Project no: 100204

p-SHIELD

pilot embedded Systems architecture for multi-Layer Dependable solutions

Instrument type: Capability Project

Priority name: Embedded Systems (including RAILWAYS)

D2.2.2: SPD metrics specifications

Due date of deliverable: M15
Actual submission date: M17

Start date of project: 1st June 2010 Duration: 19 months

Organisation name of lead contractor for this deliverable: pSHIELD Consortium

Revision [Draft B]

Project co-funded by the European Commission within the ARTEMIS-JU (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X

Document Authors and Approvals			
Authors		**Date**	**Signature**
Name	**Company**		
Andrea Morgagni	Selex Elsag		
Renato Baldelli	Selex Elsag		
Spase Drakul	THYIA		
Nastja Kuzmin	THYIA		
Iñaki Eguia	Tecnalia		
Andrea Mogani	Selex Elsag		
Reviewed by			
Name	**Company**		
Pedro Polónia	Critical Software		
Andrea Fiaschetti	University of Rome		
Approved by			
Name	**Company**		
Iñaki Eguia	Tecnalia		

Modification History		
Issue	**Date**	**Description**
Draft A	08/06/2011	First issue for comments
Draft B	28/07/2011	Second issue for comments
Draft B	15/09/2011	Review
Draft B	15/11/2011	Final Version

Contents

Figures

Tables

1 Executive Summary

The main aim of this task will be the identification of SPD metrics. As a matter of fact, for the SPD needs, metrics are required for the measurement of security, dependability, reliability, trust and reputation, availability, privacy, anonymity and traceability as well as, for all the levels (node, network communication, middleware, applications). The proposed metrics will also be based on the scenario identified in Task 6.4.

Task 2.2 aims at developing the basis for system interoperation on all levels (node, network and middleware). In order to pursue such an aim, another result of this task shall be metrics and standards for the interoperation of nodes and systems, which shall be part of the future standardization for such systems. As also influences on legislative issues might be possible, special reports may extend the task deliveries in case of detection of such issues.

The final good will be a coherent and clear description of the SPD metrics specifications, acceptable by all partners. Within the project, this task builds the basis for all subsequent steps by providing some standard metrics for the integration and test of the specific components/subsystems which are implemented for demonstration purposes.

As for Task 2.1, this task will provide a preliminary description of SPD metrics to influence the prototype development in WP3, WP4, and WP5, to start the SPD lifecycle activities in WP6 and to provide support to the validation phase. After the integration of the preliminary prototypes a refinement of the SPD metrics will be done taking into account the application scenario.

The structure and content of the document are the following:

- Chapter 1 – Purpose of the document and its structure
- Chapter 2 – Brief introduction
- Chapter 3 – Taxonomy. In this document, a pSHIELD taxonomy and his relation with other existing safety taxonomies to classify and analyze human error and accident causes may be created, improved or completed in the next version. Interesting taxonomies related to pSHIELD include the Human Factors Analysis and Classification System based on Reason's Swiss Cheese Model, the CREAM (Cognitive Reliability Error Analysis Method) and the taxonomy used by CIRAS (Confidential Incident Railway Analysis System) in the UK rail industry (useful for pSHIELD scenario).http://www.ciras.org.uk/Pages/Home.aspx
- Chapter 4 – Introduction of a framework for integrating dependability and security
- Chapter 5 – SPD functions measure calculation for basic components based on reduction of different classified faults
- Chapter 6 – pSHIELD functional component description
- Chapter 7 – SPD functions for reducing NFUA metric construction method description
- Chapters 8 and 9 – How to quantify the SPD measure of composed SPD functions, using medieval castle example and system for security assurance assessment
- Chapter 10 – Future work
- Chapter 11 – Conclusions

- Chapter 12 – Annex A – SPD functions catalogue
- Chapter 13 – Annex B
- Chapter 14 – References

2 Introduction

Measurement of Security, Privacy and Dependability (SPD) functions, both qualitatively and quantitatively, has been a long standing challenge to the research community and is of practical import to IT industry today. IT industry has responded to demands for improvement in security, privacy and dependability by increasing effort for creating what in general can be defined as "more dependable" products and services. How industry can determine whether this effort is paying off and how consumers can determine whether industry's effort has made a difference is a question which has not yet a clear answer. That's why the determination of security privacy and dependability metrics is not a trivial task. There may be different quantifiable representations of SPD metrics [8].

In this document we propose a main method to be totally defined and implemented into pSHIELD infrastructure, which takes inspiration from Common Criteria (CC) standard [5] [6] to ensure a consistently measured and expressed as a cardinal number SPD metric for embedded systems.

The CC permits comparability between the results of independent security evaluations. The CC does so by providing a common set of requirements for the security functionality of IT products and for assurance measures applied to these IT products during a security evaluation. These IT products may be implemented in hardware, firmware or software. The evaluation process establishes a level of confidence that the security functionality of these IT products and the assurance measures applied to these IT products meet these requirements. The evaluation results may help consumers to determine whether these IT products fulfill their security needs.
The CC addresses protection of assets from unauthorised disclosure, modification, or loss of use. The categories of protection relating to these three types of failure of security are commonly called confidentiality, integrity, and availability, respectively. The CC may also be applicable to aspects of IT security outside of these three. The CC is applicable to risks arising from human activities (malicious or otherwise) and to risks arising from non-human activities. Apart from IT security, the CC may be applied in other areas of IT, but makes no claim of applicability in these areas.

We don't apply CC standard exactly as it's defined in ISO 15408 but we adapt the CC philosophy to the particular demands of the pSHIELD technical annex. In particular we define:

- The SPD metric of each pSHIELD component implementing a SPD function as it's described in CC vulnerability assessment. In this way we derive a SPD metric following the approach of a rigorous international standard

- The formal description of pSHIELD SPD functions according to CC security functional requirements which are a standard way of expressing the SPD functional requirements. Annex A of this document catalogues a potential set of functional components, families, and classes. **Moreover in D6.2 document (chapter 4), it is defined the set of SPD functional components that are implemented in the pSHIELD integrated platform for verification**.

In order to apply CC concepts we introduce a taxonomy and framework for integrating dependability and security. Only apparently it could seem we forgot privacy, but in this work we intended privacy as a reason for security rather than a kind of security. For example, a system that stores personal data needs to protect the data to prevent harm, embarrassment, inconvenience, or unfairness to any person about whom data is maintained and for that reason, the system may need to provide data confidentiality service.

The second important issue faced in this document is the quantification of the SPD measure (indicated even as SPD level) of a complex system by first quantifying the SPD level of its components.
The composability problem in SPD is another long standing problem. We start from the point that the pSHIELD embedded system (ES) is designed to be predictably composable such that the properties of the resulting composite system are easily determined, and the overall SPD level is calculated according to a given method. This method starts with an intuitive graphical representation of the system (medieval

castle), and then it is converted into an algebraic expression using several abstract operators. This composability modeling method has an immediate and complete semantic ontological descriptions represented in the appropriate WP documents.

This approach is proposed for the estimation of SPD functions indifferently if the SPD function is implemented in the node, network or middleware pSHIELD layer with the purpose of making easier the monitoring of the current SPD levels of the various layers and of the overall system, as well as the assessment of the various SPD levels.

The **second method** which is defined and described for SPD metrics measuring will not be implemented in pSHIELD due to its complexity. However, it is important to bring this method through a theoretic approach to pSHIELD project that might be taken for future work for pSHIELD project continuity. This method is based on the **security assurance** measurement.

An important definition is the term **security assurance** (SA), which is defined as the objective confidence that an entity meets its security requirements [12]. Given a complex system, direct measurement of SA properties is desirable, but not always possible. In that case, we have to decompose the system under consideration into measurable parts. Then we have to combine all the obtained SA values of decomposed entities into system-wide values by taking into account relations between these entities. For this backward process aggregation methods are needed. If we know the SA values of all SA entities and all SA relations between these entities, we can determine the overall SA value of the system.

Currently, there is no automatic framework to assess and manage the SA level of a large, complex system. Existing methodologies such as ISO 17799 and recently, ISO 27004 cannot be used for large deployed system. Some specific tools exist for detecting weakness such as open ports, vulnerabilities, for example, but they do not take into account combinations and compositions of different elements. Recent advances in security evaluation using attack graphs ([19], [20], [21], [22]) suggest an approach to determine the security level of a system in terms of detected vulnerabilities and relations between them, but focuses only on the static vision, the description of the system under study as implemented, and do not take into account system dynamic behavior.

Roundel’s papers [12][15][16] introduce a general method that takes into account also a dynamic view. In this document we use that approach, which handles the following two challenges:

- the appearance of emergent properties, and
- the relations between system entities are often non-linear,

That method aggregates security assurance properties of system entities into an appropriate system wide security assurance value. It is also discussed the application of patterns to simplify the process of aggregation in particular and of security assessment in general. Here we propose an aggregation process as a combination of two approaches: “castle” approach [18], and “SA with emerging attributes” approach, which bases on Roundel’s papers ([12], [15] and [16]).

3 Terms and definitions

Availability	Readiness for correct service. The correct service is defined as delivered system behaviour that is within the error tolerance boundary.
Authorised User	A user who possesses the rights and/or privileges necessary to perform an operation
Class and Family	The CC has organised the components into hierarchical structures: Classes consisting of Families consisting of components. This organisation into a hierarchy of class - family - component - element is provided to assist consumers, developers and evaluators in locating specific components
Common Criteria	The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification. It is currently in version 3.1. Common Criteria is a framework in which computer system users can specify their security functional and assurance requirements, vendors can then implement and/or make claims about the security attributes of their products, and testing laboratories can evaluate the products to determine if they actually meet the claims. In other words, Common Criteria provides assurance that the process of specification, implementation and evaluation of a computer security product has been conducted in a rigorous and standard manner.
Confidentiality	Property that data or information are not made available to unauthorized persons or processes.
Control system	A system that uses a regulator to control its behaviour
Correct service	Delivered system behaviour is within the error tolerance boundary.
Desired objectives	Desired objectives normally specified by the user.
Desired service	Delivered system behaviour is at or close to the desired trajectory.
Disturbance	Anything that tends to push system behaviour off the track is considered a disturbance. A disturbance can occur within a system or from the external environment.
Error	Deviation of system behaviour/output from the desired trajectory.
Error tolerance boundary	A range within which the error is considered acceptable by a system or user. This boundary is normally specified by the user.
Evaluator	An independent person involved in the judgment about the measure of the SPD functions.

Fault	Normally the hypothesized cause of an error is called fault [2]. It can be internal or external to a system. An error is defined as the part of the total state of a system that may lead to subsequent service failure. Observing that many errors do not reach a system's external state and cause a failure, Avizienis et al. [2] have defined active faults that lead to error and dormant faults that are not manifested externally.
Faults with Unauthorized Access	The class of Faults with unauthorized access (FUA) attempts to cover traditional security issues caused by malicious attempt faults. Malicious attempt fault has the objective of damaging a system. A fault is produced when this attempt is combined with other system faults.
Feedback	Use of the information observed from a system's behaviour to readjust/regulate the corrective action/control so that the system can achieve the desired objectives.
Feedback control system	A control system that deploys a feedback mechanism. This is also called a closed-loop control system.
Human-Made Faults	Human-made faults result from human actions. They include absence of actions when actions should be performed (i.e., omission faults). Performing wrong actions leads to commission faults. HMF are categorized into two basic classes: faults with unauthorized access (FUA), and other faults (NFUA).
Integrity	Absence of malicious external disturbance that makes a system output off its desired service
Life-Cycle support elements	It is the set of elements that support the aspect of establishing discipline and control in the system refinement processes during its development and maintenance. In the system life-cycle it is distinguished whether it is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the system is handed over to the user.
Maintainability	Ability to undergo modifications and repairs.
NonHuman-Made Faults	NHMF refers to faults caused by natural phenomena without human participation. These are physical faults caused by a system's internal natural processes (e.g., physical deterioration of cables or circuitry), or by external natural processes. They can also be caused by natural phenomena.
Not Faults with Unauthorized Access	There are human-made faults that do not belong to FUA. Most of such faults are introduced by error, such as configuration problems, incompetence issues, accidents, and so on.
Open-loop control system	A control system without a feedback mechanism.
Plant	A system that is represented as a function P(.) that takes its input as a functional argument
Privacy	The right of an entity (normally a person), acting in its own behalf, to

	determine the degree to which it will interact with its environment, including the degree to which the entity is willing to share information about itself with others.
Regulator	A control function whose role is to regulate the input of the plant, under the influence of the environment such as instructions of the user, observed error, input disturbance, etc. to achieve the desired objective.
Reliability	Continuity of correct service even under a disturbance.
Safety	Absence of catastrophic consequences on the users and the environment.
Security	A form of protection where a separation is created between the assets and the threat.
Service failure or failure	An event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary.
System	A composite constructed from functional components. The interaction of these components may exhibit new features/functions that none of the composite components possess individually.
System output	System behavior perceived by the user.
SPD function	A software, hardware or firmware component of the pSHIELD that must be relied upon for the correct enforcement of the pSHIELD security, privacy and dependability policy.
User session	A period of interaction between users and SPD functional components.

4 Integrating Dependability and Security

With rapidly developed network technologies and computing technologies, network-centric computing has become the core information platform in our private, social, and professional lives. This information platform is dependent on a computing and network infrastructure, which is increasingly homogeneous and open. The backbone of this infrastructure is the Internet, which is inherently insecure and unreliable. With an ever-accelerating trend of integrating mobile and wireless network infrastructure, things become worse. This is because wireless radio links tend to have much higher bit error rates, and mobility also increases the difficulty of service quality management and security control. The increased complexity of the platform and its easy access has made it more vulnerable to failures and attacks, which in turn has become a major concern for society. Traditionally there are two different communities separately working on the issues of dependability and security. One is the community of dependability that is more concerned with non malicious faults, to name one of just a few. The other is the security community that is more concerned with malicious attacks or faults.

Dependability was first introduced as a general concept covering the attributes of reliability, availability, safety, integrity, maintainability, and so on. With ever increasing malicious catastrophic Internet attacks, Internet providers have realized a need to incorporate security issues. Effort has been made to provide basic concepts and taxonomy to reflect this convergence.

Measures for security and dependability have also been discussed. Based on the work by Hu J, at al. [1], we propose a framework that can generically integrate dependability and security (privacy will be considered as part of security as mentioned in the Introduction). This approach does not intend to cover every detail of dependability and security. It places major relevant concepts and attributes in a unified feedback control system framework and illustrates the interaction via well-established control system domain knowledge.

4.1 Dependability and Security attributes definition

To address Security, Privacy and Dependability in the context of Embedded Systems (ESs) it is essential to define the assets that these kinds of systems aim to protect.

The pSHIELD project assets can be categorised in two principal groups, logical and physical asset. Inside these categories it is possible to define information, services and software as logical assets and human beings, hardware or particular physical objects as physical assets.

These assets are characterized by their Dependability and Security attributes.
The original definition of dependability refers to the ability to deliver a service that can be justifiably trusted. The alternative definition is the ability to avoid service failures that are more frequent and severe than is acceptable. The concept of trust can be defined as accepted dependence, and dependability encompasses the following attributes:

- Availability: Readiness for correct service. The correct service is defined as what is delivered when the service implements a system function.
- Reliability: Continuity of correct service.
- Safety: Absence of catastrophic consequences on the users and environment.
- Integrity: Absence of improper system alterations.
- Maintainability: Ability to undergo modifications and repairs.

Security has not been introduced as a single attribute of dependability. This is in agreement with the usual definitions of security, which is viewed as a composite notion of the three following attributes:

- Confidentiality: the prevention of the unauthorized disclosure of information;
- Integrity: the prevention of the unauthorized amendment or deletion of information;
- Availability: the prevention of the unauthorized withholding of information.

Avizienis et al. [2] merged the attributes of dependability and security together, as shown in **Error! Reference source not found.**. Similarly, the above attributes can be reframed as follows:

- Availability: Readiness for correct service. The correct service is defined as delivered system behaviour that is within the error tolerance boundary.
- Reliability: Continuity of correct service. This is the same as the conventional definition.
- Safety: Absence of catastrophic consequences on the users and the environment. This is the same as the conventional definition.
- Integrity: Absence of malicious external disturbance that makes a system output off its desired service.
- Maintainability: Ability to undergo modifications and repairs. This is the same as the conventional definition.
- Confidentiality: Property that data or information are not made available to unauthorized persons or processes. In the proposed framework, it refers to the property that unauthorized persons or processes will not get system output or be blocked by the filter.

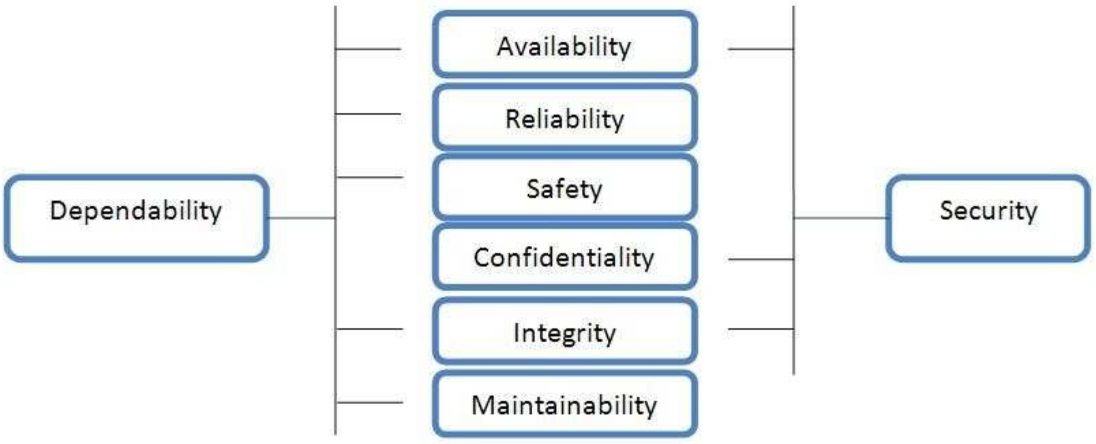
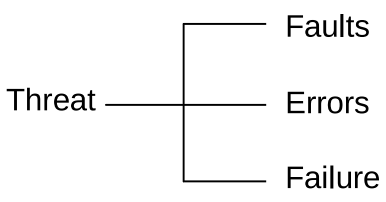


Figure 1 Attributes of Security and Dependability

The importance of such assets SPD attributes is usually expressed in terms of the consequential damage resulting from the manifestation of threats.
Threats are expressed in the following taxonomy:



where a fault is defined as a cause of an error and a failure is linked to the error that is outside of the error tolerance boundary and is caused by a fault.

4.2 Measurable and immeasurable attributes for the concepts

In the five concepts described in Annex B we can meet a lot of attributes. Some of them are qualitative attributes and some others are quantitative ones. A paper [17] gives a list of these attributes and shows which of them are measurable and which are not. If we take a closer look at this list, we can see that the common attributes among the studied concepts (Annex B) are the measurable ones. These common measurable attributes are availability, fault-tolerance, maintainability, and reliability. It stands that the more measurable attributes are contained in a concept, the easier it is to evaluate the performance of that concept and balance it with all other attributes. To determine the achieved degree of the measurable attributes quantifiable parameters are associated with this attributes.

Concept Attributes		**Measurability**		**Security attributes**	**Dependability attributes**	**Survivability attributes**
No.	**Attributes**	**Measurable**	**Immeasurable**	**Measurable**	**Measurable**	**Measurable**
1	Accessibility		o			
2	Accountability		o			
3	Authenticity		o			
4	Availability	x		x	x	x
5	Confidentiality		o			
6	Fault-Tolerance	x				**x**
7	Integrity	x		x	x	x
8	Maintainability	x			**x**	
9	Non-Repudiation		o			
10	Performability	x				**x**
11	Reliability	x			x	x
12	Safety		o			
13	Security		o			
14	Testability		o			
15	Unreliability	x			x	x
16	Unavailability	x		x	x	x

Table 1 Measurable "x" and Immeasurable "o" attributes for the concepts[1]

From
Table 1 we can see that Survivability and Dependability are covering security measurable attributes. It is also true for immeasurable attributes. So, Dependability and Survivability concepts are top level overall goal and all other concepts and their corresponding qualitative and quantitative attributes can be considered as other design requirements that assist in the overall design. Figure 2 shows the common measurable attributes of security, dependability and survivability.

[1] The Annex B gives definition of the attributes.

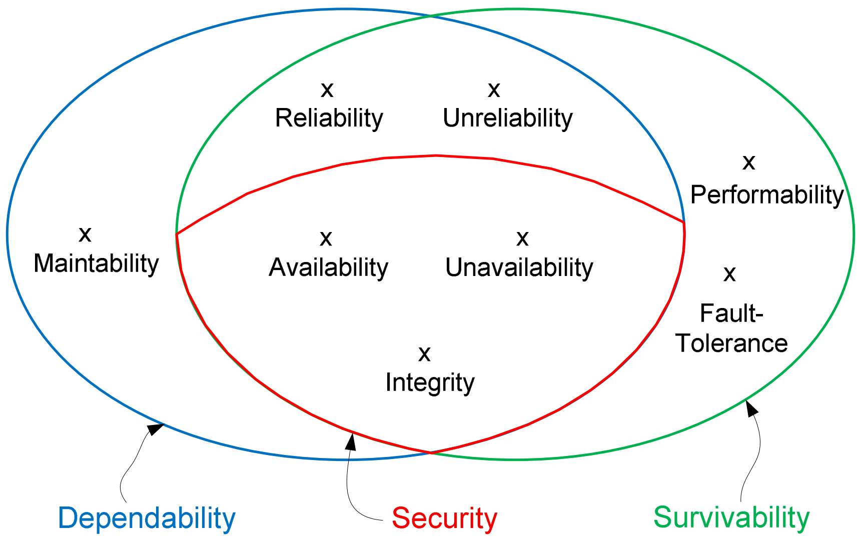


Figure 2 Security, Dependability and Survivability measurable attributes

Before we defined the SPD-tuples of measurable attributes for SPD metrics we need to perform a Concept Analysis to define a set of quantifiable parameters. What are definition and goals of Security, Dependability and Survivability for services in the pSHIELD SPD network?

- Dependability (is an umbrella concept) is ability to deliver required services during its life cycle that can justifiably be trusted.
- Security is ability to guard and protect unwanted happenings or actions and preserve confidentiality, integrity and availability (CIA, i.e., fundamental attributes).
- Survivability is ability to fulfill its mission in timely manner in the presence of attacks, failures and accidents.

An important definition is the term **security assurance** (SA), which is defined as the objective confidence that an entity meets its security requirements [12], which are covered by dependability and survivability as top level concepts. The idea of decomposing the general concept of dependability and survivability is very helpful for the pSHIELD project. The decomposition of a higher level quality concept into objective lower level factors helps reveal quantitative performance characteristics.

The pSHIELD aims represent an integrated framework in order to find required relations between quality of services and the parameters specifying system performance. There are two problems for which we need solutions. First, quantitative attributes need to be developed for specifying quality of service expected by the user. Second, a set of linear independent system performance parameters need to be identify. The five taxonomy concepts described in Annex B are overlapping to various degrees. Therefore, it is important to distinguish the meaning of these concepts. The first step toward linearly independent system performance parameters is to identify the proper union of performance parameters embodied in them.

Although the component-level and infrastructure perspectives take different approaches they both address issues of system performance through analysis and design. The infrastructure perspective gives rise to two set of requirements, one of the system and another for the services. So, the identification of System Attributes (as well as sub-system attributes) and Service Attributes is important for defining SPD-tuples (of measurable metrics) at system or service level as well as the interfaces between Service and

System Attributes, and both integrated with User and another system or sub-system. These service and system attributes with user form a high-layer abstraction for the application.

The pSHILED system perspective involves the information system infrastructure and services, and the component perspective involves computer and communication infrastructures and their services. To integrate the service perspective with the component-level and infrastructure perspectives, it is necessary to analyze user requirements specified at the application level [3], and map them into those of the physical layer.

In general we have:

I. **Network's Requirements at Lower Layers** (hierarchically structured of the OSI reference Model), e.g., design requirements (goals, functions, specifications, tradeoffs, etc), and
II. **User Requirements at Upper Layers**, with an application scenario run on the physical network where the pSHIELD performance requirements address security, dependability, privacy, reliability, fault tolerance, reliability, survivability, etc.

How to define relevant SPD-tuple or SPD-metrics as aggregation of individual or a set of attributes at components that involved computer and communication infrastructures? The quantitative models required a large number of variables to adequately describe their performance, making the use of the models unwieldy. Another source of difficulties is the evolutionary nature of the information infrastructure. For example, computer and communication technologies involve "legacy" systems that have different and usually inferior performance comparing to the new pSHIELD systems. This problem will appear in the integration of technologies with different performance. Changing the infrastructure and application scenario will introduce new design system performance targets and consequently changes in the conceptual models, selection of attributes the aggregation of them and finely to different SPD-metrics. For example, the use of a concept at component-level may no longer be sufficient to address requirements imposed on critical information infrastructures.

