

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 1 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

	Initial Security Test Patterns Catalogue
	Version: 1.0 Date : 30.05.2012 Pages : 34
	Editor: Alain-G. Vouffo Feudjio
	Reviewers: TL,itrust, TU Graz
	To: DIAMONDS Consortium
The DIAMONDS Consortium consists of: Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Get IT, Giesecke & Devrient, Grenoble INP, itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, TU Graz, University Oulu, VTT	
Status: <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input type="checkbox"/> Proposal <input checked="" type="checkbox"/> Final / Released	Confidentiality: <input checked="" type="checkbox"/> Public Intended for public use <input type="checkbox"/> Restricted Intended for DIAMONDS consortium only <input type="checkbox"/> Confidential Intended for individual partner only

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 2 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

Deliverable ID: D3.WP4.1

Title:

Initial Security Test Patterns Catalogue

Summary / Contents:

This document provides an initial catalogue of security test patterns identified in the DIAMONDS project and used for deriving security tests applicable to the featured case studies.

Contributors to the document:

Roland Groz (Grenoble INP), Stéphane Maag (Institut Telecom Paris), Laurent Mounier (Grenoble INP), Sanjay Rawat (Grenoble INP), Jean-Luc Richier (Grenoble INP), Alain-G. Vouffo Feudjio (FOKUS)



	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 3 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


TABLE OF CONTENTS

1. Introduction.....	7
2. Security Test Patterns and the DIAMONDS Methodology	7
2.1 Definition of Security Test Patterns	7
2.2 Security Test Patterns in the Methodology.....	7
2.2.1 Information Technology Security Evaluation Criteria (ITSEC)	8
2.2.2 Security Test Pattern.....	8
2.2.3 A Workable Model for Security Test Patterns	10
2.2.4 An Example of Proposed Methodology Mapped to SmartTesting Testing Process.....	11
3. A SECURITY Test Pattern Template	11
4. Initial Test Patterns Catalogue	13
4.1 Generic Security test Patterns.....	13
4.1.1 Test Pattern: Verify audited event's presence	13
4.1.2 Test Pattern: Verify audited event's content	14
4.1.3 Test Pattern: Verify default-authentication credentials to be disabled on production system	15
4.1.4 Test Pattern: Verify presence/efficiency of prevention mechanism against brute force authentication attempts (active, passive)	17
4.1.5 Test Pattern: Verify presence/efficiency of encryption of communication channel between authenticating parties (active, passive)	19
4.1.6 Test Pattern: Usage of Unusual Behavior Sequences.....	20
4.1.7 Test Pattern: Detection of Vulnerability to Injection Attacks.....	22
4.1.8 Test Pattern: Detection of Vulnerability to Data Structure Attacks.....	23
4.2 Security Test Patterns based on MITRE	24
4.2.1 Attacking a Session Management.....	24
4.2.2 Attack of the authentication mechanism	26
4.2.3 Testing the safe storage of authentication credentials.....	27
4.2.4 Open Redirect	28
4.2.5 Uploading a malicious file	29
4.2.6 Searching for documented passwords.....	30
4.2.7 Impersonating an external server.....	31
4.2.8 Accessing resources without required credentials	32
4.2.9 Ensuring confidentiality of sensitive information	33
5. Summary of Test Patterns Catalogue.....	33
References	34

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 4 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

FIGURES

Figure 1 - Security Test Pattern.....	10
---------------------------------------	----


	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 5 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

HISTORY

Vers.	Date	Author	Description
0.1	2012/04/11	A. Vouffo	Document creation
1.0	2012/05/30	A. Vouffo	Final version

APPLICABLE DOCUMENT LIST


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

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 6 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

EXECUTIVE SUMMARY

Patterns are a well-known and established mean for capturing knowledge in a structured manner to facilitate the instantiation of new solutions to recurrent problems. As a domain in which a huge amount of knowledge is available, but disseminated through various sources and thus difficult to reuse, security testing could significantly benefit from test patterns and their integration in the security testing process. The DIAMONDS project's work package 4 has therefore dedicated some efforts in designing a methodology for enabling such an integration of security test patterns.

This deliverable provides a first set of security test patterns identified so far in the DIAMONDS project through the various case studies being conducted and covering the different domains addressed in the project.

	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 7 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

1. INTRODUCTION

This deliverables presents first results of task 4.1 of the DIAMONDS project dedicated to applying patterns to model-based security testing. The document is organized as follows: The next section introduces the concept of security test patterns and how it integrates into the overall DIAMONDS methodology. Section 3 goes on and introduces a template for security test pattern that will be used for capturing patterns into the catalogue presented in Section 4. Finally, Section 5 concludes and summarizes the document.

2. SECURITY TEST PATTERNS AND THE DIAMONDS METHODOLOGY

2.1 DEFINITION OF SECURITY TEST PATTERNS

Security testing is a domain in which a lot of knowledge has been collected from a significant amount of research work done in recent years. Numerous guidelines and best practices have been identified and are used at several instances. Patterns are an established approach for facilitating the reuse of known solutions to recurring problems in various domains. Originally, patterns were introduced by Ch. Alexander[1], a construction architect, to capture the essence of sound and well-established design in the architecture of buildings so that they could serve as guidance for other architects in designing new buildings. That approach was successfully applied to software design and engineering in general by the so-called Gang-of-Four (GoF)[2] and other authors later on. More recently, there have been some efforts to applying the same approach to testing and test automation, given the similarities between those disciplines and generic software development [2][3].

Software security is another domain in which patterns have been gaining more popularity recently. In fact numerous works exist on security patterns, i.e. patterns that aim at improving security capabilities of software system (procedures, design, architecture) [7].

However, security test patterns are a relatively new research field. For example, SecurityTestPatterns.org provides a catalogue of security test patterns [15].

This catalogue lists some 10 security test patterns, based on a well-defined template. The catalogue also specifies how each of those test patterns is associated to weaknesses from MITRE's Common Weaknesses Enumeration (CWE) [13] and provides a black-box test procedure including expected results templates for each of them.

The SecurityTestPatterns.org group define security test patterns as follows:

„A software security test pattern is a recurring security problem, and the description of the a test case that reveals that security problem, that is described such that the test case can be instantiated a million times over, without ever doing it the same way twice.“


The above definition slightly differs from our definition in that, rather than emphasizing on the testing aspects of security, it focusses on the security problem itself.

