Title: Review of Security Testing Techniques

Version: 1.1.1
Date : 30.06.2011
Pages : 136

Editor: Jean-Luc Richier

Reviewers: Guiseppe Bua, Carlo Harpes, Ilkka Uusitalo

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:	**Confidentiality:**	
[ ] Draft	[X] Public	Intended for public use
[ ] To be reviewed	[ ] Restricted	Intended for DIAMONDS consortium only
[ ] Proposal	[ ] Confidential	Intended for individual partner only
[X] Final / Released		

Deliverable ID: D1.WP2

Title:

Review of Security Testing Techniques

Summary / Contents:

This document lists approaches dedicated to testing software security.

Contributors to the document:

Bernhard Aichernig (TU Graz), Michael Berger (Fraunhofer FOKUS), Fabrice Bouquet (Smartesting), Ana Cavalli (GET IT), Jürgen Großmann (Fraunhofer FOKUS), Roland Groz (Grenoble INP), Carlo Harpes (itrust), Andreas Hoffmann (Fraunhofer FOKUS), Ami Juuso (Codenomicon), Rauli Kaksonen (Codenomicon), Bruno Legeard (Smartesting), Stéphane Maag (GET IT), Wissam Mallouli (Montimage), Florian Marienfeld (Fraunhofer FOKUS), Nadja Menz (Fraunhofer FOKUS), Edgardo Montes de Oca (Montimage), Pramila Mouttapa (GET IT), Jean-Luc Richier (CNRS/Grenoble INP), Ina Schieferdecker (Fraunhofer FOKUS), Fredrik Seehusen (SINTEF), Ari Takanen (Codenomicon), Juha Matti Tirila (Codenomicon), Tuomo Untinen (Codenomicon), Miia Vuontisjarvi (Codenomicon), Franz Wotawa (TU Graz)

TABLE OF CONTENTS

FIGURES

TABLES

HISTORY

Vers.	Date	Author	Description
1.0	2011/05/31	JL Richier	Public version integrating contributors' data and reviewers' comments
1.1	2011/06/20	JL Richier	Added section 3.2.6 (moved from D1.WP3). added section 2.2.2, some editing
1.1.1	2011/06/30	JL Richier	Corrections in the bibliography references

APPLICABLE DOCUMENT LIST

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

EXECUTIVE SUMMARY

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

Model Based Testing is a successful approach to derive tests from formal description of information about the system considered to derive systematic tests. These models may describe the behaviour of the system, security constraints (for example access control), the security requirements, or information about possible security threats, faults or attacks. Chapter 1 of this document describe different models used in security testing.
Two main techniques of test are often considered: active and passive testing:
Passive testing consists of detecting faults in a system under test by observing its input/output behaviours without interfering with its normal operations. The usual approach of passive testing consists of recording the trace produced by the implementation under test and comparing this trace with a specification. Other approaches explore relevant properties required for a correct implementation, and then check them on the implementation traces of the system under test. Chapter 2 describes current approaches for passive testing.
Active testing is accomplished by applying a sequence of inputs to the implementation, by means of an external tester, and verifying whether the sequence of outputs is the one specified. These sequences may be constructed from the models discussed above. Most of the works are on testing software control parts, and technologies are related to the approaches of model-based testing. Traditional testing methods tend to test a system as a whole or to test their components in isolation. Testing these systems as whole becomes difficult due to the large number of combinations of system states and variable values, known as the state space explosion. It is a challenge to be able to minimize the number of tests needed while guaranteeing good fault coverage. This standard method is mainly oriented towards practical needs. Chapter 3 describes current approaches for active testing.
Security testing often lacks systematic approaches, that enable the efficient and goal oriented identification, selection and execution of test cases. A successful approach to resolve this problem is to use risk-oriented testing, where one uses software risks analysis as the guiding factor to solve decision problems during testing, e.g. the selection and prioritization of test cases. This approach is advocated by different security testing standards and manuals (NIST, OSSTMM etc.). Chapter 4 reviews the current approaches in this domain. It describes different risk assessment, modelling and managements approaches and techniques, such as the common criteria; ISO27000, Octave, Trick-light, and how they apply to testing.

INTRODUCTION

Testing is still the main method to check the functionality, robustness, performance, scalability, reliability and resilience of systems as it is the only method to derive objectively characteristics of a system in its target environment.
In the case of security, software systems are examined, using software security testing, for security properties such as confidentiality, integrity, authentication, authorization, availability, and non-repudiation. In general the software security testing activities can be divided into functional security testing and security vulnerability testing [88]. While security functional testing is used to check the functionality, efficiency and availability of the specified and carefully planned security functionalities and systems (e.g. firewalls, authentication and authorization subsystems, access control), security vulnerability testing directly addresses the identification and discovery of actually undiscovered system vulnerabilities that are introduced by security design flaws.
Systematically testing a system for its security is a relatively new concern. Of course, a number of tools have long been around to target specific attacks on systems (e.g. vulnerability scanners). Different broader approaches have also been proposed in the recent years.
This document lists a number of approaches dedicated to testing software security. It is organized as follows: Chapter 1 presents approaches to describe the security requirements expected. Chapter 2 describes passive testing techniques. Passive testing is the activity of detecting faults in a system under test by observing its input/output behaviours without interfering with its normal operations. In chapter 3, active testing techniques are presented; active testing consists in applying a sequence of inputs to the implementation, by means of an external tester, and verifying whether the sequence of outputs is the one expected. Chapter 4 describes risk based testing. Risk-oriented testing or risk-based testing characterize a methodology that makes software risks the guiding factor to solve decision problems during testing, e.g. the selection and prioritization of test cases.

1. SECURITY PROPERTY MODELLING METHODS

In this section we describe how to express the expected security requirements. In Section 1.1 the following security property modelling languages are detailed: First order logic and Deontic Logic which are Logic-Based languages and R-BAC, OrBAC, Invariants, Nomad, Regular Expression, Linear Temporal Logic (LTL), Computational Tree Logic (CTL). Section 1.2 shows how information can be combined to provide Vulnerability Analysis using Attack Graphs.

1.1 MODELLING SECURITY PROPERTIES

Security and reliability are of paramount importance in designing and building systems because any security failure can put critical information and environment at risk. A security property expresses the obligation, permission or interdiction to perform actions under given conditions. In this section, we describe some languages we use to model security properties like logic-based languages (First order logic, Deontic logic), R-BAC, Or-BAC that deals with security rules without involving time aspects, invariants, Nomad (that allows expressing security rules with timed constraints), regular expressions, LTL, and CTL.

1.1.1 Logic-Based Languages

First Order Logic
There are several examples that illustrate the applications of first order logic to the specification of security policies. These include the logical notation introduced in [48] and the Role Definition Language (RDL) presented in [94] and RSL99 [3]. All of these approaches are based on R-BAC. In addition to these R-BAC examples, Z is a formal specification language that combines features of first order predicate logic with set theory [230]. In [30], the use of Z to specify and validate the security model for the NATO Air Command and Control System is described. One of the main problems encountered when using first order logic for policy specification arises when negation is used together with recursive rules. This leads to logic programs that cannot be decided and cannot be evaluated in a flounder-free manner [56]. Although it is possible to avoid the use of negation or recursion, this is not practical since it diminishes significantly the expressive power of the logical language. Other languages have been then defined.

Deontic Logic
Deontic logic was developed and started in the 1950s and revisited by Castaneda [41] in 1981 by extending modal logic with operators for permission, obligation and prohibition. Known as Standard Deontic Logic (SDL), traditionally it has been used in analyzing the structure of normative law and normative reasoning in law.
Typically, deontic logic uses *OA* to mean it is obligatory that A, *PA* to mean it is permitted (or permissible) that A and *FA* to mean it is prohibited that A. The term deontic is derived from the ancient Greek ***deon***, meaning, roughly, that which is binding or proper. In the following, we summarize the basic axioms of the SDL language:

- Tautologies of propositional calculus
- $O(p \rightarrow q) \rightarrow (Op \rightarrow Oq)$
  If there is an obligation that p implies q, then an obligation to do p implies an obligation to do q.

- *Op→Pp*
  If there is an obligation to do p then p is permitted.
- *Pp ↔ ¬O¬p*
  Iff p is permitted then there is no obligation to not do p. In other words, iff p is permitted then there is no refrain policy with respect to p
- *Fp ↔ ¬Pp*
  Iff p is forbidden then there is no permission to do p.

SDL is used to represent security policies with the aim of detecting conflicts in the policy specifications. This approach is based on translating the SDL representation into first order predicate logic before performing the necessary conflict detection analysis. An inherent problem with the deontic logic approach is the existence of a number of paradoxes. For example, Ross paradox, i.e. if the system is obliged to perform the action send message, then it is obliged to perform the action send message or the action delete message. Although there is some works that offer resolutions to these paradoxes [186][187], the existence of paradoxes can make it confusing to discuss policy specifications using deontic logic notations. Besides, this is mainly the reason why there are currently very few research works devoted to the security properties testing based on deontic logics. However, some specific studies have been declined such as the Nomad languages depicted in the following.

1.1.2 R-BAC

Role-Based Access Control (R-BAC) [205] is an approach to restrict system access to authorized users. It is a newer alternative approach to mandatory access control (MAC) and discretionary access control (DAC). R-BAC is a policy neutral and flexible access control technology sufficiently powerful to simulate both DAC and MAC policies.
Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. System users are assigned to particular roles, and through those role assignments acquire the permissions to perform particular system functions. Roles permit the grouping of a set of permissions related to a position in an organization such as finance director, network operator, ward-nurse or physician. This allows permissions to be defined in terms of the position rather than the person assigned to the permission, so policies do not have to be changed when people are reassigned to different positions within the organization. Unlike Context-Based Access Control [86] (C-BAC), R-BAC does not look at the message context (such as where the connection was started from).
Since users are not assigned permissions directly, but only acquire them through their role(s), management of individual user rights becomes a matter of simply assigning the appropriate roles to the user, which simplifies common operations such as adding a user or changing a user's department.
Another motivation for R-BAC has been to reuse role specification by a form of inheritance whereby one role (often a superior in the organization) can inherit the rights of another role and thus avoid the need to repeat the specification of permissions.

Examples [53]:
(*R*, {*po.approve*, *cheque.raise*}, *static*)
- "no role can (have the permission to) approve purchase orders and raise cheques"

(*jason*, {*fin_clerk*}, *static*)
- "*Jason* cannot be assigned to the finance clerk role"

(*U*, {*fin_clerk, po_clerk*}, *static*)

– "no user can be assigned to both the finance clerk and purchasing clerk roles"
– *U* is the scope of the constraint

Chen and Sandhu [48] introduce a language based on set theory for specifying R-BAC state related constraints, which can be translated to a first-order predicate-logic language. They define an R-BAC system state as the collection of all the attribute sets describing roles, users, privileges, sessions as well as assignments of users to roles, permissions to roles and roles to sessions. They also define constraints as the specification of restrictions to R-BAC states, called invariants, as well as to state changes, called preconditions. They use this model to specify constraints for R-BAC in two ways: (i) by treating them as invariants that should hold at all times, and (ii) by treating them as preconditions for functions such as assigning a role to a user. They define a set of global functions to model all operations performed in an R-BAC system, and specify constraints which include: conflicting roles for some users, conflicting roles for sessions of some users, and prerequisite roles for some roles with respect to other users.

1.1.3 Or-BAC

Or-BAC (Organisational Based Access Control) [66] is an access control model and a language to describe security policies of an organisation. For this purpose, Or-BAC defines two abstraction layers. The first one is called "abstract" and describes a rule as a *role* having the permission, prohibition or obligation to perform an *activity* on a *view* in a given *context*. A *view* is a set of objects to which the same security rule applies. *Contexts* are used to specify the concrete circumstances where organisations grant roles permissions (or/and prohibitions, or/and obligations) to perform activities on views. The second layer or level is the "concrete" one. It is derived from the abstract level and grants permissions, prohibitions or obligations to a user to perform an *action* on an *object*. Thus, according to Or-BAC syntax, a typical security rule has the following form:

Obligation (*S*, *R*, *A*, *V*, *C*)

This rule means that within the system *S*, role *R* is obliged to perform the activity *A* targeting the objects of view *V* in the context *C*. The principle is similar for permissions (keyword *Permission*) and prohibitions (keyword *Prohibition*).

This approach manipulates three different inputs: a functional specification of the system based on a well-know mathematically-based formalism: Extended Finite State Machine, a specification of the security policy based on OrBAC model that is applied to this system and finally an implementation of the system.

Examples using Or-BAC

Obligation(Website, anonymous, Authentication, ,input = AddPostReq)

One can notice that through the obligations, the policy specifies a new activity to perform access control that is authentication. As a consequence, a differentiation is created among users. Only if a person is authenticated he is given the privilege to perform posting in a blog.

Permission (Website, admin, `Deleting Post', post, PostId < PostIdMax+1)

Admin is considered as the 'almighty' that has all rights to perform any action in the blog.

Permission (Website, blogger, `Adding Comment', comment, _)
Prohibition (Website, blogger, `Deleting Post', post, PostId < PostIdMax+1)
The second role is the blogger. This one can read and write everything but cannot delete posts and comments

Permission (Website, anonymous, `Reading Blog', blog, _)
Prohibition (Website, anonymous, `Adding Comment', comment, _)
An anonymous visitor is only permitted to perform reading on objects. All other actions are prohibited for him

1.1.4 Invariants

The invariants [8] are used to express vulnerabilities that are based on regular expression extended with predicate expressions. This language allows us to express conditional expressions to detect vulnerabilities in a code. A security invariant is an IF-THEN property. It allows expressing the desired behaviour of a communicating system and describes the correct order of messages collected in the trace with potential data and time constraints. If specific conditions on the exchanged messages hold, then the occurrence of a set of events must happen. An event is a set of conditions on some field values of the exchanged packets.

A timed extended invariant is an IF-THEN expression that allows expressing a property regarding the messages exchanged in a captured trace P = {p1, ..., pm'}. It has the following syntax [169]:

$$e1 \xrightarrow{\text{When},n,t} e2$$

This property expresses that if the event e1 is satisfied (by one or several packets pi, i∈{1,...,m}), then event e2 must be satisfied (by another set of packets pj, j∈{1,...,m}) before or after (depending on the When value) at most n packets and t units of time.
It is important to highlight that in the invariant it is possible to disable n and/or t by setting their values to -1. If (When = AFTER), we say that the invariant is a *simple invariant*. If (When = BEFORE), we have an *obligation invariant*.

Example
Here are few examples for expressing security invariants in the OLSR protocol [44]:

- ? / *MsgType* = *Hello* ^ *NeighAddr* = *none* ^ *Source* = *n*0 ^ *Dest* = *any*, * , *MsgType* = *Hello* ^ *Source* = *n*1 ^ *Dest* = *any* ^ *LinkType* = *Asym* / ?

This invariant illustrates the case where a node starts the neighbour detection mechanism by sending *HELLO* messages. In this case, the link established is asymmetrical because a neighbour answers, announcing this link. It is expressed as an obligation invariant, which means that if a packet is received by node *n*0 announcing an asymmetrical link between nodes *n*0 and *n*1, then it is mandatory that *n*0 should have already sent an empty *HELLO* message to *n*1.

- *MsgType* = *Hello* ^ *LinkType* = *Sym* ^ *Source* = *n*1 ^ *NeighAddr* = *n*0(*AsymList*) / ?, *, *MsgType* = *Hello* ^ *NeighAddr* = *n*0 ^ *Source* = *n*1 ^ *Dest* = *any* ^ *LinkType* = *MPR* / ?

This invariant illustrates the case where the current node receives a message from a node with a symmetric link, announcing that it has been selected as MPR by this node. It is expressed as an obligation invariant, which means that if a packet is received by *n*0 from *n*1 announcing a MPR link, then it is mandatory that *n*1 should have already established a symmetrical link with *n*0.

1.1.5 Nomad

Nomad language [55] provides a way to describe permissions, prohibitions and obligations related to non-atomic actions within different contexts. Nomad allows also expressing privileges on non-atomic actions. For instance, Nomad permits to express a permission to execute action β only and only if action α has just been executed. It combines deontic, temporal logics and timed aspects that allow specifying conditional privileges and obligations with deadlines.

Definition 1: (Atomic action) An *atomic action* is defined in Nomad as one of the following actions: a variable assignment, a clock setting, an input action, an output action, a process creation or a process destruction.
Definition 2: (Non-atomic action) If A and B are atomic actions, then the action (A;B) which means "A is followed immediately by B", is a *non-atomic action*.
Definition 3: (Formulae) If A is an action then start(A) (starting A), and done(A) (finishing A) are *formulae*.

Here are some properties on *actions* and *formulae*:

- If *A* and *B* are formulae then (*A v B*), ¬*A* are formulae.
- If α is a formula then $O^{-d}\alpha$ (α was true *d* units of time ago if *d* ≥ 0) is a formula too.
- If α is a formula then $O^{<-d}\alpha$ (α was true within the last *d* units of times if *d* ≥0) is a formula too.
- (α|γ) is a formula whose semantics states that in the context γ, the formula α is true.
- If α is a formula then P(α) (α is permitted), F(α) (α is forbidden), O(α) (α is mandatory) are formulae.

The operators "O" and "|" refers to timed and contextual operators respectively. Also, we use the notation $O^{[<]-d}$ to cover both cases O^{-d} and $O^{<-d}$. Notice also that using Nomad formalism, we deal with a discrete time. The choice of the unit of time can be very important and depends on the studied system.

Definition 4: (A security rule) If α and β are formulae, R(α| β) is a security rule where R denotes one of the following three deontic operators: { *P*, *F*, *O* }. The expression *P*(α|β) (respectively *F*(α|β), *O*(α|β)) means that it is permitted (respectively prohibited, mandatory) to execute α when β context holds.
Hereafter, we present some examples of security rules specifications according to the Nomad language:

P **(start (input ReqWrite(file1.doc))|**$\mathbf{O^{\leq -5s}}$ **done (input AuthReq();output AuthOK))**
This rule expresses a permission granted to any user to request to write in 'file1.doc', if earlier (within the last 5 seconds), he/she was successfully authenticated in the system.

O **(start (output DisconnectOK()) |**$\mathbf{O^{\leq -30min}}$**¬ done (input _))**
According to this obligation rule, the system must disconnect any user if the user remains inactive (¬done (input _)) for 30 minutes.

F **(start (input AuthReq()) |O≤−0.01s done (input AuthReq(); input AuthReq()))**

This prohibition rule means that it is forbidden that the system manages more than two authentication requests in the same millisecond.

1.1.6 Regular Expressions

Regular expressions [148] are often well known and used in many tools such as text editors, lexical analyzers, etc. They allow the description of sequences of patterns by means of operators such as · (sequence of patterns), * (repetition of a pattern a finite – possibly null – number of times) or + (repetition of a pattern a finite non-null number of times). Also, we sometimes use an interval instead of the operators * or + to bound the number of possible repetitions of a pattern. For example, the replacement of *a* _ by *a* 0..3 indicates that the pattern *a* can be repeated between 0 and 3 times. We also use the symbol × to denote a number of repetitions ranging into {1, 0..n, _,+}. Thus, *a*× can be interpreted either as *a*, as *a* 0..n, as *a* _ or as *a* +.

(i) Sequencing of GET CHALLENGE and EXTERNAL AUTH.
We recall that this property specifies that any call to EXTERNAL AUTH must be immediately preceded by a successful call to GET CHALLENGE. Operation EXTERNAL AUTH is denoted as EA and operation GET CHALLENGE is denoted as GC. This property can be expressed as a regular expression in the following way:

$$(op_{\overline{GC,\ EA}}{}^{*} .\ GC\_success\ .\ EA\_success)^{*}$$

(ii) Sequencing of EXTERNAL AUTH and INTERNAL AUTH.
We recall that this property states that when INTERNAL AUTH is called, the last call to EXTERNAL AUTH must have been successful. The operation INTERNAL AUTH is abbreviated as IA. The property is expressed with a regular expression as

$$((op_{\overline{EA,IA}})^{0..1} \cdot EA\_success \cdot (op_{EA\_error})^{*} \cdot IA\_success)^{*}$$

The above examples are few security properties expressed as regular expressions in the application of smart card systems.

1.1.7 Linear Temporal Logic (LTL)

Linear temporal logic (LTL) [184] is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths such that a condition will eventually be true, that a condition will be true until another fact becomes true, etc.

LTL is built up from a set of propositional variables, the usual logic connectives $\neg, \vee, \wedge, \rightarrow$and the following temporal modal operators:

X for ne**x**t (**N** is used synonymously)
G for always (**g**lobally)
F for eventually (in the **f**uture)
U for **u**ntil
R for **r**elease

The first three operators are unary, so that **X** φ is a well-formed formula whenever φ is a well-formed formula. The last two operators are binary, so that φ **U** ψ is a well-formed formula whenever φ and ψ are well-formed formulas.

Example [93]

(¬*Authentication* ^ ¬*Secure*) U (*Authentication* ^ ¬*Secure* ^ X (¬*Secure* U (*Secure* ^ ¬*Authentication* ^ X (¬*Authentication* U (*Authentication* ^ ¬*Secure* ^ X (¬*Secure* U (*Secure* ^ ¬*Authentication* ^ X (¬*Authentication* U (*Authentication* ^ ¬*Secure* ^ X (G (¬*Secure*))))))))))

The user accesses a secure page only twice and each time after a visit to Authentication page

1.1.8 Computational Tree Logic (CTL)

Computation tree logic (CTL) [103] is a branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realised. It is used in formal verification of software or hardware artefacts, typically by software applications known as model checkers, which determine if a given artefact possesses safety or liveness properties. It is in a class of temporal logics that include linear temporal logic.
The temporal operators are the following:

- Quantifiers over paths
  - **A** φ - **A**ll: φ has to hold on all paths starting from the current state.
  - **E** φ - **E**xists: there exists at least one path starting from the current state where φ holds.
- Path-specific quantifiers
  - **X** φ - Ne**x**t: φ has to hold at the next state (this operator is sometimes noted **N** instead of **X**).
  - **G** φ - **G**lobally: φ has to hold on the entire subsequent path.
  - **F** φ - **F**inally: φ eventually has to hold (somewhere on the subsequent path).
  - φ **U** ψ - **U**ntil: φ has to hold *at least* until at some position ψ holds. This implies that ψ will be verified in the future.
  - φ **W** ψ - **W**eak until: φ has to hold until ψ holds. The difference with **U** is that there is no guarantee that ψ will ever be verified. The **W** operator is sometimes called "unless".

For example, the formula: **AG (p → (EFq))** is a well-formed CTL formula. We read it as “From now on (**A**) and no matters what happens (**G**), if **p** is true, then there **E**xists a path in the model such that in the **F**uture, **q** is true”.

Example [232]

Generalized Temporal RBAC (GTRBAC) is an extension of RBAC allowing to express a wide range of timing properties. Separation of Duty (SoD) is one of these important temporal constraints. Its meaning is that if a user has activated an RBAC role then he should not be allowed to activate any other conflicting role in a particular time interval. From that, an important security timing property *P* may therefore be defined: “a user should never activate more than one conflicting role in a time interval”. By *activate*, we mean the predicate representing a hierarchical role relation, i.e. whether a user can activate a role at a time t. This is related to the predicates *active(r,t)* and *active(u,t)* mentioning respectively if *r* is active at time *t* or if u is active at time *t.* In CTL, the property *P* can be expressed such as:

AG (((active(u,t) → active(r1,t)) → ¬(active(u,t) → active(r2,t))) V
((active(u,t) → active(r2,t)) → ¬(active(u,t) → active(r1,t))))

1.2 VULNERABILITY ANALYSIS USING ATTACK GRAPH

1.2.1 What is an Attack Graph?

Attack graphs [257] are used to determine if designated goal states can be reached by attackers attempting to penetrate computer networks from initial starting states. For this use, they are graphs in which the starting node represents an attacker at a specified network location. Nodes and arcs represent actions the attacker takes and changes in the network state caused by these actions. Actions typically involve exploits or exploit steps that take advantage of vulnerabilities in software or protocols. The goal of these actions is for the attacker to obtain normally restricted privileges on one or more target hosts, where the target could be a user's computer, a router, a firewall, or some other network component. Many actions that compromise separate hosts and use them as stepping stones may be required in large attack graphs to reach the target host. A full attack graph will show all possible sequences of attacker actions that eventually lead to the desired level of privilege on the target.
As networks of hosts continue to grow in size and complexity, evaluating their vulnerability to attack become increasingly more important to automate. This makes the generation of attack graph become a classical problem for network security analysis. Various kinds of attack graphs have been proposed for analyzing network security.

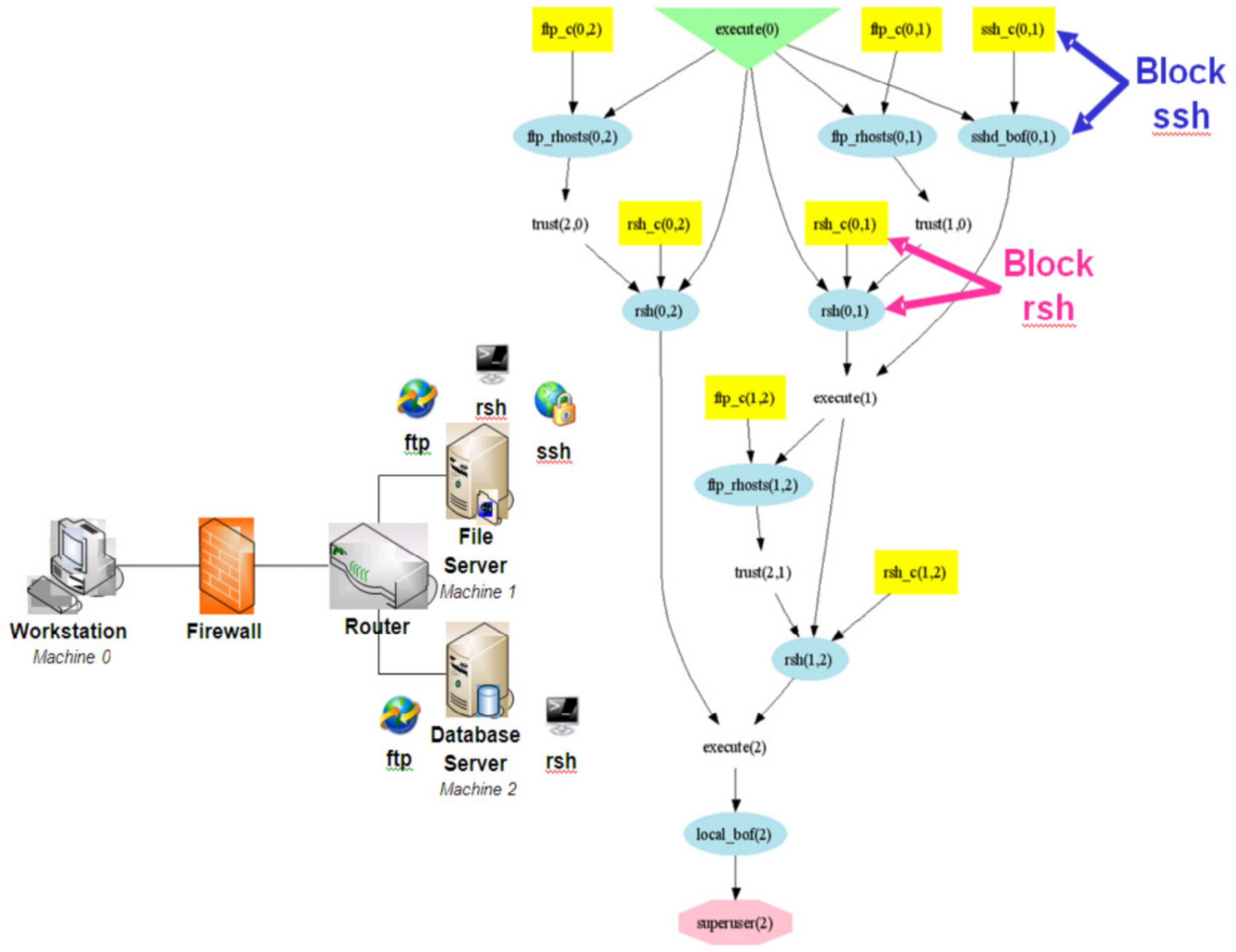


Figure 1: Example Network, Attack Graph, and Network hardening choices

Example of an Attack Graph

The attack graphs [173] model how multiple vulnerabilities may be combined for an attack. They represent system states using a collection of security-related conditions, such as the existence of vulnerability on a particular host or the connectivity between different hosts. Vulnerability exploitation is modelled as a transition between system states.

As an example, consider Figure 1. The left side shows a network configuration, and the right side shows the attack graph for compromise of the database server by a malicious workstation user. In the network configuration, the firewall is intended to help protect the internal network. The internal file server offers file transfer (ftp), secure shell (ssh), and remote shell (rsh) services. The internal database server offers ftp and rsh services. The firewall allows ftp, ssh, and rsh traffic from a user workstation to both servers, and blocks all other traffic.

In the attack graph, attacker exploits are blue ovals, with edges for their preconditions and post conditions. The numbers inside parentheses denote source and destination hosts. Yellow boxes are initial network conditions, and the green triangle is the attacker's initial capability. Conditions induced by attacker exploits are plain text. The overall attack goal is a red octagon. The figure also shows the direct impact of blocking ssh or rsh traffic (to the fileserver) through the firewall, i.e., preventing certain exploits in the attack graph.

The attack graph includes these attack paths:
(1) sshd_bof(0,1) → ftp_rhosts(1,2) → rsh(1,2) → local_bof(2)
(2) ftp_rhosts(0,1) → rsh(0,1) → ftp_rhosts(1,2) rsh(1,2) → local_bof(2)
(3) ftp_rhosts(0,2) → rsh(0,2) → local_bof(2)

The first attack path starts with sshd_bof(0,1). This indicates a buffer overflow exploit executed from Machine 0 (the workstation) against Machine 1 (the file server), i.e., against its secure shell service. In a buffer overflow attack, a program is made to erroneously store data beyond a fixed-length buffer, overwriting adjacent memory that holds program flow data. The result of the sshd_bof(0,1) exploit is that the attacker can execute arbitrary code on the file server. The ftp_rhosts(1,2) exploit is now possible, meaning that the attacker exploits a particular ftp vulnerability to anonymously upload a list of trusted hosts from Machine 1 (the file server) to Machine 2 (the database server). The attacker can leverage this new trust to remotely execute shell commands on the database server, without providing a password, i.e., the rsh(1,2) exploit. This exploit establishes the attacker's control over the database server as a user with regular privileges. A local buffer overflow exploit is then possible on the database server, which runs in the context of a privileged process. The result is that the attacker can execute code on the database server with full privileges.

System administrators analyze attack graphs [217] to understand where their system's weaknesses lie and to help decide which security measures will be effective to deploy. Attack graph input data is based on a wide variety of sources. Besides network topology information, host weakness information, and vulnerability information, attacker models, trust relationships, and security policies may be required. The larger and more detailed this data set is, the more valuable the resulting attack graph will be. Gathered data will be unified before it is aggregated into an overall model.

The first step is the information gathering. Before an attack graph can be generated, a uniform representation of data from various sources must be created. The two most important types of

information are the network model and data on vulnerable hosts' information. Information about the network infrastructure is important to understand possible access paths between hosts..
Without a network model, an attack graph tool would not be able to relate different hosts to one another and outline imminent attack paths. The network topology can be extracted from various sources, such as network scans, firewall rules, or file-based descriptions of the network structure.
Information about weaknesses on hosts is crucial to know which hosts are vulnerable to what kind of attacks. Tools such as Nessus and nmap may used. Such information includes for example the operating system, running services like an email server, and even detailed version information for a more precise assessment of the involved system.

Before an attack graph can be constructed, an attack graph tool has to know how attacks can be conducted, i.e., how do vulnerabilities relate to each other, what preconditions are necessary for an attack to take place, and what post conditions will an attack result in.

1.2.2 Procedure for constructing an attack graph

- Once both system information and vulnerability information are gathered and analyzed, an attack graph constructor generates the attack graph model. Attack graph constructor will compute a reachability matrix, i.e., what computer is directly connected to which set of other computers.
- Next, the host weaknesses as well as the vulnerability information are integrated. As a result, an attack graph has been constructed.
- The next step involves the interaction with a user. This user can now analyze the attack graph, appoint most valuable hosts, identify shortest paths, test whether the elimination of certain steps would break all attacks, and so on.
- The visualization of attack graphs can be based on the constructed attack graph or on an abstraction of it.

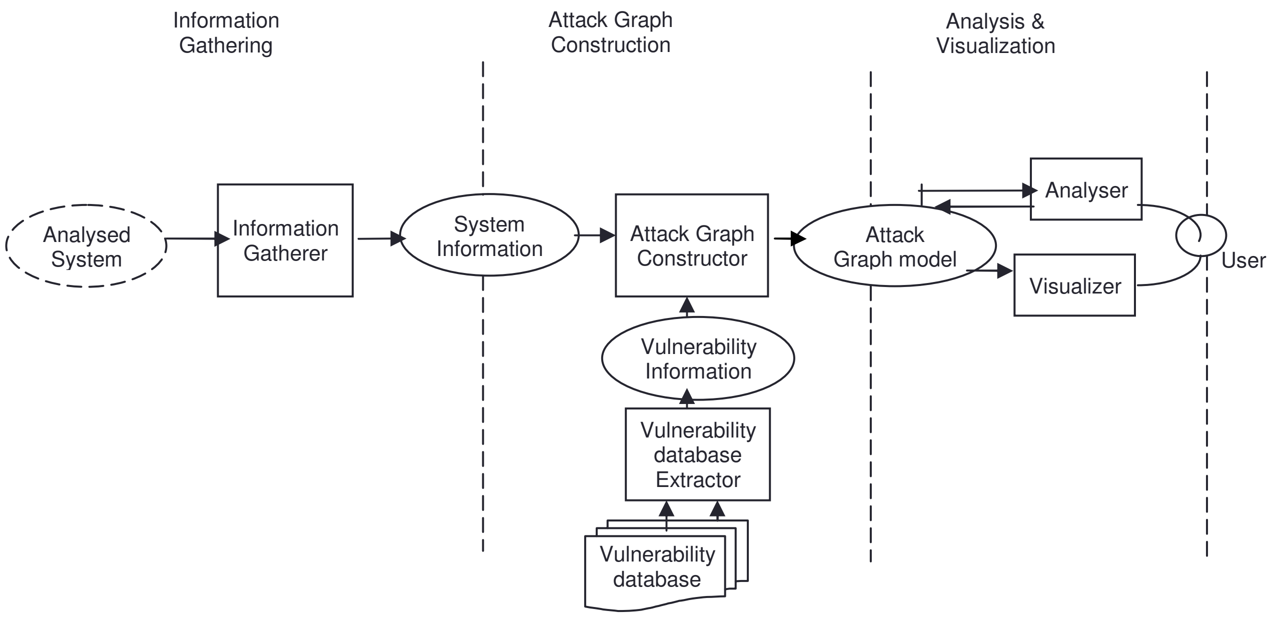


Figure 2: Vulnerability Analysis using Attack Graph

Attack graphs yield a number of advantages. Many of them are based on the ability to link single attacks together and form attack paths. Additional advantages result from the visualization possibilities facilitated by the graph data structure.

1.2.3 Data input – System information and Vulnerability information

A. Data Model for System Descriptions

Figure 3 shows the so-called System properties used to describe systems and networks [201]. System properties are characteristics and resources of a computer system. Each system property describes one specific attribute of such a system, whereas properties are related to one another as depict in Figure 3. For example, the installed version of an application can be a system property. An application's version is meaningless if it cannot be linked to a certain application. Properties and their relations may change over time due to modifications, such that an application may be upgraded to a newer version.

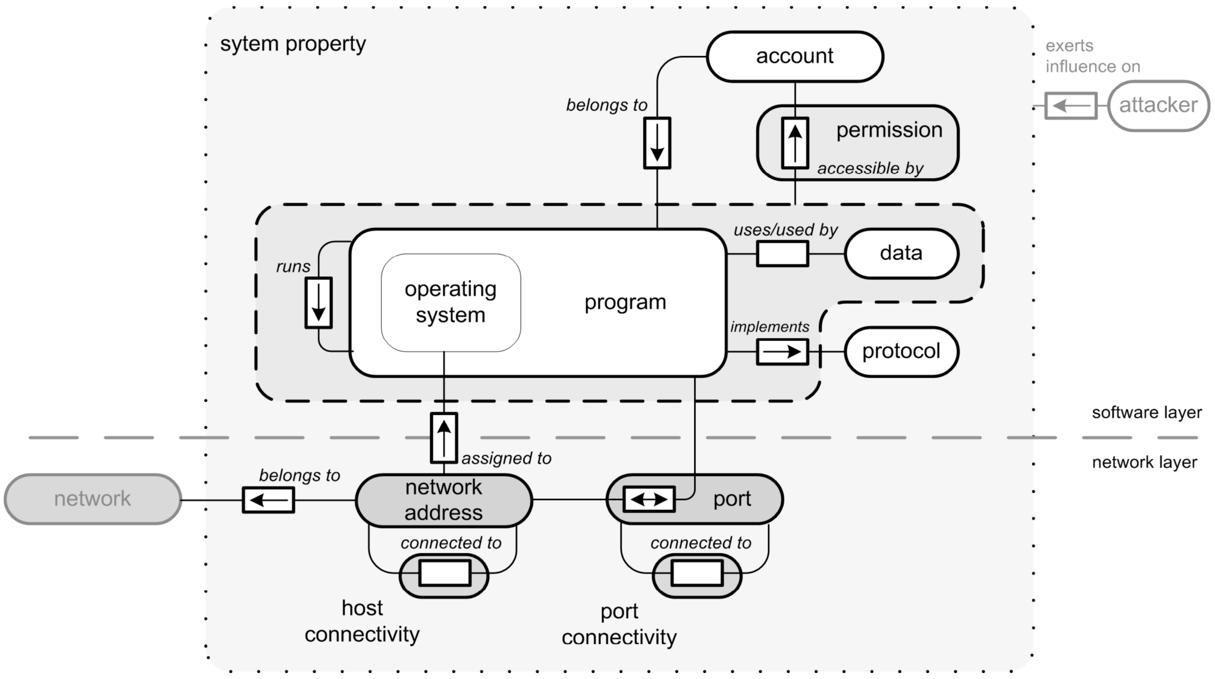


Figure 3: System Properties

System properties can be found in two layers, the network layer and the software layer. The network layer describes properties of interconnected computers, such as network addresses and port numbers. The software layer describes properties of software systems, such as programs, data, and account information. A network is a group of directly connected network addresses. A network address is an identifier of a host in a network. Directly connected means it is possible to reach from one host of a network to another host of the same network. Network addresses may have a number of open ports per address which are used by programs to communicate with other programs. Also covered are host as well as port connectivity, both are essential to capture which hosts and programs can be reached. Host connectivity is a Boolean value to describe whether one host can be reached from another host. This may be influenced by the network the corresponding hosts are in or by firewall rules, preventing certain hosts to connect to others. Port connectivity is a Boolean value to describe whether one port of a network address can be accessed from another port of a network address. Similar to host connectivity, this can be influenced by firewall rules or comparable system configuration tools.

B. Using available Vulnerability Databases
Vulnerability information is available from basically two types of sources. On the one hand, commercial or non-profit organizations act as vulnerability providers, such as Secunia security advisories [218] or the Open Source Vulnerability Database [179]. On the other hand, vulnerability information is described with standardization efforts, for example the Common Vulnerabilities and Exposures list (CVE) [167]. Based on the analysis in [200], we extract vulnerability information from the National Vulnerability Database (NVD) [172]. It provides most of the useful vulnerability information. It also has the advantage of making this data available in a well-defined XML format, which alleviates the amount of work to implement a parser. Another benefit of the NVD is the explicit inclusion of extensive CVSS information. This means no additional source must be parsed to extract this data. Additionally, the NVD refers to Open Vulnerability and Assessment Language (OVAL) descriptions that are detailed characterizations of the software configuration which is vulnerable. In [200], existing vulnerability databases are analyzed concerning their usability in attack graph construction. All these information will be more detailed in the Deliverable D3.WP4.

2. NETWORK MONITORING TECHNIQUES STATE OF THE ART

2.1 INTRODUCTION

2.1.1 Concept and objectives

Network monitoring is a laborious and demanding task that is vital for the network infrastructure. Service providers are constantly striving to keep the network operation stable, smooth and safe. If the network was vulnerable or under attack or down, even for a small period of time, service provider's ability to deliver secure and high-quality services would be questioned and could be compromised. Network administrators must be proactive rather than reactive, monitoring the network traffic and security and performance all the time, and verifying that security threats do not occur within the network perimeter.
The requirements of a network monitoring platform [2] can be summarized as follows:

- *High capturing performance*. The monitoring platform must be able to capture traffic at high traffic speeds and under high traffic volume.
- *Extensibility*. If new services are integrated on the network, it must be possible to deploy effortlessly new monitoring mechanisms for these specific services.
- *Scalability*. The monitoring platform must be able to handle the increase of traffic data as network links speeds grow and occasionally the number of devices in the network without performance degradation of the management system. Scalability can be achieved by reducing the traffic information collected using efficient packet capturing mechanisms and traffic pre-processing activities.
- *Real time functioning*. The monitoring platform must implement real time mechanisms in order to quickly detect network security/performance problems and allow the service provider to take actions in a timely manner.
- *Granularity*. The monitoring mechanisms must be able to track the security and performance of each service by capturing and analyzing the traffic of protocols used by this application.
- *Diversity*. The monitoring platform has to support the network's diversity as it contains many different kinds of network devices from multiple vendors, protocols stacks, and applications to provide services to the user.
- *Low cost*. The monitoring system should not use excessive amount of computing, storage, and communication resources so the cost of deploying and operating the monitoring infrastructure is low for the service providers.
- *Secure*: Do not add vulnerabilities to the network, or disturb network normal operation.

Network analysis (also known as traffic analysis or protocol analysis) is the process of capturing network traffic and inspecting it closely to determine what is really happening in the network [178]. Monitoring, which is also referred as passive testing, consists of observing the input and output events of an implementation in run-time. It should not disturb the natural run-time of a protocol or service that is why it is called passive testing. The record of the event observation is called an event trace. This trace is analyzed according to the specification in order to determine the conformance relation between the implementation and the specification. One should keep in mind that when an event trace conforms to the specification it does not mean that the whole implementation conforms to the specification. On the other hand, if a trace does not conform to the specification, then neither does the implementation.

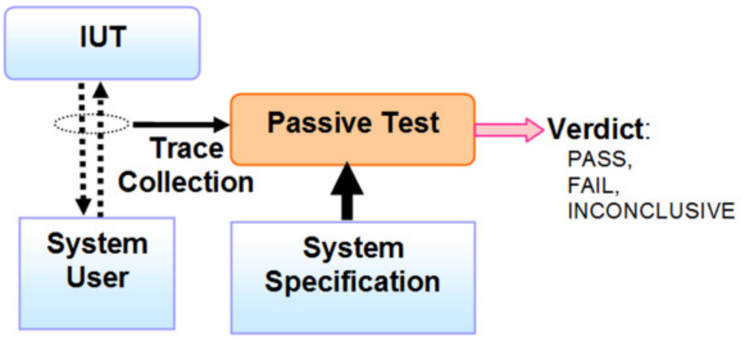


Figure 4: Monitoring techniques

Passive monitoring does not inject traffic in the network or modify the traffic that is being transmitted in the network. That is crucial because the injected messages may modify the environment, causing trouble or in worse cases the crash of the tested protocol or service. So, the passive monitoring approach seems to be the ideal means for doing a control directly on the implementation in natural run-time conditions. In addition in this approach the tests can be run during the whole protocol life time in contrast with the active testing test campaigns that are run on a limited duration. Figure 4 describes the monitoring or passive testing methodology. A trace analysis can produce a pass verdict if the trace is conforming to the system specification (or properties) and a fail verdict otherwise. The inconclusive verdict can be delivered if the trace is not long enough to allow a complete analysis.
The main objectives of monitoring are functional verification of the system under test, performance analysis, the verification of security properties and the detection of security vulnerabilities and attacks. These last two items related to security checking and threat detection techniques constitute one of the main goals of DIAMONDS project.

2.1.2 Monitoring architectures

Probe systems are becoming a fundamental tool [54] for networks and services operator for network monitoring and security analysis. Probes can be placed at any point in the network and so they provide greater flexibility above that provided by systems based on network element or IT resource measurements alone.
Basically, there are two types of probes: active and passive ones. The active ones introduce traffic in the network and send requests to service servers just as users do. They provide an end-to-end view. On the other hand, passive ones do not inject any traffic in the network and they sniff the different protocol units. They are able to provide a view of any part of the network and at any protocol level, and from both user and service perspectives:

- Passive service and network probes that gather all the sniffed traffic at main network segments are indispensable to seeing all users' traffic from core network in a non-intrusive way.
- Embedded passive agents inside end users' devices that show the edge end perspective.
- Active user probes that emulate end user activity as an end device are indispensable for end-to-end (e2e) vision.

Combining the information provided by these types of probes, which in general can be classified into passive and active probes, can offer a new vision of the telecommunication infrastructures.
The traffic collected through probes, due to the detail with which they are elaborated, provides a real vision of the behaviour of the network, its level of security and of the perception that users

have of the services (trust and quality of experience). Therefore, this architecture will allow to guarantee the service and to satisfy customer needs throughout the processed information.
On one hand, these systems allow improvement of the quality in terms of customer satisfaction and optimization of the service levels through the value chain. On the other hand, they mitigate the risks which the operators face guaranteeing the integrity of the network and the services, eliminating the risk of not developing innovating services on time, and adapting quickly to the changing circumstances of the market, and the scenarios of regulation without waiting for development of Resource Manager (Element Manager) Service enhancements.
Flexibility and multiprotocol capabilities are two basic characteristics for probes monitoring new generation networks (NGN, UMTS, IMS, etc.), and new services as IPTV, VoIP, mobile TV, triple play, etc. The development of these probe technologies must be focused on a criterion of scalability based on an open architecture that allows the monitoring of new emergent technologies.
Monitoring can be local when we rely on traffic captured from a single probe. In this case, we have a limited vision of the network. Distributed monitoring (based on distributed probes) can be beneficial to detect more complex attacks and security flows but the correlation between local traces can be a hard task that needs synchronization between traces.

2.2 MONITORING TECHNIQUES

2.2.1 Online and offline inspection

Passive monitoring can be performed with the assistance of any network sniffer. Although passive monitoring does not cause traffic overhead in the network as active testing does, it has some drawbacks. Since the huge amounts of collected data have to be processed as they are collected, sometimes this task can only be performed off-line due to processing performance issues. Moreover, the post-processing analysis can take a long time to complete.

2.2.2 Security properties and refinement

In this section, we present a state-of-the-art review of frameworks for specifying security properties and approaches for preserving security properties under refinement, i.e. the process of making an abstract specification more detailed.
It is useful to distinguish between two kinds of insecurity [156]. The first kind of insecurity allows an attacker to bypass normal access control to directly receive classified information. The Bell-LaPadula model [20] and the Biba model [24] are intended to prevent this kind of insecurity. E.g., for a system to be secure in the sense of Bell-LaPadula, every possible sequence of the system state transitions must result in a secure state; i.e., one in which no user has access to classified data unless he is authorized to have that access. The second kind of insecure system prevents direct unauthorized access, i.e., it is secure in the sense of Bell-LaPadula, but it allows for covert channels. A covert channel is an indirect communication path between users. For the second kind of insecure systems, it is often necessary for the attacker to have a Trojan horse program which is privileged to see classified data and can signal this information to the attacker. The prevention of this kind of insecurity is the motivation of secure information flow requirements.
Information flow security was first introduced in [234] as a generalization of the notion of non-interference [82][83] (which applies to deterministic systems) to nondeterministic systems. Given a system with one public (i.e., low level) and one confidential (i.e., high level) interface, non-interference requires that confidential inputs never affect public outputs of the system. If this property holds, we can conclude that no information flow is ever possible from high to low level. Information flow properties in general make use of a more general notion of low level observations and confidential behaviour, but the underlying idea is much the same, namely to prevent the low level observations from being influenced by confidential behaviour.

