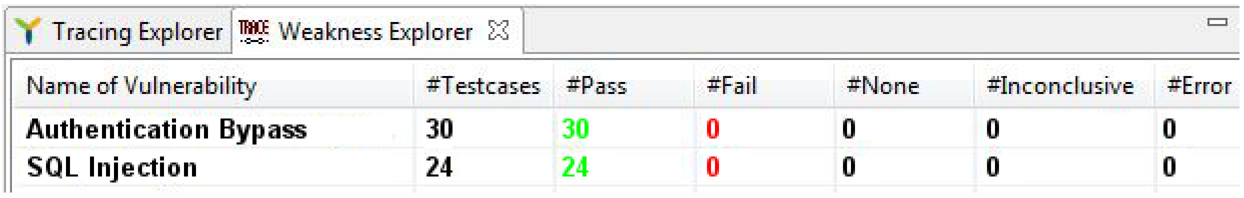
Risk models following the CORAS approach were developed on the basis of the identified threats to prevent. These models describe on one hand vulnerabilities and on the other hand the threat scenarios and unwanted incidents, vulnerabilities it may lead to, as well as the impact on assets. An example of an unwanted incident is "The integrity and confidentiality of the data is compromised" and an example of an asset "Service Availability, SLA Violation" or "revenue" which are understandable also by non-technical stakeholders. The different vulnerabilities, threat scenarios, unwanted incidents and assets are connected through edges and thus, risk models allow understanding the relationships between these elements also for non-technical stakeholders. The identified vulnerabilities in the risk models constitute the basis for the performed test. Therefore, the requirement that risk models should enable the communication with non-technical stakeholders as well as providing a reasonable test basis is fulfilled (requirement 11).

The remaining requirements (1-9) regards the technical basis of the SUT that are fulfilled by providing the SUT as a virtual machine and reusing the test adapter for the communication with the SUT. Requirements 15-18 apply to technical access to the SUT and organizational issues.

5.3.2 Test results

Based on the risk models, 30 behavioural fuzz test cases were executed on the SUT regarding an authentication bypass. Additionally, an initial set of 24 test cases using SQL injection to bypass the authentication were executed. No security-related issues were found. Considering the domain of the case study, banking, this is not surprising because it requires a much higher level for security resulting in a more secure development process than for other domains.

For measuring the coverage of risks by test cases, it was used an integration platform "Trace Management Platform for Risk-Based Security Testing" developed during the DIAMONDS project. The integration platform integrates all used tools, thus it allows for creating links called traces between the different artefacts of risk models, system model elements, security test patterns and modelled test cases as well as TTCN-3 code for test cases. Additionally, it allows for tracing back the test verdicts to the risks. Figure 10 shows how this looks like for the Giesecke & Devrient case study. It shows in each line for a vulnerability from the risk model the test verdict of the test cases that are linked back to the vulnerabilities from the risk model.



Name of Vulnerability	#Testcases	#Pass	#Fail	#None	#Inconclusive	#Error
Authentication Bypass	30	30	0	0	0	0
SQL Injection	24	24	0	0	0	0

Figure 10: Tabular overview of test execution results and vulnerabilities from the risk model

While the initial traces between vulnerabilities from the risk model, the behavioural model of the SUT and the chosen security test patterns has to be created manually, the most traces that results from test case generation and execution are generated automatically. This allows a semi-automatic measurement of risk coverage.

By executing these test cases, the risk of an authentication bypass using behavioural means was covered by applying behavioural fuzzing. Additionally, the risk of an authentication bypass by malicious input data was partially covered. SQL injection is one possibility to pass an authentication without valid authentication data. A query to the database where user input data is used as a parameter may modify the syntax of the query in a way where it e.g. returns a user record independent from the provided password. Other ways to perform an authentication bypass are by adding or modifying the records in the database where knowledge of the database schema is used. Since this knowledge was currently not used, the risk of an authentication bypass was only partially covered by SQL injection. Additionally, further manipulation of the authentication mechanism may be possible. Therefore, SQL injection is only one possibility of an authentication bypass using malicious input data. Likewise, the risk of database manipulation by SQL injection is currently partially covered because of the lack of database schema knowledge.

The Online MBBF approach has been implemented and tested using the case study. The test framework needed to be adapted to be able to deal with incorrect sequences which where correctly rejected. The results are very promising because even though no new vulnerabilities were discovered the number of fuzzing operations per test time has increased and heightened the confidence in the implementation of the SUT.

5.4 Summary and conclusion

Starting the security tests with the development of a risk model to visualize and discuss the vulnerabilities, threats and consequences has proved useful and will be adopted by the standard development process.

The applied fuzzing approaches allow reusing of existing, functional test cases to test the non-functional security aspect. This is achieved by using certain inputs (login, manual input of barcodes data that was incorrectly read by the currency processor) of functional test cases for inserting fuzz test data. Fuzz test data could be easily integrated in a test case because a TTCN-3 fuzzing extension allows direct access to the fuzz data generator. The developed behavioural fuzzing approach extends existing functional test cases towards tests of security aspects. Therefore, the applied fuzzing approaches can take advantage of the effort made for functional testing of the SUT and do not require development of new test cases for security testing. In combination with the results of the security risk assessment modelled in the risk diagrams, the focus of the generated security test cases can be narrowed to the identified vulnerabilities and thus, reduce the number of test cases that are necessary to cover these vulnerabilities. This may help saving resources when performing security tests.

6 Banking case study results

6.1 Case study characterization

The aim of the Accurate Equity (formerly known as Norse Solutions) case study was to evaluate a process which combines security risk assessment and security testing when applied to the Norse Options web-portal (which is the software that Accurate Equity provides to its customers). In the case study, Accurate Equity played the role as domain expert for Norse Options (the system under test), while the security risk assessment and security testing was conducted by SINTEF in collaboration with Accurate Equity.

Norse Options is designed to deliver streamlined administration and reporting of all forms of equity based compensation plans in compliance with the prevailing standards and requirements such as:

- Employee Share Ownership Plans (ESOPs)
- Employee Share Saving Plans (ESSPs)
- Employee Share Purchase Plans (ESPPs)
- Employee Stock Options and Warrants
- Synthetic Options
- Stock Appreciation Rights (SARs)
- Restricted Stock Units (RSUs)
- Restricted Stock Awards (RSAs)

The tool has two kinds of primary users: Employees and Administrators. Employees refer to the employees of companies that use it as their accounting system for shared based payment. Employees use the tool through a web-based interface to manage their individual share-based payment. Administrators are typically accounting personnel in companies that use the same tool. Administrators use it to manage the share-based payment on behalf of the company they belong to. The tool is structured into a 4-layered architecture as shown in figure 11.

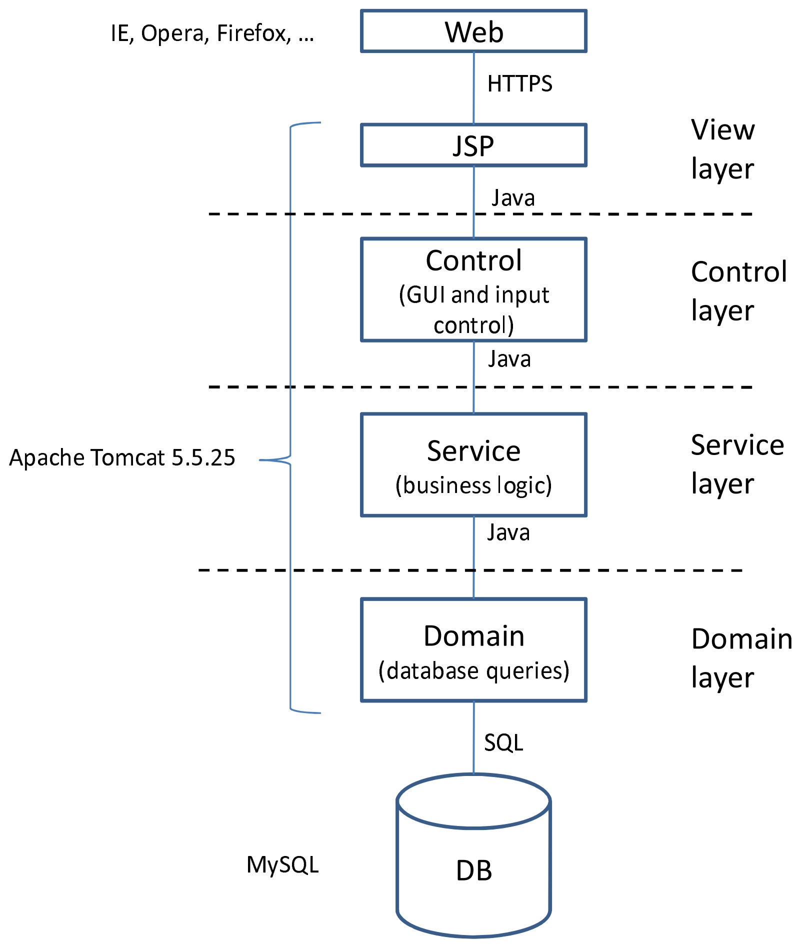


Figure 11: Layered architecture of the System Under Test

The view layer is a thin layer used to render web pages. A web browser presents the HTML pages to the users and communicates using HTTPS to a servlet container. The servlet container translates Java Servlet pages (JSP) into Java™, compiles these and executes the code. The control layer builds the request-specific HTML to be returned to the view layer. As part of this, it may request the service layer for business services to be able to fill the HTML with appropriate data. The control layer also uses the service layer if the request involves updating the data. In addition to generating the HTML, the control layer sanitizes the input and performs access control to prevent unauthorized access to data and functions.

The service layer contains the business logic. This means that all calculations are performed in the service layer. The services are invoked from the control layer and may use the domain layer to persist or retrieve data. Finally, the domain layer manages the persistent data. It transforms requests from the service layer to SQL queries to the database. The details of the security testing and the security risk assessment of the case study are confidential, and so are the security requirements that were addressed. However, in general, one can say that the tool's system handles sensitive financial information and the main overall security requirements were related to protection of the confidentiality and the integrity of the sensitive information handled by the tool.

6.2 Security testing approaches

In this clause, an overview is given of the process for test-based security risk assessment that was followed in the Accurate Equity case study. For a more a description of the current version of this process, see DIAMONDS deliverable D5.WP4 [i.4].

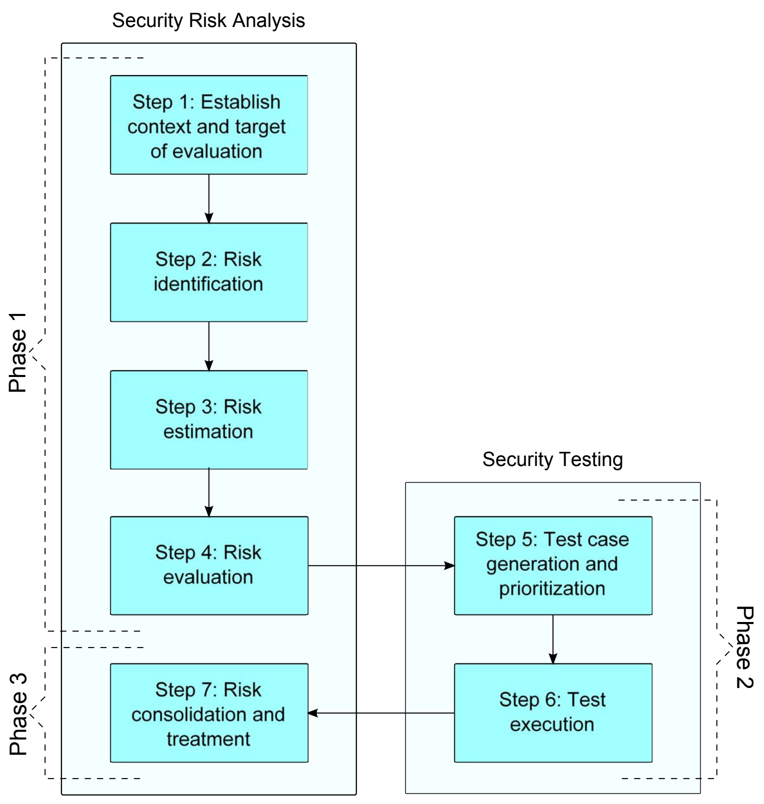
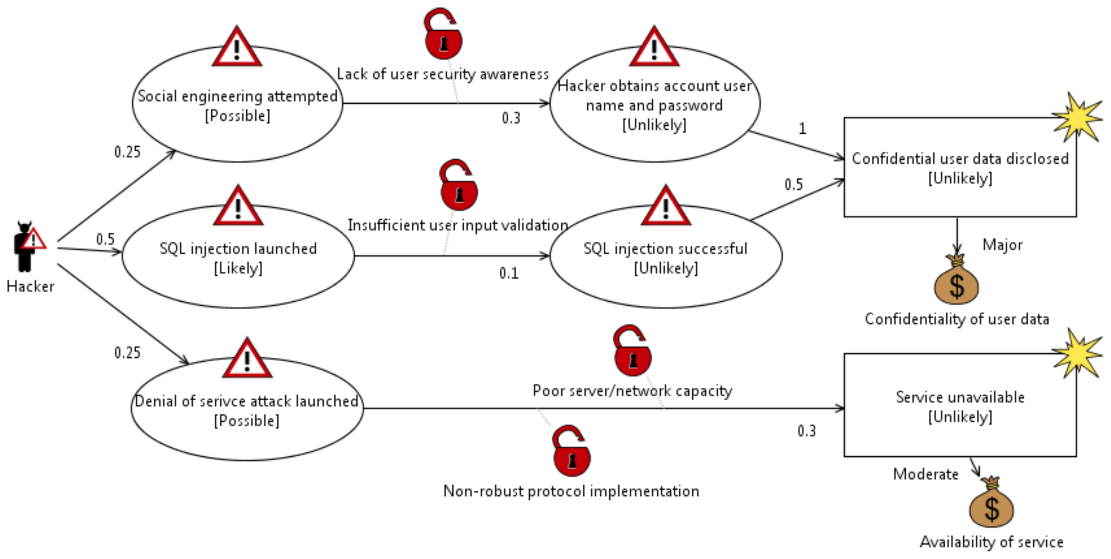

Security Risk Analysis
Step 1: Establish context and target of evaluation
Step 2: Risk identification
Step 3: Risk estimation
Step 4: Risk evaluation
Security Testing
Step 5: Test case generation and prioritization
Step 6: Test execution
Step 7: Risk consolidation and treatment
Phase 1
Phase 2
Phase 3

Figure 12: Overview of the steps in the process

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

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

Phase 1: As indicated by figure 12, the process was two-folded in the sense that it addressed both security risk assessment and security testing. Security risk assessment was conducted using the CORAS approach for model-based security risk assessment.



Hacker
0.25
0.5
0.25
Social engineering attempted [Possible]
SQL injection launched [Likely]
Denial of serivce attack launched [Possible]
Lack of user security awareness
0.3
Insufficient user input validation
0.1
Poor server/network capacity
Non-robust protocol implementation
Hacker obtains account user name and password [Unlikely]
SQL injection successful [Unlikely]
1
0.5
0.3
Confidential user data disclosed [Unlikely]
Major
Confidentiality of user data
Service unavailable [Unlikely]
Moderate
Availability of service

Figure 13: Example of a risk model

In the CORAS security risk assessment process, results are documented using risk models (an example is shown in figure 13). A CORAS risk model is a directed acyclic graph whose nodes are of one the following kinds:

- **Threat:** A potential cause of an unwanted incident (illustrated by a man with a warning sign in case of a human threat).
- **Threat scenario:** A chain or series of events that is initiated by a threat and that may lead to an unwanted incident (illustrated by ellipses with warning signs).
- **Unwanted incident:** An event that harms or reduces the value of an asset (illustrated by box with a star in the top right corner).
- **Asset:** Something to which a party assigns value and hence for which the party requires protection (illustrated by money bags).

Risks are not explicitly shown in the CORAS model of figure 13. However, in the CORAS methodology, risks correspond to unwanted incidents together with a likelihood value and a consequence value. Hence, the model in figure 13 describes two risks (implicitly). Relations may be of one of the following kinds:

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

In addition, relations may be annotated by a:

- **Vulnerability:** A weakness, flaw or deficiency that opens for, or may be exploited by, a threat to cause harm to or reduce the value of an asset (illustrated by red open locks).

Phases 2: Security testing was carried out in a structured manner by (1) identifying and prioritizing potential test scenarios based on CORAS risk model of Phase 1, (2) selecting the most important test scenarios and refining these into executable test cases, and (3) executing the concrete test cases. The security tests were carried out automatically, semi-automatically and manually. The following tools were used in the case study:

- **A** modelling and development environment that uses the Unified Modelling Language (UML) for designing architecture for C++ and Java 2 Enterprise Edition (J2EE) applications and web services. It is built on the Eclipse open-source software framework and includes capabilities focused on architectural code analysis, C++, and model-driven development (MDD) with the UML.

- **A** test design automation tool that creates tests based on system models (i.e. model based testing).
- **Selenium:** Selenium is a suite of tools specifically for testing web applications. The ones used in this case were Selenium IDE, Selenium Server and Selenium Client Drivers. Selenium IDE is a Firefox plug-in that does record-and-playback of interactions with the browser. The Selenium Server is needed in order to run either Selenium RC style scripts or Remote Selenium WebDriver scripts. The Selenium Client Driver is necessary in order to create scripts that interact with the Selenium Server or create local Selenium WebDriver scripts (e.g. in order to run the scripts directly from Eclipse).
- **OWASP WebScarab:** WebScarab is a framework for analysing applications that communicate using the HTTP and HTTPS protocols. WebScarab has several modes of operation, implemented by a number of plugins. In its most common usage, WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser. WebScarab is able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have passed through WebScarab.
- **Eclipse:** Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.
- **Wireshark:** A tool for capturing and analyzing network traffic supporting numerous communication protocols.

All executed tests were black-box tests, i.e. Norse Options system was tested through its HTTP interface. Part of the security testing that was carried in the case study was model-based. The modelling and development environment was used to build a functional model of the SUT which described typical user operations that could be performed on the client side through a web-browser. Then the test design automation tool was used to generate functional tests that were exported to Java^{TM} code, and implemented security specific tests in Java^{TM} "on top" of the functional tests that were generated. Finally the tests were carried out by using Selenium.

Phase 3: In phase 3, the test results were used to verify the correctness of the risk model (produced in phase 1). In particular, test results taken into account were so that:

- confirmed the presence of a potential vulnerability;
- identified new vulnerabilities that were not previously known;
- were not able to confirm the presence of potential vulnerabilities.

If the test results were found to be in conflict with the risk model, then the risk model was updated to take into account the additional information obtained by testing. For example, if the presence of a potential vulnerability (whose existence or non-existence was unknown before the testing), then this typically resulted in the conditional likelihood value of the relation to which the vulnerability was associated to increase.

6.3 Results

The objective of the evaluation is was assess how useful testing is for gaining confidence in the correctness the risk models produced in the security risk assessment (phase 1 above). To make the evaluation precise, the focus was specifically on the degree to which the testing yielded information that caused us to change the risk model. The overall hypothesis was that:

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

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

The evaluation suggests that the hypothesis is false. In the case study, the risk model had to be updated after testing. In particular many of the likelihood values of the threat scenarios and risk had to be changed. Moreover, the testing uncovered vulnerabilities that would never have been uncovered in the security risk assessment phase (phase 1 above), regardless of the amount of effort spent in this phase. It is then believed that the combination of security risk assessment and testing is useful.

In the following, more detailed information is provided about the difference between the risk model before and after testing.

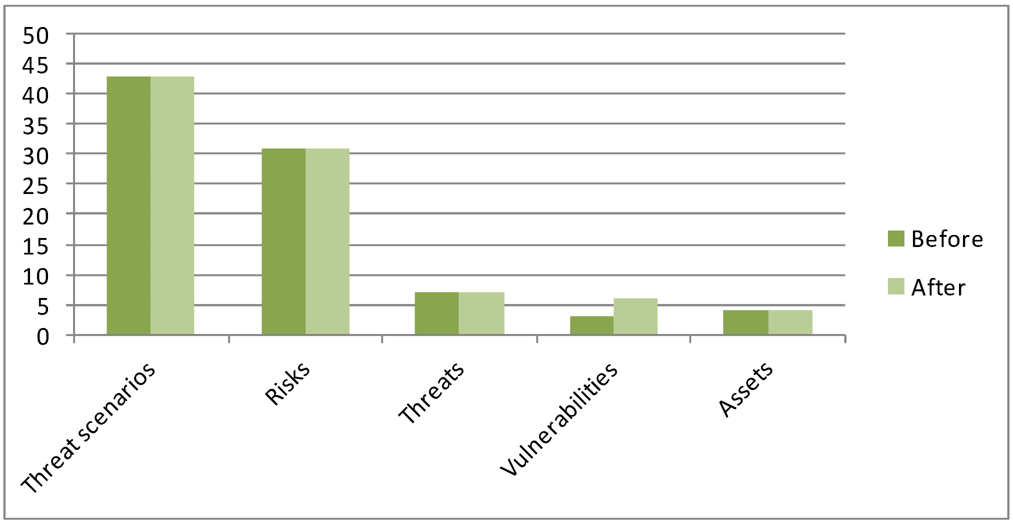


Figure 14: Number of risk model elements before and after testing

Figure 14 shows the number of risk model elements in the risk models before and after testing. Only one element was deleted after testing (a vulnerability), hence the figure shows that four new vulnerabilities were added after testing, but no new threats, threat scenarios, unwanted incidents, or assets were added.

Figure 15 shows the number of threat scenarios and risks that were tested. As can be deduced from the figure, 33 % of the threat scenarios were tested, and 42 % of the risks were tested. Note however, that it was made a distinction between those model elements that were directly tested from those that were not. One can say that a threat scenario *T* was directly tested if *T* was used a basis for deriving tests. A threat scenario or a risk *TR* is indirectly tested if there is a threat scenario or a risk leading up to *TR* that was directly or indirectly tested. From the figure one can see 14 % of the threat scenarios were directly tested, and that none of the risks were directly tested.

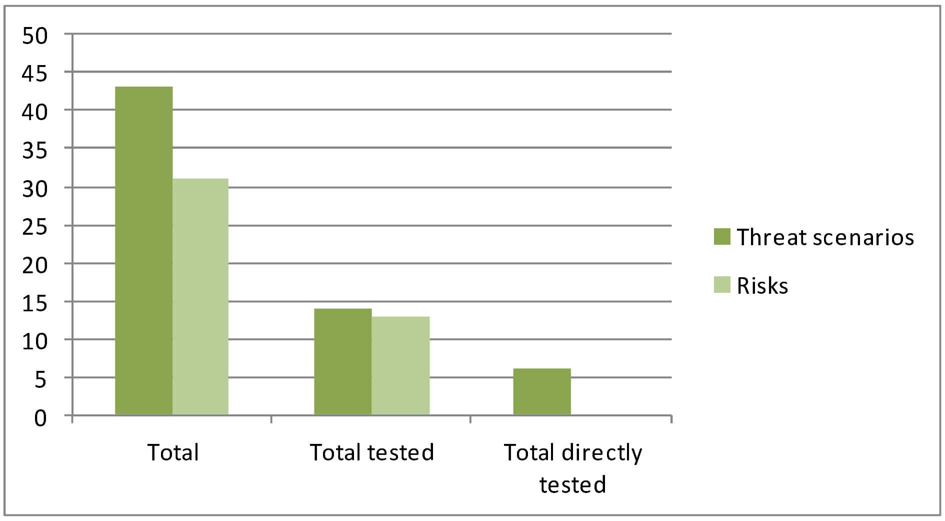


Figure 15: Number of risks and threat scenarios tested and updated

Figure 16 shows the difference between the threat scenarios and risks that *were tested* before and after testing. In the figure, each threat scenario and risk *TR* has a label of the form *i* / *j* which means that *TR* had a likelihood value of *i* before testing, and *j* after testing. The likelihood scale that was used in the case study can be mapped to a number between 1 and 5 where 1 represents the most unlikely value and 5 represents the most likely value. All the threat scenarios and risks whose likelihood values were edited after testing are in the figure 16 given a darker colour than those threat scenarios and risks that were not edited. Note that all except one risk element whose likelihood values were edited after testing were estimated to me more likely after testing than before testing.

In figure 16 the threat scenarios that were directly tested are represented by ellipses with a dotted outline; all the other elements of the diagram are indirectly tested. It can be noted that the level of indirection from the directly tested threat scenarios to the risks is quite large.

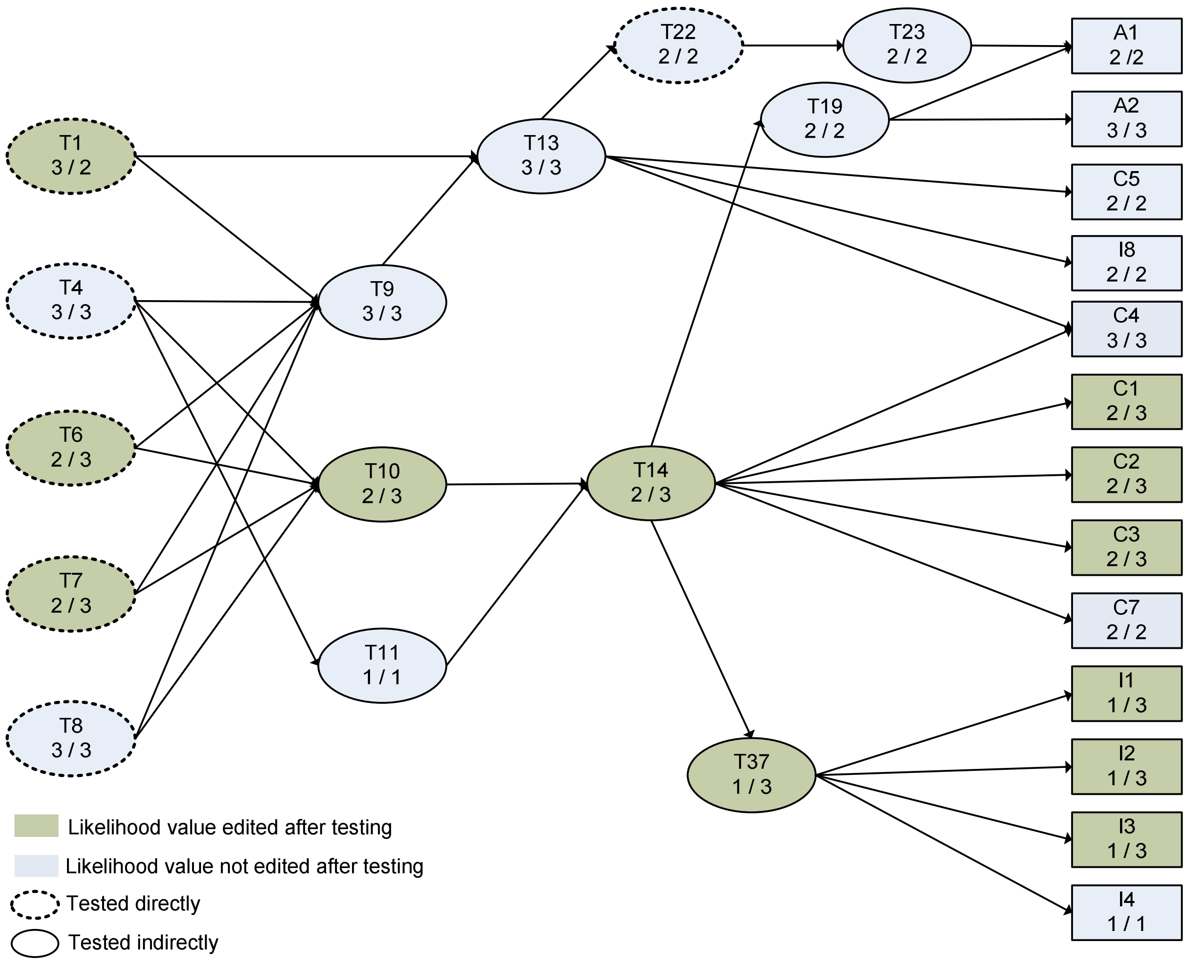


Figure 16: Difference between risk models before and after testing

Based on the previous discussion and the numbers in figure 14, it is noticeable that new vulnerabilities were added to the risk model after testing, and that no other kinds of risk elements were added. Why did the testing only yield new information about the vulnerabilities? The main reason for this is that the tests were designed from the threat scenarios. The threat scenario would typically describe some kind of security attack and the purpose of the tests were to investigate whether the system had some vulnerability that could be exploited by the attack. In other words, the tests were designed to uncover vulnerabilities; not unknown assets, threats, threat scenarios, or risks. These elements were instead part of the context in which the testing was performed.

It is believed that this result is generalizable, i.e. if the process were to be applied in another case study in the future, then the testing will most likely lead to the identification of new vulnerabilities, but not any other kinds of risk model elements. It is worth noting that vulnerabilities uncovered by testing in the case study could never have been uncovered if a security risk assessment had been performed alone (without doing the testing), regardless of the amount of effort spent. This is because the testing uncovered issues which only appeared in extremely specific circumstances which could not have been reproduced without execution the system under analysis. As discussed in the previous clause, the testing resulted in the deletion of exactly one risk element - a vulnerability. The reason why a vulnerability was deleted after testing was that the testing provided evidence that a potential vulnerability identified in the security risk assessment phase was actually not present in the system. This led us to remove the vulnerability from the risk model.

It is also believed that in general, testing can result in the deletion of vulnerabilities, since the tests can be designed to check whether a vulnerability is actually present in the system or not. However, it is unlikely that the test results will lead to the deletion of any other kinds of risk model elements. As documented figure 16, 11 % of the threat scenarios and 13 % of the risks were edited after testing. Moreover, only likelihood values were edited after testing.

For all risk elements that were edited (with the exception of one), the likelihood value was increased after testing, i.e. the risk element was believed to be more likely after testing than before testing. The reason for this was that the testing uncovered vulnerabilities that were previously unknown, and that led to the belief that certain threat scenarios were more likely to occur than believed before testing. For one of the threat scenarios, the likelihood values were decreased after testing as a result of one vulnerability being deleted.

In general, it is believed that when following the process, the testing may uncover information that may cause the conditional likelihood values of relations to be edited, and this in turn may cause the likelihood values of threat scenarios and unwanted incidents to be edited after testing. However, it is not believed that the testing can yield information about the consequence value of risks.

6.4 Summary and conclusion

The partners involved in the case study have gained important practical experience in applying a process which combines security risk assessment and testing. As a result of the case study, Accurate Equity has increased its awareness about security in their application, and will continue to use security risk assessment as part of their business process also after the completion of the DIAMONDS project. SINTEF, will, based on the experiences from the case study, improve their process for security risk assessment and testing, focusing on improved techniques for test case identification and prioritization based on security risk assessment results.

Based on the results of the evaluation, it is not possible to claim that the process leads to saved resources. This will require further evaluation, ideally trying out two different processes on the same systems to compare the effort required. It is however believed that the testing improved the security risk assessment results. This is because the testing uncovered information which resulted in the risk model having to be updated based on this information. Furthermore, the vulnerabilities uncovered by testing in the case study could never have been uncovered if a security risk assessment had been performed alone (without doing the testing), regardless of the amount of effort spent. This is because the testing uncovered issues which only appeared in extremely specific circumstances which could not have been reproduced without execution the system under analysis.

The case study process helped improve the security of the tested software since vulnerabilities were discovered and treated. In general, an improved security risk model will likely lead to a more secure system, as it gives a more accurate description of the vulnerabilities of the system and allows for appropriate mitigations/treatments to be identified.

7 Radio case study results

7.1 Case study characterization

7.1.1 Context of Mobile ad-hoc networks

The Radio Protocol Case study is based on ad hoc Radio Network. These protocols cover the following particularities:

- Decentralized mesh network. No Base Station: One major challenge of these networks is to be independent to any infrastructure between the network units. Each node (also called radio unit) has the capability to integrate an existing network with self-discovering of the nodes in this neighbourhood and exchange information for the radio resource provision for the exchange of control and data information.

- Automatic network deployment with no initial planning. One other challenge is that the radio units discovered in a network may not be initially known i.e. there may not be initial files of registered node identifier to authenticate or not the access to the network.

- Network continuity whatever may be the stations in the network. A third particularity is that the communication is done by nodes relaying between the initial and final nodes. These radio nodes implement a dynamic routing protocol able to interconnect different nodes of a network and allocate the radio resources to establish an end-to-end traffic communication points.

- "On the move" automatic network re-organization and operation. These radio nodes being mobile, the dynamic routes may be cut due to the disappearance of radio nodes, the radio resources (as the bandwidth allocated for the specific neighbour) may be managed with the respect of the number of neighbour and routes established at a particular instant of the network life.
- End-to-end heterogeneous user services transmission: voice, messages. Different kinds of user services have to be provided by the network with different requirements in terms of Quality of Service (QoS) as throughput, latency, link stability. This information has to be taken in account in the link establishment.

Due to their infrastructure less and auto-adaptation particularities, these networks are well fit to be deployed on harsh environments, and used in the domain of Private Mobile Radio (PMR).

Radio units of an ad-hoc network are interacting. Each radio unit manages the allocation of its radio resources with respect to the nodes reachable on its neighbourhood and creation of traffic routes. As this allocation management is dynamic, and offers the capability to a new node to access to the network, the network is particularly sensitive to a node intrusion.

7.1.2 Status of the test of security testing at the beginning of the project

Until recently, the security analysis of such radio equipment were mainly delegated on communication ciphering components of the application, dependent on particular values (seeds, keys) manually injected by the user for a specific mission. These security protections are still used for the protection against threats and malfunctions. However, Due to more and more nominal protection, these specific trusted components have to be completed with a more global vulnerability analysis. This analysis explicitly lists the set of security threats to be faced, but does not assume the capability to verify robustness with respect to these identified potential analysis.

The capability to enrich the validation framework at different stages of the process, in order to combine security testing with functional verification as result of the DIAMONDS project, is the major requirement from THALES Communications & Security.

7.1.3 Security testing capabilities targeted

This clause summarizes the list of security testing analysis targeted to be validated at the end of the DIAMONDS project. As shown in figure 17, the scope of vulnerability analysis is focused on Packet Data Units messages transmitted over the air between pair nodes. The information of the messages (and potential threats may be applied on) are the information provided in the header (used to route the traffic) and the data information (traffic, control information) transmitted in the message.

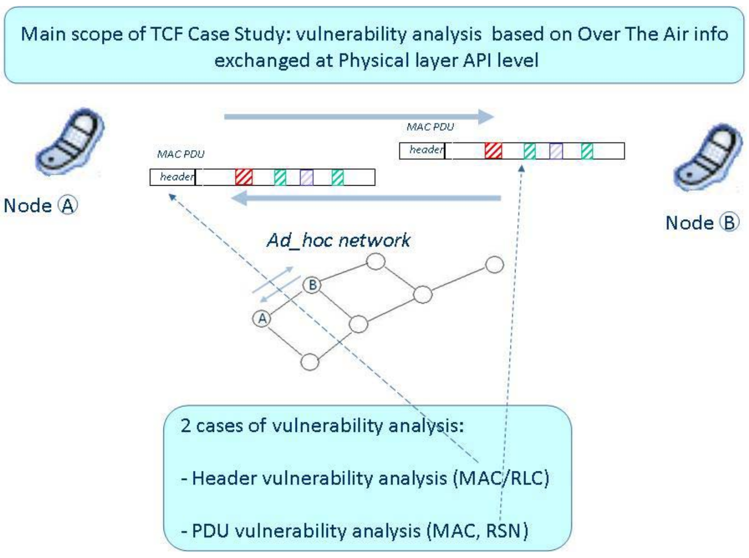

Main scope of TCF Case Study: vulnerability analysis based on Over The Air info exchanged at Physical layer API level
MAC PDU
header
MAC PDU
header
Node Ⓐ
Node Ⓑ
Ad_hoc network
B
A
2 cases of vulnerability analysis:
- Header vulnerability analysis (MAC/RLC)
- PDU vulnerability analysis (MAC, RSN)

Figure 17: Global scope of the capture of PDUs exchanged

The threats identified on the general vulnerability analysis on radio protocols are the following.

7.1.3.1 Frames analysis

The on the fly or post analysis from an intrusive node offers the capability to an intrusive node to capture sequences of messages exchanged between peer entities; and:

- to refine the understanding of the protocol behaviour and detect weaknesses for a second disturbance phase;
- to collect the traffic information exchanged.

