2nd UCAAT User Conference on Advanced Automated Testing

September 16-18 2014, Munich, Germany

ETSI World Class Standards

Model-Based Security Testing with Test Patterns

Julien BOTELLA (Smartesting)
Jürgen GROSSMANN (FOKUS)
Bruno LEGEARD (Smartesting)
Fabien PEUREUX (Smartesting)
Martin SCHNEIDER (FOKUS)
Fredrik SEEHUSEN (SINTEF)

http://www.rasenproject.eu/

RASEN Compositional Risk Assessment and Security Testing of Networked Systems

Fraunhofer FOKUS SINTEF smartesting® Optimize your Test Center SEVENTH FRAMEWORK PROGRAMME

Agenda

- Context, motivation and objectives
- Approach for Risk-Based Security Testing
- Illustration of the end-to-end process
- Conclusion and future work

Context

- *FP7 RASEN project (2012-2015)*

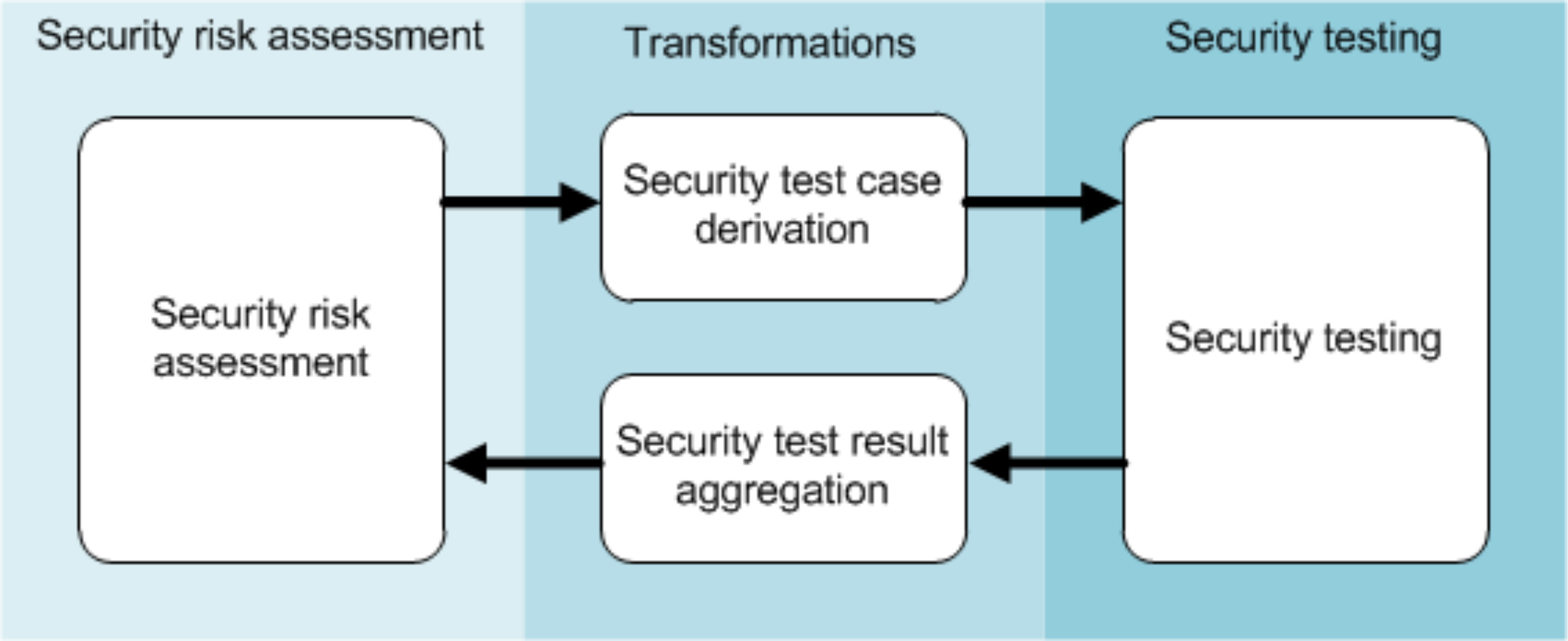
RASEN SEVENTH FRAMEWORK PROGRAMME

Compositional Risk Assessment and Security Testing of Networked Systems

SINTEF
Fraunhofer FOKUS
smartesting Optimize your Test Center
software AG
infoworld CONNECTING HEALTHCARE
UiO Universitetet i Oslo
EVRY

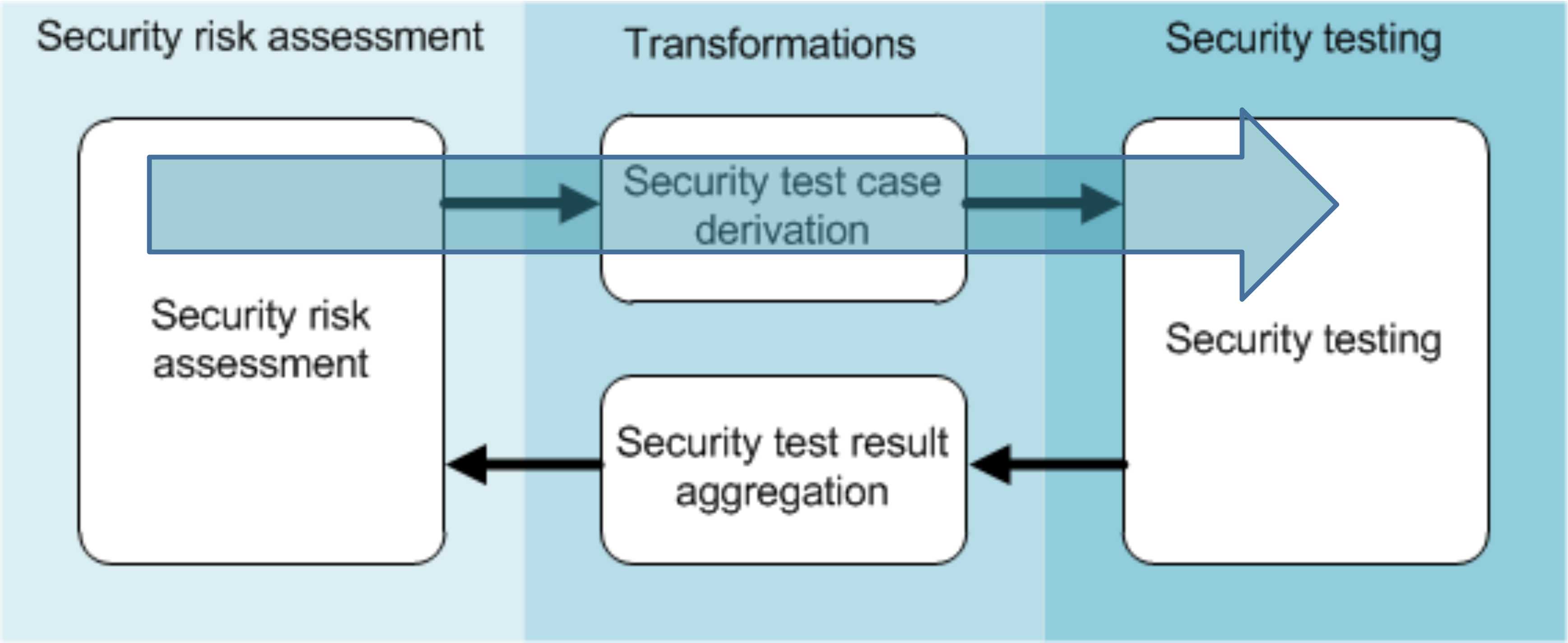
- Strengthen European organisations' ability to conduct security assessment of large scale networked systems
  - taking into account the context in which the system is used, such as liability, legal, organisational and technical issues,
  - through the combination of compliance management security risk assessment and security testing.

Motivation of the RASEN project

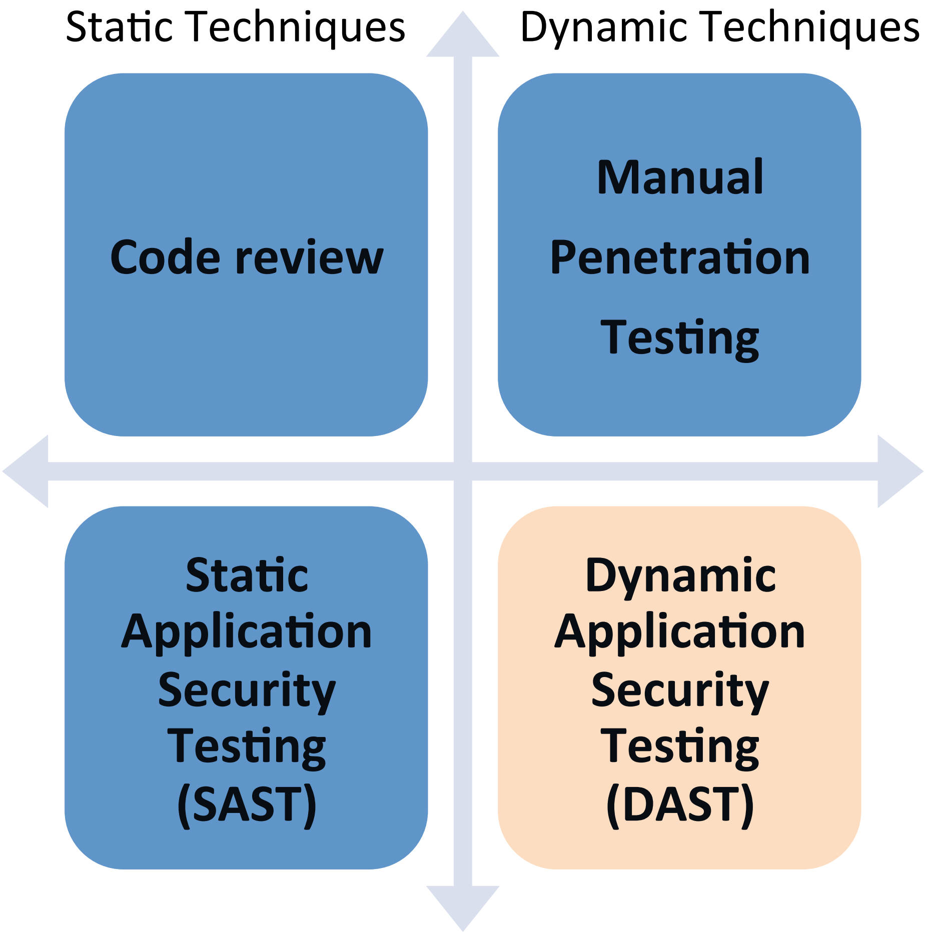


Contents of the presentation

- Security and risk-based testing approach to guide the security testing using a systematic derivation of test cases from risk assessment results.