Another difference resides in the fact that the template defined and used by the SecurityTestPatterns.org group does not align to problem-solution-consequences schema recommended in pattern literature.

The test patterns are mostly destructive and do not address testing based on security objectives or SFRs
No classification of test patterns

We define a test pattern as the expression of the essence of a well-understood solution to a recurring software testing problem. This definition is basically a transcription of the generic pattern definition provided by Christopher Alexander [1] into the software testing discipline.

2.2 SECURITY TEST PATTERNS IN THE METHODOLOGY

	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 8 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

As aforementioned, there have been numerous definitions of patterns, suitable for different contexts. Perhaps the notion of "design patterns" could be close to defining security test patterns. A design pattern is a description of a recurring problem and a well-defined description of the core solution to the problem that is described such that the pattern can be used many times but never in exactly the same way [1]. According to SecurityTestPatterns.org group [15], a software security test pattern is a template of a test case that exposes vulnerabilities, typically by emulating what an attacker would do to exploit those vulnerabilities.

The above definition [15], while being simple and short, may only suffice well to the notion of security test patterns at lower level, while that of [1] remains at much higher level. Our notion, on the other hand, is visible in the following question: "what is the pattern to test the security properties of a System under Test (SUT)?" Here the security properties can broadly be defined as CIA i.e. confidentiality, integrity and availability. We, therefore, embrace both of the above definitions to define patterns that are, on the one hand, generic enough to adapt to various testing strategies and, on the other hand, include patterns for concrete security test cases.

It should be noted that CIA can be extended to include other aspects of security properties. Our hypothesis is that in order to test the security properties of SUT (i.e. CIA) that are defined as much higher level of hierarchy, we need to define the pattern in terms of lower level functionalities/mechanism that are related to those CIA properties. To support our hypothesis, we present an analogy that describes this relation between higher and lower hierarchies.

2.2.1 Information Technology Security Evaluation Criteria (ITSEC)

The **Information Technology Security Evaluation Criteria (ITSEC)** is a structured set of criteria for evaluating computer security within products and systems [9]. This criterion provides a rather descriptive and step wise step procedure to evaluate the security of the system/product. The target of evaluation is the product/system that is being evaluated for its security features.

A Target of Evaluation (TOE) which provides security (some combination of confidentiality, integrity and availability) must contain appropriate security features [2].

In these criteria, security features are viewed at three levels. The most abstract view is of security objectives: the contribution to security which a TOE is intended to achieve. To achieve these objectives, the TOE must contain certain security enforcing functions. These security enforcing functions, in turn, must be implemented by specific security mechanisms. These three levels can be summarised as follows:

- a) Security Objectives - Why the functionality is wanted.
- b) Security Enforcing Functions - What functionality is actually provided.
- c) Security Mechanisms - How the functionality is provided.


In the above, the *Security Objectives* can be described as CIA (confidentiality, integrity and confidentiality) model. *Security Enforcing Functions* are well-known techniques for addressing objectives, such as Identification and Authentication, Access Control, Accountability, Audit, Object Reuse, Accuracy, Reliability of Service, and Data Exchange. Finally, the *Security Mechanisms* are the actual methods used to implement those enforcing functions.

The assessment consists of verifying that for each security target of the TOE, there exists at least one enforcing function and a corresponding mechanism. In case of any foreseen threat to the TOE, there are enforcing functions and mechanisms to counter that threat.

In the view of the above description, we can notice that in order to evaluate the security of the system, we need to define the security properties at some higher level and then we check the enforcing functions/mechanisms that are related to those higher level properties i.e. the functions/mechanisms that *enable* those properties.

2.2.2 Security Test Pattern

We adapt the above phenomenon to the arena of security test patterns. It should be noted that ITSEC security evaluation criteria, in a way, is positive testing for its security properties i.e. what security features

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 9 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

are present in the system. We, on the other hand, are more interested in security test patterns that compromise the security of the SUT. Therefore, from this perspective, whatever is *positive* for us is *negative* for ITSEC criteria. Following (adopting) the above terminology, we can define security test pattern in terms of three subtasks:

2.2.2.1 Security Objectives

We define the security (test) objective as “*what security properties are needed to be tested?*” Here, our aim is to establish if we can breach/break the above stated security properties. The properties of interest (other than CIA) can be an input to the framework. These properties can be obtained by other approaches, for example, risk based analysis (see, e.g., FOKUS contribution in D1&2 WP2) or manual inspection of the SUT etc. A publicly available example of risk based model is *Common Weakness Risk Analysis Framework* (CWRAF) [12].

2.2.2.2 Security Weakening Functions

The next step is to define what *software weaknesses (vulnerabilities) compromise the security properties (objectives)* (cf. the enforcing function of ITSEC). Basically, the weakening functions describe the weakness/problems that affects some (or all) of CIA security properties. The main objective of this step is to identify a functional workflow of the applications from security standpoint i.e. identify the weak points in the SUT by defining a functional model, for example, as is being followed by SmartTesting (see D2.WP3, section 2.2). By performing this step, we can figure out points in the model of the SUT that may exhibit weaknesses. We may term them as **weak-points**. The idea of defining weak-points comes from the observations that security properties are associated with the SUT at certain points in its execution. For example, authentication can be tested at point when the SUT is performing some authentication related task (e.g. login page). Another aspect of this step is to understand the SUT's behaviour w.r.t. its environment i.e. how does it interact with its environment in terms of data consumption and the associated functionality. As will be explained in the next paragraph, with this knowledge, we again get weak-points that are entry points for many attacks. These weak-points are akin to the notion of MITRE's common weakness enumeration (CWE) [13] (see section 2.2.3 for more details). At this point, we distinguish two levels of security properties:

- 1) High level properties i.e. functional security properties; and
- 2) low level properties i.e. non-functional properties.

The former covers the functional view of the security properties, for example use of appropriate cryptographic primitives, whereas the later is more concerned with the low level vulnerabilities, for example buffer overflow leading to arbitrary code execution. This terminology is similar to what is defined in OWASP testing guide [14]. An exhaustive strategy should address both of these levels adequately. Therefore, coming back to our terminology, these enforcing functions are *positive* weaknesses that we should be interested in.