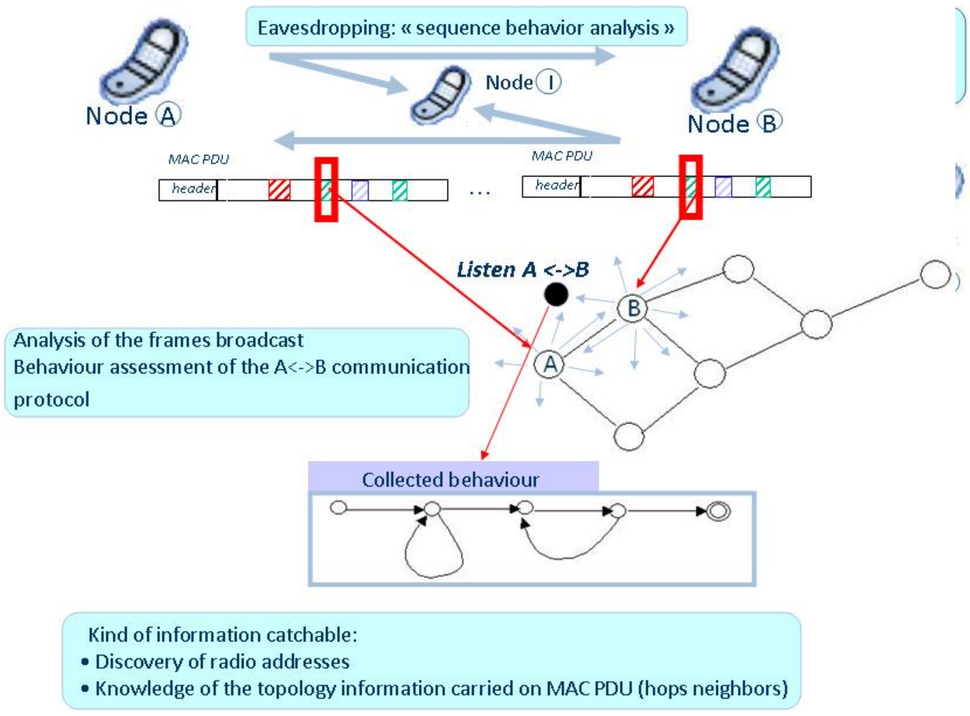

Eavesdropping: « sequence behavior analysis »
Node A
Node I
Node B
MAC PDU
header
MAC PDU
header
Listen A <->B
B
A
Analysis of the frames broadcast
Behaviour assessment of the A<->B communication protocol
Collected behaviour
Kind of information catchable:
• Discovery of radio addresses
• Knowledge of the topology information carried on MAC PDU (hops neighbors)

Figure 18: Listening and analysis of the data exchanged

Example on the radio protocol:

The sharing of the radio resources is processed with respect to a timing decomposition in timing frames of N slots. The first slot of a frame is a service slot used to emit/listen information from nodes to be known from the neighbourhood. The other slots are dedicated to traffic, and contain traffic data from the users and also some information to maintain the topology. The information of the allocation of some slots in the frame for a particular peer-to-peer communication is done on these slots. This kind of information may be captured from an intruder node. It then will know which particular slots are involved in a peer-to-peer communication.

7.1.3.2 Data alteration

One potential threat is, for a node acquiring a more or less fine knowledge of the protocol of the data exchanged, to send a message instead of a node in the network, with a modified value of the data transmitted. This modification can be done in randomly modifying the value of fields in the data sent with respect to the data that should be originally sent. This modification can be done in making a modification explicit of the data to force the receipt to behave differently as if it should do if it would have received the correct data. The denial of service presented hereafter may be considered as a particular case of the second case of data alteration. This is illustrated in figure 19.

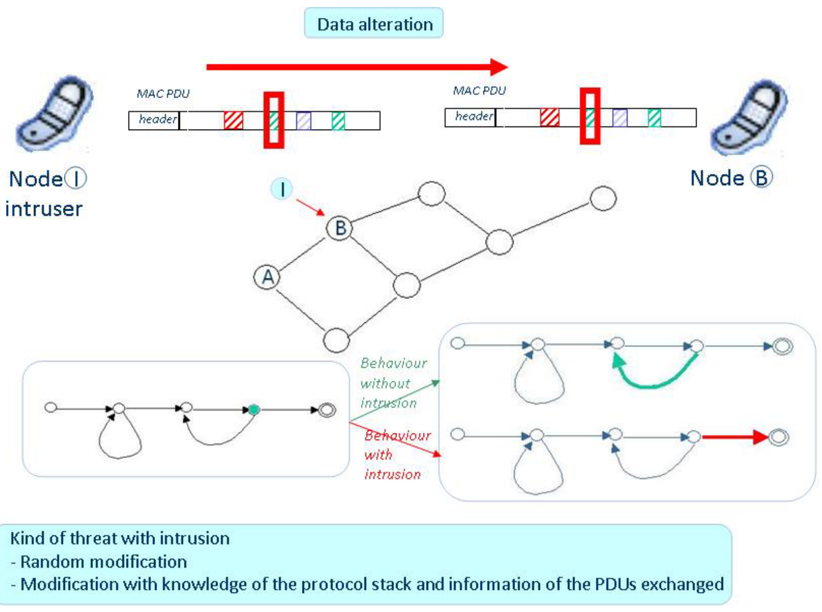

Data alteration
MAC PDU
header
MAC PDU
header
Node I
intruser
Node B
I
B
A
Behaviour without intrusion
Behaviour with intrusion
Kind of threat with intrusion
- Random modification
- Modification with knowledge of the protocol stack and information of the PDUs exchanged

Figure 19: Data alteration

Example on the radio protocol:

A phantom traffic allocation from an intruder node on a particular node with the allocation of a large set of traffic slots will imply the traffic performance of the real communications on this node.

7.1.3.3 Frames replay

Another threat situation, as described in figure 20 is to record a particular sequence of messages between two nodes, with or without modification of this sequence of messages.

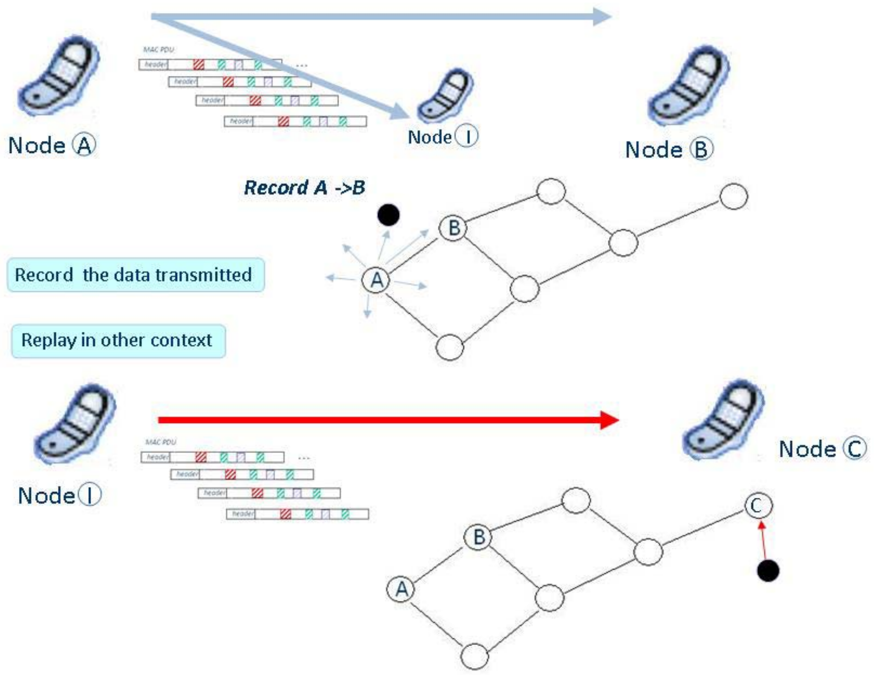


Figure 20: Frames replay

Example on the radio protocol:

The request and suppression of allocation dedicated to a particular node can be replayed in another configuration.

7.1.3.4 Denial of service

The particular situation of the denial of service is the interaction in the communication between peer entities. The intruder node replaces one of the two peer node is acknowledging at its place always positively or negatively, in function of the disturbance expected.

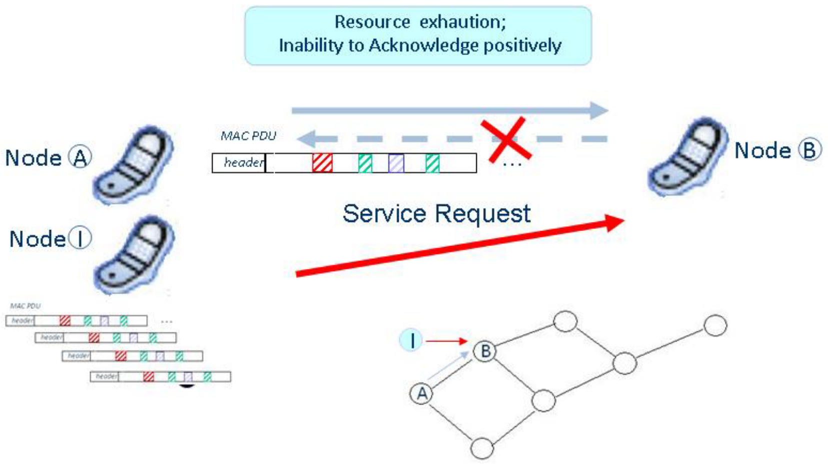


Figure 21: Denial of service

Example on the radio protocol:

As data alteration, the intruder may send traffic slots at the same time than a particular node, this node will then be suppressed from the neighbour allocation tables of its neighbourhood nodes, as the service slots will be jammed.

7.1.3.5 Tampering, malicious code injection

The following kind of threat is the injection of malicious code directly integrated in the protocol stack, for example resulting of a modifying load of new service Over The Air.

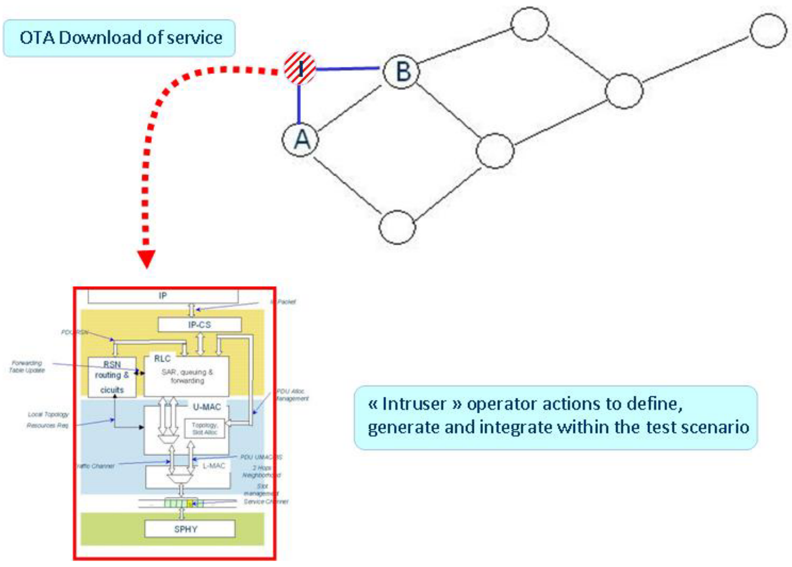

OTA Download of service
I
B
A
IP
IP-CS
RSN routing & circuits
RLC
SAR, queuing & forwarding
U-MAC
L-MAC
SPHY
« Intruser » operator actions to define, generate and integrate within the test scenario

Figure 22: Tampering, malicious code injection

Example on the radio protocol:

More accurate behaviour modification can be applied on this kind of threat.

7.1.3.6 Combination of threats

The threats can be composed as a combination of the different situation described in the precedent clause.

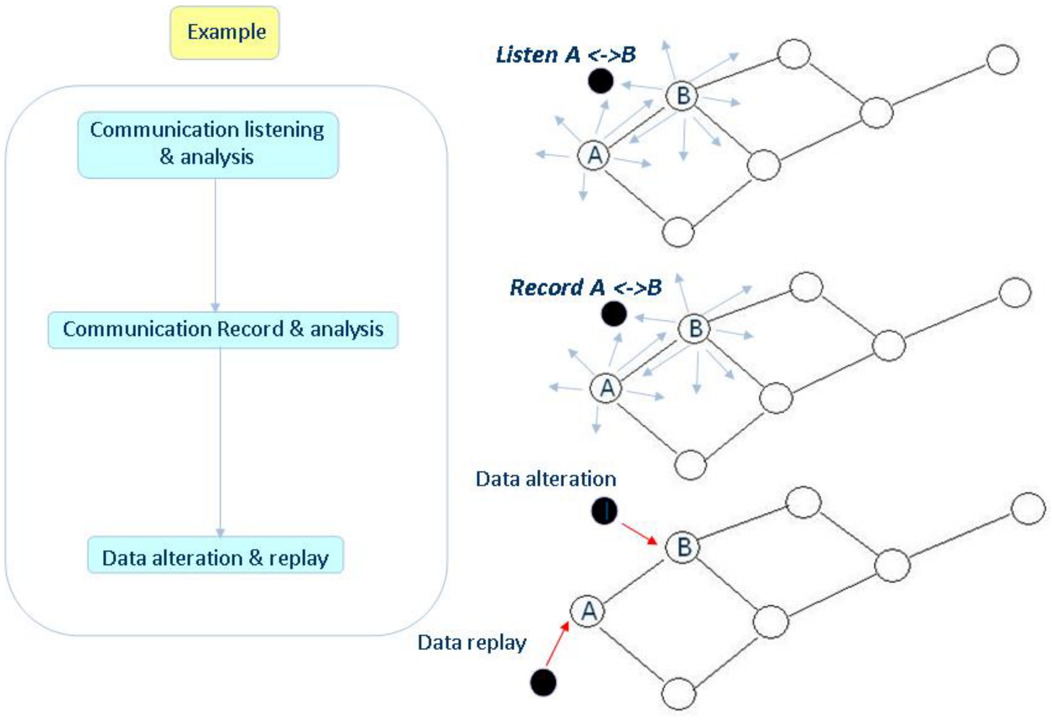


Figure 23: Combination of threats

7.1.4 Description of the use-case

The Open Systems Interconnection model (OSI model) shown in figure 24 is a product of the Open Systems Interconnection effort at the International Organization for Standardization. It is a way of sub-dividing a communications system into smaller parts called layers. A layer provides services to its upper layer while receiving services from the layer below, the data exchanged between layers are called Service Data Unit (SDU). The part of this data to be exchanged over the air between peer-entities is called Packet Data Unit (PDU). A tool, and in particular test tool for the security, validating services provided by a specific layers of a protocol stack may be applied on the monitoring and intrusion on the PDU exchanged at this layer level between pair entities, with the hypothesis the tool will have the capability to catch and extract this specific information at the physical layer the data will be really exchanged over the air.

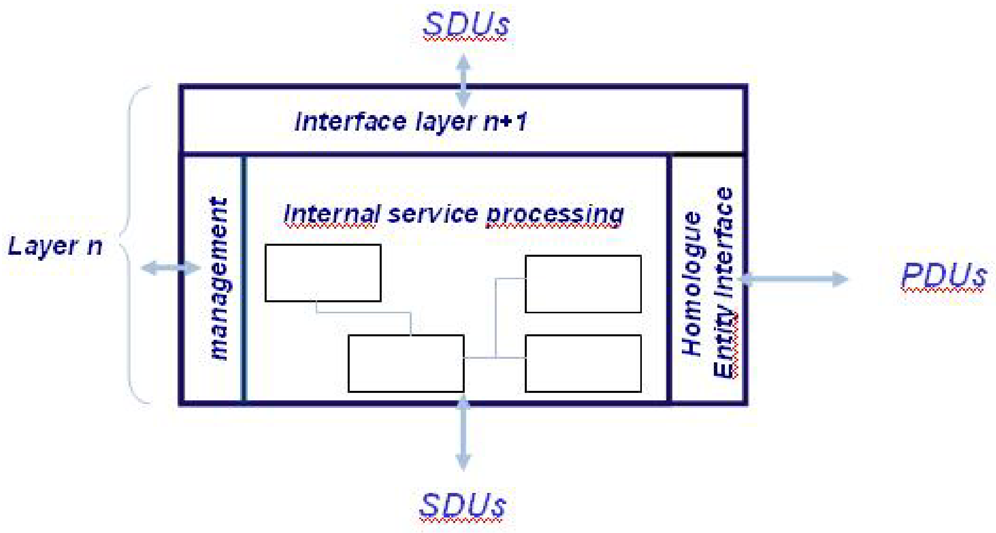


Figure 24: OSI layer decomposition

7.1.4.1 Specific application used as Use Case

The application used as Use Case is a full designed OSI layered ad-hoc protocol application, from Physical to IP Convergence sub-layer application.

- At physical layer: modulation and demodulation, management of the dwell level of the slots of the TDMA and definition of the slots of the TDMA.
- At Medium Access Control (MAC) layer: the management of the slot level of the TDMA cycle, the dynamic allocation of a service slot to each station, the construction of a meshed network (flat), the dynamic allocation of traffic slots to the station, the forwarding of topology information (bidirectional links between stations) to the RSN layer and the separation of the packets of data on the traffic slots.
- At Radio Link Control (RLC) layer, the procedure used to route data within the network with single-station relaying. This routing corresponds to an unconnected mode (hop-by-hop routing. The segmentation/reassembling procedures for the data coming from upper layers to retransmit them to the format of the TDMA slots and the queue management procedures according to the quality of service criteria set by RSN.
- At Radio Service Network (RSN) layer, the route search in the internal addressing plan on RLC request, updating of the local RLC switching information, Quality of Service (QoS) parameters management applicable to RLC queues and local route supervision.
- At IP Convergence Sub-layer (IP-CS), the matching of the destination IP address of the IP data with the MAC address of the network output station, the matching of the user data (IP packets) with the appropriate network service.

7.1.4.2 Specific context of the application of security testing tools

The properties for security testing focused on the neighbourhood management. The management of the 1 and 2 hops neighbourhood detection is processed by a specific service in the MAC layer. This service initializes and updates from PDU exchanged:

- One hop neighbours broadcast channels to emit control signal on lists management.
- Knowledge of the local topology of the One and 2 hops radio node lists.

Information on behaviour and fields of the message exchanged between pair nodes for this neighbourhood management are given in the DIAMONDS deliverables.

7.1.4.3 Specific context of the initial validation framework

The initial validation framework used for the integration of the tool set was an event based simulation (as ns-3 or OMNET). The simulations execute the real C/C++ final encoding, and are used for a first functional validation, and validation of the protocol behaviour of a set of nodes interacting with each other. Scenario files written by hands describe nodes movements and traffic setups. Log files, tagged by time slot the messages have been sent, capture the PDUs and SDUs exchanged between layers, and sent over the air.

7.2 Security testing approaches

7.2.1 General principles of the security testing tools integration

The radio equipment design process, as shown in the following picture, validates at different stages the functional behavior of the radio protocol being designed. At each phase the validation validates requirements by simulation/execution on demonstrator of scenarios. **The security testing approach is to integrate the security testing tools set into the validation framework at each step of the validation framework.**

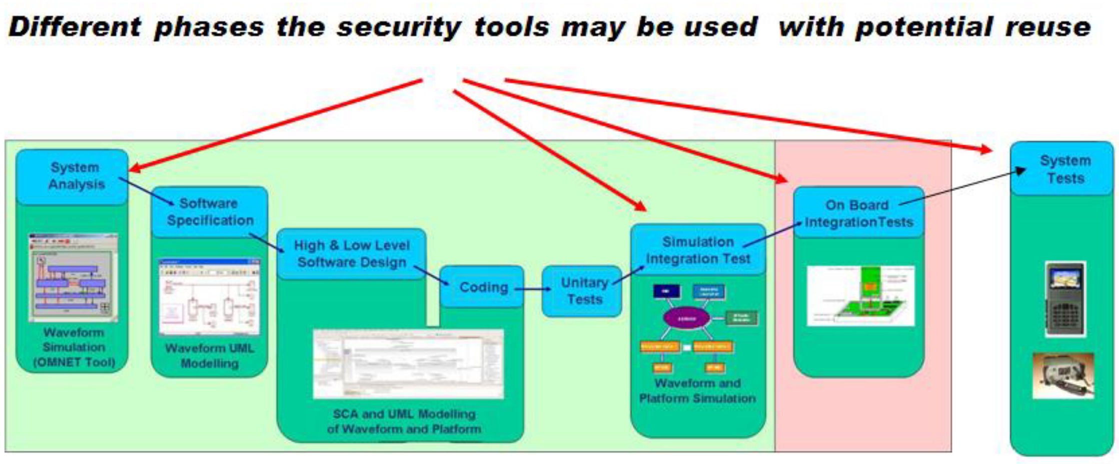

Different phases the security tools may be used with potential reuse
System Analysis
Waveform Simulation (OMNET Tool)
Software Specification
Waveform UML Modelling
High & Low Level Software Design
SCA and UML Modelling of Waveform and Platform
Coding
Unitary Tests
Simulation Integration Test
Waveform and Platform Simulation
On Board IntegrationTests
System Tests

Figure 25: Steps of the process the DIAMONDS framework is applicable

7.2.1.1 Verification framework adaptation

For each one of the process validation steps, the testing validation environment scheme may be abstracted as the left side picture below. Functional test are refined from the specification and executed from a scenario driver to the set of nodes simulated or executed.

The DIAMONDS framework extension consists on the addition of:

- An explicit set of security properties issued from the vulnerability analysis of the protocol being designed.
- The test/scenarios design/specification and generation and execution in order to check the potential violation of the security properties.
- The integration to the validation framework, of monitoring tools able to test the security properties violation.
- The capability to control the behavior of nodes to make them acting as intruder nodes, for active testing.

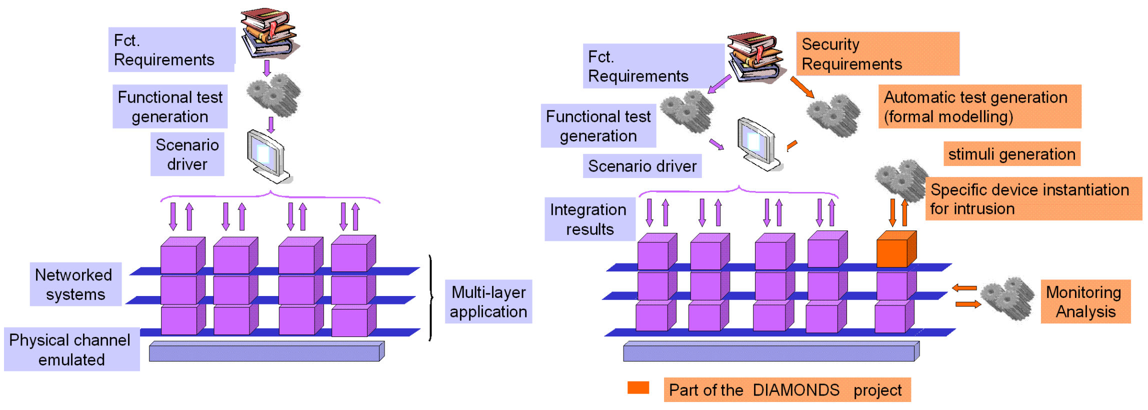

Fct. Requirements
Functional test generation
Scenario driver
Networked systems
Physical channel emulated
Multi-layer application
Fct. Requirements
Security Requirements
Functional test generation
Automatic test generation (formal modelling)
Scenario driver
stimuli generation
Integration results
Specific device instantiation for intrusion
Monitoring Analysis
Part of the DIAMONDS project

Figure 26: Integration of the DIAMONDS framework for one validation process step

7.2.1.2 Adaptation of the event driven simulation environment

Figure 27 shows in the green square the current simulation environment, which means the simulation of a set of "sane" nodes to validate functional behaviours. This initial environment is adapted in order to integrate the following features.

- A set of security testing properties to be defined from a vulnerability analysis specified by the use of a specific notation. These properties are used as an entry point of the monitoring tools. The notation needs to be able to specify properties with sufficient expressivity (timings, occurrences, operations on data, test of absence).

- An integration of monitoring tools, on which potential violation of the security properties may be tested. The validation of the properties may be validated, either on trace files generated (which means offline analysis) or from an API to verify properties on the fly which means online analysis). The offline analysis is done using a standardized format (ASN1 and PCAP) in order to integrate as smoother as possible the monitoring tools to the different validation environments used at the different stages of the process.

- The two first bullets allow passive testing, which means verification of behaviours of nodes whose behaviour has been design to meet the specification of the application. Passive testing does not offer the capability to test the robustness of the radio network with respect to specific intruder behaviour different from the sane node behaviour. The initial execution framework environment integrates the capability to instantiate specific nodes. On these nodes parts of the behaviour related the security testing properties can be controlled by the use of a specific API. Security testing with the capability to test attacks is called active testing.

- With respect to the topological and traffic flow setup situations, incidence on the activation of threats will be different. The threats activation is related to a specific context. This initial environment is adapted, from context information in order to generate automatically scenarios to get nodes in the right situations for security testing against threats. In addition, directives are sent to the monitoring tools to test the security properties related to the threat in the context of the scenario (nodes involved, timing slots the threat is effective).

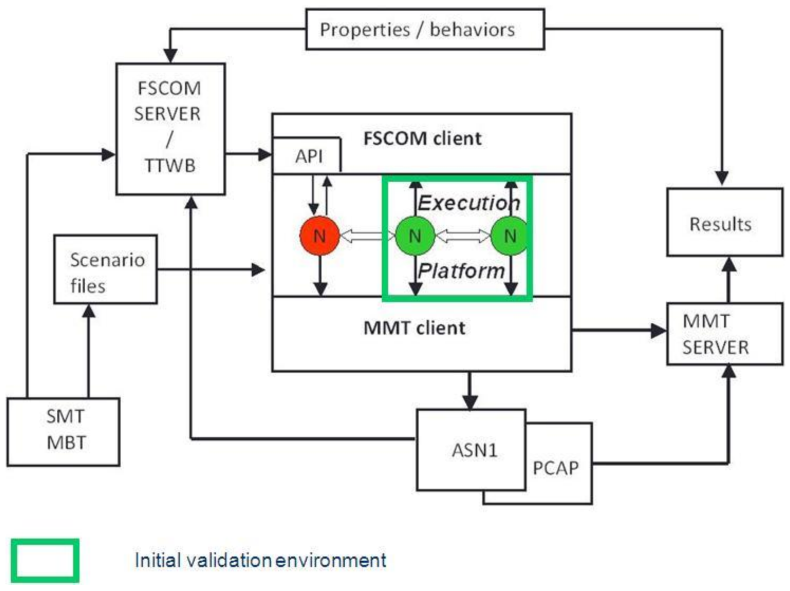


Figure 27: Tools integration (user view)

7.2.2 Properties validated

19 properties have been specified, implemented and validated, a summary of the description of these properties.

Security Rule 1	If one node receives two successive MSG_SPHY_DATA_IND messages from the same source, then these two messages have to be separated by 50 slots (two nodes with close enough to see each other with no packet loss).
Security Rule 2	If one node receives two MSG_SPHY_DATA_IND messages from different sources, then these two messages have to have two different time slots.
Security Rule 3	If node A receives from B an MSG_SPHY_DATA_IND message claiming A as a neighbour, then this means that A received from B at least 3 MSG_SPHY_DATA_IND messages in the last 4 periods (One period = 50 Time Slot).
Security Rule 4	DataUMAC within MSG_SPHY_DATA_IND (SCH) message have to have a management status equal to 10 and a channel presence equal to 10 (hexa values).
Security Rule 5	Number of neighbours has to be between 0 and 127.
Security Rule 6	DataUMAC within MSG_SPHY_DATA_IND message should have K, J, C as follows: K between 1 and 255, J between 3 and 11 and C between 0 and 7.
Security Rule 7	DataUMAC within MSG_SPHY_DATA_IND message is well formatted.
Security Rule 8	The declared neighbours of a node are distinct.
Security Rule 9	The neighbours declared by a node A do not contain the source node A.
Security Rule 10	The bit Z1 in KJC have to be equal to 0 (Tolerance X% = 0).
Security Rule 11	If node B receives from A an MSG_SPHY_DATA_IND message claiming B as a neighbour with a bi directivity bit = 1, then this means that A received from A at least 3 MSG_SPHY_DATA_IND messages in the last 4 periods (One period = 50 TS).
Security Rule 12	The directivity byte in BLOC3 (if any) has to be equal to 0 or 1).
Security Rule 13	KJCs in BLOC2 (if any) has to be different from KJC in BLOC 1).
Security Rule 14	All channels in BLOC2 are distinct.
Security Rule 15	If a node sends a message SPHY_DATA_REQ then receives and SPHY_DATA_IND from another node, the two messages need to have different broadcast Channels BLOC1.
Security Rule 16	If a node receives two SPHY_DATA_IND messages from two different nodes, the two messages need to have different broadcast Channels BLOC1.
Security Rule 17	If node A receives from B an MSG_SPHY_DATA_IND message claiming A as a neighbour, then this means that A already sent at least 3 MSG_SPHY_DATA_REQ messages in the last 4 periods (One period = 50 TS).
Security Rule 18	If node B receives from A an MSG_SPHY_DATA_IND message claiming B as a neighbour with a bi-directivity bit = 1, then this means that B already sent at least 4 MSG_SPHY_DATA_REQ messages in the last 5 periods (One period = 50 TS).
Security Rule 19	If node A receives from B an MSG_SPHY_DATA_IND message claiming A as a neighbour with a bi-directivity bit = 1, then this means that A already sent an MSG_SPHY_DATA_REQ message claiming B as a neighbour.

7.2.3 Active testing

7 threats have been specified, implemented and applied on scenarios, a summary of the description of these threats.

Threat 1	The PDU of the spy node take exactly the same values as another received PDU (specially the mac address) and sent over the air with a wrong timeslot.
Threat 2	The address and broadcast channel of the received station is directly added at the STL list LC at the first reception (normally impossible before 4 receptions).
Threat 3	Fields of the PDU are forced with bad values (K = 0,J = 12, C = 8,number of neighbours = 128).
Threat 4	The spy node retrieves the canal of broadcast [K, J, C] of a received station and uses it as broadcast channel for the PDU (uniqueness of the broadcast channel).
Threat 5	The conflict bit Z1 is put at 1 for each channel of the STL list LC.
Threat 6	At the first reception of a received station, this station is directly added at the LC list with the directivity bit forced at 1.
Threat 7	The spy node puts the received station in the LP list with the directivity bit forced at 1.

7.3 Results

As shown in the previous clauses of the present document, the process followed by the contributors of the Test for security related to the Radio protocol use case was based on requirements about security testing features. These requirements identification was the first activity step.

For the next step, the following activities were leaded. The specification, design and integration of the Radio Protocol use case, with validation through execution of multi nodes, and the analysis between tool providers on the capabilities to interconnect to each other, and to integrate this test for security flow to the Radio protocol validation environment. This last step requires (in particular for the monitoring tools designed and used by some partners) a clear knowledge of the PDU fields of the application and the sequences. These monitoring tools partners have had to implement a specific parser for these messages, validated on the application.

The next step was the specification of security properties using specific languages as entry point of the monitoring tools. These notations had to be extended to take in account the expressiveness of the properties using either the monitoring tool's properties notation or TTCN.

At this stage, passive testing properties were successfully validated at the beginning on files generation of the messages exchanged during the simulation (**offline passive security testing**). This validation was in a second time based on APIs to catch on the fly the messages exchanged (**online passive security testing**).

The following step was based on the capability to execute behaviors of intruder nodes. In order to integrate this feature, the execution platform was adapted in order to monitor particular nodes identified as intruders at the simulation set-up. This monitoring is proceeded by the specification and implementation of a specific API. This API offers the capability in scenarios to change the behavior of the tagged intruder nodes. This API offers the capability to apply a determined threat or inform on the assignment of a PDU to be sent at a particular slot time. In the same manner of the specification of security properties, threats were specified and sent to the simulation, with the test of impact on the set of nodes by the security properties monitoring. To be effective, the threats have to act in a particular context, to be sure of the robustness of these nodes. To ease and automate the scenario generation to validate reaction on threats, scenario modeling and generation with respect to these contexts were implemented with the use of the test design automation tools set. The information on this context were also provided to the monitoring tools (time period of the threat, nodes involved) to ease the analysis and diagnosis of the simulations. **At this stage online active security testing has been validated**.

THALES Communications & security considers as fully reached the capability to test threats from a vulnerability analysis as defined in clause 7.1.3.

List of the (main) metrics used to evaluate the projects achievements:

- Capability to validate the generic vulnerability analysis specified at the beginning of the project
  Result: This capability was conditioned to the capability to process online active testing, which is the case as expressed previously.

- % of coverage of properties deduced from a preliminary vulnerability analysis
  Result: As the vulnerability analysis was defined at early stage of the project, and due to the close work with security analysis tools providers, the adaptation of these tools to the specific needs in terms of execution platform integration and expressiveness of the properties. All the initial properties for security testing were validated.

- % of reuse of the elements designed for the test from a radio protocol to one another
  Result: Similarly to the functional testing, although at coarse grain the vulnerability analysis might be similar, the tune specification and design of the properties is specific to each protocol. However, a general vulnerability analysis can be used to help on the exhaustiveness of the security testing verification.

- Independence of the platform evaluation
  Result: This measure has been taken in account in order to easily adapt the DIAMONDS tools set to the different execution environments. The traces generation has been applied according to standards (e.g. ASN1, PCAP), and the API to inject intruder behaviors has been specified in order to be used in a general manner for radio protocols or other kind of embedded use cases.

- Cost in terms of expertise and time to specify and validate security properties
  Result: The test for security implies an extra effort in terms of properties specification. This effort is similar to the one for functional testing. The difficulty is to define the right level of vulnerability analysis to be exhaustive enough in terms of explicit properties. That can be helped by the definition of global security test patterns common to a particular set of industrial case studies.

7.4 Summary and conclusion

As show in the previous clauses, the test for security framework was integrated into the THALES Communications & Security execution platform into a real size protocol design (several dozen thousands of lines), with complicate Packet Data Units formats including correlations between messages (slots allocations behaviour in message sequences). The monitoring tools succeeded to parse (in a short part time) and exploit these messages traces. The properties on this protocol were specified, using in an easy to learn notation. The diagnosis window which reports the properties validated/violated and the information provided in case of property violation are sufficient for a first step analysis. The API used to control intrusion nodes has been designed in order to be generic and easy to use. Testing tools show experiments on scenarios generation according to a specific context.

The testing framework designed within the DIAMONDS project, and its successful application on the case study, demonstrate the efficiency of the tools integration and application framework on the THALES Communications and Security framework. These tools fulfil the lack of analysis needed to verify robustness against threats from the vulnerability analysis. The integration of these tools on the THALES execution platforms is scheduled, to be proposed as a new feature for the specification and analysis of the next radio protocols.

8 Automotive case study results

8.1 Case study characterization

As Information and Communication Technology (ICT) systems become more and more part of daily lives, current and future vehicles are more and more integrated into ICT networks. The consumer's smartphones, multimedia devices etc. are linked to the vehicles and allow the drivers or passengers to use the internet, to access their private phone books or to run their individual applications through the vehicle's integrated infotainment systems.

Today's most common technology to link consumer devices to in-vehicle electronics is Bluetooth, which latest version is 4.0. Such a wireless connection provides the most comfortable way for the driver to access a variety of services. However, such a wireless connection implies also the risk of possible misuse which leads to enhanced security issues and risks for the automotive electronics and with that for the passengers.

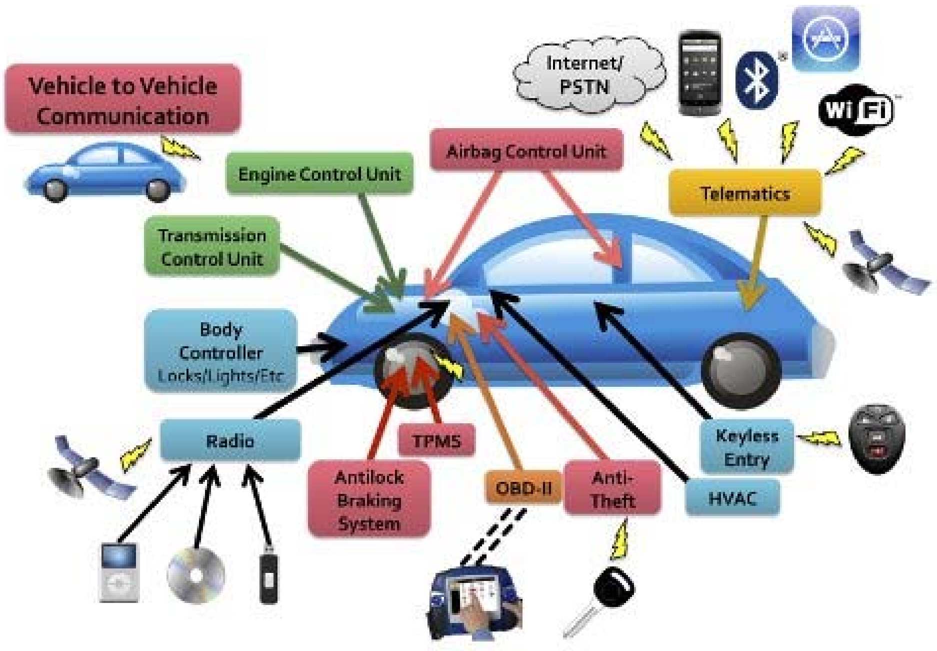


Figure 28: Attack surface of a modern vehicle

Recently the complexity of ICT systems and automotive electronics increases dramatically and requires modern verification and validation methods, which allow handling of complex systems fast and efficiently. This fact is being addressed in the automotive case study. The case study is provided by Dornier Consulting. The System-Under-Test (SUT) is an automotive connectivity module, which provides the driver an ability to connect a mobile phone to the infotainment system. The module itself is connected via the controller area network (CAN) bus to the vehicle. The phone can be linked via the Bluetooth technology. In this case study the Bluetooth specification 2.1 was used.

An overview of the SUT is shown in figure 29. As mentioned before, it is shown that the SUT is connected to the vehicle bus via the CAN network. The connection to a mobile phone is possible over the Bluetooth network, whereas additional USB devices can be attached via a USB interface.