Security testing: state of the practice


Static Techniques
Dynamic Techniques
Code review
Manual Penetration Testing
Static Application Security Testing (SAST)
Dynamic Application Security Testing (DAST)

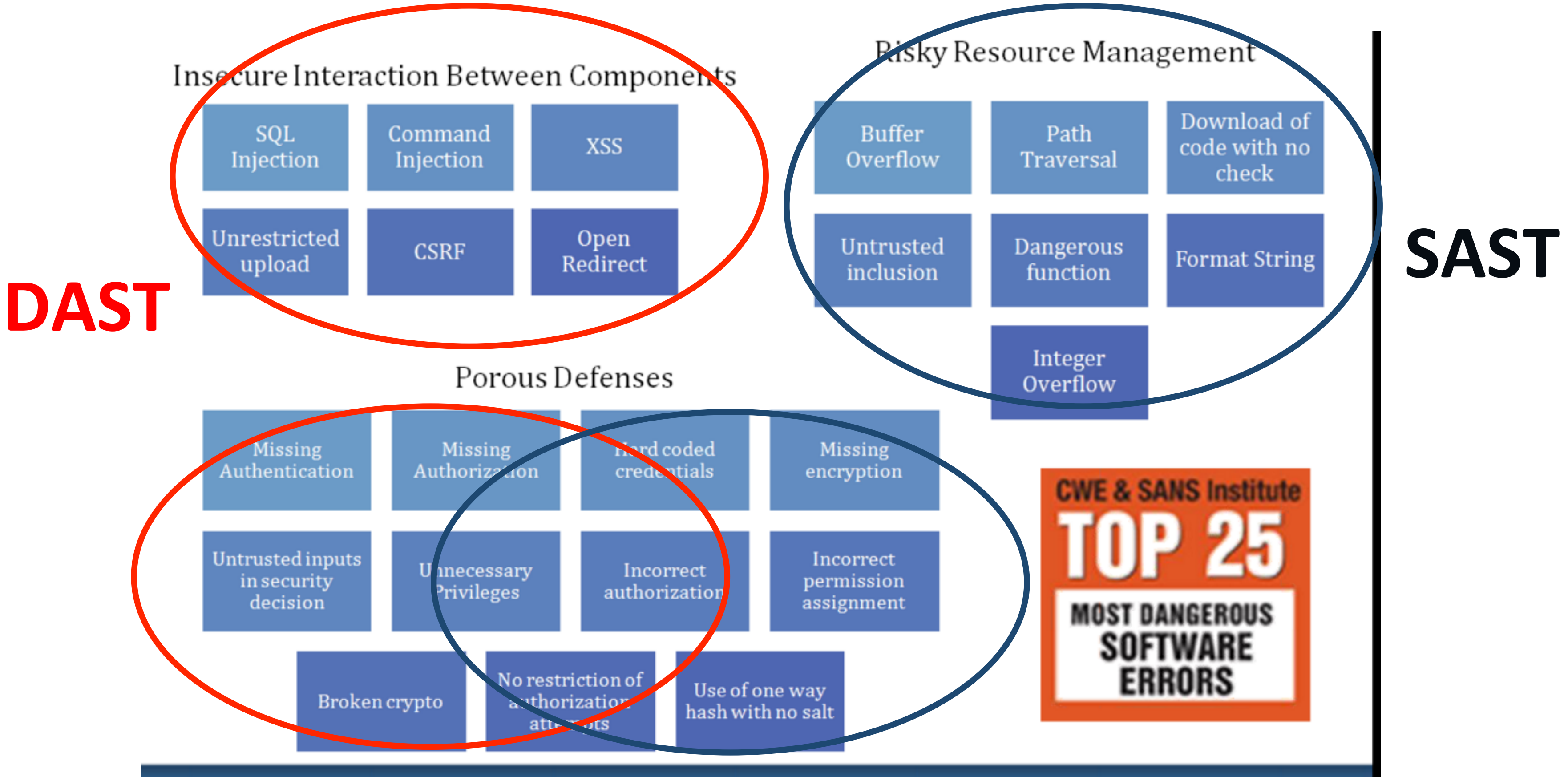
Manual Techniques

Automated Techniques

Intrusive proxies (Burp suite, Webscarab, …)

Vulnerability Scanners, Fuzzing tools, …

SAST vs DAST – Top 25 / 2011



DAST

SAST

Insecure Interaction Between Components

SQL Injection

Command Injection

XSS

Unrestricted upload

CSRF

Open Redirect

Risky Resource Management

Buffer Overflow

Path Traversal

Download of code with no check

Untrusted inclusion

Dangerous function

Format String

Integer Overflow

Porous Defenses

Missing Authentication

Missing Authorization

Hard coded credentials

Missing encryption

Untrusted inputs in security decision

Unnecessary Privileges

Incorrect authorization

Incorrect permission assignment

Broken crypto

No restriction of authorization attempts

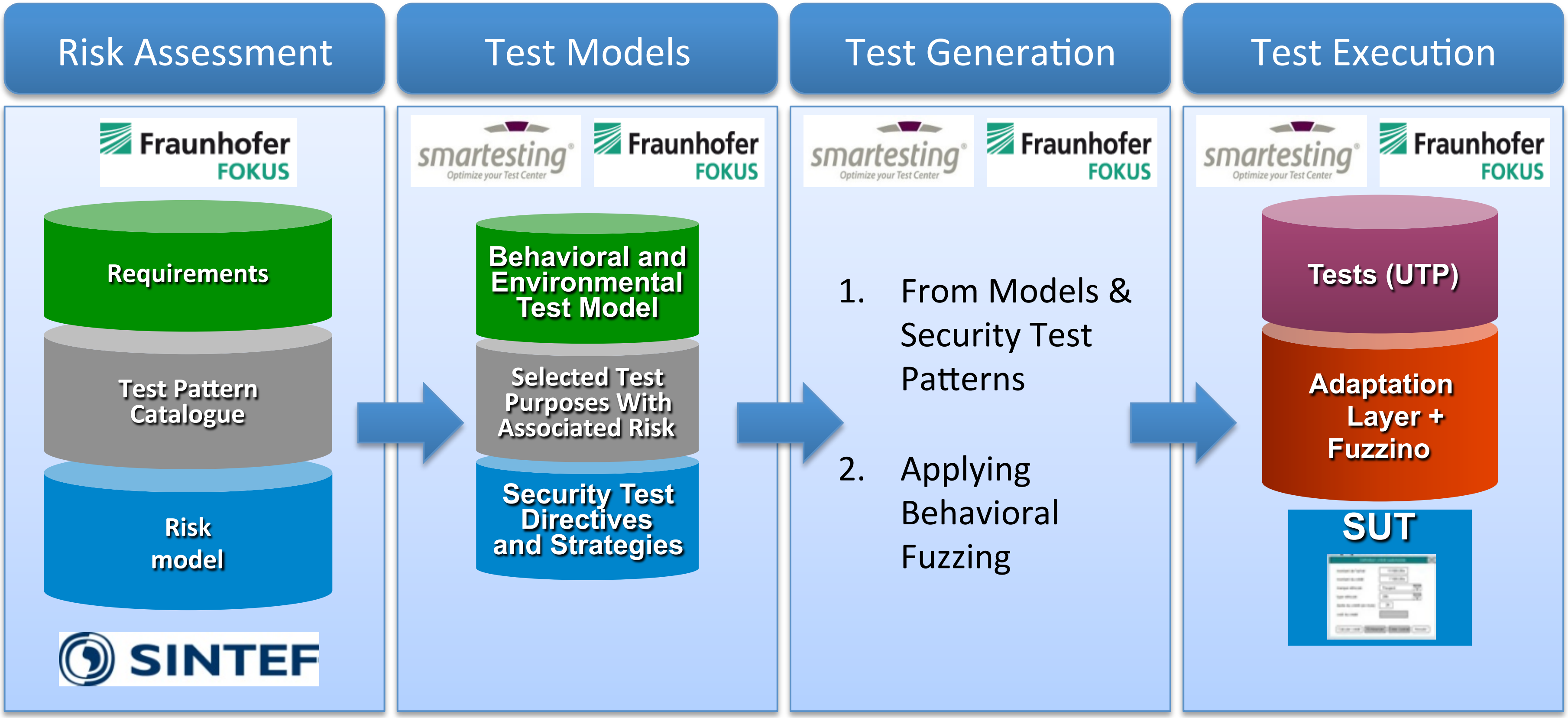
Use of one way hash with no salt

CWE & SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS

Objectives of the testing approach

- To provide a systematic guidance for DAST security testing techniques from risk assessment
- To automate test case derivation and execution using model-based security testing techniques
- To support compositional analysis to manage large scale networked system in complex environments

Risk-Based Security Testing process



Risk Assessment

Fraunhofer FOKUS

Requirements

Test Pattern Catalogue

Risk model

SINTEF

Test Models

smartesting Optimize your Test Center

Fraunhofer FOKUS

Behavioral and Environmental Test Model

Selected Test Purposes With Associated Risk

Security Test Directives and Strategies

Test Generation

smartesting Optimize your Test Center

Fraunhofer FOKUS

1. From Models & Security Test Patterns
2. Applying Behavioral Fuzzing

Test Execution

smartesting Optimize your Test Center

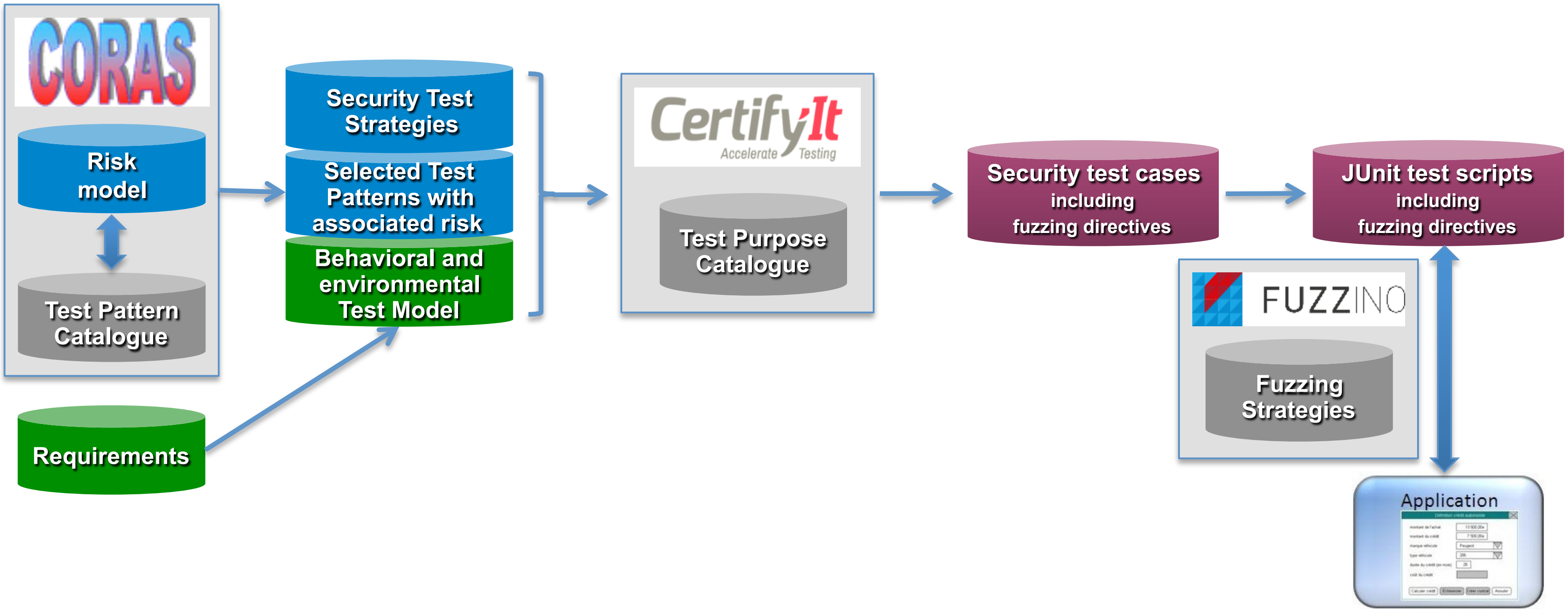
Fraunhofer FOKUS

Tests (UTP)