2.2.2.3 Enabling Mechanisms

As with the ITSEC, we now proceed to find out the patterns that *enable* them. These patterns can be called *mechanisms* to implement weakening functions. From security testing point of view, these mechanisms are nothing but the concrete tests that *may* exhibit the attacker's like behaviour. The kind of test that we want to generate depends on weak-points (i.e. weaknesses exhibited at that point). Their inputs may come from a variety of sources. We can make use of, for example, fuzzing framework to generate inputs or open repositories like MITRE's **Common Vulnerabilities and Exposures (CVE)** [11] and **Common Attack Pattern Enumeration and Classification (CAPEC)** [10]. These mechanisms can be considered either as patterns corresponding to some security properties breach or as instances of weakening functions corresponding to some security properties breach. Now, if we want to test a particular security property, for example, confidentiality, we will look for weakening functions (i.e. find the weak-points) that compromise that property (i.e. the confidentiality) and select the corresponding mechanisms to test with a concrete test case. Figure 1 illustrates the whole process.

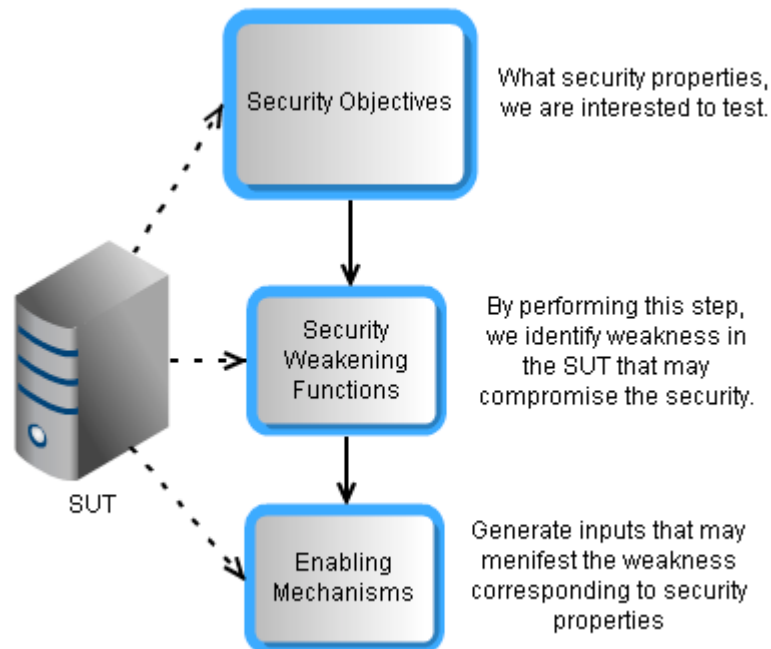


Figure 1 - Security Test Pattern

In this way, we observe that the security test objectives that are described at higher level, can be realized by considering the corresponding enabling mechanisms at lower level. Therefore, each lower level mechanism is related to one or more higher level security test objectives.

2.2.3 A Workable Model for Security Test Patterns

In this section, we provide a more concrete example of the aforementioned security test pattern in terms of CWE, CAPEC and CVE. This example is more suitable for low level security properties. The same can, however, be adapted to any other scenario by replacing CWE, CAPEC or CVE by appropriate terms as is used by the testers.

Let us assume that the security test objectives is the set {confidentiality, integrity, availability}, i.e. our goal is to test if we can breach any of them (or the stated one from the set). The next step is to find corresponding enforcing functions. For that we make use of CWE list. Each CWE ID has a field called “**Common Consequences** (Scope)” which describes the affect of the weakness in terms of above said objectives. Therefore, we can select all the CWE IDs, where the **Common Consequences** (Scope) field intersect with the objectives. Also included in each CWE ID, are two other fields, called “**Observed Examples**” and “**Related Attack Patterns**”. The first of them is related to CVE IDs which indicates that an instance of the CWE ID is observed. The later one is the attack patterns pointing to some “**CAPEC-ID**” which means there is an attack patterns to exploit the corresponding CWE ID. Therefore, for each selected CWE IDs, we get all the CVE IDs and CAPEC IDs which are related to this CWE ID. These are the mechanisms that actually test the security objectives i.e. there are our test cases.


The whole process can be described as the following algorithm:

Input:

```

A set SO of Security Objective;
CWE database;
CVE database;

```

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 11 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

```

CAPEC database;
for sec in SO:
    for cwe_id in CWE:
        if cwe_id->common Consequence is sec:
            for capec_id in cwe_id->Related Attack Patterns:
                test_case= CAPEC.get(capec_id)
            for cve_id in cwe_id-.Observed Examples:
                test_case= CVE.get(cve-id)

```

2.2.4 An Example of Proposed Methodology Mapped to SmartTesting Testing Process

In this section, we provide a possible mapping of the proposed methodology for security test pattern to smartTesting security testing process. The objective is to demonstrate the suitability and applicability of the proposed security test patterns to the existing Diamonds testing process.

SmartTesting has provided details of its security testing methodology in the deliverable D3.WP2.T2_3 under the section “Smartesting Model-Based Security Testing from behavioural models and security-oriented Test Purposes”. We identify the following three main steps involve in the testing process.


1. Inputs Artifacts: This is mainly to recognize the *security properties* that are required to be tested.
2. Working Artifacts: This step involves defining security test *purpose and objectives*. Basically, this provides a correlation between security properties and the action or behaviour that may break those properties. SmartTesting framework involves security test engineer to provide these details.
3. Output Artifacts: Finally the test cases are generated to access the security features of the SUT.

In the view of above description, it is easy to observe that point 1 corresponds to first task “security Objectives” of the proposed methodology. Point 2 is related to “security Weakening Function” task as in this step, security engineer analyzes the SUT model from security standpoint and decides the further course of action by figuring out the weaknesses. This step becomes the basis for defining the description of test generation step. As a result, point 3 directly maps the third task “enabling mechanism”. It should be noted that at point 2, security engineer may decide to also consider low level security properties to be tested. *In this context, we may like to point out that vulnerability patterns as described in deliverable D3.WP2.T2_3 “Patterns for Buffer Overflow Vulnerability” can be used to further derive the test generation process.*

Thus, we may note that the proposed “security test pattern” is generic enough to encompass different security testing methodologies, being developed in Diamonds.