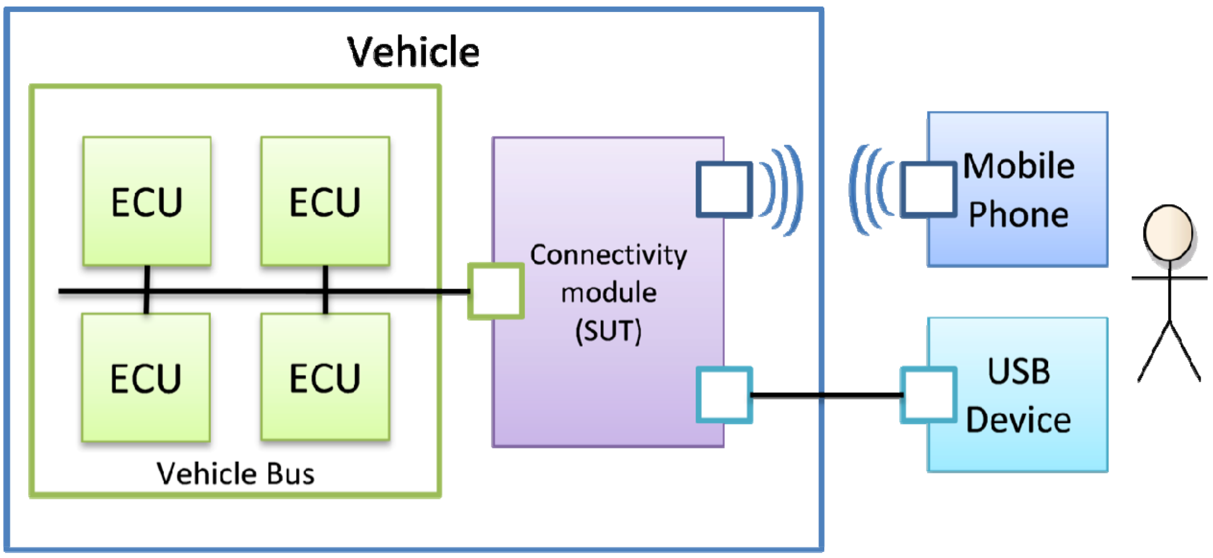


Figure 29: Overview of the SUT

In figure 30 the test SUT is shown as it was presented during the 2012 ITEA2 symposium in Paris. For demonstration purpose, a robot was used to visualize the pairing process. In the lab the SUT was triggered with CAN messages.

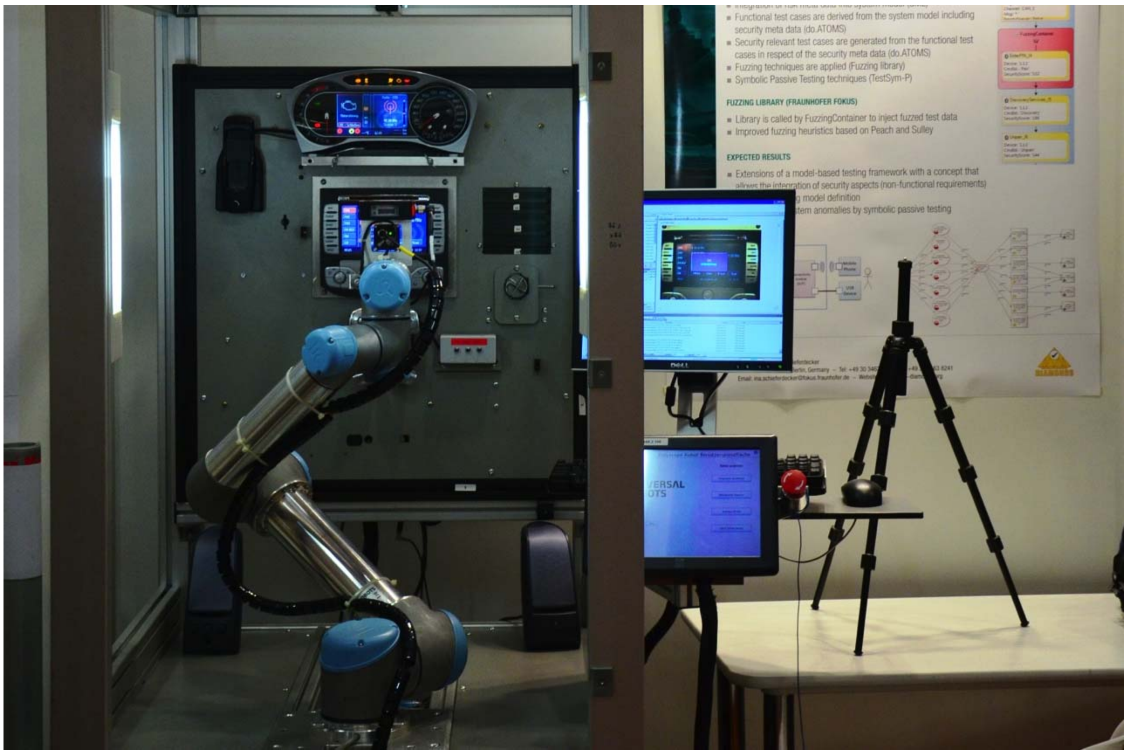


Figure 30: Demonstration of the SUT at the ITEA symposium in Paris

8.2 Security testing approaches

In this case study four different testing techniques have been applied. All of these techniques are described in work package 2 in more detail. An overview and the adaption are described below.

8.2.1 Security risk assessment

Similar to other case studies a security risk assessment has been done. This analysis was done with support of the CORAS approach. For that purpose an analysis with seven different threat scenarios were created. All of these threat scenarios have been analysed and classified with values for vulnerability, likelihood, and consequence.

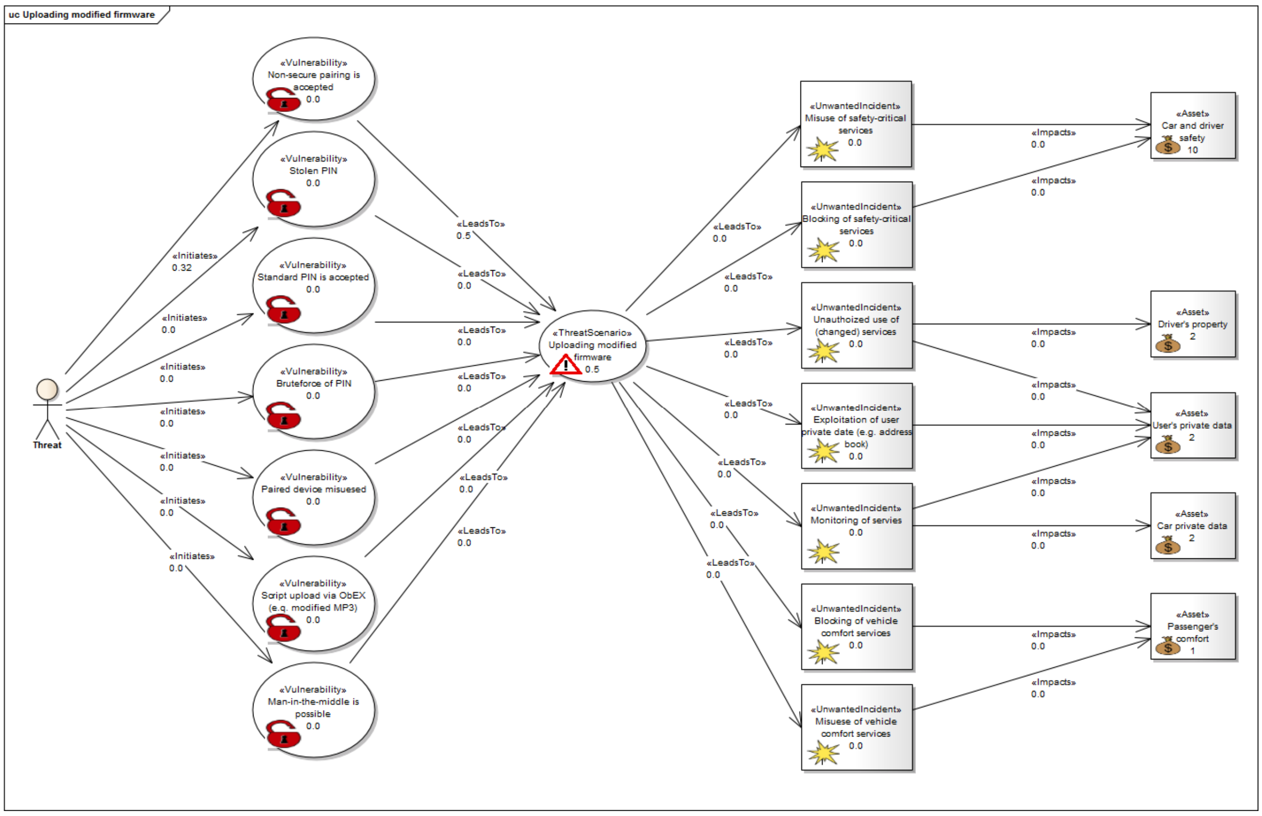
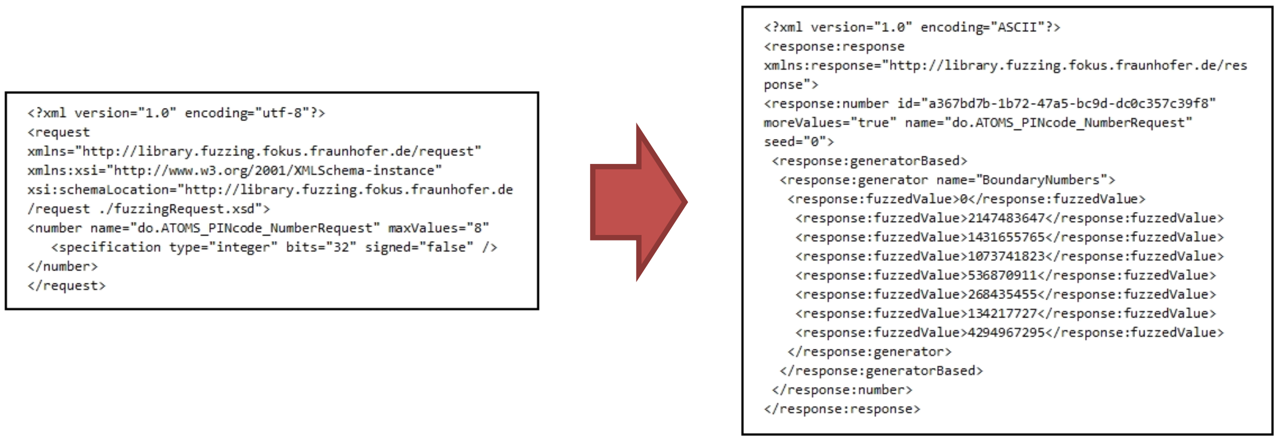

uc Uploading modified firmware
Threat
«Initiates» 0.32
«Initiates» 0.0
«Initiates» 0.0
«Initiates» 0.0
«Initiates» 0.0
«Initiates» 0.0
«Initiates» 0.0
«Vulnerability» Non-secure pairing is accepted 0.0
«Vulnerability» Stolen PIN 0.0
«Vulnerability» Standard PIN is accepted 0.0
«Vulnerability» Bruteforce of PIN 0.0
«Vulnerability» Paired device misuesed 0.0
«Vulnerability» Script upload via ObEX (e.g. modified MP3) 0.0
«Vulnerability» Man-in-the-middle is possible 0.0
«LeadsTo» 0.5
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«ThreatScenario» Uploading modified firmware 0.5
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«LeadsTo» 0.0
«UnwantedIncident» Misuse of safety-critical services 0.0
«UnwantedIncident» Blocking of safety-critical services 0.0
«UnwantedIncident» Unauthoized use of (changed) services 0.0
«UnwantedIncident» Exploitation of user private date (e.g. address book) 0.0
«UnwantedIncident» Monitoring of servies 0.0
«UnwantedIncident» Blocking of vehicle comfort services 0.0
«UnwantedIncident» Misuese of vehicle comfort services 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Impacts» 0.0
«Asset» Car and driver safety 10
«Asset» Driver's property 2
«Asset» User's private data 2
«Asset» Car private data 2
«Asset» Passenger's comfort 1

Figure 31: Example of a security risk assessment for uploading modified firmware

8.2.2 Fuzzing

Fuzzing is a technique that injects invalid or random input data in order to reveal vulnerabilities or unexpected behaviour by processing this data. How this input data is generated depends on the data format. Fuzz data generators specific for certain data formats are generally more successful in finding vulnerabilities than generating invalid input data randomly without respecting the data format. A fuzzing library that allows fuzz data generation for a wide range of data formats was developed by Fraunhofer FOKUS and integrated in the test generation and execution environment do.ATOMS^{TM}.

In order to test against the vulnerabilities identified during security risk assessment, corresponding fuzz data generators were selected. On the basis of the security score, message parameters for a PIN number and a device name are targets of fuzzed input data. Their data format was specified as well as the fuzz data generators *BadStrings*, *AllBadStrings*, and *BoundaryNumbers* within a request and the generated fuzz test data were returned by the fuzzing library as depicted in figure 32.



```
<?xml version="1.0" encoding="utf-8"?>
<request
xmlns="http://library.fuzzing.fokus.fraunhofer.de/request"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://library.fuzzing.fokus.fraunhofer.de
/request ./fuzzingRequest.xsd">
<number name="do.ATOMS_PINCode_NumberRequest" maxValues="8"
   <specification type="integer" bits="32" signed="false" />
</number>
</request>
```

```
<?xml version="1.0" encoding="ASCII"?>
<response:response
xmlns:response="http://library.fuzzing.fokus.fraunhofer.de/res
ponse">
<response:number id="a367bd7b-1b72-47a5-bc9d-dc0c357c39f8"
moreValues="true" name="do.ATOMS_PINcode_NumberRequest"
seed="0">
 <response:generatorBased>
  <response:generator name="BoundaryNumbers">
   <response:fuzzedValue>0</response:fuzzedValue>
    <response:fuzzedValue>2147483647</response:fuzzedValue>
    <response:fuzzedValue>1431655765</response:fuzzedValue>
    <response:fuzzedValue>1073741823</response:fuzzedValue>
    <response:fuzzedValue>536870911</response:fuzzedValue>
    <response:fuzzedValue>268435455</response:fuzzedValue>
    <response:fuzzedValue>134217727</response:fuzzedValue>
    <response:fuzzedValue>4294967295</response:fuzzedValue>
   </response:generator>
  </response:generatorBased>
 </response:number>
</response:response>
```

Figure 32: Fuzzing request using the library

8.2.3 IOSTS-based passive testing approach

Figure 33 shows the framework of the symbolic passive testing approach. In this approach, the passive testing is performed by integration of two techniques: Symbolic execution and Parametric Trace Slicing. Based on the system specification or requirements the behaviour/attack scenario is identified for the Bluetooth protocol and modelled using Input-Output Symbolic Transition System (IOSTS). Then the symbolic execution of the IOSTS model results in a tree like structure with each branch corresponding to either behaviour or an attack scenario. The trace of the symbolic execution of IOSTS constitutes the different branches observed in the tree. For the real trace analysis, the parametric trace slicing approach is applied. The obtained Bluetooth trace is sliced based on the parametric instance observed in the messages. Then each slice is compared (for the control + data portion) with the traces of the symbolic execution and a verdict *Pass/Fail/Attack-Pass/Inconclusive* is provided. Based on the verdicts provided for each slice a final verdict to show the implementation satisfies the behaviour is provided. A more detailed overview of the symbolic approach is documented in D5.WP2 [i.2].

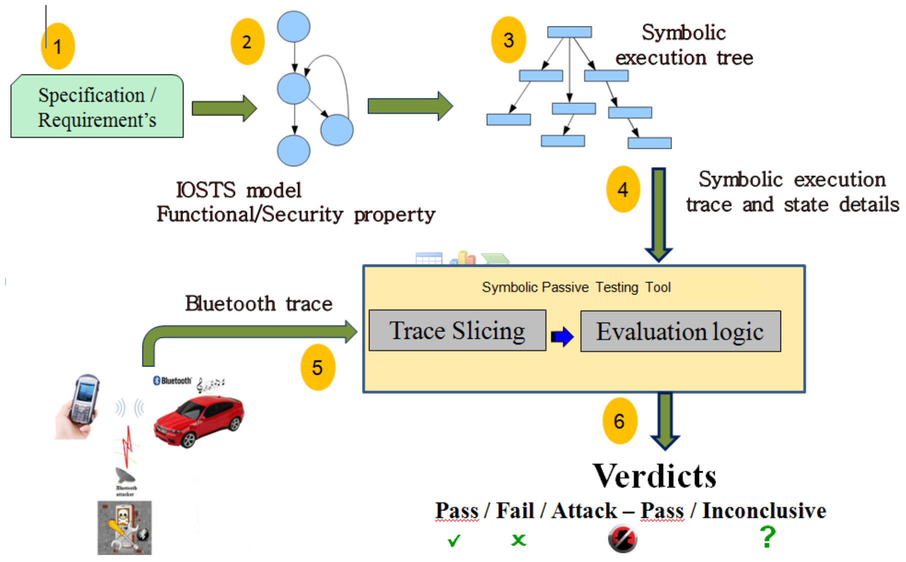

1
Specification / Requirement's
2
IOSTS model
Functional/Security property
3
Symbolic execution tree
4
Symbolic execution trace and state details
Bluetooth trace
5
Symbolic Passive Testing Tool
Trace Slicing
Evaluation logic
6
Verdicts
Pass / Fail / Attack – Pass / Inconclusive
✓
✕
?

Figure 33: Architecture for the symbolic passive testing approach

In addition to the theoretical framework, a symbolic passive testing tool was developed, TestSym-P that helps in the passive testing approach (a detailed description is provided in the DIAMONDS deliverable D5.WP3 [i.3]). The algorithms for parametric trace slicing and evaluation logic are explained in the DIAMONDS deliverable D5.WP2 [i.2] and are implemented in this tool and promising results are obtained.

8.2.3.1 Experimentation results

For the experiments, few traces were provided for the automotive case study. The output from the Bluetooth stack was provided as Bluetooth traces for the experiments. The Bluetooth traces show a pairing process of a car to the phone as shown in figure 33. Each of the Bluetooth traces obtained had a different device local name. In order to evaluate the efficiency of the approach, the experiments were performed in two ways: with unmodified traces and by manually introducing errors and also by introducing a few fake messages to create an attack scenario in the real trace as detailed in D5.WP2. The results of the experiments are shown in table 2.

Table 2: Prototype tool results on sample Bluetooth traces

Trace	No. Messages	No. Slices	Trace Output without errors				Trace Outputs with errors and attacks				
			P	F	I	Final O/P	P	F	I	AP	Final O/P
1	81	2	1	-	1	I	-	1	1	-	F
2	89	3	1	-	2	I	-	-	2	1	AP
3	81	2	1	-	1	I	-	1	1	-	F
4	81	2	1	-	1	I	-	-	1	1	AP
5	81	2	1	-	1	I	-	1	1	-	F
6	81	2	1	-	1	I	-	-	1	1	AP
7	81	2	1	-	1	I	-	-	1	1	AP

8.2.3.2 Future works

Currently, the traces are obtained from the Host Controller Interface (HCI) layer, but the efficiency of the approach could be well appreciated if we could obtain traces from other layers (such as L2CAP layers) as there are more parameters than it was possible to monitor for performing security check. However, this idea is still under discussion.

8.2.4 Security monitoring

Unlike the active fuzzing testing technique described in clause 8.2.2, passive monitoring does not inject any traffic in the network nor try to modify it. This is crucial because any injected message/packet during the system operation may modify the connectivity and communication module (SCM) environment, which is incorporated in an in-vehicle infotainment system, causing problems or crashes of this module. Passive monitoring intends to understand the behaviour of the SCM and analyze it according to several security requirements described as a set of security properties. Thus, the passive monitoring approach seems to be the ideal means for directly verifying the SCM in natural operational run-time conditions. In addition, with this approach, the tests can be run during the whole service life-time as opposed to active testing campaigns that are run for a limited period of time.

Security monitoring based on a monitoring tool is performed with the assistance of a network sniffer to capture network traffic. The analysis is done offline and the pre-captured traces are provided by Dornier Consulting internally by using a built sniffer with the standard output of the Bluetooth Stack (see figure 34). Indeed, in order to interface the user's electronic devices, the connectivity module is equipped with a Bluetooth and a USB interface. The Bluetooth interface allows the user to connect devices like mobiles or a Bluetooth audio to the car. The sniffer collects all the traffic to/from the device and stores it in a dedicated file.

```
Trace Generated at 02:34:23pm on August 6, 2012.

HCI-CMD: HCC_RESET +00.000s [Length 0]
HCI-EVT: HCE_COMMAND_COMPLETE +00.005s [Length 4]
         NumHCICommandPackets:1 Command:HCC_RESET
         Status:NO_ERROR
HCI-CMD: HCC_READ_BUFFER_SIZE +00.006s [Length 0]
HCI-EVT: HCE_COMMAND_COMPLETE +00.007s [Length 11]
         NumHCICommandPackets:1 Command:HCC_READ_BUFFER_SIZE
         Status:NO_ERROR
         ACL Buffer Len =  1021, Num ACL Buffers = 8
         SCO Buffer Len =    64, Num SCO Buffers = 1
HCI-CMD: HCC_HOST_BUFFER_SIZE +00.008s [Length 7]
         ACL Buffer Len =  1021, Num ACL Buffers = 120
         SCO Buffer Len =    60, Num SCO Buffers = 20
HCI-EVT: HCE_COMMAND_COMPLETE +00.009s [Length 4]
         NumHCICommandPackets:1 Command:HCC_HOST_BUFFER_SIZE
         Status:NO_ERROR
HCI-CMD: HCC_WRITE_PAGE_TIMEOUT +00.010s [Length 2]
         00 20
HCI-EVT: HCE_COMMAND_COMPLETE +00.018s [Length 4]
         NumHCICommandPackets:1 Command:HCC_WRITE_PAGE_TIMEOUT
         Status:NO_ERROR
….
```

Figure 34: Extract of captured traces

To be able to analyse these traces and retrieve potential security flaws and vulnerabilities, three main stages have been performed. The first stage was the implementation of a new plugin in a monitoring tool's extract library (cf. the Bluetooth sniffer block called "SNF plugin" in figure 35) to allow the parsing of the log files provided by the Bluetooth sniffer and the extraction of relevant data for the security analysis. One of the important extracted attributes was the events capture time (e.g. +00,006 s) that is crucial for the management of time constraints in the specified security properties.

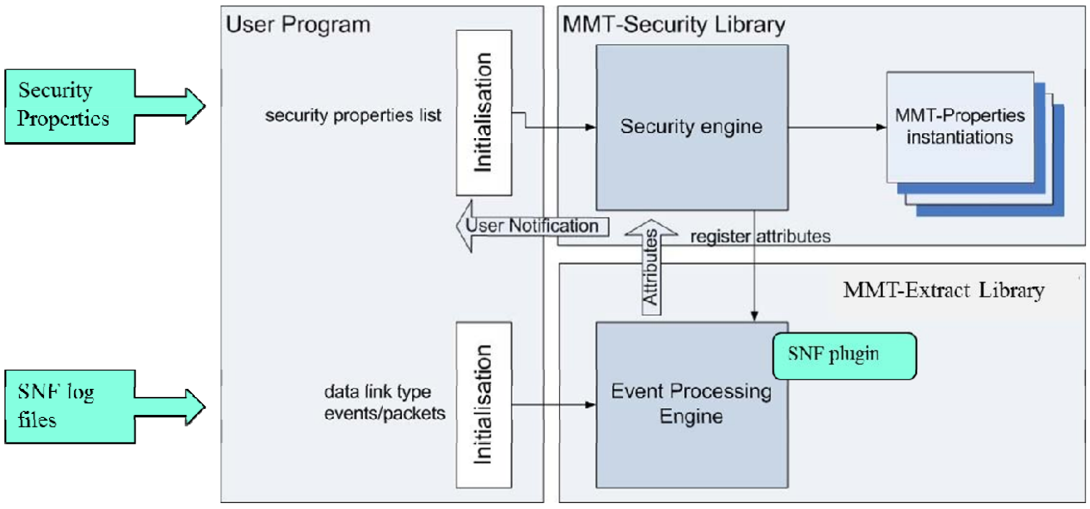
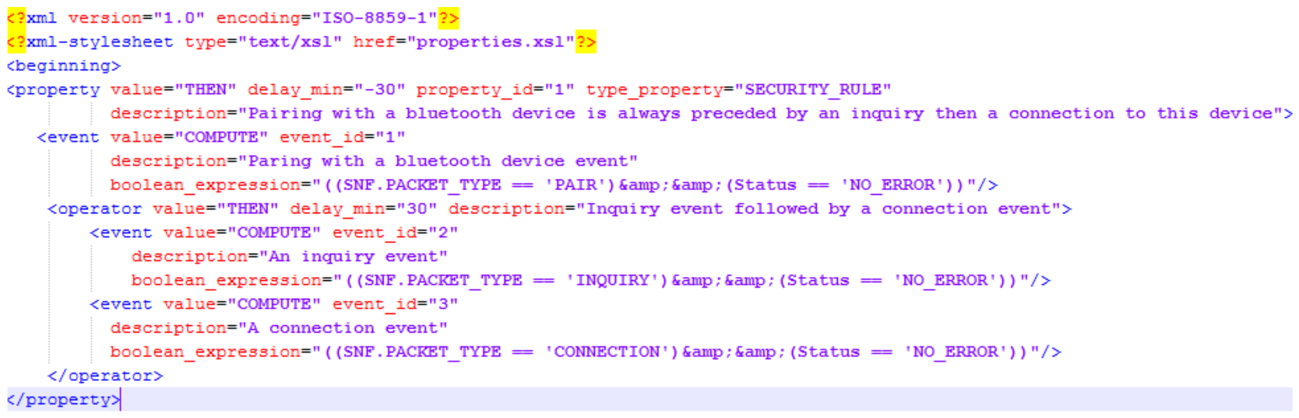


Figure 35: Security monitoring of Bluetooth communication

The second stage is the description of the system security goals denoting desired behaviour of the SCM based on the monitoring tool's security property format. The description specifies the security rules that the studied system has to respect. In the context of the automotive case study, the main idea was to implement a proof of concept of the monitoring approach in order to analyse a basic communication scenario composed of 6 steps: (1) Inquiry (2) Connect to 'Ford Audio' (3) Pair with 'Ford Audio' (4) Discovery of provided services (5) Unpair (6) Disconnect. A set of 3 security properties have been specified denoting the correct order of log files events:

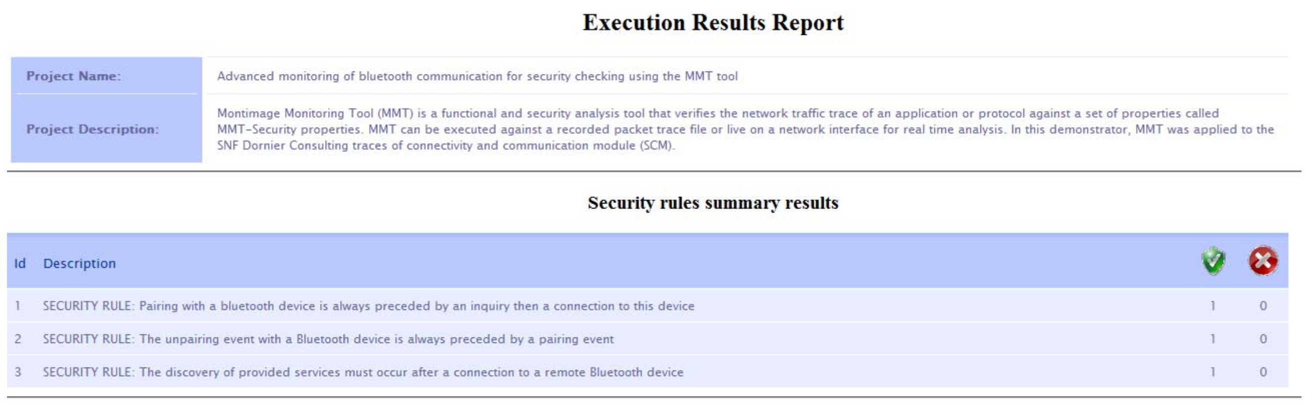
- Pairing with a Bluetooth device has to be preceded by an inquiry and a connection to the device (see figure 36 that specifies the property in an XML format).
- The "unpairing" event with a Bluetooth device is always preceded by a pairing event.
- The discovery of provided services has to occur after a connection to a remote Bluetooth device.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="properties.xsl"?>
<beginning>
<property value="THEN" delay_min="-30" property_id="1" type_property="SECURITY_RULE"
          description="Pairing with a bluetooth device is always preceded by an inquiry then a connection to this device">
   <event value="COMPUTE" event_id="1"
          description="Paring with a bluetooth device event"
          boolean_expression="((SNF.PACKET_TYPE == 'PAIR')&amp;&amp;(Status == 'NO_ERROR'))"/>
    <operator value="THEN" delay_min="30" description="Inquiry event followed by a connection event">
        <event value="COMPUTE" event_id="2"
            description="An inquiry event"
            boolean_expression="((SNF.PACKET_TYPE == 'INQUIRY')&amp;&amp;(Status == 'NO_ERROR'))"/>
        <event value="COMPUTE" event_id="3"
          description="A connection event"
          boolean_expression="((SNF.PACKET_TYPE == 'CONNECTION')&amp;&amp;(Status == 'NO_ERROR'))"/>
    </operator>
</property>
```

Figure 36: An example of a formal security property in the automotive case study

The last stage deals with the analysis of the security properties. This task is performed by a monitoring tool on a set of pre-captured traces. No violation has been detected (see next execution report) but some errors have been manually introduced in the traces to demonstrate the efficiency of the monitoring tool.



Execution Results Report

Project Name:	Advanced monitoring of bluetooth communication for security checking using the MMT tool
Project Description:	Montimage Monitoring Tool (MMT) is a functional and security analysis tool that verifies the network traffic trace of an application or protocol against a set of properties called MMT-Security properties. MMT can be executed against a recorded packet trace file or live on a network interface for real time analysis. In this demonstrator, MMT was applied to the SNF Dornier Consulting traces of connectivity and communication module (SCM).

Security rules summary results

Id	Description	✓	✗
1	SECURITY RULE: Pairing with a bluetooth device is always preceded by an inquiry then a connection to this device	1	0
2	SECURITY RULE: The unpairing event with a Bluetooth device is always preceded by a pairing event	1	0
3	SECURITY RULE: The discovery of provided services must occur after a connection to a remote Bluetooth device	1	0

Figure 37: Execution report

Passive monitoring can be coupled with DIAMONDS active testing and fuzzing techniques. It can be applied as part of the testing chain, e.g. after the execution of some security-oriented test cases, in order to collect network traffic and analyse it based on previously defined security properties. This technique allows detecting potential vulnerabilities and attacks (i.e. similar to what intrusion detection systems do) such as data alteration and QoS attacks. It can also be used to assess the robustness of the system under test according to some security requirements.

8.2.5 Framework

The described fuzzing approach for this case study was integrated within the testing framework of the study's contractor. Following this approach the framework serves as a model based testing (MBT) environment for discreet interconnected embedded systems. Test cases are generated automatically out of a given system model. The system model is based on UML and specifies the structure as well as the behaviour of the system under test (SUT). The system model gets processed by a so called model parser – depending on the UML tool used for creating the system model. The framework provides a variety of model parser implementations. The test cases can be transferred into a test case library and can be edited in an easy way by using an editor that shows the transformed test cases as workflows. During test case creation the system model parser also tries to parameterize the test cases so they are directly executable on the corresponding SUT test setup. The test case designer also handles test case management as well as the execution of the test cases and the reporting of the test results. The test execution layer of do.ATOMS^{TM} has a direct access to the interfaces provided by the SUT and follows a service-orientated approach.

8.3 Results

During the case study 1 836 test cases were generated based on functional test cases. The number of test cases is controlled by the number of variation, which can be produced with the fuzzing library. The generated test cases have a modification on the payload of the L2CAP messages. Instead of the original device name a fuzzed string was used. This string is displayed on the display to start the bonding process. An analysis showed that this string is transmitted by the Bluetooth hardware in an undecoded way, so that any potential character can be send to the SUT. In addition this is done before any authorization. Therefore this allows a perfect entry point to find a flaw in the system. Figure 38 shows the theoretically necessary steps in order to use the described scenario. In order to verify the success of such a test case and in order to verify the health of the SUT, 11 steps are necessary within one test run. A test run takes in average 36 seconds.

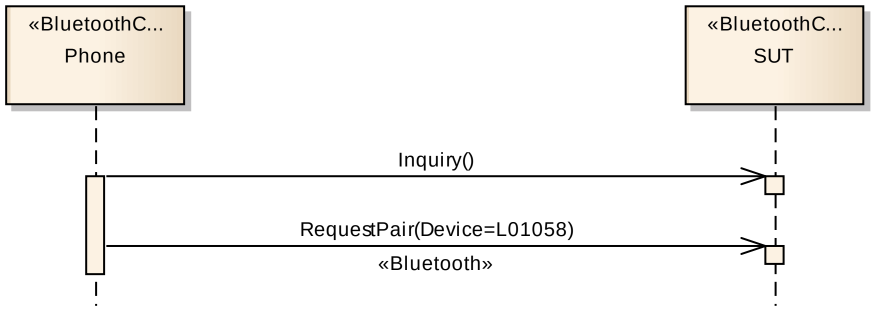
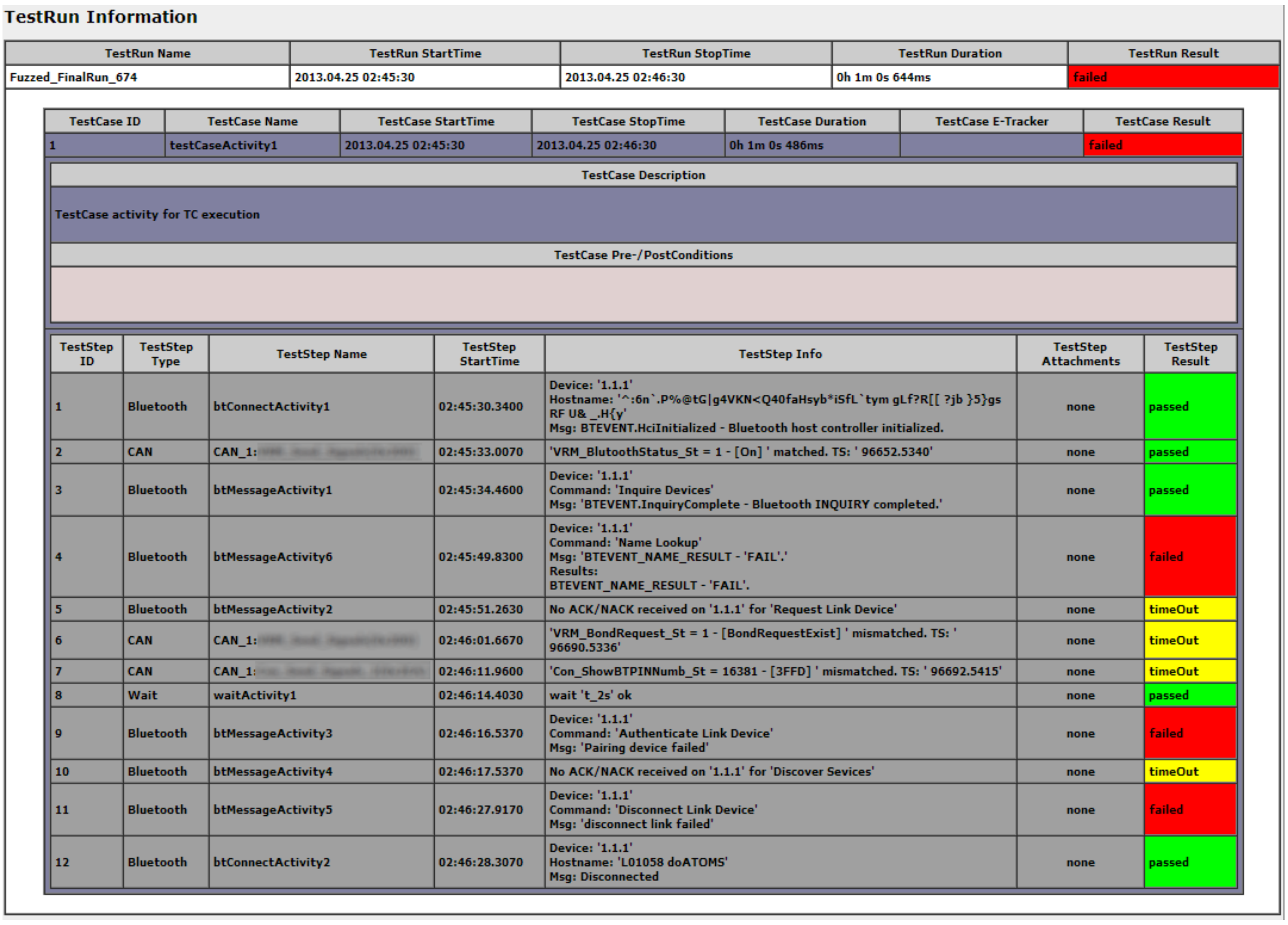


Figure 38: Entry point of fuzzed device name scenario

Table 3: Test run results

# test cases	# errors	# Bluetooth errors	# discovered flaws	duration	Comments
1 836	1	1	0	24,75 hours	
1 836	3	3	0	15,0 hours	
1 633	11	11	0	31,25 hours	Framework in debug mode

Bluetooth errors usually appear when the SUT does not show up in the inquiry list. In fact that could be a sign that the previous test cases might cause the SUT to stop reacting. But usually this is just a random error due to the technology. All appeared errors have been tested and verified as an inquiry time-out. In figure 39 a failed test case report is shown. Step 3 usually shows a list of found devices. Since the SUT was not found the step 3 does not report any devices in the report. Therefore the step 4 failed, since that is the step, which searches for the SUT in the inquiry list. Caused by the missing SUT in step 3 all the following steps cannot be finished and result in a fail or a time-out.



TestRun Information

TestRun Name	TestRun StartTime	TestRun StopTime	TestRun Duration	TestRun Result
Fuzzed_FinalRun_674	2013.04.25 02:45:30	2013.04.25 02:46:30	0h 1m 0s 644ms	failed

TestCase ID	TestCase Name	TestCase StartTime	TestCase StopTime	TestCase Duration	TestCase E-Tracker	TestCase Result
1	testCaseActivity1	2013.04.25 02:45:30	2013.04.25 02:46:30	0h 1m 0s 486ms		failed

TestCase Description

TestCase activity for TC execution

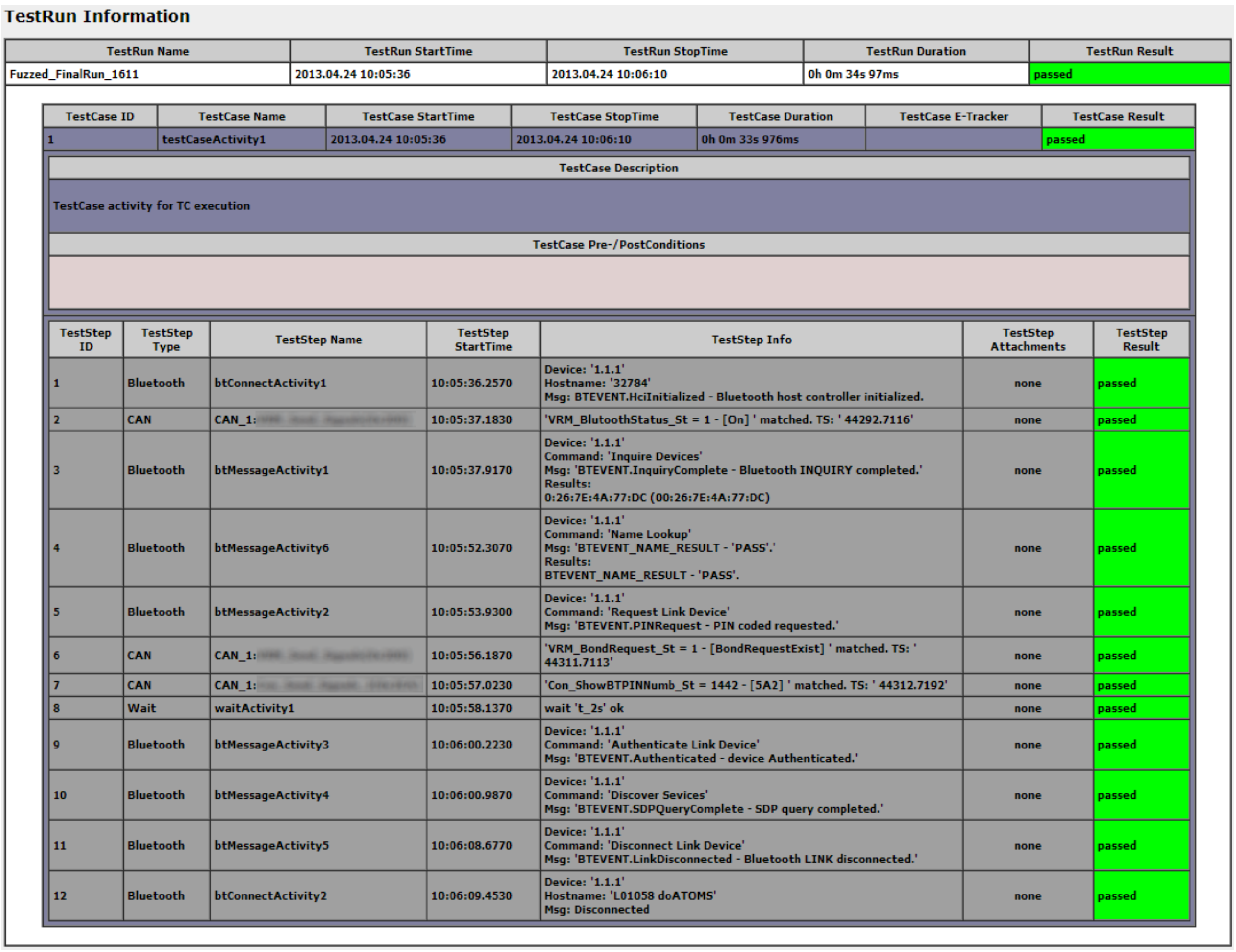
TestCase Pre-/PostConditions

TestStep ID	TestStep Type	TestStep Name	TestStep StartTime	TestStep Info	TestStep Attachments	TestStep Result
1	Bluetooth	btConnectActivity1	02:45:30.3400	Device: '1.1.1' Hostname: '^:6n`.P%@tG|g4VKN<Q40faHsyb*iSfL`tym gLf?R[[?jb }5}gs RF U& _.H{y' Msg: BTEVENT.HciInitialized - Bluetooth host controller initialized.	none	passed
2	CAN	CAN_1:[redacted]	02:45:33.0070	'VRM_BlutoothStatus_St = 1 - [On] ' matched. TS: ' 96652.5340'	none	passed
3	Bluetooth	btMessageActivity1	02:45:34.4600	Device: '1.1.1' Command: 'Inquire Devices' Msg: 'BTEVENT.InquiryComplete - Bluetooth INQUIRY completed.'	none	passed
4	Bluetooth	btMessageActivity6	02:45:49.8300	Device: '1.1.1' Command: 'Name Lookup' Msg: 'BTEVENT_NAME_RESULT - 'FAIL'.' Results: BTEVENT_NAME_RESULT - 'FAIL'.	none	failed
5	Bluetooth	btMessageActivity2	02:45:51.2630	No ACK/NACK received on '1.1.1' for 'Request Link Device'	none	timeOut
6	CAN	CAN_1:[redacted]	02:46:01.6670	'VRM_BondRequest_St = 1 - [BondRequestExist] ' mismatched. TS: ' 96690.5336'	none	timeOut
7	CAN	CAN_1:[redacted]	02:46:11.9600	'Con_ShowBTPINNumb_St = 16381 - [3FFD] ' mismatched. TS: ' 96692.5415'	none	timeOut
8	Wait	waitActivity1	02:46:14.4030	wait 't_2s' ok	none	passed
9	Bluetooth	btMessageActivity3	02:46:16.5370	Device: '1.1.1' Command: 'Authenticate Link Device' Msg: 'Pairing device failed'	none	failed
10	Bluetooth	btMessageActivity4	02:46:17.5370	No ACK/NACK received on '1.1.1' for 'Discover Sevices'	none	timeOut
11	Bluetooth	btMessageActivity5	02:46:27.9170	Device: '1.1.1' Command: 'Disconnect Link Device' Msg: 'disconnect link failed'	none	failed
12	Bluetooth	btConnectActivity2	02:46:28.3070	Device: '1.1.1' Hostname: 'L01058 doATOMS' Msg: Disconnected	none	passed

Figure 39: Report of a test case failed due missing device in the inquiry list

In figure 40 a valid test report is shown. Here it is shown that in step 3 the SUT was found and therefore the entire test case ran without any errors.

One of the final runs took place in a lab environment and therefore only one Bluetooth error happened. Besides that the system behaved as expected. It has to be mentioned that the SUT is a device, which is on the market for several years and used in many different automotive models of the OEM. Nevertheless the described combination of the different testing technologies including the security risk assessment shows a very interesting approach. The security risk assessment indicates the weak points of a system and allows a concentrated testing. Using Dornier Consulting's do.ATOMS^{TM} framework in combination with the Fuzzing Library from Fraunhofer enables a very effective automatic testing. The 1 836 automatic generated test cases were executed in the laboratory and ran unattended until all 1 836 test cases finished. During the test runs the framework records the traces, which then can be used for the post analysis.



TestRun Information

TestRun Name	TestRun StartTime	TestRun StopTime	TestRun Duration	TestRun Result
Fuzzed_FinalRun_1611	2013.04.24 10:05:36	2013.04.24 10:06:10	0h 0m 34s 97ms	passed

TestCase ID	TestCase Name	TestCase StartTime	TestCase StopTime	TestCase Duration	TestCase E-Tracker	TestCase Result
1	testCaseActivity1	2013.04.24 10:05:36	2013.04.24 10:06:10	0h 0m 33s 976ms		passed

TestCase Description
TestCase activity for TC execution

TestCase Pre-/PostConditions

TestStep ID	TestStep Type	TestStep Name	TestStep StartTime	TestStep Info	TestStep Attachments	TestStep Result
1	Bluetooth	btConnectActivity1	10:05:36.2570	Device: '1.1.1' Hostname: '32784' Msg: BTEVENT.HciInitialized - Bluetooth host controller initialized.	none	passed
2	CAN	CAN_1:[redacted]	10:05:37.1830	'VRM_BlutoothStatus_St = 1 - [On] ' matched. TS: ' 44292.7116'	none	passed
3	Bluetooth	btMessageActivity1	10:05:37.9170	Device: '1.1.1' Command: 'Inquire Devices' Msg: 'BTEVENT.InquiryComplete - Bluetooth INQUIRY completed.' Results: 0:26:7E:4A:77:DC (00:26:7E:4A:77:DC)	none	passed
4	Bluetooth	btMessageActivity6	10:05:52.3070	Device: '1.1.1' Command: 'Name Lookup' Msg: 'BTEVENT_NAME_RESULT - 'PASS'.' Results: BTEVENT_NAME_RESULT - 'PASS'.	none	passed
5	Bluetooth	btMessageActivity2	10:05:53.9300	Device: '1.1.1' Command: 'Request Link Device' Msg: 'BTEVENT.PINRequest - PIN coded requested.'	none	passed
6	CAN	CAN_1:[redacted]	10:05:56.1870	'VRM_BondRequest_St = 1 - [BondRequestExist] ' matched. TS: ' 44311.7113'	none	passed
7	CAN	CAN_1:[redacted]	10:05:57.0230	'Con_ShowBTPINNumb_St = 1442 - [5A2] ' matched. TS: ' 44312.7192'	none	passed
8	Wait	waitActivity1	10:05:58.1370	wait 't_2s' ok	none	passed
9	Bluetooth	btMessageActivity3	10:06:00.2230	Device: '1.1.1' Command: 'Authenticate Link Device' Msg: 'BTEVENT.Authenticated - device Authenticated.'	none	passed
10	Bluetooth	btMessageActivity4	10:06:00.9870	Device: '1.1.1' Command: 'Discover Sevices' Msg: 'BTEVENT.SDPQueryComplete - SDP query completed.'	none	passed
11	Bluetooth	btMessageActivity5	10:06:08.6770	Device: '1.1.1' Command: 'Disconnect Link Device' Msg: 'BTEVENT.LinkDisconnected - Bluetooth LINK disconnected.'	none	passed
12	Bluetooth	btConnectActivity2	10:06:09.4530	Device: '1.1.1' Hostname: 'L01058 doATOMS' Msg: Disconnected	none	passed

Figure 40: In contrast to the failed report, this is a report with no Bluetooth error

8.4 Summary and conclusion

The described combination of a security risk assessment, an effective test generation, and the fuzzing approach was new for the partners of the automotive case study. For upcoming projects Dornier Consulting will consider a feasible security risk assessment. This security risk assessment can be done independently from the testing approach and is useful in any means. It enables the awareness for security and shows weaknesses in the system design. Therefore the security risk assessment is definitively a strong tool for future projects. Based on the presented approach the security risk assessment is done before the security testing, which then can use the information from the security risk assessment in order to optimize the testing.

Additional to the security risk assessment the contractor integrated the Fuzzing library to its testing framework. Therefore it is able to offer the entire approach as a business plan to new and existing customers. In the current version of the framework the security testing approach is integrated as a separated process. Currently the contractor is working on a new version of its framework, which will introduce a new test case generation process. This process is suitable for the security testing approach and enables security testing in the framework. In general this will lead to more secure systems, as the security risk assessment identifies the vulnerabilities of the system and the fuzzing library allows testing these vulnerabilities.

9 eHealth case study results

9.1 Case study characterization

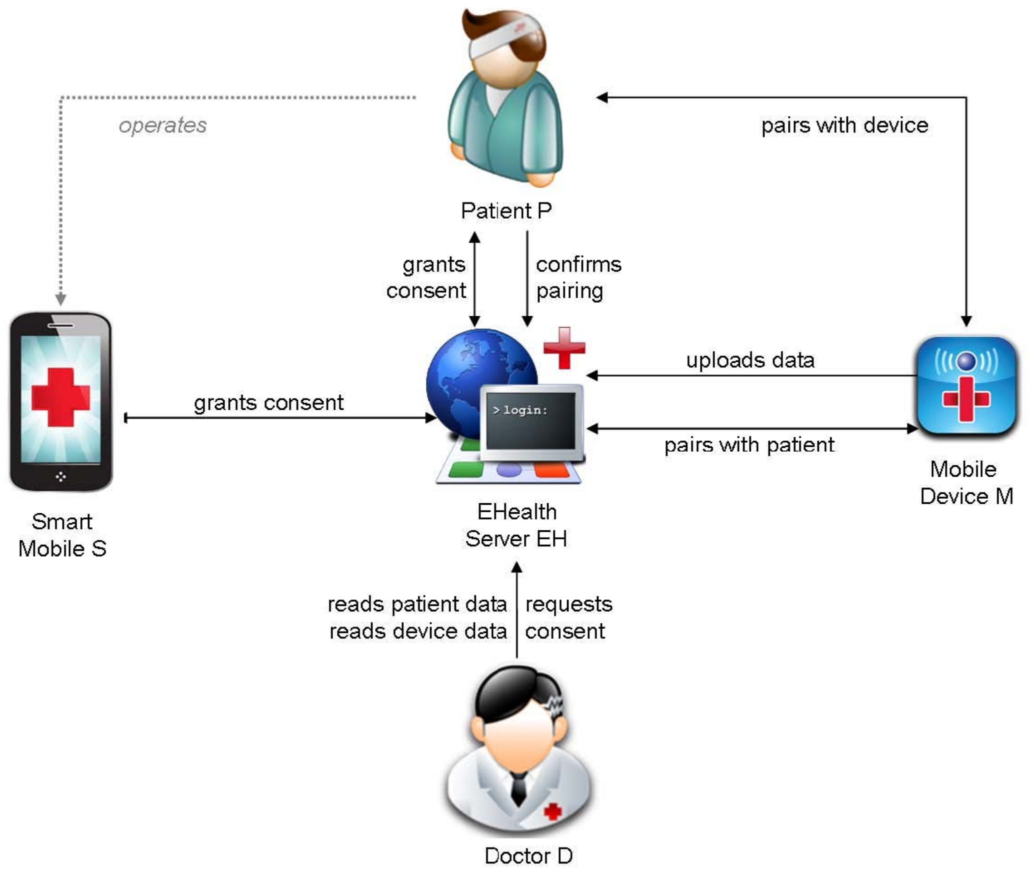


Figure 41: The eHealth scenario

In the eHealth scenario (see figure 41), ambulant patients are monitored using mobile devices worn by those patients. The mobile devices supply doctors with the vital data they recorded about their patients. The data are uploaded to a central server via wireless communication. From the central server authorized doctors may download the monitored data. The data recorded by the mobile device should be associated with the correct patient when uploaded to the central server. Vital medical decisions might be based on them. Patients should consent to a mobile device being associated with them. The patients' personal data in general and such vital medical data in particular are highly sensitive. Doctors need to obtain consent of a patient before they are allowed to retrieve patient data from the central server.

Thus it is necessary to model two authorization workflows:

- **Patient Consent:** The patient grants consent to a doctor who then may access the patient's personal data at the eHealth server. This workflow is new to the eHealth scenario and documented in this clause for the first time.
- **Device Pairing:** The patient entitles the mobile monitoring device he is wearing to upload the monitored data to the eHealth server.

What does the initial setup for the scenario look like? It is assumed that doctors, patients, and devices are registered at the central eHealth server and equipped with the necessary credentials, like passwords or other shared secrets. The doctor, for instance in a consultation session, personally may hand over the initial credentials to the patient. Patients can use these credentials to log in at the central server, to manage their personal data and to view the data recorded about them by the mobile devices associated with them.

What is the role of the mobile monitoring device? If a patient requires monitoring by a mobile device, the doctor for instance may hand over an appropriate device to the patient. These devices then monitor the vital signs of the patients, e.g. take their blood pressure, monitor their pulse, or measure their blood sugar. The mobile monitoring devices upload the recorded data to the central eHealth server. There, once the patient's consent has been obtained, doctors can view such data in order to choose an appropriate treatment for their patients.

Which mobile monitoring devices may upload data about which patients? Patients have to explicitly authorize a mobile monitoring device to upload monitoring data about themselves to the eHealth server. For this the patient needs to "pair" with the mobile device first. Only then the monitoring data recorded by the mobile device are associated with that patient. Only at most one patient may be paired with a mobile monitoring device at a given time.

Which doctors may access which patients' data at the central server? Doctors require explicit prior consent of their patients before they can access these patients' data. The central server obtains that consent on behalf of the doctor from the patient. Only after successfully having obtained this consent, the central server grants the doctor access to sensitive patient data.

The overall message workflow is sketched in figure 42. Some details of the two main sub-workflows, "pairing" and "consent" are introduced right here, a more formal modelling is provided in clause 9.4.

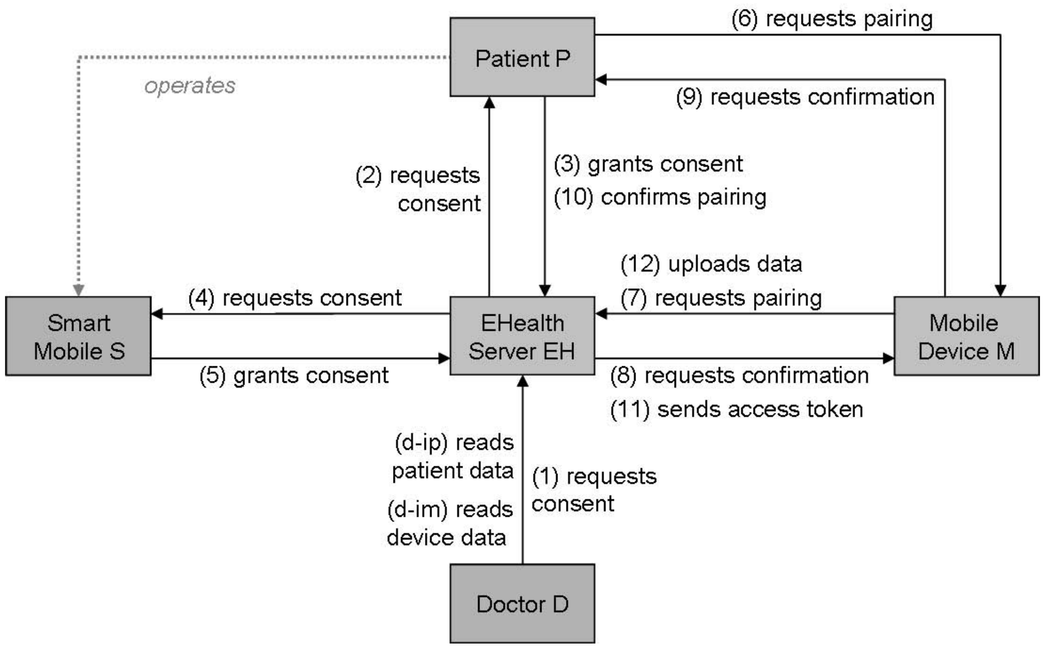


Figure 42: The overall workflow

9.1.1 Patient consent

How is the consent of a patient obtained? The consent workflow is newly introduced in this clause. It comprises a two-factor authorization both via the patient's account at the eHealth server and the patient's browser on the one hand and the patient's smart mobile on the other hand. First, the doctor applies for patient consent at the central server (1). Then the central sever both sends a consent request to the patient's account at the central server (2) and to the smart mobile associated with that patient (3). The patient has to grant consent both via the account, using a normal browser (4) and via the smart phone, using a special application (5). A consent is granted successfully only, if the patient has consented both ways.

What data may doctors access after having obtained consent? Doctors may view a patient's personal data (d-ip), as well as the patient's monitoring data uploaded by mobile devices paired with that patient (d-im).

9.1.2 Device pairing

What does the "pairing" workflow look like? In order to authorize the mobile monitoring device, the patient powers on the device (6). After turning it on, the device initiates the communication procedure defined by the device profile of OAuth 2.0, connecting to the pre-configured central eHealth server via wireless communication in order to request "pairing" (7) and receiving a nonce. The central eHealth server communicates a pairing nonce to the mobile device (8). The device displays this nonce to the patient (9, assumed to be for that patient's eyes only). The patient then logs in at the central server and confirms pairing with the device by supplying the pairing nonce (10). Thus the patient has authorized the mobile monitoring device to upload monitoring data to the eHealth serve. From then onwards the central server associates the mobile device with the patient.

How is the mobile device authenticating after pairing? The server supplies the device with credentials via wireless communication (11). Whenever that device uploads (12) monitoring data to the central server supplying the credentials, the server associates this data with the patient the device is paired with. The mobile devices can send monitoring data to the server in real time, or in case there are any connection problems, retain the information internally until the connection is re-established and then send a burst containing all stored information.

9.1.3 New application features

The eHealth application scenario described in [i.8] has been expanded to include a number of new features. As described, the eHealth system seen in figure 41 can be used to gather patient information which is recorded by patient monitoring devices. In addition, the eHealth system provides a dashboard, which the doctors can use to monitor in real time the data gathered for the different patients, while ensuring the confidentiality of all such patient information.

After having obtained the patient's explicit consent (=authorization), the doctor can use the web front-end in order to access a dashboard containing detailed information about the individual patient. Such a patient consent the doctor can request through the dashboard, resulting in a two-factor authorization request to the specified patient. After the patient has accepted the request through both the web portal and a special mobile application, the doctor is granted special rights to access patient information. This allows a doctor to access the patient information including any mobile monitoring data associated with that patient and uploaded by mobile monitoring devices previously paired with (=authorized by) that patient.

9.2 Security testing approaches

9.2.1 Formalization

9.2.1.1 Entity overview

The formal model of the eHealth system knows five types of agents (see Listing 1): a central server; doctors, patients, smart mobiles, and mobile monitoring devices recording patient data.

```
types
    server < agent;     % central server
    doctor < agent;     % doctor
    patient < agent;    % patient
    smart < agent;  % smart mobile of patient
    device < agent;     % mobile monitoring device
```

Listing 1: Actor types in the eHealth system

The eHealth server (see Listing 2) knows doctor D, patient P, his smart mobile S and his mobile monitoring device M. Associated with S is a pre-shared secret between S and EH.

```
entity
    EHealth (D: doctor, % D may get infos after Consent and Pairing
        P: patient,     % EH can authenticate P
        S: smart,   % EH trusts registered Patient's Smart Mobiles
        M: device,  % EH can authenticate M
        Actor: server) {
```

Listing 2: eHealth actor header

Doctor D (see Listing 3) knows patient P, whose data he wants to access, and the eHealth server EH, where they are stored. D has a pre-existing trust relationship with EH, he is registered there. D also knows the mobile monitoring device, which is worn by P and uploading monitoring data to EH, where D wants to access them.

```
entity
    Doctor (Actor: doctor,
        P: patient,     % D personally knows Patient P
        M: device,  % M is given to P by D
        EH: server)     % D knows trustworthy EH
```

Listing 3: Doctor actor header

Patient P (see Listing 4) knows doctor D whom he trusts. He knows the mobile monitoring device M he wears. He has an account at eHealth server EH and can log in there to perform certain tasks. Of course P also knows his smart mobile S. But to keep the model efficient, no communication between P and S was explicitly modelled. Instead it was assumed that P is operating S, so the actions performed by S are in fact being triggered and thus authorized by P.

```
entity
    Patient (D: doctor, % D has to be a doctor always
        Actor: patient,
        M: device,  % M is given to P by D
        EH: server)     % P knows trustworthy EH
```

Listing 4: Patient actor header

Smart Mobile S (see Listing 5) is owned and operated by patient P. To simplify the model, it was assumed that everything that is known to P also can be known to S. For instance, S is aware, which doctor D is acceptable to P for giving consent to. S also is aware, which consent text would be acceptable to P (made known via a suitable fact).
S knows the eHealth server EH, with which it has a pre-shared secret (made known by an appropriate fact).

```
entity
    SmartMobile (D: doctor, % acceptable Doctor
        Actor: smart, EH: server) { % M knows trustworthy EH
```

Listing 5: Smart mobile actor header

The mobile monitoring device M (see Listing 6) is worn by P. Thus any information displayed by M is assumed to be only visible to P. P can operate M. M has a pre-existing relationship to EH. It can communicate securely with EH via a wireless connection, e.g. via GSM / UMTS.

```
entity
    MobileDev (P: patient, % P physically owns M
    Actor: device,
    EH: server) {   % M knows trustworthy EH
```

Listing 6: Mobile monitoring device actor header

9.2.1.2 Environment and sessions

The session entity instantiates the party involved and informs them about their fellow actors (see Listing 7). It assigns a smart mobile S to patient P and pre-shares the necessary secret SK between S and the central eHealth server by declaring an appropriate fact.

```
body {          % of entity Session
    P->owns(S);
    SK := fresh();
    S->has(SK);
    new Doctor (D, P, M EH);
    new EHealth (D, P, M, S, EH);
    new Patient (D, P, EH);
    new SmartMobile(D, S, EH);
    new MobileDev (P, M, EH);
}
```

Listing 7: eHealth session entity

In order to render the model manageable, the model is split into two parts. At the end of the consent workflow (see Listing 8) the doctor can access general personal data about the patient. For the pairing workflow (see Listing 9) it is assumed the consent workflow having already been performed. Thus the doctor, already in possession of the patient's consent, can access the mobile monitoring data after successful pairing.

For each sub-model, two identical sessions were in parallel to check for potential replay attacks and confusion between sessions.

```
body {          % of entity Environment
    any D P S. Session(D, P, S, eh)
        where (D != P) & (D != S) & (D != eh) &
            (P != S) & (P != eh) & (S != eh);
    any D P S. Session(D, P, S, eh)
        where (D != P) & (D != S) & (D != eh)
            & (P != S) & (P != eh) & (S != eh);}
```

Listing 8: eHealth sessions for consent

```
body {          % of entity Environment
any D P M. Session(D, P, M, eh)
        where (D != P) & (D != M) & (D != eh)
            & (P != eh) & (P != M) & (M != eh);
    any D P M. Session(D, P, M, eh)
        where (D != P) & (D != M) & (D != eh)
            & (P != eh) & (P != M) & (M != eh);}
```

Listing 9: eHealth sessions for pairing

9.2.1.3 Messages

The detailed message workflows for each of the two authorization workflows, consent and pairing, are given in the following.

Consent. The consent requested by a doctor D from a patient P was modelled via a two-factor authentication of P using the eHealth server's web interface and her smart mobile S as follows. Figure 43 gives an overview of the message flow.

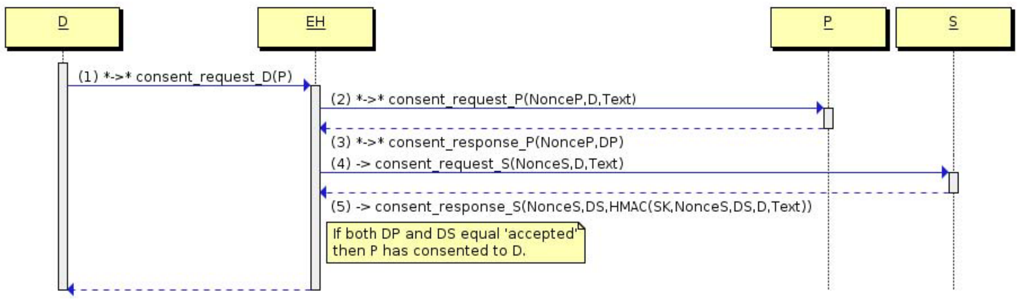

D
EH
P
S
(1) *->* consent_request_D(P)
(2) *->* consent_request_P(NonceP,D,Text)
(3) *->* consent_response_P(NonceP,DP)
(4) -> consent_request_S(NonceS,D,Text)
(5) -> consent_response_S(NonceS,DS,HMAC(SK,NonceS,DS,D,Text))
If both DP and DS equal 'accepted' then P has consented to D.

Figure 43: Message flow: patient consent

D has to obtain P's explicit consent to access P's personal data stored as the eHealth server. For this D sends an appropriate request to the eHealth server (1), indicating patient P whose consent is sought. On receiving such a request from doctor D for consent by P, the eHealth server initiates the consent process.

First the eHealth server (see Listing 10) sends two consent request messages, one to P (2) and the other one to the Smart Mobile S registered as belonging to P (3). The eHealth server and S have a pre-shared secret SK. Included in the request is the text of the consent and information which D is asking this consent. P will only give consent, if both the text of the consent and D are acceptable to P. Only if P grants consent both at his account at the eHealth server using his browser and via his smart mobile S, then the eHealth server accepts the consent as having been granted.

```
select {
% -- (1) Doctor may request Patient Consent
    on ((?D *->* Actor: consent_request_D(?P)
        & !(?P->hasConsented(?D))) & ?P->owns(?S)): % (1)
{
Text := fresh();
Text->isAcceptable;
NonceP := fresh();
Actor *->* P: consent_request_P(NonceP,D,Text); % (2)
% It is assumed that the patient has securely logged in at the eHealth server
NonceS := fresh();Actor -> S: consent_request_S(NonceS,D,Text); % (4)
}
}
```

Listing 10: eHealth server: consent request

As indicated by the starred arrow *->* (see messages 1 and 2), it is assumed that there is already a secure connection between EH and D as well as EH; and P. This is usually achieved by a HTTPS web session where P has successfully logged in to EH's web server using his browser. For the connection of EH with S it is not assumed any channel protection at all, as indicated by the simple arrow -> (see message 4). For both requests, a nonce is used as unique identifier for the matching response.

Then the eHealth server, as shown in Listing 11 awaits response from both P (3) and her smart mobile S (5). If both are positive and the HMAC used to authenticate the response from S using the pre-shared secret SK, it can conclude that the consent has been granted, which it is modelled by the introduction of the fact P->hasConsented(D):

```
select {
on (P *->* Actor: consent_response_P(P_consent:(NonceP),accepted) % (3)
    & S->has(?SK) &
    S -> Actor: consent_response_S(S_consent:(NonceS),accepted, % (5)
        hmac(secret_EH_S:(?SK),NonceS.accepted.D.Text))):
    {
        P->hasConsented(D);
    }
}
```

Listing 11: eHealth server: consent responses

After this, as shown in Listing 12, the doctor can successfully request personal Information from the eHealth server (d-ip). The server will check for the consent, before forwarding the patient's personal information to the doctor (d-ip-r).

```
select { on (D *->* Actor: info_request_D_P(?Nonce, P) % (d-ip)
    & P->hasConsented(D) ):
    {
        InfoP := fresh(); % simulates P's personal information
        Actor *->* D: info_response_D_P (Nonce, P, secret_EH_D:(InfoP));
    }
}
```

Listing 12: eHealth server: information delivery

Pairing Via device pairing, a patient authorizes a mobile monitoring device to upload monitoring data to the central eHealth server on behalf of that patient.

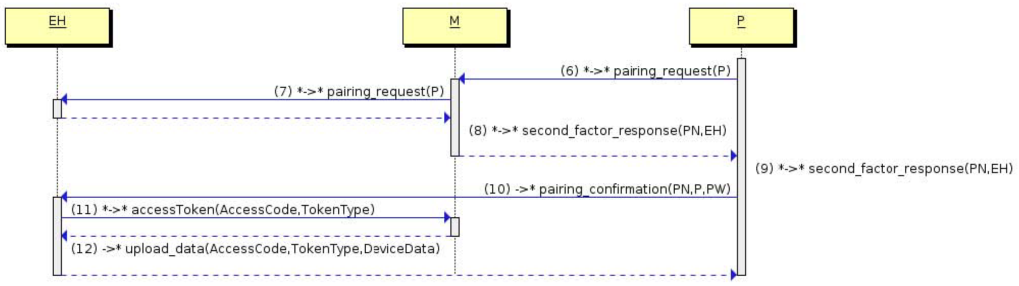

EH
M
P
(6) *->* pairing_request(P)
(7) *->* pairing_request(P)
(8) *->* second_factor_response(PN,EH)
(9) *->* second_factor_response(PN,EH)
(10) ->* pairing_confirmation(PN,P,PW)
(11) *->* accessToken(AccessCode,TokenType)
(12) ->* upload_data(AccessCode,TokenType,DeviceData)

Figure 43a: Message flow: device pairing

A mobile monitoring device worn by the patient has to be entitled to upload that patient's monitoring data to the eHealth server. This is done by the patient pairing with the mobile monitoring device, thus assuring the eHealth server of the required patient consent. The procedure is sketched in figure 41.

The patient P needs to trigger the mobile monitoring device M (6) to make it connect to the eHealth server EH to obtain the pairing nonce (see Listing 13). M connects to the pre-configured EH (7), which returns the pairing nonce (8), which M forwards (displays) to P (9).

```
% (6) P turns on M, (7) M contacts EH
% EH knows M's public device key and can authenticate M
%

?P *->* Actor: pairing_request(?P);
Actor *->* EH: pairing_request(P);
% (8) EH returns a nonce for P via M
% (9) M forwards this confirmation request to P
%
EH *->* Actor: second_factor_request(secret_EH_P_M:(?Nonce),EH);
Actor *->* P: second_factor_request(Nonce,EH);
```

Listing 13: Mobile device: pairing initiation

Once M displays (=sends) the pairing nonce to P (9), P logs in at his account at the central eHealth server EH and submits the pairing nonce (10, see Listing 14). Then EH forwards suitable credentials to M (11), which in future can use these credentials to authenticate to EH. M now can successfully upload monitoring data to EH (12), has it has been authorized to do so by P. All data uploaded by M are associated with P at EH.

```
} select {
    % (10) Patient, logged in, confirms the Nonce message to EH
    % (11) EH sends Access Code to M
    on (P *->* Actor: pairing_confirmation(EH_P_pair:(Nonce)) &
        existsAssignmentRequest(?M,secret_EH_P_M:(Nonce))
        & !P->hasPaired(?M) & !?M->hasCredentials(?,?)):
    {
        retract existsAssignmentRequest(M,Nonce);
        P->hasPaired(M);
        secret_EH_M:(AccessCode) := fresh();
        secret_EH_M:(TokenType) := fresh();
        M->hasCredentials(AccessCode,TokenType);
        Actor *->* M: accessToken_response(AccessCode,TokenType);
    }
} select {
    % (12) Devices may send Data about the Patient they are assigned to
    on (M ->* Actor:
        upload_data(EH_M_upload:(?AccessCode),
        ?TokenType,secret_DeviceData:(?DeviceData))
        & P->hasPaired(M) & M->hasCredentials(?AccessCode,?TokenType)):
    {
    M->hasUploadedData(P);
        % DeviceData may become part of the data retrievable by P from EH.
    }
} % select
```

Listing 14: eHealth server: pairing finalization

After a mobile monitoring device has successfully been paired with a patient, a doctor with prior patient consent can access the confidential monitoring data uploaded by that mobile device for that patient (d-im, d-im-r, see Listing 15).

```
select {
    on (D *->* Actor: info_request_D_P_M(?Nonce, P, M) % (d-im)
        & P->hasConsented(D) & P->hasPaired(M) ):
    {
        InfoM := fresh();
        Actor *->* D: info_response_D_P_M (Nonce, P, M, secret_EH_D:(InfoM));
    }
} % select
```

Listing 15: eHealth server: monitoring information delivery

9.2.1.4 Goals

Authenticity. The main goals of two-factor patient consent are that the consent responses by both the patient P herself (via EH's web interface) and by her smart mobile S are authentic and cannot be re-played (see P_consent and S_consent in Listing 16). In the pairing process, the eHealth server authenticates the patient on the successful return of the pairing nonce (see EH_P_pair), assuming, that only the patient physically possessing the mobile device can read the pairing nonce on the device's display. A successfully paired device on subsequent uploads is authenticated by the eHealth server via special device credentials (see EH_M_upload).

```
goals
P_consent:(_) P *->> EH;    % P's response has to be authentic and fresh
S_consent:(_) S *->> EH;    % S's response has to be authentic and fresh
EH_P_pair :(_) P *-> EH;    % EH authenticates P on Nonce
EH_M_upload :(_) M *-> EH;  % EH authenticates M on AccessCode
```

Listing 16: eHealth authenticity goals

Secrecy. For the authenticity protection of response by the smart mobile S in the consent workflow an HMAC is used with a pre-shared symmetric key SK between EH, P, and its S. Its secrecy is given as a secondary goal (see secret_EH_S in Listing 17). In the pairing workflow, the pairing nonce is to be kept secret between three parties, the eHealth server, the mobile monitoring device and the patient (see secret_EH_P_M). After a mobile device has successfully been paired with a patient, it receives and has to keep secret its personal access credentials (see secret_EH_M). The monitoring data uploaded by the device has to be kept confidential by the eHealth server (see secret_DeviceData). For reasons of simplicity secrecy is required between the eHealth server and the mobile device only, and do neither include patients, nor doctors. Of course the doctor has to keep the patient's personal secret (see secret_EH_D). For reasons of simplicity secrecy is required between the eHealth server and the doctor only, and neither includes the patient nor the mobile monitoring device.

```
secret_EH_S :(_) {EH,S};    % the secret key for HMAC, shared among EH and S,
                    % logically also by P, but this is ignored here.
secret_EH_P_M :(_) {EH,M,P};    % the pairing confirmation nonce
secret_EH_M :(_) {EH,M};    % AccessCode and TokenType
secret_DeviceData :(_) {EH,M}; % the data about P uploaded by M
secret_EH_D :(_) {EH,D};
% Personal Data about P, shared among EH and D,
% logically also by P and maybe M, but this is
% ignored here.
```

Listing 17: eHealth secrecy goals

Authorization. A doctor may only access personal information regarding a patient, if the patient has given prior consent (see Listing 18). This is monitored by a suitable LTL formula. Once the eHealth server has assured the consent process has finished appropriately, it raises a fact that the patient has consented to the doctor. Once the doctor receives the patient's personal data, he raises a fact to note down this event. If the doctor manages to obtain that information without prior patient consent, the LTL formula would be violated.