Adaptation Layer + Fuzzino

SUT

RASEN toolchain overview



CORAS

Risk model

Test Pattern Catalogue

Requirements

Security Test Strategies

Selected Test Patterns with associated risk

Behavioral and environmental Test Model

CertifyIt Accelerate Testing

Test Purpose Catalogue

Security test cases including fuzzing directives

JUnit test scripts including fuzzing directives

FUZZINO

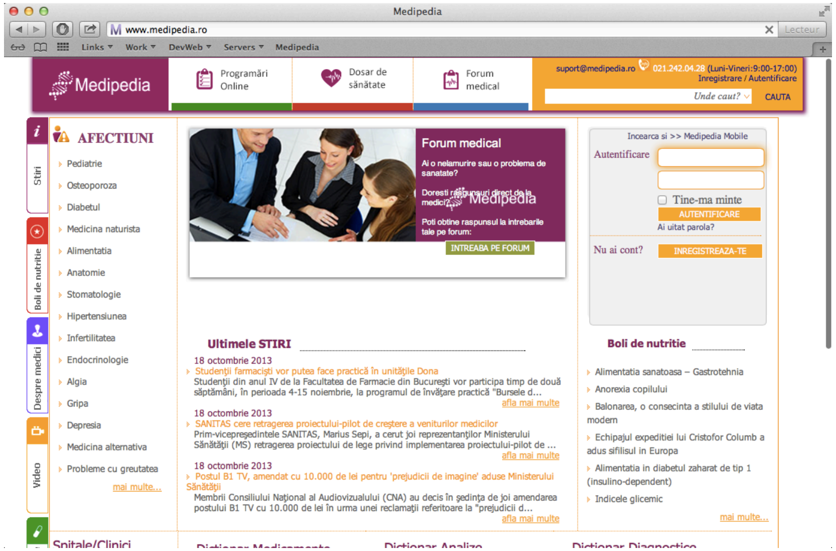
Fuzzing Strategies

Application

Use case: InfoWorld MediPedia

Medipedia is a web service that:

- allows patients to collect and organize all medical information, from multiple healthcare providers in a single health record,
- provides both public and secured username and password based access (public and secured information managing patient medical records).



suport@medipedia.ro 021.242.04.28 (Luni-Vineri:9:00-17:00)
Inregistrare / Autentificare
Unde caut?

AFECTIUNI

- Pediatrie
- Osteoporoza
- Diabetul
- Medicina naturista
- Alimentatia
- Anatomie
- Stomatologie
- Hipertensiunea
- Infertilitatea
- Endocrinologie
- Algia
- Gripa
- Depresia
- Medicina alternativa
- Probleme cu greutatea

mai multe...

Forum medical

Ai o nelamurire sau o problema de sanatate?

Doresti raspunsuri direct de la medici?

Poti obtine raspunsul la intrebarile tale pe forum:

Ultimele STIRI

18 octombrie 2013
Studenţii farmacişti vor putea face practică în unităţile Dona
Studenţii din anul IV de la Facultatea de Farmacie din Bucureşti vor participa timp de două săptămâni, în perioada 4-15 noiembrie, la programul de învăţare practică "Bursele d...
afla mai multe

18 octombrie 2013
SANITAS cere retragerea proiectului-pilot de creştere a veniturilor medicilor
Prim-vicepreşedintele SANITAS, Marius Sepi, a cerut joi reprezentanţilor Ministerului Sănătăţii (MS) retragerea proiectului de lege privind implementarea proiectului-pilot de ...
afla mai multe

18 octombrie 2013
Postul B1 TV, amendat cu 10.000 de lei pentru 'prejudicii de imagine' aduse Ministerului Sănătăţii
Membrii Consiliului Naţional al Audiovizualului (CNA) au decis în şedinţa de joi amendarea postului B1 TV cu 10.000 de lei în urma unei reclamaţii referitoare la "prejudicii d...
afla mai multe

Incearca si >> Medipedia Mobile

Autentificare

□ Tine-ma minte

Ai uitat parola?

Nu ai cont?

Boli de nutritie

- Alimentatia sanatoasa – Gastrotehnia
- Anorexia copilului
- Balonarea, o consecinta a stilului de viata modern
- Echipajul expeditiei lui Cristofor Columb a adus sifilisul in Europa
- Alimentatia in diabetul zaharat de tip 1 (insulino-dependent)
- Indicele glicemic

mai multe...

Spitale/Clinici

Dictionar Medicamente

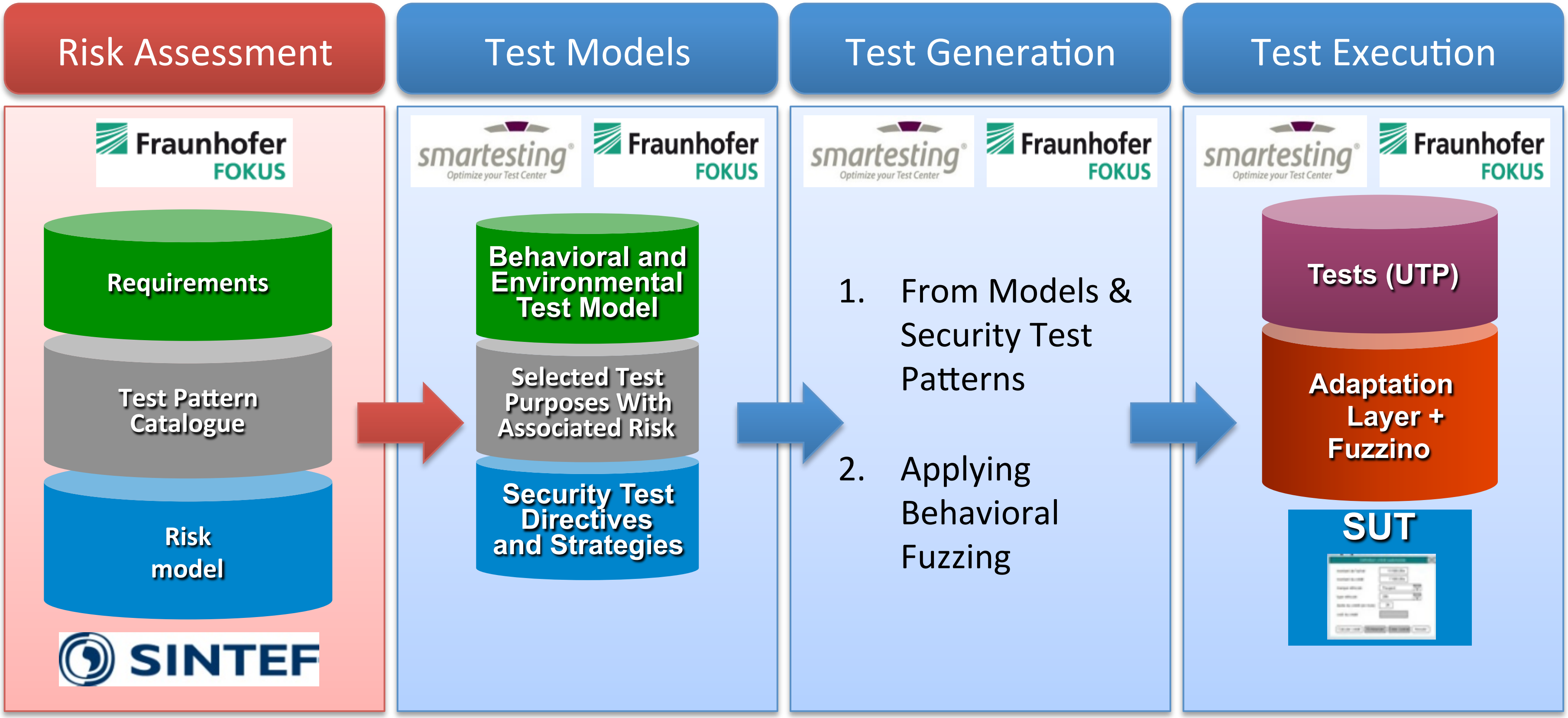
Dictionar Analize

Dictionar Diagnostice

infoworld
CONNECTING HEALTHCARE

MediPedia
The Medical Encyclopedia

1. Risk assessment inputs



Risk Assessment

Fraunhofer FOKUS

Requirements

Test Pattern Catalogue

Risk model

SINTEF

Test Models

smartesting Optimize your Test Center

Fraunhofer FOKUS

Behavioral and Environmental Test Model

Selected Test Purposes With Associated Risk

Security Test Directives and Strategies

Test Generation

smartesting Optimize your Test Center

Fraunhofer FOKUS

1. From Models & Security Test Patterns
2. Applying Behavioral Fuzzing

Test Execution

smartesting Optimize your Test Center

Fraunhofer FOKUS

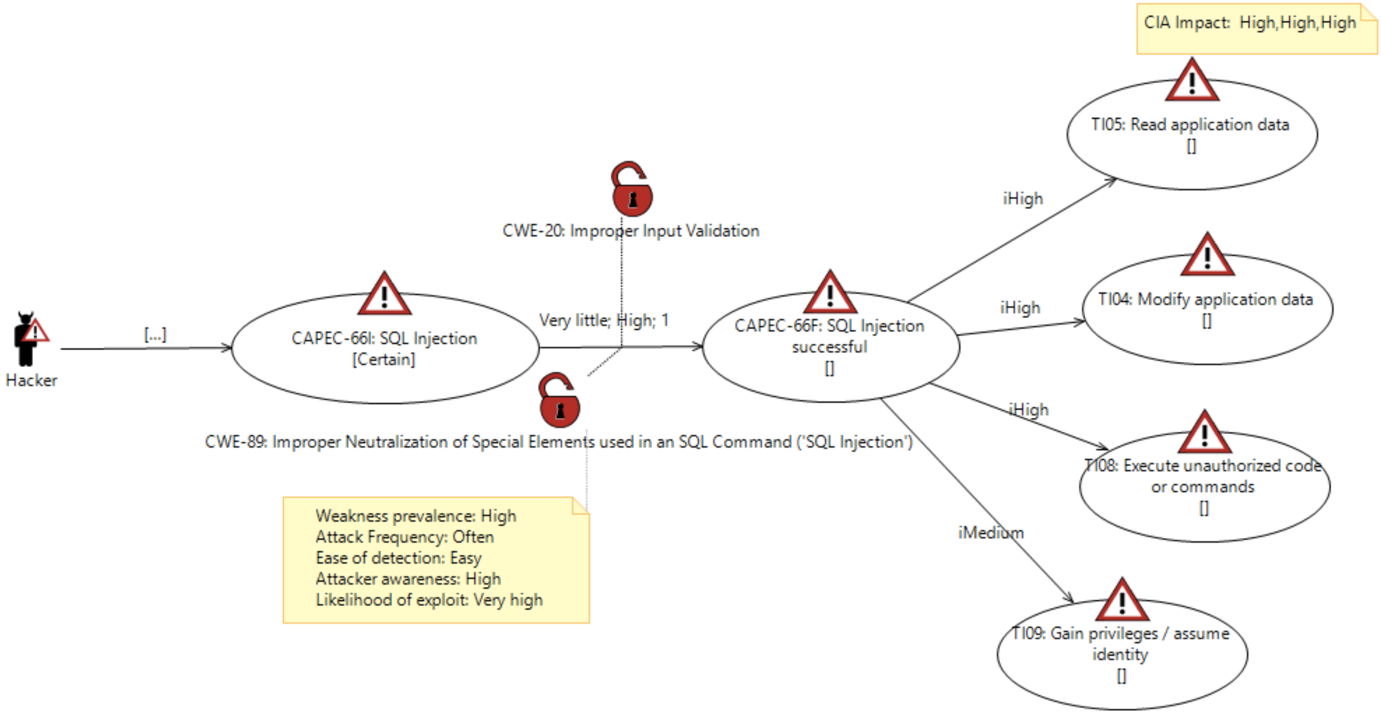
Tests (UTP)

Adaptation Layer + Fuzzino

SUT

Risk identification and prioritization

- Using the CORAS approach to provide test case identification and prioritization based on the risk analysis:
  - Definition of selected test procedures from identified risk
  - Prioritization of the test procedures regarding risk assessment results



CIA Impact: High,High,High

Hacker

[...]