3. A SECURITY TEST PATTERN TEMPLATE

While (test) patterns may be helpful for enhancing the level of automation in the software engineering process, they are mainly aimed to be readable and understandable by human beings. The template used in this section takes this into account and defines the types of information expected to be provided for each security test pattern.

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 12 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

Pattern name	A meaningful name for the test pattern.
Context	<p>To which specific context does it apply? This includes the kind of test pattern (organisational vs. design, generic, architectural, behavioural or test data etc.) as well as the security approach(es) in which the pattern may be applied. The concept of security approach is used here as presented by Schumacher et al in their book on security patterns[7]. According to them, security approaches define groups of related ways to address potential security violations. They identify the following group of security approaches:</p> <ul style="list-style-type: none"> - Planning embodies the organization-wide standard operation procedures (documentation) for prevention, detection, and response. - Prevention consists in efforts aiming at actively impeding unwanted incidents, i.e. undesirable activities that would compromise security assets. - Detection aims at identifying or detecting unwanted incidents on the element to be protected. - Diligence refers to ongoing proactive measures for updating security plans to improve the overall security posture of an organization. - Response approaches are those addressing unwanted incidents or other security violations, after they have been detected <p>As rightly indicated in [7] security approaches are not usually applied alone, but in various combinations, given the natural dependencies between each of the aspects they cover. Therefore, the context for a security test pattern may involve a combination of several security approaches.</p>
Problem/Goal	What is the testing problem this pattern addresses and which are the forces that come into play for that problem? In certain cases a pattern may not solve a specific problem, but provide a mean for achieving a particular goal with regard to security testing. In those cases, the goal(s) to be achieved by the pattern should be provided instead.
Solution	A full description of the test pattern, potentially including examples of applications. Where applicable, dedicated notations such as the UML Testing Profile (UTP), TTCN-3 or similar will be used for illustration.
Known uses	Known applications of the test pattern in existing test solutions or existing concepts enabling the application of the test pattern in existing test specification or test modelling languages.
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.
Related patterns (optional)	Test design pattern related to this one or system design patterns in which faults addressed by this test pattern might occur. This section is optional and will be omitted, if no related pattern can be named.
References (optional)	Bibliographic references to the pattern or external associated elements. This section is also optional and will be omitted, if no reference can be provided.

Table 1: Security Test Pattern Template

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 13 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4. INITIAL TEST PATTERNS CATALOGUE

4.1 GENERIC SECURITY TEST PATTERNS

Test Pattern: Verify audited event's presence	13
Test Pattern: Verify audited event's content.....	14
Test Pattern: Verify default-authentication credentials to be disabled on production system	15
Test Pattern: Verify presence/efficiency of prevention mechanism against brute force authentication attempts (active, passive)	17
Test Pattern: Verify presence/efficiency of encryption of communication channel between authenticating parties (active, passive).....	19
Test Pattern: Usage of Unusual Behavior Sequences	20
Test Pattern: Detection of Vulnerability to Injection Attacks	22
Test Pattern: Detection of Vulnerability to Data Structure Attacks	23


4.1.1 Test Pattern: Verify audited event's presence

Pattern name	Verify audited event's presence
Context	Test Pattern Kind: Behavioral Testing Approach(es): Detection
Problem/Goal	This pattern addresses how to check that a system logs a particular type of security-relevant event for auditing purpose
Solution	Test procedure template 1. Activate the system's logging functionality 2. Clear all existing log entries 3. Record current system time t_s 4. Stimulate the system to generate the expected event type 5. Check that the system's log contains entries for the expected event / Taking into account only logs displaying timestamps t_l satisfying following condition: $t_l > t_s$
Known uses	Common Criteria SFRs[17]: FAU_GEN.1 , FAU_GEN.2
Discussion	This pattern assumes that the test framework provides means for tracing and evaluating the logs produced by the SUT. Evaluation may be performed online (i.e. quasi simultaneously, while the system is still running) or offline, i.e. after the system has completed its operation. An interesting issue to be considered is how to apply this pattern in situations whereby it may be impossible or too costly to clear the logs repository or to restart the running system.
Related patterns (optional)	<ul style="list-style-type: none"> Sandwich test architecture pattern[5] Proxy test architecture pattern[5] Verify audited event's content (Section 4.1.2)
References	FAU_GEN.1, FAU_GEN.2

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 14 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.1.2 Test Pattern: Verify audited event's content

Pattern name	Verify audited events' content
Context	Test Pattern Kind: Behavioral Testing Approach(es): Detection
Problem/Goal	This pattern addresses how to check that a system logs a particular type of security-relevant event for auditing purpose
Solution	<p>Test procedure template:</p> <ol style="list-style-type: none"> 1. Activate the system's logging functionality 2. Clear all existing log entries / Record current system time t_s 3. Stimulate the system to generate <i>the expected event type</i> 4. Check that the system's log contains entries for <i>the expected event</i> / Taking into account only logs displaying timestamps t_l with $t_l > t_s$ 5. Store log entries containing the expected event type 6. Open the log entries and verify that their content meets the specified requirements
Known uses	Common Criteria SFRs: FAU_GEN.1[17] , FAU_GEN.2[17]
Discussion	<p>This pattern assumes that the test framework provides means for tracing and evaluating the logs produced by the SUT. Evaluation may be performed online (i.e. quasi simultaneously, while the system is still running) or offline, i.e. after the system has completed its operation.</p> <p>An interesting issue to be considered is how to apply this pattern in situations whereby it may be impossible or too costly to clear the logs repository or to restart the running system.</p>
Related patterns (optional)	<ul style="list-style-type: none"> • Sandwich test architecture pattern[5] • Proxy test architecture pattern[5] • Extends test pattern <i>Verify audited event's presence</i> (Cf. Section 4.1.1) by addin verification of the audited event's content.
References	CWE 311[13]


	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 15 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