The infrastructure perspective leads to a holistic view of a system as exemplified by the development of the dependability concept. This holistic view has been no more successful in generating quantitative models than the component-level perspective. The dimensions of the state space of the potential pSHIELD infrastructures (e.g., for different application scenarios) are so large as to make development of quantitative models impractical. Therefore, for the characterisation of pSHIELD-metrics, what to measure, why to measure, who to measure, and how to measure become extremely important phases in the conceptual development phase.

From the user perspective one of the biggest concern today for the computer and communication infrastructures at component-level, and the information system infrastructure at system level is security. Privacy is also very important for the users, but it is always associated with the application and services and may change from low to very high privacy requirements. Dependability concept covers all measurable attributes of security. New immeasurable attributes (like authentication that is extremely important also for privacy) may be decomposed on measurable new attributes, and combing them with other to develop emerging attributes.

4.3 Integration of dependability and security concepts

Integrating dependability and security is very challenging and ongoing research effort is needed. Avizienis et al. [2] and Jonsson [4] proposed a system view to integrate dependability and security. They tried to use the system function and behaviour to form a framework. A schema of the taxonomy of dependable and secure computing is proposed. Although the taxonomy and framework is discussed in the context of system interacting with its environment, there seems to be a lack of a cohesive and generic integration [2]. Jonsson [4] made an attempt to provide a more generic integration framework using an input output system model. In this scheme, faults are introduced as inputs to the system and delivery-of-service and denial-of-service are considered as system outputs. However, it is still unclear about the interactions among other system components. As a matter of fact both system models proposed by Avizienis et al. [2] and Jonsson [4] are of open loop systems, which are unable to provide a comprehensive description of the interaction relationship. For example, they cannot describe the relationship between fault detection and fault elimination using the proposed system. Also security attributes have been rarely addressed. In the next section, it is proposed a new framework to address these issues.

4.3.1 Proposed framework Feedback control system model

In this section it is proposed a feedback control system, shown in

Figure 3, as a framework that can generically integrate dependability and security. Key notations and concepts for the illustration of this framework are provided in the following sections

4.3.1.1 Key notations of the feedback control system model

The following are conventional notations of feedback control systems and they are tailored whenever needed for our framework.

System: a composite entity constructed from functional components. The interaction of these components may exhibit new features/functions that none of the composite components possess individually.
Plant: a system that is represented as a function P(.) that takes its input as a functional argument, i.e. P(s(t)), and transfers it into an output accordingly. Note that the predefined function P(.) can be modified via the plant adjustment channel.
Regulator: a control function whose role is to regulate the input of the plant, under the influence of the environment such as instructions of the user, observed error, input disturbance, etc. to achieve the desired objective.
Control system: a system that uses a regulator to control its behaviour.

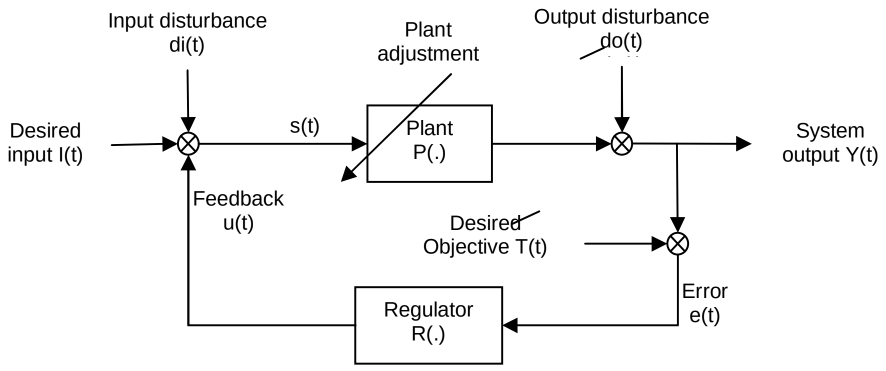


Figure 3

Feedback: use of the information observed from system's behaviour to readjust/regulate the corrective action/control so that the system can achieve the desired objectives.
Feedback control system: a control system that deploys a feedback mechanism. This is also called a closed loop control system.
Open loop control system: a control system without a feedback mechanism.
Desired objectives: desired objectives normally specified by the user.
Disturbance: anything that tends to push the system behaviour off the track is considered as a disturbance. A disturbance can occur within the system or from the external environment.

System output: system behaviour perceived by the user.

4.3.1.2 Definitions of basic concepts of dependability and security

In order to understand the relationship of various components and functions of the feedback control system shown in

Figure 3, following relationship descriptions are provided according to conventional feedback control system theory. We use parameter ε as a threshold of error. Then we will have the following equations:

$s(t) = I(t) + di(t) + u(t)$ (1)

$u(t) = R(e(t))$ (2)

$e(t) = Y(t) - T(t)$ (3)

$Y(t) = do(t) + P(s(t))$ (4)

We use following variable expressions to represent the desired or normal values: $s_0(t)$, $u_0(t)$, $e_0(t)$, $Y_0(t)$ and $P_0(.)$. Under the desired condition or normal condition, all disturbances do not appear and we will have the following relationships:

$s_0(t) = I(t)$ (5)

$u_0(t) = R(0)=0$ (6)

$E_0(t) = Y_0(t) - T(t)=0$ (7)

$Y_0(t) = P(s_0(t)) = P(I(t))$ (8)

The above description is for a single input single output (SISO) control system. In general Systems have multiple functional components where each component itself forms a system and interconnects with each other. Distributed control theory is also able to describe such multiple input and multiple output (MIMO) relationships. However, it will make little difference for our conceptual discussions in this paper whether it is a SISO system or a MIMO system. This is because multiple inputs, multiple outputs and disturbances can be represented by individual vectors, and Eqs. (1) – (8) still holdwhere symbols then become vectors. The error vector $\varepsilon_0(t)$ can be quantitatively measured by its norm $||\varepsilon_0(t)||_2$. By default, we assume a SISO system environment unless stated otherwise.
Now we can provide following definitions using this framework.

Error: deviation of the system behaviour/output from the desired trajectory represented by e(t).

Error tolerance boundary: a range within which the error is considered acceptable by the system or user. This boundary is normally specified by the user. To be more precise, this refers to $e(t) < \varepsilon$ where ε is a threshold parameter value, which has been specified by the user, and is used to denote the error tolerance boundary. For example, given that a normal service response time is 1 min, then an error tolerance boundary of 0.5 will mean a maximum tolerance response time being 1.5 min.

Desired service: delivered system behaviour is at or close to the desired trajectory, i.e., we have the feedback control system error $e(t) \rightarrow 0$.

Correct service: delivered system behaviour is within the error tolerance boundary, i.e. $e(t) < \varepsilon$.

Service failure or failure: an event that occurs when system output deviates from the desired service and is beyond the error tolerance boundary, i.e., $e(t) > \varepsilon$.

Fault: normally the hypothesized cause of an error is called fault (Avizienis et al., 2004). It can be internal or external to the system. Observing that many errors do not reach the system's external state and cause a failure, Avizienis et al. (2004) have defined active faults that lead to error and dormant faults

that are not manifested externally. Under our proposed framework, dormant faults are do(t), di(t), which do not propagate to e in Eqs. (1) – (4).
The conventional attributes of dependability and security description (defined in par.4.1) can be integrated using the proposed framework.

a) Availability: readiness for correct service. The correct service is defined as delivered system behaviour that is within the error tolerance boundary. This can be interpreted as: $e(t) < \varepsilon$ if a required service is provided to the input of the feedback control system shown in
b) Figure 3.
c) Reliability: even under the disturbances as shown in
d) Figure 3, we will have $e(t) < \varepsilon$, i.e., correct service is still maintained.
e) Safety: absence of catastrophic consequences on the users and the environment. This is the same as the conventional definition.
f) Integrity: absence of malicious external disturbance, which makes the system output off its desired service. This can be interpreted as the absence of malicious plant adjustment, which modifies the function of the plant leading to $e(t) > \varepsilon$.
g) Maintainability: ability to undergo modifications and repairs. This is the same as the conventional definition.
h) Confidentiality: property that data or information is not made available to unauthorized persons or processes. In the proposed framework, it refers to the property that unauthorized persons or processes will not be able to observe the values/contents of the sensitive variables such as $s_0(t)$or $Y_0(t)$.

4.4 Fault classification

We have provided a set of elementary fault classes with minimum overlapping. An intuitive choice is to start two classes namely, human-made faults (HMF) and nonhuman-made faults (NHMF).

4.4.1 Human-Made Faults (HMF)

Human-made faults result from human actions. They include absence of actions when actions should be performed (i.e., omission faults). Performing wrong actions leads to commission faults.
HMFs are categorized into two basic classes: faults with unauthorized access (FUA), and other faults (NFUA).

4.4.1.1 Faults with Unauthorized Access (FUA)

The class of Faults with unauthorized access (FUA) attempts to cover traditional security issues caused by malicious attempt faults. We have investigated FUA from the perspective of availability, reliability, integrity, confidentiality and safety.
Malicious attempt fault has the objective of damaging a system. A fault is produced when this attempt is combined with other system faults.
FUA and confidentiality. Confidentiality refers to the property that information or data are not available to unauthorized persons or processes, or that unauthorized access to a system's output will be blocked by the system's filter. Confidentiality faults are mainly caused by access control problems originating in cryptographic faults, security policy faults, hardware faults, and software faults. Cryptographic faults can originate from encryption algorithm faults, decryption algorithm faults, and key distribution methods. Security policy faults are normally management problems and can appear in different forms (e.g., as contradicting security policy statements).
FUA and integrity. Integrity refers to the absence of improper alteration of information. An integrity problem can arise if, for instance, internal data are tampered with and the produced output relies on the correctness of the data.
FUA and availability & reliability. In general Availability refers to a system's readiness to provide correct service and reliability refers to continuity of correct service, but according to interpretation proposed in the paragraph 4.3.1.2 (a) and (b) these attributes have been considered as one because both guarantee the

correct service with an error e(t) < ε. A typical cause of such faults is some sort of denial of service (DoS) attack that can, for example, use some type of flooding (SYN, ICMP, UDP) to prevent a system from producing correct output. The perpetrator in this case has gained access to a system, albeit a very limited one, and this access is sufficient to introduce a fault.
FUA and safety. Safety refers to absence of catastrophic consequence on System users end environment. A safety problem can arise if, for instance, an unauthorized system access can cause the possibility of human lives being endangered.

4.4.1.2 Not Faults with Unauthorized Access (NFUA)

There are human-made faults that do not belong to FUA. Most of such faults are introduced by error, such as configuration problems, incompetence issues, accidents, and so on.

4.4.2 NonHuman-Made Faults (NHMF)

NHMF refers to faults caused by natural phenomena without human participation. These are physical faults caused by a system's internal natural processes (e.g., physical deterioration of cables or circuitry), or by external natural processes. They can also be caused by natural phenomena. For example, in communication systems, a radio transmission message can be destroyed by an outer space radiation burst, which results in system faults, but has nothing to do with system hardware or software faults.

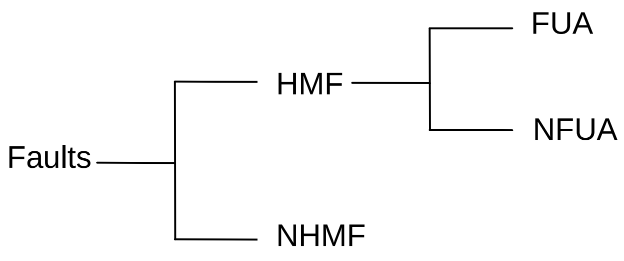


Figure 4

4.4.3 Tree representation of faults

From above discussions, we propose the following elementary fault classes.

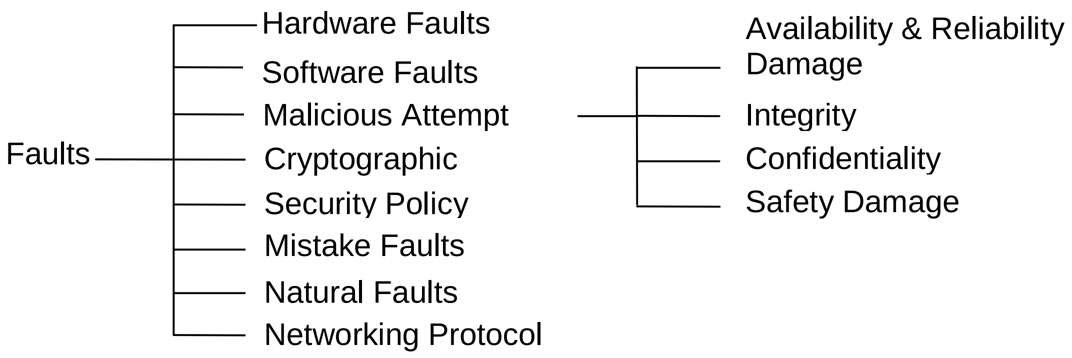


Figure 5

From these elementary fault classes, we have constructed a tree representation of faults, as shown in the next figure.

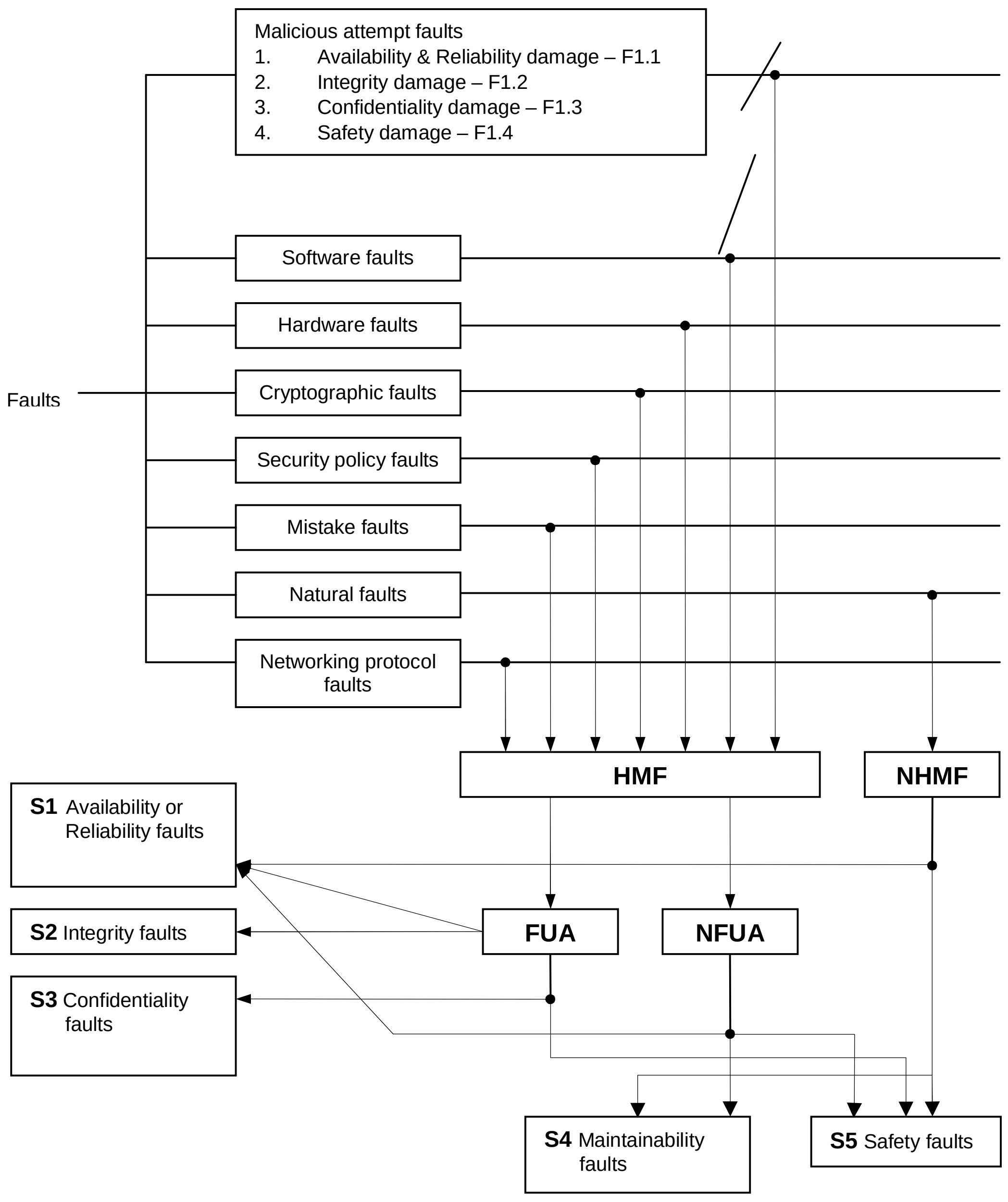


Figure 6

Figure 7 shows different types of availability faults. The Boolean operation block performs either “Or” or “And” operations or both on the inputs. We provide several examples to explain the above structure. We consider the case when the Boolean operation box is performing “Or” operations. F1.1 (a malicious attempt fault with intent to availability & reliability damage) combined with software faults will cause an availability or reliability fault. F1.1 in combination with hardware faults can also cause an availability fault. F7 (natural faults) can cause an availability fault. F1.1 and F8 (networking protocol) can cause a denial of service fault.

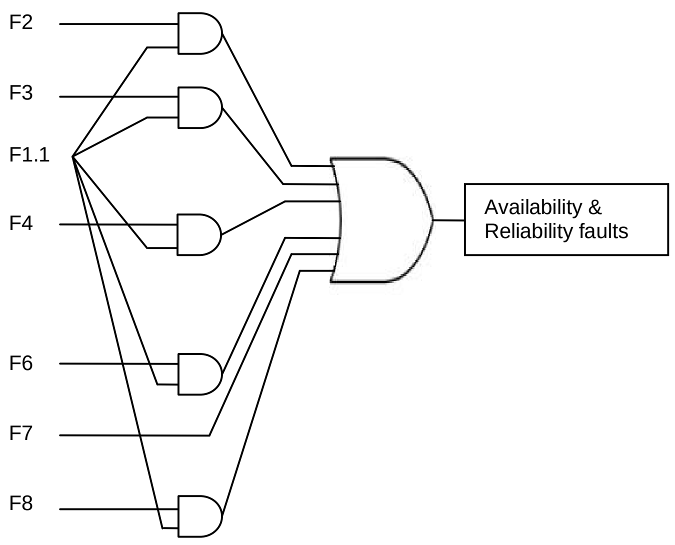


Figure 7

The interpretation of S2 (next figure) is similar to that of S1. The combination of F1.2 and F2 can alter the function of the software and generate an integrity fault. Combining F1.2 and F4 can generate a person-in-the-middle attack and so on.

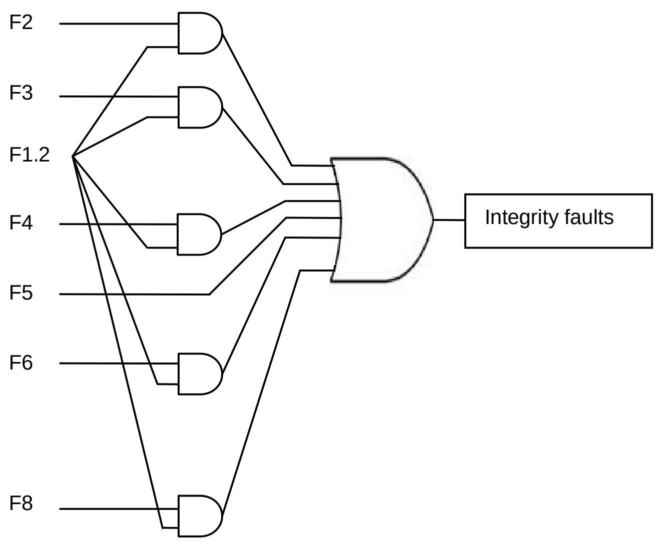


Figure 8

Figure 9 shows types of confidentiality faults.
The interpretation of S3 is very similar to those of S1 and S2. Combination of F1.3 and F2 can generate a spying type of virus that steals users' logins and passwords. It is easy to deduce other combinations.

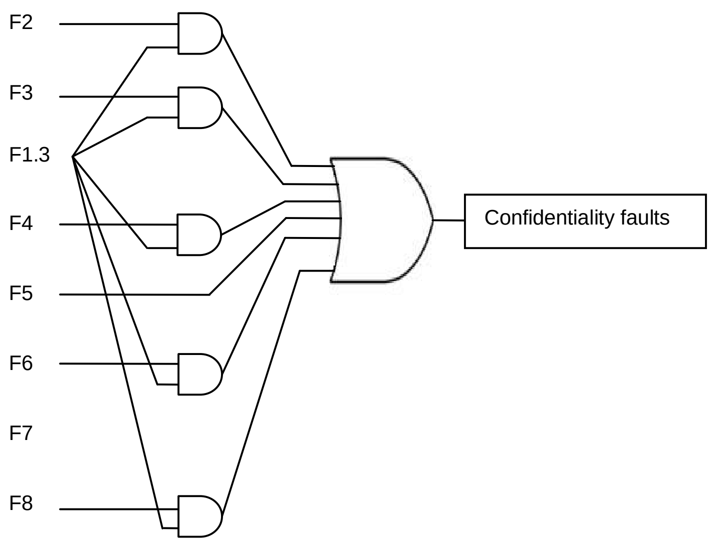


Figure 9

Figure 10 shows types of maintainability faults.
The interpretation of S4 is very easy because faults combinations are non present. So a maintainability fault can be generated by a single kind of fault belonging to F2, F3, F5, F6 or F7, or a combination of them.

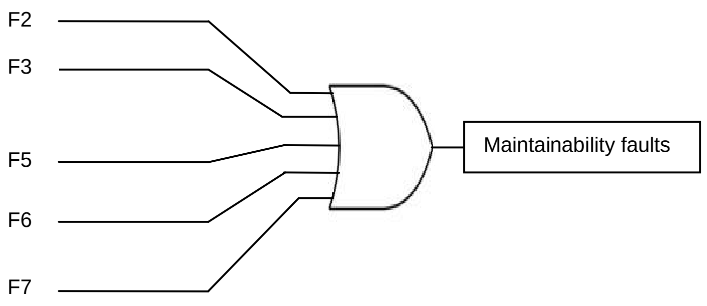


Figure 10

Figure 11 shows types of safety faults.
The interpretation of S5 is very similar to those of S1, S2 and S3. Combination of F1.4 and F2 can generate an unpredictable system state causing a catastrophic consequence on users and the environment. A similar deduction can be done for the other combinations.

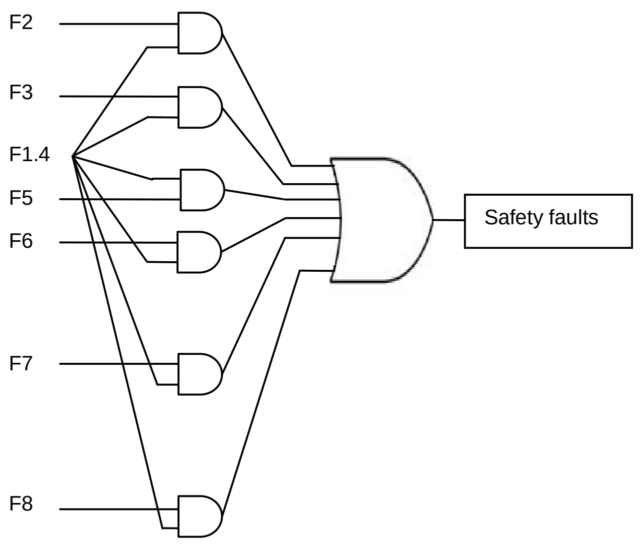


Figure 11

5 SPD metric definition for basic components

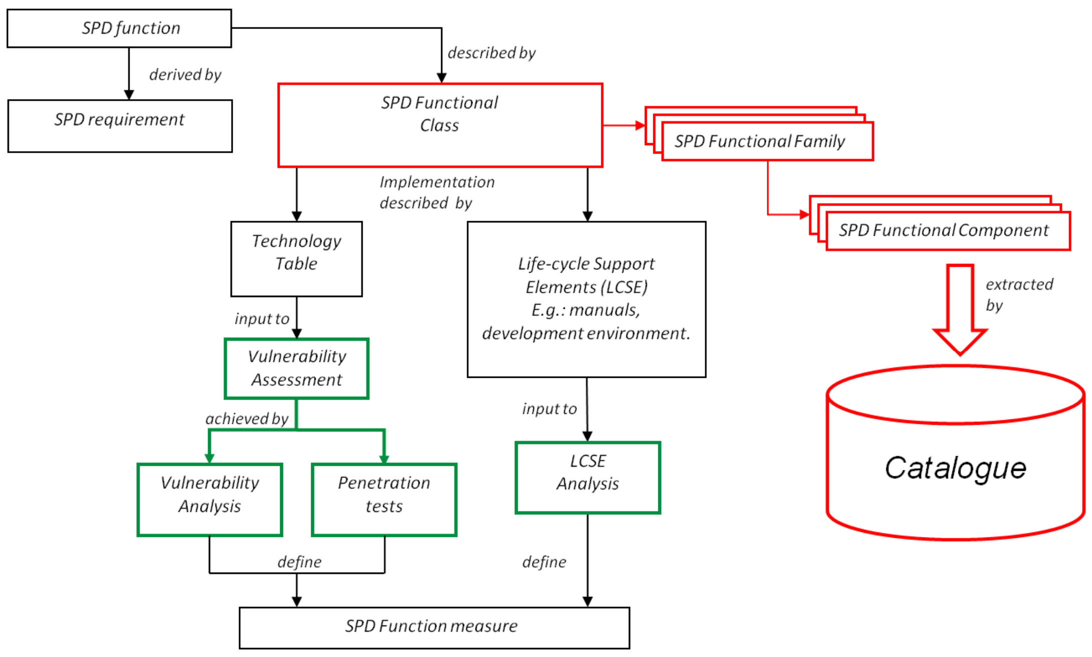


Figure 12

As we can see in the previous figure each SPD function description can be extracted by a catalogue (based on Common Criteria part 2 for FUA mitigation and Common Criteria part 3 for NFUA and NHMF mitigation) where SPD functions are grouped in Classes, Families, and Components (Figure 12 – red evidenced part),

The two parallel processes (left FUA, right NFUA and NHMF) identified in Figure 12 (green evidenced part), which lead to SPD function measure, are both based on Common Criteria.
In particular the first one is based on a vulnerability assessment and the second one on evaluation of defined Life-Cycle Support Elements (LCSE, these elements are generally defined in documents; e.g.: guidance, manuals, development environment, etc.).