CAPEC-66I: SQL Injection [Certain]

CWE-20: Improper Input Validation

Very little; High; 1

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Weakness prevalence: High
Attack Frequency: Often
Ease of detection: Easy
Attacker awareness: High
Likelihood of exploit: Very high

CAPEC-66F: SQL Injection successful []

iHigh

TI05: Read application data []

iHigh

TI04: Modify application data []

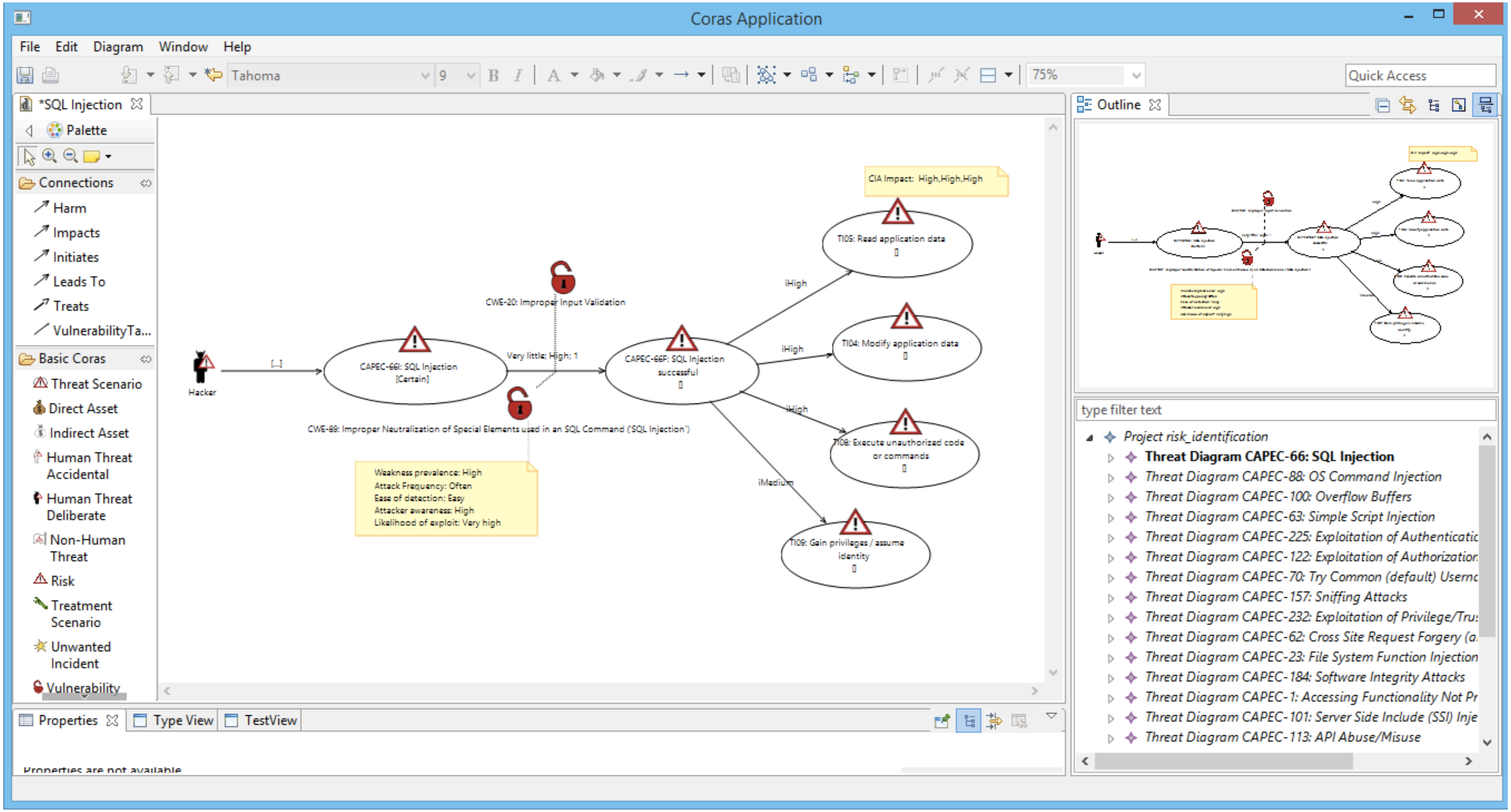
iHigh

TI08: Execute unauthorized code or commands []

iMedium

TI09: Gain privileges / assume identity []

Risk model in CORAS tool



CIA Impact: High,High,High

Weakness prevalence: High
Attack Frequency: Often
Ease of detection: Easy
Attacker awareness: High
Likelihood of exploit: Very high

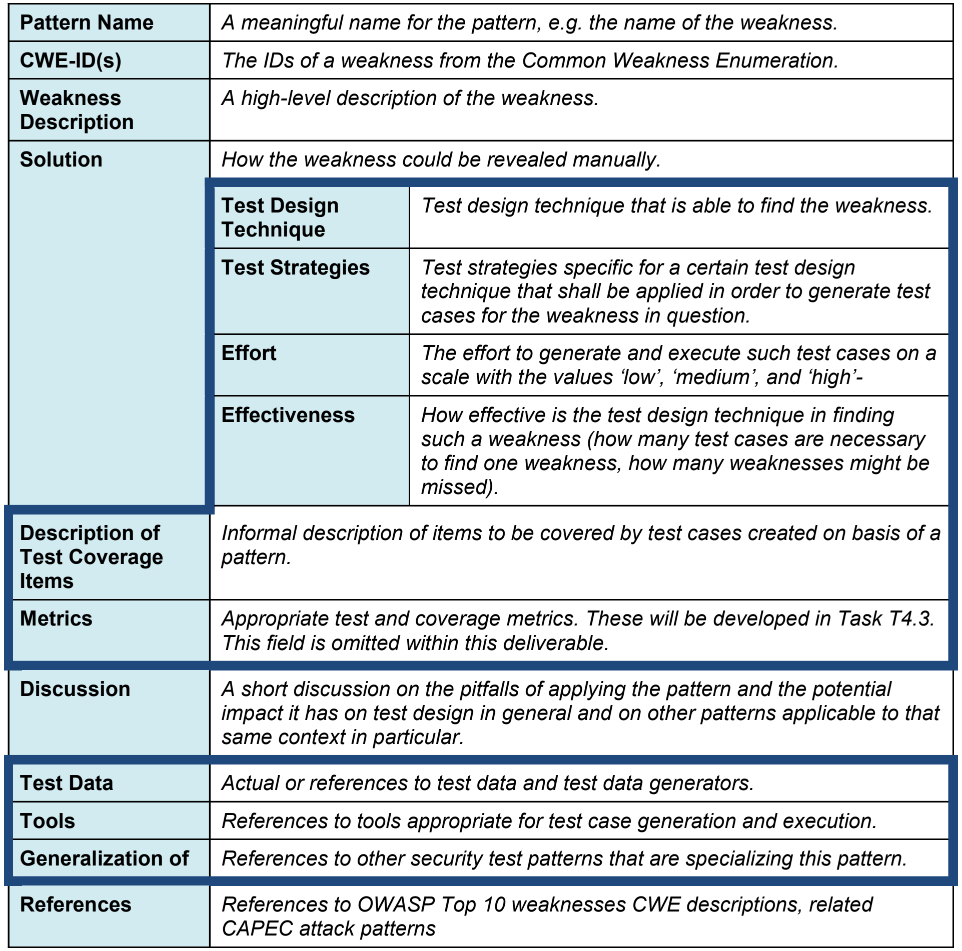
Project risk_identification
- **Threat Diagram CAPEC-66: SQL Injection**
- *Threat Diagram CAPEC-88: OS Command Injection*
- *Threat Diagram CAPEC-100: Overflow Buffers*
- *Threat Diagram CAPEC-63: Simple Script Injection*
- *Threat Diagram CAPEC-225: Exploitation of Authenticatic*
- *Threat Diagram CAPEC-122: Exploitation of Authorization*
- *Threat Diagram CAPEC-70: Try Common (default) Usernc*
- *Threat Diagram CAPEC-157: Sniffing Attacks*
- *Threat Diagram CAPEC-232: Exploitation of Privilege/Tru:*
- *Threat Diagram CAPEC-62: Cross Site Request Forgery (a.*
- *Threat Diagram CAPEC-23: File System Function Injection*
- *Threat Diagram CAPEC-184: Software Integrity Attacks*
- *Threat Diagram CAPEC-1: Accessing Functionality Not Pr*
- *Threat Diagram CAPEC-101: Server Side Include (SSI) Inje*
- *Threat Diagram CAPEC-113: API Abuse/Misuse*