4.1.3 Test Pattern: Verify default-authentication credentials to be disabled on production system

Pattern name	Verify default-authentication credentials to be disabled on production system
Context	Test pattern kind: Behavior Testing Approach(es): Prevention
Problem/Goal	<p>Enabled default authentication mechanisms, sometimes resulting from hard-coded credentials in source code are listed among MITRE's 2011 top 25 most dangerous software errors [20] from the well-known CWE. For many software products, providing such a set of default authentication credentials is unavoidable, for example in a situation whereby some initial settings require an account on the system after it is installed. Although those default credentials are supposed to be modified before the system is actually deployed and made available to the outside world, several cases have been reported in which this was omitted, thus allowing attackers to bypass the authentication procedure and obtaining access to potentially sensitive data. This is particularly relevant for systems based on open-source software, given that the parameters for those default credentials are known to a large group of potential attackers.</p> <p>Therefore, providing testcases for detecting this kind of errors is very important for any software-based system with some authenticated interface to the outside world.</p>
Solution	<p>Test procedure template: Depending on whether a black-box or a white-box testing approach is applicable, different test procedures may be appropriate.</p> <p>Black-box testing procedure template</p> <ol style="list-style-type: none"> 1. Create (or reuse) a dictionary of default credentials usually available in open source software (e.g. login: admin, password: password; login: root; password: pass; etc.) 2. Try to authenticate using each time a new combination of credentials from the dictionary of step 1 3. If any of the authentication attempts is successful set FAIL verdict. Otherwise set PASS. <p>White-box testing procedure template</p> <ol style="list-style-type: none"> 1. Create (or reuse) a dictionary of default credentials usually available in open source software (e.g. login: admin, password: password; login: root; password: pass; etc.) 2. Search the source code for any character string containing an element from the dictionary of step 1. Also include configuration files in the search.


	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 16 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

	3. If matching character strings are found, check that the source code implements a mechanism for enforcing the modification of authentication credentials.
Known uses	<p>Among the DIAMONDS project case studies, the automotive case study has identified some elements of vulnerability derived from the weakness addressed by this test pattern:</p> <p>Several Bluetooth devices use “0000” as default PIN to access control. Therefore a test case verifying that the default PIN code has been replaced by a more user-specific one makes perfect sense in that context.</p>
Discussion	If a black-box testing approach is chosen to apply this pattern, then it should be ensured that if present, a mechanism to block repetitive authentication attempts is deactivated, to avoid the SUT interpreting step 2 of the test procedure as a brute force hacking attempt, potentially leading to a cascade of other unwanted incidents unrelated with the actual test case.
Related patterns (optional)	<ul style="list-style-type: none"> Mutually exclusive relation to pattern <i>Verify presence/efficiency of prevention mechanism against brute force authentication attempts</i> (Section 4.1.4)
References	CWE 798 [13], OWASP-AT-003 [14]


	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 17 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

4.1.4 Test Pattern: Verify presence/efficiency of prevention mechanism against brute force authentication attempts (active, passive)

Pattern name	Verify presence/efficiency of prevention mechanism against brute force authentication attempts
Context	Test pattern kind: Behavior Testing Approach(es): Prevention, Detection
Problem/Goal	Password brute-forcing is a well-known attack pattern on computing systems providing a password-based authentication scheme (CAPEC 49 [10]).
Solution	<p>Test procedure template:</p> <p>The mechanism for preventing may be passive, active or a combination of both.</p> <p>An example of passive mechanisms consist in adding elements on the authentication interface that cannot be interpreted automatically by a machine, but require human intervention. This is widely used in authentication forms on web-based interfaces in the form of so-called captchas, i.e. graphical images created dynamically, but designed in a way that makes them difficult to be read automatically by a computer program. The authenticating client is required to complete his/her credentials with the information encoded in the picture to ensure that a human being is well submitting the information.</p> <p>On the other hand, active mechanisms will initiate a series of steps to impede that the number of failed authentication attempts from the same source does not exceed a predefined threshold, beyond which appropriate steps are undertaken as counter-measures.</p> <p>The following test procedure template applies for an active prevention mechanism against password brute-forcing:</p> <p>Assuming that the maximal number of failed authentication attempts that triggers the defense mechanism is F_{max}, and that T_{max} is the maximal delay beyond which the defense mechanism is expected to come into play, proceed as follows</p> <ol style="list-style-type: none"> 1. Use invalid credentials to authenticate on the system for F_{max} number of times or repetitively for a duration of T_{max} 2. Check that the SUT indicates that the used credentials are invalid and provides the user alternatives for the case he/she lost his/her credential details. 3. Optional: Check that failed authentication attempts are logged by the SUT and that the log entries contain as much information on the


	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 18 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

	<p>authentication source as possibly available.</p> <ol style="list-style-type: none"> 4. Use invalid credentials once more to authenticate on the system 5. Check that the system reacts in a way that impedes a new authentication attempt unless certain steps are undertaken by the authenticating party (i.e. the test client). Possible reactions include: <ul style="list-style-type: none"> - (Temporarily) Blocking future authentication attempts from the same client. This assumes the authentication provider is able to clearly identify the source for the authentication request (e.g. using a combination of IP-Address, Host name, Operating System, MAC-Address, MSISDN, etc.) - Introducing additional hurdles to make successive authentication attempts from the same source more difficult, both technically and from a time and resource perspective.
Known uses	<p>This security test pattern is widely used in all domains in which password-based authentication is applied (e.g. web-based applications and services, banking)</p> <p>Common Criteria SFRs: FIA_AFL.1 (Authentication Failures)[17]</p>
Discussion	
Related patterns (optional)	<ul style="list-style-type: none"> • This pattern is applicable in cases whereby the <i>Authenticator</i> security pattern[7] is used to ensure that entities accessing of a system are known as legitimate users thereof. • If the system logs all security-relevant incidents that occur at its external boundaries, as highly recommended by good practices in information systems security, then this pattern can be combined with the <i>Verify audited event's presence</i> pattern and the <i>Verify audited event's content</i> described in Section 4.1.1 and Section 4.1.2 respectively • Mutually exclusive relation with described in Section 4.1.3
References	CWE307 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 19 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.1.5 Test Pattern: Verify presence/efficiency of encryption of communication channel between authenticating parties (active, passive)