Many information flow properties have been proposed in the literature in various semantic models such as trace-based models [76][145][155][157][158][255], state-based automation models [82][165][176][249][253], and process algebraic models [74][175][202][213]. See [160] for a comparison of security properties across the semantic frameworks.

2.2.2.1 *Security frameworks*

Various security frameworks have been proposed in which security properties can be formulated and compared [10][28][75][145][157][158][253]. We give a brief review of these frameworks.
In [157][158] McLean considers specifications that are interpreted as sequences of states called traces. In McLean's framework, all security properties are closure conditions under so-called selective interleaving functions. A selective interleaving function takes two traces as input and produces the trace obtained by selecting state-objects from its arguments. According to [145], the expressiveness of McLean's framework is limited in that it cannot capture inductive definitions which are required for properties such as the perfect security property [253].
Zakinthinos and Lee [253] consider specifications expressed as traces of events. Their framework is based on so-called low level equivalence sets, i.e., the set of all traces containing the same low level observation. The framework provides little structure in defining security properties, and according to [145], the framework is not expressive enough to capture properties like separability and the perfect security property.
In [145], Mantel presents the so-called MAKS (Modular Assembly Kit for Security properties) framework based on traces of events. In Mantel's framework, each security property is specified as a conjunction of so-called basic security predicates. A basic security predicate is schema that is parameterized by a restriction and a closure condition and requires that for each trace t of a system that satisfies the restriction, there must be a trace u with the same low level observation as t and that satisfies the closure condition. The idea is that the presence of u prevents the low level user from asserting that high level behaviour has or has not occurred. Mantel's framework provides more structure than the framework of Zakinthions and Lee. At the same time it is sufficiently expressive to capture security properties such as the perfect security property.
Alur et. al. [10] considers specifications interpreted as traces of states and events. The framework is based on a so-called inferable property function that given a trace t, a property P, and an equivalence relation ≡ over traces, yields the knowledge of the observer about the property ≡ after trace t has been executed. They say that a property P is secret w.r.t. a given equivalence relation ≡, if for every trace t of a system, the low level observer cannot assert that the property holds or not holds when t is executed. The framework seems to be sufficiently general to capture the perfect security property, but the it is less modular than Mantel's framework.
The framework of Bossi et. al. [28] is defined in a process algebraic setting. It is based on a generalization of a so-called unwinding condition introduced in [83] to prove systems secure. Intuitively, the generalized unwinding condition states that if there is a high level transition from one state S to another state S′, then there should be a "simulated" transition from S to another state S″ which is low level equivalent to S′. If this holds, then the low level user cannot assert that the system has moved into state S′. Like the framework of Alur et. al., the framework of Bossi et. al. is quite general, but it seems to be less modular than Mantel's framework.
Focardi and Martinelli [75] present a uniform approach for the definition of security properties. Their model is based on the so-called Cryptographic Security Process Algebra (CryptoSPA) language. CryptoSPA is an extension of CCS [166] with primitives for manipulating messages. Focardi and Martinelli propose a general schema, called the Generalized Non Deducibility on Compositions (GNDC), for the definition of security properties. The main idea is that a system is secure iff for every possible environment, the composition of the system and the environment is secure. The authors show how the GNDC schema can be instantiated to obtain standard information flow properties as well as authentication and non-repudiation properties. The framework is therefore

quite general, but like the generalized unwinding condition, it seems less modular than Mantel's framework.

2.2.2.2 Secure information flow and refinement of under-specification

In design specifications, non-determinism often represents under-specification, i.e., design alternatives that are equivalent in the sense that it does not matter which one of them is implemented. Preserving information flow properties under refinement of under-specification has proven to be problematic. To our knowledge, Jacob [117] was the first to consider this problem. The notion of refinement he considers is based on the so-called safety ordering [99]. According to this notion, a specification S′ (interpreted as a set of traces) is a refinement of a specification S if S′ is a subset of S. This notion of refinement is known to preserve safety and liveness properties [9], but Jacob shows that this is not the case for security properties.

Since Jacob's initial investigation, a large number of papers have addressed the relationship of information flow security and refinement. These can be classified into two categories: those that propose conditions under which a given notion of refinement preserves standard security properties [10][28][29][96][146][207][208][209], and those that reformulate standard security properties in such a way that they are preserved under refinement, e.g., by closing the security properties under refinement [127][140][141][202].

In the following, we give a brief presentation of the above citations.

Mantel [146] considers the same notion of refinement as Jacob, i.e., trace-set inclusion. To overcome the refinement problem, he presents a collection of so-called refinement operators that characterize secure refinements. According to [96], the refinement operators may lead to concrete specifications that are practically hard to implement, because the changes in the refinement they induce are hard to predict and may not be easy to realize in an implementation.

Bossi et. al. [29] investigate security and refinement in a process algebraic setting. The security properties addressed are those that can be expressed as instances of the generalized unwinding condition. Refinement is defined such that a process E is a refinement of a process F if E is simulated by F. This is a stronger notion than the trace-set inclusion refinement considered by Mantel and Jacob, but it is not, in general, security preserving. Therefore, a general condition under which security is preserved is proposed. It is shown that some of the results of Mantel can be obtained as instances of this condition.

In [10], security properties are defined over traces of states and events. A notion of secrecy-preserving refinement is presented. The idea is that specification *Sc* is a secrecy preserving refinement of specification *Sa* if for each trace *tc* of *Sc*, there is an equivalent trace *ta* of *Sa* such that the low level observer can deduce less about the properties of interest when observing *Sc* execution *tc* than observing *Sa* executing *ta*.

Santen et. al. [96][207][208][209] consider CSP (Communicating Sequential Processes [99]) specifications extended with a probabilistic choice operator. The notion of refinement investigated is CSP refinement (trace refinement, failure refinement, and failure divergence refinement) modulo a so-called retrieve relation that maps concrete data to abstract data. A notion of confidentiality preserving refinement is defined. The underlying idea is similar to [10][29], i.e., that a low level observer must not gain more information at the concrete level than at the abstract level.

The papers discussed so far propose conditions under which refinements are secure. Another solution to the problem is to formulate security properties in such a way that they are preserved under refinement [127][140][141][202]. Lowe argues [140][141] that thxis idea not only ensures preservation under refinement, but that it also leads to more intuitive notions of security. In [141], Lowe therefore presents a new security property which is closed under refinement. Unlike Jacob or Mantel, he is able to close a property under refinement without making the property unreasonably strong because the notion of refinement investigated by Lowe is less liberal than the one considered by Jacob or Mantel. A similar line of reasoning is followed by Jürjens in [127], where a

secure information flow property is formalized such that each behaviour refinement to a deterministic system satisfies the property.
Roscoe [202] takes a more drastic approach. He argues that a system should only be considered secure if it is deterministic from a low level point of view. The idea is that the only way information can leak from the high level to the low level is via the process behaving differently towards the low level user depending on what the high level user has done. These kinds of security properties are closed under CSP notions of refinement. A disadvantage to this approach is that it imposes limitations on specifications. According to [146], requiring nondeterministic low level behaviour forbids common forms of parallelism for the low level and limits the possible abstractions in the abstract specifications.

2.2.2.3 Secure information flow and data refinement

The notion of data refinement relates an abstract data structure to a concrete data structure [59] as specified by a translation relation. Closing security properties under this kind of refinement is not a viable solution because all possible translation relations must be taken into consideration.
Graham-Cumming and Sanders [85] were to our knowledge the first to consider this problem. They define a refinement notion based on so-called downwards simulations and propose a condition under which a non-interference security property is preserved under this notion of refinement. The security property is defined in terms of a low level equivalence relation, and the condition demands that this relation be preserved for a refinement to be secure. Their notion of refinement is limited in the sense that only refinement of internal data is considered; in other words, they do not consider refinement of input and output data.
Santen et al. [96][207][208][209] discussed previously also addresses data refinement. That is, they allow refinements that rename data contained in events of traces. This notion of data refinement is quite limited because it does not allow one (abstract) event to be refined into a sequence of (concrete) events.
A more general notion of data-refinement of so-called configuration structures is considered by Hutter [104]. A configuration structure has sets of events as its states. The idea is that each state contains the events performed by the system in order to reach that state. A transition relation is therefore (implicitly) defined as the subset relation over states. Hutter defines so-called view refinement in which an event of a configuration structure can be replaced by another configuration structure representing the concrete behaviour of that event. This refinement is subject to certain conditions, e.g., that refinements of low level events cannot contain high level events that together ensure preservation of security.

2.2.3 Specification-based analysis

This methodology tries to map the system/network collected trace (composed of input/output sequence) to its formal specification. This formal specification can describe system/network functional behaviour but also some desired security aspects. In this section, we will present some research work that rely on system specification described using Extended Finite State Machine (EFSM). This methodology seeks for a transition (or a set of transitions) of the specification where the input/output couple matches and where the predicate on variables is true or cannot be evaluated (in the case of a trace that does not provide enough information about the system variables values). The values of variables are then deducted from the formal specification.
However, this methodology suffers from a possible loss of information. For example, assuming that *x* has previously been assessed at a value of 3, the Figure 5 shows that, in the absence of information about *y*, we have to choose two transitions leading to different values of *x* making it undefined.

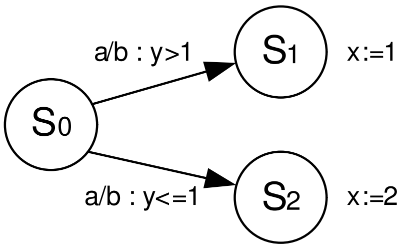


Figure 5: EFSM example

Considering these deficiencies, a more efficient algorithm for error detection was developed and presented in [133]. This algorithm is based on three aspects:

- Intervals to denote the values of variables, denoted *R(v) = [a,b]* for variable *v*.
- Assertions or predicates on variables, denoted *asrt*($\vec{x}$) on the vector $\vec{x}$ of variables.
- Candidate Configuration Sets (CCS) to formalize the analyzed environment of the system under test. A CSS is the triplet *(s, R* ($\vec{x}$)*, asrt*($\vec{x}$)*)* where *s* is the current state of the specification.

This algorithm tries to determine the values of the variables not using only a single value assignment but a set (in the form of an interval) of possible values for each variable. It refines as much as possible the intervals in which the variables take their values.
The intervals are a beginning of an answer to the information loss problem. Indeed, based on this concept, a variable is allowed to have more than one possible value. A variable *v* whose value is contained between two integers *a* and *b* will be defined by the interval *R(v)* so that: *R(v) = [a;b]*. In the particular case where *v* is a constant value *a*, we have: *R(v) = [a;a]*. We then say that this variable *v* is decided. Three operations on intervals are possible:

- The sum of two intervals. We have: [a;b]+[c;d] = [a+c;b+d]
- The subtraction of two intervals. We have: [a;b]-[c;d] = [a-d;b-c]
- The multiplication of an interval by an integer:
  - $w \times [a;b] = [w \times a ; w \times b]$ if $w \geq 0$
  - $w \times [a;b] = [w \times b ; w \times a]$ if $w \leq 0$

An assertion asrt($\vec{x}$) is a Boolean formula on the vector of variables $\vec{x}$ that must be true at the current state of the analysis. The assertions are used to record constraints on variables. These constraints can arise from transition predicates and actions. In this way, if a transition is fired then its predicate is added to the assertion as well as updates (i.e. actions) that contain undecided variables in the right member of the equality. For instance, the action $x_2 \leftarrow x_1+1$ updates x_2. In this case, every term of *asrt*($\vec{x}$) containing x_2 must be deleted and the term $x_2 \leftarrow x_1+1$ must be added to *asrt*($\vec{x}$). As soon as we discover the value of x_1 we deduct trivially the x_2 one.
A Candidate Configuration Set (CCS) is triplet *(s, R* ($\vec{x}$)*, asrt*($\vec{x}$)*)*, where:

- *s* is the current specification state,
- *R* ($\vec{x}$) is the set of constraints on the variables (i.e. intervals),
- *asrt*($\vec{x}$) is an assertion on $\vec{x}$.

The candidate configurations model the states where the system under test is. They also show the different constraints on variables. For instance, the following configuration *(S_1, R(x) = [1;5], (x<2))* means that the system is in the state S_1 and that the value of *x* is contained between 1 and 5 but inferior to 2.

The algorithm uses two lists Q_1 and Q_2, Q_1 being the set of current possible CCS and Q_2 the possible CCS of the previous step. From Q_1 and an event *e*, we must obtain the corresponding transitions. A transition *t* will be fired if there is a configuration in Q_1 whose constraints (the intervals of the variables and the assertion) are compatible with the predicate *p* of *t*.
In [8], the authors propose a new approach that can be considered as the opposite of the one presented earlier in [133]. Based on the fact that the end of a trace corresponds to a system state, we can already discover from this stage some information about the variables values if we look to the past of the trace.
The algorithm Backward-Checking operates in two stages. The first step is to go (back) in the trace *W* from its end to its beginning, mapping *W* to the specification machine. The aim is to reach all possible configurations *X* that can generate the trace *W*, i.e. all triplets $(s, R(\vec{x}), asrt(\vec{x}))$ candidate configurations set from which *W* could begin. The second step is to explore all possible paths from *X*, to verify that *W* is reachable from the initial configuration of the specification.

In fact, validating a trace *W* is based on finding a path *p* that connects a configuration *c* and an element of *X*. *p* validates *W*, if there exists a set of predicates and actions that can confirm the correction of the element of *X*. This approach results in a complexity that is at worst equal to the total parsing of the system specification, i.e. the full exploration of its graph accessibility.

2.2.4 Invariant-based analysis

Passive testing based on invariants has been inspired from runtime verification techniques provided by the verification community [136]. They focus on the application of verification techniques to a running system in order to check if an expected (or unexpected) property is observable or not through its interfaces. Several languages to express these properties have been proposed, most of them are based on temporal logics. The expressiveness of the checked properties is important. However, since invariants for passive analysis mainly come from testing studies, these invariants may also easily be applied through active testing techniques, which is not so obvious in runtime verification. Moreover, even if the expressiveness of invariants based on temporal logics is better, the timed invariants are difficult to model. Indeed, timed constraints such as *Before* and *After* are well studied by verification techniques; but timed intervals are still an issue.

Invariants based analysis follows these four steps:

- Step 1: Properties formulation. The relevant protocol properties to be tested are provided by the standards or by protocol experts.
- Step 2: Properties as invariants. Properties have to be formulated by means of invariants that express local properties; i.e. related to a local entity. Moreover, the properties are formally verified on the formal specification ensuring that they are correct with respect to the requirements.
- Step 3: Extraction of execution traces. In order to obtain such traces, POs are set up by means of a network sniffer installed on one component.
- Step 4: Test of the invariants on the traces. The traces are processed in order to obtain information concerning particular events as well as relevant data (e.g. token owner, source and destination address, origin of data to initialize a variable, etc.). During this process, the test of the expected properties is performed and a verdict is emitted (Pass, Fail or Inconclusive). An inconclusive verdict may be obtained if the trace does not contain enough information to allow a Pass or Fail verdict.

An invariant is defined as follows. Let M = (S, I, O, s_{in}, fnext, foutput) be an finite state machine (FSM) where S is a finite set of states, I a set of input actions, O a set of output actions, s_{in} an initial state, fnext : S × I → S is the transition function and foutput : S × I → O is the output function [135]. Formally, a sequence *Inv* is an invariant for M defined by the following EBNF if the conditions hold:

1- $Inv ::= i/\overline{O} \mid *, Inv \mid i/o; Inv$

2- *Inv* is verified on M where $i \in I \cup \{\theta\}$, $o \in O \cup \{\theta\}$, and $\overline{O} \subset O$

Intuitively, a sequence such as {i1/o1, …, i_{n-1}/o_{n-1}, in/on} is an invariant for M if each time in/on is observed, then the trace i1/o1,…, i_{n-1}/o_{n-1} occurs before.

Invariants may be used to express properties where the occurrence of an event must be necessarily preceded by a sequence of events. In addition to sequences of input and output symbols, the wild-card characters 'θ' and '∗' are allowed, where 'θ' represents any single symbol and '∗' represents any sequence of symbols. Invariants could be automatically generated from the model, but the complexity of the algorithms still increases with the length of the property [42].

The invariants can also be checked both on the specification and on the IUT (Implementation Under Test), allowing determining whether the property is correct according to the formal model and that the IUT behaviour is as defined by the standard. An adaptation of a classical string pattern matching strategy [130] may be performed to match the trace and the invariants. The complexity of the resulting algorithm is in the worst case in O(m.n) (n being the invariant length and m the trace length), in particular because we must check all occurrences of the pattern in the trace making it difficult to optimize the complexity to O(m). Nevertheless, since the invariant length is often (not to say "always") much smaller than the extracted trace, the complexity is almost linear with respect to the trace length. The algorithms and their complexity analysis are detailed in [8] for the specification/invariants checking and in [19] for the implementation/invariants checking. Interesting experiments based on this kind of invariants can be found in [44][45].

Timed Invariants

Later work [14] included the possibility of adding time constraints as properties that traces extracted from the IUT must hold. For example, in a wireless network, a package usually contains some neighbours addresses, link status and types, which raise several interesting timing constraints. Only the messages that are recorded from the environment are taken into account. These messages are generated for different purposes (mainly control and exchange). The set of all messages is denoted by M. A message starting with "?" is said to be an input message while messages starting with "!" are called output messages.

Passive testing makes use of the exchanged messages in order to detect faults. Traces extracted from the IUT do not only represent the value of the messages, but also include time information that denotes the amount of time elapsed between consecutive messages. The set of traces is denoted by L.

Definition: We say that $p = [p_1, p_2]$ is a time interval if $p_1 \in \mathbb{R}_+$, $p_2 \in \mathbb{R}_+ \cup \{\infty\}$, and $p_1 \leq p_2$. We note IR for the set of time intervals. We assume that for all $t \in \mathbb{R}_+$ we have $t < \infty$ and $t + \infty = \infty$.

We extend the above mentioned EBNF and we say that the sequence ψ is a *simple timed invariant* if ψ is defined according to the following EBNF:

$$\psi ::= m/p,\ \psi \mid * /p,\ \psi' \mid m' \triangleright_p A,\ q$$

$$\psi' ::= m'/p,\ \psi \mid m' \triangleright_p A,\ q$$

where $p, q \in IR$, $m' \in I \cup O$, $m \in I \cup O \cup \{\theta\}$, and $\subseteq I \cup O$ (still considering the FSM M above defined)

Intuitively, this EBNF expresses that an invariant is a sequence of symbols where each component, excluding the last one, is either:

- a pair m/p, with m being a message or the wild-message θ, and p being a time interval,
- or an expression $*$/p.

Finally, the last component of an invariant corresponding to the expression $m' \triangleright_p A, q$ is a message associated with a time interval followed by a set of messages and another time interval. This last interval is used to control the sum of time values associated to all the actions performed during matching the invariant [13].

Another kind of invariants called *continuous timed invariants* has been defined. The idea is to provide an easy representation that allows filtering the messages that we do not need to check, making them non-observable ones.

Definition: We say that the sequence ψ is a *continuous timed invariant* if ψ is defined according to the following EBNF:

$$\psi ::= m/p, \psi \mid * /p, \psi' \mid m' \triangleright\triangleright_p A, q$$
$$\psi' ::= m'/p, \psi \mid m' \triangleright\triangleright_p A, q$$

Let us describe the differences between the two classes of invariants. Consider the two following invariants:

$\boldsymbol{\psi_1 = m_f \triangleright_p A, q}$ and $\boldsymbol{\psi_2 = m_f \triangleright\triangleright_p A, q}$.

The meaning of ψ_1 is that if we see in the trace a message that matches m_f then it will be followed by an amount of time in p and with a message matching one of the elements in A. The meaning of ψ_2 is that if we observe the message m_f then we will observe any sequence of pairs message-time before having a message that matches a message in A. The amount of time that this allowed sequence can spend has to belong to p. Regarding this new notion, let us comment that although sometimes it might seem that a timed invariant and a continuous timed invariant reflect the same property, there always exist some differences.

Example: Let us consider the following two timed invariants

$$\psi_1 = m' \triangleright_{[0,0]} \{m_1, m_2\}, [0, 2]$$
$$\psi_2 = m' \triangleright\triangleright_{[0,0]} \{m_1, m_2\}, [0, 2]$$

and let $\sigma = (m, 4, m', 0, m'_3, 0, m_2)$ a trace. σ is correct with respect to ψ_2 but not with respect to ψ_1.

2.2.5 Signature-based analysis

Generally, Intrusion Detection Systems (IDS) use either a statistical anomaly analysis technique that determines normal network activity like what sort of bandwidth is generally used, what protocols are used, what ports and devices generally connect to each other- and alert the administrator or user when traffic is detected which is anomalous (not normal), or a signature based analysis that monitors packets in the Network and compares with preconfigured and predetermined attack patterns known as signatures. The issue is that there will be a lag between the time where the new threat discovered and the time where the signature is applied in IDS for detecting the threat. During this lag time, the IDS will be unable to identify the threat [154].

2.2.6 SNMP based monitoring

SNMP [197] is an application layer protocol that is part of the TCP/IP protocol suite, which facilitates the exchange of management information between network devices. SNMP allows

network administrators to manage network performance, find and solve network problems (functional and security), and plan for network growth. It collects traffic statistics through passive sensors that are implemented on routers, switches, or hosts. SNMP version 1 (SNMPv1) is the initial implementation of the SNMP protocol. It is described in RFC 1157. SNMPv2, described in RFC 1441 and RFC 1452, is built upon SNMPv1 and includes improvements in the areas of performance, security, confidentiality, and manager-to-manager communications. Although SNMPv3 makes no changes to the protocol aside from the addition of cryptographic security, its developers have managed to make it look different from previous versions by introducing new textual conventions, concepts, and terminology. Standardization of SNMPv3 is pending.
An SNMP-managed network consists of three key components: managed devices, agents, and network-management systems (NMSs). These are shown in Figure 6.

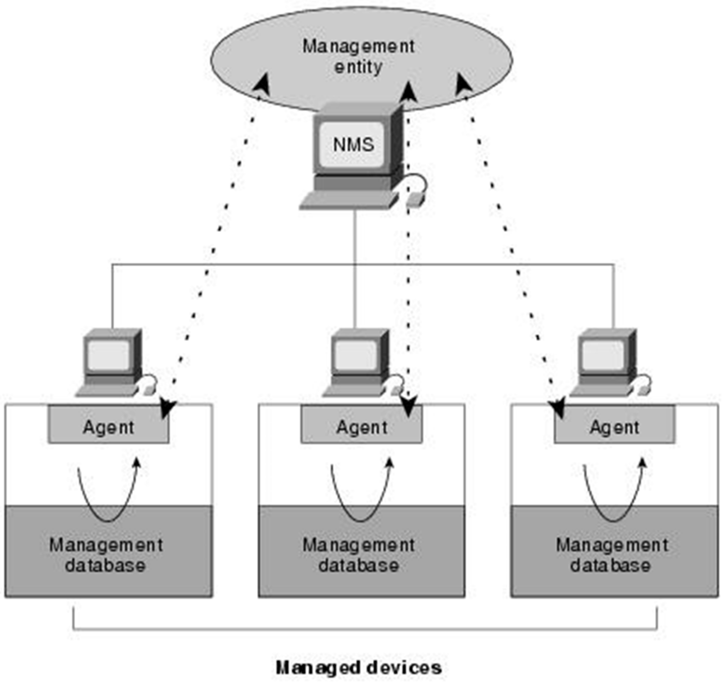


Figure 6: SNMP components

The managed devices contain the SNMP agent and can consist of routers, servers, switches, bridges, hubs, hosts, or printers. They are responsible for collecting and storing management information and making it available to the NMSs using SNMP. An agent is a network-management software module that resides in a managed device. The agent has local knowledge of management information and translates this information into a form compatible with SNMP. The NMSs execute applications that monitor and control the managed devices. Processing and memory resources required for network management are provided by the NMSs. At least one NMS must exist on any managed network. There are four basic commands used by a SNMP NMS to monitor and control the managed devices: *read*, *write*, *trap*, and *traversal* operations. The *read* command examines the variables that are maintained by the managed devices. The *write* command changes the values of the variables stored by the managed devices. The *trap* command is used by the managed devices to report the occurrence of certain events to the NMS. *Traversal* operations are used by the NMS to determine which variables a managed device supports and to sequentially collect information in variable tables, such as a routing table.
SNMP is a simple request/response protocol. The NMS issues a request and managed devices use their SNMP agents to retrieve the needed information and then return the response. SNMP

uses four protocol operations in order to operate: *Get*, *GetNext*, *Set*, and *Trap*. The *Get* operation is used by the NMS to retrieve the value of one or more object instances from an agent. The SNMPv1 request message consists of a message header and a PDU. The message PDU contains the information that is necessary to complete a request that will either retrieve information from the agent or set a value in the agent. If the agent responding to the *Get* operation cannot provide values for all the object instances in a list, it does not provide any values. The *GetNext* command is used to retrieve the value of the next object instance in a table or a list within an agent. The *Set* operation is used by the NMS to set the values of object instances within an agent. When an agent needs to inform the NMS of a significant event, it will use the *Trap* operation.
As discussed, SNMP is an application layer protocol that uses passive sensors to help administrators monitor network traffic and security and performance. Even though SNMP can be a useful tool for network administrators, it lacks any authentication capabilities, which results in vulnerability to a variety of security threats.

2.2.7 Deep Packet Inspection

Network administrators have to constantly rely on the data flow to determine the networks properties. Routers and switches throughout the network ceaselessly forward traffic data belonging to flows, but these data can be become extremely voluminous exceeding hundreds of gigabytes per hour. Therefore, it is of great importance having a flow collector to aggregate the data in real time and provide the network administrator with the desired flow information in a reasonable time. For example, the network administrator may want to know which users are connected to a server and amount of data they are transferring. In this case, the administrator would have to use a flow collector that contains the server destination IP address, and an associated packet count or byte count. The output of the flow collector could also be sorted to show which users are sending or receiving the most amounts of data.
Deep Packet Inspection (DPI) technique is an advanced method of packet filtering that functions at the application layer of the OSI reference model. The use of DPI makes it possible to find, identify, classify packets with specific data or code payloads that conventional packet filtering, which examines only packet headers, cannot detect. Deep Flow Inspection (DFI), which is an evolution of DPI, is a way to identify and classify traffic flows in a network.
Flow information is the data traffic statistics summarized by routers and exported in standard industry formats such as NetFlow [51], J-Flow [121], and sFlow [222]. A flow is typically defined as a unidirectional sequence of packets that have the same destination and source address, transport level information, ToS field bits, and protocol information. Network monitoring technologies, like NetFlow, JFlow, and sFlow, enable routers and switches to collect information about flows passing through them and then export these flow records to a central collector, where they are used for many purposes such as network traffic accounting, usage-based network billing, network monitoring, network planning, traffic analysis, data mining, and security and DoS monitoring capabilities. A flow record typically contains the source and destination IP addresses, source and destination port numbers, protocol field, ToS byte, next-hop router IP address, input and output interface numbers on the router, packet count, byte count, and start time and end time of the flow.

2.2.8 Machine learning

In general terms, intrusive behaviour can be considered as any behaviour that deviates from normal expected use of the system. Intrusion detection techniques can be divided into signature-based and anomaly-based. In signature-based schemes given patterns are searched for, limiting the detection to known attacks. In anomaly-based schemes the goal is to detect behaviour that is deemed abnormal. A big difference in these techniques is that one will detect attacks that are security risks while the other will detect deviations from normal behaviour that need to be analysed

to determine if they are security risks, potential vulnerabilities, functional errors or merely false positives.
One technique used in NID is to detect anomalies with respect to normal behaviour using machine learning techniques. In [225] the authors explain that this technique has not been adopted in operational settings due to a number of reasons presented in their paper:

- Machine-learning algorithms are better adapted to "finding similarities than identifying activity that does not belong there", e.g., outlier detection.
- Assuming that it is a closed system does not hold, e.g. "specifying only positive examples and adopting a standing assumption that the rest are negative" [250].
- Classification, such as spam detection, successfully uses ML techniques, better suited in "finding variations of known attacks".
- Semantic gap between "finding abnormal activity and attacks".
- Diversity of network traffic makes it "unpredictable over short time intervals".
- Lack of "public datasets for assessing anomaly detection systems".
- Attackers adjust "their activity to avoid detection".
- "High cost of classification errors".

The authors give several suggestions on how research efforts should be oriented to find solutions to some of these problems: understand better the system, reduce the scope, reduce costs in eliminating false positives, and improve evaluation techniques...

In [81] the authors analyse different techniques that have been used for intrusion detection. They separate these techniques into:

- statistical-based, where the behaviour is represented from a random viewpoint:
  - techniques used are: univariate [62], multivariate [138], time series model [63];
- knowledge-based, where the behaviour is derived from available system data:
  - techniques used are: finite state machines [67], description languages, expert systems;
- machine learning-based, where models are constructed to allow categorizing the observed behaviour:
  - Bayesian networks, Markov models, neural models, fuzzy logic, genetic algorithms and clustering & outliner detection.

The authors consider this last technique as the most promising and identify its main drawback to be the high amount of resources required.
Machine learning includes advanced statistical methods for regression and classification that are useful in intrusion and anomaly detection. The goal of regression is to model the interaction and relation of different variables using a mathematical equation. The goal of classification is identifying to which predefined group a new observation belongs to.
Following is a very brief presentation of the different main ML methods that can be used to implement regression and classification:

Support Vector Machines (SVM)
This technique constructs non-linear decision boundaries to perform classification [161]. SVMs are used to detect and exploit complex patterns in data by clustering, classifying and ranking the data. They are learning machines that are used to perform binary classifications and regression estimations. They commonly use kernel based methods to apply linear classification techniques to non-linear classification problems. There are a number of types of SVM such as linear, polynomial, RBF, sigmoid etc.

Radial basis functions (RBF)
RBF is a function which has built into it a distance criterion with respect to a centre. Such functions can be used very efficiently for interpolation and for smoothing of data. Radial basis functions have been applied in the area of neural networks where they are used as a replacement for the sigmoidal transfer function. For instance in [190], it is used to detect abnormal sequences of system calls.

Bayesian networks
This technique is used to combine detection with statistical schemes [95] but results are similar to threshold-based systems [131] and deviations from the assumed behaviour lead to detection errors. Naïve Bayes based on Bayes conditional probability rule is used for performing classification tasks. Naïve Bayes assumes the predictors are statistically independent which makes it an effective classification tool that is easy to interpret [181].

Markov models
A set of interconnected states with probabilistic transitions define the behaviour model. The probabilities are estimated during a training session containing normal behaviour. This technique has been used in host IDS for system calls [252], packet inspection [143][68]. The results depend highly on what is considered normal behaviour. The simplest Markov model is the Markov chain. It models the state of a system with a random variable that changes through time. In this context, the Markov property suggests that the distribution for this variable depends only on the distribution of the previous state.
A Markov decision process is a Markov chain in which state transitions depend on the current state and an action vector that is applied to the system. Typically, a Markov decision process is used to compute a policy of actions that will maximize some utility with respect to expected rewards. In [122] the authors use this technique to discover anomalous activity. A hidden Markov model is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobserved (e.g. hidden) states. In [80] it is used for measuring behavioural distances.

Neural networks
This technique simulates the neurons and synapses of a brain and adapts well to changing environments. Neural networks are used when the exact nature of the relationship between inputs and output is not known, since they learn the relationship through training (supervised or unsupervised). It has been used for user profiling [77], predicting the next command [60] and detecting intrusive behaviour [40]. It does not provide the reasons why a particular detection decision has been made.

Fuzzy logic
This technique uses approximations rather than precise logic. It is useful in anomaly detection with normal behaviour defined by interval limits [34][64]. It has been found useful in detection port scans and probes.

Genetic algorithms
Generic algorithms allow deriving classifications tools [137] or selecting appropriate parameters for the detection process [34]. It allows using search heuristics that result in a solution with no prior knowledge of the system.

Clustering and outlier detection
Clustering allows grouping information according to similarities and distance measures. A representative point serves as reference for each cluster and then new information is classified according to their distance from it [185]. Points not belonging to any cluster are outliers and represent anomalies. This technique is used by [17] and [221]. K-nearest neighbour based on Euclidean distances can be used to determine membership to a cluster [138]; other techniques can also be used, such as Mahalanobis distances [118] that take into account the correlations of the data set obtaining scale-invariance. The problem with these techniques is defining the reference points and determining if the detected outliers are actually anomalies.

k-nearest neighbours (KNN)
KNN belongs to the class of pattern recognition statistical methods. The method does not impose a priori any assumptions about the distribution from which the modelling sample is drawn. It involves a training set with both positive and negative values. A new sample is classified by calculating the distance to the nearest neighbouring training case. The sign of that point will determine the classification of the sample. In the k-nearest neighbour classifier, the k nearest points are considered and the sign of the majority is used to classify the sample. An example can be found in [138].

2.2.9 Integration with other techniques

2.2.9.1 Active Monitoring

Active monitoring introduces active probes on the network to inject network and collect measurements and analysis security traffic between at least two endpoints in the network.
The problem of active monitoring is that introducing probes on the network can produce interference in the normal traffic of the network. Since the traffic created by active probes is often treated differently than normal traffic as well, the validity of the information provided from these probes is questioned. Active monitoring is very rarely implemented as a stand-alone method of monitoring since a good portion of overhead is introduced in the network traffic. On the other hand, passive monitoring does not introduce traffic overhead in the network.
After analyzing the active and passive monitoring techniques, we could deduce that a combination of active and passive monitoring would use the best aspects of both passive and active monitoring environments.

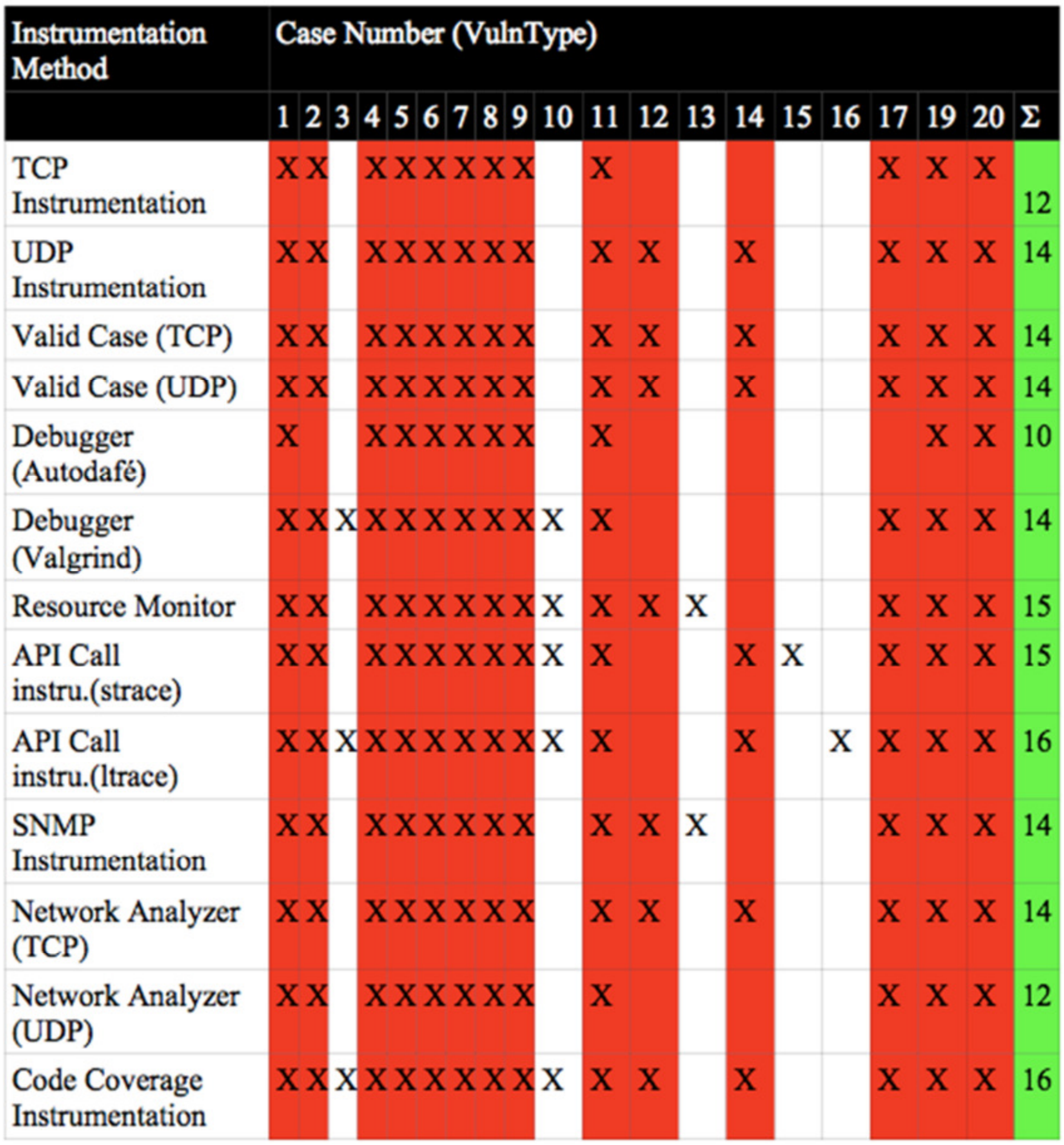
2.2.9.2 Fuzz testing and monitoring

This section is based on a thesis study [241] of instrumentation techniques used in fuzzing and robustness testing of software for security flaws.
Software testing can be done in three ways: white box testing, black box testing and gray box testing. White box testing of software is predicated on close examination of procedural detail. Logical path through the software is tested by providing test cases that exercise specific sets of conditions and/or loops. Black box testing involves test cases that are conducted at the software interface. A black box test examines some aspect of a system with little regard for internal logical structure of the software. Gray box testing is a combination of white and black box testing [188][235].

Fuzzing[1] is a form of black box testing because the tester does not need to know how the target software works internally. Fuzzing is defined as a method for discovering faults in software by injecting unpredicted input and monitoring for failures. It is typically an automated or semi-automated process that involves repeatedly manipulating and supplying data to target software for processing [235].
It is important to be able to determine whether fuzzers have successfully caused some problems. The result of the fault might not be obvious and it can be difficult to detect the results outside the target system. This subject has not been emphasized in currently available fuzzers [235].
Usually in software testing it is quite obvious to make the verdict whether a test case has failed or not. When designing a test case, there is implicit knowledge of what the verdict must be in order to pass the test case. If the result is something else, the test case has failed. In fuzzing, it is not known beforehand what a test case can cause to its target. The result can be anything from an error message to a total system crash.
Instrumentation can be divided into two major categories: out-of-band instrumentation and in-band instrumentation. Out-of-band instrumentation includes debuggers, resource monitoring or custom made tools. In-band instrumentation means that implementation is monitored through the same interface as the fuzzer delivers the test cases [180].
When doing software testing, one half of the process is generating test cases and inputting them into a SUT. The other half is determining how the SUT reacts on these inputs [235]. This is called monitoring or instrumentation [109]. The type of instrumentation depends on the SUT. Instrumentation for a Windows program is totally different than for a WLAN (Wireless Local Are Network) router [238]. On computer, it is relatively easy to install a debugger or another out-of-band instrumentation method, but on a WLAN router it is easiest to use in-band instrumentation methods [180].
Instrumentation techniques:

- TCP Connection: The TCP (Transmission Control Protocol) connection instrumentation method is most common in free and open source fuzzers. This instrumentation method is a very simple one. It checks if the TCP connection is lost, and if it is, the SUT has crashed.
- UDP Connection: The basic idea of the UDP connection instrumentation is the same as in the TCP connection instrumentation. Whenever the host receives a UDP datagram to a UDP port which has no process bound in, system should send ICMP (Internet Control Message Protocol) messages “destination unreachable” and “port unreachable”.
- Valid Case Instrumentation: A step forward from the UDP connection and the TCP connection instrumentation is the valid case instrumentation. Here the idea is that after each test case, one valid case, which has only valid input, is sent. If an answer is received for this valid case, the SUT is still operating, and if not, then the last test case has caused a crash [180].
- Debugger: Instrumentation with a debugger is totally different than with previous instrumentation methods. Instead of checking the SUT via the same channel as the fuzzed data goes in, a debugger is attached directly to the process which is under fuzzing.

[1] In security testing “robustness testing” has been used as a synonym for fuzzing since about 1991. In this section, we will discuss another meaning for that term. References:
* http://home.mit.bme.hu/~micskeiz/pages/robustness_testing.html
* http://en.wikipedia.org/wiki/Robustness_testing
* http://wiki.answers.com/Q/What_is_robustness_testing_in_software_engineering
* http://www.iist.unu.edu/www/docs/techreports/reports/report423.pdf
* http://www.slideshare.net/timmenzies/robustness-testing



Instrumentation Method	Case Number (VulnType)																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20	Σ
TCP Instrumentation	X	X		X	X	X	X	X	X		X						X	X	X	12
UDP Instrumentation	X	X		X	X	X	X	X	X		X	X		X			X	X	X	14
Valid Case (TCP)	X	X		X	X	X	X	X	X		X	X		X			X	X	X	14
Valid Case (UDP)	X	X		X	X	X	X	X	X		X	X		X			X	X	X	14
Debugger (Autodafé)	X			X	X	X	X	X	X		X							X	X	10
Debugger (Valgrind)	X	X	X	X	X	X	X	X	X	X	X						X	X	X	14
Resource Monitor	X	X		X	X	X	X	X	X	X	X	X	X				X	X	X	15
API Call instru.(strace)	X	X		X	X	X	X	X	X	X	X			X	X		X	X	X	15
API Call instru.(ltrace)	X	X	X	X	X	X	X	X	X	X	X			X		X	X	X	X	16
SNMP Instrumentation	X	X		X	X	X	X	X	X		X	X	X				X	X	X	14
Network Analyzer (TCP)	X	X		X	X	X	X	X	X		X	X		X			X	X	X	14
Network Analyzer (UDP)	X	X		X	X	X	X	X	X		X						X	X	X	12
Code Coverage Instrumentation	X	X	X	X	X	X	X	X	X	X	X	X		X			X	X	X	16

Figure 7: Instrumentation methods results on the Linux OS

- Resource Monitor: The idea behind resource monitoring is observing the system resources. Resources which can be monitored include, for instance, CPU load, memory usage, network load, process information, files used, and registry usage. These resources are available for monitoring throughout the operating system.
- API Call Instrumentation: The Application Programming Interface (API) is an interface between application and operating systems. These API calls can be traced by using special tools. In Unix systems these tools are called ltrace and strace.
- SNMP Instrumentation: SNMP stands for Simple Network Management Protocol. One purpose of SNMP is to monitor network elements [197]. So with the help of SNMP, it is possible to get some information about the system where the IUT is running. It is required that the system supports SNMP and that SNMP requires agents that support showing system resources. With SNMP, it is possible to see the current CPU load or currently available free memory [238].

- Network Analyzers: Network analyzers are useful tools when used together with network fuzzers. Network analyzers allow viewing the factual view of the network traffic. This feature is undoubtedly useful when fuzzing via network. Network analyzers, such as Wireshark [52], can also parse many network protocols and show possible returning error codes in packets. Network analyzers can also be used for detecting possible ICMP messages “host not reachable” or “port not reachable”.
- Code Coverage: Code coverage refers to measurement in software testing. It reveals what percentage of the source code of functions has been tested [199].

The results reveal that none of the instrumentation methods can find all failures in a target system. It would be wise to use more than just one instrumentation method. For instance, the valid case instrumentation could be used for detecting the Denial of Service cases and then either the API call instrumentation or the code coverage instrumentation for the rest vulnerabilities. The latter two instrumentation methods found the most failures in the Vulnerability Server. By the code coverage instrumentation it can also be seen how much of the software has been tested.

2.2.9.3 Fault Injection Approaches and monitoring techniques for robustness testing

Robustness is a specialized dependability attribute, characterizing a system reaction with respect to external faults. Accordingly, robustness testing involves testing a system in the presence of faults or stressful environmental conditions. Fault injection techniques and passive testing approaches for testing system robustness have also been studied.
Fault injection [22] consists of introducing deliberate errors in a system and observing its behaviour. This technique has been widely used for robustness testing because it allows one to evaluate the behaviour of a given system when running in a hostile environment.

1. The Hoare triple logic

Hoare logic [100] is a formal system which provides a set of logical rules based on mathematical logic. Its central feature is the Hoare Triple which describes how an execution of a set of actions changes the state of some variables. A Hoare triple is of the form {P}C{Q} where C is a program (a set of actions) and P and Q are assertions expressed in a first-order logic. Triple has the following meaning: If C is executed in a state satisfying preconditions P and if C terminates then the final state satisfies post condition Q.
The fault injection operation is defined as Hoare triple as follows.

Definition: (Injection operation) an injection operation is a Hoare triple {P}C{Q} where:

- P specifies a precondition on the intercepted message (its state before the execution of the injection operation);
- C denotes the operation itself (identified by its name and eventually a set of parameters);
- And Q is a post condition which states the effect of the operation execution on the intercepted message.

A communication message can be considered as a finite set of elements. Each element describes a part or a field of this message. Therefore, we can specify formally a communication message as a finite collection (a set where replicates are permitted) of elements S = {elt1,…,eltn}. We specify also the set of all injection operations executed during an injection experiment as a finite set of injection rules R such as each injection rule $r \in R$ specifies a Hoare triple describing an injection operation applied on an intercepted message, as follows: *{P(S)} OperationName(param$_1$,…,param$_n$) {Q(S)}*

Specification examples

Let us refer to the intercepted messages as a set of elements S. Each example describes a possible injection operation and provides its corresponding Hoare triple.

a) Operation delete

The first operation is used to delete intercepted messages. It is expressed by a Hoare triple as follows.

{¬S.isEmpty()} Delete(S) {new(S).isEmpty()}

Deletion of one message element can also be expressed as:

{S.has(elt)} Delete(S, elt) {new(S).equals(S.remove(elt))}

b) Operation Insert

{true} Insert(S,elt) {S.equals(new(S).remove(elt))}

The injection operation inserts extra data in the captured message. It can be either a malicious element or just a huge bloc of insignificant data in order to disturb the communication.

2. Real Time patterns

We consider the XCTL features [91] for specification and verification of complex real time properties. By using XCTL, we can specify both simple and correlated time constraints and the use of a single global time variable makes the specification easier. So, the algorithm inputs are a set of requirements specified as XCTL formulas and an event trace.

In the following we introduce few the periodicity and response patterns specified in XCTL for expressing the robustness properties.

a) Periodicity

The first pattern relates to events that must be hold periodically to prevent eventual security/safety issues. Given a proposition P, we can specify the periodic occurrence of P by the following XCTL formula.

$\Box((P \wedge T = x) \rightarrow \Diamond(P \wedge T = x + c))$

where the constant c represents the period duration. An example of this property can be illustrated by a system which sends periodically a liveness message to inform administrators about eventual crashes.

b) Response

This pattern is usually used to specify a simple request/response paradigm. Given two propositions P and Q, the following XCTL formula specifies that each occurrence of P must be followed by Q within c time units.

$\Box((P \wedge T = x) \rightarrow \Diamond (Q \wedge T \sim x + c))$

where $\sim \in \{<, \leq, >, \geq, =\}$, For illustration, we can specify for example that a connection establishment must not exceed 5 seconds.

$\Box((\text{ConnectReq} \wedge T = x) \rightarrow \Diamond(\text{ConnectResp} \wedge T \leq x + 5))$

3. Robustness testing methodology

A robustness testing process can be defined as an hybrid approach, faults injection plus passive testing, which involves three main stages:

a) First, faults are injected by applying some PO at the fault injector core to collect an execution trace during the injection process or we can put the PO at the fault injector interface (to collect input/output messages). But, in this case we can just verify the pre/post conditions independently of the executed operations which is not conform to Hoare logic), while the SUT is running and execution traces of both the fault injector and the SUT are collected.

b) Second, the injection process is verified. The set of injection operations that we want to inject as a set of Hoare triples is formalized. The execution trace of the fault injector is verified against the formal specification of the injected faults and a conformance verdict is issued. This conforms whether the injection was rightly performed.
c) Finally, the robustness requirements are verified. XCTL is used as a mathematical formalism for modelling the robustness properties. We rely on passive testing to issue a verdict about the conformance of the collected SUT's execution trace with respect to the provided formal specification of robustness requirements.