```
Prior_consent: forall D P.
[](D->hasReceivedPInfo(P) => (P->hasConsented(D)));
% Doctor may only read P's personal Information, if the Patient
% has consented beforehand to be this doctor's patient.
% NOTE: Strictly "<->" should be used before "hasConsented", but Tools
% have Problems with "<->".
```

Listing 18: Patient consent necessity

A mobile monitoring device may only upload monitoring data about a patient to the central eHealth server, if it had been authorized to do so by the patient. This is monitored by a suitable LTL formula (see Listing 19). The eHealth server itself checks the proper authorization of a mobile device that requests uploading of monitoring data for a given patient (see Listing 20). It records a fact about the successful upload. If however a mobile monitoring device would be able to upload data prior to pairing, the LTL formula would be violated.

```
Prior_pairing: forall P M. [](M->hasUploadedData(P) => (P->hasPaired(M)));
% M may only upload data about P to EH, once it has been
% authorized to do so by the patient via the pairing process.
```

Listing 19: eHealth server: device pairing necessity

If the doctor wants to obtain access to the mobile monitoring data of that patient, the patient should have not only given prior general consent to that doctor, but he also should have authorized the mobile monitoring device to upload these data. This is monitored by a suitable LTL formula. The eHealth server both raises appropriate facts once patient consent has been obtained and the patient has paired with (=authorized) the mobile device. The doctor in turn raises a fact once he has obtained the confidential monitoring data. If the doctor would manage to obtain these data without prior patient consent or before the patient had paired with the device, then the LTL formula would be violated.

```
Prior_consent_and_pairing: forall DPM. [](D->hasReceivedMInfo(P,M) => (P->hasConsented(D) & (P-
>hasPaired(M))));
% D may only read P's Data uploaded by M, if the patient has consented
% beforehand to be this doctor's patient and if the patient beforehand
% has paired with that mobile device.
% NOTE: Strictly "<->" should be used before "hasConsented" and "hasPaired",
% but Tools have Problems with "<->".
% Warning regarding 'Goal "Prior_consent..." is active ...' may be ignored.
```

Listing 20: Patient consent and pairing necessity

Non-Goals. Privacy protection is a main focus of the overall application. It can, however, not directly be focused in the formalization.

As with the semantics of the secrecy goals, privacy support is not supported well in the modelling language, and the model checkers cannot detect all privacy violations this way, particularly regarding those among otherwise lawful participants (excluding the intruder).

It was attempted to provide some workaround by combining authentication of entities and authorization via suitable LTL formulas in the workflows. This however implies an instrumentation of the model source code with manually set and retracted facts.

Still the aim was not to perfect protection of all potentially privacy relevant data. The aim was not to protect user privacy completely in this scenario. The objective was to explore the effectiveness of the security controls regarding the IT Security properties of authenticity, authorization and integrity. If an attempt had been made to require complete encryption and authentication for every privacy sensitive piece of information that is being transmitted, many potential attack vectors would be missed, whose detection is desirable. Thus the present model would be much more vulnerable regarding privacy than it is necessary.

Non-repudiation is out of scope for the eHealth model. The security mechanisms used mostly are of the symmetric kind, the actions are not signed and/or witnessed. Also plain accountability is not the focus for the eHealth model, thus no logs are kept. However all relevant entities are carefully authenticated and authorized, so basically, appropriate logs could be kept and actions could be matched to the parties that initiated them.

9.2.2 Analysis results using a model checker

Violations of the given goals were looked for using the model checker CL-AtSe. On the final version of the model, no attacks were found.

There was a spurious attack on a previous version of the model, which was due to a mismatch in the assignment of pre-shared secrets SK and smart mobiles S that occurred when two sessions ran in parallel for the same doctor with two different S. This problem was solved using facts of the form S->has(SK).

9.2.3 Technical details

The eHealth server consists of three different components which are able to interact with each other as well as with external providers:

1) eHealth Web Front-End

2) Device management platform

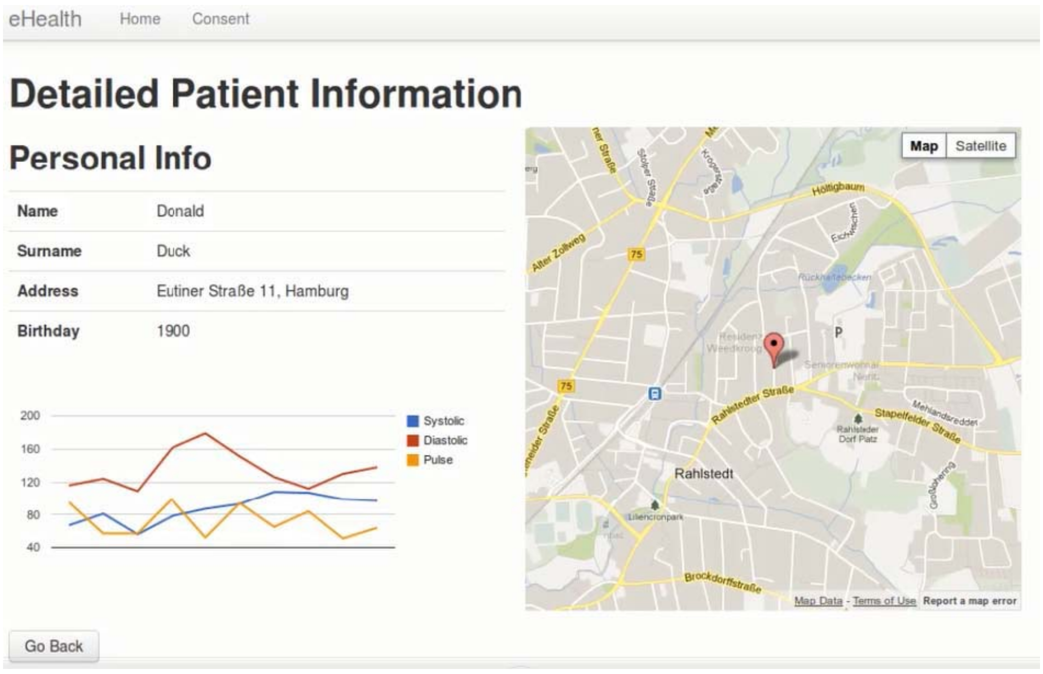
3) Two-Factor Authentication Service

In the described eHealth scenario, the server side of all three components runs on top of a Tomcat server and is located within the same trust zone. Additionally, all three components directly access databases which are also located within the same trust zone. This trust zone provides a single web interface (over HTTPS) for all incoming communication.

9.2.3.1 eHealth web front-end

The eHealth Web Front-End provides a web interface for doctors and patients. It is able to use other components, such as the device management platform or the two-factor authentication service, in order to present both with a user-friendly dashboard with available information.

After being authorized by the user using the two-factor authentication service, doctors are able to monitor in real time the data from the patient such as heart rate or blood pressure. Additionally, external information such as the registered patient location can be retrieved and presented using external providers. This capability is demonstrated in the prototype through the Google maps API.



Detailed Patient Information

Personal Info

Name	Donald
Surname	Duck
Address	Eutiner Straße 11, Hamburg
Birthday	1900

Figure 43b: The eHealth web front-end

9.2.3.2 Device management platform

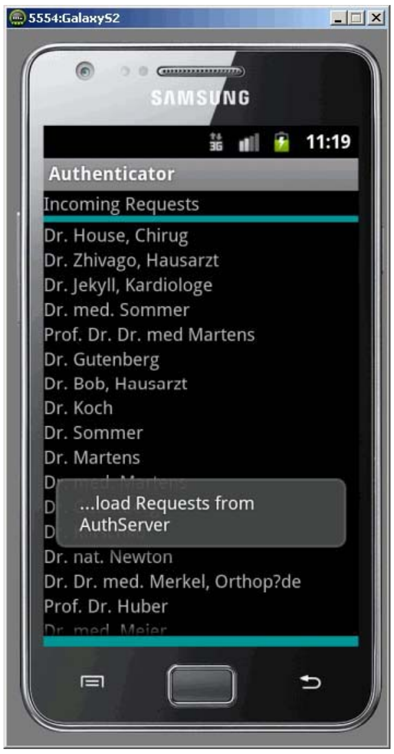
The device management platform is a central component of the eHealth system which can be used in order to manage medical devices for patient monitoring. Besides providing functionality for setting up new devices, a system is available for assigning them to users based on the OAuth 2.0 Device profile. Additionally, the device management platform provides an interface for the gathering of data from those patient monitoring devices as well as interfaces for retrieving this information after proper authentication.

These functions are implemented as RESTful services on a Tomcat server. Therefore the platform can be easily integrated in a wide variety of different scenarios. A primitive web interface that calls on those web services is also available as part of the device management platform, although it is instead recommended to integrate the functionality directly in the web application, as can be seen in the eHealth Web Front-End.

9.2.3.3 Two-factor authentication service

The Two-Factor Authentication Service consists of two components, a RESTful web service running on Tomcat and an Android™ application, and can be called upon in order to request to perform an additional authentication through the use of mobile devices. By sending a request to the web service, the eHealth web front-end is able to ask the two-factor authentication service to send an authentication request to a specific user. This user is then able to either confirm, or deny the request using the Android™ application.

All communication with between the two components is initiated by the Android™ application, and is performed by sending JSON objects through RESTful web services. As in the case of the device management platform, this design principle allows a flexible integration of this component in larger systems. While the communication itself is performed through HTTPS, a signature mechanism has been implemented in order to protect against Man-in-the-Middle-style attacks.



Incoming Requests

Dr. House, Chirug
Dr. Zhivago, Hausarzt
Dr. Jekyll, Kardiologe
Dr. med. Sommer
Prof. Dr. Dr. med Martens
Dr. Gutenberg
Dr. Bob, Hausarzt
Dr. Koch
Dr. Sommer
Dr. Martens
D
D
D
Dr. nat. Newton
Dr. Dr. med. Merkel, Orthop?de
Prof. Dr. Huber
Dr. med. Meier

...load Requests from AuthServer

Figure 44: The Android™ application

9.2.4 Improvements of the security model

The previous version eHealth security model comprised only the pairing workflow. The consent workflow is new to the eHealth model in this version. It was necessary to model the consent workflow to match the actual implementation of the eHealth system, as this process was included in the implementation of the eHealth server.

The former model implemented a lot of the details, flexibility and functionality of the eHealth server. For instance modelled patients and doctors were explicitly by logging in at and out from the eHealth server using their passwords. A session management was also provided for each of the participants and password changing facilities. The eHealth server was modelled as real server, operating in a loop and accepting the various requests in a more or less arbitrary fashion. This resulted in many messages and in the model being rather inefficient and hard to check, especially with more than one session.

In this version the eHealth server was simplified. It only accepts the predefined workflows in the predefined order and does not model any login and logout procedures not germane to the actual workflow in focus. Instead of an explicit login, known parties are authenticated with a simple *->*, and those parties are assumed to be able to establish a secure communication channel. Session management is replaced by a simple nonce for request-response-pairing. The two sub-workflows, pairing and consent, have been split into two separate sub-models, where the pairing workflow assumes the consent workflow already having taken place and vice versa. This results in an efficient and more easily manageable model so that model checkers can handle two simultaneous sessions.

The security supervision of the eHealth model was improved by introducing suitable LTL formulas checking the authorization policies of the eHealth system.

9.2.5 Considered security properties and vulnerabilities

The following security properties and vulnerabilities are considered in the experiments discussed within the present document.

9.2.5.1 Security properties

The focus is on verifying the security properties of:

1) Authenticity
2) Authorization
3) Integrity

Privacy: The aim is not primarily to protect user privacy in this scenario. The objective is to explore the effectiveness of the security controls with respect to protecting the IT-Security properties of authenticity, authorization and integrity. If encryption was required for every privacy sensitive piece of information that is being transmitted, many attack vectors would be missed, whose detection is desirable. However by using a strict and supervised authorization policy, personal patient data are protected to a certain extent.

Accountability: Non-repudiation is out of scope for the eHealth model. The security mechanisms used mostly are of the symmetric kind. Also plain accountability is not in focus for the eHealth model, thus no logs are kept.

9.2.5.2 Vulnerabilities

Similarly as for InfoBase, vulnerabilities were considered from a representative set of the most common low-level security vulnerabilities in web-applications (see for instance OWASP Top 10 https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project) and corresponding to a refinement of the analysis performed in [i.11]. The vulnerabilities considered are:

1) Cross-site Scripting (XSS)
2) SQL Injection
3) Password brute-forcing
4) Cross-Site Request Forgery (CSRF)
5) File Enumeration
6) Path traversal

9.3 Results by applying the VERA tool

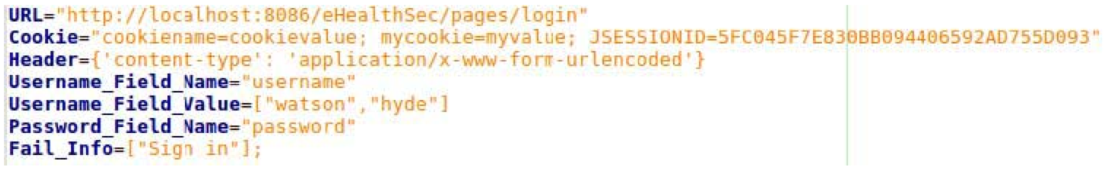
In the following the experiments are described on applying Vera on the eHealth application. Due to the nature of the attacker models so far available for Vera, the efforts concentrated on testing the Web application that allows patients and doctors to interact with the system and not the protocols with the mobile devices. The doctors (usernames: "watson" and "doctor") and the patients (usernames: "hyde", "duck" and "mouse") can interact with the web application after they had logged in. Vulnerabilities were looked for with Vera on the views of "watson", a doctor, and "hyde", a patient. As described in previous clauses, the doctor is able submit a consent request and the patient can accept or decline it. Both have the rights to edit their personal information.

9.3.1 Password brute force

Brute-force attack uses a given login name and tries to login in with a commonly used password pool in the login interface of eHealth. The library of known user accounts is:

- watson (doctor)
- hyde (patient)

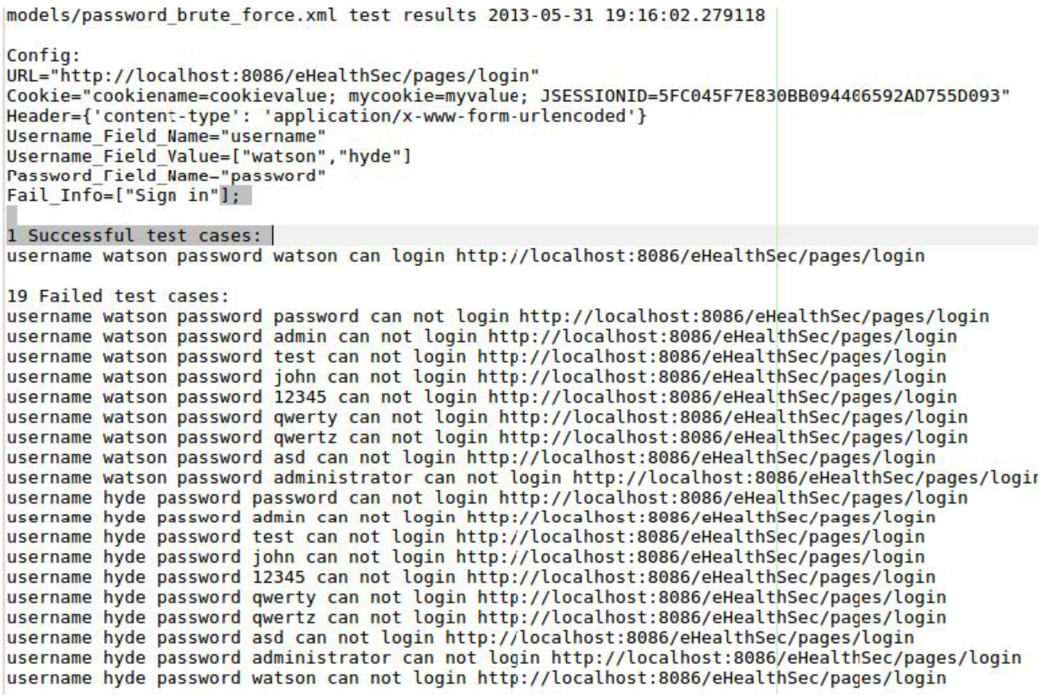
With this information and other system details, the resulting configuration file in Vera is depicted in figure 45.



```
URL="http://localhost:8086/eHealthSec/pages/login"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=5FC045F7E830BB094406592AD755D093"
Header={'content-type': 'application/x-www-form-urlencoded'}
Username_Field_Name="username"
Username_Field_Value=["watson","hyde"]
Password_Field_Name="password"
Fail_Info=["Sign in"];
```

Figure 45: Configuration for password brute force

A partial view of the results of brute forcing with a set of commonly used passwords is depicted in figure 46. Vera was able to find the password of the patient "watson".



```
models/password_brute_force.xml test results 2013-05-31 19:16:02.279118

Config:
URL="http://localhost:8086/eHealthSec/pages/login"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=5FC045F7E830BB094406592AD755D093"
Header={'content-type': 'application/x-www-form-urlencoded'}
Username_Field_Name="username"
Username_Field_Value=["watson","hyde"]
Password_Field_Name="password"
Fail_Info=["Sign in"];

1 Successful test cases:
username watson password watson can login http://localhost:8086/eHealthSec/pages/login

19 Failed test cases:
username watson password password can not login http://localhost:8086/eHealthSec/pages/login
username watson password admin can not login http://localhost:8086/eHealthSec/pages/login
username watson password test can not login http://localhost:8086/eHealthSec/pages/login
username watson password john can not login http://localhost:8086/eHealthSec/pages/login
username watson password 12345 can not login http://localhost:8086/eHealthSec/pages/login
username watson password qwerty can not login http://localhost:8086/eHealthSec/pages/login
username watson password qwertz can not login http://localhost:8086/eHealthSec/pages/login
username watson password asd can not login http://localhost:8086/eHealthSec/pages/login
username watson password administrator can not login http://localhost:8086/eHealthSec/pages/logir
username hyde password password can not login http://localhost:8086/eHealthSec/pages/login
username hyde password admin can not login http://localhost:8086/eHealthSec/pages/login
username hyde password test can not login http://localhost:8086/eHealthSec/pages/login
username hyde password john can not login http://localhost:8086/eHealthSec/pages/login
username hyde password 12345 can not login http://localhost:8086/eHealthSec/pages/login
username hyde password qwerty can not login http://localhost:8086/eHealthSec/pages/login
username hyde password qwertz can not login http://localhost:8086/eHealthSec/pages/login
username hyde password asd can not login http://localhost:8086/eHealthSec/pages/login
username hyde password administrator can not login http://localhost:8086/eHealthSec/pages/login
username hyde password watson can not login http://localhost:8086/eHealthSec/pages/login
```

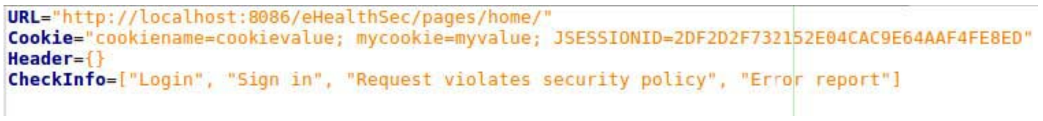
Figure 46: One successful test case for user watson

9.3.2 File enumeration

To be able to distinguish between different attack scenarios, the tests run for configuration files, backups, administrative interfaces and other hidden files and functionalities in the case of logged users (patients or doctors) and not authenticated users.

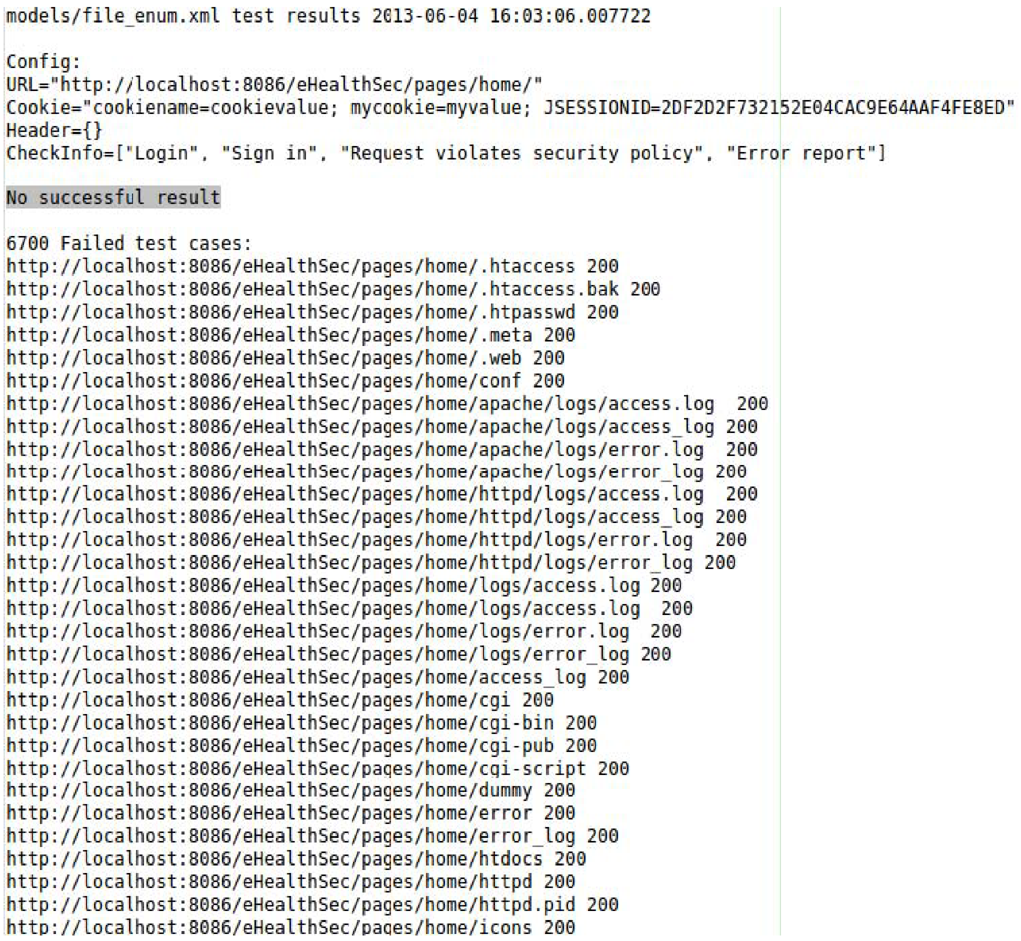
To test for logged users, the session ID of a logged user should be set in the configuration file has to be updated to the current one. It can be retrieved for instance by intercepting a package between the browser and the server (by using a proxy like Burp).

For instance in figures 47 and 48 excerpts are shown of the configuration file and the results of a file enumeration attack for authenticated users.



```
URL="http://localhost:8086/eHealthSec/pages/home/"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=2DF2D2F732152E04CAC9E64AAF4FE8ED"
Header={}
CheckInfo=["Login", "Sign in", "Request violates security policy", "Error report"]
```

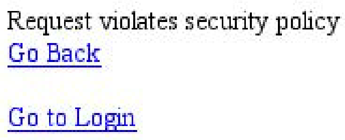
Figure 47: Configuration for file enumeration with cookies



```
models/file_enum.xml test results 2013-06-04 16:03:06.007722

Config:
URL="http://localhost:8086/eHealthSec/pages/home/"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=2DF2D2F732152E04CAC9E64AAF4FE8ED"
Header={}
CheckInfo=["Login", "Sign in", "Request violates security policy", "Error report"]

No successful result

6700 Failed test cases:
http://localhost:8086/eHealthSec/pages/home/.htaccess 200
http://localhost:8086/eHealthSec/pages/home/.htaccess.bak 200
http://localhost:8086/eHealthSec/pages/home/.htpasswd 200
http://localhost:8086/eHealthSec/pages/home/.meta 200
http://localhost:8086/eHealthSec/pages/home/.web 200
http://localhost:8086/eHealthSec/pages/home/conf 200
http://localhost:8086/eHealthSec/pages/home/apache/logs/access.log  200
http://localhost:8086/eHealthSec/pages/home/apache/logs/access_log 200
http://localhost:8086/eHealthSec/pages/home/apache/logs/error.log  200
http://localhost:8086/eHealthSec/pages/home/apache/logs/error_log 200
http://localhost:8086/eHealthSec/pages/home/httpd/logs/access.log  200
http://localhost:8086/eHealthSec/pages/home/httpd/logs/access_log 200
http://localhost:8086/eHealthSec/pages/home/httpd/logs/error.log  200
http://localhost:8086/eHealthSec/pages/home/httpd/logs/error_log 200
http://localhost:8086/eHealthSec/pages/home/logs/access.log 200
http://localhost:8086/eHealthSec/pages/home/logs/access.log  200
http://localhost:8086/eHealthSec/pages/home/logs/error.log  200
http://localhost:8086/eHealthSec/pages/home/logs/error_log 200
http://localhost:8086/eHealthSec/pages/home/access_log 200
http://localhost:8086/eHealthSec/pages/home/cgi 200
http://localhost:8086/eHealthSec/pages/home/cgi-bin 200
http://localhost:8086/eHealthSec/pages/home/cgi-pub 200
http://localhost:8086/eHealthSec/pages/home/cgi-script 200
http://localhost:8086/eHealthSec/pages/home/dummy 200
http://localhost:8086/eHealthSec/pages/home/error 200
http://localhost:8086/eHealthSec/pages/home/error_log 200
http://localhost:8086/eHealthSec/pages/home/htdocs 200
http://localhost:8086/eHealthSec/pages/home/httpd 200
http://localhost:8086/eHealthSec/pages/home/httpd.pid 200
http://localhost:8086/eHealthSec/pages/home/icons 200
```

Figure 48: Results for file enumeration with cookies

In both cases, a list of possible files was found, but it was not possible to find hidden files (the files were false positives, see figure 49).



Request violates security policy
Go Back

Go to Login

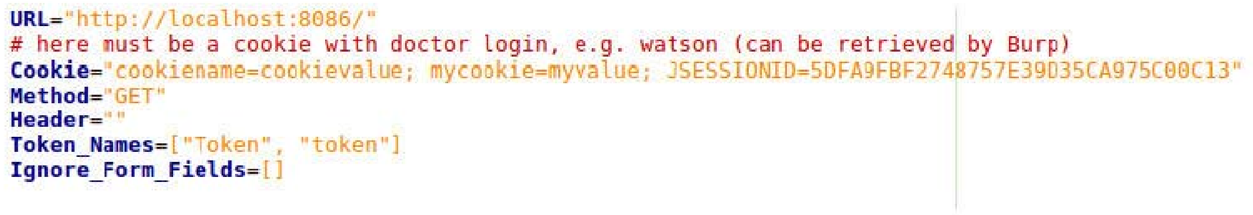
Figure 49: eHealth response to the file enumeration attack

9.3.3 CSRF token checking

To check for the presence of CSRF tokens, Vera was applied on the following pages of the eHealth application:

```
Instantiation library
1 IO=[
2 "eHealthSec/pages/login",
3 "eHealthSec/pages/home",
4 "eHealthSec/pages/edit",
5 "eHealthSec/pages/consent"
6 ]
```

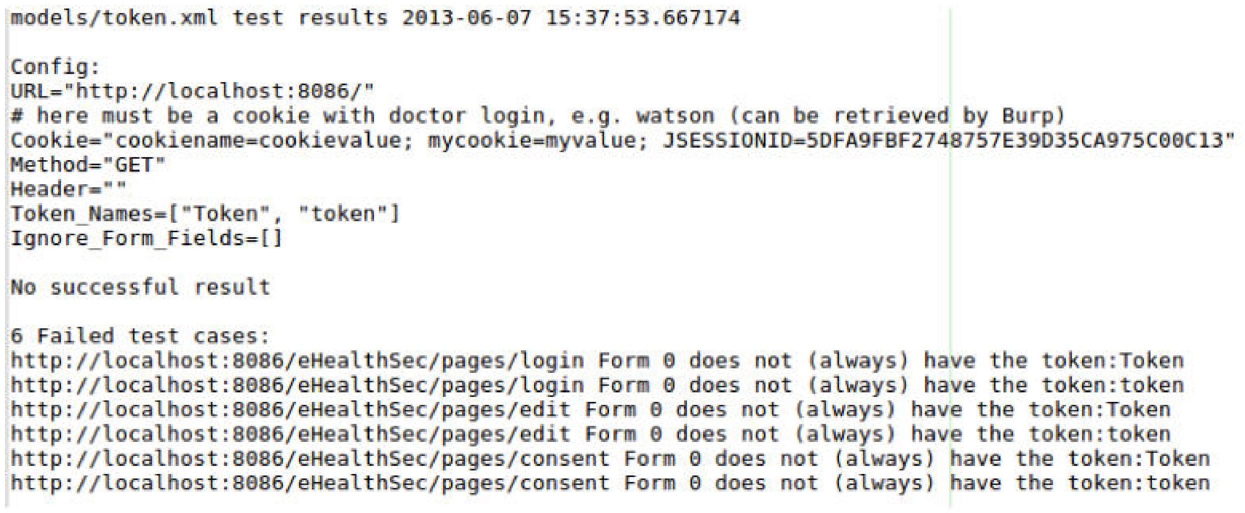
In this case the session ID of logged users (doctors and patients) was also considered to check if there were noticeable differences, for instance figure 50 depicts the configuration of Vera using a doctor's session and figure 51 the results of the test.



```
URL="http://localhost:8086/"
# here must be a cookie with doctor login, e.g. watson (can be retrieved by Burp)
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=5DFA9FBF2748757E39D35CA975C00C13"
Method="GET"
Header=""
Token_Names=["Token", "token"]
Ignore_Form_Fields=[]
```

Figure 50: Configuration for token, session ID of the doctor

In both cases (for doctors and patients), no CSRF token was found:

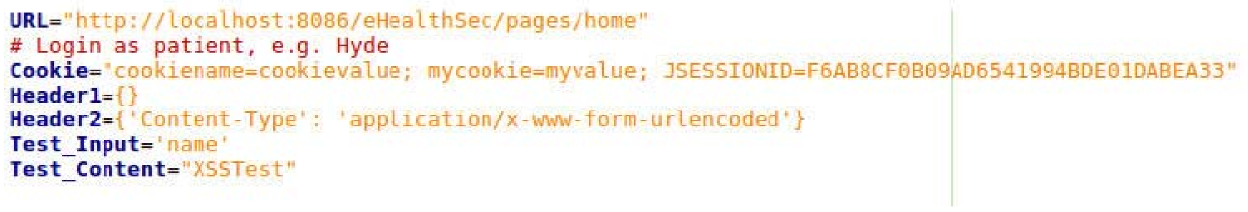


```
models/token.xml test results 2013-06-07 15:37:53.667174

Config:
URL="http://localhost:8086/"
# here must be a cookie with doctor login, e.g. watson (can be retrieved by Burp)
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=5DFA9FBF2748757E39D35CA975C00C13"
Method="GET"
Header=""
Token_Names=["Token", "token"]
Ignore_Form_Fields=[]

No successful result

6 Failed test cases:
http://localhost:8086/eHealthSec/pages/login Form 0 does not (always) have the token:Token
http://localhost:8086/eHealthSec/pages/login Form 0 does not (always) have the token:token
http://localhost:8086/eHealthSec/pages/edit Form 0 does not (always) have the token:Token
http://localhost:8086/eHealthSec/pages/edit Form 0 does not (always) have the token:token
http://localhost:8086/eHealthSec/pages/consent Form 0 does not (always) have the token:Token
http://localhost:8086/eHealthSec/pages/consent Form 0 does not (always) have the token:token
```

Figure 51: No successful result, session ID of the doctor

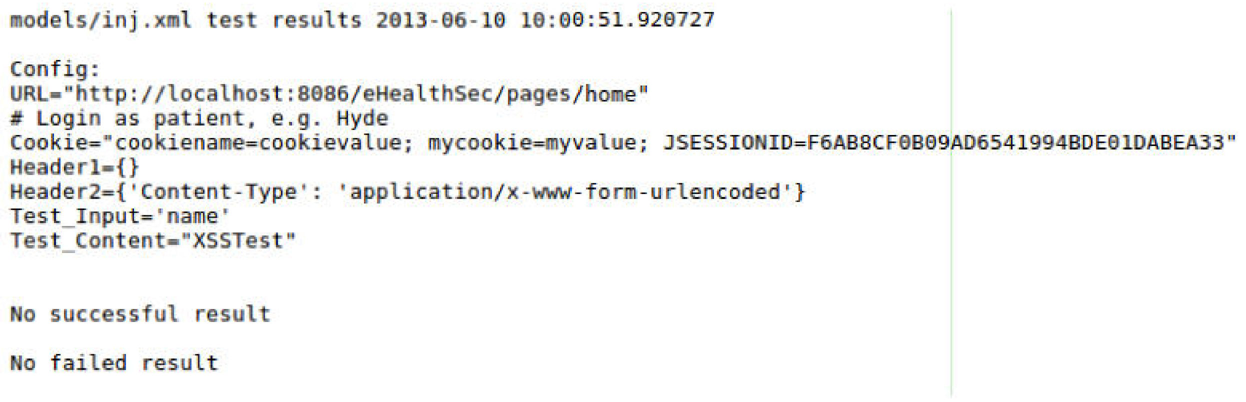
9.3.4 SQL injection

To test for SQL injections, different user roles were also used, to maximize the attack surface. For instance the configuration file for a patient is reproduced in figure 52 and the obtained results in figure 53. It was not possible to find any SQL injection vulnerability in the eHealth application.



```
URL="http://localhost:8086/eHealthSec/pages/home"
# Login as patient, e.g. Hyde
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=F6AB8CF0B09AD6541994BDE01DABEA33"
Header1={}
Header2={'Content-Type': 'application/x-www-form-urlencoded'}
Test_Input='name'
Test_Content="XSSTest"
```

Figure 52: Configuration file for SQL for patient's login

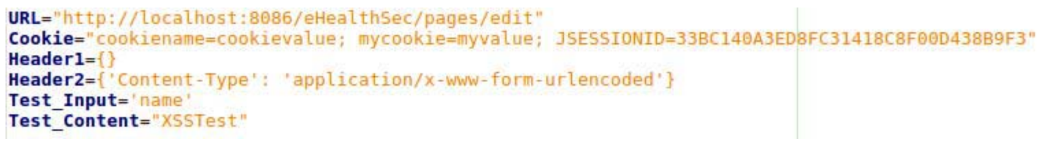


```
models/inj.xml test results 2013-06-10 10:00:51.920727

Config:
URL="http://localhost:8086/eHealthSec/pages/home"
# Login as patient, e.g. Hyde
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=F6AB8CF0B09AD6541994BDE01DABEA33"
Header1={}
Header2={'Content-Type': 'application/x-www-form-urlencoded'}
Test_Input='name'
Test_Content="XSSTest"


No successful result

No failed result
```

Figure 53: No successful/failed results

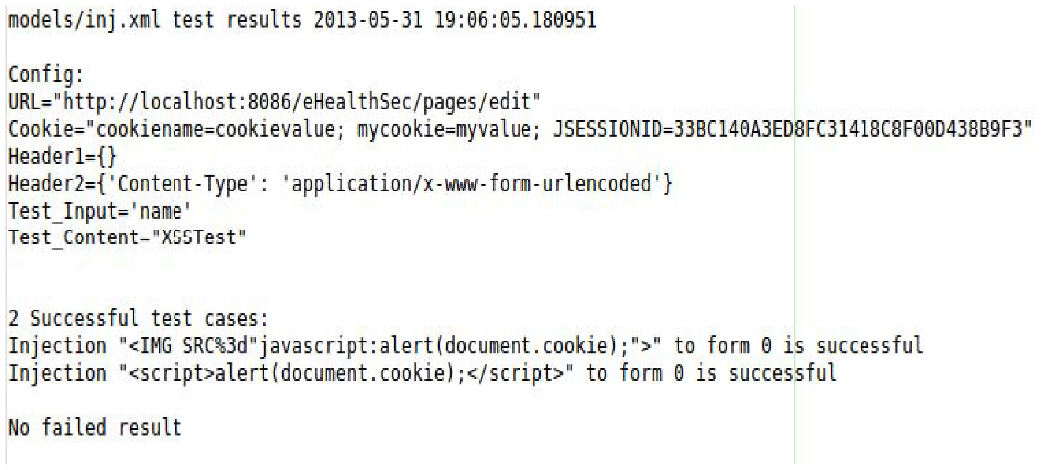
9.3.5 XSS injection

The testing efforts were focused in the edit (to edit personal data) and consent (to issue new consent requests) sites of the eHealth application. It was possible to find two different kinds of reflected XSS injection in the edit site, as depicted in figure 54 with the configuration illustrated in figure 55.