Pattern name	Verify presence/efficiency of encryption of communication channel between authenticating parties
Context	Test pattern kind: Behavior Testing Approach(es): Prevention
Problem/Goal	Man-in-the-middle attacks are known to be among the most severe attacks an information system might face with regard to its security [21]. One of the mitigation approaches consists in using encryption mechanisms (e.g. SSL) to protect the data exchange between authenticating parties from eavesdropping attempts with some of the numerous software tools freely available on the market and as open source.
Solution	Test procedure template: The steps to undertake for the test procedure are as follows: <ol style="list-style-type: none"> 1. Trigger the authentication client to start the authentication process using a well-known set of credentials 2. Check that the monitoring test component has captured the packets exchanged between both authenticating parties. 3. Check that the captured packets do not contain any information as plain-text that could easily be read and understood by an attacker without a significant computation effort.
Known uses	FTP_ITC.1 (Trusted channel)[17]
Discussion	This test procedure is only applicable with a black-box testing approach and requires a testing architecture whereby an entity is positioned between both authenticating parties, with the ability to capture data traffic in both directions between them. This kind of architecture is based on the <i>monitoring test component</i> architectural pattern described in a previous FOKUS work on test patterns [4].
Related patterns (optional)	<ul style="list-style-type: none"> • Monitoring test component architectural pattern[5] • CAPEC 94[21]
References	


	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 20 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

4.1.6 Test Pattern: Usage of Unusual Behavior Sequences

Pattern name	Usage of Unusual Behavior Sequences
Context	Test pattern kind: Behavior Testing Approach(es): Prevention
Problem/Goal	<p>Security of information systems is ensured in many cases by a strict and clear definition of what constitutes valid behavior 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[13]), some attacks (e.g. <i>Authentication bypass</i>, CAPEC 115[10]) may be possible by subjecting the system to a behavior 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.</p>
Solution	<p>Test procedure template:</p> <ol style="list-style-type: none"> 1. Use a specification of the system to clearly identify the normal behavior sequence it expects in interacting with an external party. If possible, model this behavior sequence using a notation such as UML, which provides different means for expressing sequenced behavior, e.g. sequence diagrams or activity diagrams. 2. Run the normal behavior 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 behavior sequences and for each of those <ul style="list-style-type: none"> - Check that the system does not show exceptional behavior (no live-/deadlock, no crashing, etc.) - Check that no invalid behavior 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)


	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 21 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

Known uses	Model-based Behavior fuzzing of sequence diagrams is an application of this pattern
Discussion	
Related patterns (optional)	
References	CWE 288 [13], CAPEC 115 [10])

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 22 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

4.1.7 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	<p>Injection attacks (CAPEC 152[10]) 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 behavior of a target through crafted input data submitted using an interface functioning to process data input [10]. 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.</p> <p>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.</p>
Solution	<p>Test procedure template:</p> <ol style="list-style-type: none"> 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...) 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: <ul style="list-style-type: none"> - 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 <ul style="list-style-type: none"> - Check that none of the observable events associated to an interpretation of the injected code is triggered
Known uses	

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 23 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

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 [10]
References	

4.1.8 Test Pattern: Detection of Vulnerability to Data Structure Attacks

Pattern name	Detection of vulnerability of data structure attacks
Context	Test pattern kind: Data Testing Approach(es): Prevention
Problem/Goal	Data structure attacks (CAPEC 255 [10]) consist in an attacker manipulating and exploiting characteristics of system data structures to violate the intended usage and protections of these structures and trigger the system to reach some instable state or expose further vulnerabilities that could be exploited to cause more harm. Detecting vulnerability to data structure attacks is among the key goals of security testing. The pattern provides a solution to that problem.
Solution	Test procedure template: <ol style="list-style-type: none"> 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 including invalid values, i.e. values not meeting the requirements associated to their type and thus potentially unexpected by the SUT 3. Use each of the input elements created at step 2 as input on the appropriate SUT interface, and for each of those <ul style="list-style-type: none"> - Check that the SUT does not enter an unstable state at any time during the test case (no live-/deadlock, no crash, no exception etc.)
Known uses	Data Fuzzing
Discussion	

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 24 of 34
		Version: 1.0
		Date : 30.05.2012
		Status : Final
		Confid : Public

Related patterns (optional)	• CAPEC 255
References	

4.2 SECURITY TEST PATTERNS BASED ON MITRE


As above mentioned, an interesting initiative has been carried out by the research group SecurityTestPatterns.org. This group (composed as a research community) provides its own definition of security test patterns and, from the analysis of vulnerability, attacks and weaknesses enumerated by MITRE, they develop a grounded theories to design security test patterns. These defined mechanisms aim at developing test patterns based on (i) security issues raised from the description, common consequences, demonstrative examples, and observed CAPEC, CVE or CWE examples, and (ii) the definition of a list of keywords extracted from the Certification Commission for Health Information Technology (CCHIT) Ambulatory Criteria [16] in order to help pointing a tester towards the correct security test pattern.

Although their notion of security test pattern is simple, maybe too short and does not allow completing our requirements in terms of test pattern template (see Section 3), they provide good and relevant basics to provide a first draft of security test patterns catalogue. We depict then in the following a list of nine security test patterns described using the template above proposed.


Attacking a Session Management.....	24
Attack of the authentication mechanism.....	26
Testing the safe storage of authentication credentials	27
Open Redirect	28
Uploading a malicious file	29
Searching for documented passwords	30
Impersonating an external server	31
Accessing resources without required credentials	32
Ensuring confidentiality of sensitive information	33

4.2.1 Attacking a Session Management

Pattern name	Session Management Attack
Context	Testing Approach(es): behavioral and test data
Problem/Goal	This pattern addresses how to check that the system returns an authorization error when the session information is faked or forged, and that no sensitive information is returned after requests. Relevant for managing/controlling access the system.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Set up a proxy to monitor all HTTP or TCP traffic flowing to or from the server. 2. Authenticate to the system as a registered user.