Properties are not available

Link to Security Test Patterns

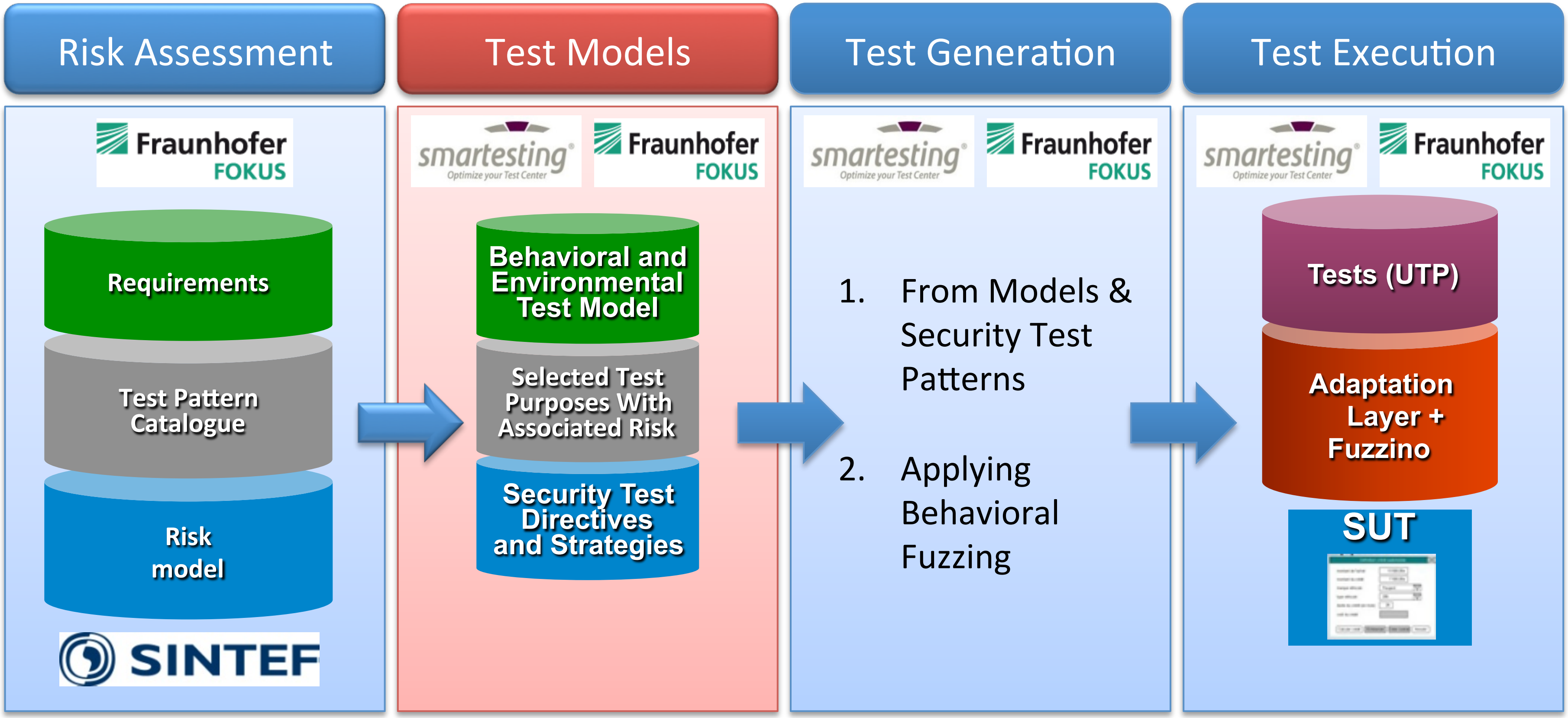
- Security test patterns are typically related to vulnerability catalogues
  - MITRE CWE & CAPEC
  - OWASP Top 10
- Solution
  - one or more **test design technique** and corresponding **strategies,** test **effort** and **effectiveness**
- Test Data
  - instructions for crafting test data
  - references to test data libraries or generators
- Tools
  - references tools that can be used to generate and execute such test cases

Generic



Pattern Name	*A meaningful name for the pattern, e.g. the name of the weakness.*	
CWE-ID(s)	*The IDs of a weakness from the Common Weakness Enumeration.*	
Weakness Description	*A high-level description of the weakness.*	
Solution	*How the weakness could be revealed manually.*	
	Test Design Technique	*Test design technique that is able to find the weakness.*
	Test Strategies	*Test strategies specific for a certain test design technique that shall be applied in order to generate test cases for the weakness in question.*
	Effort	*The effort to generate and execute such test cases on a scale with the values 'low', 'medium', and 'high'-*
	Effectiveness	*How effective is the test design technique in finding such a weakness (how many test cases are necessary to find one weakness, how many weaknesses might be missed).*
Description of Test Coverage Items	*Informal description of items to be covered by test cases created on basis of a pattern.*	
Metrics	*Appropriate test and coverage metrics. These will be developed in Task T4.3. This field is omitted within this deliverable.*	
Discussion	*A short discussion on the pitfalls of applying the pattern and the potential impact it has on test design in general and on other patterns applicable to that same context in particular.*	
Test Data	*Actual or references to test data and test data generators.*	
Tools	*References to tools appropriate for test case generation and execution.*	
Generalization of	*References to other security test patterns that are specializing this pattern.*	
References	*References to OWASP Top 10 weaknesses CWE descriptions, related CAPEC attack patterns*	

2. Test model design



Risk Assessment

Fraunhofer FOKUS

Requirements

Test Pattern Catalogue

Risk model

SINTEF

Test Models

smartesting Optimize your Test Center

Fraunhofer FOKUS

Behavioral and Environmental Test Model

Selected Test Purposes With Associated Risk

Security Test Directives and Strategies

Test Generation

smartesting Optimize your Test Center

Fraunhofer FOKUS

1. From Models & Security Test Patterns
2. Applying Behavioral Fuzzing

Test Execution

smartesting Optimize your Test Center

Fraunhofer FOKUS

Tests (UTP)

Adaptation Layer + Fuzzino

SUT

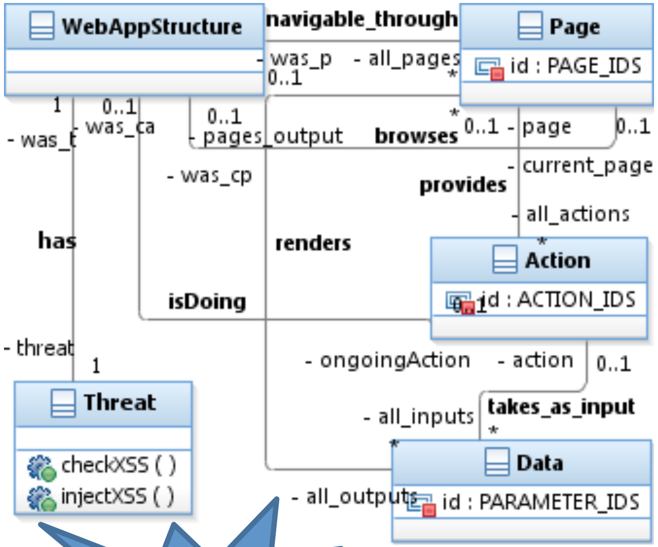
Test model and testing directives

- Testing artefacts are composed of:
  - A functional and behavioral model of the application under test
  - A set of test purposes, selected from risk assessment model (identification phase), to drive the test generation
  - The prioritization of the risk assessment model to apply an appropriate test coverage

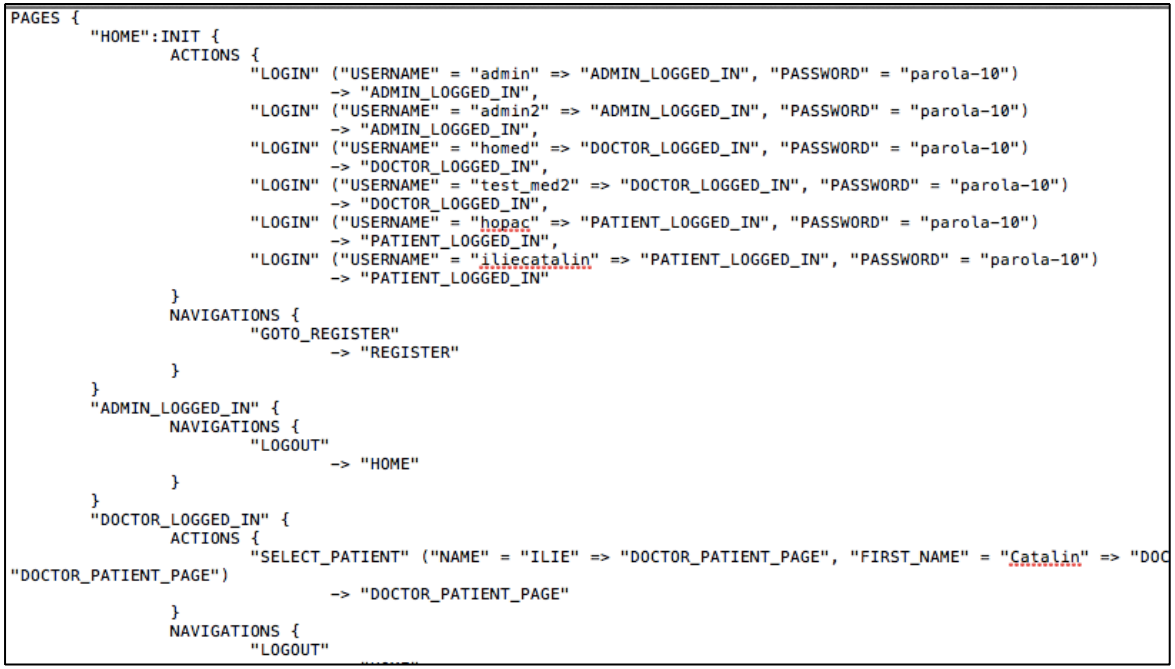
- Smartesting Test Purpose Language is used to represent Security Test Patterns into a machine-readable language:
  - Designed for security means
  - Textual language based on regular expressions
  - Reasons in term of states to be reached and operations to be called

Behavioral model design using DSML

- Behavioral modeling notation is based on UML metamodel:
  - Class diagrams specify the static structure (points of control and observation)
  - Object diagrams specify concrete business entities
  - State diagrams graphically describe its behavioural characteristics