The vulnerability assessment of SPD functions is conducted with the purpose to estimate their robustness, determining the existence and exploitability of flaws or weaknesses in its operational environment. This determination is based upon a vulnerability analysis and supported by testing.

The vulnerability analysis consists in the identification of potential vulnerabilities identified as those:

- Encountered in the public domain or in other way
- Found through an analysis of SPD functions taking into account
  - Bypassing
  - Tampering
  - Direct attacks
  - Monitoring

Penetration tests are performed to determine whether identified potential vulnerabilities are exploitable in the operational environment of the pSHIELD. They are conducted assuming an attack potential.

The following factors should be considered during the calculation of the minimum attack potential required to exploit a vulnerability:

a) Time taken to identify and exploit (Elapsed Time);

b) Specialist technical expertise required (Specialist Expertise);

c) Knowledge of the SPD functionality design and operation (Knowledge of the functionality);

d) Window of opportunity;

e) IT hardware/software or other equipment required for exploitation (Equipment).

Table 2 identifies the factors listed above and associates numeric values with the total value of each factor

Factor	**Value**
Elapsed Time	
<= one day	0
<= one week	1
<= one month	4
<= two months	7
<= three months	10
<= four months	13
<= five months	15
<= six months	17
> six months	19
Expertise	
Layman	0
Proficient	3*(1)
Expert	6
Multiple experts	8
Knowledge of functionality	
Public	0
Restricted	3
Sensitive	7
Critical	11
Window of Opportunity	
Unnecessary / unlimited access	0
Easy	1
Moderate	4
Difficult	10
Unfeasible	25**(2)
Equipment	
Standard	0
Specialised	4(3)
Bespoke	7
Multiple bespoke	9

Table 2 Factor/Value for calculation of the minimum attack potential

(1) When several proficient persons are required to complete the attack path, the resulting level of expertise still remains “proficient” (which leads to a 3 rating).

(2) Indicates that the attack path is considered not exploitable due to other measures in the intended operational environment of the pSHIELD.

(3) If clearly different test benches consisting of specialised equipment are required for distinct steps of an attack, this should be rated as bespoke.

The value of the minimum attack potential required to exploit a vulnerability is the measure of SPD function tested. In the next section this measure is indicated with the letter d. This value indicates how much is valid the implementation of that particular SPD function to increment pSHIELD faults resistance.

After calculating measures for all SPD functions, it is possible to define a single measure for the whole pSHIELD system considering the composition approach described in the next section.

LCSE Analysis is conducted to prevent the misuse of the pSHIELD SPD functions evaluating life-cycle documents.

Misuse may arise from:

- incomplete guidance documentation;
- unreasonable documentation;
- unintended misconfiguration of the pSHIELD;
- forced exception behaviour of the pSHIELD.

The measure of life-cycle documents is estimated through Common Criteria standard giving a numeric value for each passed activity and then summing it. (The standard defines each activity in a very formal way). In the next this value is indicated as d_{LC}.

This process has to be repeated for each SPD function implemented by pSHIELD components and, if there are different implementation of the same SPD function, for each of them.

We assume that all these activities are carried out by independent and unbiased IT security specialists (evaluators) in order to obtain an evaluated value of the SPD function measure (SPD level) which has the qualities of repeatability, reproducibility, impartiality, and objectivity.

In the context of IT security evaluation, as in the fields of science and testing, repeatability, reproducibility, impartiality, and objectivity are considered to be important principles.

An evaluated SPD level is repeatable, if the repeated evaluation yields the same result as the first evaluation.

An evaluated SPD level is reproducible, if the evaluation of the same implemented SPD function in the same operational environment performed by a different evaluator yields the same results as the evaluation performed by the first evaluator.

An evaluated SPD level is performed impartially, if the evaluation is not biased towards any particular result.

An evaluated SPD level is performed objectively if the result is based on actual facts uncoloured by the evaluators' feelings or opinions.

These four principles must be enforced and ensured by the pSHIELD consortium.

In the next sections we describe in more detail the guidance for completing an independent vulnerability assessment.

5.1 Technologic table

The implementation of the SPD functions can be described by technologic tables.
In the following it is shown the structure of a Technologic table.

SPD Functional Component (1)		
Technology (2)		
Parameters (3)	Name	Value
General description		
Life Cycle element reference (4)		
Note:		

Table 3 Technologic table

(1) SPD Functional Component from catalog
(2) Name of a specific technology used to implement the SPD function (it could be hardware, software, firmware, algorithm…)
(3) Name and value of all parameters that can describe the SPD function
(4) Reference to Life Cycle elements which have the scope to avoid the SPD functionality misuse

5.2 Vulnerability Assessment

The method to obtain a metric for the SPD functions enforced to reduce the risk risen by FUA, hereafter indicated as SPD FUA functions, has been built through a vulnerability assessment activity.

The purpose of the vulnerability assessment activity is to determine the existence and exploitability of flaws or weaknesses in pSHIELD in its operational environment. This determination is based upon a vulnerability analysis and supported by testing.

The evaluator has to start the vulnerability analysis simply performing a search of publicity available information to identify any known weaknesses in SPD FUA functions implementation, and go on performing a structured analysis of architectural design.

We have considered three main factors in performing the vulnerability analysis, namely:

a) the identification of potential vulnerabilities;

b) assessment to determine whether the identified potential vulnerabilities could allow an attacker with the relevant attack potential to violate the SPD FUA functions;

c) penetration testing to determine whether the identified potential vulnerabilities are exploitable in the operational environment of the pSHIELD.

The identification of vulnerabilities has been further decomposed into the evidence to be searched and how hard to search that evidence to identify potential vulnerabilities. In a similar manner, the penetration testing has been further decomposed into analysis of the potential vulnerability to identify attack methods and the demonstration of the attack methods.

These main factors are iterative in nature, i.e. penetration testing of potential vulnerabilities may lead to the identification of further potential vulnerabilities. Hence, these are performed as a single vulnerability analysis activity.

5.3 Vulnerability Analysis

The vulnerability analysis is to determine that the pSHIELD system is resistant to penetration attacks performed by an attacker possessing an attack potential. We have first assessed the exploitability of all identified potential vulnerabilities. This has been accomplished by conducting penetration testing. We assumed the role of an attacker with an attack potential when attempting to penetrate the pSHIELD.
In order to perform the vulnerability analysis, evaluator familiarity with the generic vulnerability searching guidance, described later on, is required.

5.3.1 Generic vulnerability searching guidance

The following four categories provide discussion of generic vulnerabilities:

1. **Bypassing**
2. **Tampering**
3. **Direct Attacks**
4. **Monitoring**

1. Bypassing

Bypassing includes any means by which an attacker could avoid security enforcement, by:

a) **exploiting the capabilities of interfaces to the pSHIELD, or of utilities which can interact with it;**
b) **inheriting privileges or other capabilities that should otherwise be denied;**
c) **(where confidentiality is a concern) reading sensitive data stored or copied to inadequately protected areas.**

Each of the following (where relevant) are considered in the suggested vulnerability analysis:

a) **Attacks based on exploiting the capabilities of interfaces or utilities generally take advantage of the absence of the required security enforcement on those interfaces. For example, gaining access to function that is implemented at a lower level than that at which access control is enforced. Relevant items include:**

1. changing the predefined sequence of invocation of SPD FUA function interfaces: this case should be considered where there is an expected order in which interfaces to the pSHIELD (e.g. user commands) are called to invoke a SPD FUA function interface (e.g. opening a file for access and then reading data from it). If a SPD FUA function interface

is invoked through one of the pSHIELD interfaces (e.g. an access control check), it must be considered whether it is possible to bypass the control by performing the call at a later point in the sequence or by missing it out altogether.

2. invoking an additional SPD FUA function interface (in the predefined sequence): it is a similar form of attack to the one described above, but involves the calling of some other pSHIELD interface at some point in the sequence. It can also involve attacks based on interception of sensitive data passed over a network by use of network traffic analysers (the additional component here being the network traffic analyser).

3. using a component in an unexpected context or for an unexpected purpose: it includes using an unrelated pSHIELD interface to bypass the SPD FUA function by using it to achieve a purpose that it was not designed or intended to achieve. Covert channels are an example of this type of attack. The use of undocumented interfaces, which may be insecure, also falls into this category. Such interfaces may include undocumented support and help facilities.

4. using implementation detail introduced in less abstract representations (detailed architectural design): it may allow an attacker to take advantage of additional functions, resources or attributes that are introduced to the pSHIELD as a consequence of the refinement process. Additional function may include test harness code contained in software modules and back-doors introduced during the implementation process.

5. using the delay between time of access check and time of use: it includes scenarios where an access control check is made and access granted and an attacker is subsequently able to create conditions in which, had they applied at the time the access check was made, would have caused the check to fail. An example would be a user creating a background process to read and send highly sensitive data to the user's terminal, and then logging out and logging back in again at a lower sensitivity level. If the background process is not terminated when the user logs off, the MAC (Mandatory access control) checks would have been effectively bypassed.

b) **Attacks based on inheriting privileges are generally based on illicitly acquiring the privileges or capabilities of some privileged component, usually by exiting from it in an uncontrolled or unexpected manner. Relevant items include:**

1. executing data not intended to be executable, or making it executable: it includes attacks involving viruses (e.g. putting executable code or commands in a file which are automatically executed when the file is edited or accessed, thus inheriting any privileges the owner of the file has).
2. generating unexpected input for a component: it component can have unexpected effects which an attacker could take advantage of. For example, if the SPD FUA function could be bypassed if a user gains access to the underlying operating system, it may be possible to gain such access following the login sequence by exploring the effect of hitting various control or escape sequences whilst a password is being authenticated.
3. invalidating assumptions and properties on which lower-level components rely: it includes attacks based on breaking out of the constraints of an application taking part of a pSHIELD system to gain access to an underlying operating system in order to bypass the SPD FUA function implemented by such application. In this case the assumption being invalidated is that it is not possible for a user of that application to gain such access.

c) **Attacks based on reading sensitive data stored in inadequately protected areas (applicable where confidentiality is a concern) include the following issues which should be considered as possible means of gaining access to sensitive data:**

1. disk scavenging;

2. access to unprotected memory;

3. exploiting access to shared writable files or other shared;

4. resources (e.g. swap files);

5. Activating error recovery to determine what access users can obtain. For example, after a crash an automatic file recovery system may employ a lost and found directory for headerless files, which are on disk without labels.

2. Tampering

Tampering includes any attack based on an attacker attempting to influence the behaviour of the SPD FUA function (i.e. corruption or de-activation), for example by:

a) accessing data on whose confidentiality or integrity the SPD FUA function relies;
b) forcing the pSHIELD to cope with unusual or unexpected circumstances;
c) disabling or delaying security enforcement;
d) physical modification the pSHIELD.

Each of the following has been considered (where relevant) in our vulnerability analysis.

a) Attacks based on accessing data, whose confidentiality or integrity are protected, include:

1. reading, writing or modifying internal data directly or indirectly: it includes the following types of attack which should be considered:

   - reading “secrets” stored internally, such as user passwords;

   - spoofing internal data that security enforcing mechanisms rely upon;

   - modifying environment variables (e.g. logical names), or data in configuration files or temporary files: for example it may be possible to deceive a trusted process into modifying a protected file that it wouldn't normally access. Furthermore it is necessary to confirm the absence of the following “dangerous features”:

     - source code resident on the pSHIELD along with a compiler (for instance, it may be possible to modify the login source code);
     - an interactive debugger and patch facility (for instance, it may be possible to modify the executable image);
     - the possibility of making changes at device controller level, where file protection does not exist;
     - diagnostic code which exists in the source code and that may be optionally included;
     - developer's tools left in the pSHIELD.

2. using a component in an unexpected context or for an unexpected purpose: it includes (for example), an application taking part of pSHIELD built upon an operating system, users exploiting knowledge of a word processor package or other editor to modify their own command file(e.g. to acquire greater privileges).

3. using interfaces between components that are not visible at a higher level of abstraction (e.g. architectural design): it includes attacks exploiting shared access to resources, where modification of a resource by one component can influence the behaviour of

another (trusted) component, e.g. at source code level, through the use of global data or indirect mechanisms such as shared memory or semaphores.

b) Forcing the pSHIELD to cope with unusual or unexpected circumstances should always be considered. Relevant items include:

1. generating unexpected input for a component: it includes investigating the behaviour of the pSHIELD when:

   - ➢ command input buffers overflow (possibly “crashing the stack” or overwriting other storage, which an attacker may be able to take advantage of, or forcing a crash dump that may contain sensitive information such as clear-text passwords);
   - ➢ invalid commands or parameters are entered (including supplying a read-only parameter to an interface which expects to return data via that parameter and supplying improperly formatted input that should fail parsing such as SQL-injection, format strings);
   - ➢ an end-of-file marker (e.g. CTRL-Z or CTRL-D) or null character is inserted in an audit trail.

2. invalidating assumptions and properties on which lower-level components rely includes attacks taking advantage of errors in the source code where the code assumes (explicitly or implicitly) that security relevant data is in a particular format or has a particular range of values. In these cases we have determined whether they can invalidate such assumptions by causing the data to be in a different format or to have different values, and if so whether this could confer advantage to an attacker.

3. The correct behaviour of the SPD FUA function may be dependent on assumptions that are invalidated under extreme circumstances where resource limits are reached or parameters reach their maximum value. It is important to consider (where practical) the behaviour of the pSHIELD when these limits are reached, for example:

   - ➢ changing dates (e.g. examining how the pSHIELD behaves when a critical date threshold is passed);
   - ➢ filling disks; exceeding the maximum number of users;
   - ➢ filling the audit log;
   - ➢ saturating security alarm queues at a console;
   - ➢ overloading various parts of a multi-user pSHIELD which relies heavily upon communications components;
   - ➢ swamping a network, or individual hosts, with traffic;
   - ➢ filling buffers or fields.

c) Attacks based on disabling or delaying security enforcement include the following items:

1. using interrupts or scheduling functions to disrupt sequencing: it includes investigating the behaviour of the pSHIELD when:

- a command is interrupted (with CTRL-C, CTRL-Y, etc.);
- a second interrupt is issued before the first is acknowledged.

The effects of terminating security critical processes (e.g. an audit daemon) should be explored. Similarly, it may be possible to delay the logging of audit records or the issuing or receipt of alarms such that it is of no use to an administrator (since the attack may already have succeeded).

2. disrupting concurrence: it includes investigating the behaviour of the pSHIELD when two or more subjects attempt simultaneous access. It maybe that the pSHIELD can cope with the interlocking required when two subjects attempt simultaneous access, but that the behaviour becomes less well defined in the presence of further subjects. For example, a critical security process could be put into a resource-wait state if two other processes are accessing a resource which it requires.
3. using interfaces between components which are not visible at a higher level of abstraction may provide a means of delaying a time-critical trusted process.

d) Physical attacks can be categorised into physical probing, physical manipulation, physical modification, and substitution:

1. Physical probing by penetrating the pSHIELD targeting internals of the pSHIELD, e.g. reading at internal communication interfaces, lines or memories.

2. Physical manipulation can be with the pSHIELD internals aiming at internal modifications of the pSHIELD (e.g. by using optical fault induction as an interaction process), at the external interfaces of the pSHIELD (e.g. by power or clock glitches) and at the pSHIELD environment (e.g. by modifying temperature).

3. Physical modification of pSHIELD internal security enforcing attributes to inherit privileges or other capabilities that should be denied in regular operation. Such modifications can be caused, e.g., by optical fault induction. Attacks based on physical modification may also yield a modification of the SPD FUA function itself, e.g. by causing faults at pSHIELD internal program data transfers before execution. Note, that such kind of bypassing by modifying the SPD FUA function itself can jeopardise every SPD FUA function we have considered the existence of other measures (possibly environmental measures) that prevent an attacker from gaining physical access to the pSHIELD.

4. Physical substitution to replace the pSHIELD with another IT entity, during delivery or operation of the pSHIELD. Substitution during delivery of the pSHIELD from the development environment to the user should be prevented through application of secure delivery procedures .Substitution of the pSHIELD during operation may be considered through a combination of user guidance and the operational environment, such that the user is able to be confident that they are interacting with the pSHIELD.

3. Direct attacks

1. Direct attack includes the identification of any penetration tests necessary to test the strength of permutational or probabilistic mechanism and other mechanisms to ensure they withstand direct attack.

2. For example, it may be a flawed assumption that a particular implementation of a pseudo-random number generator will possess the required entropy necessary to seed the security mechanism.

3. Where a probabilistic or permutational mechanism relies on selection of security attribute value (e.g. selection of password length) or entry of data by a human user (e.g. choice of password), the assumptions made should reflect the worst case.

4. Probabilistic or permutational mechanisms have been identified during examination of every pSHIELD design document (architectural design and implementation representation) and any other pSHIELD (e.g. guidance) documentation may identify additional probabilistic or permutational mechanisms. Where the design evidence or guidance includes assertions or assumptions (e.g. about how many authentication attempts are possible per minute), we have confirmed that these are correct. This has always been achieved through testing. Direct attacks reliant upon a weakness in a cryptographic algorithm have been considered according to scientific literature regarding this topic.

4. Monitoring

Information is an abstract view on relation between the properties of entities, i.e. a signal contains information for a system, if the pSHIELD is able to react to this signal. pSHIELD processes and stores information represented by user data. Therefore:

a) **information may flow with the user data between subjects by internal pSHIELD transfer or export from the pSHIELD;**
b) **information may be generated and passed to other user data;**
c) **information may be gained through monitoring the operations on data representing the information.**

The information represented by user data may be characterised by security attributes like “classification level” having values, for example, unclassified, confidential, secret, top secret, to control operations to the data. This information and therefore the security attributes may be changed by operations ruled by an access control policy which may describe how to decrease the level by “sanitarisation” or increase the level by combination of data. This is one aspects of an information flow analysis focused on controlled operations of controlled subjects on controlled objects.
The other aspect is the analysis of illicit information flow. This aspect is more general than the direct access to objects containing user data addressed by an access control policy. An unenforced signalling channel carrying information under control of the information flow control policy can also be caused by monitoring of the processing of any object containing or related to this information (e.g. side channels). An enforced signalling channel may be identified in terms of the subjects manipulating resources and the subject or user that observe such manipulation. Classically, covert channels have been identified as timing or storage channels, according to the resource being modified or modulated. As for other monitoring attacks, the use of the pSHIELD is in accordance with the SPD requirements.

Covert channels are normally applicable in the case when the pSHIELD has unobservability AND multi-level separation policy requirements. Covert channels may be routinely spotted during vulnerability analysis and design activities, and should therefore be tested. However, generally such monitoring attacks are only identified through specialised analysis techniques commonly referred to as “covert channel analysis”. These techniques have been the subject of much research and there are many papers published on this subject.

Unenforced information flow monitoring attacks include passive analysis techniques aiming at disclosure of sensitive internal data of the pSHIELD by operating the pSHIELD in the way that corresponds to the guidance documents.

Side Channel Analysis includes crypt analytical techniques based on physical leakage of the pSHIELD. Physical leakage can occur by timing information, power consumption or power emanation during computation of a SPD FUA function. Timing information can be collected also by a remote-attacker (having network access to the pSHIELD), power based information channels requires that the attacker is in the near-by environment of the pSHIELD.

Eavesdropping techniques include interception of all forms of energy, e.g., electromagnetic or optical emanation of computer displays, not necessarily in the near-field of the pSHIELD.

Monitoring also includes exploits of protocol flaws, e.g., an attack on SSL implementation.

5.4 Identification of Potential Vulnerabilities

Potential vulnerabilities may be identified by the evaluator during different activities. They may become apparent during a generic activity or they may be identified as a result of analysis of evidence (documentation or SPD functions implementation device) to search for vulnerabilities.

5.4.1 Encountered

The encountered identification of vulnerabilities is where potential vulnerabilities are identified by the evaluator during the conduct of generic activities on the analysed system, i.e. the evidence are not being analysed with the express aim of identifying potential vulnerabilities.

The encountered method of identification is dependent on the evaluator's experience and knowledge. It is not reproducible in approach, but will be documented to ensure repeatability of the conclusions from the reported potential vulnerabilities.

There is no formal analysis criteria required for this method. Potential vulnerabilities are identified from the evidence provided as a result of knowledge and experience. However, this method of identification is not constrained to any particular subset of evidence.

Evaluator is assumed to have knowledge of the SPD functions technology and known security flaws as documented in the public domain. The level of knowledge assumed is that which can be gained from a security e-mail list relevant to the system SPD functions type, the regular bulletins (bug, vulnerability and security flaw lists) published by those organisations researching security issues in products and technologies in widespread use. This knowledge is not expected to extend to specific conference proceedings or detailed theses produced by university research if the expected SPD functions value is not high.

However, to ensure the knowledge applied is up to date, the evaluator may need to perform a search of public domain material.

Differently if the expected SPD functions value is high the search of publicly available information is expected to include conference proceeding and theses produced during research activities by universities and other relevant organisations.

Examples of how these may arise (how the evaluator may encounter potential vulnerabilities):

a) while the evaluator is examining some evidence, it sparks a memory of a potential vulnerability identified in a similar product type, that the evaluator believes to also be present in the system under evaluation;

b) while examining some evidence, the evaluator spots a flaw in the specification of an interface that reflects a potential vulnerability.

This may include becoming aware of a potential vulnerability in a system through reading about generic vulnerabilities in a particular product type in an IT security publication or on a security e-mail list to which the evaluator is subscribed.

Attack methods can be developed directly from these potential vulnerabilities. Therefore, the encountered potential vulnerabilities are collated at the time of producing penetration tests based on the evaluator's vulnerability analysis. There is no explicit action for the evaluator to encounter potential vulnerabilities.

5.4.2 Analysis

The following types of analysis are presented in terms of the evaluator actions.

a) Unstructured Analysis

The unstructured analysis to be performed by the evaluator permits the evaluator to consider the generic vulnerabilities. The evaluator will also apply their experience and knowledge of flaws in similar technology types.

b) Focused

During the conduct of evaluation activities the evaluator may also identify areas of concern. These are specific portions of the system evidence that the evaluator has some reservation about, although the evidence meets the requirements for the activity with which the evidence is associated. For example, a particular interface specification looks particularly complex, and therefore may be prone to error either in the development of the system or in the operation of the system. There is no potential vulnerability apparent at this stage, further investigation is required. This is beyond the bounds of encountered, as further investigation is required.

Difference between potential vulnerability and area of concern:

1) Potential vulnerability - The evaluator knows a method of attack that can be used to exploit the weakness or the evaluator knows of vulnerability information that is relevant to the pSHIELD.

2) Area of concern - The evaluator may be able to discount concern as a potential vulnerability based on information provided elsewhere. While reading interface specification, the evaluator identifies that due to the extreme (unnecessary) complexity of an interface a potential vulnerability may lay within that area, although it is not apparent through this initial examination.

The focused approach to the identification of vulnerabilities is an analysis of the evidence with the aim of identifying any potential vulnerability evident through the contained information. It is an unstructured analysis, as the approach is not predetermined.

This analysis can be achieved through different approaches that will lead to commensurate value of confidence. None of the approaches have a rigid format for the examination of evidence to be performed.

The investigation of the evidence for the existence of potential vulnerabilities may be directed by any of the following:

i) areas of concern identified during examination of the evidence during the conduct of evaluation activities;

ii) reliance on particular functionality to provide separation, identified during the analysis of the architectural design, requiring further analysis to determine it cannot be bypassed;

iii) representative examination of the evidence to hypothesize potential vulnerabilities in the system.

The evaluator will report what actions were taken to identify potential vulnerabilities in the evidence. However, the evaluator may not be able to describe the steps in identifying potential vulnerabilities before the outset of the examination. The approach will evolve as a result of the outcome of evaluation activities.

The activities performed by the evaluator can be repeated and the same conclusions, in terms of the value of SPD functions in the system, can be reached although the steps taken to achieve those conclusions may vary. As the evaluator is documenting the form the analysis took, the actual steps taken to achieve those conclusions are also reproducible.

c) Methodical

The methodical analysis approach takes the form of a structured examination of the evidence. This method requires the evaluator to specify the structure and form the analysis will take (i.e. the manner in which the analysis is performed is predetermined, unlike the focused identification method). The method is specified in terms of the information that will be considered and how/why it will be considered.