	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 25 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

	<ol style="list-style-type: none"> 3. Access one other page or screen (besides the home page or welcome screen) that requires authorization. 4. Log out. 5. Examine a captured HTTP request or TCP packet that is related to the access of the page other than the homepage. Identify headers or fields within the request or packet that may identify session identification information. 6. Modify a field identified in the earlier step (either by incrementing/decrementing them, removing them, replacing them with a different value entirely) and send this packet or request again. 7. Repeat the previous step for up to five fields identified in the packet or header. 8. Examine the cookies or local connection information (for systems that are not browser-based). Identify headers or fields within the cookie or local connection information that may identify session identification information. 9. Modify a field identified in the earlier step (either by incrementing/decrementing, removing, replacing with a different value entirely) and attempt to access the page or screen again without logging in. 10. Repeat the previous step for several other fields identified in the local connection information or cookies.
Known uses	Common Criteria SFRs: FMT_MOF.1, FMT_MSA
Discussion	<p>Since field modifications and resource access have to be done, the evaluation of this pattern should be performed online.</p> <p>A difficulty would be to manage encryption on the platform as well as identification of relevant fields.</p>
Related patterns	<ul style="list-style-type: none"> - Testing the safe transmission of authentication credentials - Modify Header Data - Modify Cookies or other Stored Information
References	CWE-311 [13] and CWE-807 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 26 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.2 Attack of the authentication mechanism

Pattern name	Attacking Authentication Mechanism
Context	Testing Approach(es): detection, test data
Problem/Goal	This pattern addresses how to check that the system handles high number of authentication attempts with incorrect passwords. Relevant for authenticating multiple users through several simultaneous connections (performance).
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Write a script that captures and replays the sequence of HTTP or TCP signals for authenticating to the server. 2. Use this script to launch ten authentication requests with ten separate passwords from a list of frequently used passwords. 3. If the system attempts to block any of these incorrect authentication requests, check that there are no manipulatable fields in the headers or parameters involved in these requests that indicate the high number of the authentication requests. 4. Examine the request and response sequences for each of those HTTP or TCP signals and identify fields that may contain session identification information. 5. Run the script for 1000 connections simultaneously.
Known uses	Common Criteria SFRs: FIA_AFL.1 and FIA_UAU.1
Discussion	The evaluation should be performed online. Pitfalls: write a script capturing and replaying HTTP/TCP messages as well as searching for manipulatable fields. Some knowledge on the system under test are necessary.
Related patterns	<ul style="list-style-type: none"> - Test for Common Usernames and Passwords - Attacking the Authentication Nonce - Logging in more than X time - Obtain a Plethora of Connections
References	CWE-307 [13], CWE-798 [13], CWE-770 [13]and CWE-327 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 27 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.3 Testing the safe storage of authentication credentials

Pattern name	Testing the safe storage of authentication credentials
Context	Testing Approach(es): detection
Problem/Goal	This pattern addresses how to check that the system store in a safe way the user authentication information. Relevant for user authentication management.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Set up a connection to monitor all HTTP or TCP traffic flowing to the server or from the server. 2. Authenticate to the system as a registered user. 3. If the system is web-based, examine all cookies related to the system under test (e.g. by looking up its domain name). 4. Log out. 5. Access the system's database directly through a database management tool. 6. Find and view the table containing user authentication information (typically named similar to "users" or "userdata").
Known uses	Common Criteria SFRs: FIA_UAU.1, FIA_UID.1, FIA_UID.2
Discussion	The evaluation can be performed online or offline if the testing architecture is well defined. The information will be analyzed through the cookies and the userdata. An efficient database management tool must be used to check the user authentication information.
Related patterns	
References	CWE-311 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 28 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.4 Open Redirect

Pattern name	Redirect header manipulation
Context	Testing Approach(es): design
Problem/Goal	This pattern addresses how to check that the system handles correctly the users redirection after authentication. Relevant for URL parameters rejection.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Set up to record HTTP traffic. 2. Authenticate as a registered user 3. Browse to some pages other than the authentication page or homepage. 4. Observe the parameters sent to the web application in the URL. 5. Record any parameters that seem to indicate that the system is controlling where the user is to be redirected to after authentication. 6. Log out. 7. Manipulate the parameters recorded above to point to a dangerous or untrusted URL. 8. Log back in.
Known uses	Common CCHIT Criteria: AM 09.06 Criteria SFR: FTP_ITI.1
Discussion	The evaluation can be performed offline after 'randomly' manipulating and monitoring the system. Some parameters have to be carefully defined before their modifications.
Related patterns	
References	CWE-601 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 29 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.5 Uploading a malicious file

Pattern name	Malicious file upload
Context	Testing Approach(es): test data
Problem/Goal	This pattern addresses how to check that the system should reject the file upon selection or should not allow it to be stored. Relevant for controlling stored or uploaded files.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Authenticate as a registered user. 2. Open the user interface for action object. 3. Select and upload a malicious file in place of object. 4. View or download the malicious file.
Known uses	Common Criteria SFRs: FDP_SDI.1, FDP_SDI.2 and FDP_ITC.1
Discussion	The evaluation can be performed offline after uploading a malicious file. The system must provide the ability to save scanned documents as images.
Related patterns	- Malicious file
References	CWE-434 [13]

	<p>Initial Security Test Patterns Catalogue</p> <p>Deliverable ID: D3.WP4.T1</p>	Page : 30 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.6 Searching for documented passwords

Pattern name	Search for documented passwords
Context	Testing Approach(es): detection, test data
Problem/Goal	This pattern addresses how to check that the system should not list any default passwords or usernames that are hard-coded into the product.
Solution	<p>Test Procedure Template</p> <ol style="list-style-type: none"> 1. Search the system's documentation. 2. Look at the HTML or any marked-up text that is included with the system by default.
Known uses	Common Criteria SFRs: FPT_ITI.1 and FPT_ITC.1
Discussion	<p>The evaluation is performed offline.</p> <p>A pitfall is the identification of the elements we are looking for (users information or password).</p>
Related patterns	
References	CWE-798 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 31 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.7 Impersonating an external server

Pattern name	Impersonating trusted external resources
Context	Testing Approach(es): design, data
Problem/Goal	This pattern addresses how to check that the system refuses (or behave as such) to connect an impersonated server. Indeed, by DNS spoofing or DNS entry modifications, the authentic external server may be replaced. Relevant for checking trusted path/channels.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Set up a connection to monitor all HTTP or TCP traffic flowing to the server or from the server. 2. Authenticate as a registered user. 3. Open the user interface to action an object. 4. Identify any request that was sent to an external server and record it. 5. Impersonate the external server, either by changing the settings of the system to point to that server or by DNS spoofing the external server and replacing that DNS entry with the impersonated server. 6. Construct a response from the impersonated server that performs the same functionality as the authentic external server. 7. Open the user interface to action an object again. 8. Log out.
Known uses	Common Criteria SFRs: FTP_ITC.1 and FTP_TRP.1
Discussion	The evaluation is performed online. A pitfall could be the response to be built and sent by the tester
Related patterns	<ul style="list-style-type: none"> - DNS Spoofing an Update Site - Pointing to an Untrusted Update Site - Spoofing Functionality Provided in Untrusted Sphere
References	CWE-494 [13] and CWE-829 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 32 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