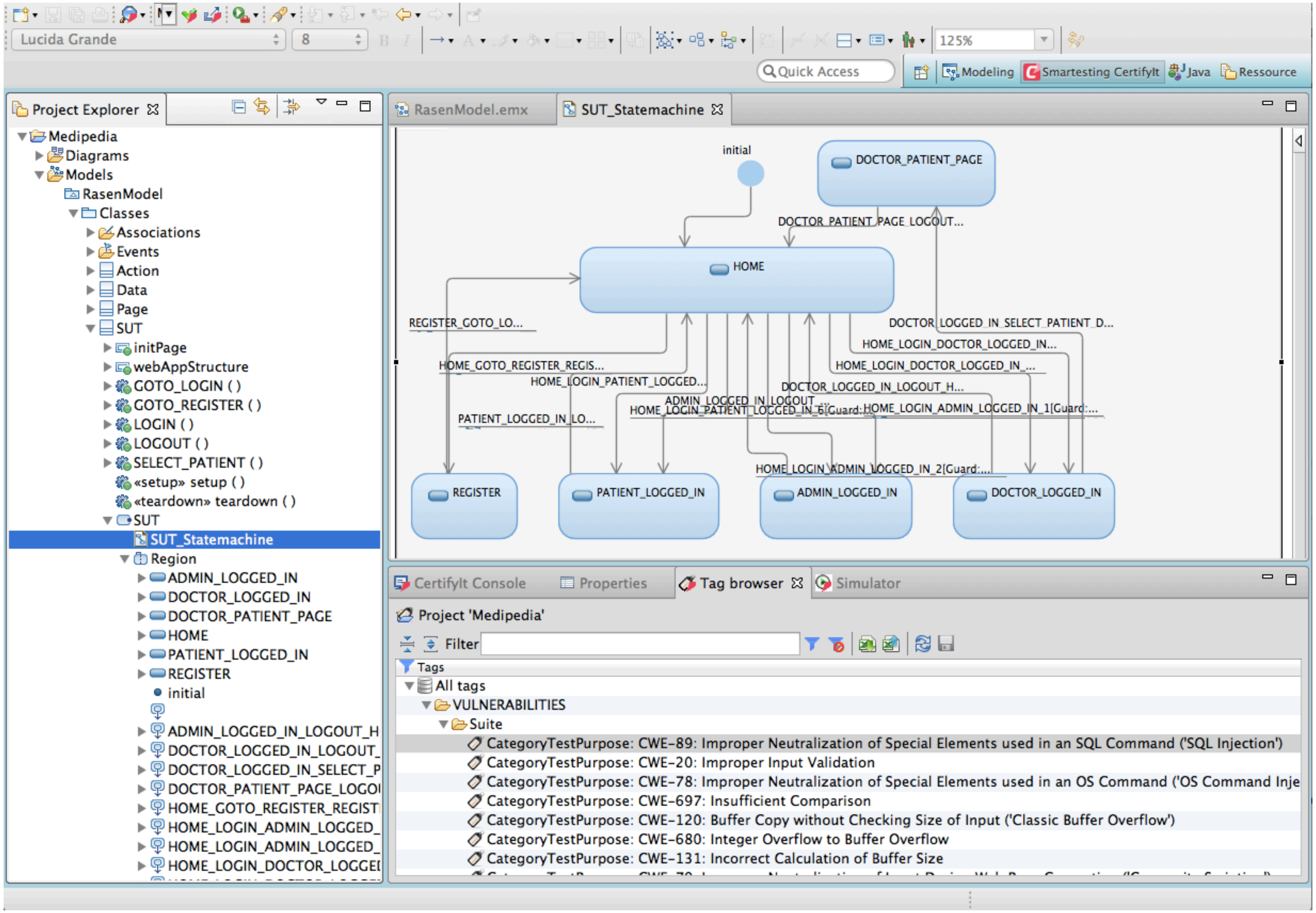


Generic



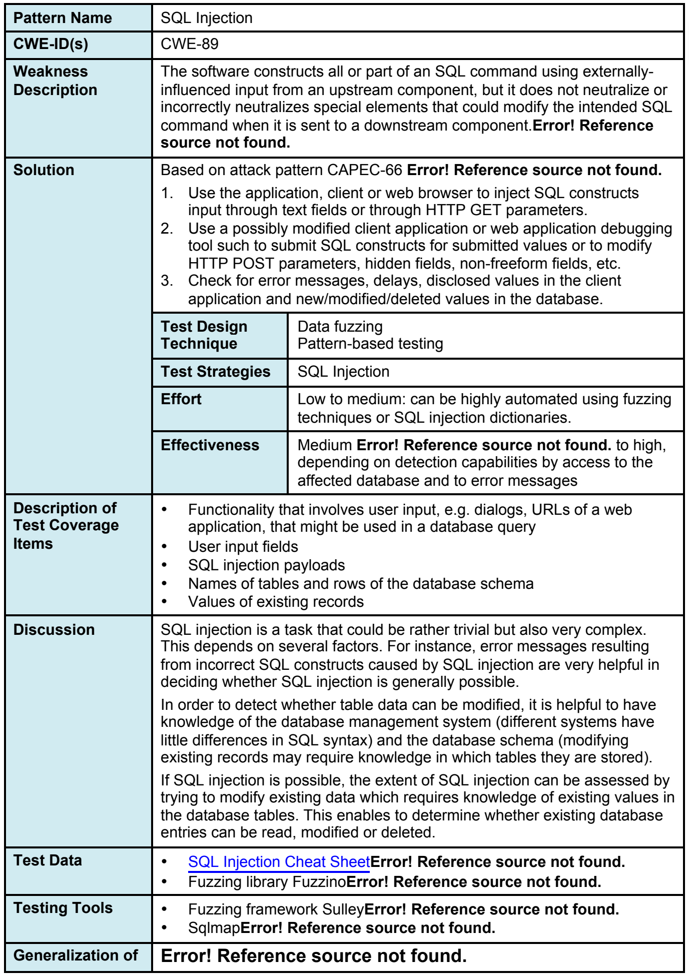
```
PAGES {
        "HOME":INIT {
                ACTIONS {
                        "LOGIN" ("USERNAME" = "admin" => "ADMIN_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "ADMIN_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "admin2" => "ADMIN_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "ADMIN_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "homed" => "DOCTOR_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "DOCTOR_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "test_med2" => "DOCTOR_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "DOCTOR_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "hopac" => "PATIENT_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "PATIENT_LOGGED_IN",
                        "LOGIN" ("USERNAME" = "iliecatalin" => "PATIENT_LOGGED_IN", "PASSWORD" = "parola-10")
                                -> "PATIENT_LOGGED_IN"
                }
                NAVIGATIONS {
                        "GOTO_REGISTER"
                                -> "REGISTER"
                }
        }
        "ADMIN_LOGGED_IN" {
                NAVIGATIONS {
                        "LOGOUT"
                                -> "HOME"
                }
        }
        "DOCTOR_LOGGED_IN" {
                ACTIONS {
                        "SELECT_PATIENT" ("NAME" = "ILIE" => "DOCTOR_PATIENT_PAGE", "FIRST_NAME" = "Catalin" => "DOC
"DOCTOR_PATIENT_PAGE")
                                -> "DOCTOR_PATIENT_PAGE"
                }
                NAVIGATIONS {
                        "LOGOUT"
```

State diagram from DSML



CategoryTestPurpose: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CategoryTestPurpose: CWE-20: Improper Input Validation
CategoryTestPurpose: CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Inje
CategoryTestPurpose: CWE-697: Insufficient Comparison
CategoryTestPurpose: CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CategoryTestPurpose: CWE-680: Integer Overflow to Buffer Overflow
CategoryTestPurpose: CWE-131: Incorrect Calculation of Buffer Size

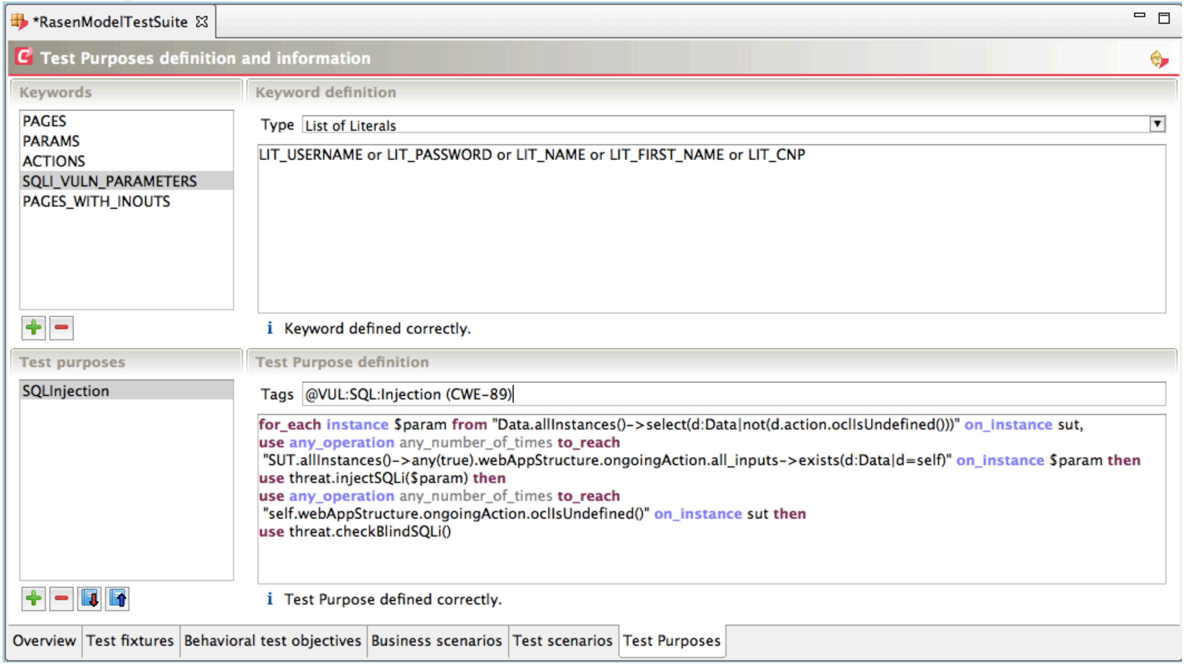
Test Purpose derivation



<table>
<tr><td><b>Pattern Name</b></td><td colspan="2">SQL Injection</td></tr>
<tr><td><b>CWE-ID(s)</b></td><td colspan="2">CWE-89</td></tr>
<tr><td><b>Weakness Description</b></td><td colspan="2">The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.<b>Error! Reference source not found.</b></td></tr>
<tr><td rowspan="5"><b>Solution</b></td><td colspan="2">Based on attack pattern CAPEC-66 <b>Error! Reference source not found.</b>
1. Use the application, client or web browser to inject SQL constructs input through text fields or through HTTP GET parameters.
2. Use a possibly modified client application or web application debugging tool such to submit SQL constructs for submitted values or to modify HTTP POST parameters, hidden fields, non-freeform fields, etc.
3. Check for error messages, delays, disclosed values in the client application and new/modified/deleted values in the database.</td></tr>
<tr><td><b>Test Design Technique</b></td><td>Data fuzzing
Pattern-based testing</td></tr>
<tr><td><b>Test Strategies</b></td><td>SQL Injection</td></tr>
<tr><td><b>Effort</b></td><td>Low to medium: can be highly automated using fuzzing techniques or SQL injection dictionaries.</td></tr>
<tr><td><b>Effectiveness</b></td><td>Medium <b>Error! Reference source not found.</b> to high, depending on detection capabilities by access to the affected database and to error messages</td></tr>
<tr><td><b>Description of Test Coverage Items</b></td><td colspan="2">• Functionality that involves user input, e.g. dialogs, URLs of a web application, that might be used in a database query
• User input fields
• SQL injection payloads
• Names of tables and rows of the database schema
• Values of existing records</td></tr>
<tr><td><b>Discussion</b></td><td colspan="2">SQL injection is a task that could be rather trivial but also very complex. This depends on several factors. For instance, error messages resulting from incorrect SQL constructs caused by SQL injection are very helpful in deciding whether SQL injection is generally possible.
In order to detect whether table data can be modified, it is helpful to have knowledge of the database management system (different systems have little differences in SQL syntax) and the database schema (modifying existing records may require knowledge in which tables they are stored).
If SQL injection is possible, the extent of SQL injection can be assessed by trying to modify existing data which requires knowledge of existing values in the database tables. This enables to determine whether existing database entries can be read, modified or deleted.</td></tr>
<tr><td><b>Test Data</b></td><td colspan="2">• SQL Injection Cheat Sheet<b>Error! Reference source not found.</b>
• Fuzzing library Fuzzino<b>Error! Reference source not found.</b></td></tr>
<tr><td><b>Testing Tools</b></td><td colspan="2">• Fuzzing framework Sulley<b>Error! Reference source not found.</b>
• Sqlmap<b>Error! Reference source not found.</b></td></tr>
<tr><td><b>Generalization of</b></td><td colspan="2"><b>Error! Reference source not found.</b></td></tr>
</table>

Automatic derivation from Test Pattern to Test Purpose:

- Linked to model by using keywords
- Testing directives inherited from Test Patterns



Keywords:
PAGES
PARAMS
ACTIONS
SQLI_VULN_PARAMETERS
PAGES_WITH_INOUTS

Type: List of Literals

LIT_USERNAME or LIT_PASSWORD or LIT_NAME or LIT_FIRST_NAME or LIT_CNP

i Keyword defined correctly.