This analysis of the evidence is deliberate and pre-planned in approach, considering all evidence identified as an input into the analysis.

The “methodical” descriptor for this analysis has been used in an attempt to capture the characterisation that this identification of potential vulnerabilities is to take an ordered and planned approach. A “method” or “system” is to be applied in the examination. The evaluator is to describe the method to be used in terms of what evidence will be considered, the information within the evidence that is to be examined, the manner in which this information is to be considered; and the hypothesis that is to be generated.

The following provide some examples that a hypothesis may take:

a) consideration of malformed input for interfaces available to an attacker at the external interfaces;

b) examination of a security mechanism, such as domain separation, hypothesising internal buffer overflows leading to degradation of separation;

c) analysis to identify any objects created in the system implementation representation that are then not fully controlled by the SPD functions, and could be used by an attacker to undermine the SPD requirements.

For example, the evaluator may identify that interfaces are a potential area of weakness in the pSHILED and specify an approach to the analysis that “all interface specifications provided in the functional specification and system design will be analysed to hypothesize potential vulnerabilities” and go on to explain the methods used in the hypothesis.

This identification method will provide a plan of attack of the system that would be performed by an evaluator completing penetration testing of potential vulnerabilities in the system. The rationale for the method of identification would provide the evidence for the coverage and depth of exploitation determination that would be performed on the system.

5.5 Calculating Attack potential

Attack potential is a function of expertise, resources and motivation. There are multiple methods of representing and quantifying these factors.

5.5.1 Treatment of motivation

Motivation is an attack potential factor that can be used to describe several aspects related to the attacker and the assets the attacker desires. Firstly, motivation can imply the likelihood of an attack - one can infer from a threat described as highly motivated that an attack is imminent, or that no attack is anticipated from an un-motivated threat. However, except for the two extreme levels of motivation, it is difficult to derive a probability of an attack occurring from motivation. Secondly, motivation can imply the value of the asset, monetarily or otherwise, to either the attacker or the asset holder. An asset of very high value is more likely to motivate an attack compared to an asset of little value. However, other than in a very general way, it is difficult to relate asset value to motivation because the value of an asset is subjective - it depends largely upon the value an asset holder places on it. Thirdly, motivation can imply the expertise and resources with which an attacker is willing to effect an attack. One can infer that a highly motivated attacker is likely to acquire sufficient expertise and resources to defeat the measures protecting an asset. Conversely, one can infer that an attacker with significant expertise and resources is not willing to effect an attack using them if the attacker's motivation is low.

5.5.2 Determining attack potential

This section examines the factors that determine attack potential, and provides some guidelines to help remove some of the subjectivity from this aspect of the SPD level definition process.

The determination of the attack potential for an attack corresponds to the identification of the effort required to create the attack, and to demonstrate that it can be successfully applied to the pSHIELD (including setting up or building any necessary test equipment), thereby exploiting the vulnerability in the pSHIELD. The demonstration that the attack can be successfully applied needs to consider any difficulties in expanding a result shown in the laboratory to create a useful attack. For example, where an experiment reveals some bits or bytes of a confidential data item (such as a key), it is necessary to consider how the remainder of the data item would be obtained (in this example some bits might be measured directly by further experiments, while others might be found by a different technique such as exhaustive search). It may not be necessary to carry out all of the experiments to identify the full attack, provided it is clear that the attack actually proves that access has been gained to a pSHIELD asset, and that the complete attack could realistically be carried out in exploitation. In some cases the only way to prove that an attack can realistically be carried out in exploitation is to perform completely the attack and then rate it based upon the resources actually required. One of the outputs from the identification of a potential vulnerability is assumed to be a script that gives a step-by-step description of how to carry out the attack that can be used in the exploitation of the vulnerability on another instance of the pSHIELD. In many cases, it is possible to estimate the parameters for exploitation, rather than carry out the full exploitation.

5.5.3 Factors to be considered

The following factors should be considered during analysis of the attack potential required to exploit a vulnerability:

a) Time taken to identify and exploit (Elapsed Time);

b) Specialist technical expertise required (Specialist Expertise);

c) Knowledge of the pSHIELD design and operation (Knowledge of the pSHIELD);

d) Window of opportunity;

e) IT hardware/software or other equipment required for exploitation (Equipment)

In many cases these factors are not independent, but may be substituted for each other in varying degrees. For example, expertise or hardware/software may be a substitute for time. A discussion of these factors follows. (The levels of each factor are discussed in increasing order of magnitude.) When it is the case, the less “expensive” combination is considered in the exploitation phase.

Elapsed time is the total amount of time taken by an attacker to identify that a particular potential vulnerability may exist in the pSHIELD, to develop an attack method and to sustain effort required to mount the attack against the pSHIELD. When considering this factor, the worst case scenario is used to estimate the amount of time required. The identified amount of time is as follows:

a) less than one day;

b) between one day and one week;

c) between one week and two weeks;

d) between two weeks and one month;

e) each additional month up to 6 months leads to an increased value;

f) more than 6 months.

Specialist expertise refers to the level of generic knowledge of the underlying principles, product type or attack methods (e.g. Internet protocols, Unix operating systems, buffer overflows). The identified levels are as follows:

a) Laymen are unknowledgeable compared to experts or proficient persons, with no particular expertise;

b) Proficient persons are knowledgeable in that they are familiar with the security behaviour of the product or system type;

c) Experts are familiar with the underlying algorithms, protocols, hardware, structures, security behaviour, principles and concepts of security employed, techniques and tools for the definition of new attacks, cryptography, classical attacks for the product type, attack methods, etc. implemented in the product or system type.

d) The level “Multiple Expert” is introduced to allow for a situation, where different fields of expertise are required at an Expert level for distinct steps of an attack.

It may occur that several types of expertise are required. By default, the higher of the different expertises factors is chosen. In very specific cases, the “multiple expert” level could be used but it should be noted that the expertise must concern fields that are strictly different like for example HW manipulation and cryptography.

Knowledge of the pSHIELD refers to specific expertise in relation to the pSHIELD. This is distinct from generic expertise, but not unrelated to it. Identified levels are as follows:

a) Public information concerning the pSHIELD (e.g. as gained from the Internet);

b) Restricted information concerning the pSHIELD (e.g. knowledge that is controlled within the developer organisation and shared with other organisations under a non-disclosure agreement);

c) Sensitive information about the pSHIELD (e.g. knowledge that is shared between discreet teams within the developer organisation, access to which is constrained only to members of the specified teams);

d) Critical information about the pSHIELD (e.g. knowledge that is known by only a few individuals, access to which is very tightly controlled on a strict need to know basis and individual undertaking).

The knowledge of the pSHIELD may graduate according to design abstraction, although this can only be done on a pSHIELD by pSHIELD basis. Some pSHIELD designs may be public source (or heavily based on public source) and therefore even the design representation would be classified as public or at most restricted, while the implementation representation for other pSHIELDs is very closely controlled as it would give an attacker information that would aid an attack and is therefore considered to be sensitive or even critical. It may occur that several types of knowledge are required. In such cases, the higher of the different knowledge factors is chosen.

Window of opportunity (Opportunity) is also an important consideration, and has a relationship to the Elapsed Time factor. Identification or exploitation of a vulnerability may require considerable amounts of access to a pSHIELD that may increase the likelihood of detection. Some attack methods may require considerable effort off-line, and only brief access to the pSHIELD to exploit. Access may also need to be continuous, or over a number of sessions.

For some pSHIELDs the Window of opportunity may equate to the number of samples of the pSHIELD that the attacker can obtain. This is particularly relevant where attempts to penetrate the pSHIELD and undermine the SFRs may result in the destruction of the pSHIELD preventing use of that pSHIELD sample for further testing, e.g. hardware devices. Often in these cases distribution of the pSHIELD is controlled and so the attacker must apply effort to obtain further samples of the pSHIELD.

For the purposes of this discussion:

a) unnecessary/unlimited access means that the attack doesn't need any kind of opportunity to be realised because there is no risk of being detected during access to the pSHIELD and it is no problem to access the number of pSHIELD samples for the attack;

b) easy means that access is required for less than a day and that the number of pSHIELD samples required to perform the attack is less than ten;

c) moderate means that access is required for less than a month and that the number of pSHIELD samples required to perform the attack is less than one hundred;

d) difficult means that access is required for at least a month or that the number of pSHIELD samples required to perform the attack is at least one hundred;

e) unfeasible means that the opportunity window is not sufficient to perform the attack (the length for which the asset to be exploited is available or is sensitive is less than the opportunity length needed to perform the attack - for example, if the asset key is changed each week and the attack needs two weeks); another case is, that a sufficient number of pSHIELD samples needed to perform the attack is not accessible to the attacker - for example if the pSHIELD is a hardware and the probability to destroy the pSHIELD during the attack instead of being successful is very high and the attacker has only access to one sample of the pSHIELD.

Consideration of this factor may result in determining that it is not possible to complete the exploit, due to requirements for time availability that are greater than the opportunity time.

IT hardware/software or other equipment refers to the equipment required to identify or exploit a vulnerability.

a) Standard equipment is readily available to the attacker, either for the identification of a vulnerability or for an attack. This equipment may be a part of the pSHIELD itself (e.g. a debugger in an operating system), or can be readily obtained (e.g. Internet downloads, protocol analyser or simple attack scripts).

b) Specialised equipment is not readily available to the attacker, but could be acquired without undue effort. This could include purchase of moderate amounts of equipment (e.g. power analysis tools, use of hundreds of PCs linked across the Internet would fall into this category), or development of more extensive attack scripts or programs. If clearly different test benches consisting of specialised equipment are required for distinct steps of an attack this would be rated as bespoke.

c) Bespoke equipment is not readily available to the public as it may need to be specially produced (e.g. very sophisticated software), or because the equipment is so specialised that its distribution is controlled, possibly even restricted. Alternatively, the equipment may be very expensive.

d) The level “Multiple Bespoke” is introduced to allow for a situation, where different types of bespoke equipment are required for distinct steps of an attack.

Specialist expertise and Knowledge of the pSHIELD are concerned with the information required for persons to be able to attack a pSHIELD. There is an implicit relationship between an attacker's expertise (where the attacker may be one or more persons with complementary areas of knowledge) and the ability to effectively make use of equipment in an attack. The weaker the attacker's expertise, the lower the potential to use equipment (IT hardware/software or other equipment). Likewise the greater the expertise the greater the potential for equipment to be used in the attack. Although implicit, this relationship between expertise and the use of equipment does not always apply, for instance, when environmental measures prevent an expert attacker's use of equipment, or when, through the efforts of others, attack tools requiring little expertise to be effectively used are created and freely distributed (e.g. via the Internet).

5.5.4 Calculation of attack potential – SPD level

Table 4 identifies the factors discussed in the previous section and associates numeric values with the total value of each factor. Where a factor falls close to the boundary of a range It should be considered use of an intermediate value to those in the table. For example, if twenty samples are required to perform the attack then a value between one and four may be selected for that factor, or if the design is based on a publicly available design but the developer has made some alterations then a value between zero and three should be selected according to the evaluation of the impact of those design changes. The table is intended as a guide. The “**” specification in the table in considering Window of Opportunity is not to be seen as a natural progression from the timescales specified in the preceding ranges associated with this factor. This specification identifies that for a particular reason the potential vulnerability cannot be exploited in the pSHIELD in its intended operational environment. For example, access to the pSHIELD may be detected after a certain amount of time in a pSHIELD with a known environment (i.e. in the case of a system) where regular patrols are completed, and the attacker could not gain access to the pSHIELD for the required two weeks undetected. However, this would not be applicable to a pSHIELD connected to the network where remote access is possible, or where the physical environment of the pSHIELD is unknown.

Factor	Value
Elapsed Time	
<= one day	0
<= one week	1
<= one month	4
<= two months	7
<= three months	10
<= four months	13
<= five months	15
<= six months	17
> six months	19
Expertise	
Layman	0
Proficient	3*(1)
Expert	6
Multiple experts	8
Knowledge of pSHIELD	
Public	0

Factor	Value
Restricted	3
Sensitive	7
Critical	11
Window of Opportunity	
Unnecessary / unlimited access	0
Easy	1
Moderate	4
Difficult	10
None	25**(2)
Equipment	
Standard	0
Specialised	4(3)
Bespoke	7
Multiple bespoke	9

Table 4 Calculation of attack potential

(1) When several proficient persons are required to complete the attack path, the resulting level of expertise still remains “proficient” (which leads to a 3 rating).

(2) Indicates that the attack path is considered not exploitable due to other measures in the intended operational environment of the pSHIELD.

(3) If clearly different test benches consisting of specialised equipment are required for distinct steps of an attack, this should be rated as bespoke.

To determine the resistance of the pSHIELD to the potential vulnerabilities identified the following steps should be applied:

a) Define the possible attack scenarios {AS1, AS2, ..., ASn} for the pSHIELD in the operational environment.

b) For each attack scenario, perform a theoretical analysis and calculate the relevant attack potential using Table 1.

c) For each attack scenario, if necessary, perform penetration tests in order to confirm or to disprove the theoretical analysis.

d) Divide all attack scenario {AS1, AS2, ..., ASn} into two groups:

- the attack scenarios having been successful (i.e. those that have been used to successfully undermine pSHIELD SPD function)

- the attack scenarios that have been demonstrated to be unsuccessful.

e) Calculate the SPD value of the tested SPD function as the lowest value of the calculated attack potential for each successful attack scenario

A qualitative rating can classified as following:

Value	**Attack potential required to exploit scenario:**	**pSHIELD resistant to attackers with attack potential of:**
0-9	Basic	No rating
10-13	Enhanced-Basic	Basic
14-19	Moderate	Enhanced-Basic
20-24	High	Moderate
=>25	Beyond high	High

Table 5 Rating of vulnerabilities and pSHIELD resistance

An approach such as this cannot take account of every circumstance or factor, but should give a better indication of the level of resistance to attack required to achieve the standard ratings. Other factors, such as the reliance on unlikely chance occurrences are not included in the basic model, but can be used as justification for a rating other than those that the basic model might indicate. It should be noted that whereas a number of vulnerabilities rated individually may indicate high resistance to attack, collectively the combination of vulnerabilities may indicate that overall a lower rating is applicable. The presence of one vulnerability may make another easier to exploit.

5.5.5 Example # 1 of calculation of SPD level

In this paragraph we show an easy example of the application of the method for the calculation of the SPD level.

Let's assume we have to calculate the SPD level of an authentication SPD function. Let's limit the vulnerability searching guidance to direct attacks to the mechanism implementing this SPD function as described in 5.3.1.

5.5.5.1 SPD function description hypothesis

Let's assume to have gathered the following information from the technologic table described in paragraph 5.1.

SPD authentication function might use a simple pass number authentication mechanism that can be overcome by an attacker who has the opportunity to repeatedly guess another user's pass number. The system can strengthen this mechanism by restricting pass numbers and their use in various ways.

The implementation mechanism places the following restrictions on the pass numbers selected by the entity:

a) the pass number must be at least **four** and no greater than **six** digits long;

b) consecutive numerical sequences are disallowed (such as 7,6,5,4,3);

c) repeating digits is disallowed (each digit must be unique).

d) the pass number mechanism is designed with a terminal locking feature. Upon the sixth failed authentication attempt the terminal is locked for one hour. The failed authentication count is reset after five minutes.

Guidance provided to the users at the time of pass number selection is that pass numbers should be as random as possible and should not be affiliated with the user in some way - a date of birth, for instance.

5.5.5.2 Attack scenario

Physical access to the terminals is not controlled by any effective means. The duration of access to a terminal is not controlled by any effective means. Authorized entities of the system choose their own pass numbers when initially authorized to use the system, and thereafter upon entity request.

5.5.5.3 Theoretical Analysis and calculation

The pass number space is calculated as follows:

a) Patterns of human usage are important considerations that can influence the approach to searching a password space. Assuming the worst case scenario and the user chooses a number comprising only four digits, the number of pass number permutations assuming that each digit must be unique is:

$$7(8)(9)(10)=5040$$

b) The number of possible increasing sequences is seven, as is the number of decreasing sequences. The pass number space after disallowing sequences is:

$$5040-14=5026$$

Based on terminal locking feature as in 5.5.5.1 description an attacker can at best attempt five pass number entries every five minutes, or 60 pass number entries every hour.

On average, an attacker would have to enter 2513 pass numbers, over 2513 minutes, before entering the correct pass number. The average successful attack would, as a result, occur in slightly less than:

$$(2513min)/(60min/hour) \sim 42 \text{ hours}$$

Using the approach to calculate the attack potential, described in the previous section, identifies that it is possible that a layman can defeat the mechanism within days, with the use of standard equipment, and with no knowledge of the implementation mechanism. The resulting sum is 1 as showed in the next table

Factor	Value
Elapsed Time	
<= one week	1
Expertise	
Layman	0
Knowledge of Mechanism	
Public	0
Window of Opportunity	
Unnecessary / unlimited access	0
Equipment	
Standard	0
Total	1

Table 6 Example 1

The attack potential required to effect a successful attack is not rated, as it falls below that considered to be Basic.

Penetration tests should be necessary in order to confirm or to disprove the theoretical analysis.

5.5.6 Example # 2 of calculation of SPD level

Let's consider now the following changes respect to the previous example:

SPD function description changes as:

a) the pass number must be at least **eight** and no greater than **ten** digits long;

Attack scenario changes for:

Physical access to the terminals is not controlled by personnel only during all nightlong when are protected by fenced environments.

Theoretical Analysis and calculation changes as:

a) Number of pass number permutations assuming that each digit must be unique is:

3(4)(5)(6)(7)(8)(9)(10)=1814400

b) The pass number space after disallowing sequences is:

1814400-4=1814396

c) On average, an attacker would have to enter 907198 pass numbers, over 907198 minutes, before entering the correct pass number. The average successful attack would, as a result, occur in slightly less than:

(907198min)/(60min/hour) ~ 15199 hours ~ 630 days

The resulting sum is 20 as showed in the next table.

Factor	Value
Elapsed Time	
> six months	19
Expertise	
Layman	0
Knowledge of mechanism	
Public	0
Window of Opportunity	
Easy	1
Equipment	
Standard	0
Total	20

Table 7 Example 2

The attack potential required to effect a successful attack is rated as High.

6 pSHIELD SPD functional components

The formal description of pSHIELD SPD functions is according to CC security functional requirements which are a standard way of expressing the SPD functional requirements. Annex of this document catalogues the set of functional components, families, and classes.

This chapter defines the content and presentation of the SPD functional components (those useful to mitigate FUA risks) and provides guidance on their organization. According to Common Criteria SPD functional components are categorised in classes and families.

6.1 Class structure

Each functional class includes a class name, class introduction, and one or more functional families.

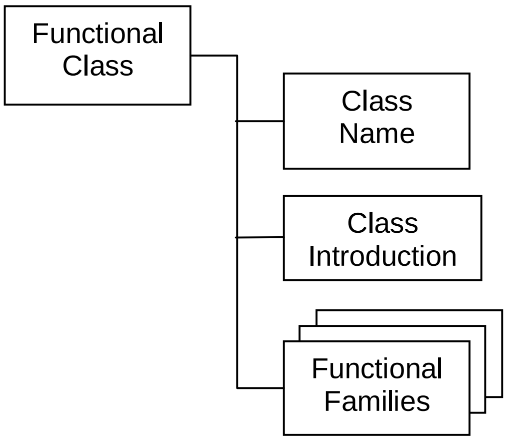


Figure 13

6.1.1 Class name

The class name section provides information necessary to identify and categorise a functional class. Every functional class has a unique name. The categorical information consists of a short name of two characters. The short name of the class is used in the specification of the short names of the families of that class.

6.1.2 Class introduction

The class introduction expresses the common intent or approach of those families to satisfy security objectives. The definition of functional classes does not reflect any formal taxonomy in the specification of the requirements.
The class introduction provides a figure describing the families in this class and the hierarchy of the components in each family

6.2 Family structure

Next figure illustrates the functional family structure in diagrammatic form.

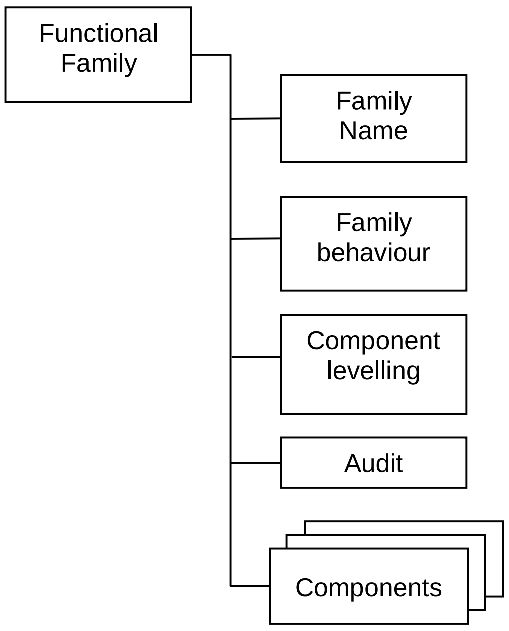


Figure 14

6.2.1 Family name

The family name section provides categorical and descriptive information necessary to identify and categorise a functional family. Every functional family has a unique name. The categorical information consists of a short name of seven characters with the first three identical to the short name of the class followed by an underscore and the short name of the family as follows XX_YYY. The unique short form of the family name provides the principal reference name for the components.

6.2.2 Family behaviour

The family behaviour is the narrative description of the functional family stating its SPD objective and a general description of the functionality offered. These are described in greater detail below:

a) The security objectives of the family address a security problem that may be solved with the help of a pSHIELD that incorporates a component of this family;
b) The description of the functionality summarises all the mechanisms that are included in the component(s).

6.2.3 Component leveling

Functional families contain one or more components, any one of which can be selected for inclusion in pSHIELD functionality package. The goal of this section is to provide information to users in selecting an appropriate functional component once the family has been identified as being a necessary or useful part of their SPD functions.
This section of the functional family description describes the components available, and their rationale. The exact details of the components are contained within each component. The relationships between components within a functional family may or may not be hierarchical. A component is hierarchical to another if it offers more SPD features that could be used to reach a higher SPD level.

6.2.4 Audit

The audit functions contain auditable events to select, if functions from the class AU: Security audit, are included in the pSHIELD functionality package. These functions include SPD relevant events in terms of the various levels of detail supported by the components of the Security audit data generation (AU_GEN) family.
As a consequence, for the evaluation of the SPD audit functionality value we have to consider the minimum value reached by vulnerability analysis and the value defined according to the above description.

6.3 Component structure

Next figure illustrates the functional component structure.

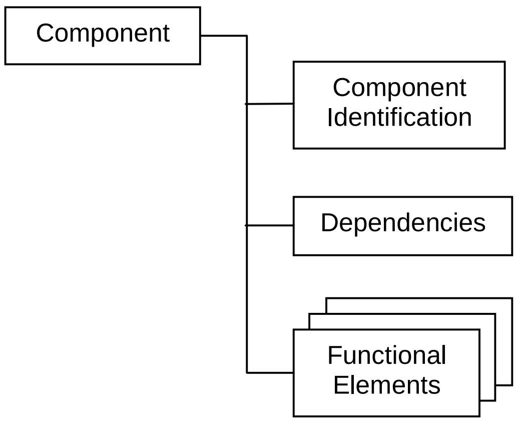


Figure 15

6.3.1 Component identification

The component identification section provides descriptive information necessary to identify, categorise, register and cross-reference a component.
The following is provided as part of every functional component:
A unique name:the name reflects the purpose of the component.
A short name: a unique short form of the functional component name. This short name serves as the principal reference name for the categorisation, registration and cross-referencing of the component. This short name reflects the class and family to which the component belongs and the component number within the family as follows XX_YYY.Z.

A hierarchical-to list: a list of other components that this component is hierarchical to and for which this component can be used to satisfy dependencies to the listed components.

6.3.2 Functional elements

A set of elements is provided for each component. Each element is individually defined and is self-contained.
An SPD functional element is the smallest functionality identified and recognised in a pSHIELD.
When building pSHIELD package, it is not permitted to select only one or more elements from a component. The complete set of elements of a component must be selected for inclusion in a pSHIELD package.

A unique short form of the functional element name is provided as follow XX_YYY.Z.I.

6.3.3 Dependencies

Dependencies among functional components arise when a component is not self sufficient and relies upon the functionality of, or interaction with, another component for its own proper functioning.

Each functional component provides a complete list of dependencies to other functional components. Some components may list “No dependencies”. The components depended upon may in turn have dependencies on other components.

6.3.4 Component catalogue

The grouping of the components does not reflect any formal taxonomy.

It contains classes of families and components, which are rough groupings on the basis of related function or purpose, presented in alphabetic order. At the start of each class is an informative diagram that indicates the taxonomy of each class, indicating the families in each class and the components in each family. The diagram is a useful indicator of the hierarchical relationship that may exist between components.

In the description of the functional components, a section identifies the dependencies between the component and any other components.

In each class a figure describing the family hierarchy similar to the next figure, is provided. In the next figure first family, named Family 1, contains three hierarchical components, where component 2 and component 3 can both be used to satisfy dependencies on component 1. Component 3 is hierarchical to component 2 and can also be used to satisfy dependencies on component 2.