4.2.8 Accessing resources without required credentials

Pattern name	Exposing functionality requiring authorization
Context	Testing Approach(es): design
Problem/Goal	This pattern addresses how to check that the system disallows a user to action an object if she has not the proper credentials.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. If access to action an object requires authentication, authenticate as a registered user. 2. Open the user interface, either inside or outside of the main application, for actioning the object. 3. Record the series of mouse clicks, GUI interactions, or URL sequences required to get to this screen. 4. Log out and/or exit this screen. 5. Attempt to repeat the series of steps recorded above.
Known uses	Common Criteria SFRs: FDP_ACC.1, FDP_ACC.2 and FDP_ACF.1
Discussion	The evaluation is performed online while an analysis of the performed actions can be made offline. Some actions could be difficult to be automatized (forms to enter, specific values to provide through a database process). It will depend on the design of the user interface.
Related patterns	<ul style="list-style-type: none"> - Exposing Critical Functionality - Force Exposure of Function Requiring Authorization
References	CWE-306 [13] and CWE-862 [13]

	Initial Security Test Patterns Catalogue Deliverable ID: D3.WP4.T1	Page : 33 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public


4.2.9 Ensuring confidentiality of sensitive information

Pattern name	Sensitive information confidentiality
Context	Testing Approach(es): architectural
Problem/Goal	This pattern addresses how to check that the system use a known safe encryption protocol. Relevant to test if any sensitive or personal information contained within an object is only accessible to the user who actioned it.
Solution	Test Procedure Template <ol style="list-style-type: none"> 1. Authenticate as a registered user. 2. Open the user interface for actioning an object. 3. If necessary, open, view, or otherwise access the actioned object. 4. Log out.
Known uses	Common Criteria SFRs: FCS_COP.1
Discussion	The evaluation is performed either online or offline. Some actions on the objects could provide different behaviors that could eventually relate on other test patterns. Finally some expected results could be that (i) the connection to the server was made using a known safe encryption protocol (e.g. HTTP over SSL, or an encrypted TCP connection), and (ii) the manipulated object is encrypted with a safe encryption protocol, password-protected, or both.
Related patterns	<ul style="list-style-type: none"> - Testing the safe transmission and storage of sensitive personal information - Testing the safe transmission of sensitive data to an outside source - Force the Export of Sensitive Information
References	CWE-311[13], CWE-212[13]

5. SUMMARY OF TEST PATTERNS CATALOGUE

This deliverable has introduced the concept of security test patterns, as defined by the DIAMONDS project and has provided an initial catalogue of patterns addressing several attack patterns and known vulnerabilities likely to affect security of information systems. A total of 17 patterns have been presented, all based on a common template specifically designed for security testing and aligning to good practices of the pattern community.

Future work in DIAMONDS will consist in consolidating these patterns and in enriching the catalogue with new test patterns identified in the case studies running in the project.

	<p align="center">Initial Security Test Patterns Catalogue</p> <p align="center">Deliverable ID: D3.WP4.T1</p>	Page : 34 of 34
		Version: 1.0 Date : 30.05.2012
		Status : Final Confid : Public

REFERENCES

- [1] C. Alexander; A Pattern Language: Town, Buildings, Construction; Oxford, UK: Oxford University Press, 1977.
- [2] R. Binder. Testing Object Oriented Systems: Models, Patterns and Tools. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.
- [3] A. Vouffo Feudjio, I. Schieferdecker; Test patterns with TTCN-3; Proceedings from the International Workshop on Formal Approaches to Testing of Software - FATES/RV , pp. 170-179, 2004
- [4] A. Vouffo Feudjio, I. Schieferdecker: A Pattern Language of Black-Box Test Design for Reactive Software Systems. EuroPLOP 2009
- [5] A. Vouffo Feudjio, A Methodology For Pattern-Oriented Model-Driven Testing of Reactive Software Systems, PhD Thesis, Feb. 2011,
http://opus.kobv.de/tuberlin/volltexte/2011/3103/pdf/vouffoFeudjio_alainGeorges.pdf
- [6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture, A System Of Patterns, Volume 1. Wiley Series in Software Design Patterns, 2001.
- [7] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann and P. Sommerlad, Security Patterns – Integrating Security and Systems Engineering, Wiley Series in Software Design Patterns, 2006.
- [8] ITSEC: Information Technology Security Evaluation Criteria, Version 1.2 June 1991.
http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf
- [9] ITSEC, <http://en.wikipedia.org/wiki/ITSEC>
- [10] Mitre, “Common Attack Pattern Enumeration and Classification”, <http://capec.mitre.org/index.html>.
- [11] Mitre, “Common Vulnerabilities and Exposures ” <http://cve.mitre.org/>
- [12] Mitre, “Common Weakness Risk Analysis Framework (CWRAF™)”, Mitre, <http://cwe.mitre.org/cwraf/>
- [13] Mitre, “Common Weakness Enumeration” <http://cwe.mitre.org>
- [14] OWASP, “OWASP Testing Guide v3”,
http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf
- [15] Reasearch research group, “Security Test Patterns”, <http://securitytestpatterns.org/doku.php>
- [16] <http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Ambulatory%20EHR%20Criteria%2020110517.pdf>, May 2011.
- [17] Common Criteria for Information Technology Security Evaluation, Version 3.1, Part 1: Introduction and general model, Revision 3, July 2009,
- [18] Common Criteria for Information Technology Security Evaluation, Version 3.1, Part 2: Security functional components, Revision 3, July 2009,
- [19] Common Criteria for Information Technology Security Evaluation, Version 3.1, Part 3: Security assurance components, Revision 3, July 2009.
- [20] S. Christey (Ed.), B. Martin, M. Brown, A. Paller, D. Kirby; 2011 CWE/SANS Top 25 Most Dangerous Software Errors; <http://cwe.mitre.org/top25/>; Document version: 1.0.1; June 27, 2011
- [21] CAPEC 94: Man in the Middle Attack; <http://capec.mitre.org/data/definitions/94.html>