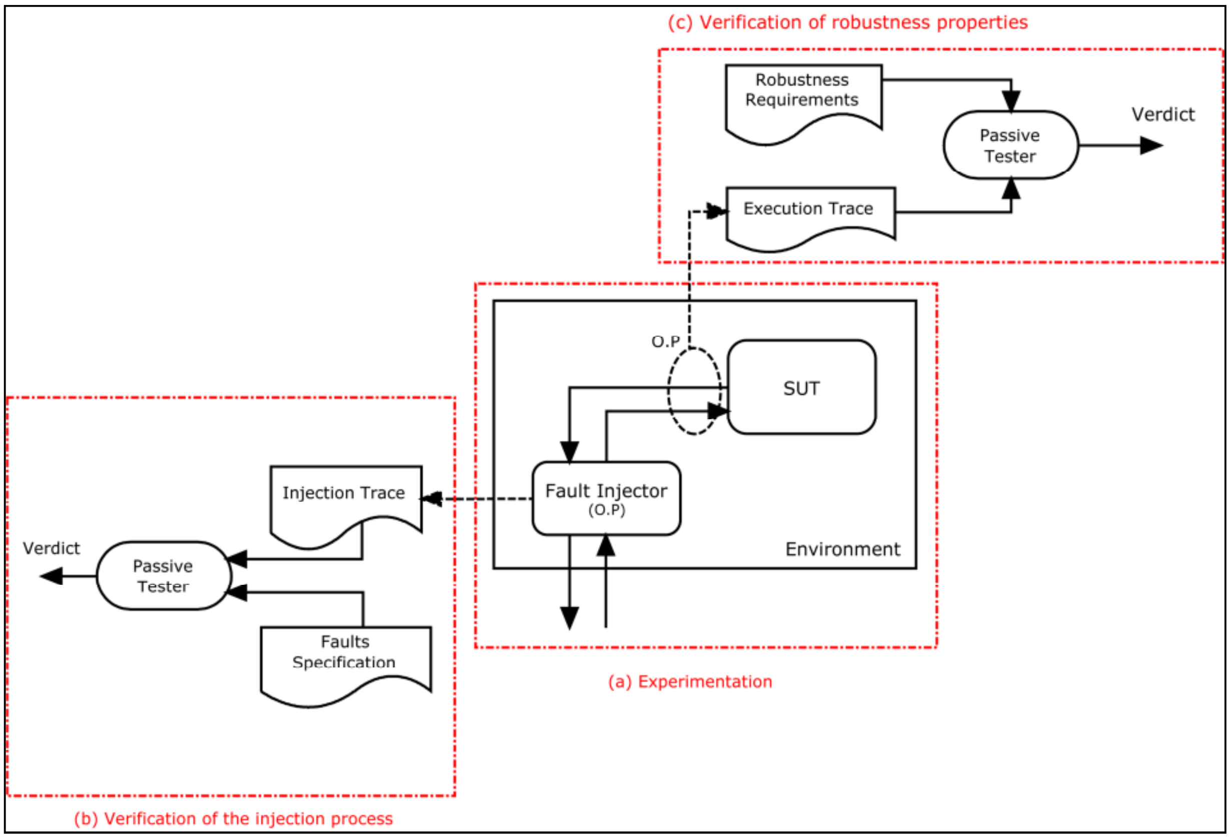


Figure 8: Architecture of a Robustness Testing approach

An interesting and efficient tool has been developed and applied in an industrial framework through a European project [23]. It deals with Web services composition robustness.

2.3 SUMMARY

Network monitoring is an important task for network traffic management because it helps achieve performance monitoring, user profiling, fault monitoring, security analysis, and intrusion detection. Being able to monitor and analyze the network is vital in the job of service providers. As described throughout this section, several monitoring techniques are available to help users and administrators in the day-to-day monitoring and analysis of their networks. These techniques can be combined with other testing techniques to achieve different/better goals.

3. ACTIVE TESTING TECHNIQUES

3.1 INTRODUCTION

Active testing is based on the execution of specific test sequences against the implementation under test. These test sequences are obtained from the formal security and functional model according to different test coverage criteria. These criteria can be applied on the specification, e.g. coverage of all logical conditions, coverage of all paths. This allows answering the question whether we have covered the specification as well as the code in testing or not. The tests may be generated automatically or semi-automatically from test criteria, hypothesis, and test goals. When generating the tests, we are faced to the feasibility problem, i.e., the problem of deciding the feasibility of a path is undecidable.

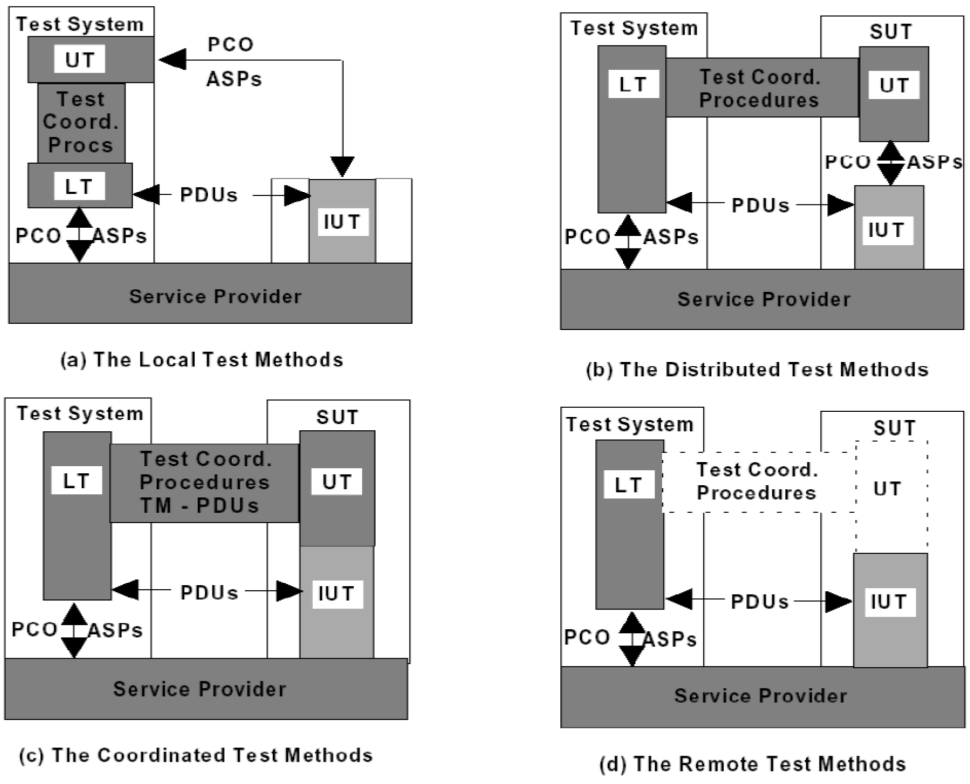


Figure 9: Test architectures

The format of test sequences, which is commonly used by the testing community, is TTCN3 [69], from which their executions are performed through PCOs (Points of Control and Observations). These PCOs are installed in the context of a testing architecture, which means the way to put the testers (e.g. upper and lower testers (UT, LT respectively) to test a specific stack layer, the different interfaces, and the oracle in order to provide a verdict on the executed tests). The International Organization for Standardization (ISO) proposes in ISO 9946-1 [111] different conformance testing architectures. The main differences between the presented architectures rely on the position, communication and synchronization of the components: lower and upper testers as well the Testing Coordination Procedure (TCP) through PDU (Protocol Data Units) or ASP

(Abstract Service Primitives). Besides, some testing architectures have been defined by the ISO as illustrated in Figure 9.

Within active testing, we distinguish two families of testing: static and dynamic active testing. The first one is based on static analysis of the source code, i.e. the implementation. The code is inspected regarding the elaborated checklist or by analyzing the control and data flow graph. Using this kind of test, we do not have to exercise the system under test with real data. On contrary, dynamic testing implies that the system under test is executed under different configurations, i.e. with different input data tests. The tests sequences to be exercised on the implementation are derived from the model described by a formal description technique. Afterwards, the inputs of the test sequence are given to the implementation and the output results are compared to those expected by the specification.

Active technique is accomplished by applying a sequence of inputs to the implementation, by means of an external tester, and verifying whether the sequence of outputs conforms to the specification. Traditional testing methods tend to test a system as a whole or to test their components in isolation. Testing systems as a whole is often difficult due to the large number of combinations of system states and variable values, known as the *state space explosion*. It is a challenge to be able to minimise the number of tests needed while guaranteeing good fault coverage. During the past years, a lot of research work has been done on developing complete FSM-based conformance testing methods, studying the fault coverage of test suites [251] and developing testing computer-aided tools. These methods, namely W, Wp, UIO, UIOv, DS, HSI and H, have been used in order to derive tests for realistic protocols [65].

Figure 10 depicts the different steps that need to be followed to carry out the active testing based approach:

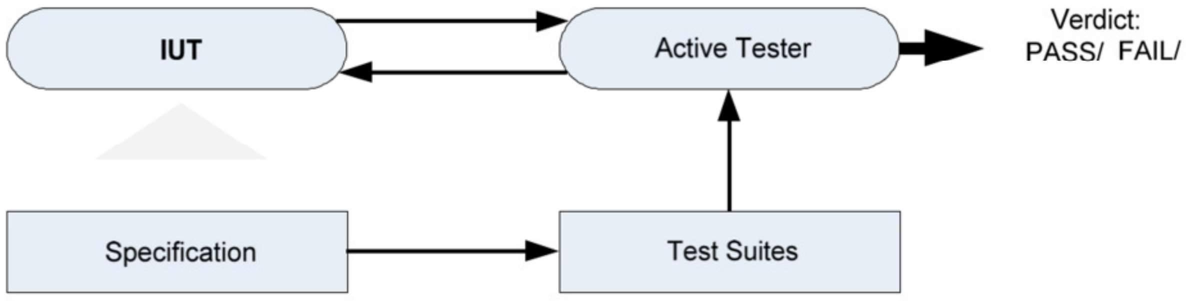


Figure 10: Active testing based approach

Within the active testing approach the following information and activities are required:

1) Specification of the functionalities and the requirements of the system using a formal language (EFSM, LTS, TEFSM etc.).
2) Definition of a set of test objectives to check the conformance of the implementation with respect to the formal specification. The goal of such test objectives is to guide the test generator to produce relevant test scenarios. In other words, these test objectives indicate the specification parts one would like to check.
3) Generation of test scenarios based on the available information and models. The outcome is a test suite.
4) Execution of the test suite on the implementation under test through the testing architecture and analysis of the results.

The outcome of the approach is information about the current state of the implementation under test (IUT), i.e., whether the IUT still contains faults or not. In the context of security testing the idea is to gain information regarding the fulfilment of security requirements.

In the following we discuss some active testing approaches that have been used in practice or at least within application-oriented research projects. The focus is on techniques with the potential to be used within the DIAMONDS project.

3.2 ACTIVE TESTING TECHNIQUES

3.2.1 Conformance testing based on security policies

Conformance testing checks whether requirements are correctly implemented in a system. More specifically, conformance testing is usually based on a detailed specification of a system, and systematic tests are derived from the specification. In the case of security, the specification will often be expressed as a security policy, which can be expressed in various forms depending on the system considered.
In the area of security testing, two major trends follow the conformance testing approach based on derivation of tests from specifications and have led to a number of publications in two directions. Firewall testing concentrates on testing firewall configurations. On a broader scale, testing can be derived from overall security policies for an information system.

A) Firewall testing
Firewall policies are defined by rules, which have a simple systematic form that can be analyzed, and from which test instances can be derived to look for potential mis-implementations. Alternatively, when the configuration of a firewall is not directly accessible, but when some global more abstract set of rules is defined (as in the case of large systems), a set of firewalls can be tested against such rules to check that all have been correctly configured. Actually, when dealing with state-based firewalls, more complex processing is involved, and there can be interactions with protocol state logic and implementation.
In [220], Senn and al. propose to model firewall rules using Mealy automata. Abstract test cases are generated to cover the transitions of the automata, and test packets are derived using a uniformity hypothesis on the range of addresses and ports.
In [35], Brucker and al. use HOL (Higher –order logic) to describe firewall (with possible stateful rules). The HOL-TestGen System is then used to test the conformable of actual implementation to the firewall rules. In [36] the authors propose rule transformation to optimize the test generation.

B) Testing from global security policies
Security policies can be expressed at the level of an information system. Model-based conformance testing can be envisioned when the policies are formalized in a model.
This approach has been proposed in [150] to generate conformation tests for RBAC models. The authors define a conformance relation, a fault model, and propose heuristics to generate tests. The method is extended to Temporal Role Based Control (Temporal RBAC) in [151].
DIAMONDS partners have experience in testing based on OrBAC models, which actually could also be applied to other types of rule based security policy languages such as Ponder or PDL. Similarly, testing has also been derived from logical translations of security rules in LTL or the Nomad deontic logic. For example, in [144], a functional model of an application, expressed as an extended Finite State Machine, is combined with security rules in OrBac. The combined model is used to generate test cases.

Another approach has been proposed by Grenoble INP in [58] to combine abstract security rules expressed in a modal logic (a form of LTL) with the actions performed by the tested application, in an executable test case.
Typical rules are:

External relays shall be in the DMZ
If an electronic mail is infected by a virus, the virus shall be deleted from the mail

The rules combine basic predicates which are system dependent with temporal operators. In order to provide instances of test sequences (test steps in the sense of TTCN-3) to test those predicates, the approach uses a library of so-called tiles that are test sequences (or test patterns). Each tile consists of low-level interactions with the system that check whether the predicate is true on the system. Each predicate is simple enough to be tested by a simple straightforward sequence (or even a single input/output pair). Theses tiles are then automatically assembled according to the syntax of the rule. The tests are carried out by testers over a network.
The approach takes as input the policy rules, and identifies the temporal and deontic dependences as combinators combining the various elementary tests (implemented by tiles). In that way, a full test suite can be derived to cover a full scale security policy. The test suite can be more or less detailed, depending on the number of elements in the system and the combination of configurations that appear as parameter of the predicates.
For example the rule "*If an electronic mail is infected by a virus, the virus shall be deleted from the mail*" in the previous example can be written in logic as (**F** is the "eventually" modality):
enterNetwork(m)[infected(m)] $\Rightarrow$ **F** *transfer((h1, h2, m) [interior(h2)*$\wedge\neg$*infected(m)*
A choice is made by the user of how to translate basic predicates into tiles. Then the method combines the tiles into a test case, chaining states and adding timers (for testing the eventuality) as shown in Figure 11.

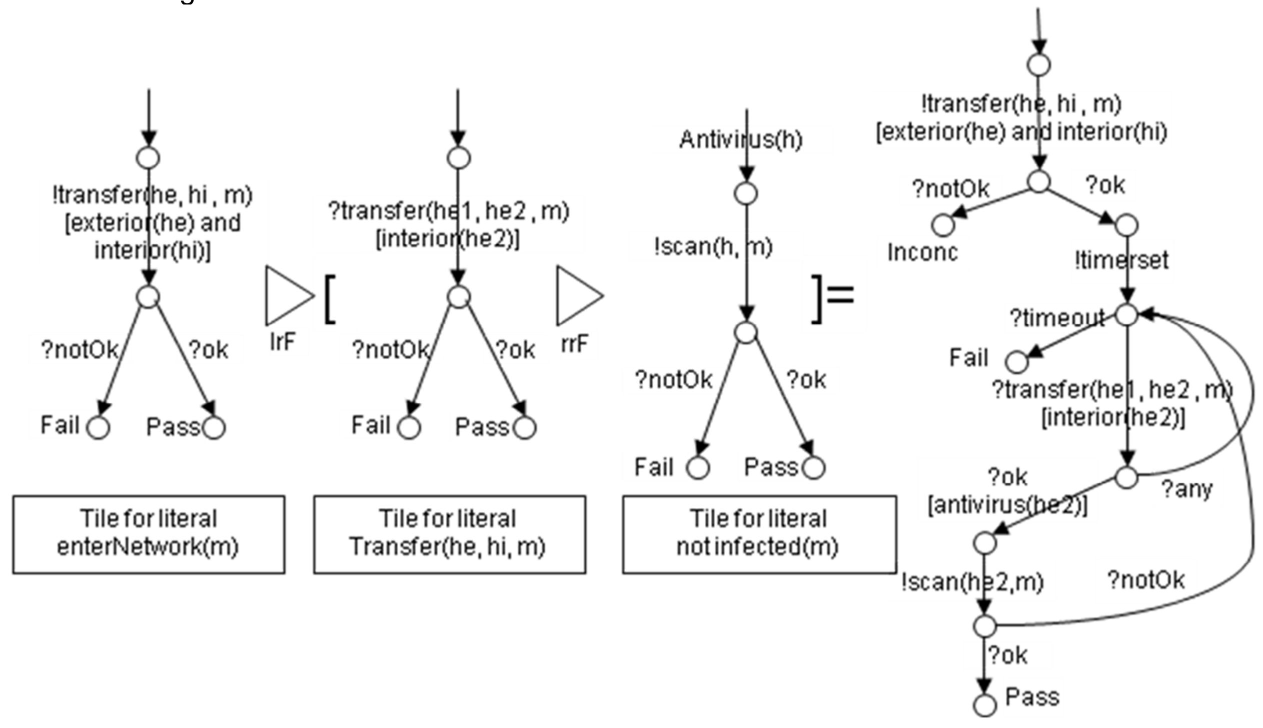

!transfer(he, hi, m) [exterior(he) and interior(hi)]
?notOk
?ok
Fail
Pass
Tile for literal enterNetwork(m)
lrF
[
?transfer(he1, he2, m) [interior(he2)]
?notOk
?ok
Fail
Pass
Tile for literal Transfer(he, hi, m)
rrF
Antivirus(h)
!scan(h, m)
?notOk
?ok
Fail
Pass
Tile for literal not infected(m)
]=
!transfer(he, hi, m) [exterior(he) and interior(hi)]
?notOk
?ok
Inconc
!timerset
?timeout
Fail
?transfer(he1, he2, m) [interior(he2)]
?ok [antivirus(he2)]
?any
!scan(he2,m)
?notOk
?ok
Pass

Figure 11: Combing tiles

The approach has been generalized in [72] to cover general LTL rules.

3.2.2 Fuzzing

Bugs are ubiquitous. Essentially all software contains hidden errors either in the form of outright implementation mistakes or subtler design issues. When not spotted during the development or testing processes, they may only manifest themselves when something unexpected imposes stress upon the system. The bug peril affects not only companies building the systems, but also the following layers in the food chain, reaching out to end customers.

Bugs are dangerous for two reasons: first, they cause systems to misbehave which induces all kinds of harm, and second, it costs money to fix the errant behaviour. Software testing tries to address these problems in the pre-release phases of the process. In this report, we will focus on what we consider the single most important conceptual insight a tester should master, and the one which is most often overlooked in practice. Should everyone involved in software testing understand the importance of robustness testing - and fuzzing as the quintessential robustness testing method - we would see fewer exploits, and also fewer and better managed vulnerability patches being pushed to the customers by software vendors.

What does it mean for software to function correctly?

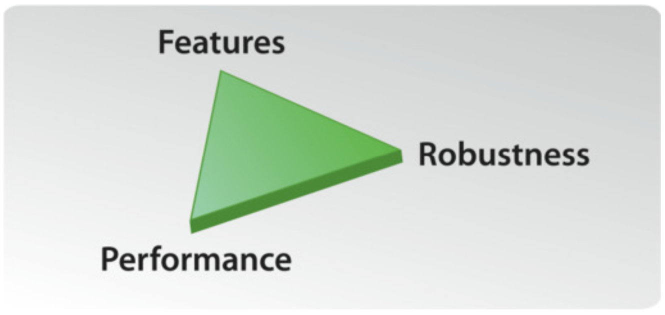


Figure 12: The three different approaches to testing

"Valid functioning" fundamentally comes in three forms, depicted in Figure 12: feature conformance, robustness, and performance. Feature conformance is easy to understand: for all valid use cases, the system should produce the correct outcome. A good system can also handle reasonable amounts of load - that is, performs well in a throughput sense. In the sequel, we simplify things a bit and consider performance as just another feature so that feature and performance testing can be collectively referred to as positive testing. However, something is still missing – robustness and performance are not the same thing.

By robustness we mean the following: Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions.

That is, a robust system restrains from doing anything undesired when faced with unexpected usage scenarios. These "mis-use-cases" may vary from oddly located mouse clicks or unexpected execution order of actions in a graphical user interface (GUI) to passing broken input through any of the communication channels of the application. Because of the mis-use-case perspective, we also use the term negative testing to refer to robustness testing.

The domains of positive and negative testing are illustrated in Figure 13. Project specification typically comes in the form of positive requirements. Sometimes some negative requirements are included, too, but only in a fraction of cases are they comprehensive and thorough. Hence, developers usually have just what we described above: a list of valid-use cases, and very little requirements on mis-use cases.

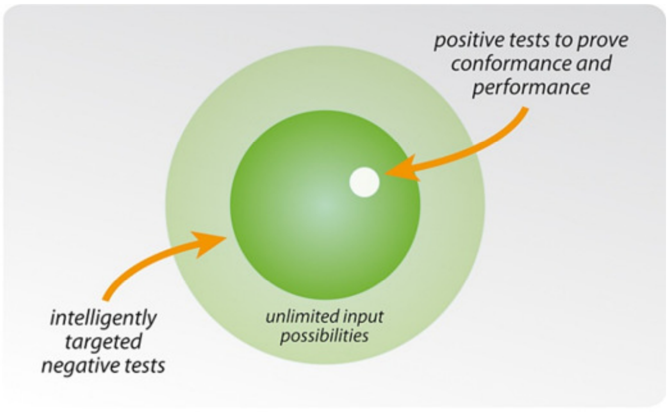

positive tests to prove conformance and performance
intelligently targeted negative tests
unlimited input possibilities

Figure 13: Positive vs. negative testing

Another aspect of software development processes is that very typically also the specified and implemented feature sets somewhat differ. Perhaps some of the specified features are not tested and remain missing, and in any case, there are usually loads of "features" outside of what has been agreed upon. Some of them are intentional additions made by the programmers, and some just programming or design mistakes. This phenomenon is visually presented in Figure 14. This discrepancy between specification and implementation is something that further elevates the need for robustness testing since such programmer inventions are typically not documented. If, then, testing is performed according to the original specification, and no robustness testing is performed, one not only misses the unintended behaviour residing in the unspecified domain, but also some parts of the code intentionally produced by the developers.

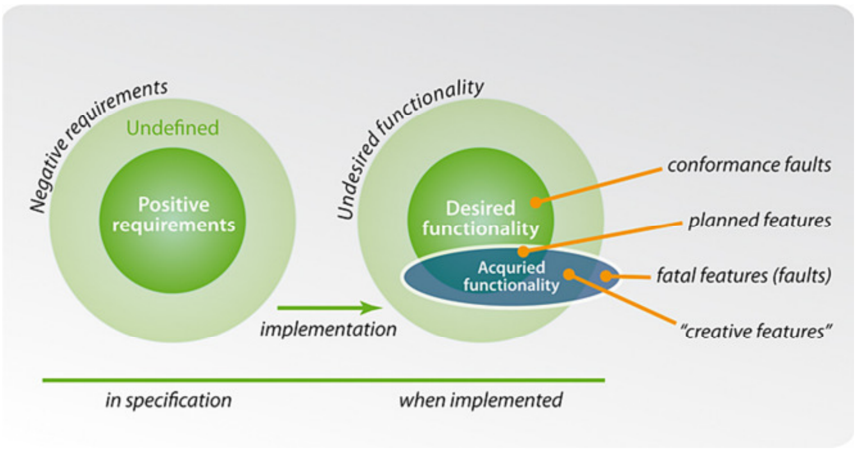


Figure 14: Specified vs. implemented functionality

What happens when negative testing is overlooked?
A shortcoming in testing leads to late discovery of errors – in the worst case, the product has already been shipped to customers. In situations where a released product has serious bugs, post-deployment fixes are needed, and they are very expensive. To think of an extreme example, you can imagine how much it has cost to car manufacturers to execute the recent recall programs to get software issues fixed.

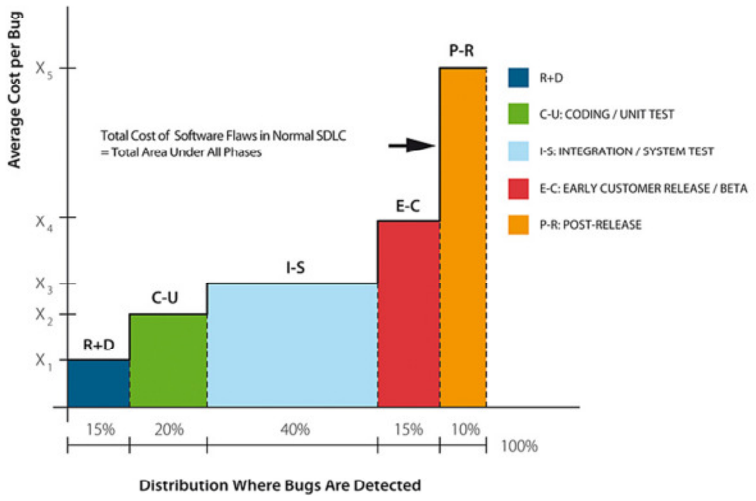


Figure 15: The cost of a bug depending on when it is discovered

Even when the software is not out yet, late bug discovery may have serious impact on the cost effectiveness of the project. According to Codenomicon studies, Figure 15 schematically presents the relationship between the discovery time of a bug and the cost associated with the fixing related to the bug. While the average costs carry no real figures, it is a commonly accepted fact that the cost of a bug fix increases dramatically with time. This means that thorough testing should be considered loss prevention: to detect and fix issues early in the development process accounts for a great loss that never occurred because of the early discovery.

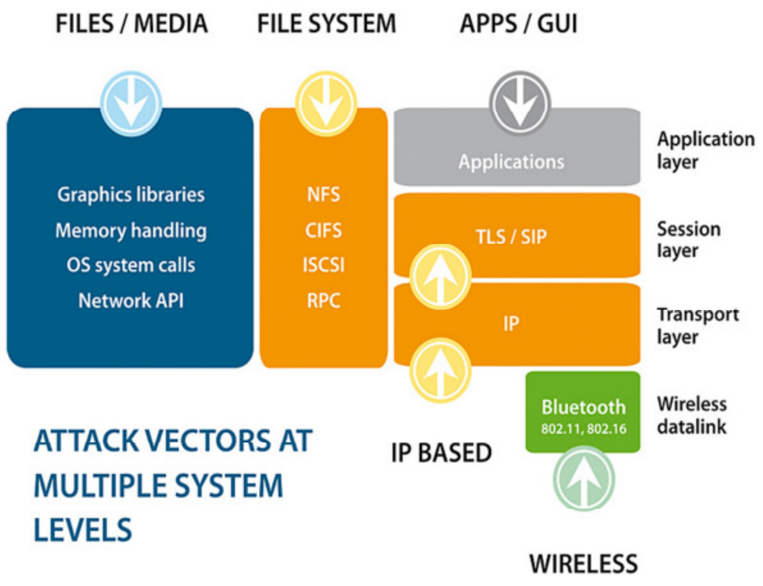


Figure 16: Attack vectors and surfaces

The need for negative testing becomes apparent when thinking about products with external interfaces and what typically happens when critical software errors are found after the product is released: very probably message sequences exist that, when sent to the system through its interfaces, cause undefined, unexpected and undesired results. The messages need not even always be sent by a malicious party: it may well be a runtime event relating to some untested program branch, or a unique load situation that is only triggered in real use that wreaks havoc: it may cause anomalous "input" to be fed into the system, and in the worst case, the consequences may be catastrophic. Some of the ways in which unexpected data can be fed into systems are illustrated in Figure 16.

3.2.2.1 Fuzzing as a method for negative testing

There are many approaches one can take to perform negative testing, but when choosing a tool, one consideration is particularly important: from a hacker's perspective, the input space is infinite so that, when testing software by systematically trying whether it can handle a plethora of malformed input, educated decisions have to be made about which messages to send. The industry leading method for generating sets of effective test sequences is called fuzz testing, or just fuzzing.
The importance of fuzzing stems from the fact it is extremely efficient in exploring the infinite input space in a clever, and is thus perhaps the only efficient way of finding unintentional security holes. Hence, also hackers invariably try to break into systems using the exact same methods as negative testers, so the builders of software-driven systems should be ahead of hackers and try their methods before deploying a product or a service and exposing it to attackers. An additional advantage fuzzing offers for hackers is that it is a black box testing method, requiring no access to source code that whenever a system is live and accessible through a network interface, hackers can target it by fuzzing.

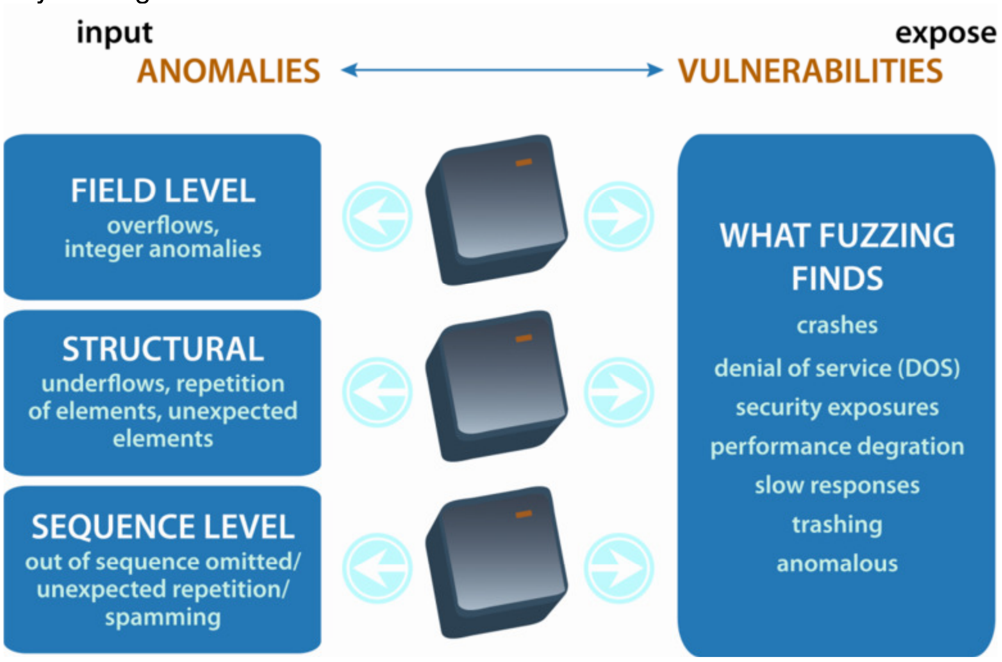


Figure 17: Types of anomalies Defensics produces, and some types of problems it is able to find

3.2.2.2 Automate the Process by Fuzzing

Fuzzing is a technique for intelligently and automatically generating and passing into a target system valid and invalid message sequences to see if the system breaks, and if it does, what it is that makes it break. An important feature of fuzzing is that it requires no knowledge of implementation details of the target system, and when it is so employed, it is thus called a black box testing method. Fuzzers can of course also use some additional information such as the

source code of the target, and depending on details, it may then be called grey or white box fuzzing.
By using the term "intelligent" we mean that a good fuzzer does not shoot at random, but rather targets typical programmer errors to increase the rate at which bugs are found. Another benefit from using carefully built test cases is that, whereas it is extremely improbable for a random feed generator to be able to produce something that passes initial validity tests or authentication handshakes, an intelligent fuzzer can be built to do just that and get to test the deeper layers of the target system.
What makes fuzz testing so valuable from a security perspective is that it is capable of detecting previously unknown issues that would have been very hard or impossible to discover otherwise – that is, by fuzzing, you can find zero day vulnerabilities.

3.2.2.3 The Three Categories of Fuzzers

Fuzzers can be categorized into three groups based on how sophisticated they are: random fuzzers, template based fuzzers (sometimes called "mutation fuzzers") and specification based fuzzers. The random variant of fuzzing is becoming more and more of a rarity since it finds bugs at a slower rate and, in practice, fails to discover many issues because of rather strong structure constraints the inputs must fulfil to be actually processed in the target system the first place. Hence, in the sequel, we restrict the analysis to fuzzers based on template data or specifications, and collectively refer to these two types of fuzzing tools as "intelligent fuzzers".
When a fuzzer is specification based (SBF), it includes full knowledge of the protocol specification the test targets, so that it is able to enumerate all or most valid protocol messages and message sequences. This is not fuzzing as such, but a SBF also includes a mechanism for anomalizing the inputs, that is, making the valid messages into something that resemble true protocol communication, but are a bit wrong in some way or the other. The protocol model and the related anomalizer of an SBF are slow and costly to build, but when a good specification based fuzzer is available, it is a very efficient tool for identifying input sequences that cause problems in the test system.
A template based fuzzer (TBF), on the other hand, means a tool that is capable of capturing real live network traffic and using it to produce similar but mutated message sequences. Just like SBF's, good TFB are efficient, too, and come with the additional benefit that they require no manual specification implementation. The anomalizer is needed, but just like in the case of SBF's, the anomalizers written for different specifications and SUT's share many common properties and methods so that when an anomalizer for a specific situation is available, chances are good that it can be easily modified to address novel test situations, too.
In addition to the fuzzer categorization based on how the protocol model is built, there are yet other division lines in the varied field of fuzzing. The testing tools can, for example, be run on different kinds of hardware. Some fuzzing tools are delivered as remote controllable dedicated appliances that are connected to a network and configured to fuzz the chosen target system. Other fuzzing tools come in the form of software products. Each type of fuzzers has its pros and cons. In situations where the configuration of the fuzzer is straightforward, an appliance might work OK. In other situations, a software fuzzer installed on standard hardware, such as an off-the-shelf laptop, allows for greater flexibility and portability to multiple test setups.

Why Fuzzing Works?
An aspect of intelligent fuzzers that makes them particularly beneficial in testing a protocol aware system is that by making use of the protocol specification or carefully analysed template data, just as intelligent fuzzers do, the tester is able to penetrate deep into the application without being caught by the first input filter or failed handshake procedure. This is also what Makes Fuzzing Work: by not just trying something but bombarding a system with systematically formed messages

the tester is capable of testing how individual program states react to different kinds of input sequences, while ignoring messages that would be too broken for the target to accept for processing at all. That is, intelligent fuzzing works because it traverses the infinite input space in such a manner that sequences more probable to cause problems are visited first. Conceptually thinking, random fuzzers are just as good, but they have the immediate practical drawback of taking too long to find anything useful.
Also, a fair share of security vulnerabilities relates to input validation issues. This makes sense: the only mechanism for a user to affect program flow is through one or the other type of input, it is the only way an attacker can make direct use of problems in software. Furthermore, in contrast to most other types of programming errors, input shortcomings in validation are, at least on the borderline, difficult to identify and typically only cursorily addressed in positive testing procedures. Outright program flow errors are much easier to hunt in positive testing settings than subtler issues lurking in the infinity of possible inputs.

3.2.2.4 Protocol modelling methods used in fuzzing tools

Codenomicon Defensics performs fuzzing, which means that software testing tools have to be capable of first creating valid message structures and message sequences, and then altering these to form nearly-valid messages that systematically anomalize some parts of the information exchange to test the target system for robustness. This section describes some methods and techniques both open source (Peach as an example) and commercial (Defensics as an example) fuzzing tools use for formally specifying messages and message sequences in the context of communication protocols.
A communication protocol is a formal system defining digital message formats and the rules describing the exchange of the messages. Thus, protocol modelling essentially consists of building a formal syntax that enables to create valid messages or confirm the validity of messages created by others, and also to create and validate sequences consisting of several messages.

The actual models that fuzzing tools use are typically variants of industry standard protocol description languages, so we will start by explaining those standards first.

3.2.2.4.1 EXAMPLES OF PROTOCOL MODELING TECHNIQUES

Augmented Backus-Naur Form

The Backus-Naur Form is a popular method of describing individual messages. As such, it is insufficient for describing message sequences. An extension to BNF which can also contain message exchange descriptions is the Augmented BNF, or ABNF.
Below is a snippet of (standard) ABNF
(from http://en.wikipedia.org/wiki/Augmented_Backus%E2%80%93Naur_Form):

```
postal-address   = name-part street zip-part

name-part        = *(personal-part SP) last-name [SP suffix] CRLF
name-part        =/ personal-part CRLF

personal-part    = first-name / (initial ".")
first-name       = *ALPHA
initial          = ALPHA
last-name        = *ALPHA
suffix           = ("Jr." / "Sr." / 1*("I" / "V" / "X"))

street           = [apt SP] house-num SP street-name CRLF
apt              = 1*4DIGIT
house-num        = 1*8(DIGIT / ALPHA)
street-name      = 1*VCHAR

zip-part         = town-name "," SP state 1*2SP zip-code CRLF
town-name        = 1*(ALPHA / SP)
state            = 2ALPHA
```

```
zip-code         = 5DIGIT ["-" 4DIGIT]
```

ASN.1

Another common format used in test automation such as fuzzing is the standard Abstract Syntax Notation One (ASN.1). It is mostly used for protocols whose specifications are written using it. ASN.1 is a method for describing the collection of individual messages in a protocol, not the message flow.
Example of ASN.1 (from http://portal.etsi.org/mbs/languages/asn.1/asn1abstract.htm):

```
ExampleProtocol DEFINITIONS  ::=

BEGIN
  ExampleProtocolMessage ::= CHOICE {
                            rlcGeneralBroadcastInformation  RlcGeneralBroadcastInformation,
                            rlcDownlinkPhyModeChange        RlcDownlinkPhyModeChange,
                            rlcDownlinkPhyModeChangeAck     RlcDownlinkPhyModeChangeAck,
                            rlcFrequencyList                RlcFrequencyList,
                            rlcConnectionAdditionSetup      RlcConnectionAdditionSetup,
                            rlcConnectionAdditionAck        RlcConnectionAdditionAck   }
  RlcGeneralBroadcastInformation
                        ::= SEQUENCE {
                            duplexMode                      DuplexMode,
                            frameOffset                     FrameOffset,
                            uplinkPowerMaxRangingStart      UplinkPowerMax,
                            infoText                        InfoText    }
    DuplexMode           ::= ENUMERATED {fdd(0), tdd(1)}
    FrameOffset          ::= INTEGER   (0 | 8..20)
    UplinkPowerMax       ::= INTEGER   (10..20)
    InfoText             ::= IA5String (SIZE (0..128))
    RlcFrequencyList     ::= SEQUENCE  (SIZE(32)) OF PairOfCarrierFrequencies
    PairOfCarrierFrequencies
                        ::= SEQUENCE {
                            uplinkCarrierFrequency          CarrierFrequency,
                            downlinkCarrierFrequency        CarrierFrequency }
    CarrierFrequency     ::= INTEGER (0..130000)
END
```

Another example from the X.509 standard that is used for specifying certain cryptographic-relates message exchange rules:

```
TBSCertificate  ::=  SEQUENCE  {
         version         [0]  Version DEFAULT v1,
         serialNumber         CertificateSerialNumber,
         signature            AlgorithmIdentifier,
         issuer               Name,
         validity             Validity,
         subject              Name,
         subjectPublicKeyInfo SubjectPublicKeyInfo,
         issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                              -- If present, version MUST be v2 or v3
         subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                              -- If present, version MUST be v2 or v3
         extensions      [3]  Extensions OPTIONAL
                              -- If present, version MUST be v3 --  }
```

XML/Schema

For testing any interfaces that use the XML specification format, the most common language that fuzzers need to support is the XML/Schema which is a method for the description of valid XML files.
Example of XML/Schema (http://www.w3schools.com/Schema/schema_example.asp):

```
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

PCAP with PDML

Fuzzing tools can also use Packet Details Markup Language (PDML) dissection of PCAP recordings to create models directly from captured messages. The PCAP + PDML are typically used in parallel as PDML alone is not enough to specify messages for test generation purposes. PDML is originally intended for describing the structure of dissected packets.
Example of PDML:

```
<pdml>
    <packet>
      ...
      <proto  name="IP"  pos="15"  showname="IPv4  (Internet  Protocol  version  4)"
size="20">
        <field name="verhlen" pos="15" show="45" showname="Version and header Length"
size="1" value="45">
          <field  mask="F0"  name="ver"  pos="15"  show="4"  showname="Version"  size="0"
value="4" />
          <field mask="0F" name="hlen" pos="15" show="5" showname="Header length"
            size="0" value="5" />
        </field>
          ...
      ...
    </packet>
    <packet>
      ...
    </packet>
    ...
  </pdml>
```

The PDML specification with samples is available here:
http://gd4.tuwien.ac.at/.vhost/analyzer.polito.it/30alpha/docs/dissectors/PDMLSpec.htm

3.2.2.4.2 PROTOCOL MODELS IN PEACH FUZZING FRAMEWORK

Peach is an open source fuzzing framework used to build fuzzing tools.
Peach is available from: http://peachfuzzer.com/

Peach PIT files

Peach uses XML Schema augmented with BNF and ASN.1 functionality.
Example data templates for text protocols (from http://peachfuzzer.com/HowDoI):

```
<!-- Create a simple data template containing a single string -->
 <DataModel name="HelloWorldTemplate">
         <String value="Hello World!" />
 </DataModel>

 <StateModel name="State" initialState="State1" >
         <State name="State1"  >
                 <Action type="output" >
                         <DataModel ref="HelloWorldTemplate"/>
                 </Action>
         </State>
 </StateModel>
```

ASN.1 example (from http://peachfuzzer.com/HowDoI):

```
<DataModel name="Asn1Ber">

         <!-- define a ber encoded string -->
         <String type="char" value="Hello World!">
                 <Transformer class="asn1.BerEncodeOctetString" />
         </String>

         <!-- define a ber encoded integer -->
         <Number size="16" signed="true">
                 <Transformer class="asn1.BerEncodeInteger" />
         </Number>
```

```
</DataModel>
```

3.2.2.4.3 CODENOMICON DEFENSICS SUPPORTED TECHNIQUES

Supported formats:

- RAW ASCII
- BNF
- ABNF
- ASN.1
- XML/Schema
- PCAP
- PDML

Internally used, proprietary formats:

- EBNF
- XML Sequences

We will first explain the Codenomicon proprietary formats/languages, and then show how a user can use these in the tool.

Extended Backus-Naur form with Java extensions

The core modelling technique at Codenomicon is an in-house extended BNF variant that allows for message exchange descriptions. The technique is called Extended BNF, or EBNF. It is described in [128].

An example of Codenomicon proprietary Extended BNF (EBNF):

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

We use extended BNF to model the syntax of messages in both binary and textual protocols. Message sequence-level behaviour is also modelled using BNF.

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

Perform I/O operations like connect, send, and receive, on message sequence level.

Calculate fields like lengths, checksums, sequence numbers, inside protocol messages.

User sequences

In addition to providing the end user with lots of ready-made test cases - that is, anomalized messages and message sequences - the users of several Codenomicon test tools may also specify the used message sequences and/or message content themselves. For this purpose we use an in-house developed XML based sequence file method. A sequence file may look like the following (the extract has been copied from Codemicon SIP test suite):

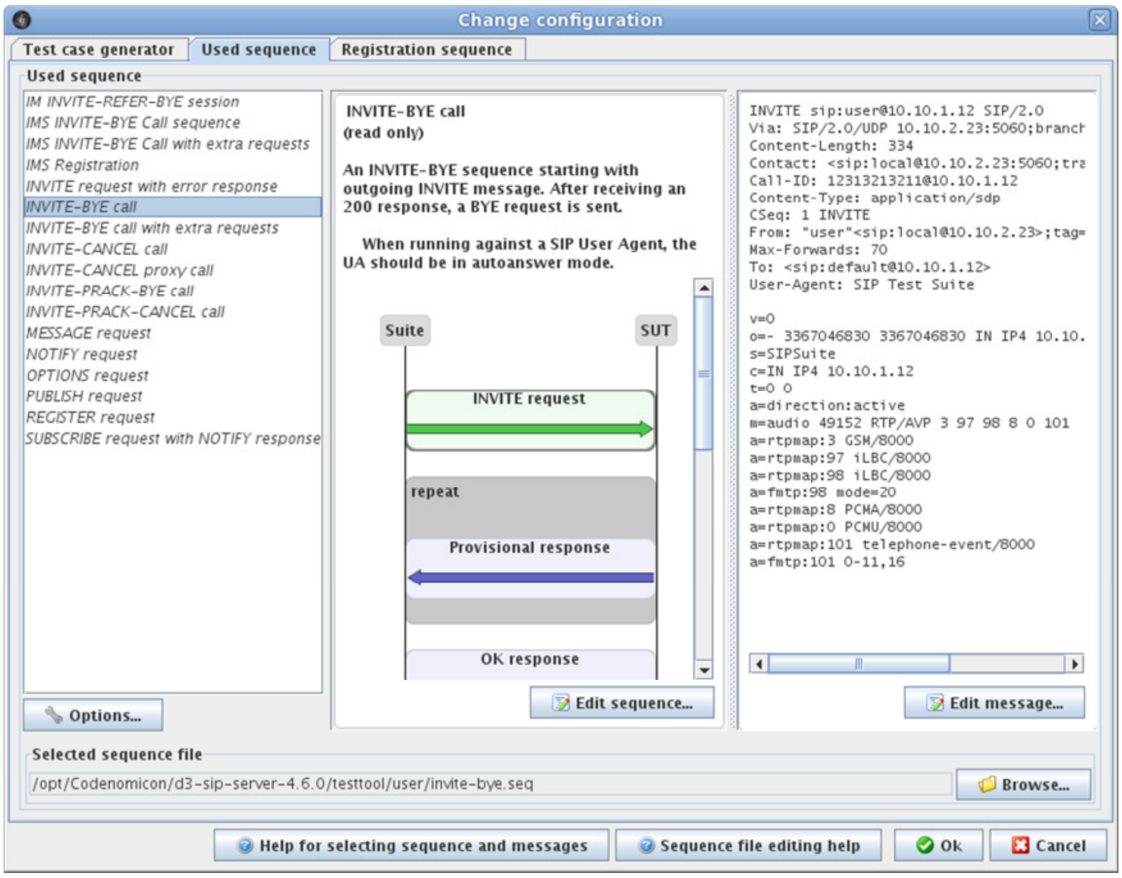
```
<sequences>
  <!-- SIP dialog definitions -->
  <sequence name="used-dialog" setting="user-sequence-file">
    <description name="PUBLISH request">A sequence sending a PUBLISH request
                and expecting a success response.</description>
```

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

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

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

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



IM INVITE-REFER-BYE session
IMS INVITE-BYE Call sequence
IMS INVITE-BYE Call with extra requests
IMS Registration
INVITE request with error response
INVITE-BYE call
INVITE-BYE call with extra requests
INVITE-CANCEL call
INVITE-CANCEL proxy call
INVITE-PRACK-BYE call
INVITE-PRACK-CANCEL call
MESSAGE request
NOTIFY request
OPTIONS request
PUBLISH request
REGISTER request
SUBSCRIBE request with NOTIFY response

INVITE-BYE call
(read only)

An INVITE-BYE sequence starting with outgoing INVITE message. After receiving an 200 response, a BYE request is sent.

When running against a SIP User Agent, the UA should be in autoanswer mode.