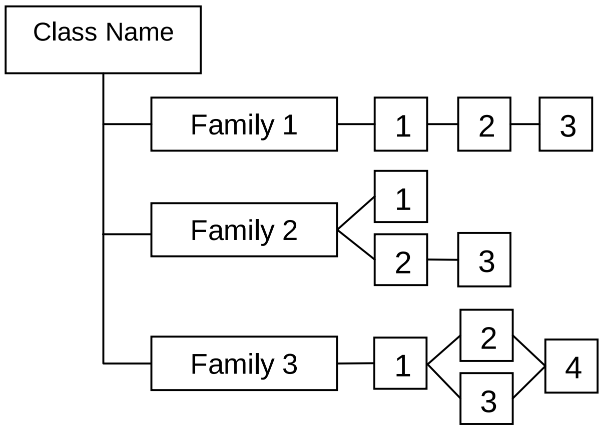


Figure 16

In Family 2 there are three components not all of which are hierarchical. Components 1 and 2 are hierarchical to no other components. Component 3 is hierarchical to component 2, and can be used to satisfy dependencies on component 2, but not to satisfy dependencies on component 1.
In Family 3, components 2, 3, and 4 are hierarchical to component 1.
Components 2 and 3 are both hierarchical to component 1, but non-comparable. Component 4 is hierarchical to both component 2 and component 3.

These diagrams are meant to complement the text of the families and make identification of the relationships easier. They do not replace the “Hierarchical to:” note in each component that is the mandatory claim of hierarchy for each component. The following paragraph shows an example of a pSHIELD SPD functional component

6.4 pSHIELD SPD function components catalogue example

6.4.1 Class AU - SPD audit

SPD audit involves recognizing, recording, storing and analyzing information related SPD relevant activities (i.e. activities controlled by pSHIELD). The resulting audit records can be examined to determine which security relevant activities took place and which user is responsible for them.

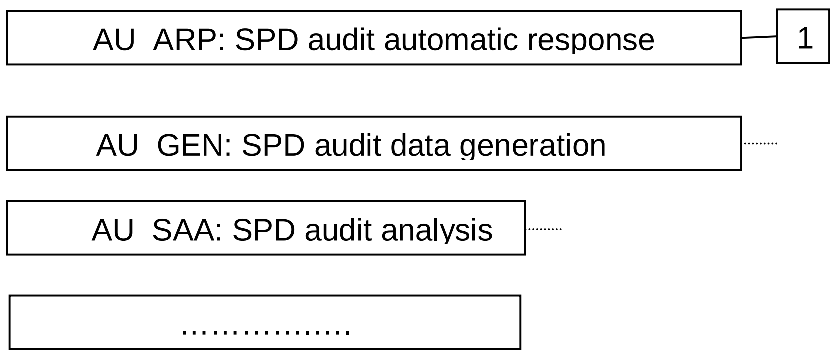

AU ARP: SPD audit automatic response
1
AU_GEN: SPD audit data generation
AU SAA: SPD audit analysis
…………….

Figure 17

6.4.1.1 SPD audit automatic response (AU_ARP)

Family Behaviour

This family defines the response to be taken in case of detected events indicative of a potential SPD violation.

Component leveling

AU ARP SPD audit automatic
1

Audit: AU_ARP.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Basic/Enhanced Basic: Actions taken due to potential SPD violations.

AU_ARP.1 SPD alarms

Hierarchical to:	No other component.
Dependencies:	AU_SAA.1 Potential violation analysis
AU_ARP.1.1	The pSHIELD shall take [assignment: list of actions] upon detection of a potential [assignement: Security, Privacy Dependability] violation.

7 SPD functions for reducing NFUA metric construction method

Faults that do not belong to FUA are those introduced by error, such as configuration problems, incompetence issues, accidents, and so on.

These kinds of errors could affect the pSHIELD at any time of its life-cycle so to reduce NFUA category it is necessary to introduce a set of controls (support functions) in its entire life-cycle.

The methodology definition about how to obtain the value of SPD assurance differs from the previous, in fact it isn't based on SPD functions enforced to reduce the risk risen by FUA but it is based on pSHIELD controls level carried out during its whole life cycle on the SPD life cycle support functions.

This process of evaluation is called Life Cycle Support Elements (LCSE) Analysis and it is conducted to prevent the misuse of the pSHIELD SPD functions evaluating life-cycle support elements (these elements are generally defined in documents; e.g.: guidance, manuals, development environment description, etc.).

Misuse may arise from:

- incomplete guidance documentation;
- unreasonable documentation;
- unintended misconfiguration of the pSHIELD;
- forced exception behaviour of the pSHIELD.

The measure of life-cycle documents is estimated through Common Criteria standard giving a numeric value for each passed activity and then summing it. (The standard defines each activity in a very formal way).

By Common Criteria experiences the value assigned for a single component is included in [0;1] interval. These functionalities are only supporting and so the total value that must be multiplied with the total value calculated for SPD Function cannot increment the SPD assurance of the pSHIELD.

As in the previous metric definition, it is possible to consider a catalogue of supporting functions which is composed by one class (Life Cycle) structured in families and components; moreover for each component are defined evaluation activities that evaluator must perform to define a value. The sum of values, defined for the activities performed on a particular component, assign the value of assurance for that SPD life cycle support component [6].

In the catalogue of support functions the value assigned for each activity is indicated in square brackets.

7.1 pSHIELD SPD Life cycle support elements

The pSHIELD SPD Life Cycle support elements structure matches that of pSHIELD SPD functional components described in chapter 6.

The only difference is that for each element it must be defined a number of evaluation activities.

7.1.1 pSHIELD SPD life cycle support elements catalogue example

7.1.1.1 Class LC - SPD life cycle

Life-cycle support is an aspect of establishing discipline and control in the processes of refinement of pSHIELD during its development and maintenance.

Confidence in the correspondence between pSHIELD requirements and the pSHIELD itself is greater if SPD analysis and the production of the evidence (pSHIELD and its documentation) are done on a regular basis as an integral part of the development and maintenance activities.

In the pSHIELD life-cycle it is distinguished whether it is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the PSHIELD is handed over to the user.

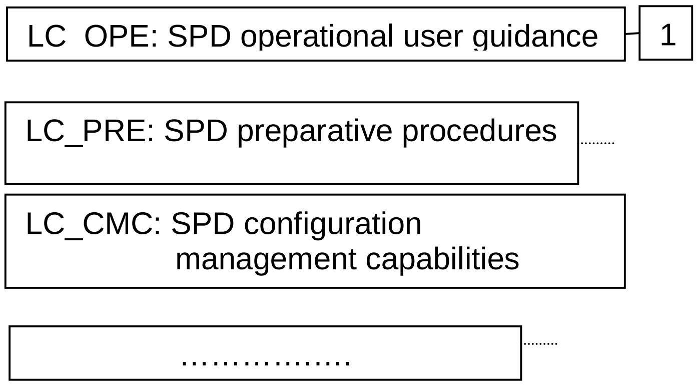

LC OPE: SPD operational user guidance
1
LC_PRE: SPD preparative procedures
LC_CMC: SPD configuration management capabilities
……………..

Figure 18

SPD operational user guidance (LC_OPE)

Family Behaviour

This family provides a measure of confidence that non-malicious users, administrators, application providers and other exercising the external interfaces of the pSHIELD will understand the SPD compliant operation and will use it as intended.

Component leveling

LC_OPE.1 Operational user guidance

Hierarchical to:	No other component.
Dependencies:	No dependencies.
LC_OPE.1.1	The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.
LC_OPE.1.2	The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the pSHIELD in a secure manner.
LC_OPE.1.3	The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate.
LC_OPE.1.4	The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the pSHIELD.
LC_OPE.1.5	The operational user guidance shall identify all possible modes of operation of the pSHIELD (including operation following failure or operational error), their consequences and implications for maintaining secure operation.

LC_OPE.1.6	The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfill the security objectives for the operational environment.
LC_OPE.1.7	The operational user guidance shall be clear and reasonable.

8 Quantifying the SPD measure of composed SPD **functions**

Several measures for Security, Privacy and Dependability of systems have been presented in the literature, including adversary work factor, adversary financial expenses, adversary time, probability like measures, or simply defining a finite number of categories for security, privacy or dependability of systems.
In this chapter it is described two types of measurements:

- A deterministic approach to give a single measure of SPD assurance level of a composed embedded system starting from an intuitive graphical representation of the system itself [7].
- A system for security assurance assessment

8.1 SPD for medieval castle

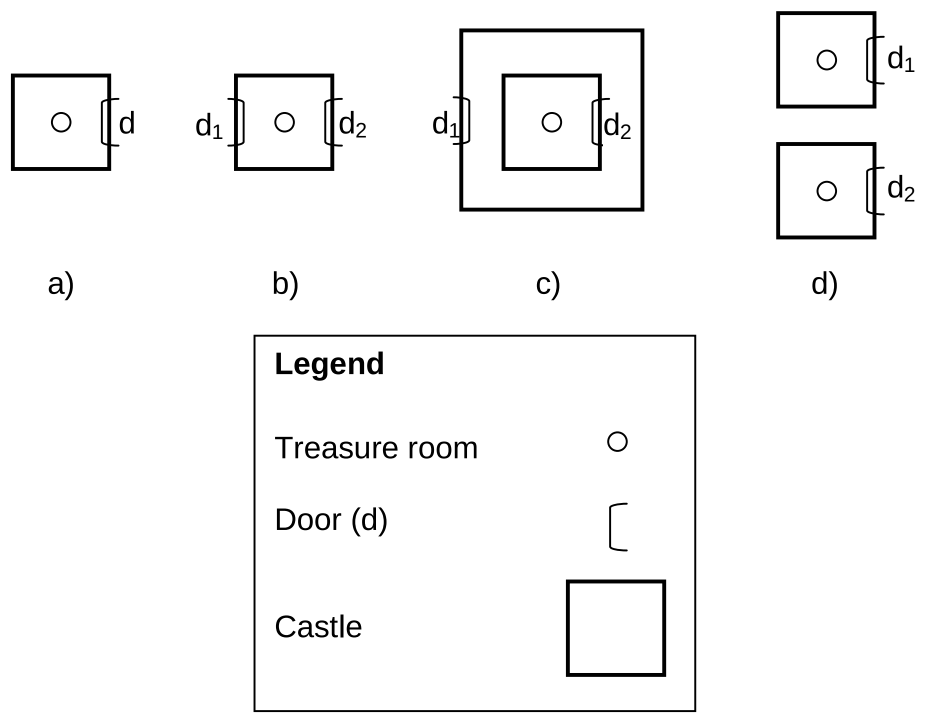


Figure 19

We can consider that, in the middle ages, security, privacy and dependability were obtained by building castles. To be useful in times of peace, castles possess doors, which we assume are the only targets for the attackers in times of war. In this section, we show how the SPD of a castle with up to two doors can be computed under the assumption that the SPD of each door is known.
Castle a) illustrated in Figure 19 is the simplest castle we can think of. It has a wall (depicted as a square), a treasure room (depicted by a circle) which is the attackers' main target, and a door d, which is the only way for the attackers to get into the castle. Thus, the SPD of the castle equals the SPD of the door.

The wall of Castle b) has two doors d_1 and d_2, and we assume that d_1 is weaker than d_2. The two doors allow the attackers to strike the castle at two points simultaneously. Thus, the castle's SPD measure will be weaker than or equal to the SPD measure of d_1 (we assume that d_1 and d_2 are totally independent so castle's SPD measure will be equal to the SPD measure of d_1).
In contrast, the SPD of castle c) may be stronger than the security of a castle with only one door. Here, the attackers must break into two doors to get into the treasure room. If we again assume that d_1 is weaker than d_2, the castle's SPD is at least as good as the SPD of d_2.
In the last example (castle d)), the attackers have two castles they may choose to attack. We consider an attack to be successful if the attackers get into one of the two treasure rooms. However, the distance of the castles is too large to allow for a simultaneous attack of both castles. Thus, the security of both castles will be in the interval $[d_1; d_2]$.

8.1.1 SPD of systems with two SPD functions

Formalizing the ideas from the previous section, the following pieces of information are needed to compute the SPD level of a (medieval or modern) system:
– a SPD measure (levelling) M
– the SPD measure of its SPD functions (or doors[2]): $d_1, d_2, \dots, d_n$
– a function: $d : M^n \rightarrow M$, which maps the SPD measure of the doors to the SPD level of the overall system.
To deal with the complexity of today's systems, we assume that the function d can be depicted as a term using different operators. A modelling method for secure systems is thus constructed by first defining a SPD measure and then defining a set of operators to combine the SPD functions measures.

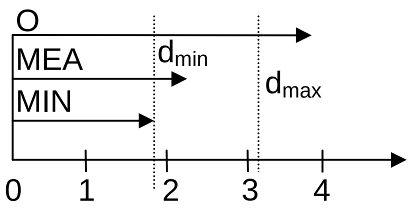


Figure 20

Figure 20 shows three operations for the unbounded continuous case with interval $[0; +\infty)$. If we call d_{min} the smaller operand and d_{max} the larger operand, we can classify these operations into:
– MIN-Operations, if $d = d_{min}$
– MEAN-Operations, if $d_{min} \leq d \leq d_{max}$
– OR-Operations, if $d_{max} \leq d \leq +\infty$
where both operands are unbounded deterministic variables and values are computed by the measure of SPD metrics of basic components as summarized in chapter 5.

8.1.1.1 MIN operation

A MIN-Operation should be used for a system which resembles castle b) from Figure 19. In this case, the defenders have to defend both doors. Thus, the system is as weak as d_{min}. In fact a potential attacker can attack both doors (SPD functions) at the same time, without additional efforts or costs, and the weaker door is the first to be broken. So the

$$d_{MIN} = MIN\ (d_{min}, d_{max})$$

[2] a castel's door can be seen without distinction as an interface of a function implementing security, privacy or dependability

When a potential attacker can attack n-doors (SPD functions) with previous hypothesis the formula becomes the following

$$d_{MIN} = MIN\ (d_1, d_2, \ldots, d_n)$$

8.1.1.2 OR operation

For systems corresponding to the situation named castle c) in Figure 19, OR-operations can be used.
To defend the castle, either d_{min} or d_{max} has to be defended. A corresponding system is at least as strong as d_{max}.
This kind of system models the concept of “defense in depth” where it is used multiple defense mechanisms in layers across the system to protect the assets. We use multiple defenses so that if one defensive measure fails there are more behind it to continue to protect them.
Under the assumption that the doors are attacked one after another, i.e. that the second door cannot be attacked before the attackers successfully broke the first door, the security of the castle can be computed by:

$$OR(d_{min}, d_{max}) = d_{OR} = d_{min} + d_{max}$$

In the presence of a n-doors sequence it can be considered the following formula:

$$OR(d_1, d_2, \ldots, d_n) = d_{OR} = d_1 + d_2 + \ldots + d_n$$

If doors are protected with the same lock, lock-picking the second lock might be much easier after successfully opening the first door, and so on. This case can model redundancy of SPD functions, we can indicate this with OR_n, where n is the number of SPD functions carrying out the redundancy.
As a first approximation we propose to calculate the SPD measure for an OR_n system by:

$$OR_n(d) = d_{OR_n} = d + \sum_{i=2}^{n} \frac{d}{i}$$

To model this situation we can indicate the castle c) (Figure 19) as depicted in the next figure just to underline that the system is protected by the redundancy of the same SPD function.

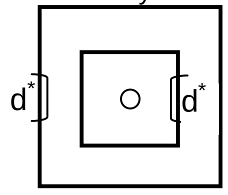


Figure 21

8.1.1.3 MEAN and POWER MEAN Operation

In some systems, the attackers may first choose from one of two doors and, in a second step, attack the system as if it had only one door. This scenario was introduced before as castle d) in Figure 19. Clearly, such a system is stronger than any system with two doors which can be attacked concurrently, but weaker than any system where the attackers must break into two doors. Thus, its security lies in the interval $[d_{min}; d_{max}]$.

Therefore, it is straightforward to apply a mathematical mean operation to model these kinds of systems, which has the same property ($d_{min} \leq d \leq d_{max}$).
If the attackers randomly choose a door with equal probability 0.5 for each door, the security of the system is:

$$MEAN(d_{min}, d_{max}) = d_{MEAN} = \frac{d_{min} + d_{max}}{2}$$

which is the arithmetic mean of d_{min} and d_{max}. In a more general case, the attackers might have some knowledge which doors are more vulnerable and prefer the doors with a lower security. In other scenarios, the defenders will have some information on the attackers' preferences and be able to strengthen the doors which are most likely to be attacked. In this case, it is more likely that the attackers choose the strong door. Both scenarios can be taken into account for using the general power mean M_p defined as:

$$d_{MEAN} = M_p = \left(\frac{d_{min}^p + d_{max}^p}{2}\right)^{\frac{1}{p}}$$

The parameter p determines the amount of knowledge the attackers and defenders have. If p equals one, the power mean equals the arithmetic mean, i.e. neither the attackers nor the defenders have an influence on which door is chosen. If p becomes greater, the knowledge of the defenders increases and thus the probability that the attackers choose the strong door. This situation can model honeypot solution to distract attackers from attacking more valuable system. For example, if p is 2, M_p becomes the root of squared means, which can be computed by:

$$d_{MEAN} = M_2 = RSM = \sqrt{\frac{d_{min}^2 + d_{max}^2}{2}}$$

In the extreme case, i.e. $p \to +\infty$, the attackers always choose the strong door and thus:

$$d_{MEAN} = d_{max}.$$

If p is smaller than 1, the attackers know the castle well and the weak door is more likely to be under attack. For example, if $p \to 0$, M_p is called the geometric mean G defined as:

$$d_{MEAN} = M_0 = G = \sqrt{d_{min} \bullet d_{max}}$$

In the other extreme case, i.e. $p \to -\infty$, the attackers will choose the weakest door and thus:

$$d_{MEAN} = d_{min}$$

However, choosing the right p weights is very difficult in practice. Where it is possible, we can estimate the value of p from the vulnerability analysis as described in chapter 5.3.
In the presence of n elements as castle d) in Figure 19, where we can consider with d_1, d_2,…d_n doors the formulas for different cases become as follows:
If the attackers randomly choose a door with equal probability 1/n for each door, the security of the system is:

$$MEAN(d_1,...,d_n) = d_{MEAN} = \frac{d_1 + ... + d_n}{n}$$

which is the arithmetic mean of d_1, …, d_n.

In a more general case, the attackers might have some knowledge which doors are more vulnerable and prefer the doors with a lower security. In other scenarios, the defenders will have some information on the attackers' preferences and be able to strengthen the doors which are most likely to be attacked. In this case, it is more likely that the attackers choose the strong door. Both scenarios can be taken into account for using the general power mean M_p defined as:

$$d_{MEAN} = M_p(d_1,....,d_n) = \sqrt[p]{\frac{1}{n}\sum_{i=1}^{n} d_n^p}$$

The parameter p determines the amount of knowledge the attackers and defenders have.
In the extreme case, i.e. $p \to +\infty$, the attackers always choose the strong door and thus:

$$d_{MEAN} = d_{MAX} = MAX(d_1, d_2, \ldots, d_n)$$

In the other extreme case, i.e. $p \rightarrow -\infty$, the attackers will choose the weakest door and thus:

$$d_{MEAN} = d_{MIN} = MIN(d_1, d_2, \ldots, d_n)$$

8.1.2 SPD measure of systems with n-SPD functions

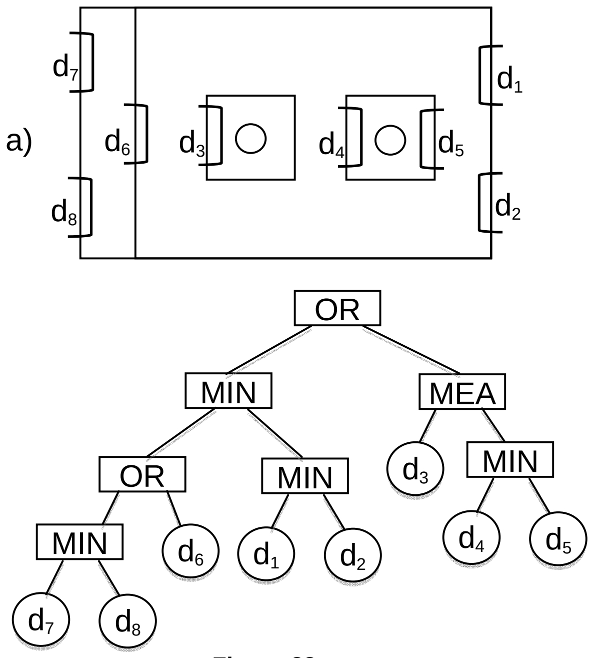


Figure 22

If we consider a castle with eight doors and two treasure rooms, as shown in Figure 22 a), we define that the attackers are successful if they are able to get into one of the two treasure rooms. A semantically equivalent representation of the system is shown in Figure 22 b). This representation can be easily implemented by the proposed ontology for the SPD functions modelling.
Figure 22 b) was created by starting at the doors, which now form the leaves of a tree. Then, the nodes were repeatedly connected in pairs by an OR, MIN or MEAN gate, respectively. For system evaluation, we can replace the abstract doors by the set of SPD functions interfaces which expose an attack surface.
A mathematical expression for the SPD measure of this system can be defined as follows:

$$d = OR(MIN(OR(MIN(d_7, d_8), d_6), MIN(d_1, d_2)), MEAN(d_3, MIN(d_4, d_5)))$$

In this case we can replace “OR” with “+” operator so this function becomes:

$$d = MIN(MIN(d_1, d_2), (d_6 + MIN(d_7, d_8))) + MEAN(d_3, MIN(d_4, d_5))$$

8.1.3 Corrective value introduced by SPD Life Cycle Support Element

In order to take into account the life cycle element (operational manuals, installation guides, etc…) the d value of the SPD measure obtained by the method described previously must be multiplied with d_{LC}. The total value is:

$$d_{TOT} = d * d_{LC}$$

Because the d_{LC} is comprised by 0 and 1, we are sure that well done life cycle elements cannot increase the total SPD metric value, but misunderstanding ones can decrease it.

8.1.4 Example of an application scenario

In this paragraph we show a practical example of application of SPD functions composition approach. We consider a reduced control system installed on a train as depicted in the following figure

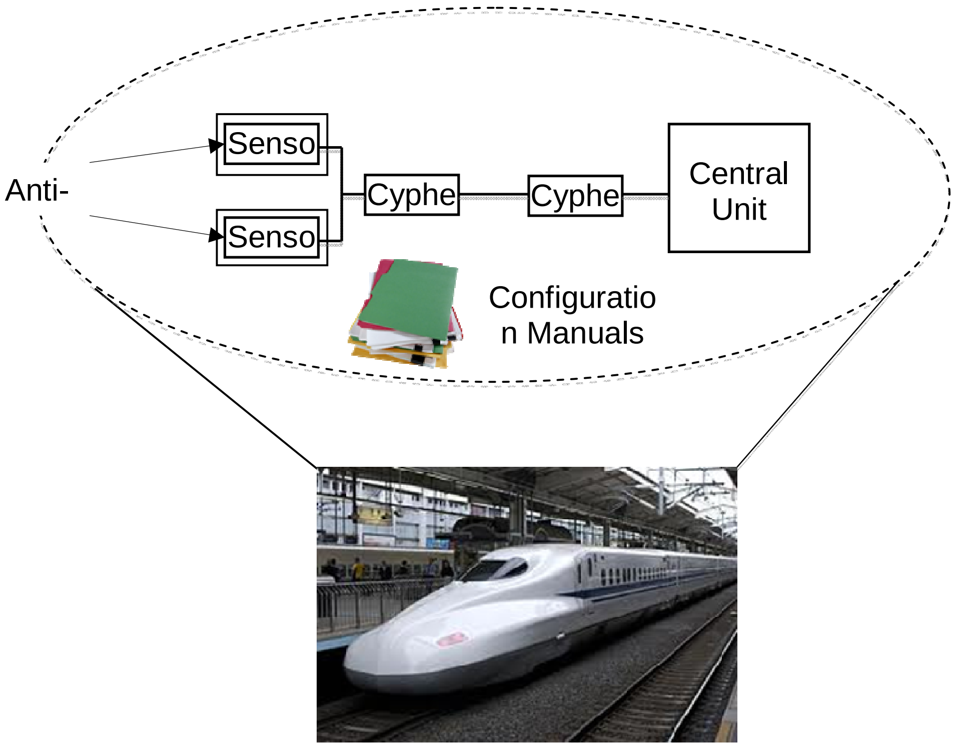


Figure 23

It is composed by a control unit connected by means of a ciphered connection to a sensor in a redundancy configuration and the related configuration manuals. The assets to protect are data that are sent by sensor to central unit and that are recorded inside the central unit itself. In this system we considered the following SPD functions:

1. Anti-tampering redundancy(sensor)
2. Configuration manuals (system)
3. Cypher (data transfer)
4. Identification & Authentication (central unit)
5. Access control (central unit)

According to the described approach this system can be modeled as shown in the following graphic:

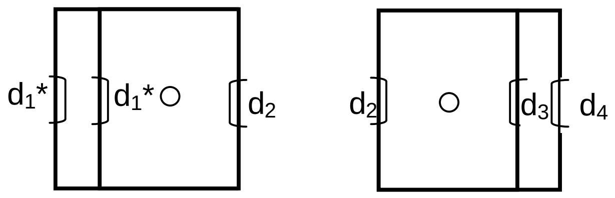


Figure 24

where:
d_1* = SPD measure of sensor anti-tampering strength in a redundant configuration
d_2 = SPD measure of cipher strength
d_3 = SPD measure of access control strength
d_4 = SPD measure of identification and authentication strength
The correspondent system tree representation of the application scenario is:

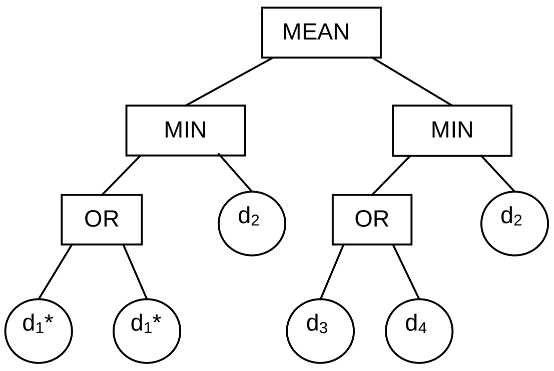


Figure 25

The mathematical expression for the SPD measure of this application scenario system can be defined as follows:

$$d_{TOT} = MEAN\left(MIN\left(OR_2\left(d_1^*\right), d_2\right), MIN\left(OR\left(d_3, d_4\right), d_2\right)\right) * d_{LC}$$

where d_{LC} = SPD measure of life-cycle documentation

9 A System for Security Assurance Assessment

Security assurance (SA) is the **objective confidence that an entity meets its security requirements**. A promised approach to determine the security level of a system in terms of detected vulnerabilities and relations between them is the attack graphs approach. Until now, this approach does not take into account system dynamic behavior but only the static vision.