```
URL="http://localhost:8086/eHealthSec/pages/edit"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=33BC140A3ED8FC31418C8F00D438B9F3"
Header1={}
Header2={'Content-Type': 'application/x-www-form-urlencoded'}
Test_Input='name'
Test_Content="XSSTest"
```

Figure 54: Configuration file for checking the "edit function" of eHealth



```
models/inj.xml test results 2013-05-31 19:06:05.180951

Config:
URL="http://localhost:8086/eHealthSec/pages/edit"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=33BC140A3ED8FC31418C8F00D438B9F3"
Header1={}
Header2={'Content-Type': 'application/x-www-form-urlencoded'}
Test_Input='name'
Test_Content="XSSTest"


2 Successful test cases:
Injection "<IMG SRC%3d"javascript:alert(document.cookie);">" to form 0 is successful
Injection "<script>alert(document.cookie);</script>" to form 0 is successful

No failed result
```

Figure 55: Successful results

9.3.6 Path traversal attack

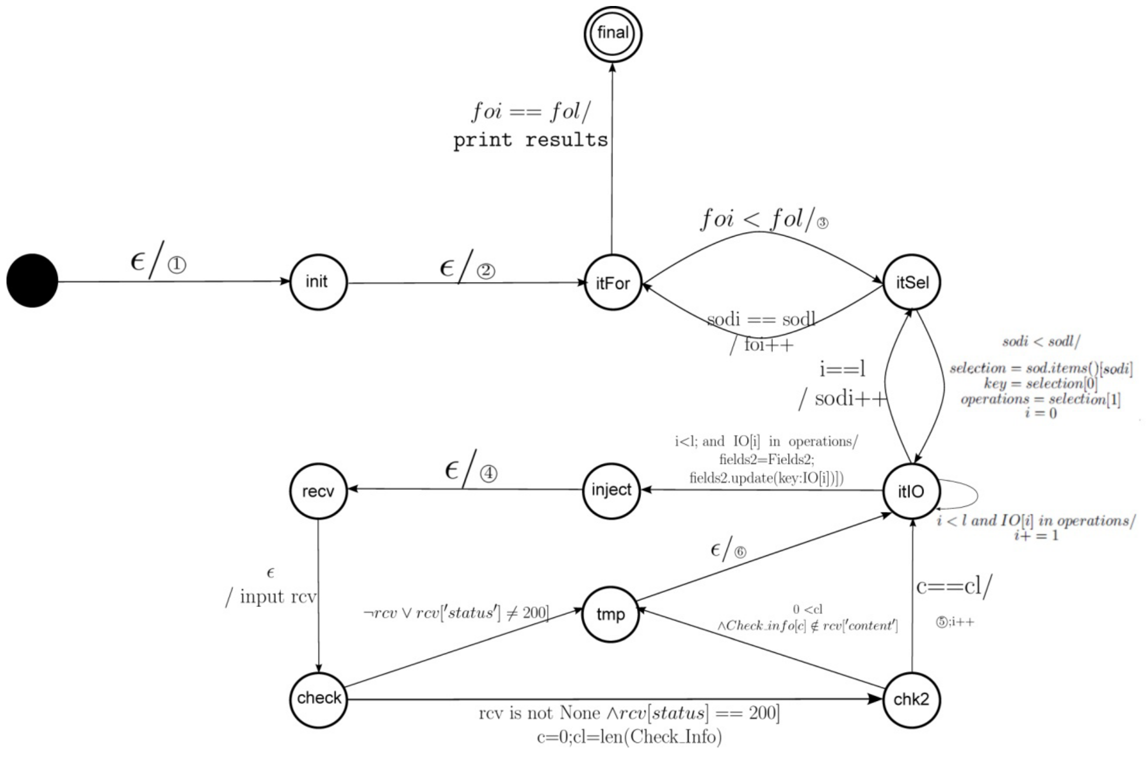
The path traversal attack was applied to all available sites of the eHealth application, but it was not possible to detect any such vulnerability.

9.3.7 Access control

To be able to test for access control violations a new attacker model was defined in Vera. This attacker tries to iterate the ID of a given parameter to exploit poorly implemented (or non-existent) access control mechanisms.

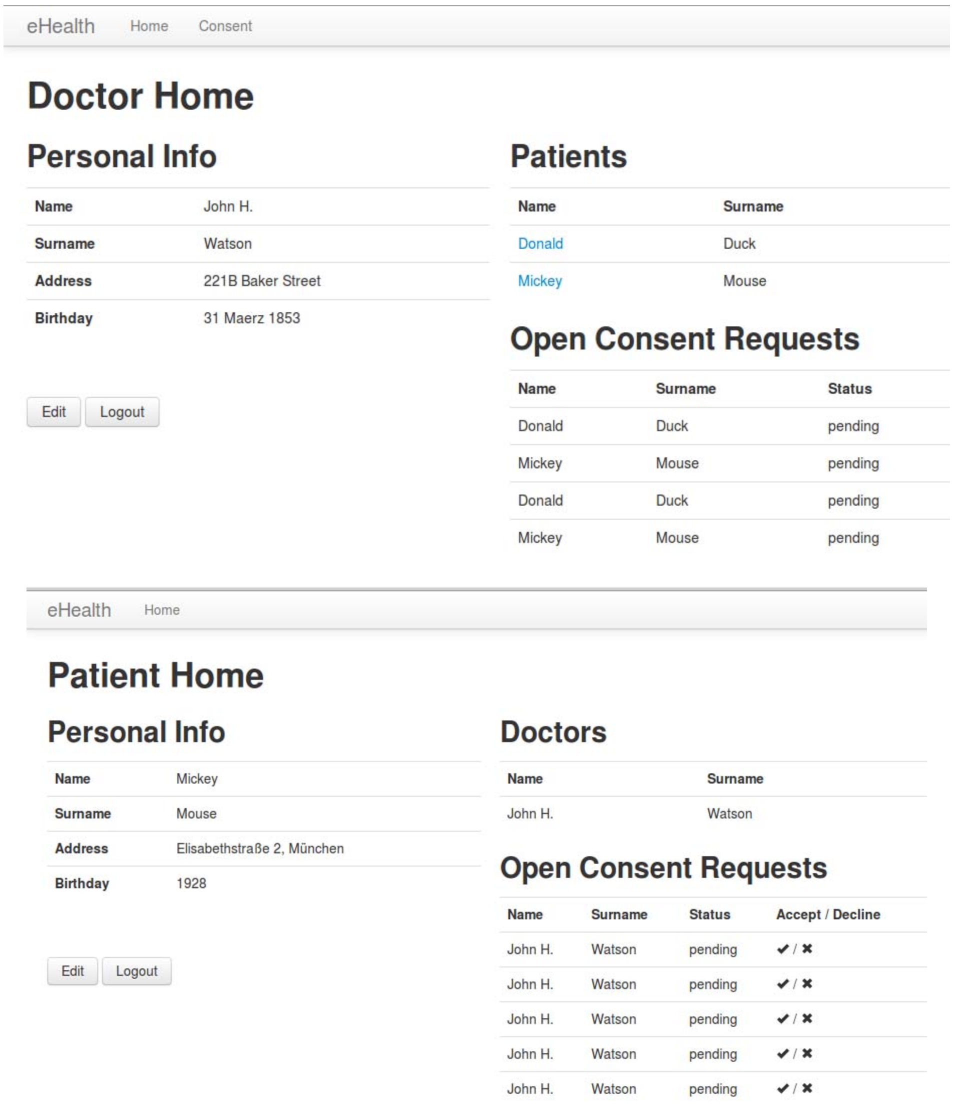
Through the GUI of the eHealth webpage, the following access control should be existing: If a doctor logs in the eHealth system, he can send patient consent request under the URL /eHealthSec/pages/consent. And to a given patient, he can just send one time request and then he should wait for the response, whether the patient accept the request or not. If the patient has accepted the request, he will be able to access his data. The doctor should not be able to send further consent requests in this case, since the permission has already been granted.

Using VERA with the model "Access Control" (see figure 56), a weakness by the access control in eHealth is found: One can always send consent request with an available "pid" to any patient with a doctor login, so that a patient can receive more than one consent request from a doctor. The attack process is illustrated in figure 57.


final
$foi == fol/$ print results
$foi < fol/$③
$\epsilon/$①
init
$\epsilon/$②
itFor
itSel
sodi == sodl / foi++
i==l / sodi++
$sodi < sodl/$ $selection = sod.items()[sodi]$ $key = selection[0]$ $operations = selection[1]$ $i = 0$
i<l; and IO[i] in operations/ fields2=Fields2; fields2.update(key:IO[i])])
recv
$\epsilon/$④
inject
itIO
$i < l$ and $IO[i]$ in operations/ $i+ = 1$
$\epsilon/$⑥
ϵ / input rcv
$\neg rcv \vee rcv['status'] \neq 200]$
tmp
0 <cl $\wedge Check\_info[c] \notin rcv['content']$
c==cl/
⑤:i++
check
chk2
rcv is not None $\wedge rcv[status] == 200]$ c=0;cl=len(Check_Info)

①	$input\ succInfo,\ failInfo$ $vHttp.snd(URL, cookie = Cookie, fields = Fields1, header = Header1, method = Method, fixrel = True)$
②	$input\ rcv$ $fo = vHttp.listForms(rcv["content"])$ $fol = len(fo)$ $foi = 0$
③	$sod = vHttp.listSelectionOptionsDict(fo[foi])$ $sodl = len(sod)$ $sodi = 0$
④	$vHttp.snd(fo[foi].actionorURL, cookie = Cookie, fields = fields2, method = fo[foi].methodor'GET', header = Header2)$
⑤	$succInfo.append(IO[i] +"\ is\ an\ available' + key + "\ and\ man\ can\ select\ it\ though\ changing\ request")$
⑥	$failInfo.append(IO[i] + "\ is\ not\ an\ available" + key")$

Figure 56: State machine model of access control flow



eHealth Home Consent

Doctor Home

Personal Info

Name	John H.
Surname	Watson
Address	221B Baker Street
Birthday	31 Maerz 1853

Edit Logout

Patients

Name	Surname
Donald	Duck
Mickey	Mouse

Open Consent Requests

Name	Surname	Status
Donald	Duck	pending
Mickey	Mouse	pending
Donald	Duck	pending
Mickey	Mouse	pending

eHealth Home

Patient Home

Personal Info

Name	Mickey
Surname	Mouse
Address	Elisabethstraße 2, München
Birthday	1928

Edit Logout

Doctors

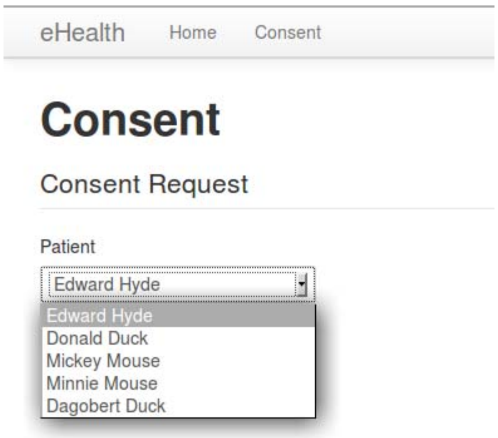
Name	Surname
John H.	Watson

Open Consent Requests

Name	Surname	Status	Accept / Decline
John H.	Watson	pending	✔ / ✖
John H.	Watson	pending	✔ / ✖
John H.	Watson	pending	✔ / ✖
John H.	Watson	pending	✔ / ✖
John H.	Watson	pending	✔ / ✖

Figure 57: Doctor home and patient home screens

Doctor John H. Watson can send now three possible patients consent request, because Duck Donald and Mouse Mickey are already his patients. Nevertheless, and attacker can still send a couple of consents request to both those patients who have already consented, filling the patient's inbox with spam.



eHealth Home Consent

Consent

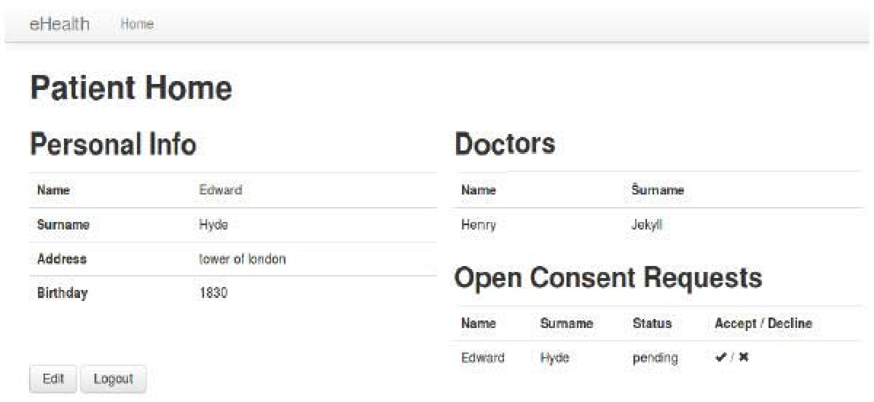
Consent Request

Patient

Edward Hyde

Edward Hyde
Donald Duck
Mickey Mouse
Minnie Mouse
Dagobert Duck

Figure 58: The consent page of patient Edward Hyde



Patient Home

Personal Info

Name	Edward
Surname	Hyde
Address	tower of london
Birthday	1830

Doctors

Name	Surname
Henry	Jekyll

Open Consent Requests

Name	Surname	Status	Accept / Decline
Edward	Hyde	pending	✓ / ✗

Figure 59: The home page of patient Edward Hyde with a consent request from himself

The VERA test results are summarized as follows:

```
models/access_control.xml test results 2013-06-07 17:08:30.912281

Config:
URL="http://localhost :8086/ eHealthSec/pages/consent"
Cookie="cookiename=cookievalue; mycookie=myvalue; JSESSIONID=3
ABA2F9837A7A6343AB40B1C2E831589"
Header1={}
Header2={'Content -Type ': 'application/x-www-form -urlencoded '}
Check_Info=["eHealth", "Home", "Consent"]


2 Successful test cases:
5 is an available pid and one can select by tampering the request
6 is an available pid and one can select by tampering the request

5 Failed test cases:
1 is not an available pid
3 is not an available pid
4 is not an available pid
9 is not an available pid
10 is not an available pid
```

Listing 21: Vera test results of eHealth

There is another access control flaw in the eHealth application. A patient should not be able to access the request consent page. But if a patient has login and then visits the ../eHealthSec/pages/consent, he can also send consent requests, as depicted in figure 58. He can for example send a consent request to himself, as depicted in figure 59.

9.4 Summary and conclusion

In the third project year of SPaCIoS, the various approaches and technologies from the project were further improved and - most importantly - integrated into a common SPaCIoS Tool environment. Using this tool environment, a number of testing exercises were executed on the suggested application scenarios. Most tools and technologies could be applied to more than one application scenario. In the following, the results from the individual tools and technologies are summarized.

VERA, low-level attacker models and testing. The VERA tool was successfully applied on eHealth. In general, positive experiences were made due to its ease of use and simplicity to setup the tool. However the results from its interactive application are prone to false positives that require manual checks afterwards.

Formalization of problem cases Several application scenarios were subjected to further formalization of selected problem cases. Updated ASLan++ models were obtained from eHealth. It proves that ASLan++ is an effective language to model and formulate security related behavior and properties.

Using the workflow of the SPaCIoS Tool it was possible to create a seamless methodology for model-based testing of security properties and vulnerability-driven testing from attacker models. The SPaCIoS deliverable D5.4 [i.8] will discuss further the performance of the different tools and technologies in more detail and compare it to existing work in security testing.

10 Document management system case study results

10.1 Case study characterization

The Infobase Document Repository (IDR) is a document management system that allows for the secure management and sharing of any documents or data files using only a web browser. It is provided by Siemens to offer a collaboration platform for joint projects involving external partners.

The repository mechanism supports web-based administration of text and binary files of any kind, e.g. text documents, spreadsheet tables, and even executables, in a hierarchical storage structure. The following characteristics can be noted:

1) Upload and download of entire directory trees as zip archives
2) Version management
3) File locking for team-oriented editing
4) Cut/copy/paste mechanisms for files and directory trees via a clipboard
5) Symbolic links
6) Fine-granular access control (for users, groups, and company), where to each object in the repository individual access rights can be allocated

For a thorough high-level overview of the Infobase case study, see SPaCIoS deliverable 5.1 [i.6], clause 7.1. The next clause describes the execution of a systematic security risk assessment of the document management application, which provides initial direction for the test and validation steps that follow. Afterwards, the specification-based testing approach is described that comprises the steps:

1) specification of security models in ASLan++ and model-checking of required security goals;
2) mutation of the correct model to generate abstract attack traces (AATs) using the SPaCiTE tool;
3) implementation of the AATs in a test automation environment and execution of the concrete tests. A second series of experiments follows the vulnerability-driven testing approach using the VERA tool. It performs tests based on a description of potential attacker behavior using the initial analysis results as a guide.

10.2 Security testing approaches

10.2.1 Security risk assessment of the Infobase application scenario

10.2.1.1 Background

Today's security testing is often not systematic not enough standardized. In particular, there are no clearly defined criteria for selecting relevant tests. Thus different analysts come to different results and sound quality assurance is hardly possible.

Literature suggests basing the choice and prioritization of tests on risk considerations but lacks a systematic approach for a traceable transition from abstract and business-oriented security risk assessment into the concrete and technical security testing world. In SPaCIoS deliverable 3.3 [i.5] it is recommended to bridge this gap in two steps:

1) The first one bridges between high-level and non-technical "business worst case scenarios" and less abstract "technical threat scenarios" using a technical description of the system and a systematic STRIDE-based elicitation approach.

2) The second is a rule-based step that maps technical thread scenarios to "test types", that is, to classes of tests that need to be adapted to the particular system under validation.

10.2.1.2 Scope and goal of the case study

In this clause, it is outlined how the proposed procedure was applied to the Infobase application which was introduced in the SPaCIoS deliverables 5.1 [i.6] and 5.2 [i.7]. The goal of this effort was to apply the methodology to a real world example and in this way, first and foremost, to collect practical experiences and lessons learned to improve practicability and thus acceptance for future real-world assessments. This being the main purpose, and to avoid getting lost in details, not all steps were presented in all detail and the (intermediate) results are often not complete in this exposition. A secondary goal of the effort is to find which parts of the methodology could be simplified in order to find the most important vulnerabilities but with a much less effort.

The rest of this clause is structured as follows: In the first part, a short general description of all steps of the method is given, and the results of their exemplary application on Infobase are presented. The second part contains the lessons learned and the suggestions for possible improvements.

10.2.1.3 Method walk-through

10.2.1.3.1 Describe general usage scenarios

Briefly describing the main usage scenarios helps to get a basic understanding of the SUT's purpose and its external actors. Both are prerequisites for all subsequent steps. The following scenarios were noted:

- The system allows to store, upload and download files (artefacts)
- The possibly sensitive artefacts can be shared with other, possibly external, users based on pre-defined access control properties

10.2.1.3.2 List assets

In this step, the system owner lists the non-technical assets that the SUT comprises, uses, and protects. In case of the document management application, these were, among other things:

- The sensitive repository content such as contract documents or price lists
- The correct functioning of the repository

10.2.1.3.3 Define security requirements

Security requirements consist of a tuple of a non technical asset and a security property. Considering the previous results, security requirements of the following type were derived:

- (sensitive artefact [class], confidentiality)
- (sensitive artefact [class], integrity)
- (repository, availability)

10.2.1.3.4 Identify relevant threats

After a discussion with the Infobase responsible and considering the usage scenarios, two main groups of possible attackers were identified:

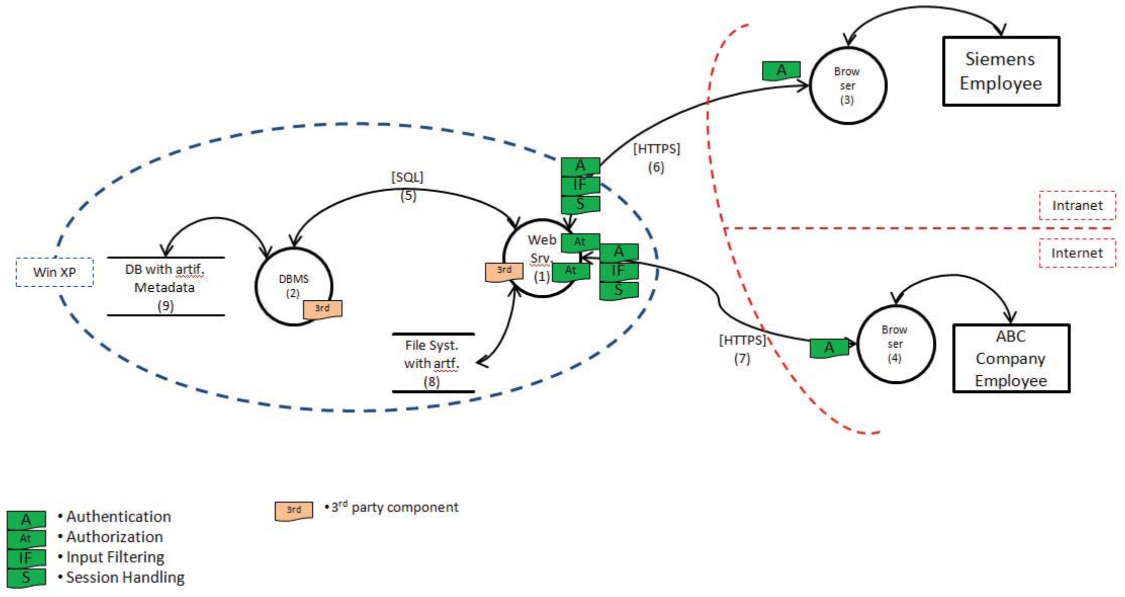


Figure 60: Infobase security overview with two different user roles

- Internal attackers, that is, legitimate users of the system such as employees or external partners with permission to use the system
- External attackers with no user accounts, in particular competing companies willing to perform industrial espionage

10.2.1.3.5 Define or derive a Business Worst Case Scenario (BWCS)

A BWCS is given by the non-technical description of a possible situation that might disrupt the achievement of objectives. The BWCSs should -to be useful for the purposes relate to the previously elaborated assets. Assuming a violation of each of the previously collected security requirements leads to a minimal set of relevant BWCSs. Given the above elaborated requirements, the following BWCSs were noted:

- Sensitive artefacts are disclosed to or modified by unauthorized internal or by external attackers (impact rating: high)
- The entire repository is made unavailable (impact rating: medium)

10.2.1.3.6 Generate Security Overview

The security overview is the result of a technical system description which captures and structures the security relevant technical aspects of the SUT. Besides providing a better technical understanding of the SUT, the security overview is crucial for the transition from security risk assessment results to security tests: It contains the data flow diagram elements that are part of the TTSs and provides the technical system information needed to identify and instantiate appropriate test types. Figure 60 shows the simplified and truncated Infobase Security Overview.

10.2.1.3.7 Map BWCS to Technical Threat Scenario (TTS)

There are two approaches to the mapping of BWCS to technical threat scenarios: (1) Top-down For every BWCS, examine which technical threats could lead to the BWCS. (2) Bottom-up For each DFD model element, brainstorm if any technical threat could pose a security problem which could lead to a, possibly not yet identified, BWCS. Practical experience suggests that analyzing each model element in a DFD is often time-consuming and leads to overlapping results for different elements. Alternatively, one can instead examine entire data flows from source to sink or system interactions with external actors:

1) Sensitive artefacts are disclosed to or modified by external attackers:

    - (P1, Escalation of Privilege)

   - (DF5(Web Srv. -> DBMS), Spoofing)
   - (DF6 (Employee -> Web Srv.), Spoofing)
   - etc.

2) The entire repository is made unavailable:
   - (P1, Denial of Service)
   - (DS8, Denial of Service)

10.2.1.3.8 Map TTSs to test types

The TTSs are still too abstract and need to be further concretized. For this purpose, the concept of test types was suggested. The rules that map TTSs to test types have the following structure:

1) A pattern in an annotated DFD. Besides a mandatory TTS which includes the security property violation, the pattern can include additional system elements and further annotations.
2) The level of sophistication for the security tests. It is determined by risk considerations such as the expected attacker and the desired assurance.
3) A reference to the suggested test type that fits to the above characteristics.

Given the intermediate results from the previous steps and applying the exemplary rules listed in the appendix yields the following test types:

10.2.1.4 Lessons learned

The limited time frame of real world security assessments is the most significant obstacle for the industrial application of the proposed full risk-based test selection procedure. (This is also true for any other analysis method that requires additional time). Indeed, security risk assessment and system pre-analysis do take time and care should be taken that they do not consume too much of the available time budget planned for practical testing which – at the end of the day – yields the actual "tangible" results: exploitable vulnerabilities that the system owner should fix.

Therefore the analysis method should be as light-weight as possible. Once the analysts have understood the framework and the dependencies of the steps, may be advised not to apply all proposed steps in full detail. This will help to get "the biggest security bang for the buck". Many security practitioners and paying customers do not want to spend much time analyzing the security architecture of the system. This activity is perceived by them as a less exciting "overhead", which goal is planning and prioritizing which test to perform. They would rather start with practical security tests as soon as possible, especially if the effort and time dedicated to the total activity is rather restricted.

The Infobase case study indicated that increased traceability for security test selection is appreciated but may not be sufficient for a sustainable industry acceptance. The latter would be easier to achieve if the security risk assessment method provides the following additional benefits:

1) Improved test coverage by suggesting a large number of adequate test types, especially less common ones that may not be thought of by the ordinary tester.
2) More concrete indications for test selection that goes beyond the test type. For example, instead of suggesting Intermediate Security Attestation Level fuzzing test for interface registry, one could add the exact location of the value, its type, and the functions the fuzzing data passes.

The first benefit can be achieved with a well-stocked test library; the second requires more technical effort but could leverage the information extracted for a more technical security overview.

10.2.2 Improvements of the security model – detecting Cross-Site Request Forgery at ASLan++ level

10.2.2.1 Description of CSRF in Infobase

In this clause the aim is to validate the Infobase specification with respect to Cross-Site Request Forgery (CSRF). Considering that the objective is to search for CSRF at ASLan/ASLan++ level, it is first defined how to model a web application scenario for this purpose.

In order to exploit a CSRF vulnerability, and attack a web application (in this context, with "attacking a web application" it is meant that an intruder can perform requests to the web application that it should not be allowed to do), mainly three parties have to get involved: an intruder, a user and a server. The intruder is the entity that wants to exploit the vulnerability and attack the web application hosted on the server. The server is then the entity that represents the web application host and, finally, the user entity is the honest agent who interacts with the web application (i.e. with the server).

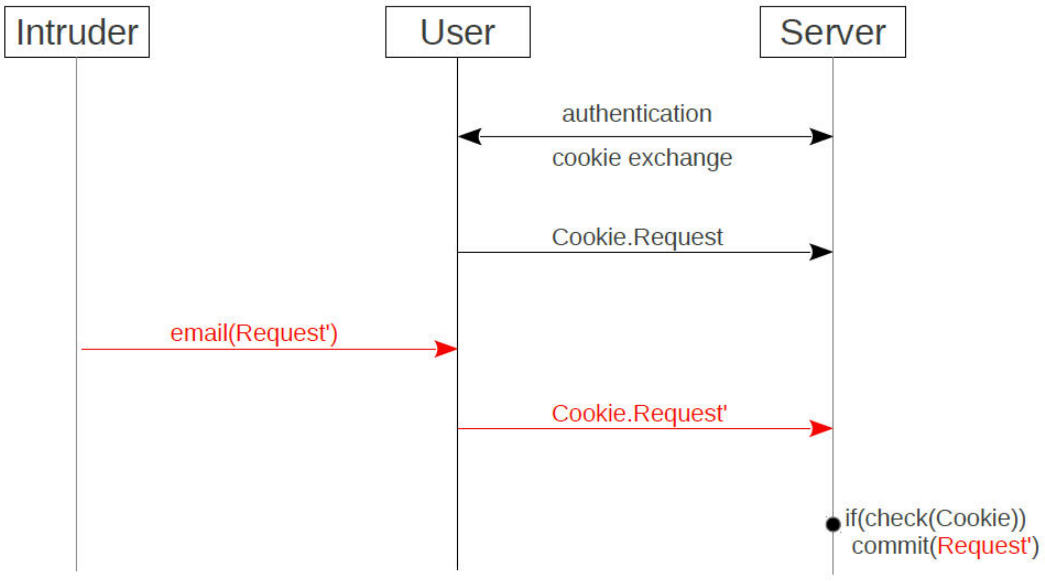


Figure 61: CSRF MSC

If the web application is vulnerable to CSRF, the intruder can trick the user to perform requests to the server in behalf of him (figure 61). This attack scenario can be summarized by the following five steps:

1) The user logs in to the web application (authentication phase).

2) The server sends a cookie to the user who will store it (in the web browser). From this point on, the cookie will be automatically attached by the web browser to every request sent by the user to the server.

3) The intruder sends to the user a malicious link containing a request for the web application on the server.

4) If the user follows the link, the web browser will automatically attach the cookie and will send the malicious request to the server.

5) The web application cannot distinguish a request made by the intruder and forwarded by the user from one made and sent by an honest agent, and then it accept the request.

The state-of-the-art protections against this vulnerability are mainly two and can be used together:

1) The server asks the user for a confirmation at every request the user sends to the server.

2) A secret (e.g. a pseudo-random token) shared between the user and the server has to be attached at every request.

In figure 62 the model of a web application that uses both CSRF protection mechanisms is reported. In this way, the intruder cannot simply send a request to the user and wait for its execution. In fact, the user will not confirm the request and the browser will not automatically add the secret to the request.

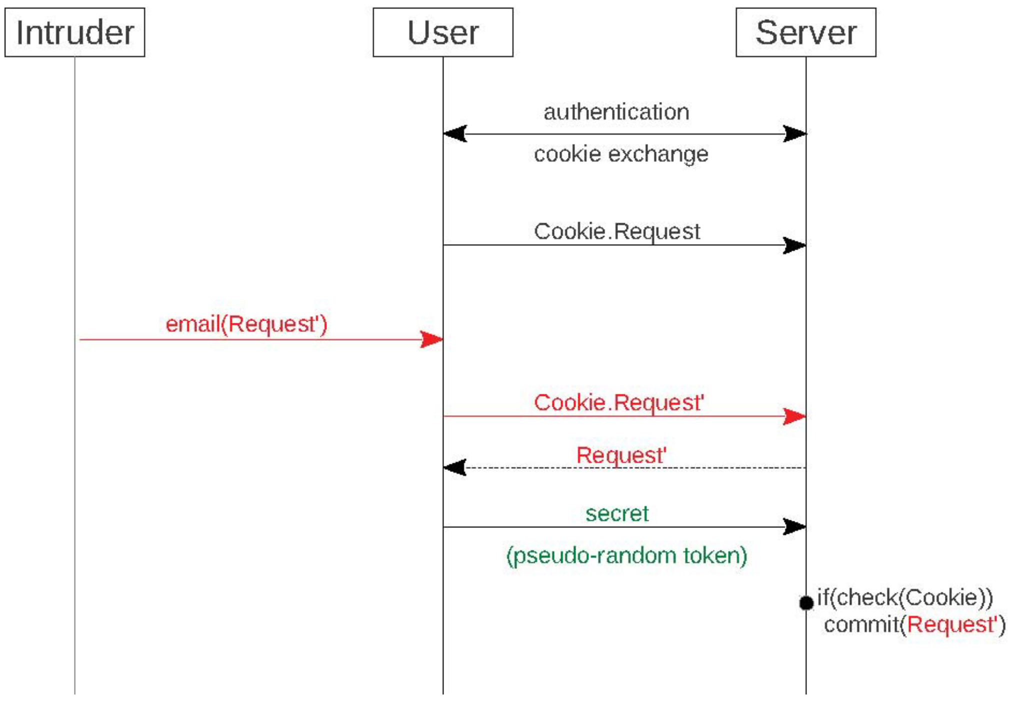


Figure 62: CSRF protection MSC

The objective is to check if the Infobase protections against CSRF are strong enough; that is, to check if there is a way for the intruder to bypass protections and commit a request that it is not allowed to do.

It is important to observe that, from the given description of CSRF an intruder uses the user as an oracle. The intruder does not see the communication between the user and the server but it will send a request to the user and wait for it to be executed (figure 63). where it is shown the scenario from the intruder point of view that cannot see the communication between the other two entities.

10.2.2.2 Modeling CSRF in ASLan++

In this clause it is described how the Infobase ASLan++ specifications, to check for CSRF, are defined. In order to check for CSRF in the ASLan++ specifications two entities are considered: Client/Oracle entity and Server entity.

10.2.2.2.1 Client

In the Client entity there is a first authentication phase to obtain the cookie and logging in to the web application.

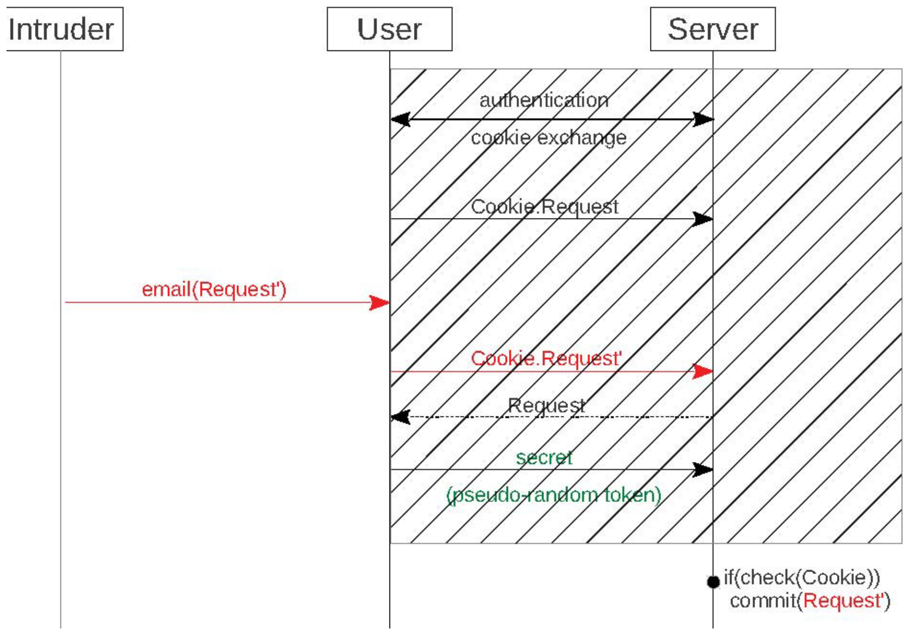


Figure 63: CSRF Oracle MSC-the image is from the intruder point of view and the grey part is not visible to the intruder

```
% sends his/her name and password to the server’s login service
Actor ->* Server: Actor.UserName.Password;
% the server 's login service responds to the login request with a cookie
select { on (Server *->* Actor: ?Cookie &
?Cookie=cookie(UserName ,?,?)): {} }
```

After this phase, the Client can perform requests to the Server asking for services. When a user wants to send a request to the Infobase system, it first load the web page using a web browser. The Server produces the web page and sends it together with a CSRF token (i.e. a fresh pseudo-random token linked to the session ID of the Client). At specification level it is possible to model this mechanism by creating a variable Request that the Client want to submit. When the Client sends this Request to the Server, the latter will generate and send the token back to the Client. Now the Client sends the Request together with the cookie and the CSRF token as follows.

```
% load request page with the csrf token
% user asks for a web page
% and the server sends it to him/her together with a csrf token

Actor *->* Server: Cookie.Request;
Server *->* Actor: ?CSRFToken;
Actor *->* Server: Cookie.Request.CSRFToken;
```

Between the authentication and the request submission part, a reception of a message is added. This message contains a variable Request and it is sent from an unknown entity: ?-> Actor: ?Request;. In this way, the scenario in which the Client receives a malicious email from a third party is modelled; the email contains a link to submit a request to the web application.

Finally, the Client will receive from the Server the confirmation that the request has been executed by the web application.

```
% the Server's frontend sends back to the user the answer
% of the repository
% To avoid replay of an answer that does not fit to the current
% request, "Request" is added:
%
Server *->* Actor: Request.?Answer;
UserName->got(Answer);
```

10.2.2.2.2 Server

The server entity accepts three different kinds of requests: authentication, request for a web page and request that it has to commit to the web application.

With authentication request a Client (not already authenticated) sends to the Server its username and password asking to log in. The Server will check the received credentials and, if they are correct, it will generate a Cookie that will be sent back to the Client.

```
% Case 1: login service receives the user request
 %% and generation of a new cookie for the session
%

on((? ->* Actor: ?UserIP.?UserName.?Password
  & !dishonest_usr(?UserName) |
% In case the user is dishonest, the UserIP may be forged,
% and therefore it is not required auth_Login:(?UserName)
% nor secret_Password:(?Password)
% as these implicitly rely on UserIP.
% In this model, it is sufficient to state
% secret_Password:(Password) at the Client.

?->* Actor: ?UserIP.?UserName.?Password & dishonest_usr(?UserName)) &
% checks if the data are available in the database
%% "select..on" is more efficient than "if"

loginDB->contains((?UserName,?Password,?Role))): {

% At this point, it was checked, using the password,
% that the user is legitimate.
% With the query, the role of the legitimate user is extracted.
% It creates the cookie and sends it back to the user
Nonce := fresh(); Cookie := cookie(UserName,Role,Nonce);
% adds the Cookie into the DB associated
% with the name of the user
cookiesDB->add(Cookie);

% uses the IP address sent by the client
% to communicate the cookie to the correct user

Actor *->* UserIP: Cookie; }
```

The second type of request is a web page request. The Client asks for a web page before sending a request to the web application. Here the Client is already logged in and then it sends the request together with the Cookie. The Server will check the Cookie and generate a fresh token that will send back to the Client.

```
% Case 2:having a cookie, a user makes a request to the frontend
%% without the CSRF token
%% and receives the respective token from the repository

on(?UserIP *->* Actor: cookie(?UserName,?Role,?Nonce).?Request & cookiesDB-
>contains(cookie(?UserName,?Role,?Nonce))): {
CSRFToken:=fresh(); csrfTokenDB->add((UserIP,Request,CSRFToken)); Actor *->* UserIP: CSRFToken;
}
```

The third case is when a Client sends a request to the Server. The Server checks the cookie and the token and commit the request.