```
INVITE sip:user@10.10.1.12 SIP/2.0
Via: SIP/2.0/UDP 10.10.2.23:5060;branch
Content-Length: 334
Contact: <sip:local@10.10.2.23:5060;tra
Call-ID: 12313213211@10.10.1.12
Content-Type: application/sdp
CSeq: 1 INVITE
From: "user"<sip:local@10.10.2.23>;tag=
Max-Forwards: 70
To: <sip:default@10.10.1.12>
User-Agent: SIP Test Suite

v=0
o=- 3367046830 3367046830 IN IP4 10.10.
s=SIPSuite
c=IN IP4 10.10.1.12
t=0 0
a=direction:active
m=audio 49152 RTP/AVP 3 97 98 8 0 101
a=rtpmap:3 GSM/8000
a=rtpmap:97 iLBC/8000
a=rtpmap:98 iLBC/8000
a=fmtp:98 mode=20
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11,16
```

/opt/Codenomicon/d3-sip-server-4.6.0/testtool/user/invite-bye.seq

Browse...
Help for selecting sequence and messages
Sequence file editing help
Ok
Cancel

3.2.2.5 Conclusion

Section 3.2.2 illustrated that when testing software to ensure its quality and security, it is essential to consider all three aspects of testing:

- Feature testing
- Performance testing

- Robustness testing

As a further incentive to test properly, we demonstrated that sound testing also has positive cost-benefit effects due to the fact that an early discovery of bugs enables them to be fixed quickly, without additional costs.
Moreover, we introduced fuzzing as the standard robustness testing methods and summarized the three categories of fuzzers:

- Random fuzzers
- Model based fuzzers
- Specification based fuzzers

Remembering the messages displayed in Figure 12 and Figure 13, we conclude this section by emphasizing that to achieve software security, it is essential to harness the power of robustness testing to its full potential, and never look back again.

3.2.3 Active testing based on random walks

One of the active testing tools, TestGen-IF implements an automated test generation algorithm called *Hit-or-Jump* [43]. It allows obtaining a guided random walk to generate test suites and to cover all the interactions of the component in its context. The essence of this approach is as follows. At any moment, we conduct a local search from the current state in a neighbourhood of the deployed transition system. If an untested part of the component is found (a Hit, i.e., an interaction that has to be tested), we keep it for the final test sequence, and then continue the search process from there. Otherwise, we move randomly to the frontier of the neighbourhood searched (Jump), and resume the process from there. This procedure avoids the building of a whole system accessibility graph. Accordingly, the space required is determined by the user, e.g., a depth limit or a maximum number of states, and it is independent of the system under consideration. On the other hand, a random walk may get trapped at certain part of the component under test. The algorithm is designed to jump out of the trap and pursue the exploration further. We build at each step a partial accessibility graph to avoid the state-number explosion problem. The algorithm finally produces a test sequence as a transition tour of the component in its context.
IF[2] refers to an open validation platform developed by Verimag for asynchronous timed systems such as telecommunication protocols or distributed applications. It uses the IF (Intermediate Format) language based on timed automata extended with discrete data variables, various communication primitives, and dynamic process creation and destruction. The IF simulator allows the execution of the specification to produce the partial reachability graph and the derived test cases. The TestGen-IF tool architecture is illustrated in Figure 18.

The *Automatic Test Generation* box represents the Test Generation combined with the IF Specification of IUT and the Test Purposes. The properties which represent security test purposes can be described using any type of signals allowed by the IF specification (input, output, informal...), states or clock variables. It is up to the user to choose the type of exploration he wants to be performed during the test generation: in depth or in breadth exploration of the generated partial reachability graph.

[2] http://www-verimag.imag.fr/article58.html?artpage=2-5&lang=en

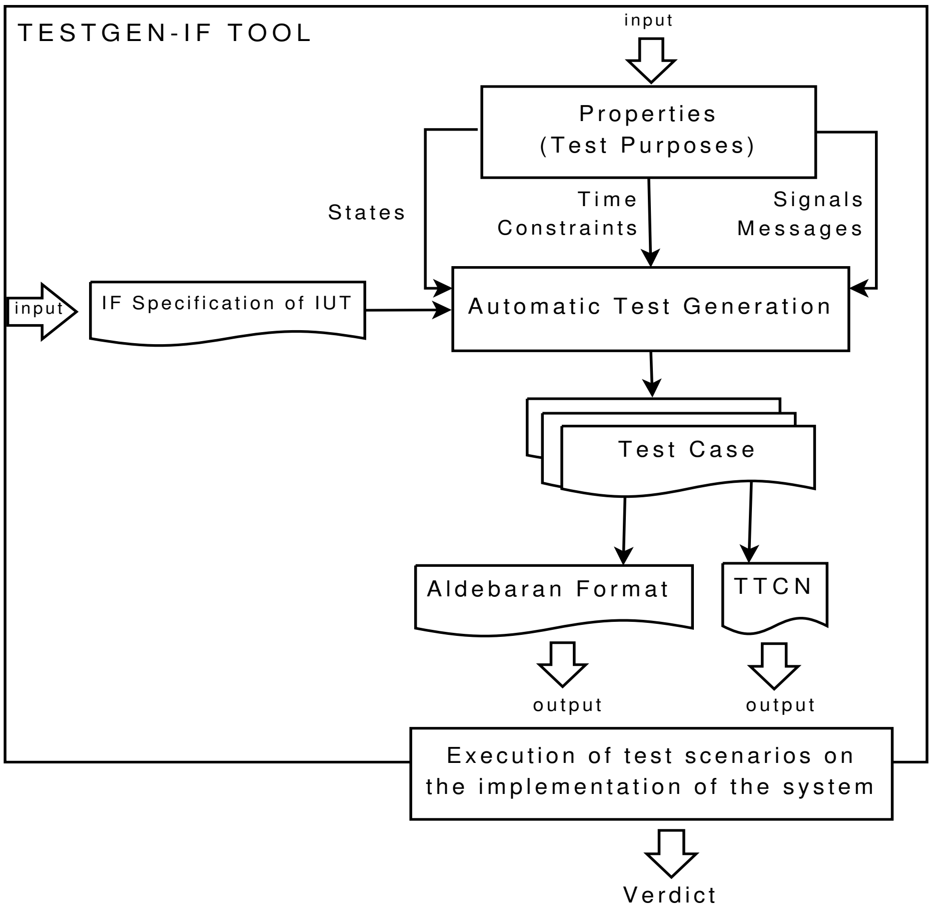


Figure 18: TestGen-IF: an active testing tool

Using this approach, the tool generates a test suite that satisfies the given set of test purposes. A test suite is composed of several test cases (or scenarios) that are related or cooperate with each other. A test case or Test Case Suite is a sequence of states and transitions that validates one or more system requirements and generates a pass/fail verdict.
During the execution, when a test purpose is satisfied, a HIT message is displayed to inform the user. It is followed by the description of the test purpose that was found (a signal, a state or a clock). The number of test purposes already found and the number of those missing is also displayed. If a JUMP is performed a corresponding message is displayed.
The exploration ends either when all test purposes are found or when the system reaches the final leaf node. The user is then informed about how many test purposes were satisfied and how many, if any, were not. As output, three files can be generated: the sequence of states/transitions visited (passed or failed) in Aldebaran[3] format, a text file listing the states visited and a detailed an XML log file giving the states, transitions, and the values of the variables used.

TestGen-IF has been successfully used in the European project SHIELD[4] [223]. A module has been added to enable TestGen to benefit from the formalisms designed in SHIELDS and generate test cases targeting systems' security goals in order to detect potential vulnerabilities. Starting from

[3] http://www.inrialpes.fr/vasy/cadp/man/aldebaran.html
[4] http://www.shieldsproject.eu/

security goals described in different SHIELDS models (e.g. Security Activity Graph (SAG), Security Goal Indicator Tree (SGIT), threat models, etc.), a set of security rules are manually generated. These security rules are then formally specified in Nomad formal language (see Section 1.1.5) and constitute the first input for the TestGen tool. The approach has notably been applied to the PCE communication protocol [198] and RTSP.

3.2.4 Model-based mutation testing

A promising testing technique for ensuring security is model-based mutation testing. It is a combination of model-based testing and mutation testing.

Mutation testing is a way of assessing and improving a test suite by checking if its test cases can detect a number of injected faults in a program. The faults are introduced by syntactically changing the source code following patterns of typical programming errors. These deviations in the code are called mutations. The resulting faulty versions of the program are called mutants. Usually, each mutant includes only one mutation. Examples of typical mutations include renaming of variables, replacing operators, e.g. an assignment for an equivalence operator, and slightly changing Boolean and arithmetic expressions. The number and kind of mutations depend on the programming language and are defined as so called mutation operators. A mutation operator is a rewrite rule how certain terms in the programming language are replaced by mutations. For every occurrence of the term the mutation operator rewrites the original program into a new mutant. After a set of mutants has been generated, the test cases are run both on the original and on each mutant. If a test case can distinguish a mutant from the original program, i.e., different output behaviour can be observed, we say that this test case kills a mutant. The goal is to develop a test suite that kills all mutants. This technique of program testing has been invented by Hamlet and DeMillo in the70-ies.

Mutation testing has three basic assumptions: (1) the competent programmer assumption, assumes that programmers make only small errors. This argument supports the use of small variations in the code to represent the fault models, i.e. the typical faults of programmers; (2) the chosen mutation operators is a representative set of those errors; (3) via a coupling effect, more subtle errors can be detected by testing against the simple errors only.

However, even with these assumptions there is a fundamental difficulty in this approach. Not all mutations represent actual faults producing observable failures. For example, a mutation in dead code, i.e. code that is not reachable, will not lead to failures. Mutants with such an equivalent behaviour are called equivalent mutants and can never be killed by any test case. This poses a serious limitation to the mutation testing technique: for a mutant surviving the tests, i.e. it is not killed, we do not know, if this is due to our inability to come up with a proper test case, or due to the fact that there is no such test case, because it is an equivalent mutant. Unfortunately, we cannot automatically exclude equivalent mutants from the process, since the problem is undecidable in general. Otherwise we would be able to distinguish terminating from non-terminating programs automatically, which would solve the undecidable halting problem.
Hence, manual inspection of the surviving mutants was often needed in order to exclude the equivalent ones. In recent years, with the advent of model checking techniques, the situation improved considerably: Today, the equivalence of two mutants can be decided for a growing class of programs assuming finite data-types. This may explain the returning interest in mutation testing. In a recent survey on mutation testing Jia and Harman point out the problem of equivalent mutants:

``One barrier to wider application of Mutation Testing centres on the problems associated with Equivalent Mutants. As the survey shows, there has been a sustained interest in techniques for reducing the impact of equivalent mutants. This remains an unresolved problem." [123]
In this survey the authors also mention the need for test case generation techniques:
``Most work on MutationTesting has been concerned with the generation of mutants. Comparatively less work has concentrated on the generation of test cases to kill mutants." [123]

TU Graz's past research and current work addresses this gap in research. In the following, we explain our use of model mutation to generate test cases and discuss our specific contribution in this area.

Mutation of Models. In model-based mutation testing, we mutate models of the system under test. Like in standard model-based testing the aim is to use the model both for generating test vectors and as a test oracle. Hence, we generate test cases from a model for testing the conformance of a SUT. In contrast to classical model-based testing, only those test cases are generated that would kill a set of mutated models. The generated tests are then executed on the SUT and will detect if a mutated model has been implemented. Hence, model-based mutation testing tests rather against non-conformance, than for conformance. In terms of epistemology, we are rather aiming for falsification than for verification. It is a complementary fault-centred testing approach.

Protocol Mutation Testing. In recent years TU Graz has successfully applied mutation testing to several protocol implementations, including HTTP [6], Session Initiation Protocol (SIP) and the Conference Protocol [248]. All these protocols have been modelled in the process algebra LOTOS. These models represent the exchange of messages as events and describe the possible sequences of such events in form of processes. Consequently, the conformance (refinement) relation between such models is defined on the event level. The conformance relation of our choice was ioco, the input-output conformance relation of Tretmans.
Informally, ioco conformance is given, if for all event traces possible in the specification model, the implementation does not produce output that is not allowed by the specification. It supports incomplete (partial) specification models. This is realized by splitting the events into input and output events, i.e. controllable and observable events. Hence, an SUT may react to unspecified input events in an arbitrary way, like programs outside their specification precondition. This is an important feature for industrial application, where it is unrealistic to model the complete behaviour of a system under test. In addition, ioco knows a special quiescence event for modelling timeouts.

We have implemented an automatic mutator for LOTOS specifications, altering the model according to a set of predefined mutation operators. Furthermore, we developed an ioco-checker that returns a counter-example in cases original models and mutants do not conform. These counter-examples serve as test purposes for the test case generator TGV producing all possible test cases leading to the error. The test cases are then mapped to and executed at the concrete protocol-level. Our case studies on real protocol implementations demonstrated that this technique is complementary to other testing approaches. In the case of the SIP protocol mutation testing revealed an additional fault that was detected neither by random testing nor by scenario-based testing.

3.2.4.1 Embedded Systems Mutation Testing

Model-checkers can be exploited to generate mutation tests. The idea is to generate counter-examples to the property that a mutant is observationally equivalent to its original model. At TU Graz this idea has been investigated on testing embedded systems with the SMV model checker. For a general survey on model checkers and testing see [78].

Model-based mutation testing has also been applied in the European FP7 project MOGENTES (www.mogentes.eu). In order to support the formal testing from informal UML diagrams TU Graz and AIT defined a precise semantics for UML by mapping it to formal intermediate models. As a target language Back's Action Systems have been chosen. Action systems are a kind of guarded command language for modelling concurrent reactive systems. To our knowledge this was the first time that Action Systems have been used as test models in model-based testing. MOGENTES exploited an object-oriented extension of action systems to support the object-orientation in UML and labelled the actions to interpret them as labelled transition systems. This semantic mapping from UML via Action Systems to labelled transition systems provided the link to the existing body-of-knowledge on formal testing that is mostly based on labelled transition systems. Consequently, TU Graz and AIT were able to develop a formal input-output conformance relation and mutation testing technique for non-trivial UML State Transition Diagrams. Unlike the existing UML tools for test case generation, we support parallel regions, nested states, non-deterministic models, inheritance and time-out events.

ULYSSES is a test case generator that has been developed in MOGENTES. It supports model-based mutation testing: ULYSSES takes two action systems, an original and a mutated one, and generates a test case that kills the mutant. To our knowledge, this is the first mutation test case generator for action systems, as well as for UML. ULYSSES expects the action being labelled as input, output and internal actions. For deterministic models, the generated test case is a sequence of input and output events leading to the faulty behaviour in the mutant. For non-deterministic models a tree-like adaptive test case is generated. ULYSSES explores both action systems, determinizes them, and produces a synchronous product modulo the ioco-conformance relation. ULYSSES is implemented in Sicstus Prolog exploiting the backtracking facilities during the model explorations [33].

Different strategies for selecting the test cases from this product are supported: linear test cases to each fault, adaptive test cases to each fault, adaptive test cases to one fault. ULYSSES also checks if a given or previously generated test case is able to kill a mutant. Only if none of the test cases in a directory can kill a new mutant, a new test case is generated. Furthermore, ULYSSES is able to generate test cases randomly. Our experiments showed that for complex models it is beneficial to generate first a number of long random tests for killing the most trivial mutants. Only, when the randomly generated tests cannot kill a mutant, the computationally more expensive product calculation is started [4].

Hybrid Systems. A unique feature of our ULYSSES test case generator is its ability to handle hybrid systems. Hybrid systems involve discrete and continuous state updates as typically found in controllers interacting with a physical environment. Many embedded systems interact with a continuous environment and hence there is a strong interest in applying model-based testing to such systems. The main contribution of this line of research was an integration of model-based testing and qualitative reasoning. Qualitative reasoning is a technique from Artificial Intelligence to abstract and simulate continuous systems qualitatively. We first extended our semantic action system models with qualitative differential equations modelling the dynamics of the environment. Then the ASIM qualitative reasoning engine was integrated into ULYSSES. A new label for qualitative actions was added to the labelled transition system semantics. These qualitative actions represent the environmental updates of the environment reacting to the controller. The semantics of these new qualitative action systems was formally worked out including a refinement law for qualitative actions. Canonical water tank examples served as demonstrating case studies for the formal semantics and the test case generator [5].

3.2.5 Behavioural modelling for testing security

In the (behavioural) model-based testing (MBT) approach, the description of the system under test (SUT) is realized in a behavioural model and the test cases are automatically generated from this behavioural model and some test selection criteria.

Model-based testing originally focuses on functional testing (i.e. testing the expected behaviour of the system under test) but is now also used for security testing by extending the purpose of the model (for example focusing the behavioural model on security functions of the system under test) and by adapting the test selection criteria for security testing (for example by mapping security properties with dedicated test selection criteria).
The typical process of Model-Based Testing is depicted in Figure 19. It is composed of 5 main steps:

1. The first step consists in designing a formal model from the initial requirements (in- formal). It is mandatory that this model takes into account the (functional, security, etc.) requirements that the system must fulfil.
2. The second step consists in choosing a test selection criterion that will make it possible to exploit the model content so as to produce the test cases. These criteria may aim at: the coverage of a model functionality, the coverage of the model states/data, a random walk in the model, highlighting a given set of faults, etc.
3. The third step is the formalization of the test cases, which gives a high level description of the test cases that one wants to generate. It represents an intermediate step that formalizes the test selection criterion so as to make it operational (to be automatically applied on the model).
4. The fourth step consists in the generation of the test cases themselves. These tests are said to be abstract since they are expressed at the model level, using the model entities (data and operations). This step can be automated by a test generator that is able to deal with test case formalization produced at step 3. The resulting abstract test cases are made to fulfil the chosen test selection criterion.
5. The fifth and final step consists in executing the tests on the System Under Test (SUT) and assigning the test verdicts. This is usually done in two steps.
   5–1 Concretization of the abstract tests. This substep consists in translating the abstract test cases into concrete calls to the SUT interfaces.
   5–2 Verdict assignment. This substep consists in interpreting the results obtained from the SUT in response to the stimuli. First, these results are translated back into the formalism of the model, in order to be compared to the expected results that were computed from the model. A conformance relationship is used to check that the results conform to each other.

Notice that, in case of online testing (i.e. when tests are generated and played at the same time on the SUT), this step is coupled with step 4.

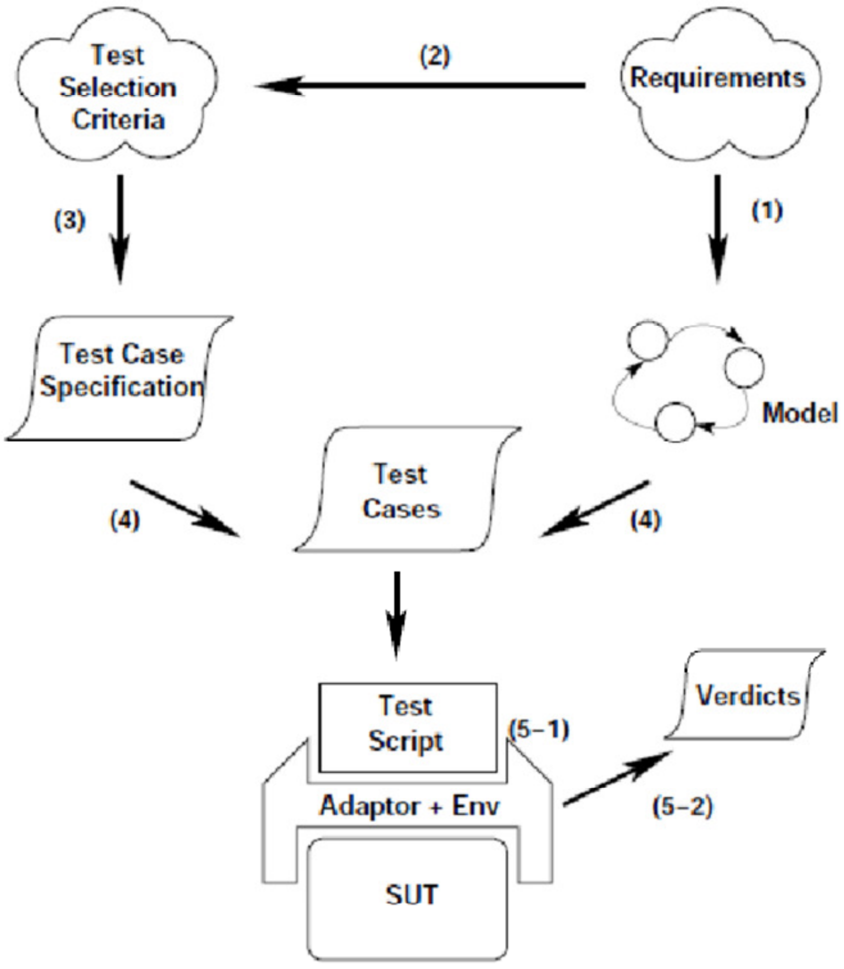


Figure 19: Typical MBT Process

3.2.5.1 Main characteristics of the behavioural model-based testing approach

We briefly review the classification proposed in [243]. This classification takes into account three main aspects, which are decomposed into subcategories:

- the model, that is seen as a scope, the characteristics of the modelled elements, the modelling paradigm,
- the test generation technique, involving test generation criteria and an underlying test generation technology,
- the execution of the tests, that can be either online (tests are generated and played on the SUT at the same time), or offline (tests are generated and stored in a test repository, before being translated to be run subsequently on the SUT.
- We now detail the first two aspects.

Model

As explained previously, the models are classified according to three criteria:

Scope
The model scope defines whether the model is input-only, or input-output. Input-only models are models that only specify valid inputs for the system without providing any information on the expected outputs. Such models can be used to generate test data (but, not necessarily test cases). However, they cannot be used for conformance testing. Nevertheless, these kinds of models can notably be used when performing robustness testing, when the test objective is to ensure that the system does not crash. On the contrary, input-output models specify both valid inputs and expected outputs which make them a suitable choice for conformance testing.

Characteristics
The model characteristics answer the three following questions:
- Is the model timed or untimed?
- Is it a deterministic or non-deterministic model?
- What is the dynamics of the model: continuous time, discrete time, or both?

Modelling paradigms
The modelling paradigms are the following.
- State-based notations: these notations describe a system using a set of state variables and operations modifying them. Each operation is specified using preconditions (describing the licit uses of the operations) and postconditions (describing the effect of the operation). Some state-based notations: B [1], Z [231], VDM [124], JML [38], UML/OCL [38][203][246].
- Transition-based notations: these notations describe the transitions between the states of the system. These are usually graphical notations, using nodes to represent states, and edges to represent transitions. These formalisms can be extended to take into account state variables, guards on the transitions, etc. Some transition-based notations: finite state machines (FSM) [134], (input-output) labelled transitions systems [240], statecharts [89].
- History-based notations: these notations describe a system through its acceptable traces. Such notations make it possible to address different notions of time (discrete or continuous, linear or tree-based, etc.) The message sequence charts (MSC) [90] formalism falls into this category.
- Functional notations: these notations describe the system using a set of mathematical functions (first order functions for algebraic specifications [79]), or of higher order (as in HOL).
- Operational notations: these notations describe the system as a set of executable processes that are run in parallel. These notations cover the process algebra, CSP, CSS, or Petri Nets [196].
- Stochastic notations: these notations describe the system using a probabilistic model of the events and input data. They are mainly used for modelling the SUT environment rather than the SUT itself. For example, Markov chains can be used to model operational profiles of the SUT.
- Data-flow notations: these notations are focused on the data flow rather than the control flow. Lustre [182] or Matlab Simulink [25] are classified into this category.

It is worth noticing that some notations are not restricted to a single modelling paradigm, e.g. UML combines a state-based formalism (using OCL constraints) and a transition-based formalism (using the statecharts diagrams)

Test Generation
The test generation aspects rely on two elements: the test selection criteria that can be applied, and the test generation technology that is used to compute the test cases.

Test Selection Criteria

The considered test selection criteria are the following:

- **Structural model coverage**: this criterion depends on the modelling formalism that is used. On pre/postconditions models, it aims at covering the cause-effects, the disjunctions in the preconditions, etc. On transition-based models, it aims at covering the entities of the automaton (states, transitions, etc.). On textual models, some of the structural testing criteria can be revisited and adapted to cover the code of the model.
- **Data coverage**: it represents the test data selection strategy that can be very simple (random values), or more complicated (e.g. involving pairwise testing or a boundary value analysis).
- Requirement coverage: this criterion uses the requirements (functional or security requirements) to target specific parts of the model (e.g. states, transitions, transitions sequences, etc.) to be exercised.
- **Test cases specification**: this criterion consists, for the user, to provide a test scenario that he wants to execute on the model. This scenario can be expressed in a textual way [57], [71], or using an automaton [120].
- **Random or stochastic criteria**: these criteria aim at the coverage of an operational profile of the system. Different probabilities of usage are associated to different parts of the model. The test generation process will produce tests accordingly to these probabilities.
- **Fault-based** criteria: these criteria aims at using a fault model that has to be covered by the test generation approach, meaning that the tests are produced so as to detect all the faults that are provided in the fault model.

Some approaches presented in this part are associated to tools as described in section 2 of deliverable D1.WP3 of DIAMONDS.
An issue of model-based testing is to be able to measure and ensure coverage from test to the model. The intention guarantees only test coverage on model and not on the SUT. We identified two kinds of criteria to insure coverage. Static criteria are based on two coverage approaches: control flow oriented and data flow oriented.

Control flow graph criteria

A method for structural testing based on the control flow coverage is to propose a certain set of paths in a graph in order to form test campaigns. Satisfying a structural testing method for a given coverage criterion is therefore to find tests that enhance control paths (i.e. paths of execution) and covering paths provided by the method adopted. There are several coverage criteria based on graph control:

- Coverage of all-nodes or statement coverage,
- Coverage of all-arcs or decision coverage,
- Coverage of path and internal boundaries,
- Coverage of all i-paths,
- Coverage of all paths.

This list is ordered from the lowest to the highest criterion to test (except ’coverage of paths and internal boundaries’ which is not classifiable). In general, the stronger this criterion is, the higher is the number of test data to satisfy.
To cover criteria, a test suite must activate a dedicated part of specification as follows [242]:

- **State Coverage (SC):** test suite must execute every reachable statement

Decision Coverage (DC): test suite must ensure that each reachable decision is made true by some tests and false by other tests. Decisions are the branch criteria that modify the flow of control in selection and interaction statement.

- **Path Coverage (PC):** test suite must execute every path to satisfy through the control flow graph.
- **Condition Coverage (CC):** test suite achieves CC when each condition is tested with a true result and also with a false result. For condition containing N conditions, two tests can be sufficient to achieve CC.
- **Decision/Condition Coverage (D/CC):** test suite achieves D/CC when it achieves both decision coverage (DC) and condition coverage (CC).
- **Full Predicate Coverage (FPC):** test suite achieves FPC when each condition is forced to true and to false in a scenario where that condition is directly correlated with the outcome of the decision.
- **Modified Condition/Decision Coverage (MC/DC):** This coverage strengthens the directly correlated requirement of FPC by requiring the condition c to independently affect the outcome of the decision d. A condition is shown to independently affect a decisions outcome by varying just that condition while holding fixed all other possible condition.
- **Multiple Condition Coverage (MCC):** test suite achieves MCC if it exercises all possible combination of condition outcomes in each decision.

In [162], for code-based coverage we have PC > DC > SC, where C1 > C2 indicates that every test suites satisfies C1 also satisfies C2. More generally as proposed in [242], Figure 20 gives hierarchy between criteria.

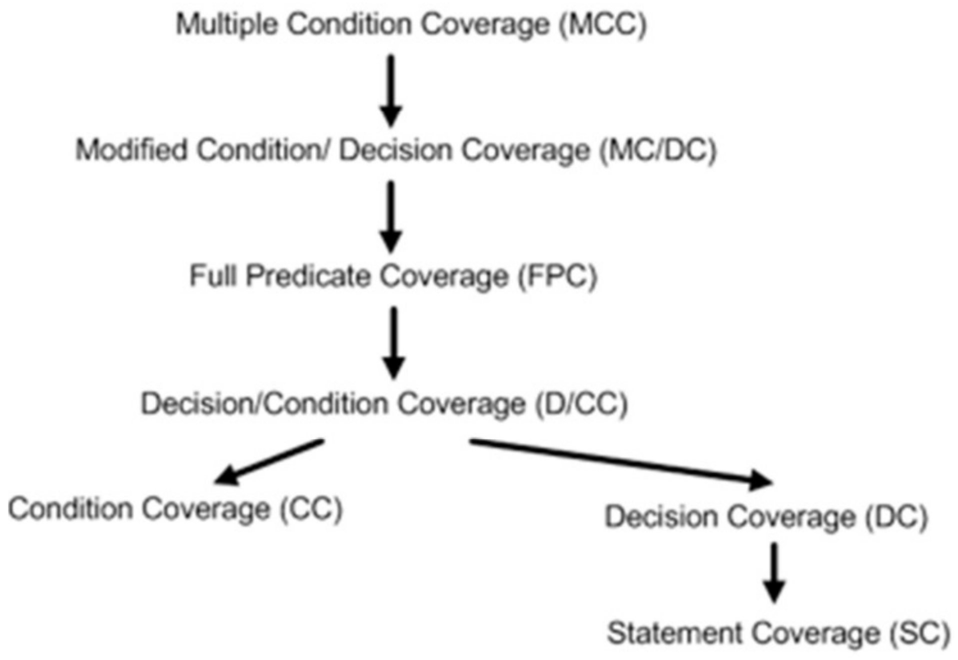

Multiple Condition Coverage (MCC)
Modified Condition/ Decision Coverage (MC/DC)
Full Predicate Coverage (FPC)
Decision/Condition Coverage (D/CC)
Condition Coverage (CC)
Decision Coverage (DC)
Statement Coverage (SC)

Figure 20: The hierarchy of control flow coverage criteria

In some case, we can define dedicated criteria as transition based coverage criteria as in Finite State Machines. These criteria are close to the previous criteria.

Data flow criteria
Data flow can be annotated with extra information regarding the definition and use of data variables. Informally, a definition of a variable is a write to the variable and a use of a variable is a read from it. For a given variable v, we say that (d, u) is a def-use pair if d is a definition of v and u is a use of v, and there is a path from d to u that is free to other definitions of v. So data flow criteria attempt to cover:

- **All-defs**: the all-definition criterion requires a test suite to test at least one def-use pair (d, u) for every definition d, that is, at least one path from each definition to one of its feasible uses.
- **All-uses**: the all-uses criterion requires a test suite to test all def-use pairs (d, u). This means testing all feasible uses of all definitions.
- **All-def-use-paths**: The all-def-use-paths criterion requires a test suite to test all def- use pairs (d, u) and to test all paths from d to u.

We have this hierarchy:
All-def-use-paths → All-uses → All-defs

In the context of security, model is used to compute the oracle of the test but the test need comes from security properties.

3.2.5.2 Adaptation of Behavioural MBT for testing security properties

One typical use of (behavioural) MBT is Security Functional Testing, which verifies whether the behaviour of a product or system conforms to the security features claimed by the manufacturer (i.e., the product does what it is supposed to do).
As show in Figure 21, the main adaptation concerns test selection criteria, which capture the test intention related to each security properties to be tested. The behavioural model formalizes the functional specifications with a special focus on security functions.
Many works define complementary criteria with external additional information on the model. Dynamic criteria help to explain about the sequencing of states or actions of the model. Several ways have been explored to express such criteria. For example in [12], the dynamic criterion is a sequencing of states expressed as a temporal logic (PLTL) property. It is a sequencing of actions expressed in the shape of an IOLTS in [39] and [119], or as a regular expression in [132].

One approach studied in the DIAMONDS project (see D1.WP3 – section 3), uses dynamic criterion, denoted as TP for Test Purpose, as a sequencing of states and actions. Its semantics is an automaton whose states are interpreted as state properties and whose transitions are labelled by action names. In our approach, the validation engineer manually describes by means of a test purpose TP how he intends to test the system, according to his know-how. In [126] a language based on regular expressions, to describe a TP as a sequence of actions to fire and states to reach (targeted by these actions).

The detail of this approach is given in section 3.2.6 below.

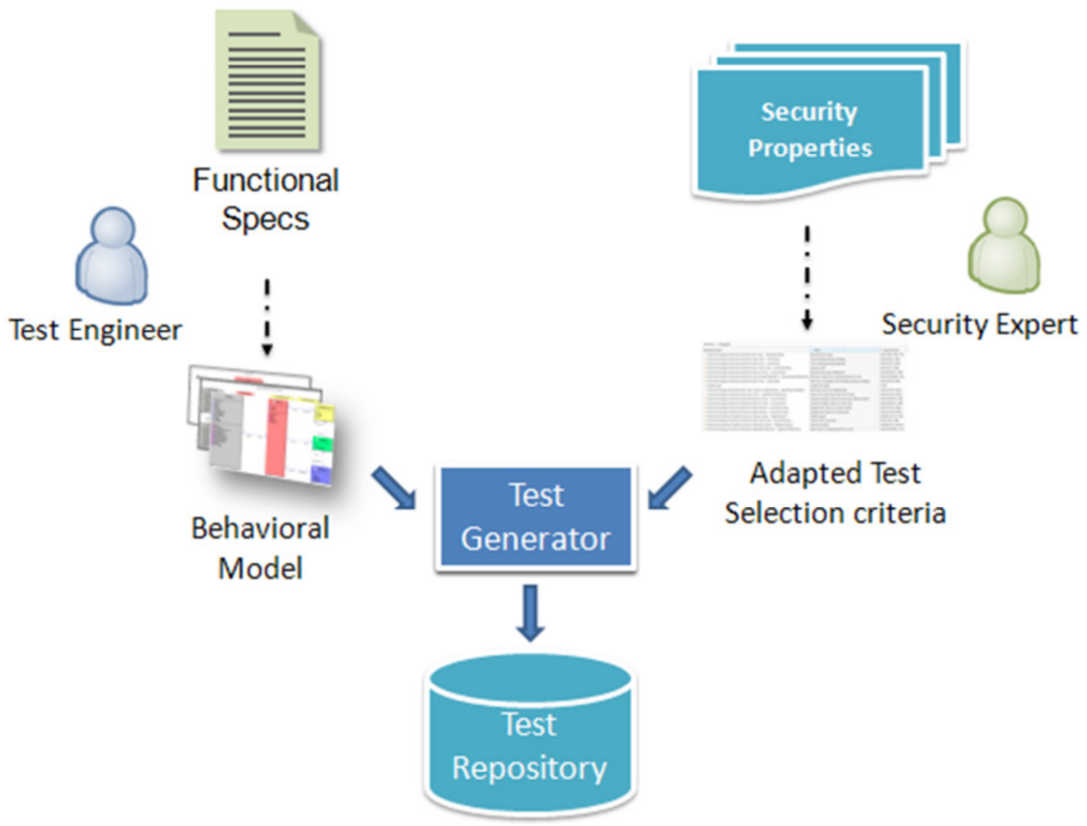


Figure 21: Adapted test selection criteria helps to formalize test intentions related to security properties

3.2.6 Smartesting approach for model-based security testing

In this section we explain the principle that Smartesting adopts for testing security properties. We use for that a Security Test Schema language that is described in Section 3.2.6.2. Further, we show two examples in the smart card domain.

3.2.6.1 Principle

Model-based testing makes use of test selection criteria that indicates how to select the tests to be computed from the model. These criteria usually ensure a given structural coverage of the model, such as all the states, or all the transitions, etc. Each test is a sequence of operation calls with parameter values, which yields a distinguished execution of the model.

The test results are predicted by the model. Our approach for testing security properties relies on defining additional selection criteria in the shape of test schemas. A test schema is a high level expression that formalizes a test intention linked to a security property. The test schema drives the automated test generation on the behavioural model.

Indeed, structural testing essentially provides control- and/or data-flow coverage of the model. The tests exercise the functionalities of the system by directly activating and covering the corresponding operations.

Industrial studies have proven the efficiency of the method to detect faults in an implementation. Nevertheless, structural selection criteria may become insufficient to exercise the SUT in tortuous situations. We think for example of particular scenarios where a security property could become violated due to an unusual sequencing of the operation calls. These scenarios can be described by means of a test schema, which we consider as a dynamic selection criteria in the sense that it orchestrates the successive calls of the operations of the model. The tests extracted from the model by means of a test schema are sequences of operation calls corresponding to the unfolding of that schema.

In our approach, the security requirements that a system must fulfil are expressed as a set of security properties. We propose test schemas as a means to exercise the system for validating that it behaves as predicted by the model w.r.t. these security properties. Based on his know-how, an experienced security engineer will imagine possible scenarios in which he or she thinks the property might be violated by an erroneous implementation. Then on the basis of this test intention, the security engineer will formalize test schemas to drive the automated test generation.

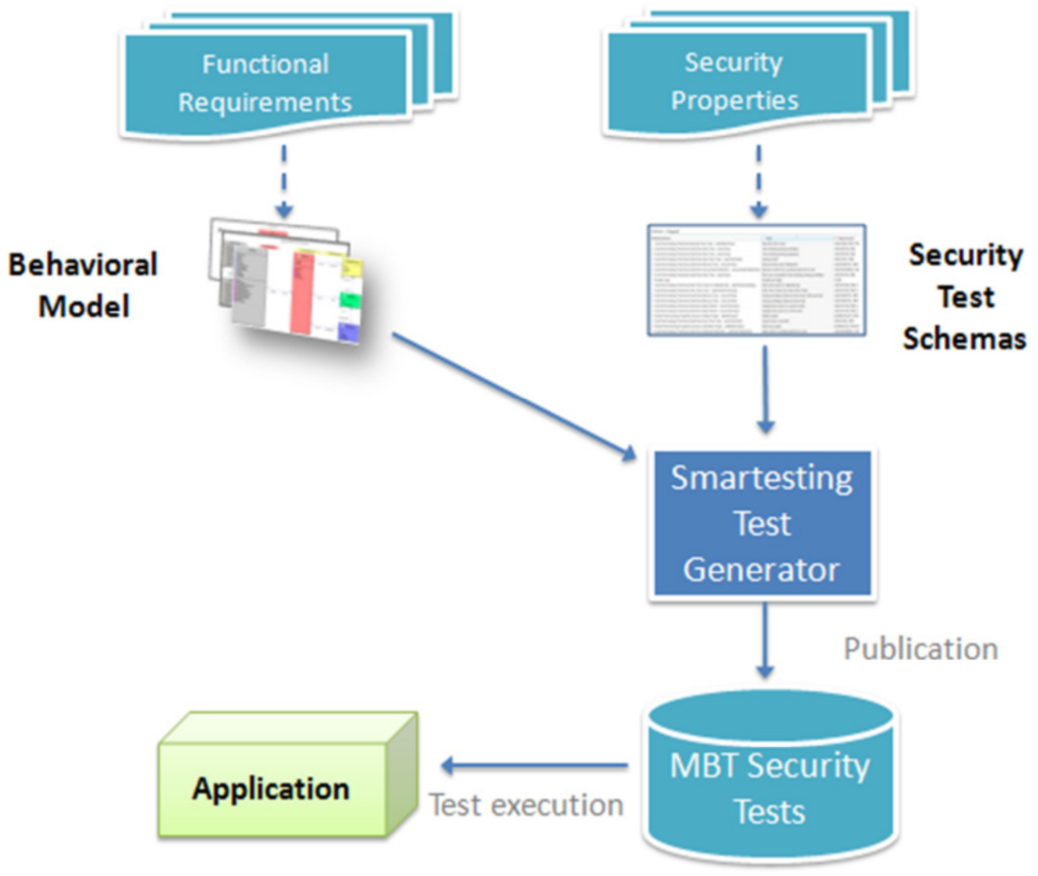


Figure 22: Smartesting approach based on security test schemas

We have defined the concepts of such security test schemas (see Figure 22) by means of a language. The language is based on regular expressions and allows the security testing engineer to conceive its test schemas in terms of states to be reached and operations to be called. The formal semantics of this language is defined in [149].

Based on this conceptual language, an operational language has been defined. This language is called Smartesting Security Test Schema language. We now present it and provide a couple of illustrative examples.

3.2.6.2 Smartesting Security Test Schema language

A dedicated Security Test Schema language editor has been implemented as a plug-in of the Smartesting CertifyIt software. Its aim is to provide a means to express security properties at a high level, close to a textual representation or by using usual computer programming paradigms. The expression of these properties allows for generating test specifications, called Test Case Specification - TCS that are high level scenarios from which tests will be generated by Smartesting.
The language relies on combining keywords, to produce expressions that are both powerful and easy to read by a validation engineer. We first define the keywords of the language and then its syntax.

In Table 1 we list the keywords of the language. For each keyword, we give its intuitive meaning.

Key word	Description
for_each	quantifier for an operation or a behaviour
From	to introduce a list of operations or behaviours
Then	a separator for sequencing the targets to be reached
Use	to introduce an operation, a behaviour or a variable to use
to_reach	to introduce a state to be reached
to_activate	to introduce a behaviour to be activated
state_respecting	to introduce a constraint that characterize a set of states
on_instance	to introduce an instance on which a constraint holds
any_operation	the set of all the operations of the model
any_operation_but	the set of all the operations of the model minus the ones whose list follows
Or	for a disjunction of operations or of behaviours
any_behaviour_to_cover	the set of all the behaviours of the model
any_behaviour_to_cover_but	the set of all the behaviours of the model minus the ones whose list follows
behaviour_activating	to introduce a list to be covered of behaviours tagged in the model
behaviour_not_activating	to introduce a list whose complementary must be covered of behaviours tagged in the model
at_least_once	repetition operator indicating to apply at least once the operation or behaviour previously specified
ant_number_of_times	repetition operator indicating to apply any number of times the operation or behaviour previously specified
$ something	variable prefix
REQ	to introduce a tag that corresponds to a requirement
AIM	to introduce a tag that corresponds to an aim

Table 1: Language key words

SCHEME	::=	(QUANTIFIER_LIST ,)? SEQ
QUANTIFIER_LIST	::=	QUANTIFIER (, QUANTIFIER)□
QUANTIFIER	::=	for_each VAR from (BEHAVIOR_CHOICE | OP_CHOICE)
BEHAVIOR_CHOICE	::=	any_behaviour_to_cover |
		any_behaviour_to_cover_but BEHAVIOR_LIST
BEHAVIOR_LIST	::=	BEHAVIOR (or BEHAVIOR)□
BEHAVIOR	::=	behaviour_activating TAG_LIST |
		behaviour_not_activating TAG_LIST
TAG_LIST	::=	{TAG(,TAG)□}
TAG	::=	REQ: tag name | AIM: tag name
OP_CHOICE	::=	any_operation | OP_LIST |
		any_operation_but OP_LIST
OP_LIST	::=	OPERATION (or OPERATION)□
OPERATION	::=	operation name
SEQ	::=	BLOC (then BLOC)□
BLOC	::=	use CONTROL (RESTRICTION)? (TARGET)?
CONTROL	::=	OP_CHOICE | BEHAVIOR_CHOICE | VAR
VAR	::=	$variable name
RESTRICTION	::=	at_least_once | any_number_of_times
TARGET	::=	to_reach STATE |
		to_activate BEHAVIOR |
		to_activate VAR
STATE	::=	state_representing ocl constraint on_instance instance name

Table 2: Grammar of language

The syntax of the language is defined by means of the keywords given in Table 1 and grammar given in Table 2. Roughly speaking, the language makes it possible to design test schemas as a sequence of steps, each step being composed of a set of operations (possibly iterated at least once, or many times) and aiming at reaching a given target (a specific state, the activation of a given operation, etc.).

To illustrate the use of the language, we give in this section two examples of security properties and their associated schemas. These examples refer to the Smart card Global Platform norm. They are related to the card life cycle. As show in Figure 23, a card has several states, the last one being the TERMINATED state. The Card Terminate privilege for an application allows setting the card to the TERMINATED state, therefore killing the card by permanently disabling all card content, management, and life cycle functions. In other words, once in the TERMINATED state, a card cannot revert to another state.

3.2.6.2.1 Example 1

The first security property for which we exhibit test schemas is expressed informally (i.e. in the natural language) as: “For any execution, whenever the card is put in the TERMINATED state by means of a set_status issued by a privileged application, then it should not be possible to revert to another state”.

Test Intention: A scenario to test this security property can be described informally as:

- Select an application with the Card Terminate Privilege;
- Set the status of the card to TERMINATED;

- Try all operations (to see if they behave as predicted by the model, i.e. by returning a status word of error).

Test schema: With the Smartesting CertifyIt Security Test Schema language, we express it as:

```
for_each $Operation1 from any_operation,
for_each $Operation2 from any_operation,
use $ Operation1 at_least_once to_reach state_respecting
(self.selectedApp.cardTermPriv = true)
on_instance "card" then
use set_status to_reach state_respecting (self.state = TERMINATED) on_instance "card"
then
use $ Operation2
```

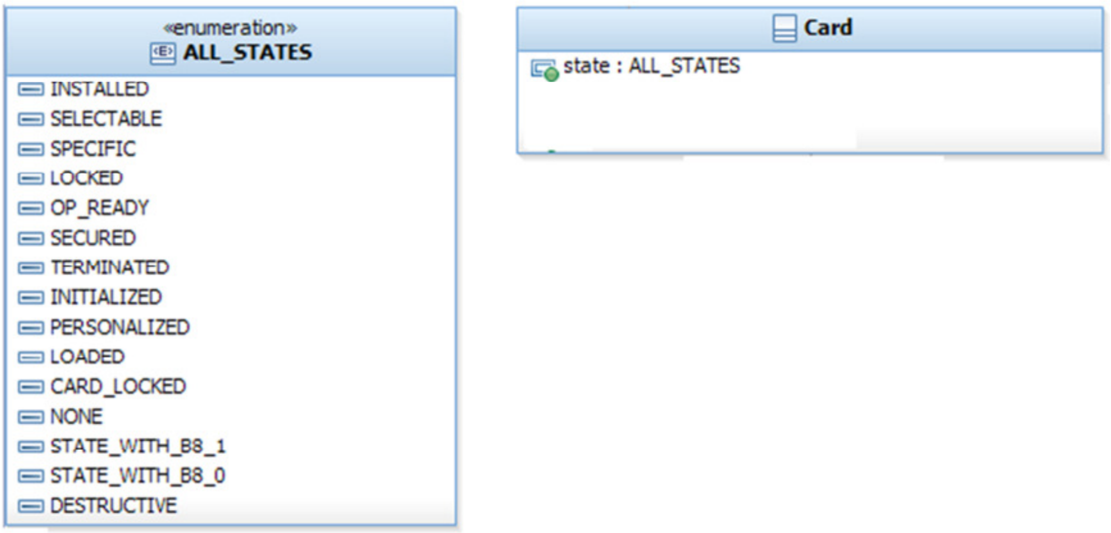


Figure 23: Card status

3.2.6.2.2 Example 2

The second security property that we exhibit is expressed informally as: “It should not be possible for an application that doesn’t have the Card Terminate privilege to switch the card life cycle state to TERMINATED, whether via a set_status command (if the application is an SD) or via the invocation of the GPSystem.terminateCard method”.

Test Intention: A scenario to test the nominal case of failure of this security property can be described informally as:

- Select any application without the Card Terminate Privilege;
- Set the status of the card to TERMINATED or invoke the GPSystem.terminateCard method.

Test Schemas: This scenario gives two schemas in the Smartesting Security Test Schema language.
The first one is:

```
for_each $Operation1 from any_operation,
use $Operation1 at_least_once to_reach state_respecting
(self.selectedApp.cardTermPriv6= true)
on_instance "card" then
use set_status to_reach state_respecting (cardState = TERMINATED) on_instance "card"
```

and the other one is:

```
for_each $Operation1 from any_operation,
use $Operation1 at_least_once to_reach state_respecting
(self.selectedApp.cardTermPriv6= true)
on_instance "card" then
use GPSystem.terminateCard
```

3.3 SUMMARY / OUTLOOK

Active testing techniques allow for testing implementations based on running test cases and comparing the outcome with the underlying expectations. Because of the fact that active testing requires a test suite the generation of test cases is of special importance. The approaches presented in this chapter are more or less based on available models or specifications, and requirements. In particular the chapter briefly describes the following methods: (1) Random testing and fuzzing, (2) Conformance testing based one models, and (3) Mutation-based testing.

In random testing test, cases are generated purely randomly or from models via traversing the model more or less in a random way. The conformance testing method uses the model to obtain test cases. In contrast to random testing the test cases are extracted from the model in a controlled and exhaustive way. Mutation-based testing uses fault models, i.e., mutants of models or implementations, to generate test cases. The idea is to search for test cases that allow distinguishing the implementation or specification from their mutants. All these techniques can be used also for testing security requirements.

4. RISK ANALYSIS FOR RISK BASED TESTING

The term software security testing characterizes activities to experimentally check the security features of software implementations. Software security testing checks software systems for confidentiality, integrity, authentication, authorization, availability, and non-repudiation. In general the software security testing activities can be divided into security functional testing and security vulnerability testing [88]. While security functional testing is used to check the functionality, efficiency and availability of the specified and carefully planned security functionalities and systems (e.g. firewalls, authentication and authorization subsystems, access control), security vulnerability testing directly addresses the identification and discovery of actually undiscovered system vulnerabilities that are introduced by security design flaws.

Unfortunately security testing, especially security vulnerability testing, lacks systematic approaches, which enable the efficient and goal oriented identification, selection and execution of test cases. Risk-oriented testing or risk-based testing characterize a methodology that makes software risks the guiding factor to solve decision problems during testing, e.g. the selection and prioritization of test cases.

The terms risk-oriented or risk- based testing are defined in different domains with different considerations and perspectives. All approaches have in common that tests are intended to check a specific risk or a set of risks that are previously identified through risk analysis. The majority of the approaches have been developed in the context of critical systems. Redmill [194][195] describes the use of risk analysis results for the overall test planning. Risk is quantified using the probability and the consequence of system faults. Amland [11] calculates the risk of individual software system functions and assesses the failure probabilities for these functions using several weighted factors like design quality, size, and complexity. The risk figures are used to prioritize tests and to decide which functions and subsystems are to be tested more extensively. Zimmermann et al. [258] presented a method that allows to automatically generating test cases for risk-based testing of safety-critical systems using Markov chain test models to describe the stimulation and usage profile of the system under test. Chen et al. [50] use risk analysis results to drive regression testing. In this approach the test cases are prioritized by their risk values so that the test cases with the highest-risk values are preferred for regression testing and Sauve [210] proposes to use the risk mitigation capabilities of test cases to rank the tests. Last but not least Stallbaum et al. [233] describe a model based approach that allows for the automatic derivation of system test cases from activity diagrams as well as their prioritization based on risk. Other attempts are guided economically, e.g. Bach [16] examines the economic impact of putting too much effort into the development and quality assurance process and uses risk analysis to optimize the development and software quality assurance activities. In the field of software security testing risk oriented testing approaches are described in [163][256]. Michael et al. [163] outline a general procedure for the testing of security features of software applications. Their risk-based approach is based on the analysis of fault models and attack trees and is used complementary to pure functional testing to integrate the environmental conditions of the software and software use in the testing process. Zech [256] describes a methodology for risk-based security testing in cloud computing environments. His approach uses dedicated and formalized test models to identify risks and specify negative requirements by means of so called misuse cases [73]. Moreover general technical recommendations on security testing techniques [170][97] propose the use of risk analysis results to guide security testing. The latter recommendations are very general in nature and describe in this sense no real method for risk-based testing. In summary we can say that except the approach described in [73] so far no model-based method exist that addresses the risk-based testing of security properties for software. The published approaches describe the basic idea to combine risk analysis and testing, not more. Specific techniques are provided first and foremost in the field of functional safety.

In Section 4.1 we provide a review on current information security risk definitions and the respective risk analysis techniques and processes. Section 4.2 provides a typically risk management policy that organisations often use as part of an Information Security Management System. Section 4.3 provides a more concise perspective on risk metrics. Section 4.4 provides an overview on state-of-the-art risk modelling techniques and section 4.5 reviews approaches to derive risks from vulnerability analysis. Section 4.6 gives a more concise description of risk-based testing techniques and to outlines how they can be used in the field of IT security. Our focus remains on considering especially model-based testing techniques. Section 4.7 summarizes the report.

4.1 IT SECURITY RISKS AND SECURITY RISK ANALYSIS

In the literature there are various informal and formal definitions of **risk** and **IT security risk**. The ISO 27000 [113] defines the general term **risk** as the *“combination of the probability of an event and its consequence”* and the more specific term **information security risk** as the *“potential that a threat will exploit a vulnerability of an asset or group of assets and thereby cause harm to the organization”.* Experts of the CERT define operational cyber security risks as “*operational risks to information and technology assets that have consequences affecting the confidentiality, availability, or integrity of information or information systems”* [46]. Alternatively, more formal definitions can be found in the context of the different risk analysis methods. For example in [147] the Department of Homeland Security generally defines risks related to information security with

“*Risk = THREAT * VULNERABILITY * CONSEQUENCE*”

and the DREAD Method, which has been mainly promoted by Microsoft, provides a little more differentiated but with respect to the level of abstractness not stronger definition with

“*Risk = (DAMAGE + REPRODUCIBILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5*”.

If we look at other risk assessment methodologies like OCTAVE [7], FAIR [125] and Trike [204] we will discover even more of such informal or semi-formal risk definitions. More formal approaches are discussed in the academia. Wawrzyniak describes [247] the use of qualitative methods for the identification and calculation of information security risks. He proposes well known and easy to use methods such as Annual Loss Expected (ALE), Return on Investment for a Security Investment (ROSI) or the Information Security Risk Analysis method (ISRSM). Especially ALE and ROSI address monetary business impacts of security threats and thus are particularly suitable for describing risks in an economically-driven environment. In [211][212][15] the use of regression models for risk analysis are discussed. Regression models can be used to estimate a measure, in this case information security risks, as a function of a set of independent variables. The approaches [212][15] propose to use measures on security strength as a basis.

Other approaches are more dedicated to integrate risk analysis with software modelling. Taubenberger and Jürjens [239] propose an approach for risk analysis based on business process models. They enhance these models with security requirements, information about critical processes and system boundaries. Wang et al. [245] especially addresses risk analysis for multi stage attacks. Their method is based on final state machine that model attack behaviour.

Risk assessment methodologies like ETSI TRVA [70], CVSS [147], STRIDE/DREAD [101], OCTAVE [7], FAIR [124] and Trike [204] may help to capture risks and the risk driving factors systematically but are often unspecific on how to measure the individual factors. The main purpose of these kinds of risk analysis methods is to provide systematic process and the definition of a consistent and unambiguous vocabulary for risk identification and handling. In this regards the CERT provides taxonomy on operational cyber security risks [46]. The taxonomy identifies sources of operational cyber security risks and organizes them into four classes. It distinguishes between risks established by actions of people, by systems and technology failures, by failed internal processes, or by external events. Each class is broken down into further subclasses, which are described by individual elements (e.g. “actions of people” is subdivided into “Inadvertent Actions”, “Deliberate Actions” and “Inaction”). The Factor Analysis of Information Risk (FAIR) [125] provides an information security risk taxonomy, which is comprised of two main branches according to the FAIR’s overall risk definition “*Risk = Loss Event Occurrence and Probable Loss Magnitude*”. Figure 24 shows the upper layers of the risk factor definition tree. The abstraction may continue to allow further refinements of the concepts.

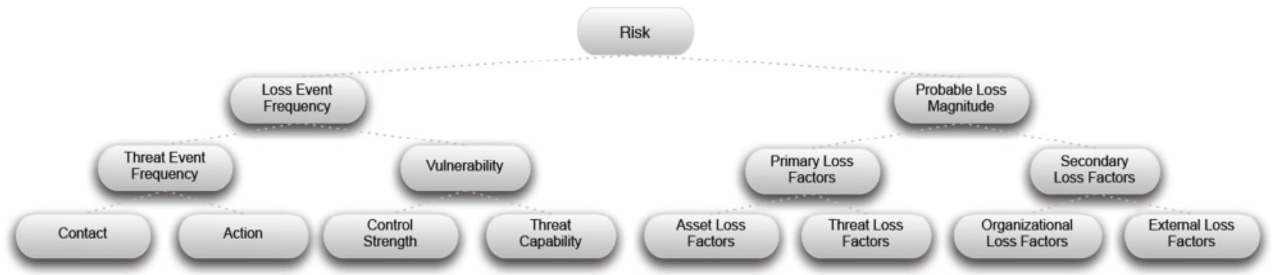


Figure 24: The higher-level risk factor abstractions within the FAIR framework.

From the process point of view, the risk assessment methodologies have differences in detail but mainly propose the same basic actions namely:

- identification of assets,
- threat analysis,
- vulnerability analysis, and
- the identification of mitigation strategies.

The OCTAVE method for example it defines the main tasks during risk assessment with threats identification, security measures identifications, definition of business impacts, and the definition of security measures costs and their standardized values. A step by step approach eases the estimations on the individual risk factors. It starts with the definition of asset-based threat profiles. In this phase the members of an organization identify important information assets, the threats to those assets and the security requirements of the assets. A second phase targets the identification of infrastructure vulnerabilities. Especially the information technology infrastructure is examined for weaknesses (technology vulnerabilities) that can lead to unauthorized action. The last phase is dedicated to the development of a security strategy. The information generated by the organizational and information infrastructure evaluations are carefully analysed to identify risks to the organization and to the organization’s mission as well as to identify countermeasures.

However, the general problem in the provision of IT risks is usually the lack of reliable information about how to qualify or quantify the individual factors that make the risk of an individual application, a network or system. The quality of this information, regardless of the chosen risk calculation formula, is the fundamental basis for a most accurate risk assessment.

Considering the risk assessment for individual organizations, individual application, individual networks or individual systems most approaches propose to use qualitative methods for risk assessment. NIST and the OCTAVE method [7] explicitly states to use scales with no more than 3 different values. Scales with a higher resolution won't make any further accuracy because of the expected uncertainties in the input variables.

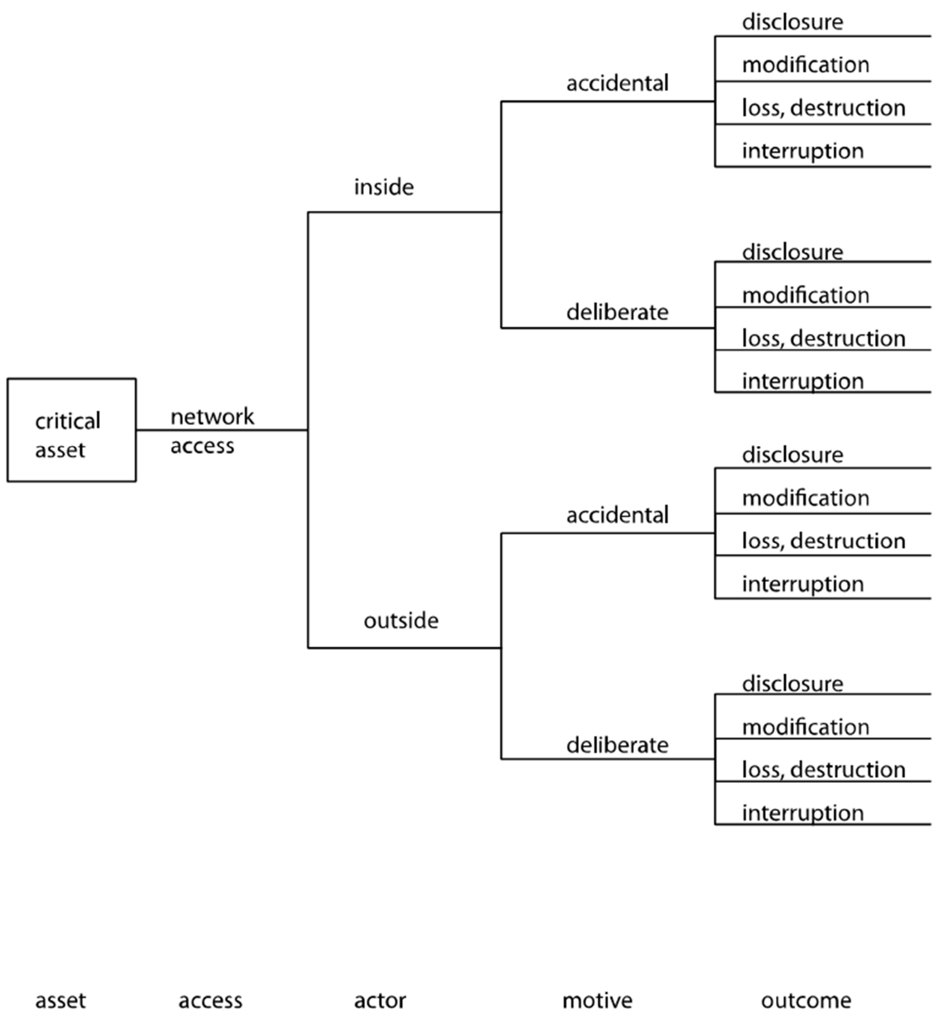


Figure 25: OCTAVE threat profile example

Quantified statistical valid data are available only on high level. There are currently different institutions like the CERT [47] or NIST [172] that maintain vulnerability databases and monitor attacks. These databases can be used to identify trends and the evolution of high level security risks. According to SANS [206], for example, the top security risks in 2009 have been unpatched client-side software, vulnerable web sites facing the internet, and a rising number of zero-day vulnerabilities. According to [98] these trends are continuing in 2010. Moreover the later states that especially the increased usage of consumer technologies and services in enterprise context (e.g. Facebook, Twitter etc.) has led to a multitude of security issues and that the attackers have become more organized and sophisticated regarding attack methods and concealment techniques. The applicability of such high-level quantitative analysis for accurate risk assessment of individual software products is difficult. On the one hand side vulnerability databases are certainly a good basis to determine the vulnerability of commercial of the shelf software products, systems and networks. On the other hand side they are inaccurate, because many companies still refuse to report security incidents and many systems differ in detail, so that complex vulnerabilities assessment are not transferable from one solution to another.

However Schneier [216] cited a study from Homeland Security, which asked for the best metrics to measure the value of security efforts in software development. The answers were not surprising. Cost of business interruption was the most helpful metric, cited by almost 64 percent of respondents. That metric was followed by vulnerability assessments (60 percent), benchmarks against industry standards (49 percent), the value of the facilities (43.5 percent), and the level of insurance premiums (39 percent). However, the main problems remain to information security risk assessment is that the total of risk factors are hardly measurable (e.g. the precision of measurements is limited, the results can often not be validated with empirical data), the risk factors are not completely under control of the system providers (i.e. the environmental factors are changing over time, new vulnerabilities are discovered and published, and the risk factors change over time (e.g. the system changes because of update updates, patches etc., the environment changes i.e. the attackers motivation, vulnerabilities, system accessibility). In the following we will more precisely address the fundamentals of risk analysis and risk-based testing by providing an overview on selected techniques for risk analysis and risk modelling.

4.2 INFORMATION SECURITY RISK MANAGEMENT POLICY

In the following, we provide a typically risk management policy that organisations often use as part of an Information Security Management System. The one given here follows ISO 27005 [116].

4.2.1 Principles

Risk analysis

Risk management relies on a documented analysis of risk obtained through a documented methodology. A yearly updated description of the status of risk contains information on threats, vulnerabilities, current security and estimation of the current level of risk.

Risk treatment plan

The risk analysis must lead to an action plan in order to reduce the risks to an acceptable threshold. This plan:

- Proposes a pool of security measures relevant for the risk reduction;
- Classifies those measures by priority;
- Contains other common safeguards that are not implemented and justifies the ineffectiveness of those safeguards for the organisation;
- Defines when risks will become unacceptable if planned safeguard are not operational as planned.

A quantitative and qualitative analysis method

The risk analysis is performed using both a quantitative and qualitative method, based on estimations that have to be confirmed by the Managing Director:

- Qualitative aspect: Use of scales based on people's backgrounds (working groups with key people associated to one or two external experts) and their personal experience of risk to qualify list of typical threats and asset and identify items requiring dedicated countermeasure;
- Quantitative aspect: Rating probability and impact on different asset of some typical risk scenarios in order to estimate which missing good practices are economically viable for the organisation.

Method based on financial computation

The "Risk Management" policy is based on following indicators:

- ALE (Annual Loss Expectancy): Estimation of the yearly financial losses resulting from the current risks expressed in Euro per year;
- ROSI (Return On Security Investment): similar to ROI (Return On Investment) but considering any reduction of risk as return.
- Risk reduction factors, a factor indicating how much a risk is reduced if addition safeguards are deployed.

These estimations constitute the background of the risk analysis. They define:

- What are the financial losses of some organisation's departments, if they do not consider recommendations;
- Maximum saving obtained by additional security;
- How far the risk can be reduced until economically unviable measures have to be taken.

These estimations are used to assess the relevance of the security measures provided by ISO 27002 [115] and are considered as relevant input to define the Statement of Applicability (SOA) and the risk treatment plan.

4.2.2 Processes

The following figure highlights the processes of the risk management, which have been taken from ISO 27005 [116].

Context establishment

This process specifies the initial criteria (risk assessment, impact, risk acceptance, resource availability…), the target and the scope of the analysis. It describes the environment and the purpose of the risk management process.

Risk identification

Risk identification aims at determining what could cause losses and understand how, where, and why these losses could happen. This phase prepares the actual risk assessment. It includes the following steps:

- Identification of assets;
- Identification of threats;
- Identification of existing security measures;
- Identification of vulnerabilities (inherent of identification of the risk specificity);
- Identification of consequences (considering the impact criterion).

This process aims at defining the list of assets whose risks must be managed.

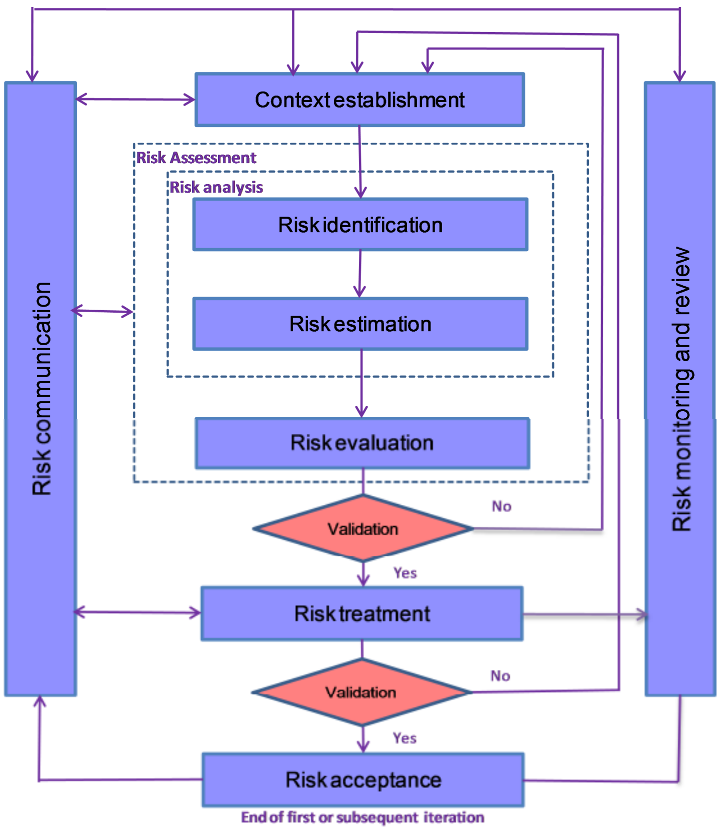


Figure 26: Risk analysis process

Risk estimation

Risk estimation includes several phases:

- Choice of the methodology;
- Assessment of the consequences;
- Assessment of the likelihood of threat;
- Assessment of risk level.

Risk evaluation

This phase uses the understanding of the risk obtained thanks to the risk analysis. Besides the estimated risks, it considers the contractual obligation, legal aspects and regulations issues, as well as risk acceptance criteria.

Risk treatment

This last step proposes the measures to setup following recommendation of the chosen standard. This means that we need to sort measure and define:

- Security measures to consider for the computation of the treatment plan;
- Order of importance and priorities;
- Implementation cost.

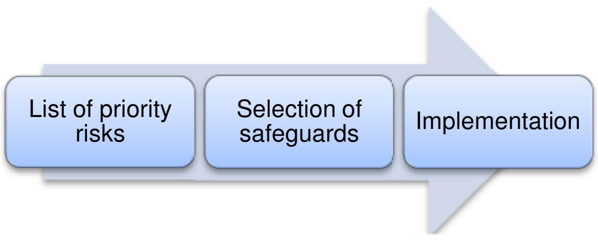


Figure 27: Flow to treat risks

Risk treatment is not assured by analysis tools. However, they enable to help the organisation to make choices about security in order to treat risk through help to the setup of an action plan.
This phase can be assimilated to the Risk treatment phase of ISO 27005 [116].
The risk treatment process proposes 4 options:

- The risk reduction, which consists in reducing risk by selecting relevant safeguard and security measures.
- The risk retention, which consists to accept the current risks without any action.
- The risk avoidance, which consists to renounce to continue the activity or the domain generating risks.
- The risk transfer, which consists to transmit risk to a third party (e.g. assurance, subcontracting).

Obtained residual risks must be validated by the Senior Management of the organisation.

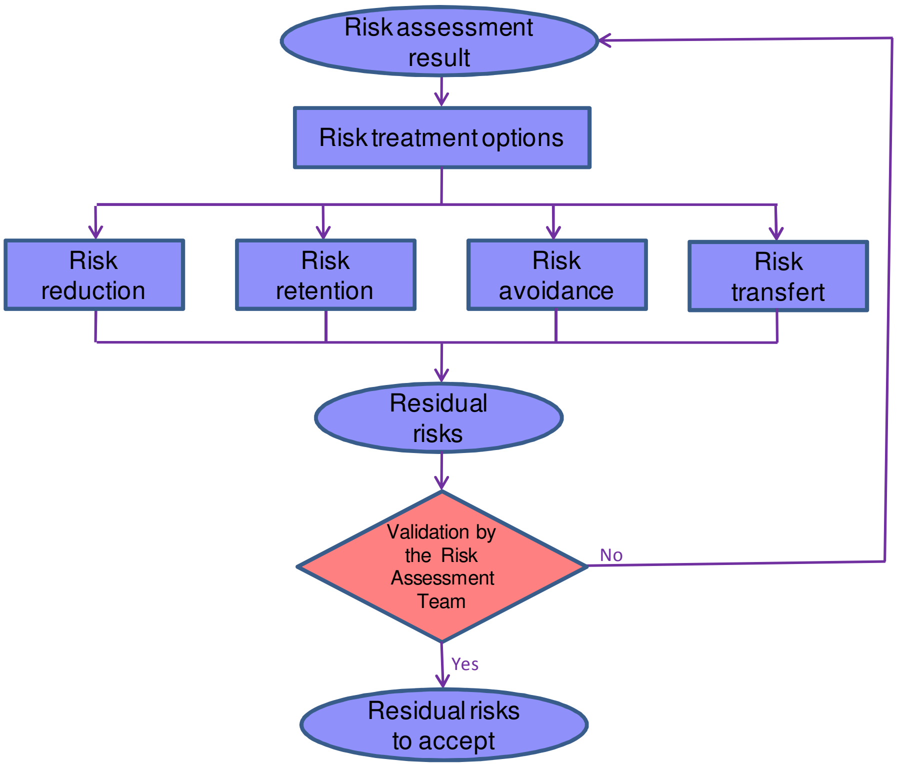


Figure 28: Description of Risk treatment process

Risk acceptance

Risk acceptance means validation of the choices made during the risk treatment by the Senior Management of the organisation.

Risk communication

This is a continuous process enabling to exchange and share risk information between decision maker and other stakeholder. The risk communication has the following objectives:

- Reduce misunderstanding with decision maker;
- Improve coordination to react to security events;
- Share results of the risk assessment and present the treatment plan;
- Obtain new knowledge about security;
- Improve the sensitisation.

Surveillance and risk reassessment

This process consists of monitoring and reassessing risks:

- Value and category of the assets;
- Impact, threats, vulnerabilities, likelihood;
- External elements (legal context, environmental);
- Modification of the approach of the risk estimation (impact criterion, assessment, risk acceptation);
- …

The update is made as recommended by ISO 27006:

- Specific review each year;
- Complete review every 3 years.

4.2.3 Risk context of the organisation

Estimation and risk treatment

Security objectives are defined with regards of a risk analysis, so that useless risks are avoided and that security costs stay appropriate. Bliin does not aim at completely reducing risks, but take the responsibility for residual risks. However, these risks will have to be correctly understood, formally analysed, documented and accepted by authorised staff.

General considerations

All analysis and propositions are made by the risk assessment team composed by key people and by one or two external experts. The result is presented as a report that the management of Bliin must approve.
Risk management aims at supporting the setup of the required security level in order to efficiently complete the mission of the different services of the organisation. This risk management proposes an improvement plan of the security which has to be validated by the direction of the organisation.
The chosen approach for the risk assessment is a combination of following methods:

1. Risk identification according to the threat cartography provided by the MAGERIT methodology and documentation of the dedicated countermeasures during expert meeting;
2. Quantitative estimation of risks by asset, according to the procedure “Risk analysis method”;
3. Assessment of the implementation level of security measures proposed by ISO 27002 [115];

4. Estimation of the profitability of the security measures of ISO 27002;
5. Weighting of the security measures in order to build a treatment plan of the risks.

Base criteria

Risk assessment criterion

Risks are assessed considering:

- The importance of the information essential to assure the operation of the target;
- Direct financial consequences;
- All three aspects of the security, confidentiality, integrity, and availability, are considered.

Impact criterion

Impact of a security incident is estimated in Euros. When this estimation is too uncertain, impact is estimated considering a category: vital, extremely serious, very serious, serious, minor, insignificant. This is a more or less logarithmic scale with fixed cost for each level.
Estimates reputation loss and damage to customers for whose data have been compromised are criteria proper to the service provided by Bliin.

Risk acceptance criterion

During expert meeting, threats and risks logically ordered according to different criteria (MAGERIT method) are considered and assessed according to the level ++ and + in case organisation is very exposed, n for a normal exposition, and - or -- whether the organisation is less exposed than the normal level.

1. Risk whose threats are assessed as - or -- are accepted and not explicitly considered in the treatment risk;
2. Risks whose threats are assessed as ++ or + require an identification of countermeasure. Risks must be accepted by the direction, after assessment of the feasibility of the identified countermeasures;
3. Risks considered as *normal* are accepted if the Senior Management is aware of the risk estimation and if:
   a. Either all security measures anticipated in ISO 27002 and assessed as profitable are planned;
   b. Or there are enough measures planned so that the estimated annual loss expectancy is lower than a given percentage of the turnover.

Target

Global risk analysis of the organisation is the base of the definition of requirements and security measures.
Target of this risk analysis is the same than the target of the security policy.
Risk analysis of the organisation is split under several risk analysis to consider its different entities. In this case, the direction review will stop up that all activities of the target are covered by at least one risk analysis.
Each of these analyses will document the flowchart, activities, the possible excluded assets and the general conditions (work processes, functional organisation, legal aspects…) in the standard reporting the conclusions of the risk analysis.

4.2.4 Documents linked to the Risk Management policy

Risk Management policy include separate document, which enable to treat more deeply some domains. These documents are listed below:

- The standard “Risk analysis method” defines the tool and the methodology used by the organisation to treat its risks;
- The standard “User guide” describes the utilisation of the risk analysis tool selected by the organisation;
- Standard “Risk analysis…” reports on the global risk analysis done on the different entity of the organisation;
- Standard “Statement of Applicability” (SOA) defines the applicable security control.

4.3 RISK CALCULATION AND ASSESSMENT TECHNIQUES

Looking at current techniques for risk calculation and assessment, two distinct approaches can be identified. Qualitative risk analysis is done using estimations of cost, impact and likelihood. Qualitative analysis uses real monetary values and is normally more complex. Every security incident needs to be estimated concerning its monetary consequences, which is not easy. Currently qualitative techniques are more common than quantitative techniques. The reason for this is mostly that it is much easier to use estimations and measures with qualitative scales than with quantitative scales. Moreover the whole calculation mechanism is easier to be accomplished for qualitative approaches.

4.3.1 Qualitative techniques

Every risk oriented analyzing method starts with building up a special team for analyzing the system or the requirements. The main focus of the analyze is

- Identification of hazards
- Analytical partitioning down to basic faults
- Prioritization and categorization of detected faults, threats, hazards and vulnerabilities
- Detection of required countermeasures

For these goals the team can use different existing methods:

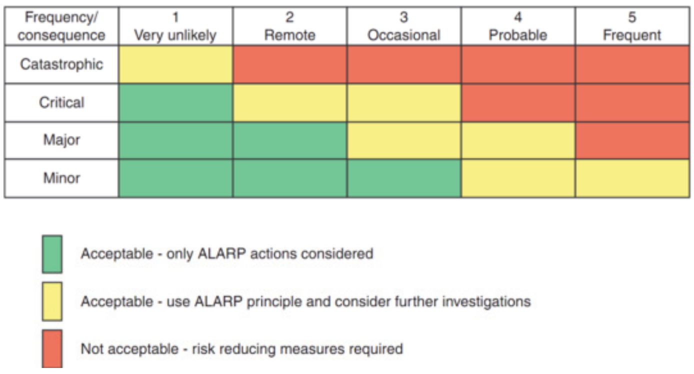
To identify hazards the **HAZOP** [191] (hazard and operability studies) works as follow: The team detects main system parameters and changes them into critical values depending on so called “guide-words” and discusses or analyzes the impact for the system. Also the causes and consequences of such deviation are checked.

Another “what-if” approach is the method **FME(C)A** [32][168] (failure mode and effects and critically analysis) that helps to identify possible failure modes: The functions of a system will be observed separately and the team tries to detect how they can fail and what consequences the failure modes can ensure. Additionally, and that is the meaning of the C for critically in the method name, the failure mode will be qualified.

A third approach is to use existing catalogues of known faults and vulnerabilities. **STRIDE** [101] is a list of six best known and most used attacks and more than 80 percent of all IT security attacks are results of single or combination of them (STRIDE stands for Spoofing Identify, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege).

All these methods are suitable to identify possible threads. In the next step such threads have to be analyzed in detail. **FTA** (fault tree analyze) divides complex fault top-down to basic events in a tree structure to identify the cause of the failure. **ETA** (event tree analyze) works bottom-up to the system behaviour and analyzes the consequences. As a connection of both methods the team can use **CCA** (Cause-consequence diagram). Starts from a thread the causes (top-town) and the consequences (bottom-up) will be analyzed simultaneously.
A variant of fault trees are **attack trees** [152]. These trees are security specific and start from a complex attack down to simple attacks (basic events) and all simple attacks have a scaled attack probability like "rare" or "certain" to get the likelihood of one attack path. Instead of probability you can also refer a cost to an attack.

PHA [192] (Preliminary Hazard Analysis) has the main focus of identification of hazards but in fact it is only a connection of many known methods for detection (HAZOP and FME(C)A are also used). But additionally a suggestion of how to prioritize these threads is given: The hazard attributes are Frequency (like "very unlikely", "remote", "occasional", "probable", "frequent", be scaled on logarithm scale) and Consequences (like "minor", "major", "critical", "catastrophic", depends on the accident size or the number of accidents the thread causes) and added into a risk matrix.



Frequency/ consequence	1 Very unlikely	2 Remote	3 Occasional	4 Probable	5 Frequent
Catastrophic	🟨	🟥	🟥	🟥	🟥
Critical	🟩	🟨	🟨	🟥	🟥
Major	🟩	🟩	🟨	🟨	🟥
Minor	🟩	🟩	🟩	🟨	🟨

🟩	Acceptable - only ALARP actions considered
🟨	Acceptable - use ALARP principle and consider further investigations
🟥	Not acceptable - risk reducing measures required

Figure 29: PHA risk ranking (Source: [192])

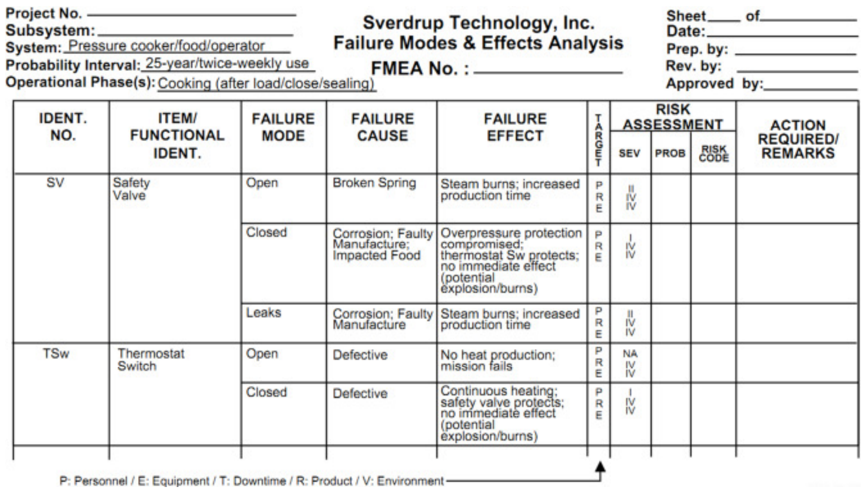
Another known proposal is the **DREAD** [101] method: The identified threads are evaluated in five categories and the sum of all is a proposal for a priority. (DREAD stands for Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability). Last not least, **CVSS** [159] (Common Vulnerability Scoring System) can be used to categorize vulnerabilities. The System works with different metrics in different analyzing steps. Next to an ordered category and priority for the vulnerability a countermeasure for equivalent vulnerabilities is also suggested.

4.3.1.1 Failure Mode and Effects Analysis (FMEA/FMECA)

The method **FMEA (Failure Mode and Effect analysis)** resp. **FMECA (Failure Mode, Effects and Critically Analysis)** [32][168] was developed in 1995 for safety risk assessment in the industry for power plant construction and chemical process systems like most of the safety analysis tools e.g. HAZOP. FMEA is also known as "Fault Hazard Analysis" (FMEA and FMECA is in mostly only called FMEA because the critically analysis is included in every FMEA analysis).

FMEA can be summarized as the procedure of dividing the system into smaller pieces to find a piece that results in intolerable/undesirable loss if a failure occurs (dividing is needed to detect the part of the system that gets into the undesirable state).

The **Failure Mode Analyze** represents the checking of individual elements or operations for failure modes, i.e. the way in which the element faults. For example the element “cable” has the failure modes “to stretch”, “to break”, “to fray” and so on. The consequences of the failure mode for the system are represented by the **Failure Effect Analysis**. If no consequences exist the failure mode is no failure. Both will be analyzed together.
The additional step called **Critically Analyze** assigns probabilities and severities to the failure modes.



Project No.
Subsystem:
System: Pressure cooker/food/operator
Probability Interval: 25-year/twice-weekly use
Operational Phase(s): Cooking (after load/close/sealing)

Sverdrup Technology, Inc.
Failure Modes & Effects Analysis
FMEA No. :

Sheet of
Date:
Prep. by:
Rev. by:
Approved by:

IDENT. NO.	ITEM/ FUNCTIONAL IDENT.	FAILURE MODE	FAILURE CAUSE	FAILURE EFFECT	TARGET	RISK ASSESSMENT			ACTION REQUIRED/ REMARKS
						SEV	PROB	RISK CODE	
SV	Safety Valve	Open	Broken Spring	Steam burns; increased production time	P R E	II IV IV			
		Closed	Corrosion; Faulty Manufacture; Impacted Food	Overpressure protection compromised; thermostat Sw protects; no immediate effect (potential explosion/burns)	P R E	I IV IV			
		Leaks	Corrosion; Faulty Manufacture	Steam burns; increased production time	P R E	II IV IV			
TSw	Thermostat Switch	Open	Defective	No heat production; mission fails	P R E	NA IV IV			
		Closed	Defective	Continuous heating; safety valve protects; no immediate effect (potential explosion/burns)	P R E	I IV IV			

P: Personnel / E: Equipment / T: Downtime / R: Product / V: Environment

Figure 30: Example for FMEA with analyzing an heirloom pressure cooker

Source: (http://icecube.wisc.edu/~kitamura/NK/Flasher_Board/Useful/FMEA.pdf)

The focus of FMEA is the detection of failure modes but not the causes. That’s why **FTA** is used additionally: FTA transforms the complex failure modes into simple failures to detect the cause of failure. So both complementary methods are connected together to analyze the hazard.

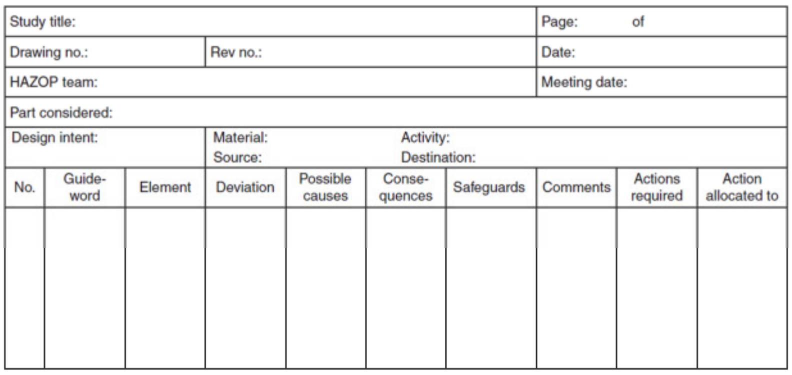
4.3.1.2 HAZOP (Hazard and Operability)

The method **Hazard and Operability (HAZOP) analysis** [191] also focuses on the impact for the system in case of deviations of parameters/elements/values of the system. So-called “guide-words” are used like “no”, “more”, “less”, “as well as”, “reverse”, “other than” and so on and are combined with parameters and generate new parameters, for example:

Value Time: time + “more” = too long, too late; time + “none” = sequence step skipped;
Value pressure: pressure + “less” = low pressure; pressure + “none” = vacuum

With the combined new parameters deviations and possible impacts are detected and rated and also the cause, the consequences and the required action for prevention are discussed directly.
The advantage is to think about new parameter values that are often not born in mind. But it is also a disadvantage because of reducing the analysis on combinations of defined “guide-words” only.

Another aspect of HAZOP is the separation between **hazard** as catastrophic results of an operation or an action and **operability** as negative impact as result of an operation inside the design.



Study title:							Page: of		
Drawing no.:			Rev no.:				Date:		
HAZOP team:							Meeting date:		
Part considered:									
Design intent:			Material: Source:		Activity: Destination:				
No.	Guide-word	Element	Deviation	Possible causes	Conse-quences	Safeguards	Comments	Actions required	Action allocated to

Figure 31: a HAZOP analyze table (Source: [191])

In case of systems with moderate or high complexity it is tedious to analyze all failure modes with all possible combinations of circumstances. The combination of Model Checking and behaviour trees provides FMEA with automated support [254][87].

The failure modes are integrated into the behaviour trees and are checked by model checking.

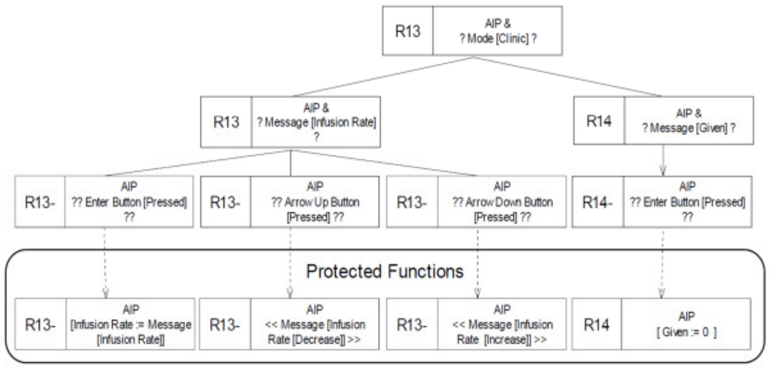

R13
AIP & ? Mode [Clinic] ?
R13
AIP & ? Message [Infusion Rate] ?
R14
AIP & ? Message [Given] ?
R13-
AIP ?? Enter Button [Pressed] ??
R13-
AIP ?? Arrow Up Button [Pressed] ??
R13-
AIP ?? Arrow Down Button [Pressed] ??
R14-
AIP ?? Enter Button [Pressed] ??
Protected Functions
R13-
AIP [Infusion Rate := Message [Infusion Rate]]
R13-
AIP << Message [Infusion Rate [Decrease]] >>
R13-
AIP << Message [Infusion Rate [Increase]] >>
R14
AIP [Given := 0]

Figure 32: Function accessible modelled with behaviour trees (Source: [254])

4.3.2 Quantitative techniques

In this section we introduce a number of quantitative security risk metrics. The most common metric is Annual Loss Expected (ALE). Others metrics, i.e. Security savings and security benefit (SB), and Return On Investment (ROI, IRR) are somehow derivatives of ALE that highlight certain monetary aspects like the invest in security measures or the overall costs of security measures. A good summary of state of the art quantitative security metrics can be found in [211].

The main problem of quantitative security metrics remains the lack of statistical reliable historic data and thus the uncertainties in forecasting future values. There is a significant difference between probabilistically events as they are accepted as such in the calculation of risk of natural disasters and IT security events. The latter are the result of intentional processes. The attackers learn to identify and exploit the weaknesses of a system. In general the frequency and severity of security incidents change over time and is not likely that they remain stationary in an environment, which is driven by advancing technology

4.3.2.1 Annual Loss Expectancy

The most common measure for the risk is Annual Loss Expectancy (ALE). ALE is characterized by the anticipated monetary loss that can be expected for an asset due to a risk over a one year period. It is defined as the product of the expected yearly rate of occurrence of a harmful event times the expected loss resulting from each occurrence.

$$ALE = ARO \times SLO$$

Thus, ARO is the Annualized Rate of Occurrence that is the expected frequency of event occurrence calculated for one year. SLE is the Single Loss Expectancy that is the monetary value expected from the occurrence of a risk on an asset.

In literature there are several other mathematical model describing ALE [31]. For example

$$ALE = \sum_{i=1}^{n} I(O_i)F_i$$

expresses the sum of all harmful outcomes $I(O_i)$ multiplied with their respective frequency F_i. Apart from which model is used to describe ALE, the meaning remains the same. ALE is used to estimate the loss in order to minimize it to an acceptable amount. The ALE metrics is the overall basis for a number of other metrics like Security Savings and Benefits (SSB), Return on Invest (ROI) or Internal Rate of Return (IRR).

4.3.2.2 Security Savings and Benefit (SSB)

The “Security Savings and Benefits” approach is described in [227]. It mainly reflects the benefits of security investments and relates them to the reduction in expected costs of security incidents. It uses an ALE based methodology to calculate the security savings S. A security saving is defined as the difference between ALE calculated without a security safeguard ($ALE_{baseline}$) and ALE calculated with a security safeguard ($ALE_{with\,safeguard}$).

$$S = ALE_{baseline} - ALE_{with\,safeguards}$$

The benefits BOS are calculated as the sum of the security savings S and additional profits from new ventures that could have been securely introduced with support of the security safeguards.

$$BOS = S + (profit\ from\ new\ ventures)$$

4.3.2.3 Return of Invest

Return on Invest (ROI) is a metric that has gained popularity because security investment will be received not only in the first year, but in all subsequent years. ROI for security invest is also known as security ROI. Security ROI is defined by Blakley [27] as the amount of the annual benefit over its cost. The benefit is calculated as it was specified in 4.3.2.2, by adding the sum of the security savings S and additional profits from new ventures. Security ROI is defined as

$$ROI = \frac{BOS}{COS}$$

where BOS is the benefit of a safeguard and COS is the cost of a safeguard.

Another simple equation for calculating the Return on Investment for a security investment is given in [226].

$$ROSI = \frac{(RiskExposure\ \times PercentOfRiskMitigated) - Solution\ Cost}{SolutionCost}$$

Similarly, in [92] this notion has been called relative ROSI:

$$ROSI_{rel} = \frac{\Delta ALE - Cost}{Cost}$$

4.3.2.4 IRR

The economic rate of return, usually called the internal rate of return (IRR), is the appropriate metric for evaluating investments, including information security investments. Gordon and Loeb [84] claim that companies should discard the usual ROI formula and instead use the Internal Rate of Return. In contrast to ROI IRR incorporates discounted cash flows for investments that have different costs and benefits in different years. Let C_0 be the initial cost of an investment and let C_t and B_t be the respective costs and benefits in year t, IRR can be resolved by solving the following equation.

$$C_0 = \sum_{t=1}^{n} \frac{B_t - C_t}{(1 + IRR)^t}$$

4.4 RISK MODELLING TECHNIQUES

In this section, we give a comprehensive overview of different approaches to risk modelling. Then we describe three approaches in more details. These three approaches all have a solid mathematical foundation that supports analysis which may be useful when combined with test-based approaches.

4.4.1 Overview of risk modelling approaches

Current approaches to system modelling and analysis can be broadly distinguished in two categories. The first category comes from the field of system development and analysis and focuses on modelling the intended or actual behaviour of a system. Typical examples include UML, SDL, and various forms of automata, state machines and process algebras [174][110][99]. Common for these approaches is that they have only a single concept of behaviour, representing either the desired or the actual system behaviour, depending of the purpose of the model. Although certain sequence diagrams notations, such as UML sequence diagrams [174], include a separate concept of negative behaviour, this is aimed at explicit representation of prohibited system behaviour in cases where the general description is incomplete, rather than an explicit notion of malicious or harmful behaviour. An exception is misuse case diagrams [224], which are based on UML use case diagrams and allow explicit representation of malicious or harmful behaviour as well as desired behaviour. However, these are to a large degree based on textual descriptions and therefore less suited to any kind of formal analysis.

The second category is more typically associated with the field of risk analysis, and is usually focused on modelling potential undesired behaviour. A good overview of different approaches can be found in [108]. The techniques are typically based on some kind of trees or graphs. The fault tree notation used in fault tree analysis [108] is used to describe the causes of an event. The top node represents an unwanted incident or failure, while events that can lead to the unwanted incident are modelled as leaf nodes or intermediate nodes combined by AND and OR gates. The probability of the top node is calculated based on the probability of the leaf nodes. Event trees [105] are similar to fault trees, but here the branches from the top node represent different possible consequences of the top node, rather than causes. Cause-consequence diagrams [171] combine the features of fault trees and event trees, by including both the causes and consequences of an event. The OCTAVE method for security analysis has its own tree notation [246] which has much in common with both event trees and fault trees. The notation supports the modelling of both the causes and the outcomes of incidents. Other tree-based approaches include attack trees [256][215], defence trees [26].

Among the graph-based notations, Bayesian networks [139][21][193] and Markov models [102][107][129] are perhaps the most well-known. Both of these have a solid mathematical foundation and can be used as tools for probabilistic analysis. A Bayesian network is a directed, acyclic graph whose intermediate nodes represent causes or contributing factors to the top node. Markov models are more suitable for showing the operation modes of a system that may transit between states, rather than a chain of events related to a security attack which is more likely to be a one way chain. Block diagrams [193][106] are often used in reliability assessments and show how the components of a system are parallel or serial, which can be useful for identifying weak points. Microsoft has developed an approach called Threat Modelling [164][237] which proposes data flow diagrams to graphically represent the target of analysis. Based on such diagrams, the target is decomposed into components that are analysed with respect to identified threats. The method used threat trees for the modelling and analysis of attack paths. CORAS threat diagrams [61][142] describe how different threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents, and also which assets are affected. Likelihood can be assigned to threat scenarios, unwanted incidents, and relations between them.

4.4.2 Bayesian Networks

A Bayesian network [139][21][193][183] is a directed, acyclic graph. The intermediate nodes represent causes or contributing factors to the top node, which in Figure 33 is a “system failure” (taken from [183]). A Bayesian network is both a graphical and a probabilistic model that may be used to for instance predict the number of faults in a software component. In Figure 33 the causes that contribute strongest to the event (A1-A3) are placed directly before the event. The causes are grouped into three categories: organisational factors, human factors and technical factors. When a Bayesian network is analysed quantitatively, each node holds a table with a probability distribution reflecting its parent nodes. For any manipulation of the probabilities of the nodes, the effects both forwards (towards child nodes and the top node) and backwards (towards parent nodes) can be computed. A Bayesian network can be utilized both quantitatively and qualitatively. If the Bayesian network is analysed qualitatively, it provides relations between causes and effects. When analysed quantitatively, one uses its powerful mathematical model for computing probabilities, which is not only based on the probabilities for the leaf nodes like in fault trees, but also on intermediate nodes.