In general **vulnerability** means probability that a threat will impact a system's operation. These two targets, SA for the proposed approach here and vulnerability for the medieval castle approach makes difference in the aggregated metric.

In the next section we will concentrate on security issues, because they will dominate SPD-tuples targeted for pSHILED project.

9.1 SA Assessment

Attribute: property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means.

It is not always possible to directly measure SA attributes of a given complex system. In that case we have to decompose the system under consideration into entities, which are measurable parts. To combine all the measured SA values of entity attributes into system wide values by taking into account the relations between these attributes we need aggregation methods.

The SA assessment process has five steps:

- **Modelling**: decompose the system into SA-relevant irreducible entities and capture SA-related properties of system structure.
- **Metrics assignment**: assign metrics to each identified entity.
- **Measurement**: measure the SA level of irreducible entities by means of selected metrics.
- **Aggregation**: combine measured results to derive an SA level per entity and for the system.
- **Interpretation**: evaluate the assurance posture of the system based on the aggregated values and visualize the results.

9.2 SA Value

The **SA level of a system entity** is a set, where the value of each element in that set corresponds to the elementary SA value of one of the SA attributes. **An SA attribute describes an aspect of an SA relevant functionality of the system entity.**

If we compare low-level SA data from different sources it can be totally incomparable. We solve this problem by normalizing considered SA values in the range of $[0,1]$ where 0 means that the SA attribute is not present in the entity and 1 means that it is perfect. Values in between may be quantitatively described as weak or strong.

9.3 SA Relations

Real physical connections are modelled as a pair of unidirectional **logical relations**, one in each direction. Figure 26 shows the abstraction of real connections from logical interactions between the elementary SA attributes of the corresponding entities.

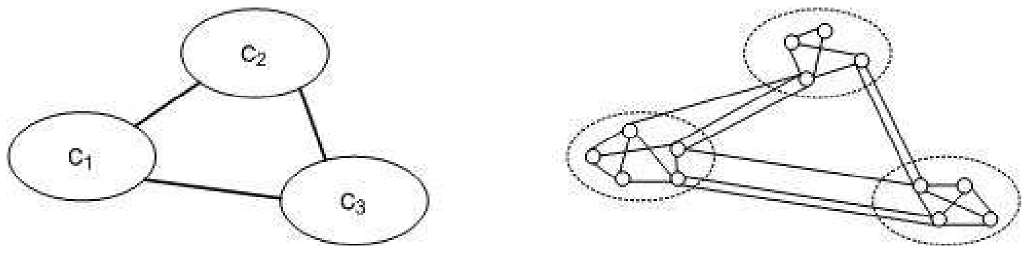


Figure 26 Actual relations between entities are represented as dependencies of their attributes

It is assumed that the relations between the entities are known since identification of the relations between elementary attributes of system entities it takes place in the modelling phase.

9.4 System Aggregation Process

The **aggregation problem** is a process of aggregating *n*-tuples, *n* ≥ 1, of objects all belonging to a given set of real numbers and a given scale, into a single object of the same scale. An **aggregation operator** is a function $Aggreg: \mathbf{R}^n \longrightarrow \mathbf{R}$, which assigns a real number *y* to any *n*-tuple $(x_1, x_2, \dots, x_n)$:

$$y = Aggreg(x_1, x_2, \dots, x_n) \tag{1}$$

The single SA value representing the SA level of an entity is calculated by the application of an aggregation operator *Aggreg*. The attributes of the system entities at a level of abstraction cannot be always directly measured. In this case we have to repeat the decomposition until the decomposed sub-entities attributes are measurable.

In a hierarchical graph called **entities-dependency graph** each level corresponds to a level of decomposition. The components of this graph are:

- the nodes which represent the system entities and
- the edges which represent the functional relationships between the identified entities.

From entities-dependency graph, we obtain an **attributes-dependency graph -** a tree in which the values of the leaves are known. It is assumed that the SA values of the attributes of the irreducible entities can be obtained. This graph has some features:

- an actual entity (invisible in this graph) is a combination of some nodes of the same level,
- the nodes can have the same nature due to the fact that a SA attributes can appear in many entities.

From the attributes-dependency graph we can calculate the values of the nodes of higher levels and finally, arrive at the overall system SA value as the root of the tree, by taking into account the relations between a node and their child nodes. When a number of entities operate together, forming more complex behaviours as a collective, **emergent attributes** can appear. The complex behaviour or attributes are not an attribute of any such single entity, nor can they easily be predicted or deduced from behaviour of the lower-level entities.

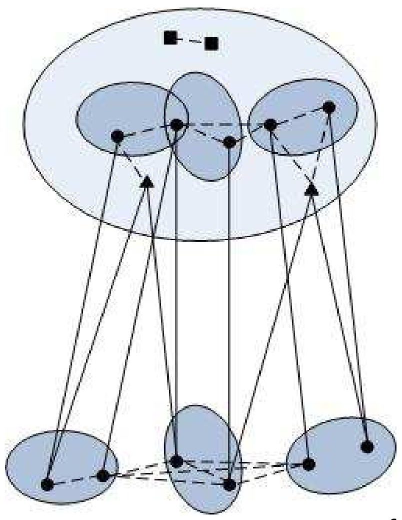


Figure 27 Emergent attributes can appear as a consequence of connecting system entities

Dependent emergent attributes (triangles) may be formed as a result of the composition of lower-level attributes. Independent emergent attributes (squares) are irreducible and do not depend on lower-level attributes.

There are two possible approaches to develop the SA attributes representing the SA level of each entity:

1) **Bottom-up**: find out all possible SA attributes of the irreducible entities and then assess how well these attributes link to overall system SA value, or
2) **Top-down***:* begin with some general attributes representing the whole system and then break them down into somewhat more details and finally, arrive at measurable attributes of the irreducible entities.

The second is more applicable for the following reasons:

- when assessing high-level entities, we are often interested in general attributes and not the detailed ones,
- system's detailed attributes of the first approach are hard to define but we usually find the general attributes.

The relations between the attributes that cannot be decomposed into the relations of lower-levels are therefore called **emergent relations**.

9.4.1 Aggregation with emergent attributes

The **attributes-dependency graph** is a hierarchical graph illustrated in Figure 28, where:

- the nodes are the security assurance (SA) attributes of system entities,
- the leaves are the attributes of the irreducible entities,
- the edges are either decomposition relations (normal lines) or emergent relations (dashed lines).

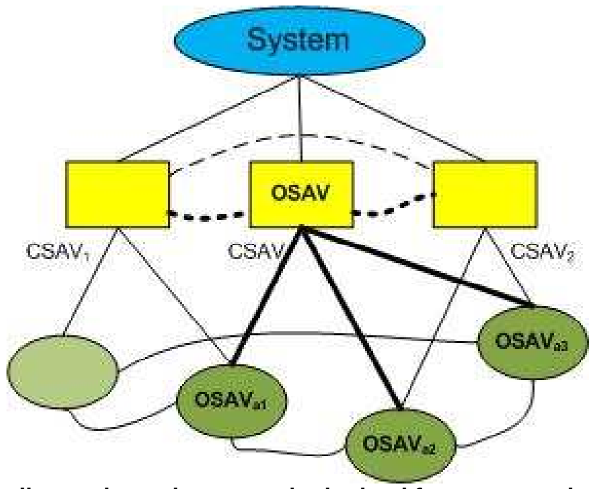


Figure 28 The attributes-dependency graph obtained from system decomposition

The calculation of the SA value of a node in the attributes-dependency graph is made in two steps:

1. calculate the composition value from its child nodes,
2. use the obtained result to determine the actual operational SA value by taking into account the emergent relations with other nodes.

Let denote $CSAV_a$ as the **SA value of the node under consideration**, and $OSAV_{a1}, \dots, OSAV_{an}$ as **operational SA values** of its child nodes. The first step, i.e. the **composition value of a node ($CSAV$) in** the attributes-dependency graph is calculated from the operational values ($OSAVs$) of its child nodes by applying a function $Aggreg_{com}: \mathbf{R}^n \rightarrow \mathbf{R}$ on these values depending on the decomposition relations between the node and its child nodes:

$$CSAV_a = Aggreg_{com}(OSAV_{a1}, \dots, OSAV_{an}) \qquad (2)$$

The first step has to be computed in every node of the same level. The results are the composition value ($CSAV$) for each attribute. Now we denote $OSAV_a$ **as the operational SA value of the node**, $CSAV_a$ as the composition value of the node, $CSAV_{ai}$ (1 ≤ *i* ≤ *m*) as the composition values of the nodes having the emergent relations with the node under consideration. We define a function $Aggreg_{emer}: \mathbf{R}^n \rightarrow \mathbf{R}$ which describes the effect of the emergent relations. In case that there are no emergent relations, it follows that $OSAV_a = CSAV_a$. The operational SA value ($OSAV$) of a node is:

$$OSAV_a = Aggreg_{emer}(CSAV_a, CSAV_{a1}, \dots, CSAV_{am}) \qquad (3)$$

The values of the nodes in the upper levels are obtained from the values of the leaves in the attributes-dependency graph by using the steps represented in (2) and (3). The **operational SA value of the overall system** is determined as a result of applying an appropriate aggregation operator $Aggreg_{com}$ to its attributes.

Finding correct emergent relations between nodes or appropriate decomposition relations between nodes and their parent node in the attributes-dependency graph is crucial for choosing appropriate aggregation operators ($Aggreg_{com}$ and $Aggreg_{emer}$).

In general relations are different, thus we have to select different aggregation operators for each kind of these relations.

9.4.2 Aggregation Operators

We define **aggregated node** as a node of evaluation and **elementary nodes** as the arguments of the aggregation operator which can include the aggregated node. The **SA value of an aggregated node** is represented as $SAV_{aggregated}$. Here are presented some aggregation operators that can be used to combine SA values of some nodes in the attributes-dependency graph.

9.4.2.1 Max

The **max** operator can be used when *S* reflects a positive compensation between nodes. That it should result in a positive effect on the SA value of the aggregated node.

Let us consider two antivirus applications AV_1 and AV_2 where the AV_1 can destroy equal important viruses v_1, v_2 and v_3 and the AV_2 can destroy only the v_1. Then we get the aggregated SA value of these two antiviruses from:
$SAV_{aggregated} = \max(SAV_1, SAV_2)$

where SAV_1 and SAV_2 are the SA values of AV_1 and AV_2 respectively.

9.4.2.2 Union

We can use the mathematical function for a **union** in the case where the AV_1 can destroy the virus v_1, v_2 and v_3 while the AV_2 can destroy the v_1, v_4 and v_5:

$$SAV_{aggregated} = SAV_1 + (1 - SAV_1).SAV_2 = SAV_1 + SAV_2 - SAV_1.SAV_2.$$

9.4.2.3 Min

If *S* represents a relationship between elementary nodes that are vital to the success of their aggregated node then failure of any child node to carry out its SA functionalities will result in the inability of the parent node to perform its SA functionalities.

Let us take an example. If the house is composed of a door and a window, the integrity of the house depends on these two elements. The unauthorised break-ins can occur if either the door or the window is compromised. In this case the **min** operator is used.

Thus, when combining *n* user accounts of a single computer, the aggregated value could be calculated as:
$SAV_{aggregated} = \min(SAV_1, \dots, SAV_n)$

where SAV_i is the SA value of the account *i* (1 ≤ *i* ≤ *n*).

9.4.2.4 Multiply

The **multiply** operator can be used to determine the SA value of the node under consideration in the case where *S* expresses a negative compensation between elementary nodes. If SAV_i, 1 ≤ *i* ≤ *n*, is the SA value of the elementary argument *i*, this operator is defined as follows:

$$SAV_{aggregated} = SAV_1 \times SAV_2 \times \cdots \times SAV_n$$

9.4.2.5 Weighted-sum and Average

The operator **weighted-sum** could be used in the case where *S* describes a relation among elementary nodes each contributing to an independent aspect of the aggregated node. The failure of any elementary node not necessarily causes the functional failure of the corresponding aggregated node. In this case, we have to define the relative importance between elementary nodes. The SA value of the aggregated node can be described as:

$$SAV_{aggregated} = \sum_{i=1}^{n} w_i \, SAV_i \quad \textbf{with} \sum_{i=1}^{n} w_i = 1$$

where n is the number of elementary nodes, SAV_i is the SA value of elementary node i, w_i is the corresponding weight percentile.

Average is a special case of the weight-sum operator where the weight percentiles are equal for all elementary nodes.

The weighted-sum operator can be extended to represent the emergent effects by reserving a relative importance factor (weight) for these effects. In this case, the SA value of the observed node is:

$$SAV_{observed} = ew.SAV_{emer} + \sum_{i=1}^{n} w_i \, SAV_i \quad \textbf{with} \quad ew + \sum_{i=1}^{n} w_i = 1$$

where ew is the weight percentile for the emergent effects which has been calculated as SAV_{emer}.

9.4.3 Aggregation Patterns

Using standard entities and relations, which have already been separately assessed can simplify the SA assessment process.

Patterns are a format for capturing insights and experiences of expert. They are representations of knowledge of how particular problems can be solved. The more information a pattern has, the more important structure becomes. Structure leads to uniformity of patterns. Thus, people can compare them easily.

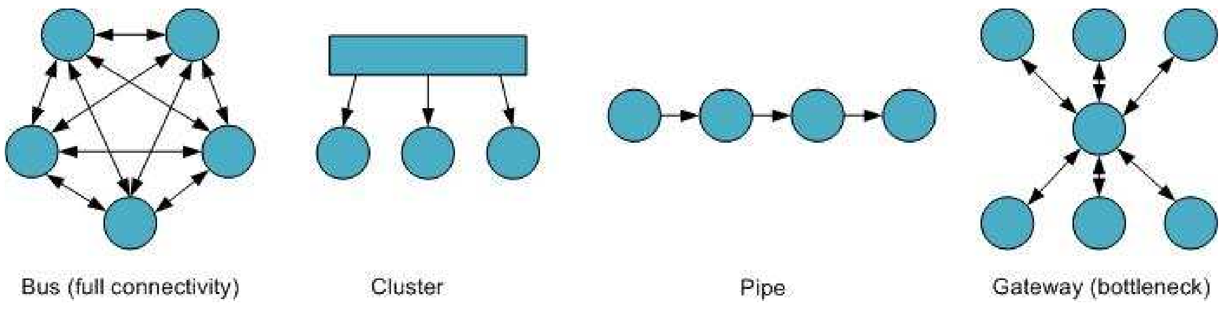


Figure 29 Examples of canonical architectures

In system security assurance assessment we decompose the given system down to know blocks (patterns) for which every aggregation operator is known a priori.

9.4.4 Aggregation

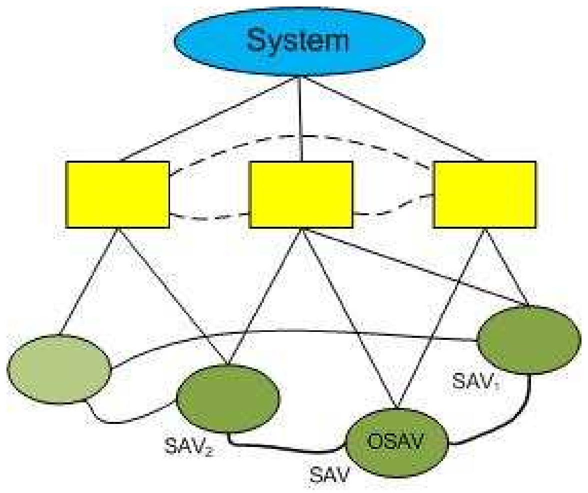


Figure 30 Attributes-dependency graph obtained when combining a number of evaluated entities

After combining a number of entities, which have been evaluated independently, we can use the following method to calculate the **SA value of the whole system.**

After each independent evaluation, the obtained results are the set representing the values of the SA attributes for each entity. The combination of these entities forms an **attributes-dependency graph,** which is almost the same as the graph illustrated in Figure 30, except that the values of the leaves are the independent ones. There are therefore some relations between the leaves.

If ***SAV*** is the **independent SA value of the attribute** being evaluated and SAV_i ($1 \le i \le k$) is the **independent value of the related leaf** and *k* is the number of the relevant leaves, the $OSAV_a$ of one leaf can be calculated by taking into account the relations with other leaves:

$$OSAV_a = Aggreg_{con}(SAV, SAV_1, \dots, SAV_k) \qquad (4)$$

The function $Aggreg_{con}: \mathbf{R}^n \rightarrow \mathbf{R}$ represents the **effect of the connection relations**. The $OSAV_a$ of every leaf in the attributes-dependency graph illustrated in Figure 30 has to be determined using equation (4). The value of the higher-level nodes can be obtained through the two steps process as described above and finally, arrive at the overall operational SA value.

9.5 Attack Graphs

The static vision describes the system as implemented and estimates the level to which the system can be secured during operation. This level depends on system vulnerabilities, the configurations of deployed hardware and software and especially the relations between systems. **Attack graphs** allow us to consider **potential attacks** and give us a clear picture about what attacks might happen in a network and about their consequences.

An **attack tree** is a structure where each possible exploit chain ends in a leaf state that satisfies the attacker's goals. An **attack graph** is an attack tree in which some or all common states are merged. In the attack graphs nodes and arcs represent actions the attacker takes and changes in the network state caused by these actions. Attacker wants with these actions to obtain normally restricted privileges on one or more network components such as user's computers, routers, or firewalls by taking advantage of vulnerabilities in software or communication protocols.

9.5.1 Definitions

Each **host** has at least one **interface** which have zero or more open **ports**, which accept connections from other hosts. Each port has zero or more **vulnerability instances** and depending on them, an attacker is able to obtain one of four **access levels** (vulnerability post-conditions) on a host:

- "root" or administrator access,
- "user" or guest access,
- "DoS" or denial-of-service, or
- "other", a confidentiality and/or integrity loss.

An attacker **state** is the combination of a host and an access level, and may provide the attacker zero or more **credentials,** any information used as access control as passwords or private keys. A vulnerability instance may require zero or more of credentials. Vulnerability **locality** (remotely or locally exploitable) and credentials serve as **preconditions** to exploitation of a vulnerability instance.

A **vulnerability** is a weakness where an attacker could gain access to a system such as software flaws, trust relationships, and server mis-configuration.

Reachability analysis determines which hosts are reachable from an already compromised host and which communication ports can be used to connect and compromise new vulnerable hosts. Reachability analysis is critical for building attack graphs, since an attack can only proceed to new victims that can be reached from compromised hosts.

9.5.2 Building Attack Graph

Example: **Sample network** (
Figure 31)
The attacker starts in a position A. All other hosts have a single open port and a single remotely-exploitable vulnerability, except G which has only a single open port. The attacker from host A can directly compromise hosts B, C and D. The firewall FW drops all the traffic, except the one from C and D to host E, thus the attacker can pass through the firewall and compromise host E from C and D. From E, the attacker can compromise F, completing the process, since G has a personal firewall preventing other hosts to reach it but it can reach F.

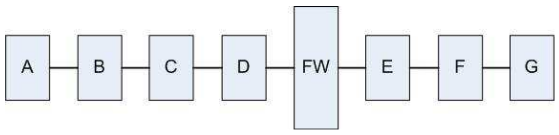
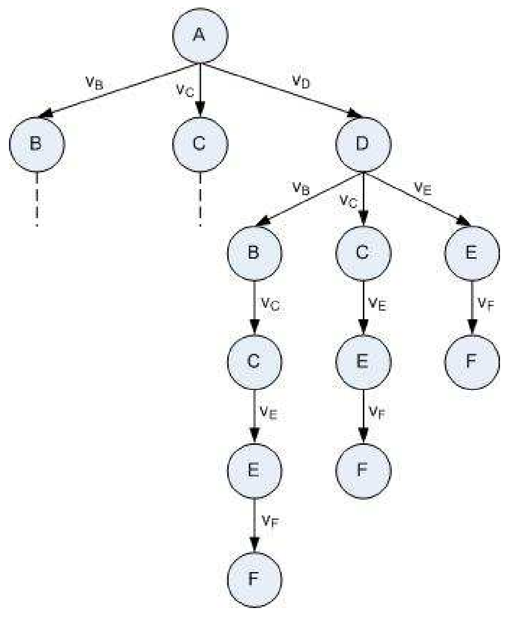


Figure 31 Sample network

Figure 32 shows the **full graph** for the sample network (the children of the B and C nodes at the top level are not shown in the figure) in which **nodes are states**, and **edges** correspond to **vulnerability instances**. Path of **full graph** for the sample network is shown in Figure 32 (the children of the B and C nodes at the top level are not shown in the figure) in which **nodes are states**, and **edges** correspond to **vulnerability instances**. Full graphs depend heavily on the starting position of the attacker. We can see in the example that internal attacks originating from G to E cannot be represented. In the full graphs computational complexity quickly becomes challenging as the network size increases, since they contain a lot of redundant information, e.g. sub trees from C to E and to F are repeated two times in Figure 32.



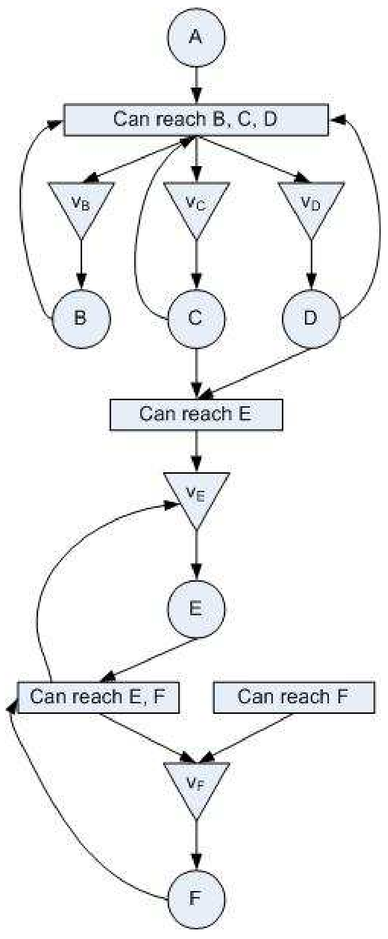


Figure 32 Full graph for sample network

Figure 33 MP graph for sample network

The **multi prerequisite graph (MP graph)** for example network is shown in Figure 33. It has the contentless edges and three types of nodes:

- **state nodes** (circles),
- **pre-condition nodes** (rectangles) and
- **vulnerability instance nodes** (triangles).

In the MP graph no node is explored more than once, and a node only appears on the graph if the attacker can successfully obtain it. A weak point of the MP graph is that it cannot represent all possible positions of the attackers.

Reachability is the only pre-condition in our example. Outbound edges can go only from state nodes to pre-condition nodes, and then from the pre-condition nodes can go only to vulnerability instance nodes. Finally, outbound edges from vulnerability instance nodes can go only to state nodes. That shows that a state node provides post conditions, which are used as pre-conditions to exploit vulnerability instances, which provide more states to the attacker.

MP graph is an intermediate step to build the **host-based attack graph** shown in Figure 34 where:

- the edges are the **vulnerability instances** and
- the nodes are corresponding **hosts**.

It is possible to bypass the weakness of MP graph with host-based graph in a following way: We can consider an external attacker, i.e. to build a graph with an attacker position outside the considered network. Then we examine all the hosts which have not been explored with the algorithm and build MP graphs from these hosts until all edges to already explored hosts in the first step have been taken into consideration. Hence, the host-based graph shows all hosts which can be compromised from any host the attacker has compromised.