Test purposes:
SQLInjection

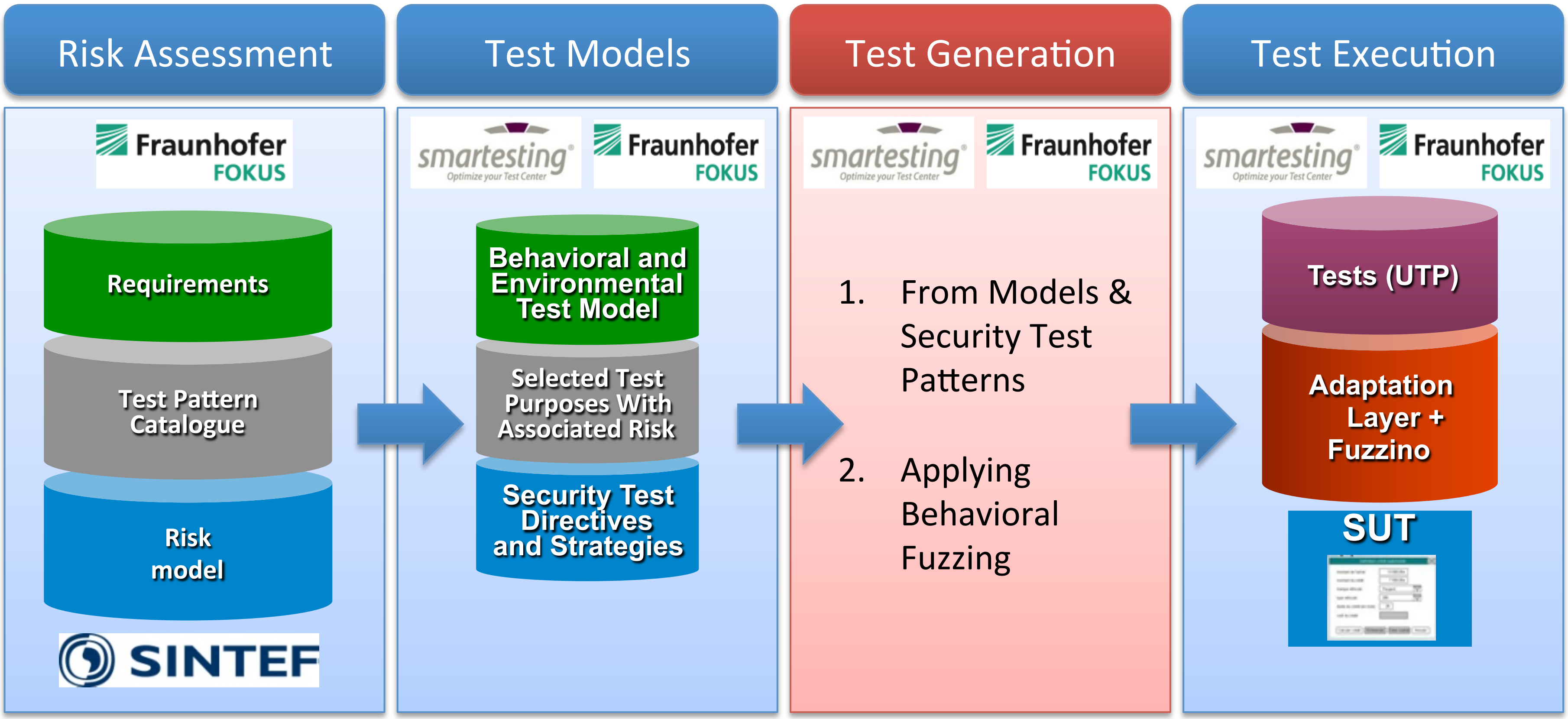
Tags: @VUL:SQL:Injection (CWE-89)

```
for_each instance $param from "Data.allInstances()->select(d:Data|not(d.action.oclIsUndefined()))" on_instance sut,
use any_operation any_number_of_times to_reach
 "SUT.allInstances()->any(true).webAppStructure.ongoingAction.all_inputs->exists(d:Data|d=self)" on_instance $param then
use threat.injectSQLi($param) then
use any_operation any_number_of_times to_reach
 "self.webAppStructure.ongoingAction.oclIsUndefined()" on_instance sut then
use threat.checkBlindSQLi()
```

i Test Purpose defined correctly.

Generic

3. Security test generation



Risk Assessment

Requirements

Test Pattern Catalogue

Risk model

Test Models

Behavioral and Environmental Test Model

Selected Test Purposes With Associated Risk

Security Test Directives and Strategies

Test Generation

1. From Models & Security Test Patterns
2. Applying Behavioral Fuzzing

Test Execution

Tests (UTP)

Adaptation Layer + Fuzzino

SUT

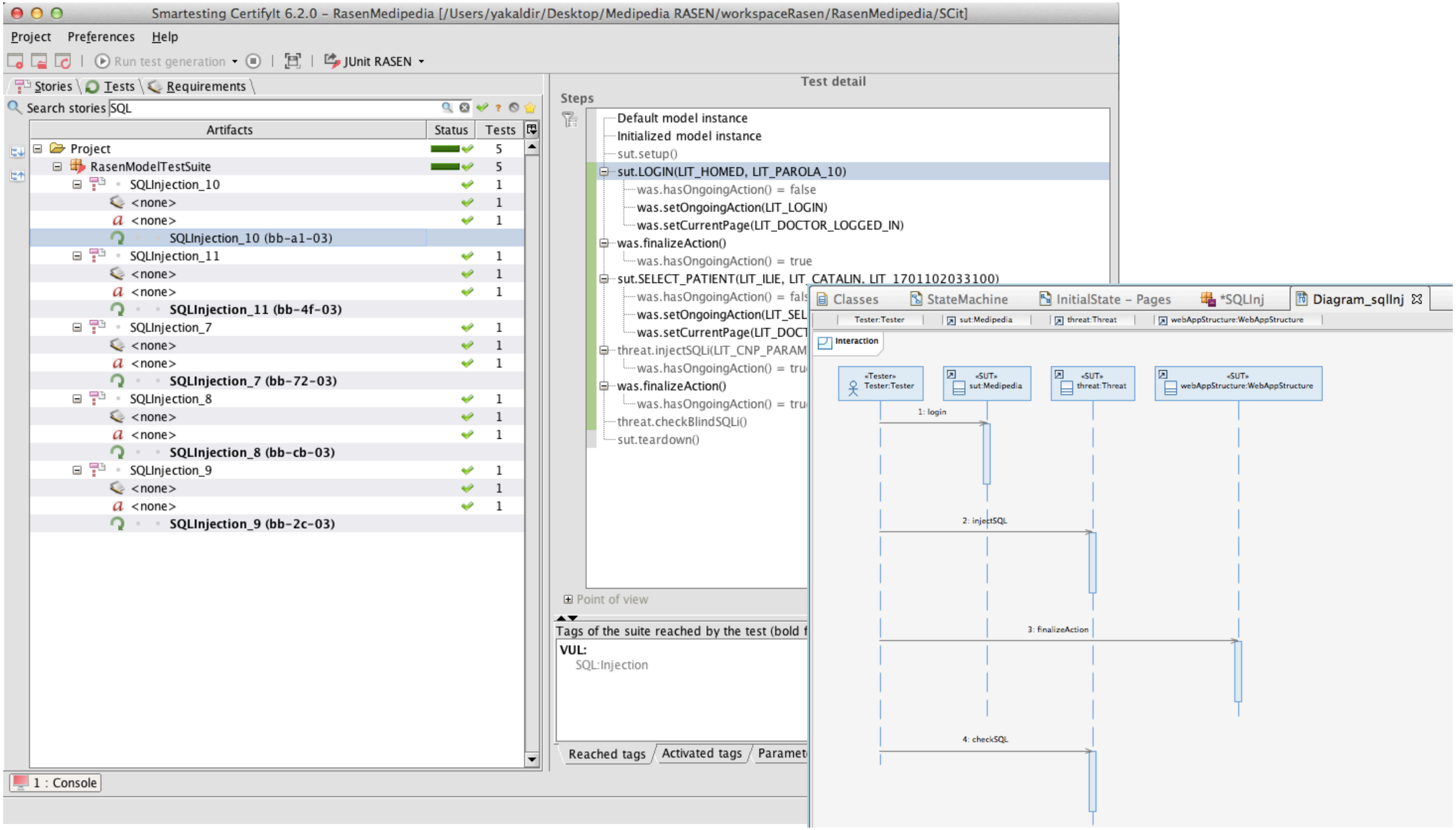
Test generation strategies

Test cases are automatically generated using Smartesting CertifyIt by composing behavioral models and test purposes:

- For one Test Purpose, several (or many) test cases by:
  - Applying usual Test Purpose coverage criteria
  - Applying behavioral fuzzing strategy given from Test Patterns
- Traceability management from security requirements to generated tests is build-in

Result: a suite of abstract security test cases

Test generation results using CertifyIt



Search stories SQL

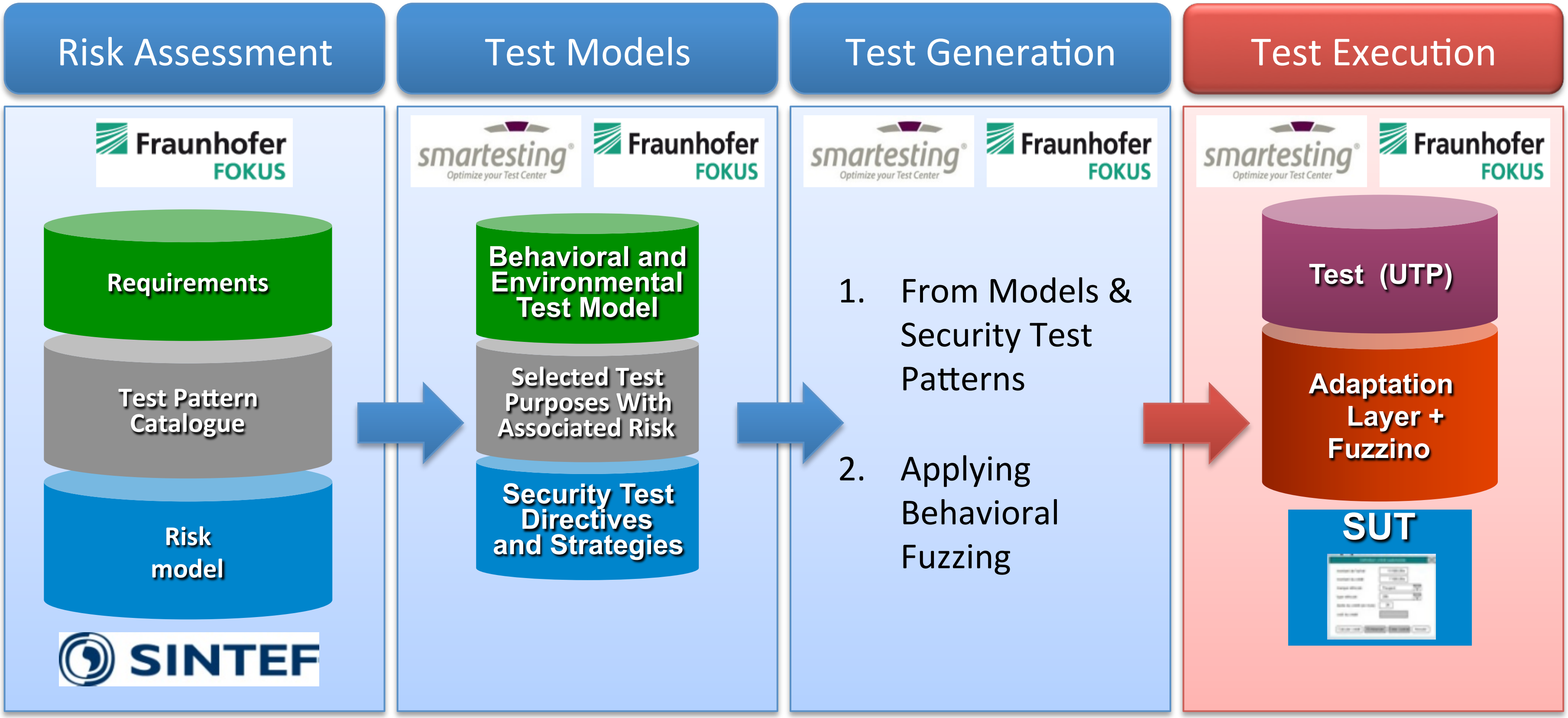
Artifacts	Status	Tests
Project	✓	5
RasenModelTestSuite	✓	5
SQLInjection_10	✓	1
<none>	✓	1
<none>	✓	1
SQLInjection_10 (bb-a1-03)		
SQLInjection_11	✓	1
<none>	✓	1
<none>	✓	1
SQLInjection_11 (bb-4f-03)		
SQLInjection_7	✓	1
<none>	✓	1
<none>	✓	1
SQLInjection_7 (bb-72-03)		
SQLInjection_8	✓	1
<none>	✓	1
<none>	✓	1
SQLInjection_8 (bb-cb-03)		
SQLInjection_9	✓	1
<none>	✓	1
<none>	✓	1
SQLInjection_9 (bb-2c-03)		

```
Default model instance
Initialized model instance
sut.setup()
sut.LOGIN(LIT_HOMED, LIT_PAROLA_10)
    was.hasOngoingAction() = false
    was.setOngoingAction(LIT_LOGIN)
    was.setCurrentPage(LIT_DOCTOR_LOGGED_IN)
was.finalizeAction()
    was.hasOngoingAction() = true
sut.SELECT_PATIENT(LIT_ILIE, LIT_CATALIN, LIT_1701102033100)
    was.hasOngoingAction() = fals
    was.setOngoingAction(LIT_SEL
    was.setCurrentPage(LIT_DOCT
threat.injectSQLi(LIT_CNP_PARAM
    was.hasOngoingAction() = tru
was.finalizeAction()
    was.hasOngoingAction() = tru
threat.checkBlindSQLi()
sut.teardown()
```