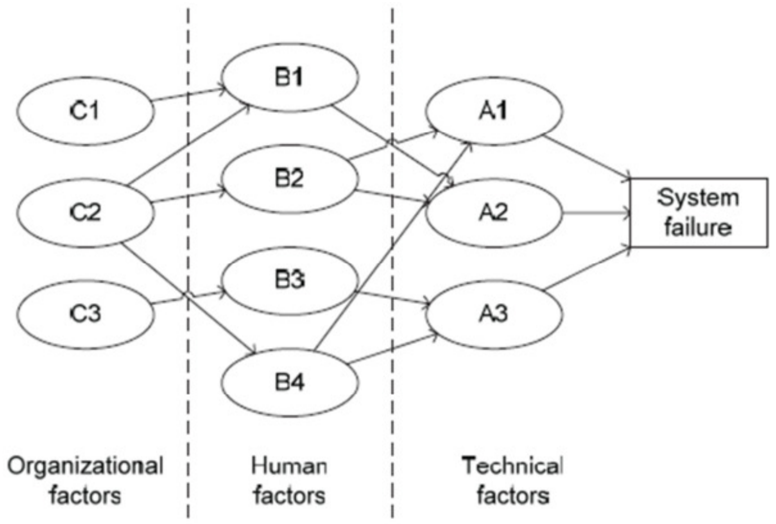


Figure 33 Example of a Bayesian Network

4.4.3 CORAS Risk Modelling Language

The CORAS risk modelling language [142] has been designed to support communication, documentation and analysis of security threat and risk scenarios. It was originally defined as a UML profile, and has later been customised and refined in several aspects, based on experiences from industrial case studies, and by empirical investigations. It consists of the graphical syntax of the CORAS diagrams, and a textual syntax and semantics translating the graphical elements into English.

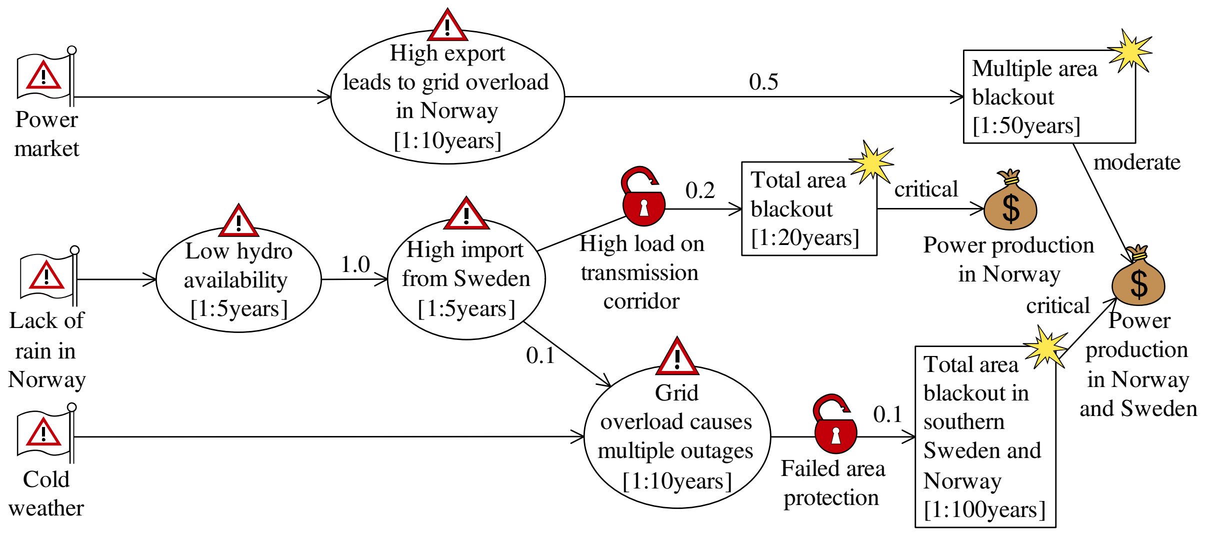

Power market
High export leads to grid overload in Norway [1:10years]
0.5
Multiple area blackout [1:50years]
moderate
0.2
Total area blackout [1:20years]
critical
Power production in Norway
Lack of rain in Norway
Low hydro availability [1:5years]
1.0
High import from Sweden [1:5years]
High load on transmission corridor
critical
Power production in Norway and Sweden
0.1
Cold weather
Grid overload causes multiple outages [1:10years]
Failed area protection
0.1
Total area blackout in southern Sweden and Norway [1:100years]

Figure 34 CORAS threat diagram

CORAS threat diagrams are used during the risk identification and estimation phases of the CORAS risk analysis process. They describe how different threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents, and which assets the unwanted incidents affect. A threat diagram organises this information in a directed acyclic graph, offering the same flexibility as cause-consequence diagrams and Bayesian networks, but using a graphical syntax that is more intuitive and easy to read. At the same time the semantics ensures that the translation of a diagram into English is unique.

CORAS diagrams were originally designed for qualitative analysis. Likelihood and consequence values are assigned directly by workshop participants during brainstorming sessions. However, the CORAS method provides a calculus for computing likelihood and consequence values. The likelihood of a vertex may be deduced given the likelihood assigned to its parent vertices and the relations leading to it, and the likelihood of a vertex composed of several sub-vertices may be deduced from their likelihoods. The calculus is also used to checking the consistencies of assigned likelihood values.

4.4.4 Markov Diagrams

Markov analysis [102][214][129], is a stochastic mathematical analysis method that looks at sequences of events and analyses the tendency of which event that will be followed by another. Markov analysis may be used to analyse the reliability of systems that have a large degree of component dependencies. In contrast to FTA, Markov analysis does not assume complete component independence. It is also well suited to analyse systems that may partially fail or experience degraded states. A Markov analysis considers the system as a number of states, and transmissions between these states. The states are modelled graphically and statistical calculations are performed to determine the probability of each state transmission. Markov analysis is among others promoted by ISO/IEC61508. Markov models are more suitable for showing the operation modes of a system where one may transit forth and back between states, than a chain of events of a security attack which is more likely to be a one way chain. Nevertheless, describing the operation modes of a system also includes describing the different barriers that should prevent an attack or reduce the consequences of an attack and for this purpose Markov analysis may be a useful tool. Using Markov analysis requires a well-specified system and may not be as suitable for high-level analyses.

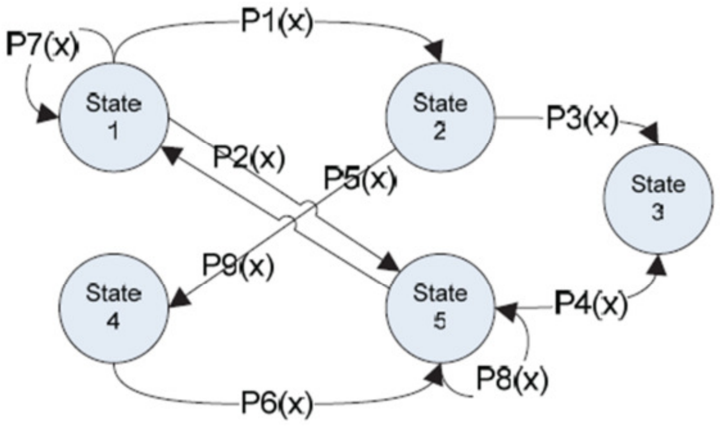


Figure 35 Markov model

4.4.5 Trick-Light

4.4.5.1 Historic

Information Security Management System (ISMS) are becoming widely used, since requirements of ISMS have been defined by ISO 27001 in 2005, providing background for security certification similar to ISO9001's quality certification. The norm 27001 requires that security management decisions are based on a formal risk management process. According to ENISA, much scientific

work needs to be done to make risk assessments and risk management methods attractive for SME, the major target for securing critical information infrastructure.
itrust chose TRICK as acronym for “**T**ool for **R**isk management of an **I**SMS based on a **C**entral **K**nowledge base” in the context of a FP7 project proposal. This proposal intended to develop a software tool and an inherent methodology allowing fixing risk management problems encountered by participating SMEs. This tool is intended to be used across many countries, and enables consultants to share experience and figures automatically. It focuses on fast but quantitative risk evaluation. It models security measures with risk reduction properties that can easily be tailored to a specific organisation. It integrated know-how from multiple standards such as ISO 27002 [115] and risk assessment methods like the French EBIOS and the German IT Grundschutz, with experiences of consultants in one central knowledge base, based on a high-level risk management language defined herein. It develops algorithms to assess risks and to derive residual risk after security implementation, and to compute their ROSI. It develops a concept for easy adaptability to new standards and lessons learned from exploitation.

As this initial project was not financed, itrust decided to develop a prototype called Trick Light in the context of the internship of a master student. This first version was limited to a predefined asset and threat list, grouped by types.
In the context of the CELTIC project BUGYO project, a new version was developed enabling risk assessment for multiple assets, used of tailored risk scenarios, generations of risk treatment plan and statement of applicability for ISO 27001 certification, and last but not least, easy usability as it comes in form of Excel tables to be filled in, and Excel Macros to compute ROSI and risk treatment plans and to draw summary charts.

4.4.5.2 Concept implemented in Trick Light

The Figure 36 highlights the different steps implemented in Trick Light:

Risk assessment

Context definition

The risk assessment starts with the definition of the context of the risk analysis. In this step, we collect information about e.g. the type and business processes of the studied organisation. This information will be used in the following steps by the auditor in order to evaluate e.g. what are the most important assets regarding the sector of the organisation.

Asset identification

This sub step consists of the inventory of the organisation assets considered as important for the organisation business. The assets are identified by name and grouped by types. Other information that can be defined are the value of the asset and a comment in order to justify the value.

Threat identification

As for asset identification, threat identification consists of retrieving a majority of information concerning the threat that could occur on the assets of the organisation. This identification of threats must take into account the context defined previously (e.g. an organism located on a seismic zone should consider the destruction of its premises as a potential threat). A list of typical threats are classifies into 5 classes.
Then the assessor considers the likelihood of occurrence, and as before, classifies a list of typical risk into 5 classes.
A small number of risk scenarios are retained from the previous risk identification for a quantitative risk assessment. As for assets, there is information that can be defined by the risk assessor: name,

type, description and several values concerning the specificity of the risk scenario regarding the risk (this concept will be more precisely described below).

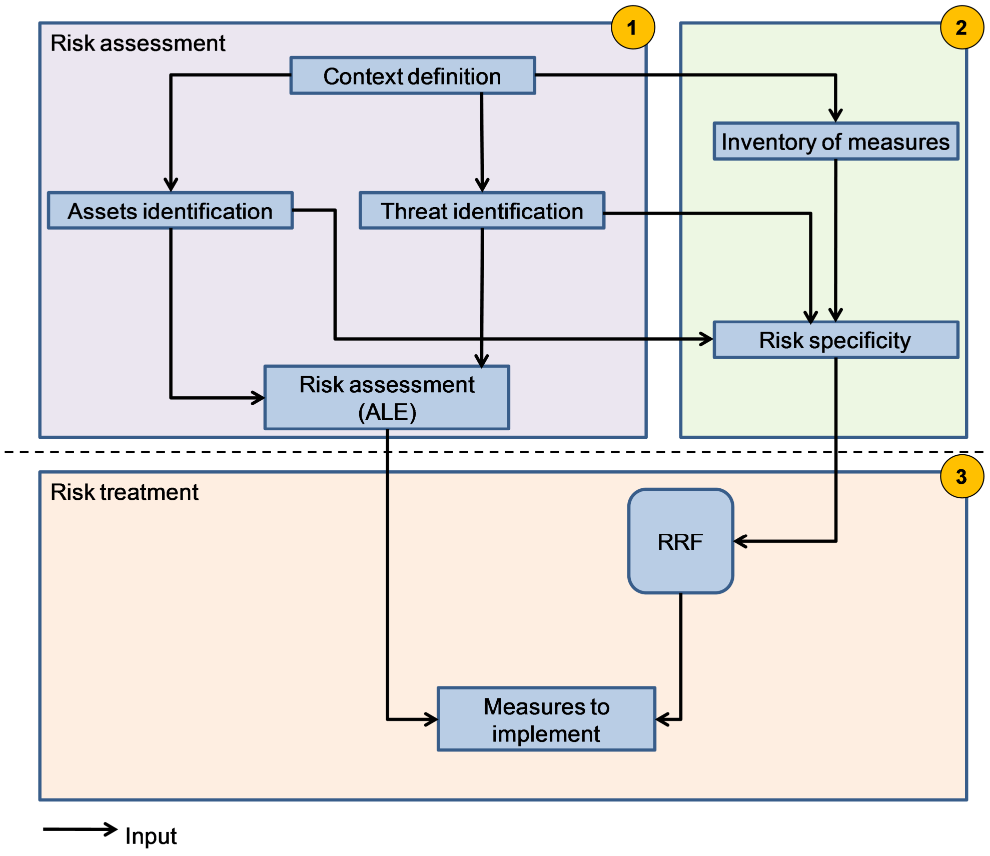


Figure 36: Steps of Trick Light

Risk assessment

This step is one of the most important of the Trick Light process. The risk assessor estimates the annual loss expectancy (ALE) of each couple asset/risk scenario. The ALE is computed multiplying the impact (in Euros) that a threat could have if occurring on an asset, with the potentiality (as annual probability) that a threat could occurs on an asset.

Inventory of measures and risk specificity

Inventory of the measures

This step consists of defining for each security measure of the ISO 27002 norm its current implementation rate and the cost required in order to complete the implementation of the measure.
This allows determining the security measures already completely implemented and consider only the remaining for the risk treatment plan.
It is also possible to exclude some measures which would not be relevant regarding the organisation context or to consider some measures as mandatory in order to force them to appear at the beginning of the action plan and in order to manage the case where there are dependencies between measures.

Risk specificity

Context

This step was performed by the developers of the tools and specificity values are freely available for the security measures of the ISO 27002 and the asset type, limited to 6 elements (Service, Information, Software, Hardware, Network, Personnel, and Immaterial value.
The only elements which require defining a specificity are the risk scenarios added by the user and the customized security measures not covered by ISO27002.

Objectives

The objective of this step is to quantify the risk specificity for the generation of the RRF (Risk Reduction Factor). We distinguish three ones specificity:

- For each **risk scenario**, we determine if it specifically relates to confidentiality, integrity or availability, if it is of intentional, accidental or environmental cause, etc.
- For each **security measure**, we qualify their influences on every security criteria;
- For **assets**, we directly define the influence of each security measure on each asset.

These three specificities are combined together to compute the global influence of the security measure on the ALE generated by the occurrence of a threat on an asset.

Risk treatment

Risk Reduction Factors

All previously mentioned security criteria are now used to compute the Risk Reduction Factor (RRF) associated to each triple Asset/Threat/Measure. Concretely, by associating these criteria together using a weighted computing, we determine a global coefficient of the influence of the security measures on the ALE generated by the occurrence of a threat on an asset.
A RRF is thus a coefficient representing the ALE reduction generated by complete implementation of the security measure.
The computing of the RRF is split into 5 parts.

$$RRF = Asset/Measure * \ Strength * CIA * Type * Source$$

Security criteria

Security criteria used to compute the RRF are sorted by group in the following figure.

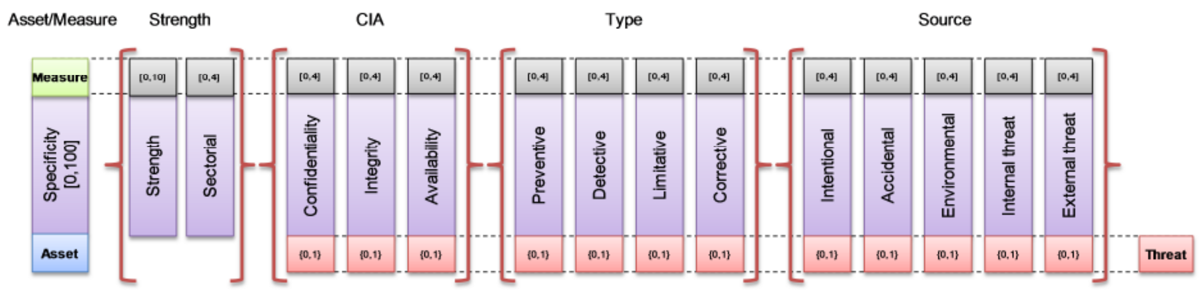


Figure 37: Security criteria used to compute the RRF

Computing of the security measures to implement

Δ ALE

The Δ ALE is the ALE reduction generated by complete implementation of a measure.

Let $ALE_{T,A}^{start}$ be the ALE generated by the occurrence of a threat T on an asset A, e_M be the implementation rate of a security measure M and $RRF_{T,M,A}$ be the RRF associated to the triple (T,A,M).
The formula used to compute the ALE reduction generated by a measure is the following:

$$\Delta\ ALE_{T,M,A} = ALE_{T,A}^{start} * RRF_{T,M,A} * \frac{1 - e_M}{1 - RRF_{T,M,A} * e_M}$$

Considering the Δ ALE, we deduce the general Δ ALE generated by complete implementation of a measure:

$$\Delta\ ALE_M = \sum_{T,A} \Delta\ ALE_{T,M,A}$$

ROSI

The ROSI (Return On Security Investment) is based on the ROI concept, which consists of investing a sum and gaining at least the equivalent, the ideal being to pass the invest sum by a maximal margin.

$$ROI = Return - Investment$$

For example, for 4 000 € invested and 10 000 € of return, the ROI would be of 10 000 – 4 000 = 6 000 €.
The reasoning of the ROSI considers the investment made when implementing the security measure: by analogy with the ROI, the cost of the implementation of a measure corresponds to the invested sum and the Δ ALE corresponds to the gains. Thus, we have the following formula:

$$ROSI_M = \Delta\ ALE_M - cost_M$$

Example: Considering the threat "Deletion of data" which impacts the "know-how" of an organisation, it results an ALE which can be estimated to 100 000 €[5]. The whole implementation of a solution of "data backup" would enable to decrease the ALE to 75 000€. Knowing that the cost of the implementation of this measure of backup is 5000 €, we have a ROSI of 70 000 € (Δ ALE – cost).

Choice of the measures

The selection of the measures to implement is done by comparing the ROSI of each available security measure: concretely, we compute the ROSI of each measure and we choose the one with the maximum of the ROSI.
In the case where the maximum ROSI is superior to 0, we add it to the list of measure to implement and we continue with the same comparison with the others measures. If the maximum ROSI is negative, we stop the computation because there is no relevance to select a measure whose ALE reduction is lower than the cost of the implementation.

Computation of the mandatory and applicable measures

After having given all information previously described, we can start the computation of the measures to implement. This computation is based on our methodology which consists in putting

[5] In our case, ROSI is expressed in k€, and not in €.

first the measure with the highest ROSI. This obviously reduced the ROSI of the remaining security measures, as the *relative* reduction for each risk scenario security measure and asset is constant. During the risk assessment, the security measures have been split into three groups:

- The mandatory measures: these measures are mandatory and will be added first to the list of implementation, even if their ROSI are negatives;
- The applicable measures: these measures may be added to the list of implementation, except if their ROSI are negative;
- The non-applicable measures: these measures will not be taken into account in the computation.

Phase computing

A second computation takes in account all applicable and all mandatory measures. For each measure in this new list, we define several phase and associate them to the measures.
The computation is then made phase by phase.

4.5 RISK ASSESSMENT BASED ON VULNERABILITY ANALYSIS

Most technical IT risks are based on the existence of vulnerabilities, and therefore Oulu University Secure Programming Group (OUSPG) and Codenomicon have been using a simplified risk equation [147]:

Risk = Threat * Vulnerability

In practice, this equation is almost always complemented with “Value”, or “Consequence” as seen for example in [147] the Department of Homeland Security risk equation

*Risk = THREAT * VULNERABILITY * CONSEQUENCE*

Risk assessment that builds in vulnerability analysis therefore attempts to measure the threats and the vulnerability of the system separately.

4.5.1 Attack vectors and Attack surface analysis

Organizations typically have complex networks with more exposed interfaces than they are aware of. Comprehensive attack surface analysis helps in threat modelling. Threats are easiest to map to practise using attack vectors. Attack surface analysis on the other hand maps those attack vectors to real devices and software components, building a catalogue of products that need to be tested. The attack surface analysis is a starting point for mapping threats and understanding the exposure of a system.

An attack vector is the means of exploiting a specific vulnerability in a system. Attack surface analysis maps the attack vectors, those interfaces that can be used to inject malicious inputs (the attacks) into the network, or through the network into other client software. A traditional network scanning based approach (using tools such as “nmap” and “nessus”) will not provide enough insight in what needs to be security tested. A traffic analysis provides more insight of what can be attacked. It provides clear mapping of all interfaces, protocols, and both server and client-side implementations that need testing. With the right setup, also the interfaces inside the network components such as local traffic can be seen. Recording and analyzing real traffic in the network to determine the potential attack vectors can reveal hidden interfaces that would have otherwise been missed in the test plans.

Codenomicon Network Analyzer records traffic at multiple points in your network capturing the entire traffic. It automatically creates visualizations illustrating different aspects of the captured data. You can drill up and down from looking at high-level visualizations to inspecting the corresponding packet data, also in real time, and as a result you will reveal those hidden interfaces and even possible exploits.

4.5.2 CVSS vulnerability metrics

The Common Vulnerability Scoring System (CVSS) provides an open framework for communicating the characteristics and impacts of IT vulnerabilities [147].

CVSS consists of 3 groups: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score.

- The Base group represents the intrinsic qualities of a vulnerability.
- The Temporal group reflects the characteristics of a vulnerability that change over time.
- The Environmental group represents the characteristics of a vulnerability that are unique to any user's environment.

CVSS enables IT managers, vulnerability bulletin providers, security vendors, application vendors and researchers to all benefit by adopting this common language of scoring IT vulnerabilities. While the primary use of CVSS is to estimate the impact of realized vulnerabilities, CVSS scores provide an easy way to calculate numerical values for potential attack vectors.

CVSS Exploitability Metrics form a part of the overall CVSS Scores. They describe how easy an attack vector is to exploit. Components and component values of these CVSS Metrics are:

- Access Vector (AV)
  - Local (L)
  - Adjacent network (A)
  - Network (N)
- Access Complexity (AC)
  - Low (L)
  - Medium (M)
  - High (H)
- Authentication (Au)
  - None (N)
  - Single Instance (S)
  - Multiple Instances (M)

The highest score and the highest risk (10) are accredited to an interface with the following features:

Access Vector : Network
Access Complexity : Low
Authentication : None

Expressing these features using the terms defined in the CVSS Guide provides the following CVSS Vector:

AV:N/AC:L/Au:N.

When looking at threat analysis from the protocol perspective, the attack surface analysis is clearly a critical component. In addition, the motivation and the likelihood of an attack should be considered when making the final decision about what interfaces to protect. Factors like financial gain or the possibility to acquire user database information increase the likelihood of attacks. Overall threat analysis is a broader topic than protocol level analysis and therefore it is out of scope for this study. A comprehensive threat analysis can include factors such as the physical tampering of devices, access control, and IT policies on passwords.

In this study, the impact of compromising each of the affected interfaces was considered by applying the CVSS Impact Metrics and adding them to the Exploitability Metrics explained above. Impact Metrics are another component of the CVSS Base Score, and they provide a tool for estimating the impact of compromising a certain function or service. The CVSS Impact Metrics are the following:

- Confidentiality Impact (C)
  - None (N)
  - Partial (P)
  - Complete (C)

- Integrity Impact (I)
  - None (N)
  - Partial (P)
  - Complete (C)

- Availability Impact (A)
  - None (N)
  - Partial (P)
  - Complete (C)

The most severe attack would be one that could affect all three factors completely:

Confidentiality Impact : Complete
Integrity Impact : Complete
Availability Impact : Complete

Expressed in CVSS terms, the CVSS Vector for these terms would be:

C:C/I:C/A:C.

The overall CVSS Base Score is calculated by combining the Exploitability Metrics and Impact Metrics. An attack which might have score high in the Exploitability Metrics might only have a minor score in the Impact Metrics when considering the overall IMS deployment. For instance, attacking a WLAN Gateway might score high in the Exploitability Metrics, but it is questionable how serious the damage caused by compromising it could be to the complete IMS system.

The most severe attack would consist of the following factors:
Access Vector : Network
Access Complexity : Low
Authentication : None
Confidentiality Impact : Complete

Integrity Impact : Complete
Availability Impact : Complete

Expressed in CVSS terms, the CVSS Base Score for these factors would be 10, and the CVSS Vector would be:

AV:N/AC:L/Au:N/C:C/I:C/A:C.

It should be noted that CVSS is just one possible way of prioritizing interfaces for testing. Other formal classification methods do exist, and in many cases common sense of weighing the possible attack vectors against their criticality is enough.

4.5.3 Risk based planning for a technical security test/audit

The purpose of this attack surface analysis is to identify the most critical interfaces and to help in prioritizing the testing effort. In an ideal world, all available interfaces would be tested, but in reality, budgets, deployment schedules and the availability of tools often impose limitations on what is feasible. Thorough attack surface analysis and vulnerability analysis facilitates the risk based test planning, and the selection process for security tests. It is important also for eliminating unnecessary open services and ports during software design.

The complexities of modern day networks are overwhelming for people conducting security assessment in especially in the area of telecommunications. But as the network architectures remain mostly the same in all deployments, all metrics and guidelines in use for risk assessment also prove to be invaluable and easily reusable when planning a security test. Especially in the area of threat and vulnerability assessment, techniques such as attack surface analysis and identification of attack vectors are critical starting point for testing.

After interfaces are identified and analysed, vulnerability metrics such as CVSS will help prioritizing the interfaces for testing. Although CVSS was designed for completely different purpose, for evaluating the threat from reactive security findings by third parties, it also proves to be useful for proactive work where the actual vulnerabilities are still hiding, unknown to everyone.

We hope that the DIAMONDS case studies will demonstrate how easy the test planning is by using attack vector prioritization using attack surface analysis and CVSS. These techniques will also provide insight into the engineering and specification process as the people involved rarely consider such security metrics when building the software and network architectures.

4.6 RISK RELATED TESTING TECHNIQUES

As the name suggests, Risk-Based Testing (RBT) is based on software risks. Each test case is intended to probe a specific risk that was previously identified through risk analysis [163]. In chapter 4.6.1 ideas for a test process that is based on software risks is introduced. Then test coverage and selection, test specification and modelling as well as test generation strategies are proposed. Chapter 4.4.5 ends with a description of security assurance and common criteria evaluation.

4.6.1 Risk-based testing process

Standard testing approaches usually address risks implicitly. A very basic and unstructured risk analysis might be conducted during the testing process or areas that are known to be security relevant are tested more intensely. This ad-hoc approach is highly dependent on the security testing expertise of the tester and therefore not applicable when testing based on software risks is required.

The goal of Risk-Based Testing is to improve the testing process in order to cover especially risky areas of the application under test and at the same time minimize the time to market and improve the use of resources by focusing testing work on areas with the highest risks. In RBT risk factors are identified and risk-based test cases created and prioritized according to an applicable selection strategy.

Two risk-based testing process models are described in [170] and [229]. Figure 38 and Figure 39 show the respective process models.

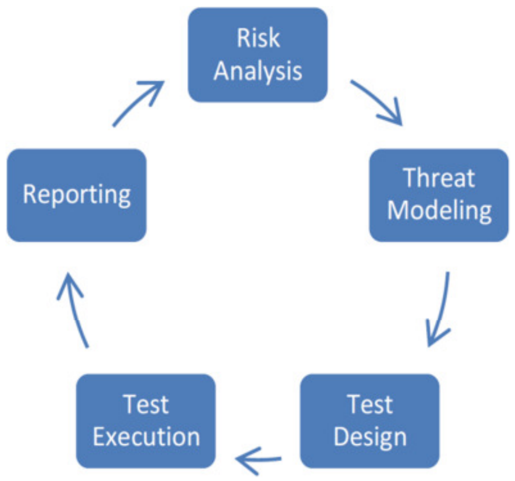


Figure 38: Test Process - Murthy et al.

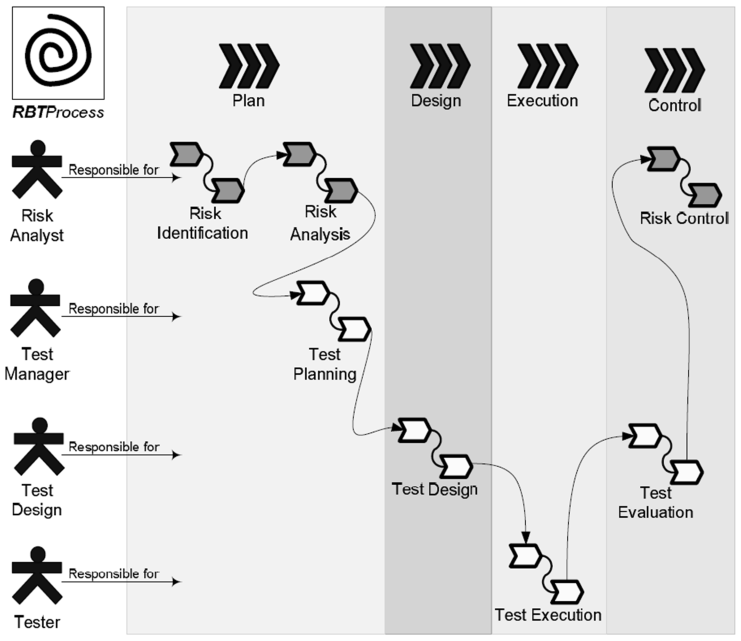


Figure 39: Test Process - Souza et al.

Murthy et al. suggest a five-step iterative process derived from Industry Standards like ISO/IEC 27001 as well as Security Testing Guidelines [177]. Based on the likelihood of a threat and the impact it would cause if it occurred in the field, overall risk ratings are calculated. During the following phases the ordering of tests, their comprehensiveness, the test resourcing and the test iterations are all based on the calculated risk ratings. This allows for a stringent test process in which test efforts are focused on the most likely risks with the highest impact. A more detailed description of this approach can be found in [170].

The process proposed by Souza et al. is fairly similar. Typical technical risks for the type of system under test (SUT) are identified and the SUTs functionalities are analysed and prioritized based on a heuristic risk analysis. Based on the results of this analysis, a test plan is devised, test cases are designed and executed and then evaluated. In the last phase of the process, found defects are mitigated and the test efforts adjusted according to risk control and monitoring. In order to support the testing activities a set of risk-related metrics is applied during the test process. Souza et al. recommend, among others, metrics for risk-based test cases, prioritized risks, identifying risky categories, verifying risk reduction and the effectiveness of RBT. More information on this work can be found in [228] and [229].

4.6.1.1 Test coverage and test selection strategies

Exhaustive testing is infeasible except for trivial cases [233]. As a result, only a subset of all possible test cases can be executed when testing a system. While thorough testing decreases the

risk of unknown faults remaining in the system, it also increases the costs for the development process as well as the time-to-market. The size of the subset is therefore highly influenced by the available testing budget, the desired time-to-market as well as the chosen test coverage. Thus, in order to find a sufficient set of test cases, an adequate selection strategy has to be devised.
Test selection criteria are used to control the generation of test cases by being formalized into test case specifications. The chosen selection criterion assigns priorities to the test cases which define the order in which they will be executed. The criterion also indirectly defines properties of the generated test suites, like their fault detection power, cardinality, complexity, etc.
The most common selection criteria are either based on the structure of the test model, system requirements, data coverage, test case specifications, stochastic characterizations or a well-defined set of faults [114]. The "best" test selection criterion for a specific project is highly dependent on the specified test objectives. In the case of the DIAMONDS project the main objective during testing is security. Therefore the main goal is to minimize the risk of faults remaining in the SUT, occurring during system operation and leading to high damage. When trying to find faults in the SUT, fault-based criteria like mutation coverage are usually employed. As described in [114] mutation coverage involves mutating the model and then generating tests that would distinguish between the mutated model and the model it was based on. This approach is based on the assumption that there is a correlation between faults in the model and in the SUT, and between mutations and real-world faults. Other risk-related selection strategies are described in the following sections.

4.6.1.2 Risk-based stopping criteria

As laid out in the previous section, the decision when to stop testing is always a matter of the remaining test budget, the desired time-to-market and the probability of critical faults staying undiscovered. [31] proposes a test-sequencing method that uses a system test model as its foundation. In the first step, a cost-optimal test tree is derived from the model. It includes a set of test sequences, including tests that cover fault states. Those are states that result in the failure of the system and that can be detected during testing. Then, for each fault state, a presence probability is calculated. This probability can be identified using risk analysis, historical data about the failure of components, or FMEA (see chapter 0).
The test-sequencing method then incorporates several stopping criteria, which allow the tester to stop the testing process when the test cost, signified by the time it takes to run the tests, exceeds the cost of remaining risks. Examples for criteria that base the decision of when to stop testing on the risk that remains in the system are:

- Determining a test sequence that reduced the risk as soon as possible to 20% of the original risk
- Determining the test sequence that makes the risk as low as possible within 1 week of testing
- Determining the test sequence that minimizes the total expected test cost, while leaving no risk in the system

A criterion consists of an objective ("test sequence that makes the risk as low as possible") and zero or more constraints ("within 1 week of testing"). Both can address the test cost or the remaining risk cost.

4.6.1.3 Risk-based regression test selection strategy

Generally, regression testing is employed after changes have been made to the software (e.g. introduction of new functionality and bug fixes). The main goal of this kind of testing is to determine whether new errors were introduced into the system. In [49] Chen et al. propose a regression test strategy which is based on the systems specification. They use risk analysis to quantitatively measure the quality of the existing regression test suite. The proposed process is divided into four steps:

1. Estimate the cost for each test case
2. Derive severity probability for each test case
3. Calculate Risk Exposure for each test case
4. Select test cases that have the highest values of Risk Exposure as Safety Tests

The cost of a test case is estimated based on the consequences for the customer as well as the consequences for the vendor. The severity probability that is derived for each test case depends on the number of defects as well as its severity. In step three the risk exposure for a test case is calculated based on the results of the first two steps. The following formula is used:

$$RE(f) = C(f) \cdot P(f).$$

The cost C(f) of a fault being executed in function f is multiplied with the probability P(f) of the fault being occurring in function f during operation. During the last step, the test cases with the highest Risk Exposure value are executed as *Safety Tests*. They verify that serious failures will not occur during system operation.
Specification-based regression test strategies have already been developed by companies such as IBM.

4.6.1.4 RiteDAP

Another model-based testing technique that uses the cost and probability of a fault for the prioritization of test cases is described in [233]. RiteDAP (Risk-based test case Derivation And Prioritization) uses activity diagrams as system models to automatically generate test case scenarios. Those test models are augmented with the risk values needed for the test case prioritization. The risk values are calculated during a risk assessment phase by using the same formula as described in 4.6.1.2: the probability of a fault is multiplied with the total damage caused by this fault. A sample test model with risk information is shown in Figure 40.

A two-step test case generation process is proposed. First, a set of unordered test case scenarios is derived from the test model. Then the test case scenarios are ordered based on the risk information in the test model. The test case scenarios that were generated during the first step represent a path through the test model but abstract from concrete test data. In order to derive executable test cases, the test case scenarios are augmented with the needed test data. For the second step, two different risk-based prioritization strategies have been implemented. With *Total Risk Score Prioritization* (TRSP), the scenarios are scheduled in the descending order of their associated risks. When applying *Additional Risk Score Prioritization* (ARSP), the first scenario is chosen according to the total risk score. Then, only risks not already covered by a selected scenario are taken into account for further prioritization.

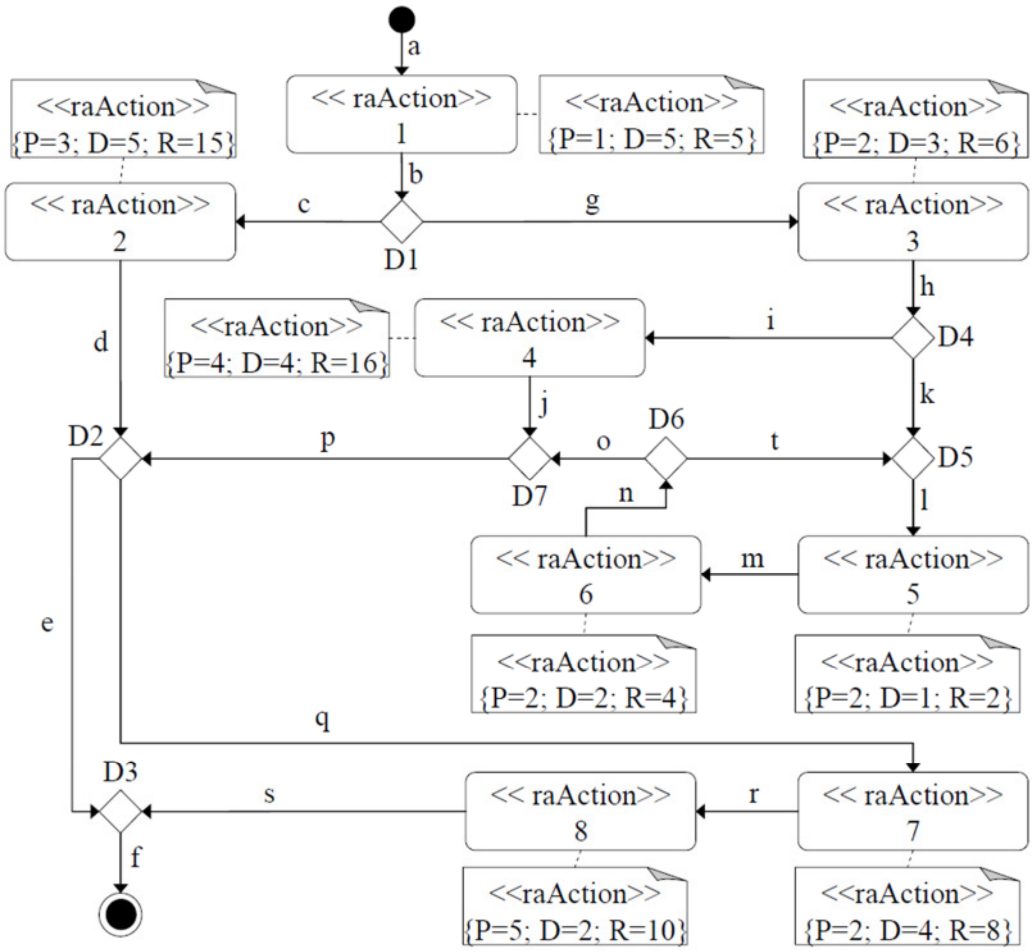

a
<< raAction>> 1
<<raAction>> {P=1; D=5; R=5}
<<raAction>> {P=3; D=5; R=15}
<< raAction>> 2
b
c
g
D1
<<raAction>> {P=2; D=3; R=6}
<< raAction>> 3
h
d
<<raAction>> {P=4; D=4; R=16}
<< raAction>> 4
i
D4
j
k
D2
p
D7
o
D6
t
D5
n
l
e
<< raAction>> 6
m
<< raAction>> 5
<<raAction>> {P=2; D=2; R=4}
<<raAction>> {P=2; D=1; R=2}
q
D3
s
<< raAction>> 8
r
<< raAction>> 7
f
<<raAction>> {P=5; D=2; R=10}
<<raAction>> {P=2; D=4; R=8}

Figure 40: Augmented Test Model

4.6.2 Test specification and modelling strategies

Model-based Testing (MBT) constitutes a number of technologies, methods, and approaches with the aim, to improve the quality, the efficiency, and the effectiveness of test processes, tasks and artefacts. A model is usually an abstract, partial representation of the system under tests desired behaviour. The test cases derived from this model are functional tests on the same level of abstraction as the model. For a SUT, various system models might exist, such as:

- Requirements models
- Behavioural models
- Risk models
- Usage models

Model-based testing involves the following major activities: defining the system requirements, building the model, defining test selection criteria and transforming them into operational test case specifications, generating tests, conceiving and setting up adaptor components and the test environment and then executing the tests on the SUT. Figure 41 shows a generic model-based testing process as defined in [114].

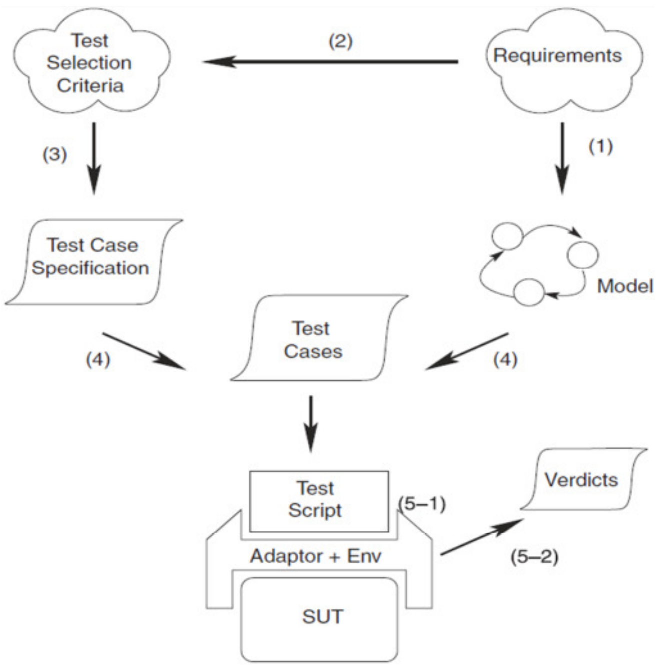


Figure 41: MBT Process

4.6.2.1 Risk model for cloud systems

With the increasing popularity of Cloud Computing, new methods to ensure the security of systems in the cloud need to be developed. Security testing is especially important in this domain, because of the high attack potential coupled with the possibly high damage to sensitive data.
In [256], a new model-based based approach for risk-based security testing of cloud environments is proposed. Negative security requirements are derived from risk analysis in order to cover those security aspects not coved by existing positive requirements. Misuse cases are then defined based on the negative requirements. They depict test cases for the cloud under test (CUT).
Zech uses UML as a modelling language and proposes the use of two models: a cloud system model and a separate test model. The system model is manually defined by the developers. The test model and it's sub-models on the other hand are intended to be generated manually. The three models used during testing are the Risk Model (RM), the Negative Requirements Model (NRM) and the Misuse Case Model (MCM). The intension of the approach is to use Model–2–Model (M2M) transformations to generate these three models. The RBT process is shown in Figure 42.

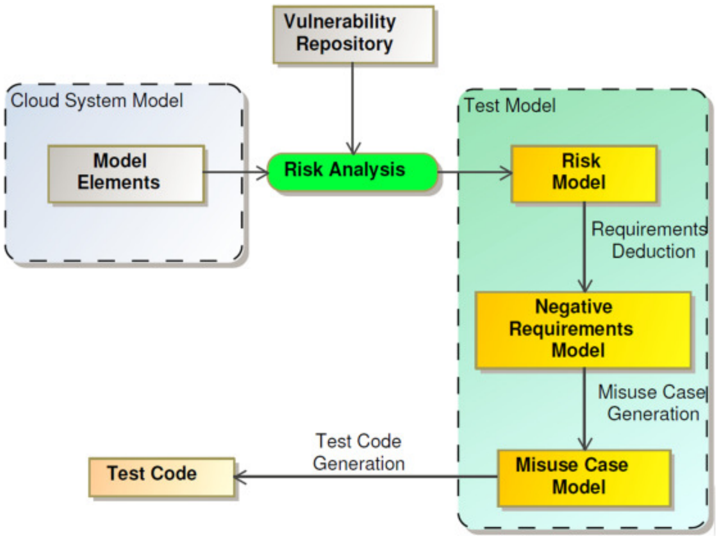


Figure 42: RBT Process for Cloud Environments

Through risk analysis and by exploiting the Vulnerability Repository, which contains past experience concerning risk-related aspects, the Risk Model is created. The RM describes all identified risks, through the affected assets, their threat level, the possibility of the risk occurring during operational mode and a concrete risk description. The Negative Requirements Model is formulated out of the identified risks and is then transformed into the Misuse Case Model, which contains UML activities, resembling the behaviour of a malicious user. The aim of the MCM is to depict possible and performable attacks against the CUT. In the last step, UML activities are incorporated into the process to generate executable test code.

4.6.2.2 Model-based statistical testing

According to [18] the statistical testing produces several artefacts starting from the original requirements until the test results. The artefacts created are:

- Requirements documents (textual, requirements database)
- Black-box system specification
- Test model (Markov chain usage model)
- Test cases (model paths)
- Test results and reliability estimates of the test object

Model-based statistical testing (MBST) is a black-box testing technique that enables the generation of representative tests from the tester's or user's perspective. MBST is used to construct the generic test models and to automatically generate test cases from the concrete test models. Figure 43 shows the steps of the model-based statistical testing approach. In the first stage, a test model is built from the requirements that represent relevant system inputs and usages and the expected system responses. The test automation stage deals with the automated generation, execution, and evaluation of test.

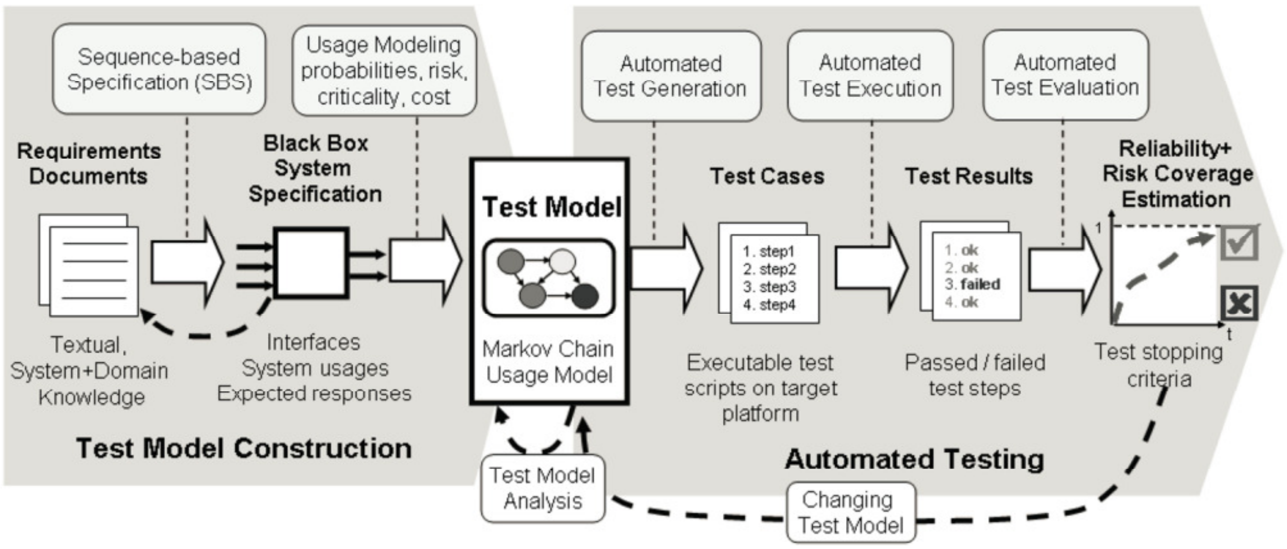

Sequence-based Specification (SBS)
Usage Modeling probabilities, risk, criticality, cost
Requirements Documents
Black Box System Specification
Textual, System+Domain Knowledge
Interfaces System usages Expected responses
Test Model
Markov Chain Usage Model
Test Model Construction
Test Model Analysis
Automated Test Generation
Automated Test Execution
Automated Test Evaluation
Test Cases
Test Results
Reliability+ Risk Coverage Estimation
1. step1
2. step2
3. step3
4. step4
1. ok
2. ok
3. failed
4. ok
Executable test scripts on target platform
Passed / failed test steps
Test stopping criteria
Automated Testing
Changing Test Model

Figure 43: MBST Process

[18] describes how to use XML based techniques in order to derive Markov Chain Usage Models from sequence diagrams. They define statistical testing as a testing process that requires random selection of test cases from the input domain of a software system according to the intended usage distribution. In Markov Chains, a system with a finite state-space can transition from one state to another. The needed usage distribution for the random selection of test cases is added to the underlying model by annotating the transitions according to their probability of occurrence.
MBST has been extended for risk-based testing as in [210]. In the proposed approach, risk analysis techniques such as FMEA are employed to identify all critical situations that indicate high risk. More information on risk analysis can be found in chapter 4.3. Zimmermann et al. employs an algorithmic method for transforming a given test model (Figure 44) into a test model (Figure 45) that focuses on the generation of test cases that trigger critical situations.

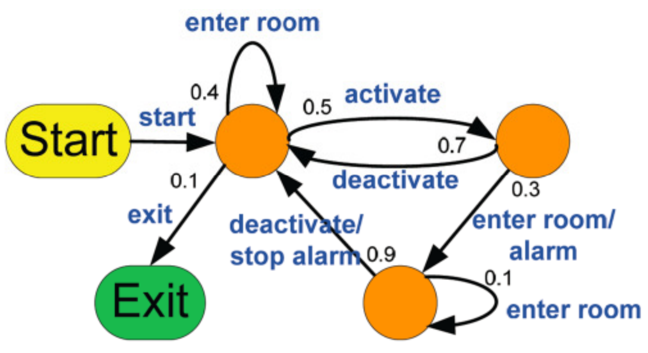


Figure 44: Original Test Model

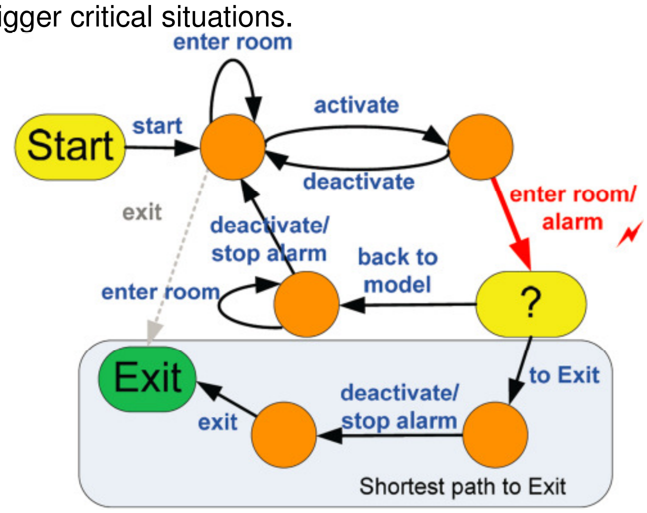


Figure 45: New Test Model

[258] describes the algorithm to create the new test model as follows: In the original model M, all Exit transitions are deleted. After each critical transition, a new decision state is inserted. In this state, a decision between staying in the test model and going to EXIT has to be made. Hence, the original test model is modified in such a way that only critical test cases are derived. Therefore all test cases that are generated from the new model should contain at least one critical transition.

Currently there a several tools that support model based statistical testing. The most important representatives are the two tools and JUMBL and MaTeLo, which we describe in the following.

4.6.2.2.1 JUMBL

The "J Usage Model Builder Library" (JUMBL) [189] is a Java class library and set of command-line tools for statistical testing on basis of usage models. JUMBL supports construction and analysis of usage models, generation of test cases, automated execution of tests, and the analysis of models and testing results. The usage models used as input for JUMBL are finite-state, time homogeneous Markov chains [129] which can be represented by deterministic finite automata with probabilistic transitions. Such a model is depicted in Figure 46. It describes the operation of a simple telephone. A "use" of the phone is any path from the state *On Hook*, the so called source, to the state *Exit*, the so called sink.

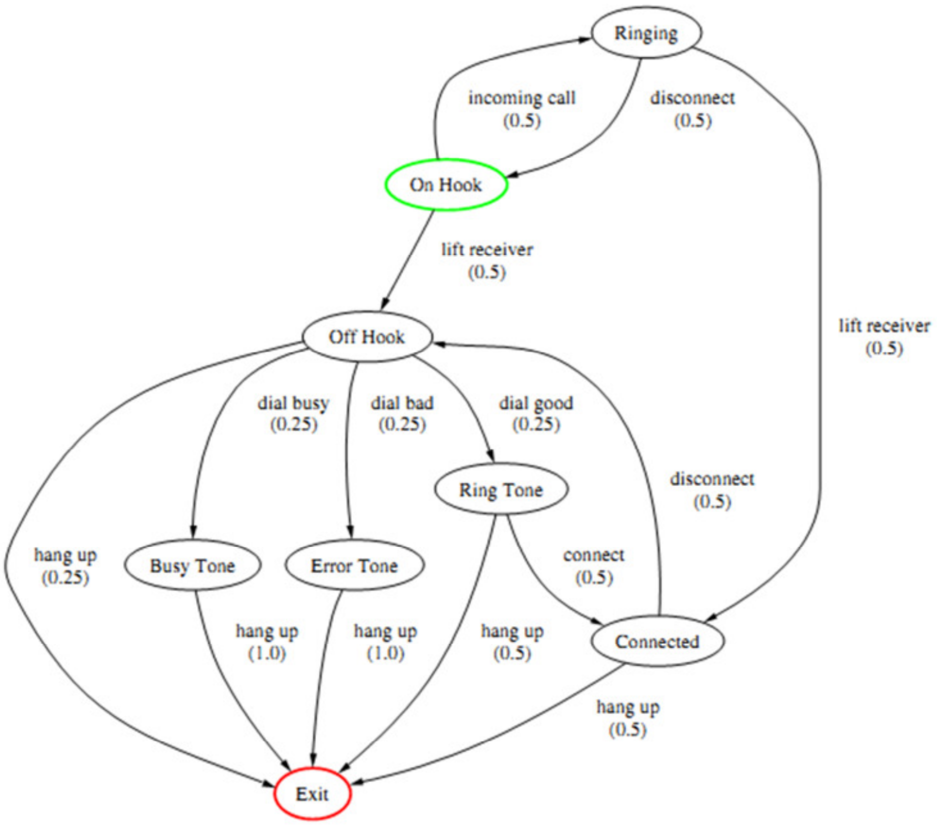


Figure 46: JUMBL usage model of a simple telephone [189]

JUMBL provides different test generation algorithms based on the model. The main idea of test generation is the calculation of weighted paths of the user model. Each path describes a distinct use of the SUT and represents a test. The weighting of the paths is calculated from the weights of individual edges. JUMBL generates test cases in the following ways:

1. Coverage with minimal cost: test cases are generated that cover the model with the minimum cost.
2. Random sampling: test cases are generated by random sampling.
3. Order by probability: test cases are generated in order by probability.
4. Manually: test cases are created manually by means of an interactive test case.

5. Interleaving: test cases are created by interleaving the events of other test cases.

To achieve coverage with minimal cost, JUMBL uses the Chinese postman algorithm to construct a collection of test cases which cover all edges of the model. A nonnegative cost is associated with each edge of the model. The algorithm minimizes the sum of these costs and thus the length of the resulting test cases. Running this minimum-cost coverage set gives a measure of assurance that the usage model and the system as delivered match; as a result, the minimum-cost coverage set is typically run before running any other tests generated from the model. Random test cases are created by randomly choosing outgoing edges in accordance with a probability distribution on the model. The test case with the greatest probability is generated first, the next-most-likely is generated next, etc. This allows testing those cases which are expected to occur most often. In the interactive mode, JUMBL allows the user to select certain parts of the model that should be covered by a test case. Last but not least interleaving allows the combination of test cases from different models. This is especially useful to provide modular modelling approaches.

JUMBL supports or accept various representation formats of such models. The main input format is The Model Language (TML), which is a simple language for describing usage models in terms of states and edges, and supports submodels, labels, and constraints. In addition to TML, JUMBL supports standard formats like comma-separated value (CSV), the DOT language used by the Graphviz tools, GML as used by Graphlet, and Model Markup Language (MML). MML is an XML extension language created specifically for representing Markov chain usage models.

The software is made available by the Software Quality Research Laboratory (SQRL) for partners of the lab. Moreover evaluation copies of the JUMBL are available for use for a limited time. To find out the current licensing arrangements for the JUMBL, contact the SQRL (http://sqrl.eecs.utk.edu/esp/jumbl4/).

4.6.2.2.2 Matelo

Matelo (**Ma**rkov **Te**st **Lo**gic) [153] is a commercial tool for statistical automated generation of functional test cases based on a usage model that completes the model of the SUT. Matelo consist of the four features, namely Requirements Management, MaTeLo Model Editor, MaTeLo Testor and MaTeLo Test Campaign Analysis. MaTeLo is a Model-based Test Case Generator. The model that is used as base for test case generation is designed in MaTeLo using a proprietary notation. The model consists of Markov chains, i.e. UML state charts, for which the transitions are annotated according to their probability of occurrence.

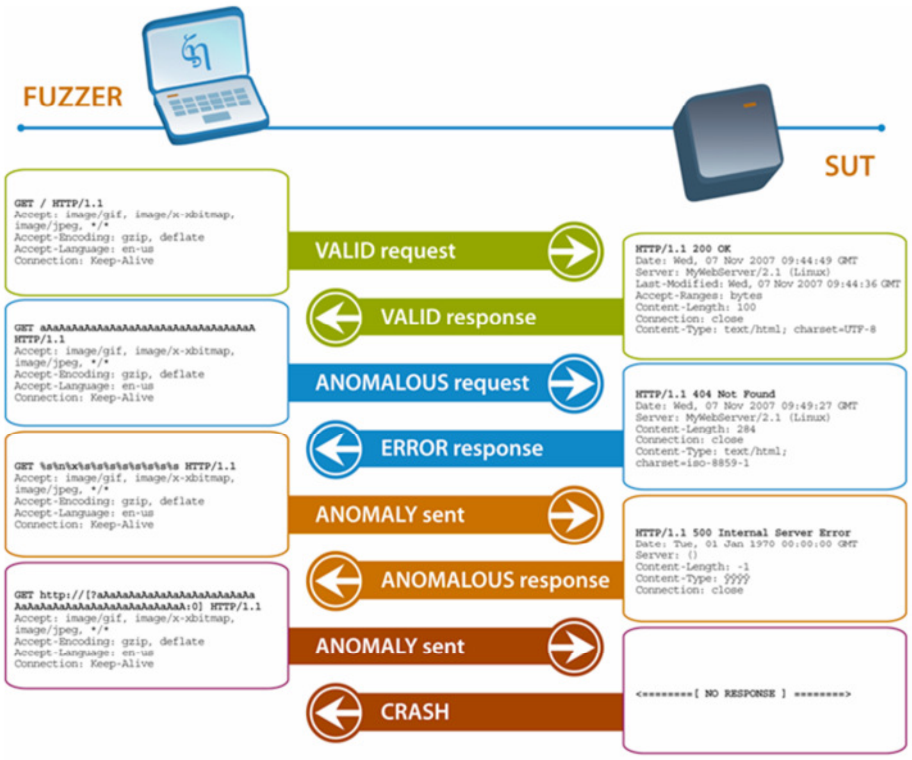
MaTeLo supports integration, system and acceptance test processes. It is mainly used in the automotive industry for acceptance and functional as well as configuration testing. Test execution is out of scope, but the generated test cases can be executed manually based on their XML/HTML representation or automatically in the target test automation platform according to the selected test scripting notation. MaTeLo support offline test case generation exclusively.

SUT models designed with IBM Rational software and IBM Rational Software Architect may be imported into the MaTeLo tool, provided they met certain requirements stated in the user manual. However, the import functionality for UML models is restricted to UML sequence diagrams. Requirements can be imported from Rational DOORS or other CSV-based datasources (Excel) or XML (RIF). Test cases can be exported into many test execution platform like HP Quick Tester Professional, TTCN-3 or IBM Rational Functional Tester.

The tool is sold by the company All4Tec. All4Tec provides professional support and training for test engineers. MaTeLo is available for Unix/Linux, Windows 2000, Windows XP and Windows Vista.

4.6.3 Fuzz test generation strategies

Fuzzing is a technique for intelligently and automatically generating and passing into a target system valid and invalid message sequences to see if the system breaks. A practical example of fuzzing would be to send malformed HTTP requests to a web server, or create malformed Word document files for viewing on a word processing application. The web server example is graphically illustrated in the figure below.