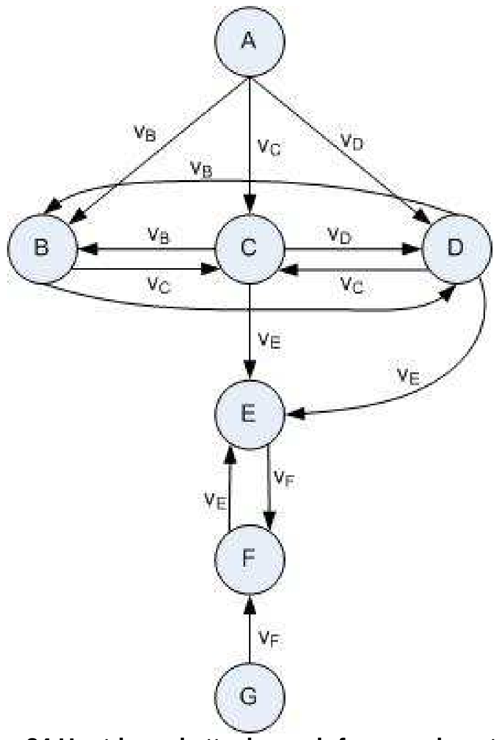


Figure 34 Host-based attack graph for sample network

9.5.3 Static Evaluation Based on Attack Graph

Host-based graphs can show every malicious action placed at any host or attacker starting location to any host in the network. All the direct attack paths that other hosts in the considered network can compromise to that node are represented with the number of inbound edges of a node in the host-based attack graph. Let consider the attackability metric of a host in the context of the system under study, which can be computed from the number of inbound edges.

The **attackability metric** represents the likelihood that a node will be successfully attacked. If *nb_inbound_edges* is the number of inbound edges of the node under consideration and Σ *nb_inbound_edges* is the total number of inbound edges of every nodes in the graph (i.e. number of edges in the graph), the **attackability metric** is calculated as follows:

$$SA_{Attackability} = \left(1 - \frac{nb_{inbound_{edges}}}{\sum nb_{inbound_{edges}}}\right)$$

9.6 Examples

9.6.1 Generic Approach

Let have a system composed by entities. We can construct an entities-dependency graph after decomposing the system. From entities-dependency graph we can construct attributes-dependency graph.

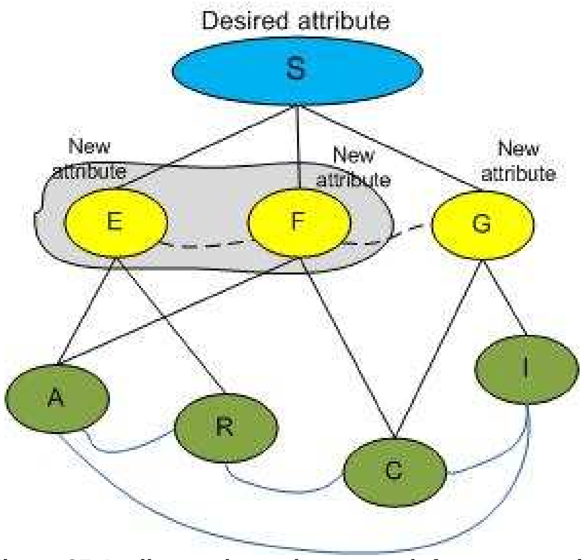


Figure 35 Attributes-dependency graph for an example

First of all we have to calculate the SA values of the leaves. Values of the leaves can be derived from an attack graph. We can make an attack graph for our system, which helps us to see all the vulnerabilities and in the following all the attributes needed for a secure system. For example, we can calculate the attackability metric and thus get an SA value: SAV. Then we have to make a pre-step.

Pre-step:
In this step we have to calculate the operational SA value for all leaves, considering the relation between leaves and known values SAV. This is done as follows:

$$OSAV_A = Aggreg_{con}(SAV_A, SAV_I, SAV_R)$$
$$OSAV_R = Aggreg_{con}(SAV_A, SAV_C, SAV_R)$$
$$OSAV_C = Aggreg_{con}(SAV_C, SAV_I, SAV_R)$$
$$OSAV_I = Aggreg_{con}(SAV_A, SAV_I, SAV_C)$$

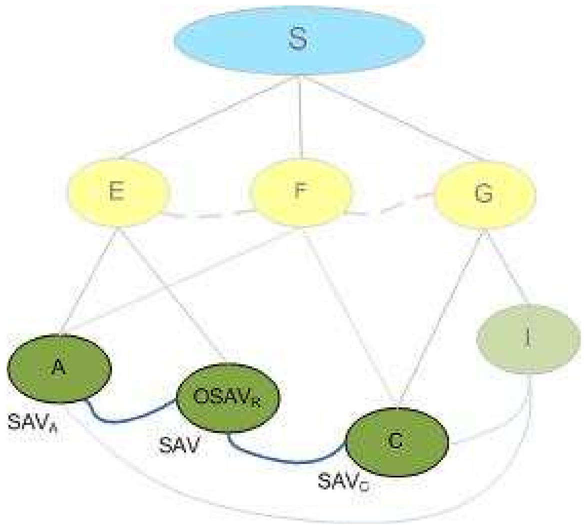


Figure 36 Pre-step for the node R

First step:
In this step we calculate the composition value of the node under composition from its child nodes.
$CSAV_E = Aggreg_{com}(OSAV_A, OSAV_R)$
$CSAV_F = Aggreg_{com}(OSAV_A, OSAV_C)$
$CSAV_G = Aggreg_{com}(OSAV_C, OSAV_I)$

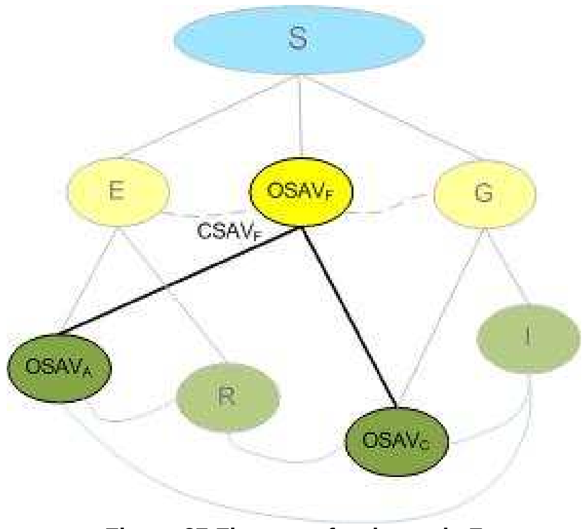


Figure 37 First step for the node F

Second step:
Now we have CSAV for every node – for each attribute. We have to consider emerging attributes and calculate operational SA values for each node.
$OSAV_E = Aggreg_{emer}(CSAV_E, CSAV_F)$
$OSAV_F = Aggreg_{emer}(CSAV_E, CSAV_F, CSAV_G)$
$OSAV_G = Aggreg_{emer}(CSAV_G, CSAV_F)$

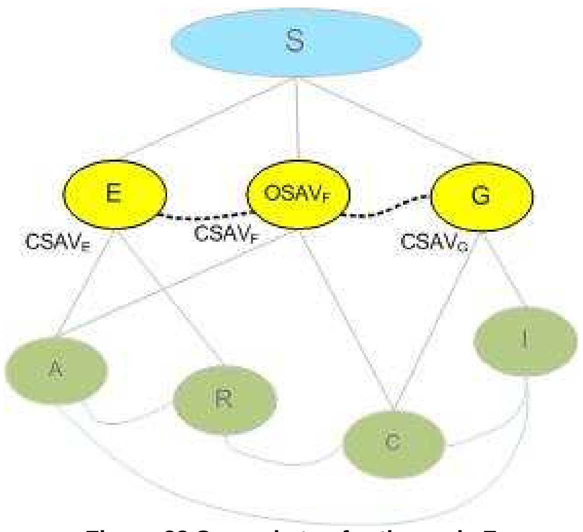


Figure 38 Second step for the node F

Last step:
The last step is the calculation of SA value under composition of the system.
$CSAV_S = Aggreg_{com}(OSAV_E, OSAV_F, OSAV_G)$
$OSAV_S = CSAV_S$

10 Future Work

The proposal is to combine the two approaches, the “castle” approach and approach “SA with emerging attributes”, which have similarities.

Let consider the pSHIELD system and decompose it into entities until the decomposed sub-entities attributes are measurable. In that way is obtained an entities-dependency graph. From entities-dependency graph we have to construct attributes-dependency graph with emerging attributes (e.g. top-down approach). Since the values of the leaves have to be known for the composition, we have to calculate them. This can be done by calculating metrics for each attribute that we are interested in, e.g. minimal attack potential. This value is denoted with SAV (d in document M0.2 and d_{LC}, which has to be considered). Next step is the composition considering the emerging relations with the aggregation process. The idea is the combination of the approach “SA with emerging attributes” and “castle” approach composition. The “SA with emerging attributes” process for aggregation can be used, since it considers emerging relations (operator $Aggreg_{emer}$) with operators from the “castle” approach, where are specifically defined.

This approach measures security assurance of the nodes, which represent SPD attributes. The advantage of this approach is that considers the emerging attributes, which appear when a number of entities operate together. In that way all attributes of SPD are considered and especially the attributes in which we are interested.

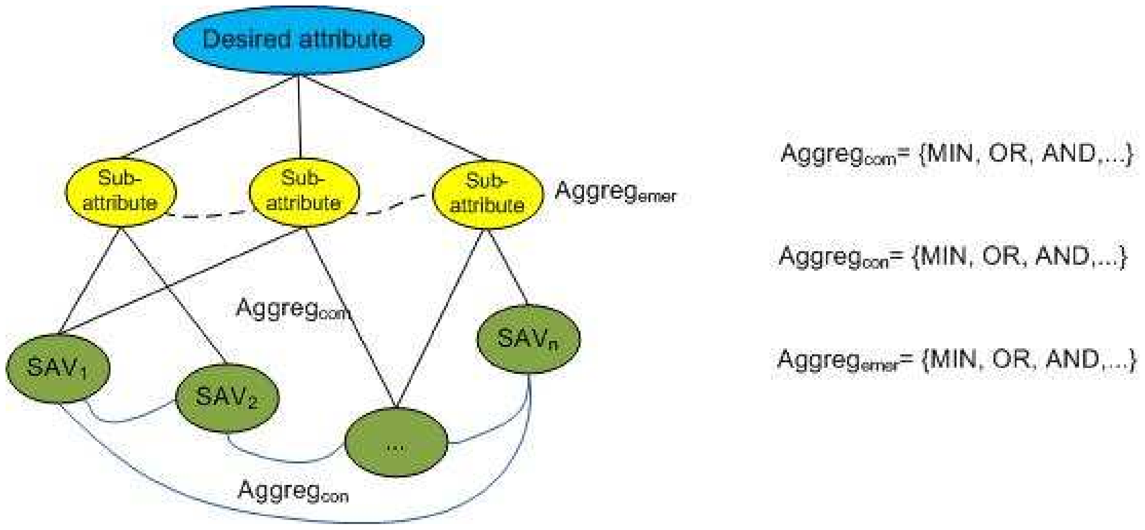


Figure 39 An example of proposal approach: the right operators have to be chosen

11 Conclusions

This deliverable presents a step towards modelling SPD systems by a “divide and conquer” approach as it is widely known from the area of reliability/availability modelling.

For this purpose, basic SPD measures are provided by Common Criteria approach described in chapter 5. Then Chapter 8 explains the basic ideas of how to compute a system’s SPD measure from measures of its components. Repeatedly, two SPD measures of components are combined into a single value by operators. In section 8.1.4 an example shows how to apply the proposed method in an application scenario.

At the same time, in the same section SA with emerging attributes” approach is presented, which is a divide and conquer approach to measure security assurances in the intelligent complex systems. The presented aggregation method constitutes an appropriate approach to combine all the operational security assurance values of relevant system entities. The interactions between entities are abstracted as relations between their representing attributes. The effects of the emergent relations are taken into account in the calculation of the security assurance value of an attribute in the context of a system. Some possible aggregation operators and their applications are described. Also attack graphs are presented, which allow considering potential attacks and giving a clear picture about what attacks might happen in a network and about their consequences. A generic example of “SA with emerging attributes” approach is given and some strengths and weaknesses.

In Annex A there is the catalogues definition, in particular these catalogues are defined specifically for pSHIELD.

12 Annex A

12.1 pSHIELD SPD functional components catalogue

12.1.1 Class AU - SPD audit

SPD audit involves recognizing, recording, storing and analyzing information related to SPD relevant activities (i.e. activities controlled by pSHIELD). The resulting audit records can be examined to determine which relevant activities took place and which user is responsible for them.

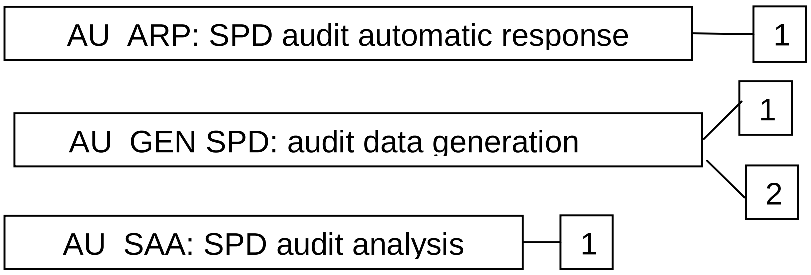

AU ARP: SPD audit automatic response
1
AU GEN SPD: audit data generation
1
2
AU SAA: SPD audit analysis
1

Figure 40

12.1.1.1 SPD audit automatic response (AU_ARP)

Family Behaviour

This family defines the response to be taken in case of detected events indicative of a potential SPD violation.

Component leveling

AU ARP SPD audit automatic response
1

Audit: AU_ARP.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Basic/Enhanced Basic: Actions taken due to potential SPD violations.

AU_ARP.1 SPD alarms

Hierarchical to:	No other component.
Dependencies:	AU_SAA.1 Potential violation analysis
AU_ARP.1.1	The SPD functionalities shall take [assignment: list of actions] upon detection of a potential [assignment: Security, Privacy Dependability] violation.

12.1.1.2 SPD audit data generation (AU_GEN)

Family Behaviour

This family defines requirements for recording the occurrence of security relevant events that take place under SPD functionalities control. This family identifies the level of auditing, enumerates the types of events that shall be auditable by the SPD functionalities, and identifies the minimum set of audit-related information that should be provided within various audit record types.

Component leveling

AU GEN SPD: audit data generation
1
2

Audit: AU_GEN.1, AU_GEN.2

There are no auditable events foreseen.

AU_GEN.1 Audit data generation

Hierarchical to:	No other component.
Dependencies:	PT_STM.1 Reliable time stamps
AU_GEN.1.1	The SPD functionalities shall be able to generate an audit record of the following auditable events:

a) Start-up and shutdown of the audit functions;

b) All auditable events for the [selection, choose one of: minimum, basic, detailed, not specified] level of audit; and

c) [assignment: other specifically defined auditable events].

AU_GEN.1.2 The SPD functionalities shall record within each audit record at least the following information:

a) Date and time of the event, type of event, subject identity (if applicable), and the outcome (success or failure) of the event; and

b) For each audit event type, based on the auditable event definitions of the functional components [assignment: other audit relevant information].

AU_GEN.2 Audit data generation

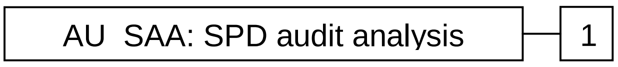
Hierarchical to:	No other component.
Dependencies:	AU_GEN.1 Audit data generation
	PT_STM.1 Reliable time stamps
AU_GEN.2.1	For audit events resulting from actions of identified users, the SPD functionalities shall be able to associate each auditable event with the identity of the user that caused the event.

12.1.1.3 SPD audit analysis (AU_SAA)

Family Behaviour

This family defines requirements for automated means that analyses system activity and audit data looking for possible or real security violations. This analysis may work in support of intrusion detection, or automatic response to a potential security violation.

Component leveling



Audit: AU_SAA.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Minimal: Enabling and disabling of any of the analysis mechanisms;

b) Minimal: Automated responses performed by the tool.

AU_SAA.1 Potential violation analysis

Hierarchical to:	No other component.
Dependencies:	AU_GEN.1 Audit data generation
AU_SAA.1.1	The SPD functionalities shall be able to apply a set of rules in monitoring the audited events and based upon these rules indicate a potential violation of the enforcement of the SPD functionalities.
AU_SAA.1.2 events:	The SPD functionalities shall enforce the following rules for monitoring audited

a) Accumulation or combination of [assignment: subset of defined auditable events] known to indicate a potential security violation;

b) [assignment: any other rules].

12.1.2 Class CS – Cryptographic support

The SPD functionality may employ cryptographic functionality to help satisfy several high-level security objectives. These include (but are not limited to): identification and authentication, non-repudiation, trusted path, trusted channel and data separation. This class is used when the pSHIELD implements cryptographic functions, the implementation of which could be in hardware, firmware and/or software.

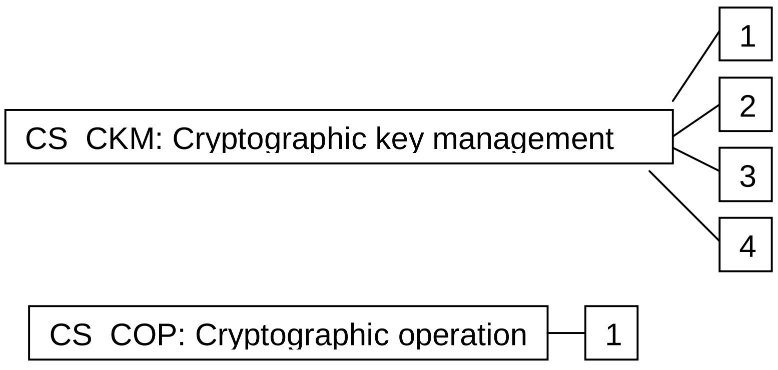

CS CKM: Cryptographic key management
1
2
3
4
CS COP: Cryptographic operation
1

Figure 41

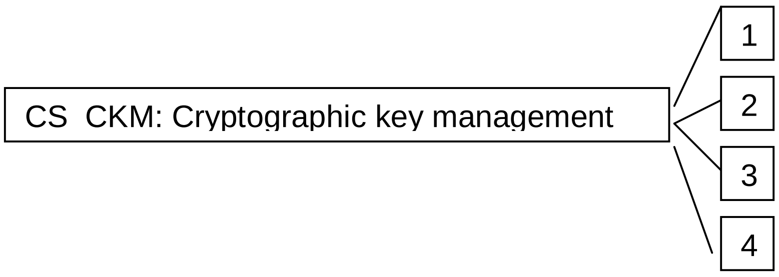
12.1.2.1 Cryptographic key management (CS_ CKM)

Family Behaviour

Cryptographic keys must be managed throughout their life cycle. This family is intended to support that lifecycle and consequently defines requirements for the following activities: cryptographic key generation, cryptographic key distribution, cryptographic key access and cryptographic key destruction.

This family should be included whenever there are functional requirements for the management of cryptographic keys.

Component leveling



Audit: CS_CKM.1, CS_CKM.2, CS_CKM.3, CS_CKM.4

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Minimal: Success and failure of the activity.

b) Basic: The object attribute(s) and object value(s) excluding any sensitive information (e.g. secret or private keys).

CS_CKM.1 Cryptographic key generation

Hierarchical to:	No other component.
Dependencies:	[CS_CKM.2 Cryptographic key distribution, or CS_COP.1 Cryptographic operation]
	CS_CKM.4 Cryptographic key destruction
CS_CKM.1.1	The SPD functionalities shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm [assignment: cryptographic key generation algorithm] and specified cryptographic key sizes [assignment: cryptographic key sizes] that meet the following: [assignment: list of standards].

CS_CKM.2 Cryptographic key distribution

Hierarchical to:	No other components.
Dependencies:	CS_CKM.1 Cryptographic key generation
	CS_CKM.4 Cryptographic key destruction
CS_CKM.2.1	The SPD functionalities shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method [assignment: cryptographic key distribution method] that meets the following: [assignment: list of standards].

CS_CKM.3 Cryptographic key access

Hierarchical to:	No other components.
Dependencies:	CS_CKM.1 Cryptographic key generation

CS_CKM.4 Cryptographic key destruction

CS_CKM.3.1 The SPD functionalities shall perform [assignment: type of cryptographic key access] in accordance with a specified cryptographic key access method [assignment: cryptographic key access method] that meets the following: [assignment: list of standards].

CS_CKM.4 Cryptographic key destruction

Hierarchical to: No other components.

Dependencies: FCS_CKM.1 Cryptographic key generation

CS_CKM.4.1 The SPD functionalities shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method [assignment: cryptographic key destruction method] that meets the following: [assignment: list of standards].

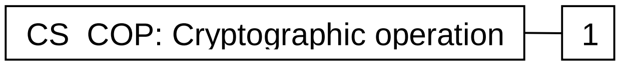
12.1.2.2 Cryptographic operation (CS_COP)

Family Behaviour

In order for a cryptographic operation to function correctly, the operation must be performed in accordance with a specified algorithm and with a cryptographic key of a specified size. This family should be included whenever there are requirements for cryptographic operations to be performed.

Typical cryptographic operations include data encryption and/or decryption, digital signature generation and/or verification, cryptographic checksum generation for integrity and/or verification of checksum, secure hash (message digest), cryptographic key encryption and/or decryption, and cryptographic key agreement.

Component leveling



Audit: CS_COP.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Minimal: Success and failure, and the type of cryptographic operation.

b) Basic: Any applicable cryptographic mode(s) of operation, subject attributes and object attributes.

CS_COP.1 Cryptographic operation

Hierarchical to: No other component.

Dependencies: CS_CKM.1 Cryptographic key generation

CS_CKM.4 Cryptographic key destruction

CS_COP.1.1 The SPD functionalities shall perform [assignment: list of cryptographic operations] in accordance with a specified cryptographic algorithm [assignment: cryptographic algorithm] and cryptographic key sizes [assignment: cryptographic key sizes] that meet the following: [assignment: list of standards]. .

12.1.3 Class IA – Identification and authentication

Families in this class address the requirements for functions to establish and verify a claimed user identity.
Identification and Authentication is required to ensure that users are associated with the proper security attributes (e.g. identity, groups, roles, security or integrity levels).
The families in this class deal with determining and verifying the identity of users, determining their authority to interact with the pSHIELD, and with the correct association of security attributes for each authorised user. Other requirements classes (e.g. User Data Protection, Security Audit) are dependent upon correct identification and authentication of users in order to be effective.

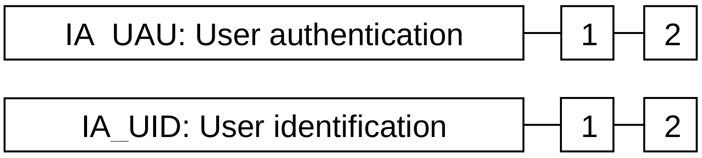


Figure 42

12.1.3.1 User authentication (IA_ UAU)

Family Behaviour

This family defines the types of user authentication mechanisms supported by the SPD functionalities. This family also defines the required attributes on which the user authentication mechanisms must be based.

Component leveling

IA_UAU: User authentication
1
2

Audit: IA_UAU.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Minimal: Unsuccessful use of the authentication mechanism;

b) Basic: All use of the authentication mechanism;

c) Detailed: All SPD functionalities mediated actions performed before authentication of the user.

Audit: IA_UAU.2

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the pSHIELD functionalities package:

a) Minimal: Unsuccessful use of the authentication mechanism;

b) Basic: All use of the authentication mechanism.

IA_UAU.1 Timing of authentication

Hierarchical to:	No other component.
Dependencies:	IA_UID.1 Timing of identification

IA_UAU.1.1	The SPD functionalities shall allow [assignment: list of SPD functionalities mediated actions] on behalf of the user to be performed before the user is authenticated.
IA_UAU.1.2	The SPD functionalities shall require each user to be successfully authenticated before allowing any other SPD functionalities mediated actions on behalf of that user.

IA_UAU.2 User authentication before any action

Hierarchical to:	IA_UAU.1 Timing of authentication.
Dependencies:	IA_UID.1 Timing of identification
IA_UAU.2.1	The SPD functionalities shall require each user to be successfully authenticated before allowing any other SPD functionalities mediated actions on behalf of that user.

12.1.3.2 User identification (IA_ UID)

Family Behaviour

This family defines the conditions under which users shall be required to identify themselves before performing any other actions that are to be mediated by the SPD functionalities and which require user identification.

Component leveling

IA UID: User identification
1
2

Audit: IA_UID.1, IA_UID.2

The following actions should be auditable if AU_GEN SPD audit data generation is included in the pSHIELD functionalities package:

a) Minimal: Unsuccessful use of the user identification mechanism, including the user identity provided;

b) Basic: All uses of the user identification mechanism, including the user identity provided.

IA_UID.1 Timing of identification

Hierarchical to:	No other component.
Dependencies:	No dependencies.
IA_UID.1.1	The SPD functionalities shall allow [assignment: list of SPD functionalities mediated actions] on behalf of the user to be performed before the user is identified.
IA_UID.1.2	The SPD functionalities shall require each user to be successfully identified before allowing any other SPD functionalities mediated actions on behalf of that user.