```
%% Case 3: having a cookie, a user makes a request to the frontend
%% and receives the respective answer from the repository
on(?UserIP *->* Actor: cookie(?UserName,?Role,?Nonce).?Request.? CSRFToken &

% checks if the token is the right one
csrfTokenDB->contains((?UserIP,?Request,?CSRFToken)) &

% checks if the user is allowed to do this request
% and if the user is linked to the cookie

checkPermissions(?UserName,?Request) & cookiesDB->contains(cookie(?UserName,?Role,?Nonce))): {

% if the user has the right credential, then the frontend
% sends the request to the repository which will return the
% answer

Answer := answerOn(Request);

%% shortcut for simplicity: no extra Repository

commit(Request); Actor *->* UserIP: Request.Answer; }

% Case 3: otherwise the user is either a cheater
% who has not achieved
% his goal or a user that has an invalid cookie to issue
% the request
}
```

10.2.2.2.3 Goal

The goal is to check if there is a way for the intruder to commit a request to the web application.

```
csrf_goal: [](!commit(intruderRequest));
```

From the specification, the only way that the intruder has to commit a request is to bypass the CSRF protection (i.e. CSRF Token).

```
1 <acflaw> <authz>checkPermissions |contains</authz> </acflaw>
```

Listing 22: Configuration for the ACFlaw operator used for the Infobase's model

To model that the intruder wants to submit a request that an honest agent does not, a particular request (intruderRequest) in the Session entity is introduced as follows:

```
body { %% of the Environment entity
role1->can_exec(request1);
role1->can_exec(intruderRequest);
role2->can_exec(request2);
any UserIP. Session(UserIP, usr1, role1, request1)
where !dishonest(UserIP);
new Session(i , usr2, role2, request1);
}
```

10.2.2.3 Result of the analysis of the Infobase model

Both SATMC and CL-Atse concludes that the specification is safe with respect to the CSRF goal. This means that the CSRF protection (i.e. CSRF token) cannot be bypassed, in the modeled scenario, from the DY intruder.

10.2.3 Mutation-based test generation

From the three semantic mutation operators presented in [i.10], one is applicable to the Infobase model, namely the Access Control Flaw (ACFlaw) operator.

The purpose of the ACFlaw operator is to inject into the original model a "Missing Authorization" vulnerability. This task is carried out by removing a symbolic function that models an authorization check from the set of facts that has to hold in order to trigger a transition step. Although the applied modification is of syntactical nature, i.e. removing a fact from the LHS of a step, the ACFlaw operator is a semantic operator because its application has to be narrowed to a specific set of facts built with the symbolic functions modeling an authorization check.

The mutant operator cannot identify which symbolic functions model authorization checks, therefore the modeler has to provide a configuration file containing a regular expression that the ACFlaw operator uses to identify only the facts build using those specific functions.

Listing 22 shows the configuration file used to specify that checkPermissions and contains are the symbolic functions that model an authorization check throughout the specification.

While checkPermission has been defined by us in the model, "`contains`" is a symbol defined in the ASLan Prelude File [i.1] which states what are the elements present in a set (e.g. contains(Set,E) means that E is in the set E). The function symbol contains has been included in the configuration file because in the Infobase model there are checks on the presence of credentials into a database (modeled using the sets loginDB and cookiesDB) on the server side. Therefore, removing these checks from the LHS of the steps in the original model, corresponds to removing authorizations checks. In fact, by removing those facts, the intruder can perform actions he is not allowed to execute.

Applying the ACFlaw operator, 4 mutants are obtained, out of which 3 led to an AAT that are described and taken into account for the concretization phase.

10.2.4 Test automation

Test automation in this context is concerned with deriving executable tests from the abstract attack traces generated using the mutation-based test generation technique introduced in clause 8.5. In the following clauses the test automation process and the test tool ScenTest which is used to generate executable test code are described.

10.2.4.1 The ScenTest tool for scenario-based testing

ScenTest is a tool that enables the description of test scenarios, i.e. test cases, as UML sequence diagrams and the generation of executable JUnit tests. It supports the black-box test of concurrent and distributed systems based on message-based communication. The tool builds on a software modeling tool for modeling the test scenarios and the Eclipse framework as the IDE. A test scenario that is expressed in terms of UML sequence diagrams consists of a single System Under Test (SUT) lifeline and one or more lifelines that describe the different interfaces and interaction points of the SUT with its environment, which is replaced by the tester during test execution. The interaction flow of messages can be strictly sequential, concurrent, or alternative. An entire test suite can be described as a set of test scenarios. In addition, various test scenario fragments (scenario building blocks) can be combined into a single test scenario graph, from which new test scenarios for test execution can be generated according to structural coverage criteria such as node or edge coverage. A comprehensive description of ScenTest is provided in [i.9].

10.2.4.2 General approach to test automation of AATs

Figure 64 depicts the complete test automation process followed for deriving executable tests from the abstract attack traces generated using the mutation-based test generation technique. The process can be summarized as follows:

Manually concretize the attack traces by mapping them to a SUT specific test scenario.

Represent the test scenario as message-exchanges between the involved parties (the testers) and the SUT in terms of a UML sequence diagram.

Develop a Test Adaptor that maps the test logic to the SUT logic and handles connection and communication with the SUT. The test adaptor also the place where the test verdict is declared. The test verdict is the condition to be satisfied to consider a test as "Passed".

The generated test cases are represented as executable JUnit (http://www.junit.org) tests. The tool HtmlUnit (http://htmlunit.sourceforge.net) is used within the Test Adaptor to emulate the system ï£¡s client, which is a web browser in the case of Infobase system. HtmlUnit can emulate different browsers and different versions of a specific browser. Therefore it allows for full control over the emulated browser which is of great importance when dealing with complex test scenarios. Additionally, the emulated browsers provided by HtmlUnit
GUI-less. Avoiding testing over browser GUI makes testing more independent of operating system features and browser-specific implementations. Other features like the full support of JavaScripts and reliance on an HTML object model allow to validate web pages to the finest level of detail.

10.2.4.3 Derived test case, test execution and test results

In this clause a listing of attack traces and the test case derived from them are provided as well as the test execution results.

10.2.4.3.1 Test scenario 1:

`Infobase_Scene1_contains_step008`
Generated abstract attack trace. Follows a listing of the exchanged messages according the generated attack traces used to derive this test case.

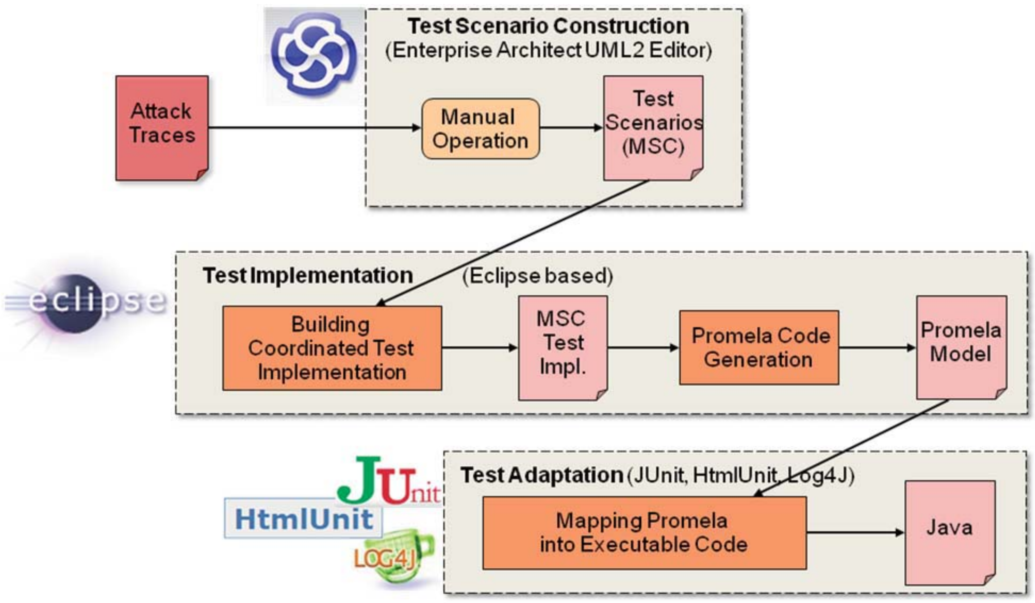

Attack Traces
Test Scenario Construction
(Enterprise Architect UML2 Editor)
Manual Operation
Test Scenarios (MSC)
eclipse
Test Implementation
(Eclipse based)
Building Coordinated Test Implementation
MSC Test Impl.
Promela Code Generation
Promela Model
JUnit
HtmlUnit
LOG4J
Test Adaptation (JUnit, HtmlUnit, Log4J)
Mapping Promela into Executable Code
Java

Figure 64: Test automation approach implemented in ScenTest

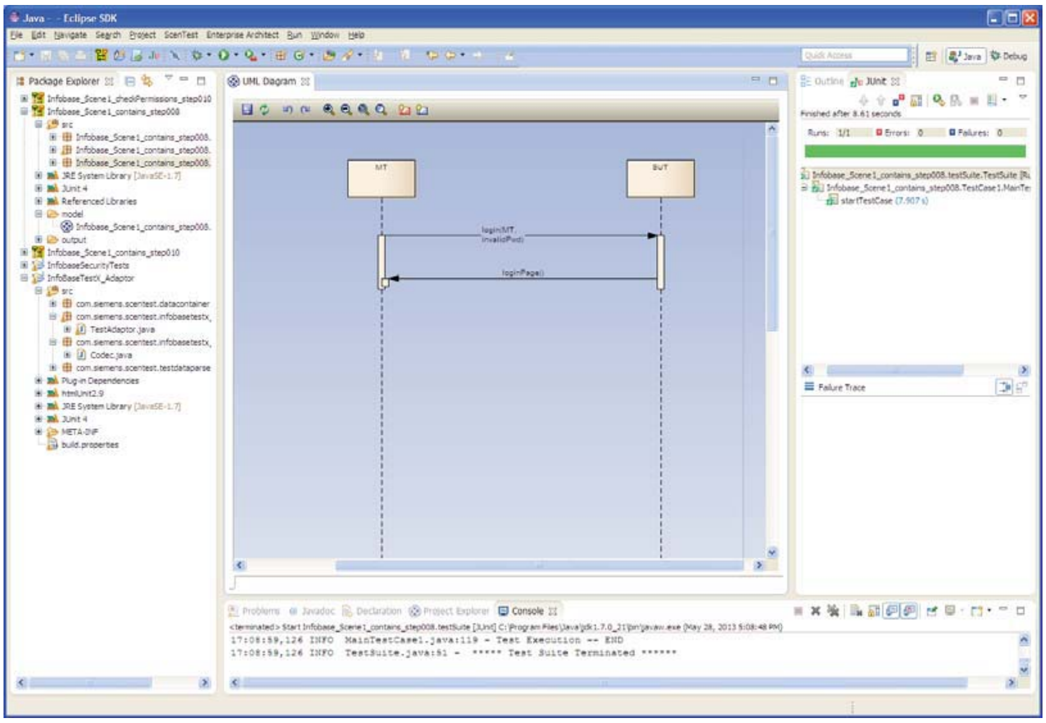
```
1 <?> ->* server : UserIP(123).UserName(123).Password (123)
2 server *->* <UserIP(123)> :
3 cookie(UserName(123),Role(123),n123(Nonce))
```

Listing 23: AAT Infobase_Scene1_contains_step008

Concretized abstract test case (figure 65)

- Pre-condition:
  - Martin Tester (Quality Ltd.) is an Infobase registered user.
- Test sequence:
  - Martin Tester attempts to log in with a valid user name and an invalid password.
- Expected result: The user should not be allowed to log in and receive a valid cookie from the Infobase system.

Result from test run (figure 65): The user was not allowed to log in and did not receive a valid cookie from the Infobase system. Test PASSED.



Finished after 8.61 seconds

Runs: 1/1 Errors: 0 Failures: 0

Infobase_Scene1_contains_step008.testSuite.TestSuite [R
Infobase_Scene1_contains_step008.TestCase1.MainTe
startTestCase (7.907 s)

```
<terminated> Start Infobase_Scene1_contains_step008.testSuite [JUnit] C:\Program Files\Java\jdk1.7.0_21\bin\javaw.exe (May 28, 2013 5:08:48 PM)
17:08:59,126 INFO  MainTestCase1.java:119 - Test Execution -- END
17:08:59,126 INFO  TestSuite.java:51 -  ***** Test Suite Terminated ******
```

Figure 65: Test automation for Infobase_Scene1_contains_step008

10.2.4.3.2 Test scenario 2:

`Infobase_Scene1_contains_step010`
Generated abstract attack trace. Follows a listing of the exchanged messages according the generated attack traces used to derive this test case.

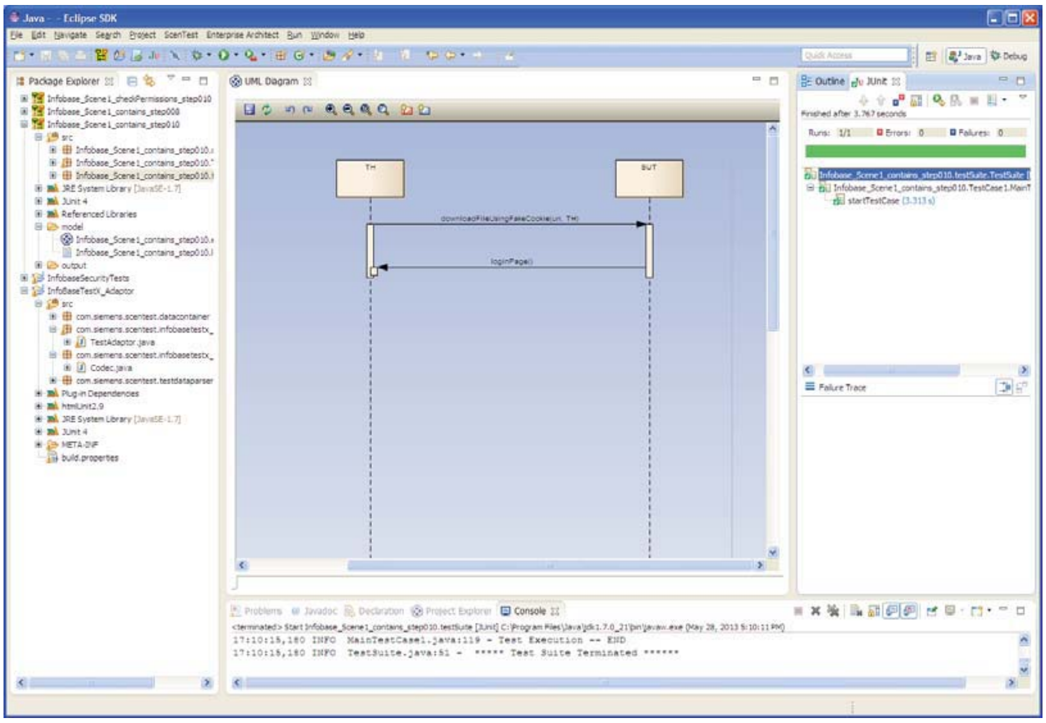
```
4 <i> *->* server :
5 cookie(usr1,Role(127),Nonce(127)).request1 server *->* <i>
6 : request1.answerOn(request1)
```

Listing 24: AAT Infobase_Scene1_contains_step010

Concretized abstract test case (figure 66)

- Pre-condition:
  - Thomas Hacker has no permission to access Infobase.
- Test sequence:
  - Thomas Hacker (an intruder) constructs a fake cookie to be used for Infobase requests.
  - Thomas Hacker attempts to send a request to Infobase using his fake cookie.
- Expected result: The intruder Thomas Hacker should not be allowed to execute his request.

Result from test run (figure 66): The intruder was not allowed to execute his request. Test PASSED.



```
<terminated> Start Infobase_Scene1_contains_step010.testSuite [JUnit] C:\Program Files\Java\jdk1.7.0_21\bin\javaw.exe (May 28, 2013 5:10:11 PM)
17:10:15,180 INFO  MainTestCase1.java:119 - Test Execution -- END
17:10:15,180 INFO  TestSuite.java:51 -  ***** Test Suite Terminated ******
```

Figure 66: Test automation for Infobase_Scene1_contains_step010

10.2.4.3.3 Test Scenario 3:

`Infobase_Scene1_checkPermissions_step010`
Generated abstract attack trace. Follows a listing of the exchanged messages according the generated attack traces used to derive this test case.

```
7 UserIP(125) ->* <server > : UserIP(125).usr1.n113(Password)
8 <?> ->* server : i.usr2.n111(Password) server *->* <i>:
9 cookie(usr2 ,role2 ,n125(Nonce)) <i> *->* server :
10 cookie(usr2 ,role2 ,n125(Nonce)).request1 server *->* <i> :
11 request1.answerOn(request1)
```

Listing 25: AAT Infobase_Scene1_checkPermissions_step010

Concretized abstract test case (see figure 67)

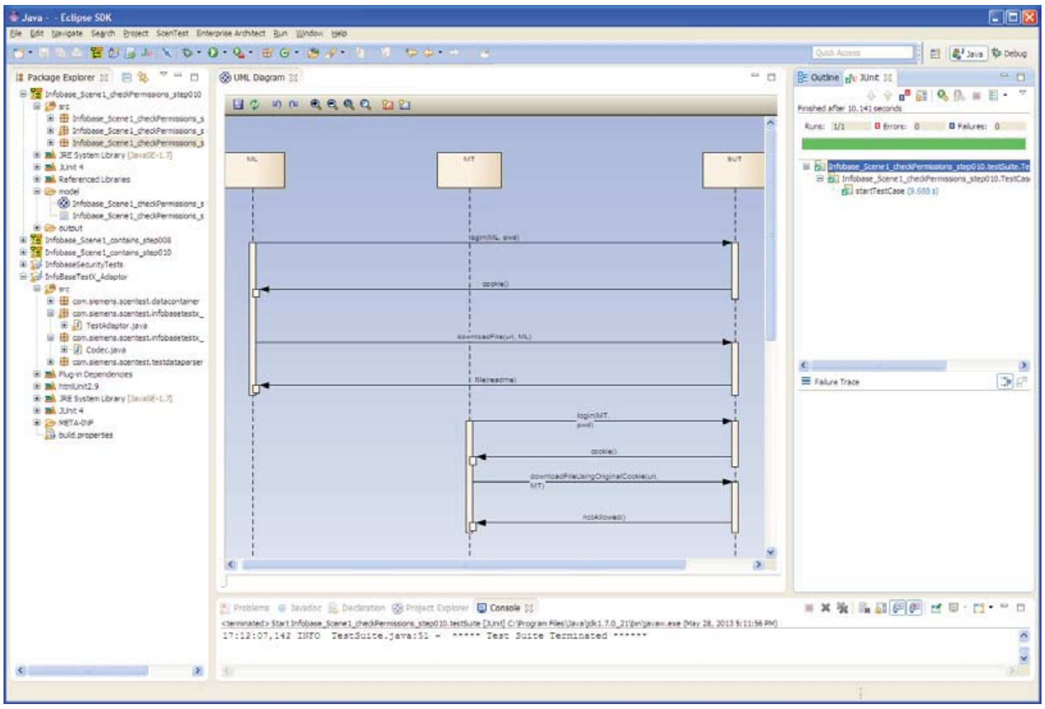
- Pre-condition
- Maggie Lee and Martin Tester are registered users
- Maggie Lee has the required privileges to access the repository "Spacios"
- Martin Tester does not have the requires privileges to access the repository "Spacios"
- Test sequence
- Maggie Lee retrieves (downloads) the document README.txt from the repository "Spacios"
- Martin Tester attempts to retrieve the document README.txt from the repository "Spacios"
- Expected result: It has not to be possible for Martin Tester to retrieve the file README.txt from the repository "Spacios"

Result from test run (figure 67): It was not possible for Martin Tester to retrieve the file. Test PASSED.

10.3 Results by applying the VERA Tool

10.3.1 Considered vulnerabilities

In the following it is summarized the low-level vulnerabilities considered for this problem case. The considered vulnerabilities are a representative set of:



```
<terminated> Start Infobase_Scene1_checkPermissions_step010.testSuite [JUnit] C:\Program Files\Java\jdk1.7.0_21\bin\javaw.exe (May 28, 2013 5:11:56 PM)
17:12:07,142 INFO  TestSuite.java:51 -  ***** Test Suite Terminated ******
```

Figure 67: Test automation for Infobase_Scene1_checkPermissions_step010

The most common low-level security vulnerabilities in web-applications and correspond to a refinement of the analysis performed in [i.6]. For the most common level security vulnerabilities in web-applications, see for instance OWASP-Top-10 https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project.

Cross-Site Scripting (XSS)

The presence of this vulnerability allows attackers to execute arbitrary JavaScript code on the client side, if the user is misled into visiting maliciously prepared links. As a consequence, an attacker can potentially get hold of session IDs (depending on the cookie configuration for the site), steal confidential information stored in the site and manipulate user requests among others, thus posing a threat to confidentiality and integrity goals.

SQL Injection

Allows attackers to execute SQL statements to the database. Depending on the privileges of the application, this can have several damaging effects, ranging from authentication by-pass (by adding trivial conditions to password checkers for example, the infamous "OR 1=1" injection) and reading/writing data of the application database, to even execution of OS commands and thus complete takeover of the server hosting the application.

Weak passwords

If the application does not enforce the use of strong passwords attackers can try to guess credentials by brute-force. This is usually done by trying lists of commonly used passwords (typically in the order of magnitude of the few thousands). The probability of success of attackers is increased if the application does not have a lock-out mechanism for repeated unsuccessful login attempts for a given account.

CSRF vulnerabilities

Allow attackers to trick users into issue requests to the server with potentially malicious side-effects (such as change permissions, modify sensitive data), by exploiting the fact that the browser will always send cookies to the victim domain even if the requests are issued by a malicious web-site. To prevent this issue, critical requests should contain an extra random value associated to the user session that is not contained in the cookies. The server can thus check for the validity of this extra value (called CSRF token). Attackers would have to guess for this value to craft valid malicious requests.

File Enumeration

Hidden administrative interfaces, old backup copies of source code and vulnerable scripts that are not referenced by the main application URLs can be automatically found by attackers searching for this hidden files/folders based on commonly used file-names and extensions. As a consequence, attackers can expand their attack surface on their application (by for instance gaining unrestricted access to the data-base in case of a miss-configured database administration interface).

Path traversal

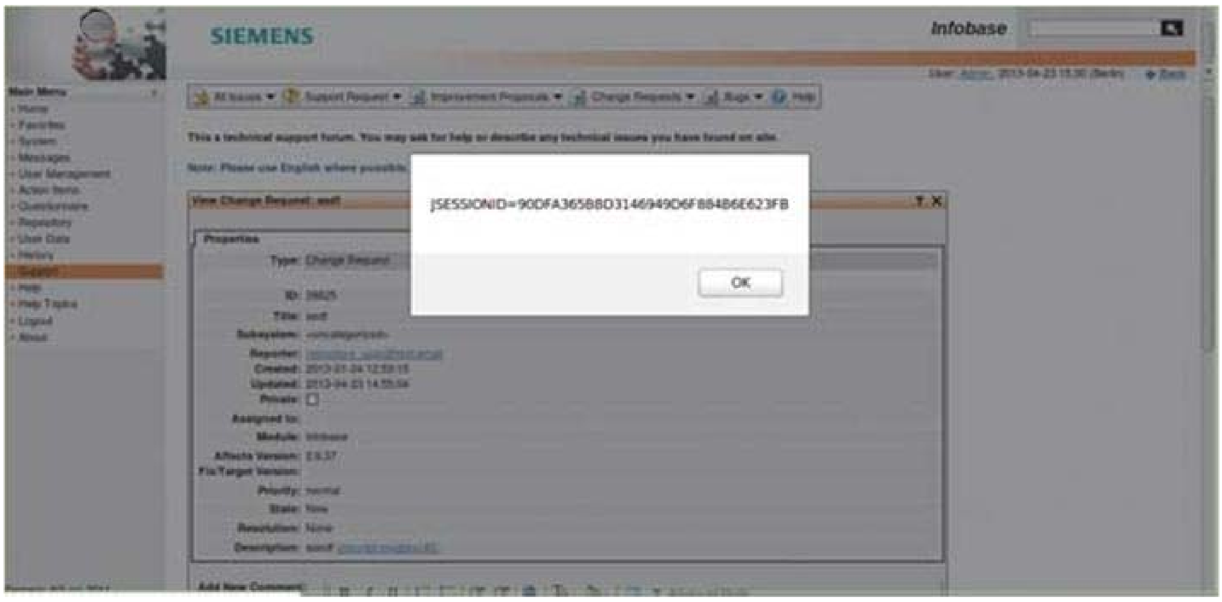
Web-applications may refer to system resources directly by providing a path (for instance to present log files, images, etc.). If this path is partially constructed with user input, attackers can potentially manipulate it to access arbitrary systems resources by traversing the directory structure of the server.

The document management application implementation was tested against attacker models for a number of vulnerabilities using the VERA approach. A number of security issues present in the current version of Infobase were found: one instance was found of persistent cross-site scripting, a hidden administration interface and missing protection against brute force attacks. The results and lessons learned are detailed in the following clauses.

10.3.2 Cross-Site Scripting (XSS)

It was applied the general injection model described in [i.5] with JavaScript payloads to InfoBase, but obtained no interesting results. However, during these tests it was noticed that some interesting tests cases were being neglected because the initial model did not take CSRF tokens present in InfoBase into account. This triggered the development of a generalized injection model.

After applying this model to InfoBase, a stored XSS injection vulnerability was found in the issue reporting site, as depicted in figure 68.



JSESSIONID=90DFA365B8D3146949D6F884B6E623FB

Figure 68: Successful XSS injection by adding a new comment on an issue using `<script>alert(document.cookie);</script>`

Since the InfoBase cookies are not protected with the HTTP_only flag, it was possible to directly read the session ID, which indicates that an adversary might steal this sensitive information if he manage to trick a logged user into reading a malicious comment that exploits this vulnerability. Here is reported an excerpt of the results as reported by VERA:

Stored XSS Vera test results on InfoBase

```
models/token_inj.xml test results 2013-05-17  17:34:31.853168

Config:
URL="http://localhost :8086/"
Cookie="JSESSIONID=F66D0F63033496C466213E73A52D3C98"
Method="GET"
Header=""
Header2={'content -type ': 'application/x-www-form -urlencoded'}
Path="support/Issue/view.do?reqCode=view&type=1&id=28203"
Action="support/Issue/IssueComment/create.do"
Ignore_Form_Fields=[['reqCode ', '_history_id_ ']]
Test_Input="comment"
Test_Content="TokenXSSTest"
Correct_Fields={'reqCode':'saveNew'}

1 Successful test cases:
Injection "<script>alert(document.cookie);</script> to form 1" is successful

1 Failed test cases:
Injection "<IMGSRC%3d"javascript:alert('XSS');">toform1 " failed
```

10.3.3 SQL injection

To be able to test for SQL injection, the modified injection model was necessary as discussed in the previous clause. However, no SQL injection vulnerability was found in InfoBase.

10.3.4 Password brute-forcing

The goal of this experiment was to test the password brute force model as introduced in [i.5] to the InfoBase implementation, in particular to the log-in main interface (figure 70). For readability, this model is recalled in figure 69.

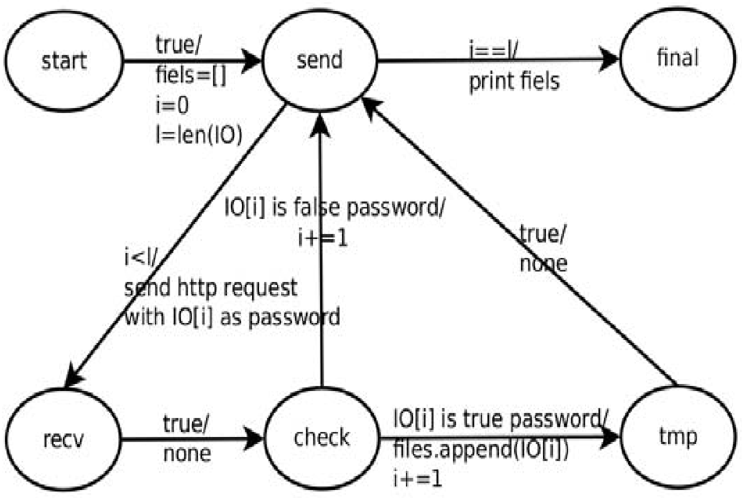


Figure 69: Model "password_brute_force"

For this purpose, the following known user accounts were fed to the instantiation library: Administrator and RepositoryAdmin. For the password payload a library was used containing commonly used passwords, successfully finding the password of the Administrator user (admin).

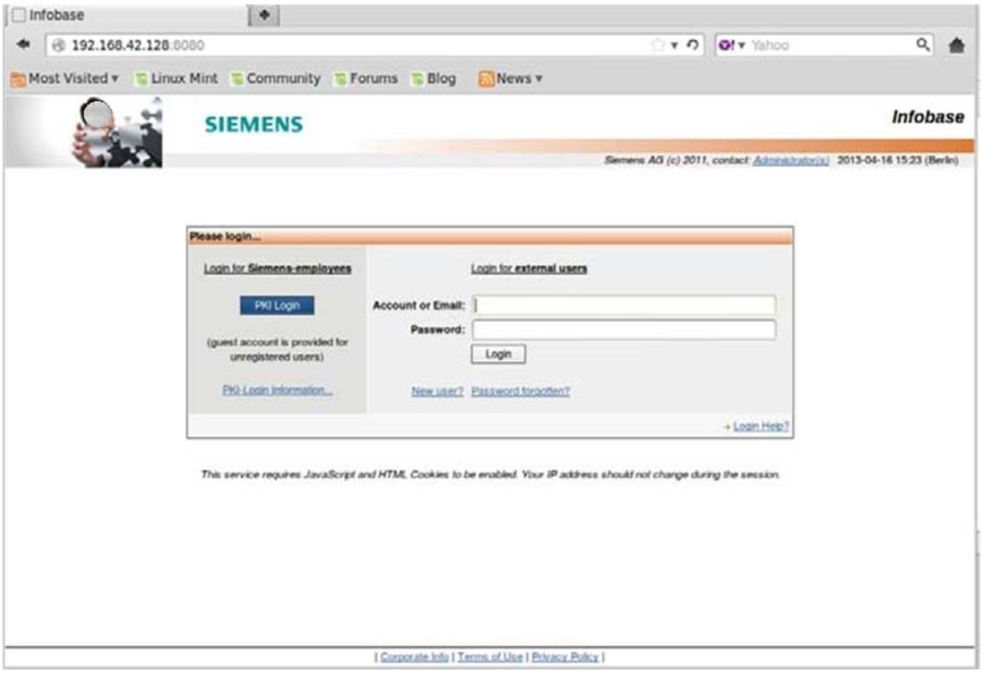

Infobase
192.168.42.128:8080
Yahoo
Most Visited
Linux Mint
Community
Forums
Blog
News
SIEMENS
Infobase
Siemens AG (c) 2011, contact: Administrator(s) 2013-04-16 15:23 (Berlin)
Please login...
Login for Siemens-employees
PKI Login
(guest account is provided for unregistered users)
PKI-Login information...
Login for external users
Account or Email:
Password:
Login
New user?
Password forgotten?
Login Help?
This service requires JavaScript and HTML Cookies to be enabled. Your IP address should not change during the session.
| Corporate Info | Terms of Use | Privacy Policy |

Figure 70: Infobase user login interface

In the following it is reported the used configuration and a partial result of the experiments for these account.

Test result summary with account "Administrator"

```
models/password_brute_force.xml test results 2013-05-17 17:33:39.057231

Config:
URL="http://localhost:8086/"
Cookie=""
Header={'content-type': 'application/x-www-form-urlencoded '}
Username_Field_Name="_requested_url_=%2F&_modname_=infobase
&_login_param_=true&_nopki_=true&login_"
Username_Field_Value="Administrator"
Password_Field_Name="password_"

1 Successful test cases:
admin
```

10.3.5 Cross-Site Request Forgery (CSRF)

Testing for Cross-site Request Forgery in an automated way is a challenging task, because the side-effects of a vulnerable action may vary widely from application to application. Therefore the focus was on a task that is more amenable to automatic testing: validating the strength of CSRF tokens. Infobase has CSRF tokens for most POST and GET request, making it a difficult target for these kind of attacks.

A model reported was developed, that automatically assesses whether the CSRF tokens of an application are regenerated within a session. In general, the longer the validity of a CSRF token is, the weaker guarantees it provides. It was found that most tokens within Infobase are regenerated after every visit of their containing website, whereas some of them are valid for the whole session, as summarized in the following result excerpt:

Excerpt of token strength test for Infobase

```
models/token.xml test results 2013-05-17 18:31:09.363780

Config:
URL="http://localhost :8086/"
Cookie="JSESSIONID=F66D0F63033496C466213E73A52D3C98"
Method="GET"
Header=""
Token_Names=['org.apache.struts.taglib.html.TOKEN ',' _infobase_token ']
Ignore_Form_Fields=[['reqCode ', '_history_id_ ']]

105 Successful test cases:
http://localhost:8086/ quickSearch.do?reqCode=quickSearch&
_modname_=infobase&searchType=text&parameter=a Form1 has a strong token: _infobase_token
http://localhost:8086/commons/favorites/Favorite/create.do? reqCode=create&favoriteMenu=true
Form1 has a strongtoken: org.apache.struts.taglib.html.TOKEN
http://localhost:8086/commons/favorites/Favorite/create.do? reqCode=create&favoriteMenu=true
Form1 has a strong token: _infobase_token
...

43 Failed test cases:
http://localhost:8086/info.do?reqCode=menuAction Form1 has a weak token:
org.apache.struts.taglib.html.TOKEN
http://localhost:8086/system/registry/search.do?reqCode= search Form1 has a weak token:
org.apache.struts.taglib.html.TOKEN
http://localhost:8086/system/layout/search.do?reqCode= search Form1 has a weak token:
org.apache.struts.taglib.html.TOKEN
...
```

10.3.6 File enumeration

The File Enumeration model of figure 71 was applied to InfoBase, with an instantiation library containing common directory and file names, in order to detect hidden interfaces to the system or forgotten backup files.

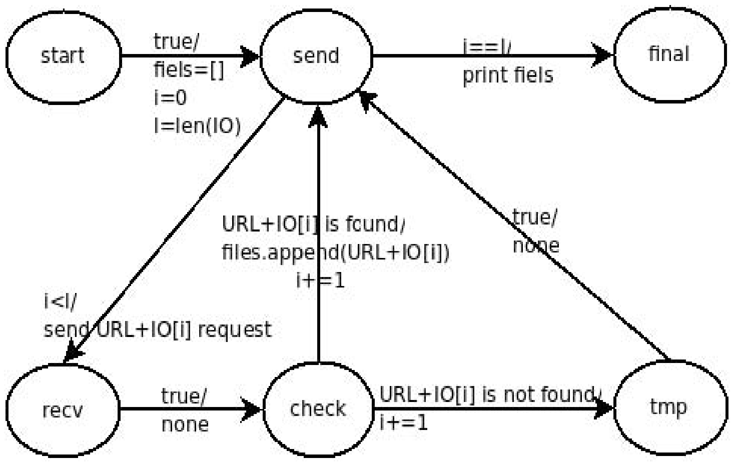


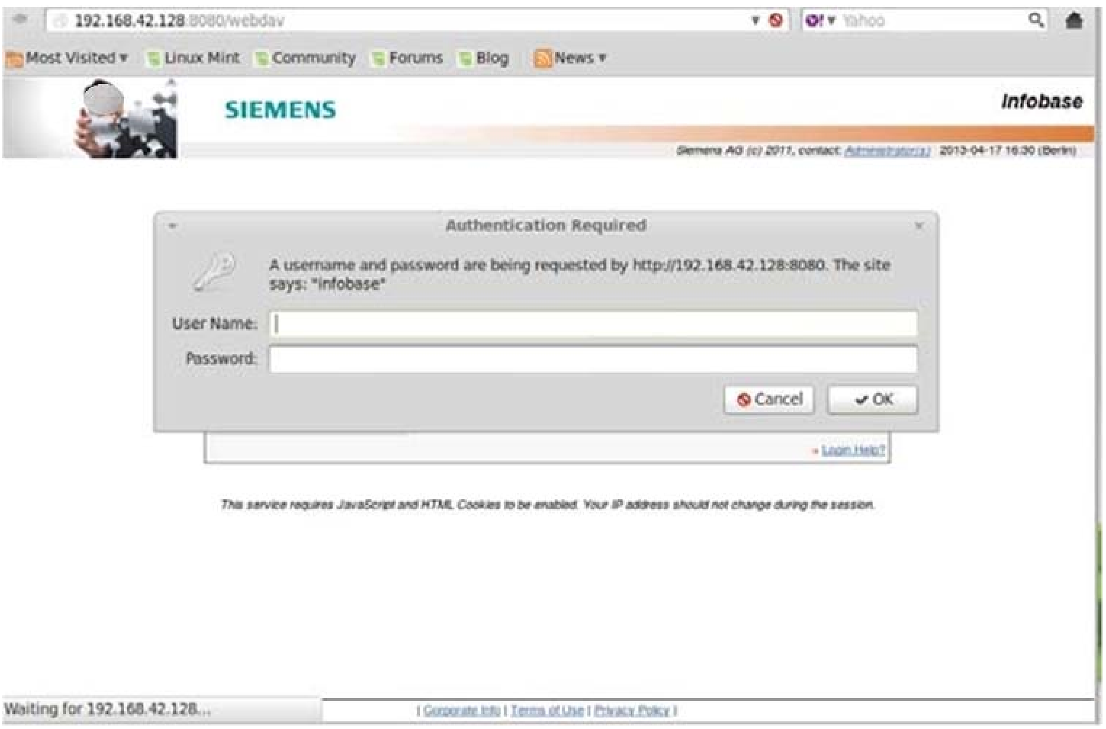
Figure 71: File enumeration model

The model checks if the tested file is found by looking at the response code. Initially, all responses different to 404 (not found) were marked as successful. As a result, the following restricted files (response code 401) were obtained, possibly belonging to the webserver management interface:

```
http://192.168.42.128:8080/webdav
http://192.168.42.128:8080/webdav/index.html
http://192.168.42.128:8080/webdav/index.html
http://192.168.42.128:8080/webdav/servlet/org.apache.catalina.servlets. WebdavServlet/
http://192.168.42.128:8080/webdav/servlet/webdav/
http://192.168.42.128:8080/webdav/index.html
```

By manually accessing those pages an authentication interface was obtained as shown in figure 72. Those files are potentially interesting for attackers to expand their attack surface: for instance, by trying password brute forcing or standard passwords. This issue was not further tested.