```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap,
image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Wed, 07 Nov 2007 09:44:49 GMT
Server: MyWebServer/2.1 (Linux)
Last-Modified: Wed, 07 Nov 2007 09:44:36 GMT
Accept-Ranges: bytes
Content-Length: 100
Connection: close
Content-Type: text/html; charset=UTF-8
```

```
GET aAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaA
HTTP/1.1
Accept: image/gif, image/x-xbitmap,
image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

```
HTTP/1.1 404 Not Found
Date: Wed, 07 Nov 2007 09:49:27 GMT
Server: MyWebServer/2.1 (Linux)
Content-Length: 284
Connection: close
Content-Type: text/html;
charset=iso-8859-1
```

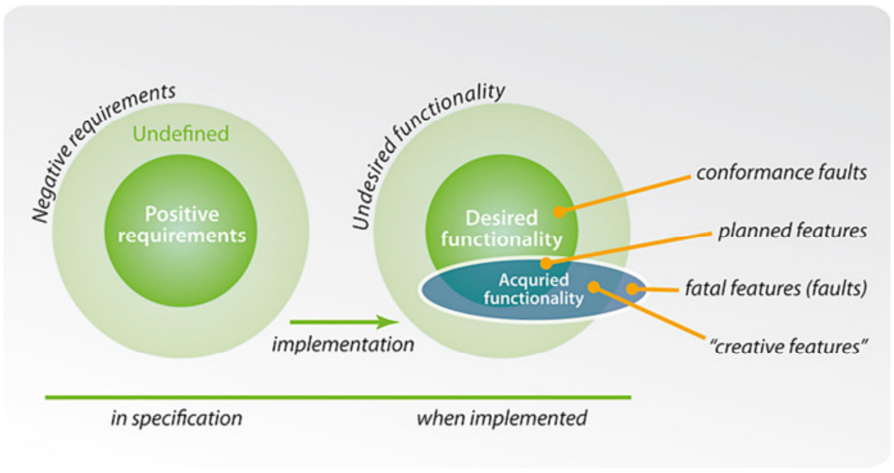
```
GET %s%n%x%s%s%s%s%s%s%s%s HTTP/1.1
Accept: image/gif, image/x-xbitmap,
image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

```
HTTP/1.1 500 Internal Server Error
Date: Tue, 01 Jan 1970 00:00:00 GMT
Server: ()
Content-Length: -1
Content-Type: ????
Connection: close
```

```
GET http://[?aAaAaAaAaAaAaAaAaAaAaAaAa
AaAaAaAaAaAaAaAaAaAaAaAaAaA:0] HTTP/1.1
Accept: image/gif, image/x-xbitmap,
image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

```
<========[ NO RESPONSE ] ========>
```

Fuzzing has proven to be the most efficient method for finding previously unknown vulnerabilities in software. This is due to the nature of software creation processes: invariably, irrespective of the original specification, features end up in the resulting implementation without no-one knowing where they are there. These features may be not be harmful, and are sometimes even useful. However, a majority of them cripple the robustness of the software and sometimes manifest themselves as exploitable vulnerabilities that hackers can make use of to compromise systems. This gap between specification and implementation is illustrated in the figure below.



As stated earlier, fuzzing comprises transmitting input sequences to a target system. This is performed as part of the following process:

1. Create protocol model (if possible)
2. Create test cases from the model or from templates
3. Analyze test results by monitoring the test target

In the context of model-based fuzzing, a model can always be created - if nothing else, it's just a simple model that produces purely random sequences of bits.

The test case generation as part of the fuzzing process is based on the acquired protocol model. How to do this depends on various factors, including the following:

- Is there a "grammar" (specification) available that specifies which kinds of inputs are valid and which are not?
- If not, is it possible to capture some kinds of example sequences as templates?
- Can the creator of the fuzzer afford the time to implement a system that can produce valid message sequences to an extent that represents the space of all input sequences sufficiently well?

4.6.3.1 Specification based interface models

If there is a specification and enough time available, the most efficient fuzzer can be created by modelling the specification of the communication interface, and use that model for creating valid message sequences. The test generation is then about insert anomalies into the sequences. These anomalies are message elements that are known to cause problems in software. That is, the valid input sequence is broken in a way so as to be as likely to create problems in the target system as possible. This approach that uses a protocol specification is called specification based protocol fuzzing.

Specification based fuzzing tools come in two flavours:

- Either the model has been pre-built into the tool so that the user just needs to connect it to the target system and start testing
- Or, alternatively, the "test tools" is just a framework that facilitates the process of creating the protocol model, but contains no model as such

4.6.3.2 Template based models

If the input model cannot be created from a specification, one can in some cases capture example sequences and, if their structure can be analyzed to a sufficient degree, modify the examples with inserted anomalies and then use the resulting sequences as test cases. The templates used to build the model may, depending on the situation, be flat binary data or, for example ASCII character streams, and the model may be constructed with no relation to the underlying protocol or file specification. Alternatively, if an attempt to actually model the protocol is wished, more advanced algorithms exist that try and understand the input structure on a more sophisticated level.

4.6.3.3 Random model

If all else fails, test cases can also be created partially or fully randomly. Even though this approach contains the least amount of insight into the target system, even random test cases are known to often trigger bugs in software.

4.6.3.4 Test generation techniques

When the model has been built, one needs next to create the test cases. This may be performed in one of the following ways:

- **Online**: Create the test cases in real time as testing proceeds, basing new test cases on what has been done previously
- **Offline**: Create static tests that are later injected to test target. The "static tests" may be files, preformatted message sequences or other types of input.

4.6.3.5 Conclusions

According to studies by Codenomicon, all types of fuzzing discover new bugs and they all have their time and place. The more input awareness one can build into a fuzzer, the more efficient it its likely to be. Therefore, whenever sophisticated specification based fuzzers are available for a specific purpose, they are the safest bet to use.

It is a difficult and time consuming task to build good specification based fuzzers so that if a fuzzer is needed for a novel purpose, with no off-the-shelf tool available, the decision boils down to whether it makes sense to build one from scratch, possibly using a fuzzing framework, or deploy a template based fuzzer instead.
In some situations, also test execution speed may be critical. In these situations, specification based fuzzers also shine as the test cases can be pre-built according to the specification, and then efficiently injected into the target system as the test proceeds. This partly applies to template based and random fuzzing, too. However, in the template-based approach, is often beneficial to first do some work with finding good templates. Random fuzzing suffers from a need to have a huge deal of test cases for even a moderate coverage of the input space.

4.6.4 Common criteria evaluation

4.6.4.1 Introduction

The standard ISO 15408, also called Common Criteria [112], aims at facilitating writing Security Targets, i.e. a set of security features which has to be applied for a specific product (e.g. the X-Company #xxx Firewall) to reach a certain security level. “To allow consumer groups and communities of interest to express their security needs, and to facilitate writing Security Targets, the CC provides two special constructs: packages and Protection Profiles (PPs)”.
A PP is the description of a security profile applied to a type of product (e.g. any home firewall or business firewall) to reach the same security level. The aim of the PP is to describe, according to the more rational manner as possible, the system and its environment, the threats which can impact the system, the required security objectives to reach the chosen security level, the Security

Functional Requirements to reach these objectives and then to be able to define and to assess the technical solution implemented.

The PPs project aim to:

- facilitate the writing of Security Targets for specific business application based on the project and fulfilling specific customer requirements in term of security level;
- verify rationally that the technical features implemented in the test bed fulfil the security functional requirements and the security assurance requirements according to a required assurance level.

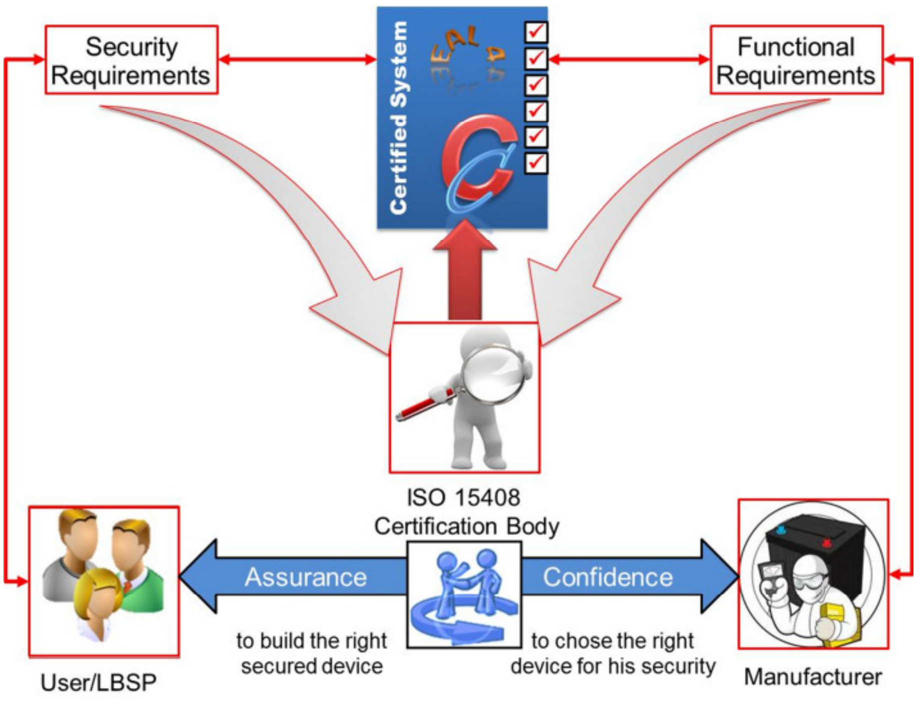

Security Requirements
Certified System
EAL
Functional Requirements
ISO 15408 Certification Body
Assurance
Confidence
to build the right secured device
to chose the right device for his security
User/LBSP
Manufacturer

Figure 47: Common Criteria approach

Why to use this type of approach? Figure 47 describes the advantage of this type of approach: The standardised approach allows:

1. Choosing security objectives and assumption to cover identified treats.
2. Designing Security Functional Requirements to cover objectives.
3. Certifying through a specific Certification Body that the system is secure if operated in the conditions it has been designed for.
4. Providing confidence to manufactures that the device is secure enough.
5. Allows users to trust in the security of a given device.

The ISO/IEC 15408 defines two forms for expressing IT security functional and assurance requirements.

- The **protection profile (PP)** construct allows creation of generalised and reusable sets of such security requirements. The PP can be used by prospective consumers for specification and identification of products with IT security features which will meet their needs.

- The **security target (ST)** expresses the security requirements *and* specifies the security functions for a particular product or system to be evaluated, called the target of evaluation (TOE). The ST is used by evaluators as the basis for evaluations conducted in accordance with ISO/IEC 15408.

This standard proposes to evaluate the assurance level of a security component. ISO/IEC 15408 is a convergence of the Orange Book standards (NSA) and ITSEC standards (Europe).
The standard is split into 3 sub-standards:

Name	Description
ISO/IEC 15408-1:2005 **Introduction and general model**	ISO/IEC 15408-1:2005 defines two forms for expressing IT security functional and assurance requirements. The protection profile (PP) construct allows creation of generalized reusable sets of these security requirements. The PP can be used by prospective consumers for specification and identification of products with IT security features which will meet their needs. The security target (ST) expresses the security requirements and specifies the security functions for a particular product or system to be evaluated, called the target of evaluation (TOE). The ST is used by evaluators as the basis for evaluations conducted in accordance with ISO/IEC 15408.
ISO/IEC 15408-2:2008 **Security functional components**	ISO/IEC 15408-2:2008 defines the content and presentation of the security functional requirements to be assessed in a security evaluation using ISO/IEC 15408. It contains a comprehensive catalogue of predefined security functional components that will meet most common security needs of the marketplace. These are organized using a hierarchical structure of classes, families and components, and supported by comprehensive user notes. ISO/IEC 15408-2:2008 also provides guidance on the specification of customized security requirements where no suitable predefined security functional components exist.
ISO/IEC 15408-3:2008 **Security assurance components**	ISO/IEC 15408-3:2008 defines the assurance requirements of the evaluation criteria. It includes the evaluation assurance levels that define a scale for measuring assurance for component targets of evaluation (TOEs), the composed assurance packages that define a scale for measuring assurance for composed TOEs, the individual assurance components from which the assurance levels and packages are composed, and the criteria for evaluation of protection profiles and security targets. ISO/IEC 15408-3:2008 defines the content and presentation of the assurance requirements in the form of assurance classes, families and components and provides guidance on the organization of new assurance requirements. The assurance components within the assurance families are presented in a hierarchical order.

Table 3: ISO/IEC 15408 description

4.6.4.2 Protection Profile and Security Target

The common criteria defined some frequently used terms:
A **Protection Profile (PP)** defines objectives and security requirements, independently of the implementation. The PP is reusable and normally public. A PP specifies generic security evaluation criteria to substantiate vendors' claims of a given family of information system products.

A **Security Target (ST)** is an implementation-dependent statement of security needs for a specific identified **TOE (Target of Evaluation**: e.g. a new implementation: the location assurance system studied in this project).
The Security Target provides a structured description of the TOE to determine correctness in the form of **Security Assurance Requirements (SAR**s), formulated in standardised language. This will allow avoiding the incorrect or incomplete design or implementation of the TOE.
The common criteria propose two types of evaluation:

- "White box": product evaluation (router, proxy, smart card, ATM…);
- "Black box": system evaluation. This system could be based on components already tested.

This evaluation is useful to provide materials and systems to governmental organisations. The price of an evaluation is determined by the assurance level required and the complexity of the job.
The standard description, which we follow here, is described in Figure 48 below:

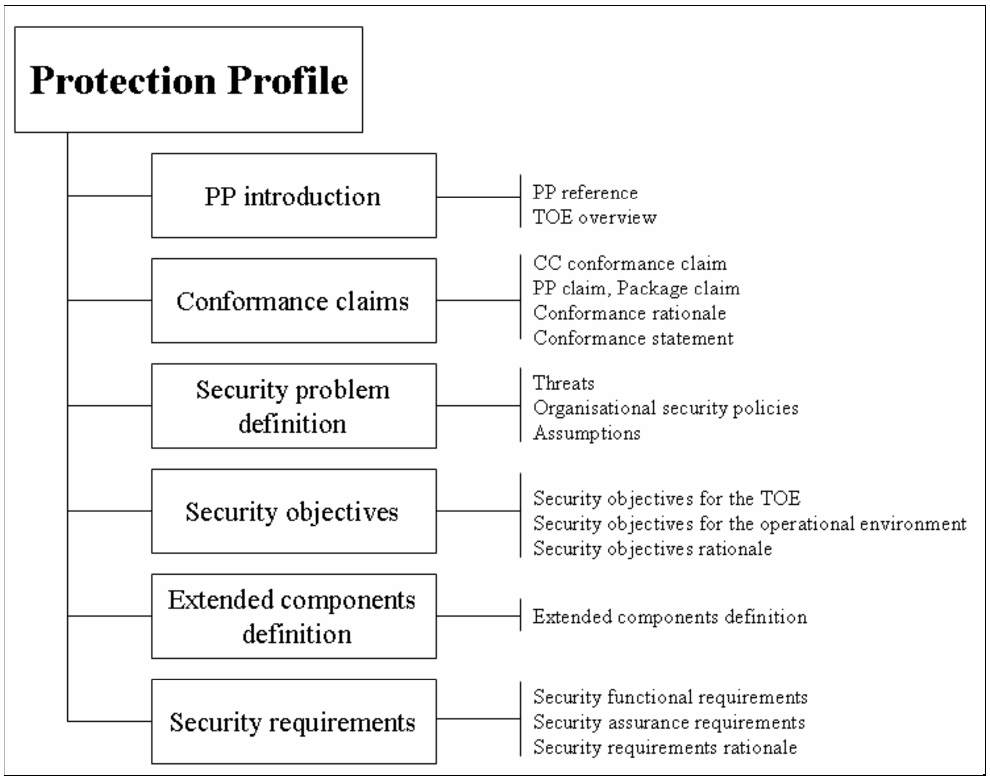

Protection Profile
PP introduction
PP reference
TOE overview
Conformance claims
CC conformance claim
PP claim, Package claim
Conformance rationale
Conformance statement
Security problem definition
Threats
Organisational security policies
Assumptions
Security objectives
Security objectives for the TOE
Security objectives for the operational environment
Security objectives rationale
Extended components definition
Extended components definition
Security requirements
Security functional requirements
Security assurance requirements
Security requirements rationale

Figure 48: Protection Profile Contents [ISO 15408]

4.6.4.3 Assurance Level and Composition of EAL

The last part of the ISO 15408 standard deals with the assessment of the Assurance Level of a system previously described in PP or ST. The standard describes seven levels of assurance according to the following table:

Evaluation of Assurance Level	**Short Description**	**Developer Task**
EAL 1	functionally tested	An evaluation at this level should provide evidence that the TOE functions in a manner consistent with its documentation

Evaluation of Assurance Level	Short Description	Developer Task
EAL 2	structurally tested	This EAL represents a meaningful increase in assurance from EAL1 by requiring developer testing, a vulnerability analysis (in addition to the search of the public domain), and independent testing based upon more detailed TOE specifications.
EAL 3	methodically tested and checked	This EAL represents a meaningful increase in assurance from EAL2 by requiring more complete testing coverage of the security functionality and mechanisms and/or procedures that provide some confidence that the TOE will not be tampered with during development.
EAL 4	methodically designed, tested, and reviewed	This EAL represents a meaningful increase in assurance from EAL3 by requiring more design description, the implementation representation for the entire TSF, and improved mechanisms and/or procedures that provide confidence that the TOE will not be tampered with during development.
EAL 5	Semi-formally designed and tested	This EAL represents a meaningful increase in assurance from EAL4 by requiring semiformal design descriptions, a more structured (and hence analysable) architecture, and improved mechanisms and/or procedures that provide confidence that the TOE will not be tampered with during development.
EAL 6	Semi-formally verified design and tested	This EAL represents a meaningful increase in assurance from EAL5 by requiring more comprehensive analysis, a structured representation of the implementation, more architectural structure (e.g. layering), more comprehensive independent vulnerability analysis, and improved configuration management and development environment controls.
EAL 7	formally verified design and tested	This EAL represents a meaningful increase in assurance from EAL6 by requiring more comprehensive analysis using formal representations and formal correspondence, and comprehensive testing.

Table 4: EAL description

As it is usually more relevant and easier to describe a system as a set of different components, the standard allows assessing the assurance level of different components and to define the assurance of system thanks to a define methodology taking into account the interaction of the assessed components and their environment:

"While a dependent component can be evaluated using a previously evaluated and certified base component to satisfy the IT platform requirements in the environment, this does not provide any formal assurance of the interactions between the components or the possible introduction of vulnerabilities resulting from the composition. Composed assurance packages consider these interactions and, at higher levels of assurance, ensure that the interface between the components has itself been the subject of testing. A vulnerability analysis of the composed TOE is also

performed to consider the possible introduction of vulnerabilities as a result of composing the components.”
The norm considers three Composed Assurance Packages:

Name	Short Description	Developer Task
CAP A	*Structurally composed*	*CAP-A provides assurance through unique identification of the composed TOE (i.e. IT TOE and guidance documentation).*
CAP B	*Methodically composed*	*The analysis is supported by independent testing of the interfaces of the base component that are relied upon by the dependent component, as described in the reliance information (now also including TOE design), evidence of developer testing based on the reliance information, development information and composition rationale, and selective independent confirmation of the developer test results. The analysis is also supported by a vulnerability analysis of the composed TOE by the evaluator demonstrating resistance to attackers with basic attack potential.*
CAP C	*Methodically composed, tested and reviewed*	*This CAP represents a meaningful increase in assurance from CAP-B by requiring more design description and demonstration of resistance to a higher attack potential.*

Table 5: *Composed Assurance Packages* (CAP) description

4.6.5 Security Assurance

Security assurance has been analysed in the Celtic project BUGYO Beyond [37]. This project aims at assessing the assurance level of an organisation or an infrastructure called Target of Measurement (ToM), by applying a specific methodology called the assurance measurement program. The assurance level assessment is based on two concepts, the Assurance Profile (AP) and the Operational Security Assurance.

Assurance Profile

An assurance profile is the expression of requirements to deploy a security assurance program in order to measure, monitor and maintain security assurance of a telecommunications infrastructure for a particular service. The objectives of this concept is similar to the Protection Profile [111] concept promoted by the Common Criteria, but are focused on the operational aspects. Such a program can be illustrated in the Figure 49 where assurance management is a continuation of risk management and an input for trust management. We address in this document security assurance by measurement: infrastructures measured by metrics that generate evidence that leads to assurance.

Operational Security Assurance

The security assurance is defined as ground for confidence that security countermeasures are running as expected; this is illustrated in Figure 50: Operation Security Assurance in Security life cycle. The operational security assurance is the last step in the overall security life cycle process. The diagram presents the different steps and how security assurance is related to all of them. The figure also shows how some of the most relevant standards are related to these different steps.

The first step is called risk management. A service infrastructure is exposed to threats and is subject to vulnerabilities (inherent risks). Managing risks consists in first identifying those risks (known/identified risk) and then deciding the ones that can be covered by security objectives and those that are considered residual risks. There are several standards concerned with risk management e.g. ISO 27005 [116] and ETSI e-TVRA are some of the most appropriate and used standards related to IT and telecommunications infrastructures.

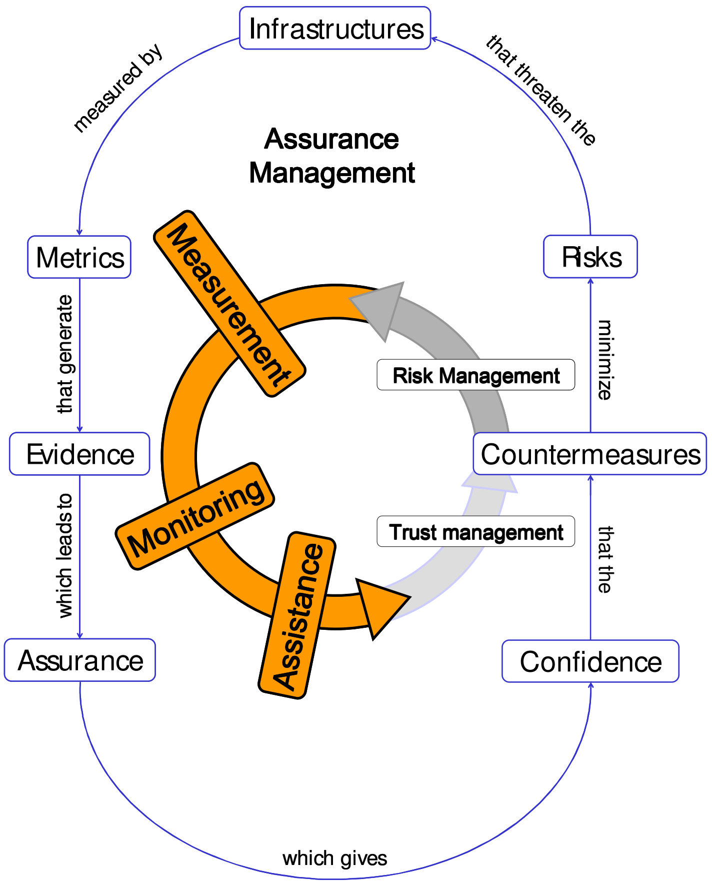


Figure 49: Risk, Assurance and Trust

The second step is the design of security countermeasures that will lead to architecture and implementation. The drift that can occur can be measured with ISO 15408 standards that will bring assurance that the implemented system will satisfy security objectives.

The third step is the deployment phase where security architecture is deployed and configured. Drift that can occur is called deployment drift as pictured in Figure 2.

The last step is the operational phase. In this phase as stated in the assurance definition, operation security assurance will give an evaluation (with more or less precision depending on the assurance level) of the operational drift. To achieve and quantify this drift, measures (metrics) are performed on the target of measurement. Those measures are processed in order to give means of the operational drift from deployed security architecture and effectively running security architecture.

In few introductive words, an Assurance Profile as defined in this document defines the Target of measurement, specifies what to be measured (Measurement requirements) and give means for combining the measurements results in a useful way (Security Assurance view).

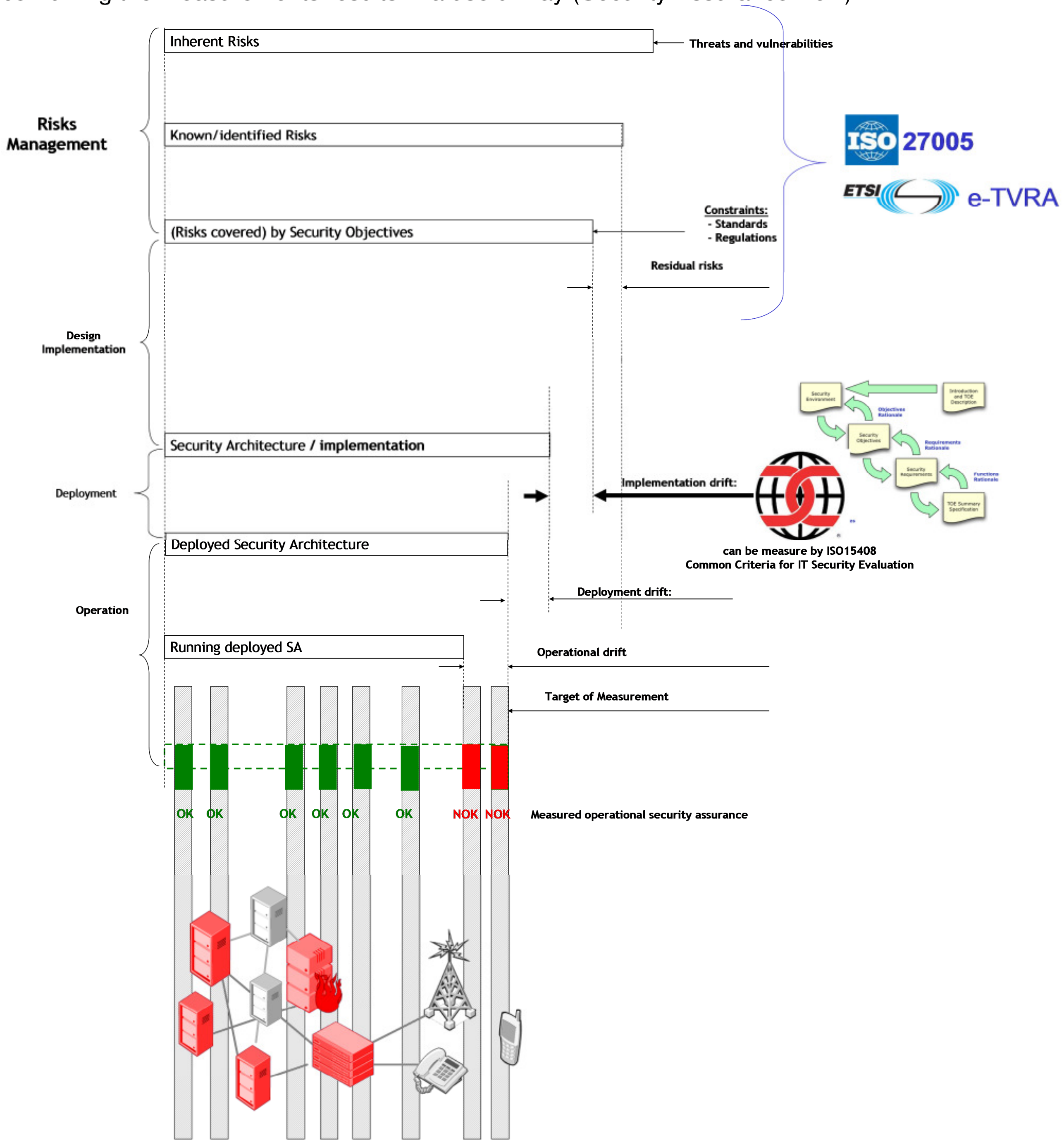


Figure 50: Operation Security Assurance in Security life cycle

The operation BUGYO methodology is composed of six main steps. This methodology enables to specify and deploy the program an entity decided to implement for the purpose of monitoring in operational situation the security assurance of a communication service. Each step addresses a specific process that enable to build the security assurance program. The methodology is decomposed in two parts:

- **Preparatory steps**, that implement a feedback loop to address the necessary learning process,
- **Operational steps**, which are naturally a continuous process.

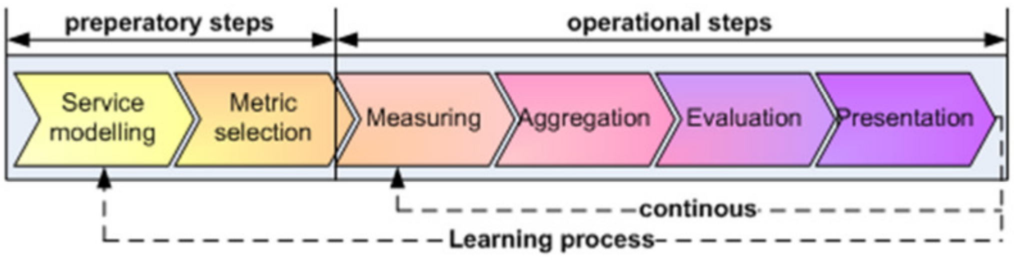


Figure 51: BUGYO six-step Operational Methodology.

The first step consist in **modelling** the security assurance of the communication service we want to continuously monitor using a published model formalism that is detailed in 2.4.1. This step is crucial and defines the service security assurance needs by taking into account its critical parts and their dependencies in term of security rather than network organisation.

The second step of **metric selection** determines the maximum assurance level that can be assessed from each parts of the service. A targeted assurance level is chosen for each modelled device of the service infrastructure. Targeting a low level may provide a low confidence but will be easier to measure and maintain.

In the **measuring** step, measuring probes, on which the metrics defined at step two of the methodology relies, are selected and deployed in order to locally and continuously measure the implemented safeguards during the life-cycle of the communication service.

The fourth step enables the **aggregation** of the local assurance levels delivered continuously by the specified and deployed metrics along the assurance model of the communication service.

The fifth step of **evaluation**, monitor the evolution of the security assurance. By monitoring those fluctuation, both at local and service level, and by assess the consequences against defined constraints it enables administrators to perform the management actions to maintain or recover the target security assurance level for the communication service.

The remaining **presentation** step provides, to administrators, supervision means of the security assurance at the global service level, but also at the most detailed piece of the assurance modelling.
Each step of the operational methodology is presented in more detail in the following sections.

4.6.5.1 Service modelling

The model is a semantic representation of the relevant elements composing a communication service that monitors security assurance. The communication service is not modelled globally, but rather concentrates on the most critical and important security relevant elements of the service infrastructure.

The importance of a constitutive element of a communication service infrastructure should be understood relative to the business impact of a potential perturbation of the service in case a problem arises on an element. This task that consists in prioritizing the business critical element of a service is usually done when applying a risk assessment methodology. The modelling has the purpose of representing this prioritization.

Even if the established model rely on three different views (i.e. hierarchical, topological, flow oriented), representing a communication service infrastructure in three different perspectives, the most important and exploitable view is a hierarchical representation.

The methodology provides no specific process to build a model of a communication service. Nevertheless it is recommended that the model is built by a security expert with solid knowledge in the modelled communication service.

The first step is to decompose the service into main sub-components All infrastructure Objects which are refined are called white box objects as their contribution to the service assurance is understood and actively controlled. In respect to their behaviour in aggregation we distinguish <<Aggregation>> objects, where the assurance value is calculated as the combination of their sub-components and <<Systems>> objects which in addition have metrics directly assigned.

At this level it is recommended to start considering metrics and adding them to the hierarchy view. Owing to the hierarchical nature of the modelling, each of these steps can be repeated until the resulting model reaches a sufficiently realistic representation of the service.

To reduce the initial threshold, which would result from the need to model the entire service at once, the introduction of black box objects is allowed, for which no active metrics are assigned but which have a static assurance level.

4.6.5.2 Metric selection

To be able to assign attribute weights and metrics the method propose using results of risk and impact assessment. A typical risk analysis will result in the identification of risks within the infrastructure along with the assessment of their importance and severity in case of breaches. Location and severity of an impact will lead to the component in the modelling and its relative weight. The type of security safeguards deployed on an infrastructure objects will also lead to the determination of suitable metrics. Based on the findings from Vaughn [244], Seddigh [219] and Swanson [236] we suggest using metric taxonomies for this purpose. From a pragmatically viewpoint we also found that the availability of metrics (i.e. their implementation in a probe) determines the metric assignment.

The assigned metric aims to determine an assurance value for its infrastructure object. Each metric consists of normalized measures of various parts/parameters of security safeguards implemented on infrastructure object(s). One metric is linked to one and only one infrastructure object while one infrastructure object can be measured by several metrics which combine their results.

In order to design those metrics and to guarantee that they can produce normalized assurance levels a standardized metric decomposition is necessary. This decomposition allows simplifying the problem of normalizing the raw measurement results or base measures (in reference to ISO terminology).

4.6.5.3 Measurement

Even if no specific guidelines are provided for the deployment of a measurement infrastructure in communication service infrastructures, some generic remarks and recommendation are provided that a technical expert may follow in the design of a base measurement system for continuous security assurance monitoring.

The continuous assessment of the security assurance is based on the measurement of the conformity, effectiveness and efficiency of security safeguards deployed on operating devices of a communication service infrastructure. But, the method recommends designing a measurement system as least intrusive as it is possible. Indeed, it is not desirable that the measurement system disrupts the security safeguards operation, introduce new threats or impact the measurement results themselves.

In order to maintain this objective, an impact study of the deployed measurement probes and their chosen use (i.e. to which metrics a probe deliver measurement results) on the communication service infrastructure is recommended. For instance, a family of the assurance taxonomy relates to the frequency (i.e. MC_FR: Frequency, see table 2-2) of the measure. If the measurement frequency increases, the confidence will increase. But, it is not recommenced to have such a high frequency in measurement so that the measurement and collect process will consume an important part of the monitored service capacity.

To build such kind of base measurement infrastructure, several approaches are possible:

- A centralized system, that connects directly the core supervision system to each device,
- A distributed system, relaying on a Multi-Agent System,
- A hybrid system.

4.6.5.4 Aggregation

The method goal is to express one security assurance level for a service. As the service is decomposed until the measuring can be started, the method now has to consider how to re-integrate the metric results. To be able to do this the method makes some assumptions:

- First of all, it assumes during modelling that all infrastructure objects are basically independent from each other. This means that it does not have to consider correlation between objects,
- Secondly, it assumes that a metric is specifically adapted to an infrastructure object and will deliver a result that is unique for the infrastructure object. In other words, the assurance level gained from a metric is always correct in scope.

Based on those two assumptions a number of potential algorithms exist. From simple binary aggregation (e.g. min, max) which it may be find unstable (i.e. they have too great variation), to arithmetic average functions.

For completeness, one may find interesting to experiment other algorithms like game theory approaches and differential equations. But, while one may find these algorithm suitable for the problem, it is not negligible to remark the problems of unambiguously derive all the necessary input parameters in a real world scenario.

To aggregate an assurance level AL_{super} for a <<System>> the weighted assurance levels of all subcomponents assurance levels plus the results from the metrics are combined (see Equation 1). To account for the defensive approach the method advocates to take the integer value (i.e. truncate the result) as the assurance level of the <<System>>.

$$AL_{super} = \sum AL_{sub} \times weight_{sub} + (\sum Metric_i \times weight_i) \times (1 - \sum weight_{sub}) \quad (1)$$

The assurance level calculation can be simplified this for an <<Aggregation>> because no metric is assigned and get Equation 2.

$$AL_{super} = \sum AL_{sub} \times weight_{sub} \quad (2)$$

4.6.5.5 Evaluation

Evaluation is the process step where the automatically derived assurance level is compared against the expected value (nominal/actual value comparison). This activity is performed to allow an automatic analysis and support a human decision maker. An automatic system allows for more advanced analysis. Indeed, an automatic system is capable to continuously compare not only the service assurance but also the assurance of lower level infrastructure objects and correlations between objects' assurances.

During the automatic evaluation process, an algorithm checks if an infrastructure object has a nominal assurance level defined in its threshold tag and performs a comparison with the actual value. A rule-based decision process then interprets the result of this comparison and, in case of a mismatch, an alarm is triggered. The rule-based decision allows for various comparison alternatives.

Apart from the immediate operational evaluation, the system could also provide means to compare trends. For that purpose, statistical process control can be used to identify extraordinary fluctuations.

4.6.5.6 Presentation

As the last step in the methodology, the assurance results have to be presented to various user groups that may have different and specific responsibilities. The method distinguishes three main stakeholder groups that are interested in these assurance results:

- The first group is composed of the control centre staff (e.g. from the Network Operation Center or the Security Operation Center), which needs a visualization of the security assurance level,
- The second group is composed of the maintenance people, which have to react to assurance alerts,
- And finally, we found that customers, management and auditors constitute a stakeholder group that is interested in assurance reports.