Tags of the suite reached by the test (bold f

VUL:
SQL:Injection

4. Test concretization for execution


Risk Assessment
Fraunhofer FOKUS
Requirements
Test Pattern Catalogue
Risk model
SINTEF
Test Models
smartesting Optimize your Test Center
Fraunhofer FOKUS
Behavioral and Environmental Test Model
Selected Test Purposes With Associated Risk
Security Test Directives and Strategies
Test Generation
smartesting Optimize your Test Center
Fraunhofer FOKUS
1. From Models & Security Test Patterns
2. Applying Behavioral Fuzzing
Test Execution
smartesting Optimize your Test Center
Fraunhofer FOKUS
Test (UTP)
Adaptation Layer + Fuzzino
SUT

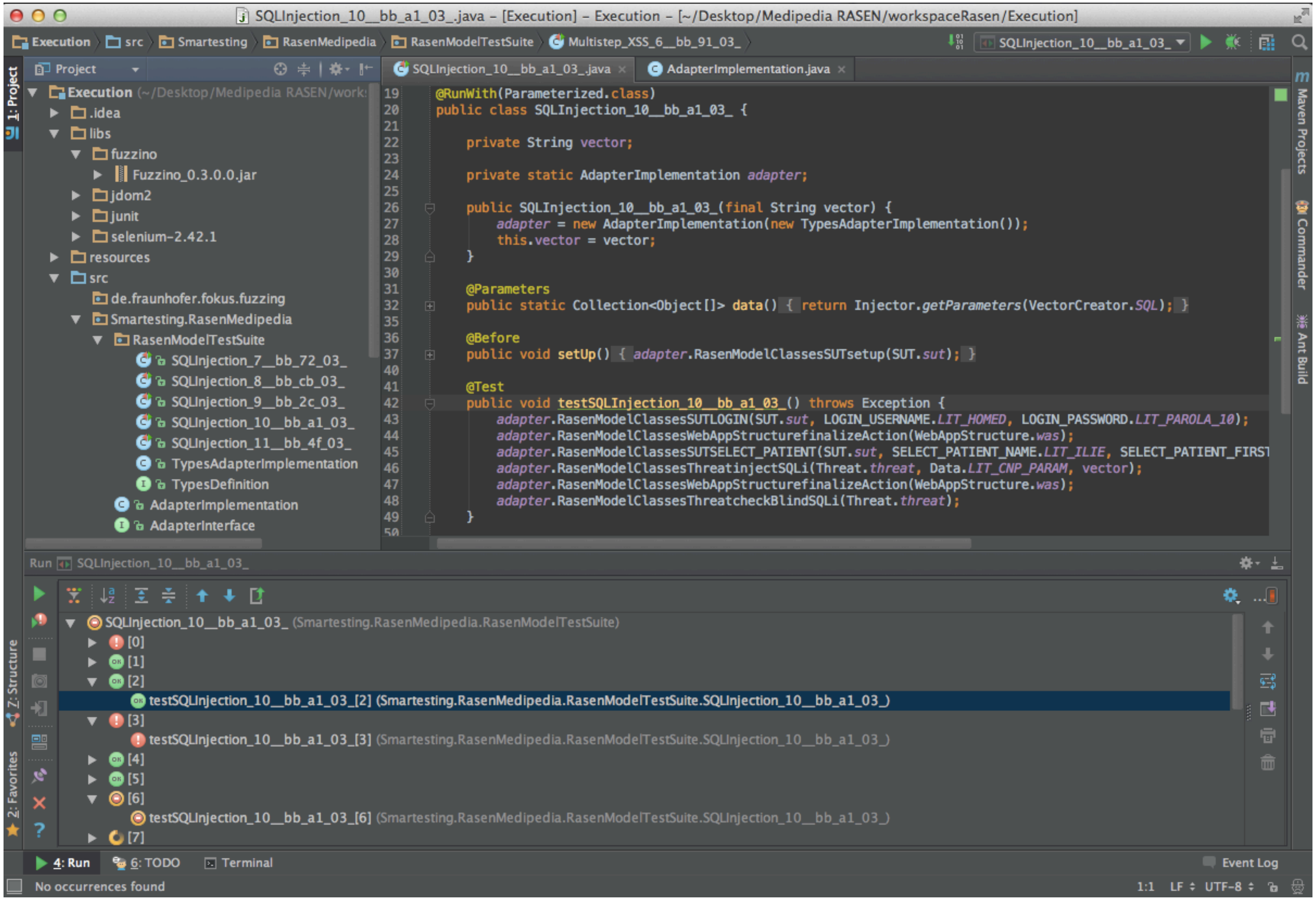
Generation of executable test scripts

JUnit test scripts are automatically generated by CertifyIt using an adaptation layer concretizing abstract data into concrete values:

- For one abstract test case, several (or many) executable test cases by:
    - Using a set of selected test data given from Test Patterns
    - Applying data fuzzing strategy given from Test Patterns
- Traceability management from security requirements to executable tests is build-in

Result: a set of executable security test scripts

Tests Execution in JUnit environment


SQLInjection_10__bb_a1_03_.java - [Execution] - Execution - [~/Desktop/Medipedia RASEN/workspaceRasen/Execution]
Execution › src › Smartesting › RasenMedipedia › RasenModelTestSuite › Multistep_XSS_6__bb_91_03_
SQLInjection_10__bb_a1_03_
Project
SQLInjection_10__bb_a1_03_.java
AdapterImplementation.java
1: Project
Maven Projects
Commander
Ant Build
7: Structure
2: Favorites
4: Run
6: TODO
Terminal
Event Log
1:1 LF UTF-8

```
Execution (~/Desktop/Medipedia RASEN/work
  .idea
  libs
    fuzzino
      Fuzzino_0.3.0.0.jar
    jdom2
    junit
    selenium-2.42.1
  resources
  src
    de.fraunhofer.fokus.fuzzing
    Smartesting.RasenMedipedia
      RasenModelTestSuite
        SQLInjection_7__bb_72_03_
        SQLInjection_8__bb_cb_03_
        SQLInjection_9__bb_2c_03_
        SQLInjection_10__bb_a1_03_
        SQLInjection_11__bb_4f_03_
        TypesAdapterImplementation
        TypesDefinition
      AdapterImplementation
      AdapterInterface
```

```java
@RunWith(Parameterized.class)
public class SQLInjection_10__bb_a1_03_ {

    private String vector;

    private static AdapterImplementation adapter;

    public SQLInjection_10__bb_a1_03_(final String vector) {
        adapter = new AdapterImplementation(new TypesAdapterImplementation());
        this.vector = vector;
    }

    @Parameters
    public static Collection<Object[]> data() { return Injector.getParameters(VectorCreator.SQL); }

    @Before
    public void setUp() { adapter.RasenModelClassesSUTsetup(SUT.sut); }

    @Test
    public void testSQLInjection_10__bb_a1_03_() throws Exception {
        adapter.RasenModelClassesSUTLOGIN(SUT.sut, LOGIN_USERNAME.LIT_HOMED, LOGIN_PASSWORD.LIT_PAROLA_10);
        adapter.RasenModelClassesWebAppStructurefinalizeAction(WebAppStructure.was);
        adapter.RasenModelClassesSUTSELECT_PATIENT(SUT.sut, SELECT_PATIENT_NAME.LIT_ILIE, SELECT_PATIENT_FIRST
        adapter.RasenModelClassesThreatinjectSQLi(Threat.threat, Data.LIT_CNP_PARAM, vector);
        adapter.RasenModelClassesWebAppStructurefinalizeAction(WebAppStructure.was);
        adapter.RasenModelClassesThreatcheckBlindSQLi(Threat.threat);
    }
```

```
Run  SQLInjection_10__bb_a1_03_
SQLInjection_10__bb_a1_03_ (Smartesting.RasenMedipedia.RasenModelTestSuite)
  [0]
  [1]
  [2]
    testSQLInjection_10__bb_a1_03_[2] (Smartesting.RasenMedipedia.RasenModelTestSuite.SQLInjection_10__bb_a1_03_)
  [3]
    testSQLInjection_10__bb_a1_03_[3] (Smartesting.RasenMedipedia.RasenModelTestSuite.SQLInjection_10__bb_a1_03_)
  [4]
  [5]
  [6]
    testSQLInjection_10__bb_a1_03_[6] (Smartesting.RasenMedipedia.RasenModelTestSuite.SQLInjection_10__bb_a1_03_)
  [7]
No occurrences found
```

Conclusion and future work

- Extended security test patterns for risk-based test case generation
- Formalization of security test patterns into test purpose language to drive the risk-based test generation
- Risk-based testing approach combining RASEN partners risk assessment and testing techniques:
  - Risk identification and prioritization using CORAS method
  - Import of risk assessment results from CORAS tool into CertifyIt
  - Test purpose generation method (CertfyIt)
  - Behavioral and data fuzzing strategies (Fuzzino)

- Definition of more accurate testing strategies regarding risk prioritization
- Extension of security test patterns and related test purposes
- Improvements of the tool integration (especially Test Purpose / fuzzing)
- Deeper use case evaluation, especially to validate the approach regarding large scale systems

Thank you for your attention !

Questions and Comments?

http://www.rasenproject.eu/

RASEN Compositional Risk Assessment and Security Testing of Networked Systems

The project can also be followed on Twitter & LinkedIn:

@RASENProject
#RASENProject
http://www.linkedin.com/groups?home=&gid=7429037