```
http://localhost:8085/info2www’(../../../../../../../bin/mailroot</etc/ passwd> Status: 505
http://localhost:8085/scripts/slxweb.dll/getfile?type=Library&file= [invalidfilename] Status: 505
http://localhost:8085/clusterframe.jsp Status: 200
http://localhost:8085/webdav Status: 401
http://localhost:8085/webdav/index.html Status: 401
http://localhost:8085/nsn/..%5Cutil/copy.bas Status: 400
http://localhost:8085/nsn/..%5Cutil/del.bas Status: 400
```



Siemens AG (c) 2011, contact: Administrator(s) 2013-04-17 16:30 (Berlin)

A username and password are being requested by http://192.168.42.128:8080. The site says: "infobase"

This service requires JavaScript and HTML Cookies to be enabled. Your IP address should not change during the session.

Waiting for 192.168.42.128...

Figure 72: An "Authentication Required" dialog appears by request the above mentioned URLs

On the other hand, several false positives were also obtained. All URLs with response status 200 400 401 505 are marked as successful, but in this case these URLs are not available. For example:

```
http://localhost:8085/info2www’(../../../../../../../bin/mailroot</etc/ passwd> Status: 505
http://localhost:8085/scripts/slxweb.dll/getfile?type=Library&file= [invalidfilename] Status: 505
http://localhost:8085/clusterframe.jsp Status: 200
http://localhost:8085/webdav Status: 401
http://localhost:8085/webdav/index.html Status: 401
http://localhost:8085/nsn/..%5Cutil/copy.bas Status: 400
http://localhost:8085/nsn/..%5Cutil/del.bas Status: 400
```

After manually checking those addresses, it was found that the server configuration automatically returned certain response codes to those particular URLs, but the files were not physically present in the server.

10.4 Summary and conclusions

In the third project year of SPaCIoS, the various approaches and technologies from the project were improved further and most importantly integrated into a common SPaCIoS Tool environment. Using this tool environment, a number of testing exercises were executed on the suggested application scenarios. Most tools and technologies could be applied to more than one application scenario. In the following, the results from the individual tools and technologies are summarized.

SPaCiTE, mutation-based testing and test execution The approach was applied in the application scenario of the document management application. The tool SPaCiTE is well integrated into the overall SPaCIoS Tool environment. Based on the ASLan security model and a selection of mutants to be applied the tool generates automatically AATs. After manual inspection of these AATs, they can be implemented and made executable using external test technologies, e.g. as demonstrated in the document management application scenario.

VERA, low-level attacker models and testing The VERA tool was successfully applied on the document management application. In general, positive experiences were made due to its ease of use and simplicity to setup the tool. However the results from its interactive application are prone to false positives that require manual checks afterwards.

Formalization of problem cases Several application scenarios were subjected to further formalization of selected problem cases. Updated ASLan++ models were obtained from the document management application. It proves that ASLan++ is an effective language to model and formulate security related behavior and properties.

Using the workflow of the SPaCIoS Tool it was possible to create a seamless methodology for model-based testing of security properties and vulnerability-driven testing from attacker models. The upcoming SPaCIoS deliverable D5.4 [i.8] will discuss further the performance of the different tools and technologies in more detail and compare it to existing work in security testing.

11 Evaluation and assessment of case study results

To analyse the effectiveness of the model-based security testing techniques, tools and methods the DIAMONDS project has developed a profiling and assessment scheme called STIP (Security Testing Improvement Profile), which allows an objective, detailed analysis and evaluation of the case studies. The scheme allows an assessment of model-based security testing processes and shows how security testing techniques, tools and methods fit together. Finally STIP may be used to provide recommendations for other on how to pragmatically integrate results from research to improve security-testing processes on hand. STIP was applied to all of the case studies in the DIAMONDS project and to the two case studies from the SPaCIoS project. The approach can be used to effectively assess and compare model-based security software testing processes and develop process improvements by leveraging the maturity of such a process in certain key areas.

11.1 Approach: Security Testing Improvements Profiling (STIP)

The Security Testing Improvement Profiling Scheme (STIP Scheme) has been developed in the DIAMONDS project to assess the maturity and performance of the case studies and their model-based security testing processes. The approach was based on the general ideas of TMMi [i.12] and TPI™ [i.13], [i.14]. Thus, a selected set of key areas were defined as considered relevant for model-based security testing. The key areas describe major aspects or activities in a security testing process and are chosen in that way, that they are aligned with the DIAMONDS MBST methodology and that they cover the most relevant DIAMONDS innovations. The key areas were defined to be self-contained and distinct so that each of the areas represents a relevant aspect of a MBST process. Table 4 lists the key areas that have been defined to assess the DIAMONDS project and are the ones that build the basis for the STIP Evaluation.

Table 4: Key areas in security testing

Key area	Description
Security risk assessment	Security risk assessment is a process for identifying security risks.
Security test identification	Test identification is the process of identifying test purposes and appropriate security testing methods, techniques and tools.
Automated generation of test models	For model-based security testing (e.g. fuzzing, mutation based testing) various kinds of models are required, which can be either created manually or generated automatically.
Security test generation	Security test generation is about the automation of security test design.
Fuzzing	Fuzzing is about injecting invalid or random inputs in order to reveal unexpected behave or to identify errors and expose potential vulnerabilities.
Security test execution automation	The automation of security test execution conducts the automatic application of malicious data to the SUT, the automatic assessment of the SUT's state and output to clearly identify a security flaw, and the automatic control of the test execution with respect to different kind of coverage.
Security passive testing/ security monitoring	Security monitoring based on passive testing consists of detecting errors, vulnerabilities and security flaws in a system under test (SUT) or in operation by observing its behaviour (input/output) without interfering with its normal operations.
Static security testing	Static security testing involves analysing application without executing it. One of the main components is code analysis.
Security test tool integration	Tool integration is the ability of tools to cooperate with respect to data interchange.

For each of the key areas a four level performance scale was defined with levels that are hierarchically organized and build on each other. The levels can be used to evaluate concrete security testing processes with respect to their performance in the belonging key area. Each level with a higher number represents an improvement for the underlying security testing process.

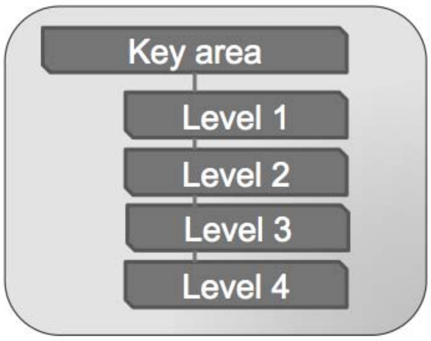


Figure 73: STIP performance level scheme

STIP provides an objective, detailed analysis and evaluation of the DIAMONDS research & development. It shows how tools, techniques and methodologies fit together and provide recommendations for other on how to pragmatically integrate the results to improve security-testing processes on hand. Each higher level is better than its prior level in terms of time (faster), money (cheaper) and/or quality (better). The key areas and the respective maturity levels are described in the following clauses.

11.1.1 Security risk assessment

Security risk assessment is a process for identifying security risks consisting of the following steps: establishing context, security risk identification, security risk estimation, security risk evaluation, and security risk treatment.

Table 5: Progress level for security risk assessment

#	Name	Description
L1	Informal security risk assessment	Security risk assessment is a process for identifying security risks consisting of the following steps: establishing context, security risk identification, security risk estimation, security risk evaluation, and security risk treatment.
L2	Model-based security risk assessment	At this level, the security risk assessment is conducted in an unstructured manner without a specific notation/language for document risk assessment results or a clearly defined process for conducting the security risk assessment.
L3	Model and test-based security risk assessment	At this level, the security risk assessment is conducted with a language for documenting assessment results and a clearly defined process for conducting the assessment.
L4	Automated model and test-based security risk assessment	At this level, the model-based security risk assessment is uses testing for verifying the correctness of the risk assessment results.

11.1.2 Security test identification

Test identification is the process of identifying test purposes and appropriate security testing methods, techniques and tools. This can either be done by means of analysing the requirements of a system or by taking additional sources of information on the system, the relevance of its features and its environment (e.g. threat models, security risk assessments).

Table 6: Progress level for security test identification

#	Name	Description
L1	Security test identification based on requirements analysis	Test identification can be based on the analysis of the functional security requirements (SFR) and their coverage through testing. Often these requirements have priority numbers that additionally provide guidance on the importance of a requirement and the related test purpose.
L2	Security test identification based on threat/vulnerability models	Security threat/vulnerability models additionally allow for the identification of penetration tests that are based on estimations on potential threats and potential vulnerabilities. This allows testing for unwanted incidents that are not covered by the security functional requirements.
L3	Security test identification based on risk models and test pattern	The combination of risk models and security test pattern additionally provides best practices for the identification and selection of testing means dedicated to well-known classes of threats or vulnerabilities. This approach provides extensive guidance to identify adequate test purposes and to apply approved security testing methods, techniques and tools.
L4	Risk-based security test identification + prioritization	Risk-based security test identification and prioritization combines the advantages of Level 3 with a prioritization of the test purposes by considering probabilities of the unwanted incident and estimations on their consequences (quantified security risks). The integration of test identification with security risk assessment allows for a problem and business specific prioritization of the identified tests purposes and testing approaches.

11.1.3 Automated generation of test models

For model-based security testing (e.g. fuzzing, mutation based testing), a template or various levels of behavioural models are required. These templates or models can be either created manually or generated automatically from the system's input and output.

Table 7: Progress level for automated generation of test models

#	Name	Description
L1	Block-based structural intelligence (stateless)	The template (file or network traffic) is automatically converted into a flat model consisting of data elements such as Type-Value pairs, like in HTTP header values, or web form data. Recognition of basic data types such as strings and integers. No intelligence on data sub-structures, sequences or dynamic content is used.
L2	State-aware or sequence-aware models (stateful)	Sequences of messages are converted into sequence diagrams or state-charts, with message names or purposes automatically added as meta data into the model.
L3	Structural model (stateful)	The template (file or network traffic) is automatically converted into a structural multi-level meta-model that understands data values, their substructures, and can understand the protocol layers in the message sequences (IPv4, TCP, HTTP, XML). Testing can be targeted to a specific layer.
L4	Automatically generated full behavioural model (stateful)	The template (file or network traffic) is automatically converted into a full behavioural model that understands functional elements such as length fields, check-sums, and other complex data elements such as URLs in the structure. Encodings and decodings are performed automatically. Sequences are included in the models, and variables between messages within a sequence or between sequences can be used.

11.1.4 Security test generation

Security test generation is about the automation of security test design. The initial level consists in a fully manual design of security tests, and the higher level consists in an optimized security test generation process, including a complete coverage of targeted security properties and/or vulnerabilities.

Table 8: Progress level for security test generation

#	Name	Description
L1	Fully manual security test design	At this level, security test engineers define and execute manually security tests. This first level of maturity is labour-extensive and bound to the ingenuity of single security test engineers.
L2	Supported security test design	At this level, security test design activity is supported by some tools, such as intrusive proxies (e.g. OWASP WebScarab?), or ad-hoc testware, to help them to develop security tests. The coverage of security properties and vulnerabilities is not fully controlled and the results are still bound to the ingenuity of security test engineers.
L3	Dynamic Application Security Testing	At this level, tools like web application vulnerability scanners ensure an automated test generation and execution based on security test patterns. This ensures a systematic discovery of known vulnerabilities (depending of the capabilities of the tool-set). The limit is the blindness of such tools, that does not use any behavioural knowledge of the application. This leads to false positive and false negative.
L4	Automated Model-based security testing	At this level, modelling of attacks, security test patterns and behavioural aspects of the System Under Test leads to an automated test generation of accurate and precise security tests. This level use a continuous process from security risk assessment to automated test generation supported by modelling activities.

11.1.5 Fuzzing

Fuzzing is about injecting invalid or random inputs in order to reveal unexpected behaviour to identify errors and expose potential vulnerabilities. Ideally, fuzzers generate semi-valid input data, i.e. input data that is invalid only in small portions. Depending on fuzzer's knowledge about the protocol, fuzzers can generate totally invalid to semi-valid input data.

Table 9: Progress level for fuzzing

#	Name	Description
L1	Random data fuzzing	Random-based fuzzers generate randomly input data. They do not know nearly anything about the SUT's protocol.
L2	Model-based data fuzzing	Model-based fuzzers employ a model of the protocol. The model is executed on- or offline to generate complex interactions with the SUT. Thus, it is possible to fuzz data after passing a particular point (e.g. after authentication).
L3	Model-based evolutionary fuzzing	Model-based evolutionary fuzzers learn the mutations of the protocol by feeding the SUT with data and interpreting its responses or other information available from the SUT by using evolutionary algorithms. Model-based evolutionary fuzzing complements model-based data fuzzing by optimizing the fuzzing with respect to information gained from the SUT.
L4	Model-based data and behavioural fuzzing	Model based data and behavioural fuzzing combines data fuzzing with behaviour fuzzing. Behaviour fuzzing addresses a complete other class of vulnerabilities by stimulating the SUT with invalid sequences of messages. This allows for additional identify flaws in the security functionality e.g. vulnerabilities in the authentication logic.

11.1.6 Security test execution automation

During active security testing, the test environment applies malicious input data based on attack scenarios in order to find existing security flaws. The automation of security test execution conducts the automatic application of malicious data to the SUT, the automatic assessment of the SUT's state and output to clearly identify a security flaw, and the automatic control of the test execution with respect to source code coverage, data coverage, or other kind of coverage that are gained by extensively monitoring the SUT.

Table 10: Progress level for security test execution

#	Name	Description
L1	Manual security testing	The initial level intents to stress the system with manual attack scenarios.
L2	Automated application of test scenarios (black box)	At this level test cases are implemented as test scripts that stimulate the SUT with malicious scenarios and data. Scenarios or data are either implemented or generated beforehand or generated on the fly.
L3	Automated assessment of the system's output	The test scripts are applied to control the stimulation of the system as well as the automated assessment of the SUT's state and output to clearly identify misbehaviour and unwanted incidents.
L4	Automated assessment of the system's internal states (e.g. code/data coverage)	The test scripts are applied to control the stimulation of the system as well as the automated assessment of the SUT's state and output to clearly identify misbehaviour and unwanted incidents. Additionally the test environment controls code or data related coverage criteria by additional sources of information (e.g. instrumentation of the SUT).

11.1.7 Security passive testing/ security monitoring

Security monitoring based on passive testing consists of detecting errors, vulnerabilities and security flaws in a system under test (SUT) or in operation by observing its behaviour (input/output) without interfering with its normal operations (no external stimulations).

Table 11: Progress level for security passive testing/security monitoring

#	Name	Description
L1	Security Information and Event Management (SIEM) and Business Activity Monitoring (BAM)	SIEM and BAM technology provides real-time analysis of security alerts generated by networks and applications. SIEM/BAM solutions come as software, appliances or managed services, and are also used to log security data and generate reports for compliance purposes. This type of solution is installed at the system level and, in general, is customized for the targeted business.
L2	Signature based analysis and anomaly based analysis.	Intrusion detection techniques can be divided into signature-based and anomaly-based. In signature-based schemes given patterns are searched for, limiting the detection to known attacks. In anomaly-based schemes the goal is to detect behaviour that is deemed abnormal.
L3	Context aware security monitoring and model driven analysis	The security analysis is based on the monitoring of events obtained from different levels (physical environment, hardware, network, operating system, end-user specific applications, etc.). The analysis correlates different events to detect complex attack behaviours.
L4	Intelligent monitoring for security checking	To be able to detect 0-days attacks, intelligent monitoring uses techniques, such as statistics, performance evaluation, and machine learning to improve intrusion and anomaly detection. An example of a machine learning technique used is supervised learning based on regression or classification analysis. Regression analysis allows modelling the interaction and relation between different variables using a mathematical equation. Statistical classification allows identifying to which predefined group a new observation belongs to.

11.1.8 Static security testing

Static security testing involves analysing application without executing it. The main objective of static security testing is to find vulnerabilities in the applications that are caused by code level bugs, missing functionality, configuration error etc. One of the main components is code analysis. The code could be source code (in higher languages like C/C++/Java™, etc.) or compiled binary code (in x86 assembly code or Java byte code, for example). The main advantage of static analysis is the whole execution coverage of the application. However, it suffers from false positives.

Table 12: Progress level for static security testing

#	Name	Description
L1	High Level Threat Model	A high level design diagram (e.g. Use-case) is analysed to understand the overall architecture of the application. One of the main objectives is to produce a data flow diagram and a class/module dependency diagram (at higher level, e.g. between modules/classes etc.) of the application. By analysing such high level diagrams, various security mechanisms can be identified that should be in place e.g. session management, cryptographic primitives etc.
L2	Input Output Data Validation	Most of the time, application makes use of well-known library functions for getting input and then performs specific operations on that data by again calling well known library functions. However, if not used with caution, such functions can make application vulnerable. Therefore, at all points in the code where these functions are called, it is necessary to make sure that the input data is validated as per the "expected data properties" (interface specifications, preconditions while calling a particular function etc.). This is also termed as "input sanitization". This analysis can be done manually or by automated tools.
L3	Intra- and Inter-procedural Analysis	At this level, the analysis gets more complex as it addresses the issues that are at low-level when compared to above levels. Intra-procedural analysis examines each function of the application to verify various dataflow and control flow related properties, like data dependence, buffer overflow, null pointer usage etc. Inter-procedural analysis extends dataflow and control flow analysis across functions. This analysis can detect insecure usage of input data by tracing it across functions (i.e. by computing information-flow). This analysis can be used to establish "non-interference" which is a well-known technique for access control. When performed at binary level, this analysis may also be useful for analysing malware embedded binaries (with limited scope due to various factors like code obfuscation, self-modification, virtualization etc.).
L4	Validating Security Vulnerabilities i.e. exploitability of errors	At this level, the emphasis is on reducing the false positives that are produced at Level 3. Using techniques like symbolic execution and static taint analysis, it can be established that certain inputs are feasible to exploit weakness in the application. When coupled with concurrent execution, the process can even be more practical to be applied on real world large applications. The results from this level can be used to prioritize the patching mechanism.

11.1.9 Security test tool integration

Tool integration is the ability of tools to cooperate. Typically, tools work on their own data structures that are well suited to the task, which needs to be performed with or by the tool. So the tool can only process data that is relevant for the tool. Tools can save and load their internal data to a file which may have a proprietary format. In such cases it is very difficult to make use of the tool specific data in a different context than the respective tools. So the question is how to transfer the data between the tools

Table 13: Progress level for security test tool integration

#	Name	Description
L1	Separated tools	No integration. All tools work separately. Tools do not always need to be integrated. If a tool has a good user interface that is consistent with the host platform, does not share or require data other tools, or has limited relationships with other resources, there may be no need to integrate it with other tools. However, if larger processes are considered this is not feasible for all of the tools.
L2	Bilateral tool coupling (tool coalition)	A tool coalition is based on point-to-point connection between tools. Tool coalitions are often used in small and ad-hoc environments but have problems when it comes to more tools and larger environments (no scalability).
L3	Common data model and traceability tool federation	Tool federations are based on a central integration platforms and repositories that provides a common set of data to be exchanged and respective interfaces. Tool federations better fit to larger tool environments because the existence of a common set of interfaces eases the integration of new tools. However, the definition of a common data set and common interfaces is more complex as defining bilateral tool couplings.
L4	Live cycle support	Live cycle support is focusing not only on data exchange but on how tools may interact in order to support specific activities in a development or testing process. For this kind of integration a common data model is complemented with a life cycle model that specifies the activities and the roles of tools with respect to the activities. Besides interfaces for data exchange, the tools provide interfaces that propagate life cycle events, which are used trigger actions in other tools. Tool integration platforms with live cycle support pose strict integration requirements on the tools to be integrated.

11.2 Evaluation results: STIP evaluation of the Case Studies

The DIAMONDS project has carried out eight case studies that show the applicability of the DIAMONDS innovations in relevant industrial domains like Banking, Smart Cards, Industrial Automation, Radio Protocols, Transport/Automotive, and Critical Infrastructures. The STIP approach has been used to evaluate all of the DIAMONDS case studies. To explicitly show the progress that has been made during the DIAMONDS project, two assessments were carried out for each case study. The first assessment explicitly considers the application of the DIAMONDS techniques & tools and thus provides us an impression of the security testing processes in the case studies at the end of the DIAMONDS project. The second assessment intentionally disregards the results DIAMONDS and thus gives us an impression of the maturity of the testing processes before DIAMONDS.

11.2.1 Evaluation of the banknote processing machine case study

Banknote processing machines are used in central, large and medium banks and also in CITs (cash in transport) and other organizations that handle large amounts of banknotes. These machines are usually configured to work in a network. During the DIAMONDS project the focus of security tests has been on two major subsystems of a banknote processing machine, the currency processor and the reconciliation station. The currency processor as well as the reconciliation station was provided as virtual machines, where external interfaces are replaced by simulation. The main focus of the research applied to this case study have been the development of techniques and tools for risk-based security testing as well as model-based fuzz testing and their integration into an integrated platform for traceability and security test automation. The overall approach can be summarized as follows: A comprehensive model based security risk assessment that indicates potential threats, vulnerabilities, and incidents as well as related probabilities and consequences, is used as a basis for the identification and selection of appropriate security test pattern. These pattern cover security testing best practices in a domain independent a reusable way, for example, the application of fuzz testing techniques like MBBF. Once identified, the most appropriate security testing approach is applied and assessed with respect to the risk values from the security risk assessment.

In order to assess the results of the DIAMONDS project on the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 74 shows the score before the project started in red and after the project in blue. The case study advanced in nearly every aspect of model-based security testing.

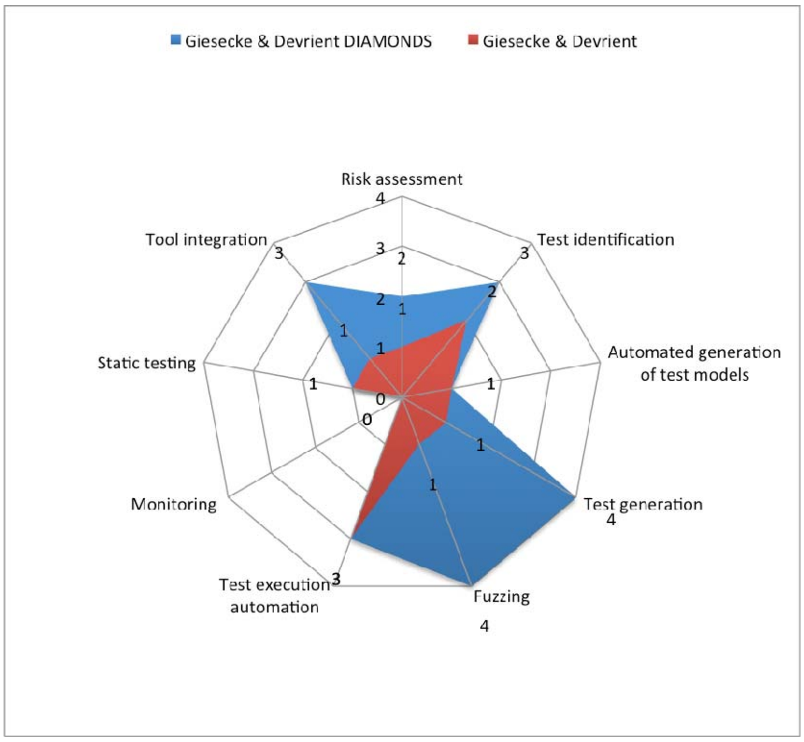


Figure 74: Security Test Improvement Profile Comparison of the Banknote Processing Machine Case Study

Therefore the case study gained from nearly all developments of the DIAMONDS project with the exception of monitoring. The biggest gains were made in the areas where the case study was used as a driver for the research project. Moreover the case study provided an interesting field to research and application of security testing techniques.

11.2.2 Evaluation of the banking case study

In order to assess the results of the DIAMONDS project on the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 75 shows the score before the project started in red and after the project in blue.

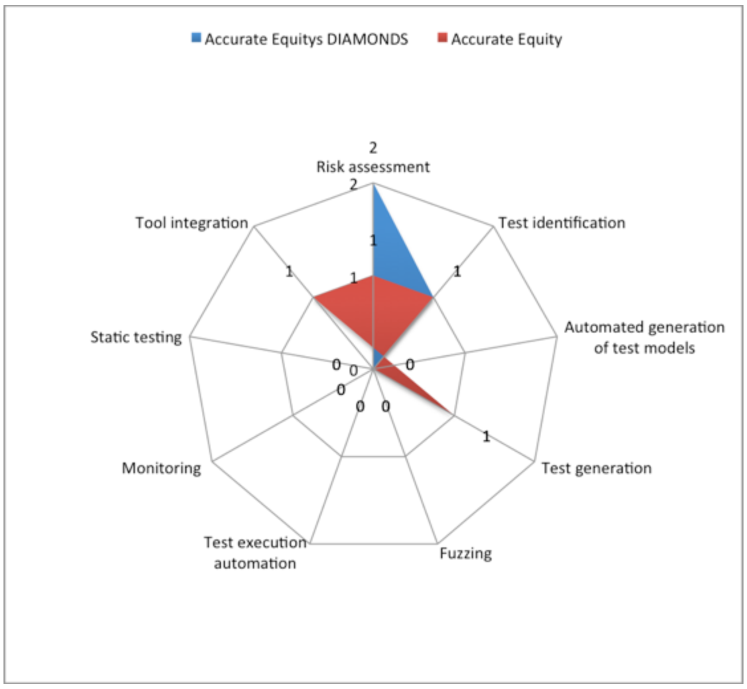


Figure 75: Security Test Improvement Profile Comparison of the Banking Case Study

11.2.3 Evaluation of the radio protocol case study

In order to assess the results of the DIAMONDS project on the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 76 shows the score before the project started in red and after the project in blue.

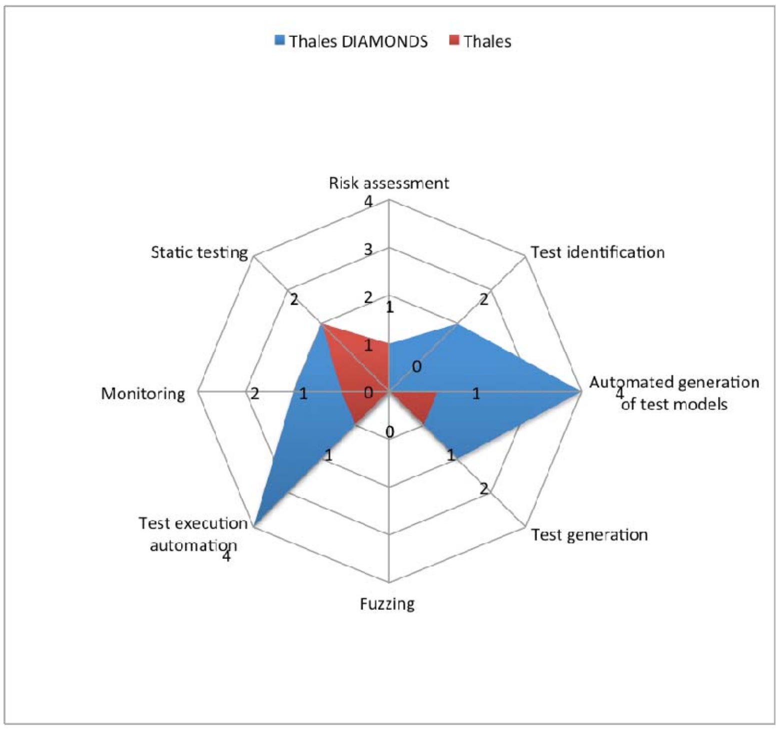


Figure 76: Security Test Improvement Profile Comparison of the Radio Protocol Case Study

11.2.4 Evaluation of the automotive case study

In order to assess the results of the DIAMONDS project on the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 77 shows the score before the project started in red and after the project in blue.

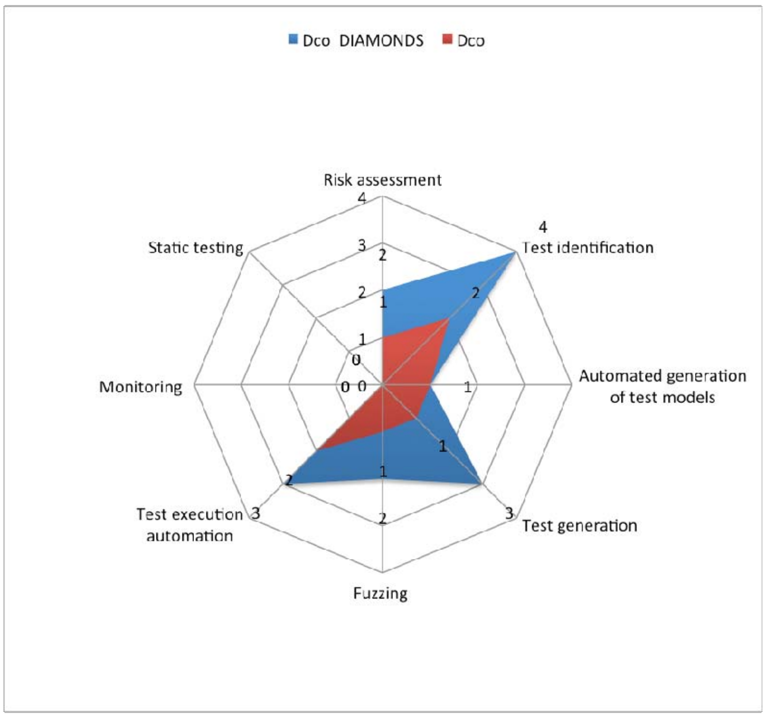


Figure 77: Security Test Improvement Profile Comparison on the Automotive Case Study

11.2.5 Evaluation of the eHealth case study

eHealth is an area of rapid innovation. Many different solutions are being discussed that intend to integrate mobile Patient Monitoring and centralized or distributed electronic Health Records Management systems. Siemens is developing, testing, and assessing the security of different variants to implement such a system. The one used in this study incorporates device credentials bootstrapping (via device pairing) and two important privacy principles (two-factor user authentication and patient consent) in a user-friendly solution.

The main focus of the research applied to this case study has been the development of techniques and tools for model-based security risk assessment as well as attacker-model low level vulnerability testing. The overall approach can be summarized as follows: A comprehensive model of the application was constructed (using ASLan++) and model-checked to verify that no design errors were present in the model. After a couple of revisions a final design was chosen and implemented. Based on the model, the possible attacker interfaces and general strategies and possible implementation faults (low-level vulnerabilities) were determined. Once identified, the most appropriate security testing approach was applied (using the VERA tool), that revealed in fact the presence of implementation faults creating low-level vulnerabilities. Standard, advanced fuzzing tools were also used, not in the scope of the project. Static monitoring and static testing were not used.

In order to assess the results of the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 78 shows the score before the project started in red and after the project in blue. The case study advanced in nearly every aspect of model-based security testing.

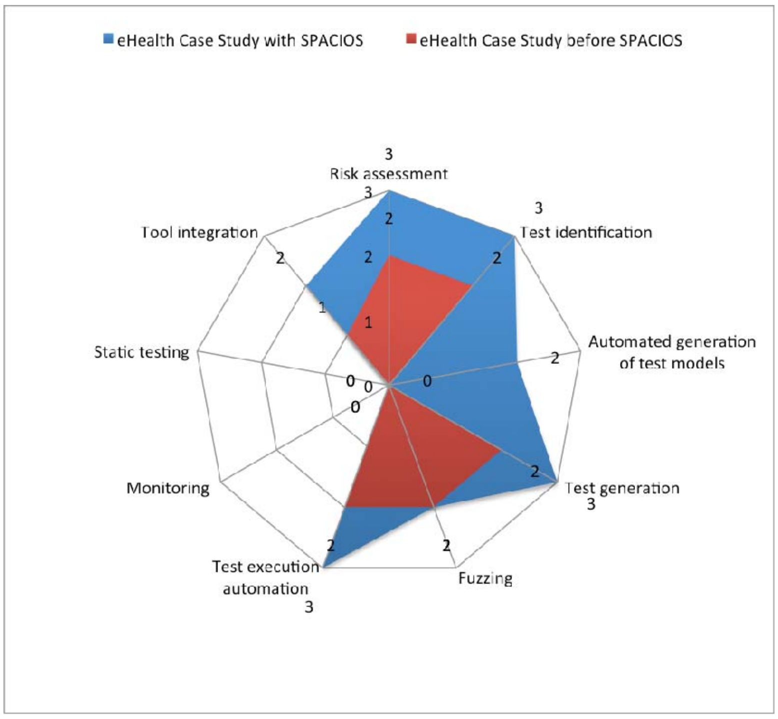


Figure 78: Security Test Improvement Profile Comparison on the eHealth Case Study

11.2.6 Evaluation of the document management case study

The Infobase Document Repository (IDR) is a document management system that supports the collaboration of different users, with different security levels and from different administrative domains. The management and sharing of documents or other types of files is done via a web browser. It is provided to offer a platform for joint projects involving external partners. The repository mechanism supports web-based administration of text and binary files of any kind, e.g. text documents, spreadsheet tables, and even executables, in a hierarchical storage structure. The system includes a fine-granular access control for the different users, groups, and company, where each object in the repository can be bound to different access rights.

The overall approach and the main focus of the research applied to this case study have been threefold: 1) the development of techniques and tools for a model-based security risk assessment based on annotated, technical Data-Flow diagrams, 2) a mutation-based testing approach, where ASLan++ models were mutated to feed into a test-generator and test-driver, and, finally, 3) an attacker-model low level vulnerability-based testing procedure.

The can be summarized as follows: A data-flow model was constructed and analyzed for technical indicators of possible attack points. Then a comprehensive model of the (already existing) application was constructed (using ASLan++) and model-checked to verify that no design errors were present. Based on the two models (DFD and ASLan++) the possible attacker interfaces and general strategies and possible implementation faults (low-level vulnerabilities) were determined. Once identified, the most appropriate security testing approach was applied (using the VERA tool), that revealed in fact the presence of implementation faults creating low-level vulnerabilities. Standard, advanced fuzzing tools were also used, not in the scope of the project. Static monitoring and static testing were not used.

In order to assess the results of the SPaCIoS project on the case study one can look at the Security Test Improvement Profile (STIP) before the start of the project and now. Figure 79 shows the score before the project started in red and after the project in blue. The case study advanced in nearly every aspect of model-based security testing.

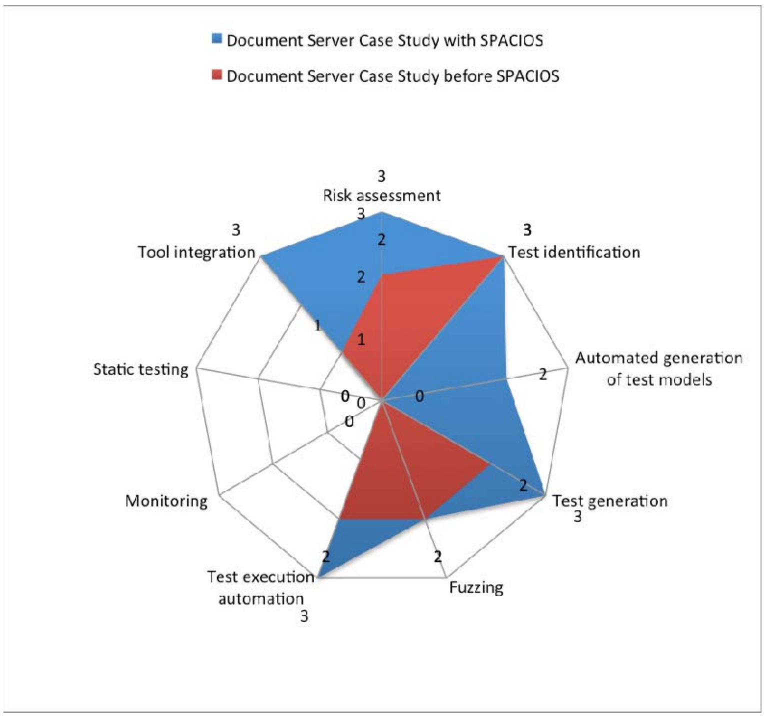


Figure 79: Security Test Improvement Profile Comparison on the Document Server Case Study

Annex A: Bibliography

ITEA2 DIAMONDS Deliverable D5.WP1: "Final Case Study Results", 2013.

SPaCIoS. Deliverable 2.2.1: "Method for assessing and retrieving models", 2013.

SPaCIoS. Deliverable 2.2.2: "Combined black-box and white-box model inference", 2013.

SPaCIoS. Deliverable 2.3.1: "Definition and Description of Security Goals", 2012.

SPaCIoS. Deliverable 2.4.1: "Definition of Attacker Behavior Models", 2012.

SPaCIoS. Deliverable 3.2: "SPaCIoS Methodology and technology for property-driven security testing", 2013.

SPaCIoS. Deliverable 4.2: "SPaCIoS Tool v.1 and Validation methodology patterns (final version)", 2012.

SOGETI: Website of SOGETI, 2009.

NOTE: Available at http://www.sogeti.nl/.

TMMi Foundation, Website of the TMMi Foundation.

NOTE: Available at http://www.tmmi.org/.

History

Document history		
V1.1.1	June 2014	Publication