4.7 SUMMARY

The goal of risk-based or risk-oriented testing is to improve the testing process in order to cover especially risky areas of an application, system or network and to optimize at the same time the use of resources. Basis for risk-based or risk-oriented testing is a substantial risk assessment.

In this deliverable we have evaluated the current state of the art for risk analysis techniques and methods with respect to their suitability for risk-based testing. Basically, risk analysis for critical information technology systems are already successfully implemented for some time. There are a number of methods and process models (e.g. OCTAVE, CRAMM, Trick-light, NIST, FAIR), which outline the specifics of an information security risk analysis for critical systems at large. All process models in common are their very general character. Basis of any risk analysis is the identification, extensive collection and assessment of risk-determining factors such as assets, threats and vulnerabilities. The techniques usually used for this are often not fixed by the process models and can be replaced flexibly. In some cases, there are recommendations for specific techniques (e.g.

the use of attack trees for threat modelling [204]).

For further use in the DIAMONDS project we recommend to use formalized methods of gathering risk-determining factors. The best known methods are the already mentioned attack trees (see Section 4.3). Considerably more extensive are modelling concepts provided by the CORAS method and tool set. CORAS supports modelling the known risk-determining factors such as assets, threats and vulnerabilities and additional allows specifying cross-references and dependencies between the individual factors. Moreover CORAS has its own toolset, which supports modelling in an UML like fashion.

The third chapter examines several classical risk calculation metrics and techniques. We distinguish mainly between qualitative and quantitative metrics. In principle the process models from Section 4.1 can integrate both quantitative and qualitative risk calculation metrics and techniques. On one hand, OCTAVE and NIST especially warn against using quantitative methods without sufficient valid data base. On the other hand, Trick-light encourages using quantitative models as results expressed in expected monetary losses are easier to understand and accept than risk levels described in words. Modelling techniques such as CORAS can be used with both qualitative and quantitative calculation models techniques as Trick-light use a combination of both.

The last chapter highlights known methods of risk-based testing and related topics. The chapter begins with a list of risk-oriented stopping criteria and coverage criteria. Then, we describe various risk-based testing approaches and the approaches to statistical testing. Statistical testing works quite similar to the risk-based testing. It provides optimization methods, which allows test generation based on probability models. In particular, statistical testing approaches are mature and provide adequate tool support. Selected test tools are introduced at the end of the chapter.

4.8 GLOSSARY

Information Security Management System (ISMS)	That part of the overall management system, based on a business risk approach, to establish, implement, operate, monitor, review, maintain and improve information security
Process	Set of interrelated or interacting activities which transforms inputs into outputs
Residual risk	Risk remaining after risk treatment
Risk	Combination of the likelihood of an event and its consequence
Risk analysis	Systematic use of information to identify sources and to estimate the risk.
Risk assessment	Overall process of risk analysis and risk evaluation
Risk avoidance	Decision not to become involved in, or action to withdraw from, a risk situation
Risk criteria	Terms of reference by which the significance of risk is assessed
Risk estimation	Process used to assign values to the probability and consequences of a risk
Risk evaluation	Process of comparing the estimated risk against given risk criteria to determine the significance of the risk

Risk identification	Process to find, list and characterize elements of risk.
Risk management	Coordinated activities to direct and control an organization with regard to risk.
Risk optimization	Process related to a risk to minimize the negative and to maximize the positive consequences and their respective probabilities.
Risk reduction	Actions taken to lessen the probability negative consequences or both, associated with a risk
Risk retention	Acceptance of the burden of loss, or benefit of gain from a particular risk
Risk transfer	Sharing with another party the burden of loss or benefit of gain, for a risk
Threat	A potential source of an incident that may result in adverse changes to an asset, a group of assets or an organization
Vulnerability	Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat

5. REFERENCES

[1] J. Abrial, The B Book: Assigning Programs to Meanings. Cambridge University Press, 1996.

[2] S. Agrawal, C. N. Kanthi, K. V. M. Naidu, J. Ramamirtham, R. Rastogi, S. Satkin, and A. Srinivasan. 2007. "Monitoring infrastructure for converged networks and services". Bell Labs Technical Journal, 12(2):63–77, August 2007.

[3] Gail-Joon Ahn and Ravi S. Sandhu. The RSL99 language for Role-based Separation of Duty Constraints. In ACM Workshop on Role-Based Access Control, 1999.

[4] Bernhard K. Aichernig, Harald Brandl, Elisabeth Jöstl, and Willibald Krenn. Efficient mutation killers in action. In ICST 2011, Fourth International Conference on Software Testing, Verification and Validation, Berlin, Germany, March 21–25 , 2011. IEEE Computer Society, 2011.

[5] Bernhard K. Aichernig, Harald Brandl, and Franz Wotawa. Conformance testing of hybrid systems with qualitative reasoning models. In B. Finkbeiner, Y. Gurevich, and A.K. Petrenko, editors, Proceedings of Fifth Workshop on Model Based Testing (MBT 2009), York, England, 22 March 2009, volume 253 (2) of Electronic Notes in Theoretical Computer Science, pages 53–69. Elsevier, October 2009

[6] Bernhard K. Aichernig and Carlo Corrales Delgado. From faults via test purposes to test cases: on the fault-based testing of concurrent systems. In Luciano Baresi and Reiko Heckel, editors, Proceedings of FASE'06, Fundamental Approaches to Software Engineering, Vienna, Austria, March 27–29, 2006, volume 3922 of Lecture Notes in Computer Science, pages 324–338. Springer-Verlag, 2006.

[7] Alberts, Christopher & C., J. and Dorofee, Audrey. A. J., "OCTAVE Threat Profiles. Software Engineering Institute, Carnegie Mellon University, Criteria Version 2.0", Tech. report CMU/SEI-2001. http://www.cert.org/archive/pdf/OCTAVEthreatProfiles.pdf-TR-016. ESC-TR-2001-016, 2001.

[8] B. Alcalde, A.R. Cavalli, D. Chen, D. Khuu and D. Lee. "Network protocol system passive testing for fault management: A backward checking approach". In FORTE 2004, volume 3235 of Lecture Notes in Computer Science, pages 150-166. Springer, 2004.

[9] B. Alpern and F. B. Schneider. Defining liveness. Information Processing Letters, 21(4):181–185, 1985.

[10] R. Alur, P. Cerny, and S. Zdancewic. Preserving Secrecy Under Refinement. In Proc. of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06), volume 4052 of Lecture Notes in Computer Science, pages 107–118. Springer, 2006.

[11] Amland, Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. Journal of Systems and Software 53(3): 287-295 (2000)

[12] P. Ammann, P. E. Black, and W. Majurski, "Using Model Checking to Generate Tests from Specifications," in ICFEM, 1998, pp. 46-46.

[13] Andres, C., Maag, S., Cavalli, A., Merayo, M. G., and Nunez, M. (2009). Analysis of the OLSR protocol by using formal passive testing. In Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference, APSEC '09, pp. 152-159, Washington, DC, USA. IEEE Computer Society.

[14] Andres, C., Merayo, M. G., and Nunez, M.. Passive testing of timed systems. In Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis, ATVA '08, pp. 418-427, 2008.

[15] Aven, T.: A unified framework for risk and vulnerability analysis covering both safety and security *Reliability Engineering & System Safety,* **2007**, *92*, 745 – 754

[16] Bach, J., James Bach on Risk-Based Testing – How to conduct heuristic risk analysis, Software Testing & Quality Engineering November/December 1999, p. 23-28

[17] Barnett V, Lewis T. Outliers in statistical data. Wiley, ISBN 9780471930945; 1994.

[18] Bauer, T., Böhr, F., Landmann, D., Beletski, T., Eschbach, R. & Poore, J. H.: From Requirements to Statistical Testing of Embedded Systems. Software Engineering for Automotive Systems - SEAS 2007, ICSE Workshops, Minneapolis, USA.

[19] Bayse, E., Cavalli, A., Nuñez, M., and Zaïdi, F. (2005). A passive testing approach based on invariants: application to the WAP. Comput. Networks, 48(2):247-266.

[20] D. E. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, MITRE, 1976.

[21] Ben-Gal, I. Bayesian networks. In: Ruggeri, F., Kenett, R. S., Faltin, F. W. (eds.) Encyclopedia of Statistics in Quality and Reliability. John Wiley & Sons, Chichester (2007).

[22] Fayçal Bessayah, Ana R. Cavalli, Eliane Martins: A formal approach for specification and verification of fault injection process. International Conference on Interaction Sciences (ICIS). pp 883-890, Seoul, Korea, 2009.

[23] Fayçal Bessayah, Ana Cavalli, Willian Maja, Eliane Martins and Andre Willik Valenti. A Fault Injection Tool for Testing Web Services Composition, TAIC PART 2010, Windsor, UK, September 2010.

[24] K. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE, 1977.

[25] R. V. Binder, Testing object-oriented systems: models, patterns, and tools. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[26] Bistarelli, S. J., Fioravanti, F., and Peretti, P., Defense tree for economic evaluations of security investment, in Proc. 1st Int. Conference on Availability, Reliability and Security (ARES'06), pp. 416-423, 2006.

[27] Bob Blakley. An imprecise but necessary calculation. Secure Business Quarterly: Special Issue on Return on Security Investment, 1(2), Q4, 2001. A publication of @stake.

[28] A. Bossi, R. Focardi, D. Macedonio, C. Piazza, and S. Rossi. Unwinding in information flow security. Electronic Notes in Theoretical Computer Science, 99:127–154, 2004.

[29] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement operators and information flow security. In Proc. of the 1st International Conference on Software Engineering and Formal Methods (SEFM'03), pages 44–53. IEEE Computer Society, 2003.

[30] Anthony Boswell. Specification and validation of a Security Policy Model. IEEE Transactions on Software Engineering, 1995.

[31] Boumen, R., de Jong, I., Vermunt, J., van de Mortel-Fronczak, J. M. & Rooda, J.Wawrzyniak, D.: Information security risk assessment model for risk management Management, 2006, 21-30

[32] Bouti, A., K. A.: A state-of-the-art for FMEA/FMECA. Int. J. Reliab. Qual. Saf. Eng. 1, 1994

[33] Harald Brandl, Martin Weiglhofer, and Bernhard K. Aichernig. Automated conformance verification of hybrid systems. In Proceedings of QSIC 2010: the 10th International Conference on Quality Software, Zhangjiajie, China, July 14-15, 2010. IEEE Computer Society, 2010.

[34] Bridges S.M., Vaughn R.B. Fuzzy data mining and genetic algorithms applied to intrusion detection. Proceedings of the National Information Systems Security Conference; 2000. p. 13–31.

[35] Brucker, A.D., Brügger, L., Wolf, B. (2008). Model-Based Firewall Conformance Testing. In TestCom/FATES 2008, pages 103-118, LNCS 5047.

[36] Brucker, A.D., Brügger, L., Kearney, P., Wolf, B. (2010). Verified Firewall Policy Transformations for

Test Case Generation. In ICST 2010, pages 345-354, IEEE Comp. Society.

[37] BUGYO Beyond: Building security assurance in open infrastructures, beyond - 2009.

[38] L. Burdy et al., “An overview of JML tools and applications,” Int. J. Softw. Tools Technol. Transf., vol. 7, no. 3, pp. 212-232, 2005.

[39] J. R. Calame, N. Ioustinova, and J. V. D. Pol, “Automatic Model-Based Generation of Parameterized Test Cases Using Data Abstraction,” in Proceedings of the Doctoral Symposium affiliated with the Fifth Integrated Formal Methods Conference (IFM 2005), 2007, vol. 191, pp. 25-48.

[40] Cansian A.M., Moreira E., Carvalho A., Bonifacio J.M. Network intrusion detection using neural networks. International Conference on Computational Intelligence and Multimedia Applications (ICCMA’97); 1997. p. 276–80.

[41] H. Castaneda. The Paradoxes of Deontic Logic. New Studies in Deontic Logic: Norms, Actions and the Foundations of Ethics, Hingham, MA, Reidel Publishing Company:37-85, 1981

[42] Cavalli, A., Gervy, C., and Prokopenko, S. (2003). New approaches for passive testing using an extended finite state machine specification. Journal of Information and Software Technology, 45:837-852.

[43] Cavalli, A., Lee, D., Rinderknecht, C., and Zaidi, F. (1999). Hit-or-jump: An algorithm for embedded testing with application to IN services. In Wu, J., Chanson, S., and Gao, Q., editors, Formal Method for Protocol Engineering and Distributed Systems, FORTE XII/PSTV XIX'99, volume 156 of IFIP Conference Proceedings, Beijing, China. Kluwer.

[44] Cavalli, A., Maag, S., and de Oca, E. M. (2009). A passive conformance testing approach for a MANET routing protocol. In 24th Annual ACM Symposium on Applied Computing SAC'09.

[45] Cavalli, A., Maag, S., de Oca, E. M., and Zaidi, F. (2009). A formal passive testing approach to test a MANET routing protocol. In Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications, pages 1-6, Washington, DC, USA. IEEE Computer Society

[46] James J. Cebula, L. R. Y. A Taxonomy of Operational Cyber Security Risks Carnegie Mellon, Software Engineering Institute, CERT Program, 2010

[47] CERT: Carnegie Mellon University,Risk-Based Testing: A Case Study. Seventh International Conference on Information Technology: New Generations (ITNG), 2010, 2010, 1032-1037.

[48] Fang Chen and Ravi S. Sandhu. Constraints for Role-Based Access Control. In ACM workshop on Role-Based Access Control, 1995.

[49] Chen, Y. & Probert, R. L.: A Risk-based Regression Test Selection Strategy. Proceeding of the 14th IEEE International Symposium on Software Engineering Institute; http://www.cert.org/

[50] Chen, Y., Probert, R. and Sims, D. P. 2002. Specification-based regression test selection with risk analysis. In Proceedingspicture with a notion of the 2002 Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, September 30 - October 03, 2002). D. A. Stewart and J. H. Johnson, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 1.

[51] Cisco IOS NetFlow technology, http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/product_data_sheet0900aecd80173f71.html, April 2011.

[52] Combs G. and contributors (accessed on 22 January.2008) Wireshark Network Protocol Analyzer, URL: http://www.wireshark.org/

[53] Jason Crampton, Constraints in Role-Based Access Control In: 8th ACM Symposium on Access Control Models and Technologies, 2003

[54] A. Cuadra-Sanchez, C. Casas-Caballero, "End-to-End Quality of Service Monitoring in Convergent IPTV Platforms," Next Generation Mobile Applications, Services and Technologies, International Conference on, pp. 303-308, 2009 Third International Conference on Next Generation Mobile

Applications, Services and Technologies, 2009.

[55] F.Cuppens, N.Cuppens, T.Sans, Nomad: A Security Model with Non Atomic Actions and Deadlines, Proceedings of the 18th IEEE workshop on Computer Security Foundations, 2005

[56] Evgeny Danstin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. ACM Comput. Surv., 2001.

[57] F. Dadeau and R. Tissot, “jSynoPSys -- A Scenario-Based Testing Tool based on the Symbolic Animation of B Machines,” Electron. Notes Theor. Comput. Sci., vol. 253, no. 2, pp. 117-132, 2009.

[58] Darmaillacq, V., Fernandez, J-C., Groz, R., Mounier, L., Richier, J-L. (2006). Test Generation for Network Security Rules. TestCom 2006, pages 341–356, LNCS 3964, 2006.

[59] W.-P. de Roever and K. Engelhardt. Data Refinement: Model-Oriented Proof Methods and their Comparison. Number 47 in Cambridge tracts on theoretical computer science. Cambridge University Press, 1998.

[60] Debar H., Becker M., Siboni, D. A neural network component for an intrusion detection system. IEEE Symposium on Research in Computer Security and Privacy; 1992. p. 240–50.

[61] den Braber, F., Hogganvik, I., Lund, M.. TANDI: Threat . S., Stølen, K. and Vraalsen, F. Model-based security analysis in seven steps – a guided tour to the CORAS method. BT Technology Journal, pages 101-117, Springer, 2007.

[62] Denning DE, Neumann PG. Requirements and model for IDES – a real-time intrusion detection system. Computer Science Laboratory, SRI International; 1985. Technical Report #83F83-01-00.

[63] Report on Detecting Hackers (Analyzing Network Traffic) by Poisson Model Measure. Eighth PIMS-MITACS Industrial Problem Solving Workshop, PIMS-UBC, Vancouver, BC, Canada, May 17-21, 2004.

[64] Dickerson J.E. Fuzzy network profiling for intrusion detection. Proceedings of the 19th International Conference of the North American Fuzzy Information Processing Society (NAFIPS); 2000. p. 301–6.

[65] Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A. R., and Yevtushenko, N. (2010). Fsm-based conformance testing methods: A survey annotated with experimental evaluation. Inf. Softw. Technol., 52:1286-1297

[66] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, Gilles Trouessin, Organization based access control, Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, 2003.

[67] Estevez-Tapiador JM, Garcıa-Teodoro P, Dıaz-Verdejo JE. Stochastic protocol modeling for anomaly based network intrusion detection. Proceedings of IWIA 2003. IEEE Press, ISBN 0-7695-1886-9; 2003. p. 3–12.

[68] Estevez-Tapiador J.M., Garcıa-Teodoro P., Dıaz-Verdejo J.E. Detection of web-based attacks through Markovian protocol parsing. Proc. ISCC05; 2005 p. 457–62.

[69] ETSI/ES 201 873-1 (2007). Methods for Testing and Specification (MTS). The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, v3.2.1. Technical report, ETSI

[70] ETSI/ETSI TS 102 165-1 (2009). Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Methods and protocols; Part 1: Method and proforma for Threat, Risk, Vulnerability Analysis.

[71] M. Felderer, B. Agreiter, R. Breu, and A. Armenteros, “Security Testing by Telling TestStories,” in Modellierung 2010, 2010, vol. 161, pp. 195-202.

[72] Falcone, Y., Fernandez, J-C., Mounier, L., Richier, J-L. (2007). A Compositional Testing Framework Driven by Partial Specifications. TestCom/FATES 2007, pages 107–122, LNCS 4581, 2007.

[73] Firesmith, D.: Security Use Cases. Journal of Object Technology, pp. 53–64, 2003.

[74] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. Journal of Computer Security, 3(1):5–33, 1995.

[75] R. Focardi and F. Martinelli. A Uniform Approach for the Definition of Security Properties. In Proc. of the World Congress on Formal Methods (FM'99), volume 1708 of Lecture Notes in Computer Science, pages 794–813. Springer, 1999.

[76] S. N. Foley. A Universal Theory of Information Flow. In Proc. of the IEEE Symposium on Security and Privacy (S&P'87), pages 116–122. IEEE Computer Society, 1987.

[77] Fox K., Henning R., Reed J., Simonian, R. A neural network approach towards intrusion detection. 13th National Computer Security Conference; 1990. p. 125–34.

[78] Gordon Fraser, Franz Wotawa, Paul Ammann: Testing with model checkers: a survey. Softw. Test., Verif. Reliab. 19(3): 215-261, 2009.

[79] M. C. Gaudel and P. R. James, "Testing Algebraic Data Types and Processes: a unifying theory," Formal Aspects of Computing, vol. 10, pp. 436-451, 1998.

[80] Gao, Debin, Reiter, Michael K., and Song, Dawn, "Behavioral Distance Measurement Using Hidden Markov Models" (2006). Department of Electrical and Computing Engineering. Paper 22.

[81] P. Garcıa-Teodoro, J. Dıaz-Verdejo, G. Macia-Fernandez, E. Vazquez, Anomaly-based network intrusion detection: Techniques, systems and challenges . Computers & security N° 28 (2009) pages 18–28, Elsevier 2009.

[82] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In Proc. of the 1982 IEEE Symposium on Security and Privacy (S&P'82), pages 11–20. IEEE Computer Society, 1982.

[83] J. A. Goguen and J. Meseguer. Inference Control and Unwinding. In Proc. of the IEEE Symposium on Security and Privacy (S&P'84), pages 75–86. IEEE Computer Society, 1984.

[84] Lawrence A. Gordon and Martin P. Loeb: Return on information security investments: Myths vs. realities. Strategic Finance, pages 26–31; November 2002.

[85] J. Graham-Cumming and J. W. Sanders. On the refinement of non-interference. In Proc. of the IEEE Computer Security Foundations Workshop, pages 35–42. IEEE Computer Society Press, 1991.

[86] Thomas Gross, Context-Based Access Control, IBM Zurich Research Laboratory, PhD thesis, 2003

[87] Grunske, L., Lindsay, P. A., Yatapanage, N. & Winter, K.: An Automated Failure Mode and Effect Analysis Based on High-Level Design Specification with Behavior Trees; IFM, 2005, 129-149

[88] Gu Tian-yang, Shi Yin-sheng & Yuan, F. Y.: Research on Software Security Testing World Academy of Science, Engineering and Technology 69 2010, 2010

[89] D. Harel, "Statecharts: A visual formalism for complex systems," Sci. Comput. Program., vol. 8, no. 3, pp. 231-274, 1987.

[90] D. Harel and P. S. Thiagarajan, "Message sequence charts," pp. 77-105, 2003.

[91] Harel, E., Lichtenstein, O., and Pnueli, A. 1990. Explicit-clock temporal logic. In Proceedings of the 5th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, New York, pp. 402-413

[92] Harpes C., Adelsbach A., Zatti S., Peccia N., Quantitative Risk Assessment with ISAMM on ESA's Operations Data System, In "4th ESA International Workshop on Tracking, Telemetry and Command Systems for Space Applications", 2007

[93] May Haydar. A formal framework for Run-Time verification of Web based applications using Scope-Extended LTL, published by VDM Verlag, 2009.

[94] R. J. Hayton, J. M. Bacon, and K. Moody. Access control in an Open Distributed Environment, 1998.

[95] Heckerman D. A tutorial on learning with Bayesian networks. Microsoft Research; 1995. Technical Report MSRTR-95-06.

[96] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-preserving refinement. In Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01), pages 295–306. IEEE Computer Society, 2001.

[97] Herzog, P.: OSSTMM 2.1. Open-Source Security Testing Methodology Manual. Institute for Security and Open Methodologies, 2003

[98] Hewlett Packard: Secure your network, 2010 Top Cyber Security Risks Report; http://dvlabs.tippingpoint.com/toprisks2010

[99] Hoare, C. A. R. Communicating Sequential Processes. Prentice-Hall, 1985.

[100] C.A.R. Hoare, An Axiomatic basis for Computer Programming, In Comm. ACM, 12(10), 1969

[101] Howard, M. & Leblanc, D. E.: Writing Secure Code; Microsoft Press, 2002

[102] Howard, R. A., Dynamic Probabilistic Systems: Volume 1: Markov models: John Wiley & Sons, 1971.

[103] Michael Huth and Mark Ryan, Logic in Computer Science (Second Edition). Cambridge University Press, 2004

[104] D. Hutter. Possibilistic Information Flow Control in MAKS and Action Refinement. In Proc. of Emerging Trends in Information and Communication Security, International Conference (ETRICS'06), volume 3995 of Lecture Notes in Computer Science, pages 268–281. Springer, 2006.

[105] IEC60300-3-9, Dependability management - Part 3: Application guide - Section 9: Risk analysis of technological systems - Event Tree Analysis (ETA), 1995.

[106] IEC61078, Analysis techniques for dependability - Reliability block diagram method, 1991.

[107] IEC61165, Application of Markov techniques, 1995.

[108] IEC/FDIS 31010:2009, Risk management — Risk assessment of network data and information Proceedings of SPIE - The techniques, 2009.

[109] The Institute of Electrical and Electronic Engineers (1990) IEEE Standard Glossary of Software Engineering Terminology

[110] International Society for Optical Engineering, Citeseer, 2006, 6242, 114–129Telecommunication Union. Recommendation Z.100 — Specification and description language (SDL), 1999.

[111] ISO 9646-1 (1994). Information technology - open systems interconnection - conformance testing methodology and framework - part 1: General concepts. Technical report, ISO

[112] ISO/JTC1/SC27: ISO/IEC 15408 Information technology -- Security techniques -- Evaluation criteria for IT security: Part 1: Introduction and general model (2009), Part 2: Security functional components (2008), Part 3: Security assurance components (2008)

[113] ISO/IEC 27000:2009: Information technology — Security techniques — Information security management systems — Overview and vocabulary. http://standards.iso.org/ittf/PubliclyAvailableStandards/c041933_ISO_IEC_27000_2009.zip

[114] ISO/IEC 27001 (2005), Information technology – Security techniques. Information security management system.

[115] ISO/IEC 17799 (or 27002), Information technology – Security techniques. Code of practice for information security management (15/06/2005 renamed in ISO/IEC 27002 at 01/07/2007).

[116] ISO/IEC 27005 (2008), Information technology – Security techniques. Information security risk management.

[117] J. Jacob. On the derivation of secure components. In Proc. of the IEEE Symposium on Security and

Privacy (S&P'89), pages 242–247. IEEE Computer Society, 1989.

[118] A. Jamdagni, Z. Tan, P. Nanda, X. He, R. Liu, Mahalanobis Distance Map Approach for Anomaly Detection of Web-based Attacks, The 8th Australian Information Security Management Conference, Perth, Australia, November 30 - December 2, 2010, pp.8-17. ISBN: 978-0-7298-0689-3

[119] C. Jard and T. Jéron, "TGV: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems," Int. J. Softw. Tools Technol. Transf., vol. 7, no. 4, pp. 297-315, 2005.

[120] B. Jeannet, T. Jéron, V. Rusu, and E. Zinovieva, "Symbolic Test Selection Based on Approximate Analysis," in TACAS, 2005, pp. 349-364.

[121] J-Flow, http://www.juniper.net/, April 2011

[122] S. Jha, K. Tan, R. A. Maxion. Markov Chains, Classifiers, and Intrusion Detection. Proceedings of the 14th IEEE workshop on Computer Security Foundations, 2001.

[123] Jia Y. and Harman M. An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering., 2010, in press.

[124] C. B. Jones, Systematic software development using VDM (2nd ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990.

[125] Jones, Jack A.: An Introduction to Factor Analysis of Information Risk (FAIR); http://www.riskmanagementinsight.com/media/docs/FAIR_introduction.pdf

[126] J. Julliand, P. Masson, and R. Tissot, "Generating Tests from B Specifications and Test Purposes," in ABZ'2008, Int. Conf. on ASM, B and Z, London, UK, 2008, vol. 5238, pp. 139-152.

[127] J. Jürjens. Secure systems development with UML. Springer, 2005.

[128] Kaksonen, Rauli. A Functional Method for Assessing Protocol Implementation Security (Licentiate thesis). 2001. Espoo. Technical Research Centre of Finland, VTT Publications 447. 128 p. + app. 15 p. ISBN 951-38-5873-1 (soft back ed.) ISBN 951-38-5874-X (on-line ed.).

[129] J.G. Kemmeny and J.L. Snell: Finite Markov Chains; New York, NY: Springer, 1976.

[130] Knuth, D., Morris, J., and Pratt, V. (1977). Fast pattern matching in strings. SIAM Journal on Computing, 6(1):323-350

[131] Kruegel C., Mutz D., Robertson W., Valeur F. Bayesian event classification for intrusion detection. Proceedings of the 19th Annual Computer Security Applications Conference; 2003.

[132] Y. Ledru, L. du Bousquet, O. Maury, and P. Bontron, "Filtering TOBIAS Combinatorial Test Suites," in Fundamental Approaches to Software Engineering, 7th International Conference, FASE 2004, 2004, vol. 2984, pp. 281-294.

[133] D. Lee, D. Chen, R. Hao, R. E. Miller. J. Wu and W. Yin. "A formal approach for passive testing of protocol data portions". In ICNP conference, pages 122-131. IEEE Computer Society, 2002.

[134] D. Lee and M. Yannakakis, "Testing Finite-State Machines: State Identification and Verification," IEEE Transactions on Computers, vol. 43, pp. 306-320, 1994.

[135] Lee, D. and Yannakakis, M. (1996). Principles and Methods of Testing Finite State Machines - a Survey. In The Proceedings of IEEE, volume 84, pages 1090-1123.

[136] Leucker, M. and Schallhart, C. (2009). A Brief Account of Runtime Verification. Journal of Logic and Algebraic Programming (JLAP), 78(5):293-303.

[137] Li W. Using genetic algorithm for network intrusion detection. C. S.G. Department of Energy; 2004. p. 1–8.

[138] Yihua Liao, V. Rao Vemuri. Use of K-Nearest Neighbor classifier for intrusion detection; Computers & Security Vol 21, No 5, pp 439-448, 2002; Elsevier Science Ltd

[139] Lindley, D. V., Introduction to Probability and Statistics from a Bayesian Viewpoint: Cambridge University Press, 1965.

[140] G. Lowe. Quantifying Information Flow. In Proc. of the 15th IEEE Computer Security Foundations Workshop (CSFW'02), pages 18–31. IEEE Computer Society, 2002.

[141] G. Lowe. On information flow and refinement-closure. In Proc. of the Workshop on Issues in the Theory of Security (WITS '07), 2007.

[142] Lund, M. S., Solhaug, B., Stølen, K. Model-Driven Risk Analysis. The CORAS Approach. 1st Edition., 2011, XVI, 400 p., Hardcover. ISBN: 978-3-642-12322-1 (To appear).

[143] Mahoney M.V., Chan P.K. Learning nonstationary models of normal network traffic for detecting novel attacks. Proceedings of the Eighth ACM SIGKDD; 2002. p. 376–85.

[144] Mallouli, W., Orset, J-M., Cavalli, A.R, Cuppens-Boulahia, N. Cuppens, F. (2007). A formal approach for testing security rules. SACMAT 2007, 12th ACM Symposium on Access Control Models and Technologies, pages 127-132, June 2007, ACM.

[145] H. Mantel. Possibilistic Definitions of Security - An Assembly Kit. In Proc. Of the IEEE Compuer Security Foundations Workshop (CSFW'00), pages 185–199. IEEE Computer Society, 2000.

[146] H. Mantel. Preserving Information Flow Properties under Refinement. In Proc. of the IEEE Symposium on Security and Privacy (S&P'01), pages 78–91. IEEE Computer Society, 2001.

[147] Masse, T., O'Neil, S. & Rollins, J.. The Department of Homeland Security's Risk Assessment Methodology: Evolution, Issues, and Options for Congress *The Department of Homeland Security's Risk Assessment Methodology,* 2007.

[148] Pierre-Alain Masson, Jacques Julliand, Jean-Christophe Plessis, Eddie Jaffuel, Georges Debois. Automatic Generation of Model Based Tests for a class of Security Properties. In: Proceedings of the 3rd International Workshop on Advances in model-based testing, NY, USA, 2007.

[149] Pierre-Alain Masson, Marie-Laure Potet, Jacques Julliand, Régis Tissot, Georges Debois, Bruno Legeard, Boutheina Chetali, Fabrice Bouquet, Eddie Jaffuel, Lionel Van Aertrick, June Andronick, and Amal Haddad. An Access Control Model Based Testing Approach for Smart Card Applications: Results of the POSÉ Project. JIAS, Journal of Information Assurance and Security, 5(1):335-351, 2010.

[150] Massood, A., Bhatti, R., Mathur, A.. Scalable and Effective Test Generation for Role-Based Access Control Systems. IEEE Trans. on Software Engineering, 35(5)) pages 654-668, Sept./Oct. 2009.

[151] Massood, A., Ghafoor, A., Mathur, A.. Conformance Testing of Temporal Role-Based Access Control Systems. IEEE Trans. on Dependable and Secure Computing, 7(2) pages 144-158, April-June 2010.

[152] Mauw, S. & Oostdijk, M.: Foundations of Attack Trees. Information Security and Cryptology - ICISC 2005, Springer Berlin / Heidelberg, 2006, 3935, 186-198

[153] MaTeLo, All4Tec In., http://all4tec.com/ (last visited April 5, 2011)

[154] V. Mattord. "Principles of Information Security". Course Technology. pp. 290–301. ISBN 9781423901778. 2008.

[155] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In Proc. of the IEEE Symposium on Security and Privacy (S&P'87), pages 161–166. IEEE Computer Society, 1987.

[156] D. McCullough. Noninterference and the composability of security properties. In Proc. of the IEEE Symposium on Security and Privacy (S&P'88), pages 177–186. IEEE Computer Society, 1988.

[157] J. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In Proc. of the IEEE Symposium on Research in Security and Privacy (S&P'94), pages 79–93. IEEE Computer Society, 1994.

[158] J. McLean. Security models. In Encyclopedia of Software Engineering. John Wiley & Sons, 1994.

[159] Peter Mell, Karen Scarfone, S. R.: A Complete Guide to the Common Vulnerability Scoring System, 2007.

[160] R. Meyden and C. Zhang. A Comparison of Semantic Models for Noninterference. In Proc. of the 4th International Workshop on Formal Aspects in Security and Trust (FAST'06), Lecture Notes in Computer Science, pages 235–249. Springer, 2007.

[161] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. Neurocomputing 55(1–2): 169–186, 2003.

[162] G. J. Myers and C. Sandler, The Art of Software Testing. John Wiley & Sons, 2004.

[163] Michael, C. C. & Radosevich, W.: Risk-Based and Functional Security Testing; Cigital, Inc., 2005

[164] Microsoft Corporation: Security Development Lifecycle. Version 5.0 (2010).

[165] J. K. Millen. Hookup Security for Synchronous Machines. In Proc. of the IEEE Computer Security Foundations Workshop (CSFW'90), pages 84–90. IEEE Computer Society, 1990.

[166] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[167] Mitre Corporation: "Common vulnerabilities and exposures", Website:http://cve.mitre.org/ (accessed Mar 2010).

[168] Mohr, R. R.: Failture Modes and Effect Analysis; February 2002

[169] Gerardo Morales, Stéphane Maag, Ana Cavalli, Wissam Mallouli, Edgardo Montes de Oca, Bachar Wehbi. Time Extended Invariants for the passive Testing of Web Services. In: IEEE International Conference on Web Services, Miami, Florida, 2010.

[170] Murthy, K. K., Thakkar, K. R. & Laxminarayan, S.: Leveraging Risk Based Testing in Enterprise Systems Security Validation. Proc. First Int Emerging Network Intelligence Conf, 2009, 111-116.

[171] Nielsen, D. S., The Cause/Consequence Diagram Method as a Basis for Quantitative Accident Analysis: Danish Atomic Energy Commission, RISO-M-1374, 1971.

[172] NIST, "National Vulnerability Database", NVD Website: http://nvd.nist.gov/ (accessed Mar 2010).

[173] Steven Noel, Sushil Jajodia, Lingyu Wang, Anoop Singhal. Measuring Security Risk of Networks using Attack Graphs. In *International Journal of Next-Generation Computing*, Vol. 1, No.1, July 2010.

[174] Object Management: Concepts Group. UML Superstructure Specification Version 2.2, formal/2009-02-02, 2009.

[175] C. O'Halloran. A calculus for information flow. In Proc. of the 1st European Symposium on Research in Computer Security (ESORICS'90), pages 147–159. AFCET, 1990.

[176] D. Oheimb. Information Flow Control Revisited: Noninfluence = Noninterference + Nonleakage. In Proc. of the 9th European Symposium on Research Computer Security (ESORICS'04), Lecture Notes in Computer Science, pages 225–243. Springer, 2004.

[177] Open Web Application Security Project. OWASP Testing Guide version 3.0, December 2008.

[178] A. Orebaugh, G. Ramirez, J. Burke, L. Pesce, "Wireshark & Ethereal Network Protocol Analyzer Toolkit" (Jay Beale's Open Source Security), Syngress Publishing, 2006.

[179] OSV Database: "Open source vulnerability database", Website:http://osvdb.org/ (accessed Mar 2010).

[180] OUSPG, (accessed on 4 December 2007) PROTOS Test-Suite:c07-sip,
URL: http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/#h-ref15

[181] M. Panda and M.R. Patra. 2007. Network intrusion detection using naive bayes. IJCSNS International Journal of Computer Science and Network Security, 7, 258–263.

[182] I. Parissis, “A Formal Approach to Testing LUSTRE Specifications,” in Proceedings of the 1st International IEEE Conference on Formal Engineering Methods, Hiroshima, Japan, 1997.

[183] Perrow, C., Normal accidents: living with high-risk technologies, Princeton University Press, 1999.

[184] A.Pnueli, The temporal logic of programs. Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS), 1977

[185] Portnoy L., Eskin E., Stolfo S.J. Intrusion detection with unlabeled data using clustering. Proceedings of The ACM Workshop on Data Mining Applied to Security; 2001.

[186] Henry Prakken. Dialectical Proof Theory for Defeasible Argumentation with Defeasible Priorities (Preliminary Report). In John-Jules Ch. Meyer and Pierre-Yves Schobbens, editors, ModelAge Workshop, volume 1760 of Lecture Notes in Computer Science. Springer, 1997.

[187] Henry Prakken and Marek J. Sergot. Contrary-to-duty Obligations. Studia Logica, 1996.

[188] Pressman, Roger S., Ince Darrel (1994) Software Engineer A Practitioner's Approach. McGRAW-HILL Book Company Europe, England. 915 p.

[189] Prowell, S. J.: JUMBL: A Tool for Model-Based Statistical Testing. Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Society Press, 2003

[190] Rapaka, A., Novokhodko, A., Wunsch, D.. Intrusion detection using radial basis function network on sequences of system calls, Neural Networks, 2003. Proceedings of the International Joint Conference on, Volume: 3, pp. 1820-1825, July 2003.

[191] Rausand, M.: HAZOP Hazard and Operability Study. System Reliability Theory (2nd ed), Wiley, 2004

[192] Rausand, M.: Preliminary Hazard Analysis. System Reliability Theory (2nd ed), Wiley, 2004

[193] Rausand, M. and Høyland, A., System reliability Theory: Models, Statistical Methods, and Applications. Richard D. Irwin, Inc., Homewood, Illinois, USA, 1974, 2 ed: Wiley, 2004.

[194] Redmill, F. 2004. Exploring risk-based testing and its implications: Research Articles. Softw. Test. Verif. Reliab. 14, 1 (Mar. 2004), 3-15.

[195] Redmill, F. 2005. Theory and practice of risk-based testing: Research Articles. Softw. Test. Verif. Reliab. 15, 1 (Mar. 2005), 3-20.

[196] W. Reisig, Petri nets: an introduction. New York, NY, USA: Springer-Verlag New York, Inc., 1985.

[197] RFC1157, Simple Network Management Protocol (SNMP), URL: http://tools.ietf.org/html/rfc1157

[198] RFC5440, Path Computation Element (PCE) Communication Protocol (PCEP), IETF, 2009

[199] Rontti T. (2004) Robustness Testing Code Coverage Analysis. Master’s Thesis. The University of Oulu, Department of Electrical and Information Engineering, Oulu, Finland

[200] Roschke, S., Cheng, F., Schuppenies, R., and Meinel, Ch.: “Towards Unifying Vulnerability Information for Attack Graph Construction”, In: Proceedings of 12th Information Security Conference (ISC’09), Springer LNCS, vol. 5735, pp. 218-233, Pisa, Italy (Sep 2009).

[201] Roschke, S., Cheng, F., Meinel, Ch.:Using Vulnerability Information and Attack Graphs for Intrusion Detection In: Proceedings of 6th International Conference on Information Assurance and Security (IAS'10), IEEE Press, Atlanta, United States, pp. 104-109 (August 2010).

[202] A. W. Roscoe. CSP and determinism in security modelling. In Proc. of the IEEE Symposium on Security and Privacy (S&P’95), pages 114–127. IEEE Computer Society, 1995.

[203] J. Rumbaugh, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education, 2004.

[204] Saitta, P., Larcom, B. & Eddington, M.: Trike v.1 Methodology Document; 2005

[205] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access

control Models. IEEE Computer, 1996.

[206] The SANS (SysAdmin, Audit, Network, Security) Institute: The Top Cyber Security Risks; http://www.sans.org/top-cyber-security-risks/

[207] T. Santen. A Formal Framework for Confidentiality-Preserving Refinement. In Proc. of the 11th European Symposium on Research in Computer Security (ESORICS'06), volume 4189 of Lecture Notes in Computer Science, pages 225–242. Springer, 2006.

[208] T. Santen. Preservation of probabilistic information flow under refinement. Information and Computation, 206(2-4):213–249, 2008.

[209] T. Santen, M. Heisel, and A. Pfitzmann. Confidentiality-Preserving Refinement is Compositional - Sometimes. In Proc. of the 7th European Symposium on Research in Computer Security (ESORICS'02), volume 2502 of Lecture Notes in Computer Science, pages 194–211. Springer, 2002.

[210] Sauve, J.: Risk-based service testing confidence. In Proc. 3rd IEEE/IFIP Int. Workshop Business-driven IT Management BDIM 2008, 2008, 106-109

[211] Schechter, S. E.. Computer security strength and risk: a quantitative approach. Harvard University, 2004

[212] Schechter, S. E.: Toward Econometric Models of the Security Risk-Based Stopping Criteria for Test Sequencing from Remote Attacks; IEEE Transactions Security and Privacy, 2004, 3, 2005.

[213] S. Schneider. May Testing, Non-Interference, and Compositionality. Electronic Notes in Theoretical Computer Science, 40:30–50, 2000.

[214] Schneier, B., Attack trees: Modeling security threats, Dr. Dobb's Journal, vol. 24 (12), pp.21-29, 1999.

[215] Schneier, B., Secrets & Lies: Digital Security in a Networked World: John Wiley & Sons, 2000.

[216] Schneier, B: Why Management Doesn't Get IT Security http://www.schneier.com/blog/archives/2006/11/why_management.html

[217] Robert Schuppenies, Automatic Extraction of Vulnerability Information for Attack Graphs, Ph.D Thesis, 2009.

[218] Secunia Advisories, Website: http://secunia.com/advisories/ (accessed Mar 2010).

[219] Seddigh N., Pieda P., Matrawy A., Nandy B., Lambadaris J., Hatfield A. – "Current trends and advances in information assurance metrics", In Proceedings of Privacy-Security-Trust 2004, 2004

[220] Senn, D., Basin, D., Caronni, G. (2005). Firewall conformance testing. In TestCom 2005, pages 226-241, LNCS 3502.

[221] Sequeira K., Zaki M. ADMIT: anomaly-based data mining for intrusions. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2002. p. 386–95.

[222] sFlow, http://www.sflow.org/, April 2011.

[223] SHIELDS D2.2: Initial Modelling Methods and Prototype Modelling Tools, deliverable of the SHIELDS research project within the European Community's Seventh Framework Programme, 2008, http://www.shields-project.eu

[224] Sindre, G. and Opdahl, A. L., Eliciting security requirements with misuse cases. Requirements Engineering Journal. 10(1), pp. 34-44. Springer, 2005.

[225] R. Sommer and V. Paxson, Outside the Closed World: On Using Machine Learning For Network Intrusion Detection, Proceedings of the IEEE Symposium on Security and Privacy 2010, Oakland, California, pp. 305-316, May 2010

[226] Sonnenreich W.: Return On Security Investment (ROSI): A Practical Quantitative Model. A summary

of Research and Development conducted at SageSecure (2002)

[227] Kevin J. Soo Hoo. How Much Is Enough? A Risk-Management Approach to Computer Security. PhD thesis, Stanford University, June 2000.

[228] Souza, E.; Gusmao, C.; Alves, K.; Venancio, J. & Melo, R.: Measurement and control for risk-based test cases and activities. Proc. 10th Latin American Test Workshop LATW '09, 2009, 1-6.

[229] Souza, E., Gusmao, C. & Venancio, John Wack, Miles Tracy, M. S.: Guideline on Network Security Testing -- Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-42, 2003

[230] J. M. Spivey. An Introduction to Z and Formal Specifications. J. Software Engineering, 1989.

[231] J. M. Spivey, The Z notation: a reference manual. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1992.

[232] Shamik Sural, Shamik Sural and Vijayalakshmi Atluri, Security analysis of GTRBAC and its variants using model checking, Computers and security, Elsevier, 2010.

[233] Stallbaum, H., Metzger, A. & Pohl, K.: An automated technique for risk-based test case generation and prioritization. Proceedings of the 3rd international workshop on Automation of software test, ACM, 2008, 67-70.

[234] D. Sutherland. A model of information. In Proc. of the 9th National Computer Security Conference, pages 175–183, 1986.

[235] Sutton M., Greene A. Amini P. (2007) Fuzzing Brute Force Vulnerability Discovery. Pearson Education, Inc. United States. 543 p.

[236] Swanson M., Bartol N., Sabato J., Hash J., Graffo L. – “Security metrics guide for information technology systems”, Technical Report NIST 800-55, National Institute of Standards and Technology, July 2003

[237] Swiderski, F., Synder, W.: Threat Modeling. Microsoft Press. Redmond (2004)

[238] Takanen A., DeMott J., Miller C. (2008) Fuzzing for Software Security and Quality Assurance. Artech House Inc., Norwood, MA. United States of America. 230 p.

[239] Taubenberger, S. & Jürjens, J.: IT Security Risk Analysis based on Business Process Models enhanced with Security Requirements

[240] J. Tretmans, “Model Based Testing with Labelled Transition Systems,” in Formal Methods and Testing, 2008, vol. 4949, pp. 1-38.

[241] Untinen T. J., (2008) Network Fuzzer Instrumentation in Software Testing. University of Oulu, Department of Electrical and Information Engineering, Master’s Thesis, pp. 82.

[242] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[243] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches,” Software Testing, Verification and Reliability, 2011.

[244] Vaughn R., Henning R., Siraj A. – “Information assurance measures and metrics – state of practice and proposed taxonomy”, In Proceedings of the 36th Hawaii International Conference on System Sciences, 2002

[245] Wang, Y.-M., Liu, Z.-L., Cheng, X.-Y. & Zhang, K.-J.: An analysis approach for multi-stage network attacks. Proceedings of 2005 International Conference on Machine Learning and Cybernetic Vol. 7, pp. 3949–3954, IEEE.

[246] J. Warmer and A. Kleppe, The Object Constraint Language: Precise Modeling with UML. Addison-Wesley, 1999.

[247] Wawrzyniak, D., Fischer-Huebner, S., Furnell, S. & Lambrinoudakis, C. (Eds.) Information Security Risk Assessment Model for Risk Management Trust and Privacy in Digital Business, Springer Berlin / Heidelberg, 2006, 4083, 21-30

[248] Martin Weiglhofer, Bernhard Aichernig, and Franz Wotawa. Fault-based conformance testing in practice. International Journal of Software and Informatics, 3(2–3):375–411, June/September 2009.

[249] J. T. Wittbold and D. M. Johnson. Information Flow in Nondeterministic Systems. In Proc. of the IEEE Symposium on Security and Privacy (S&P'90), pages 144–161. IEEE Computer Society, 1990.

[250] I. H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (2nd edition). Morgan Kaufmann, 2005.

[251] Yao, M. Y., Petrenko, A., and von Bochmann, G. (1994). Fault coverage analysis in respect to an fsm specification. In INFOCOM, pages 768-775.

[252] Yeung DY, Ding Y. Host-based intrusion detection using dynamic and static behavioral models. Pattern Recognition 2003:36(1): 229–43.

[253] W. D. Young and W. R. Bevier. A State-Based Approach to Non-Interference. In Proc. of the IEEE Computer Security Foundations Workshop (CSFW'94), pages 11–21. IEEE Computer Society, 1994.

[254] Zafar, S. & Dromey, R. G.: 2005, A.-P. S. E. C. (Ed.). Integrating Safety and Security Requirements into Design of an Embedded System. IEEE Computer Society Press, 2005, 629-636.

[255] A. Zakinthinos and E. S. Lee. A general theory of security properties. In Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy (S&P'97), pages 94–102. IEEE Computer Society, 1997.

[256] Zech, P.: Risk-Based Security Testing in Cloud Computing Environments. PhD Symposium at the Fourth IEEE International Conference on Software Testing, Verification and Validation (ICST), 2011 Trust Management (IFIPTM'2009), pages 215-233, Springer, 2009.

[257] Zhang Lufeng, Tang Hong, Cui, YiMing, Zhang JianBo. Network security evaluation through Attack Graph Generation. In *World Academy of Science, Engineering and Technology*, 2009.

[258] Zimmermann, F., Eschbach, R., Kloos, J. & Bauer, T.: Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems Proceedings of 12th European Workshop on Dependable Computing, HAL - CCSD, 2009.