IA_UID.2 User identification before any action

Hierarchical to:	IA_UID.1 Timing of identification.
Dependencies:	No dependencies.
IA_UAU.2.1	The SPD functionalities shall require each user to be successfully identified before allowing any other SPD functionalities mediated actions on behalf of that user.

12.1.4 Class PT – Protection of the SPD functionalities

This class contains families of functional requirements that relate to the integrity and management of the mechanisms that constitute the SPD functionalities and to the integrity of SPD functionalities data.

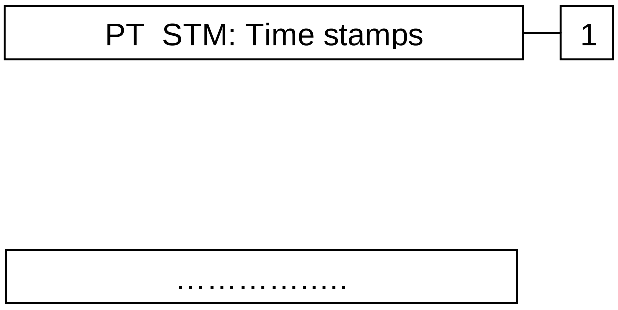


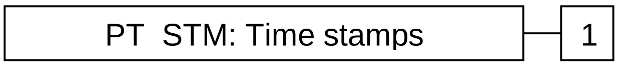
Figure 43

12.1.4.1 Time stamps (PT_STM)

Family Behaviour

This family addresses requirements for a reliable time stamp function within pSHIELD.

Component leveling



Audit: PT_STM.1

The following actions should be auditable if AU_GEN SPD audit data generation is included in the pSHIELD SPD functionalities:

a) Minimal: changes to the time;

b) Detailed: providing a timestamp.

PT_STM.1 Reliable time stamps

Hierarchical to:	No other component.
Dependencies:	No dependencies
PT_STM.1.1	The SPD functionalities shall be able to provide reliable time stamps.

(5) SPD Functional Component from catalog
(6) Name of a specific technology used to implement the SPD functionality (it could be hardware, software, firmware, algorithm…)
(7) Name and value of parameters that describe the SPD functionality
(8) Reference to life Cycle elements which have the scope to avoid the SPD functionality misuse

12.2 pSHIELD SPD life cycle support elements catalogue

12.2.1 Class LC - SPD life cycle

Life-cycle support is an aspect of establishing discipline and control in the processes of refinement of pSHIELD during its development and maintenance.

Confidence in the correspondence between pSHIELD requirements and the pSHIELD itself is greater if SPD analysis and the production of the evidence (pSHIELD and its documentation) are done on a regular basis as an integral part of the development and maintenance activities.

In the pSHIELD life-cycle it is distinguished whether it is under the responsibility of the developer or the user rather than whether it is located in the development or user environment. The point of transition is the moment where the pSHIELD is handed over to the user.

The LC class consists of five families; the following figure shows the families and the hierarchy (if existing) of components within the families.

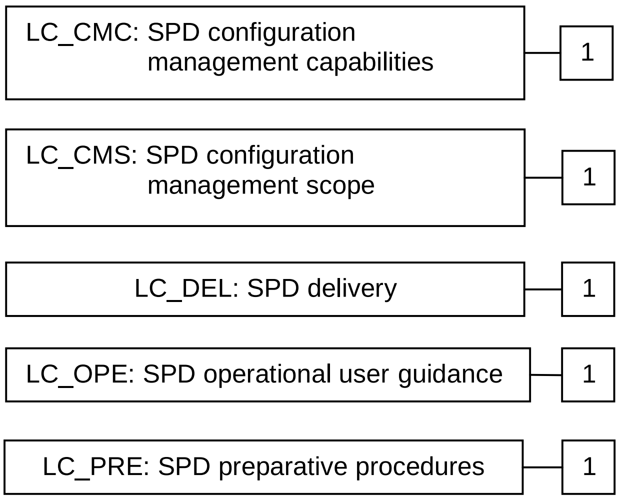

LC_CMC: SPD configuration management capabilities
1
LC_CMS: SPD configuration management scope
1
LC_DEL: SPD delivery
1
LC_OPE: SPD operational user guidance
1
LC_PRE: SPD preparative procedures
1

Figure 44

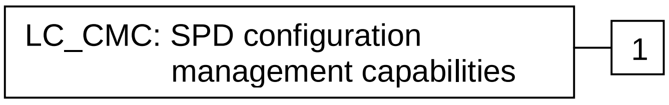
12.2.1.1 SPD configuration management capabilities (LC_CMC)

Family Behaviour

The objective of this family is to require the developer's Configuration Management (CM) system to have certain capabilities. These are meant to reduce the likelihood that accidental or unauthorised modifications of the configuration items will occur. The CM system should ensure the integrity of the pSHIELD from the early design stages through all subsequent maintenance efforts.
The objective of introducing automated CM tools is to increase the effectiveness of the CM system. While both automated and manual CM systems can be bypassed, ignored, or proven insufficient to prevent unauthorised modification, automated systems are less susceptible to human error or negligence. The objectives of this family include the following:

a) ensuring that the pSHIELD is correct and complete before it is sent to the consumer;
b) ensuring that no configuration items are missed during evaluation;
c) preventing unauthorised modification, addition, or deletion of pSHIELD configuration items.

Component leveling



LC_CMC.1 Labelling of the pSHIELD

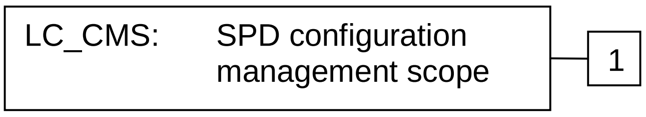
To evaluate this component the developer shall provide the pSHIELD and a reference for the pSHIELD.

Hierarchical to:	No other component.
Dependencies:	ALC_CMS.1 pSHIELD Configuration Management coverage.
LC_CMC.1.1	The pSHIELD shall be labelled with its unique reference. [1]

12.2.1.2 SPD configuration management scope (LC_CMS)

Family Behaviour

This family provides the identification of items to be included as configuration items and hence placed under the CM requirements of CM capabilities (LC_CMC).
Component leveling



LC_CMS.1 pSHIELD configuration management coverage

To evaluate this component the developer shall provide a configuration list for the pSHIELD.

Hierarchical to:	No other component.
Dependencies:	No dependencies.
LC_CMS.1.1	The configuration list shall include the following: the pSHIELD itself and all guidance and manuals [0,5]
LC_CMS.1.2	The configuration list shall uniquely identify the configuration items. [0,5]

12.2.1.3 SPD delivery (LC_DEL)

Family Behaviour

This family provides a measure of assurance that transfer of the finished pSHIELD from the development environment into the responsibility of the user is secure.
Component leveling

LC_DEL.1 Delivery procedures

To evaluate this component the developer shall provide the delivery documentation.

Hierarchical to:	No other component.
Dependencies:	No dependencies.
LC_DEL.1.1	The developer shall use the delivery procedures [0,5]
LC_DEL.1.2	The delivery documentation shall describe all procedures that are necessary to maintain security when distributing versions of the pSHIELD to the consumer. [0,5]

12.2.1.4 SPD operational user guidance (LC_OPE)

Family Behaviour

This family provides a measure of confidence that non-malicious users, administrators, application providers and other exercising the external interfaces of the pSHIELD will understand the SPD compliant operation and will use it as intended.

Component leveling

LC_OPE: SPD operational user guidance — 1

LC_OPE.1 Operational user guidance

To evaluate this component the developer shall provide the operational user guidance.

Hierarchical to:	No other component.
Dependencies:	No dependencies.
LC_OPE.1.1	The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings. [0,1]
LC_OPE.1.2	The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the pSHIELD in a secure manner. [0,2]
LC_OPE.1.3	The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate. [0,15]
LC_OPE.1.4	The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the pSHIELD. [0,2]
LC_OPE.1.5	The operational user guidance shall identify all possible modes of operation of the pSHIELD (including operation following failure or operational error), their consequences and implications for maintaining secure operation. [0,2]
LC_OPE.1.6	The operational user guidance shall, for each user role, describe the SPD measures to be followed in order to fulfill the SPD objectives for the operational environment. [0,1]
LC_OPE.1.7	The operational user guidance shall be clear and reasonable. [0,05]

12.2.1.5 SPD Preparative procedures (LC_PRE)

Family Behaviour

This family provides a measure of assurance that the pSHIELD has been received and installed in a secure manner as intended by the developer. The requirements for preparation call for a secure transition from the delivered pSHIELD to its initial operational environment. This includes investigating whether the pSHIELD can be configured or installed in a manner that is insecure but that the user of the pSHIELD would reasonably believe to be secure.

Component leveling

LC_PRE: SPD Preparative procedures — 1

LC_PRE.1 Preparative procedures

To evaluate this component the developer shall provide the pSHIELD including its preparative procedure.

Hierarchical to:	No other component.
Dependencies:	No dependencies.
LC_PRE.1.1	The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered pSHIELD in accordance with the developer's delivery procedures. [0,5]
LC_PRE.1.2	The preparative procedures shall describe all the steps necessary for secure installation of the pSHIELD and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment. [0,5]

13 Annex B

13.1 Possible pathways for SPD taxonomy

13.1.1 Dependability Concept Taxonomy

Dependability is a collection of related measures including some attributes such as reliability, availability, and safety [17] and is not a single property measure. Different authors describe dependability of a system as a set of properties or attributes. For instance, dependability concept includes some attributes such as reliability, maintainability, safety, availability, confidentiality, and integrity where the last three are shared with the security concept. Some of these attributes are quantitative (e.g., reliability and availability) and some of them are qualitative (e.g., safety). Figure 45 shows a generalized view of dependability attributes along with its threats and the means to achieve dependability.

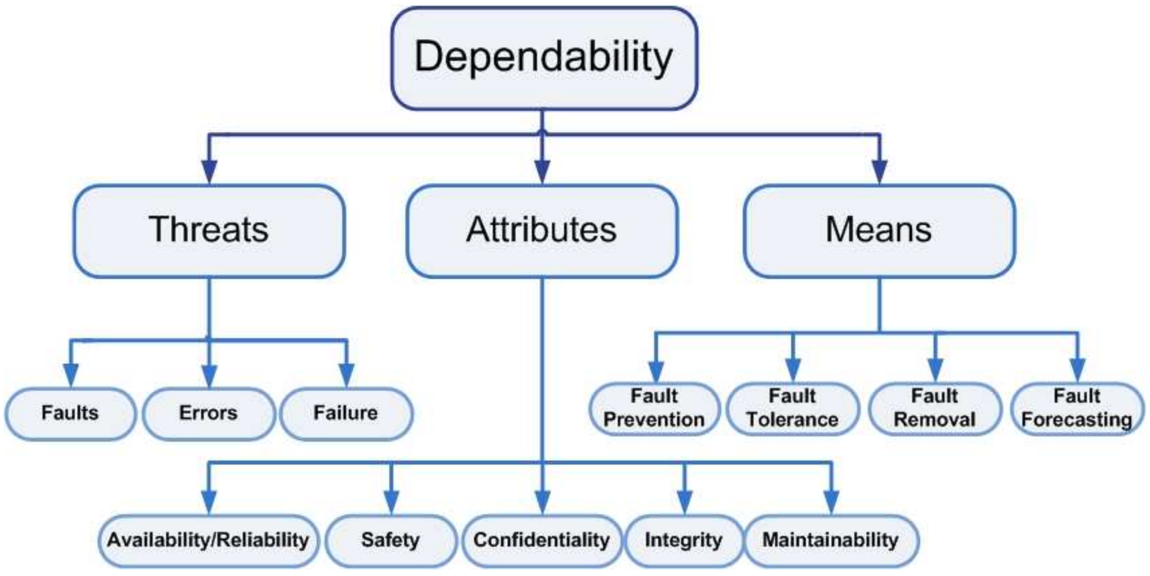


Figure 45 Dependability Concept Taxonomy

13.1.2 Fault-Tolerance Concept Taxonomy

Fault-tolerance is a property that is designed into a system to achieve some design goals. Various authors discuss different attributes of fault-tolerance. Some of the most common attributes are availability, performability, maintainability, and testability [17]. Graceful degradation is also an important feature that is closely related to performability. The concept taxonomy for fault-tolerance is shown in Figure 46.

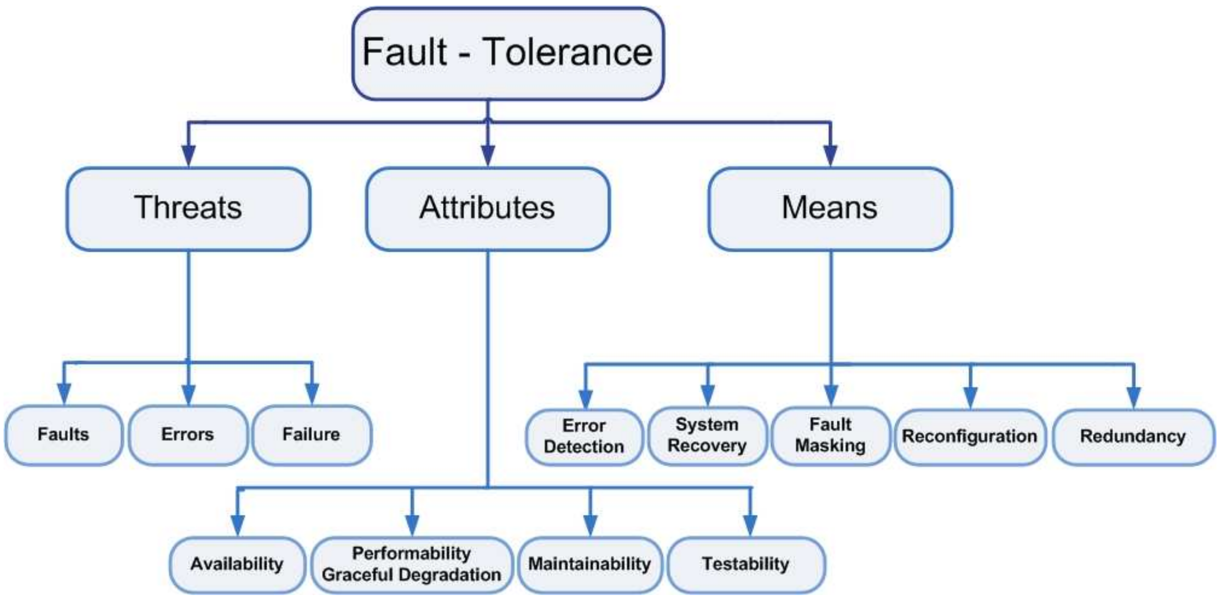


Figure 46 Fault-Tolerance Concept Taxonomy

13.1.3 Reliability Concept Taxonomy

Two attributes used in specifying fault-tolerance are also attributes of reliability. These attributes are maintainability and testability. Availability is commonly considered as an attribute of reliability. In fact, as mentioned before, it could be viewed as a special case of reliability. For instance, McCabe [25] shows that availability is needed to measure reliability since an unavailable system is not delivering the specified requirements at the first place. The reliability concept taxonomy is shown in Figure 47.

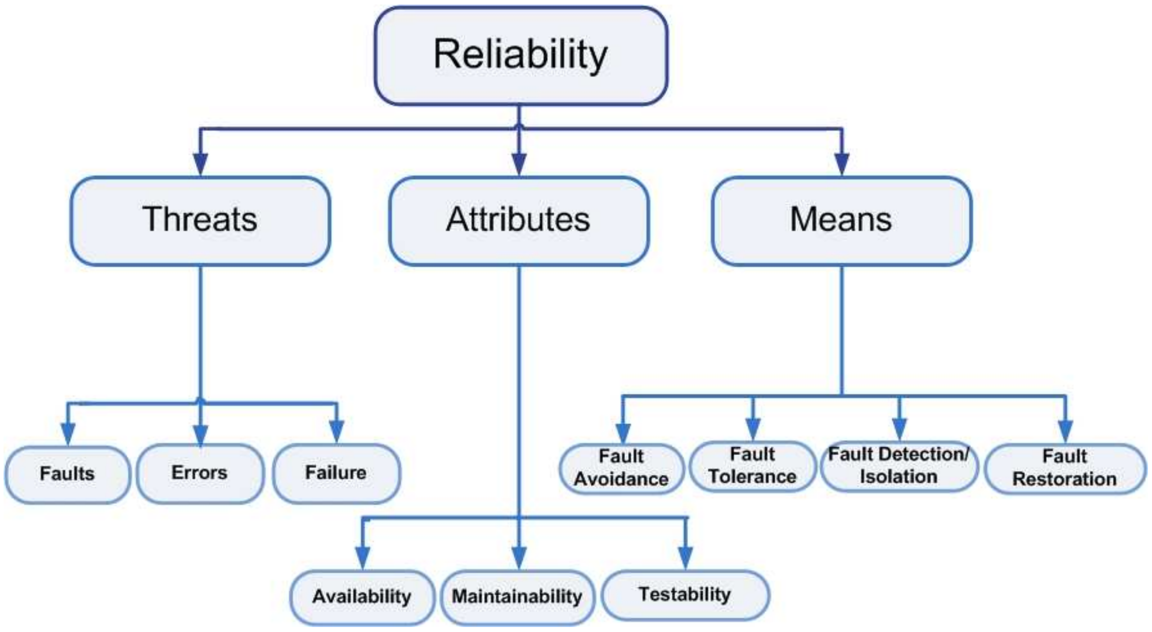


Figure 47 Reliability Concept Taxonomy

13.1.4 Security Concept Taxonomy

The security concept structure is shown in Figure 48 considering the context of security assurance and in a similar fashion as the previously presented concepts' structures and it merges all various attributes in one comparable framework. Security does share some attribute with other concepts and at the same time it exclusively encloses other attributes.

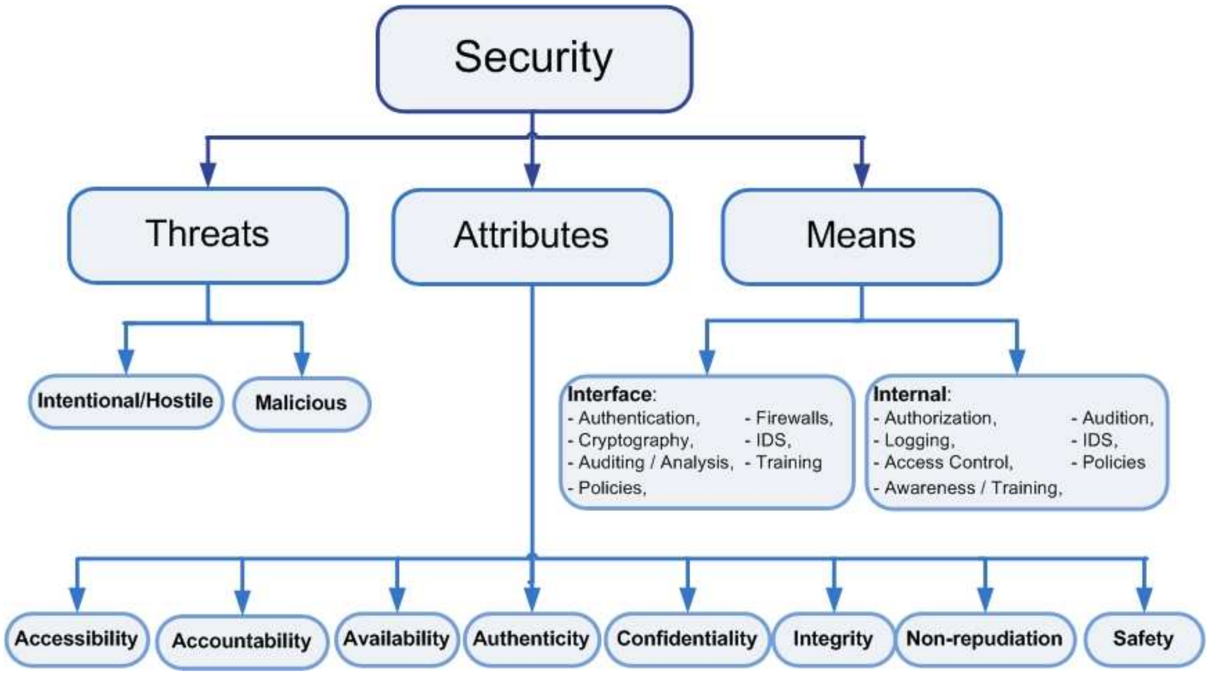


Figure 48 Security Concept Taxonomy

Not all attributes are measurable.
Table 1 in the chapter 4.2 shows a list of attributes along with an indication if they are measurable or not.

13.1.5 Survivability Concept Taxonomy

Similar to the dependability concept, the survivability has minimum levels of quality attributes such as reliability, availability, safety, fault-tolerance, security, and performability [17]. Prioritization of services in a survivable system involves balancing these attributes. The survivability concept structure is illustrated in Figure 49.

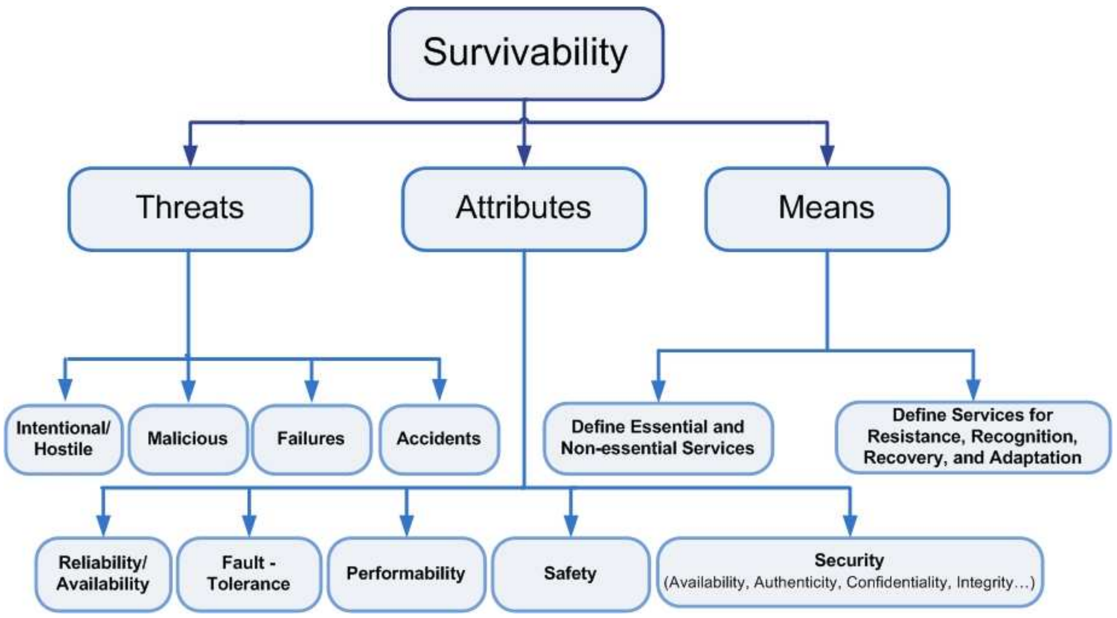

Survivability
Threats
Attributes
Means
Intentional/ Hostile
Malicious
Failures
Accidents
Define Essential and Non-essential Services
Define Services for Resistance, Recognition, Recovery, and Adaptation
Reliability/ Availability
Fault - Tolerance
Performability
Safety
Security (Availability, Authenticity, Confidentiality, Integrity…)

Figure 49 Survivability Concept Taxonomy

13.2 Definitions, requirements and Attributes taxonomies [17]

This section reviews (in alphabetical order) the definitions of the five concepts and the structural taxonomy view for each concept using the respective attributes.

- *Accessibility:* Ability to limit, control, and determine the level of access that entities have to a system and how much information they can receive.
- *Accountability*: The ability to track or audit what an individual or entity is doing on a system.
- *Authenticity:* The property of being able to verify the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system.
- *Confidentiality:* Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.
- *Fault Avoidance (Prevention)*: A technique used in an attempt to prevent the occurrence of faults.
- *Fault Containment (Isolation)*: The process of isolating a fault and preventing its effect from propagating.
- *Fault Detection*: The process of recognizing that a fault has occurred.
- *Fault Forecasting (Prediction)*: The means used to estimate the present number, the future incidence, and the likely consequence of faults.
- *Fault Location*: The process of determining where a fault has occurred so a recovery can be used.
- *Fault Masking*: The process of preventing faults in a system from introducing errors into the informational structure of that system.
- *Fault Removal*: The means used to reduce the number and severity of faults.
- *Fault Restoration (Recovery)*: The process of remaining operation or gaining operational status via reconfiguration event in the presence of faults.
- *Graceful Degradation*: The ability of a system to automatically decrease its level of performance to compensate for hardware or software faults.
- *Integrity*: Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity.
- *Maintainability*: The ease with which a system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.
- *Non-Repudiation (Non-Repudiability)*: Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the information.

- *Performability*: The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage. It is also defined as a measure of the likelihood that some subset of the functions is performed correctly.
- *Safety*: The property that a system does not fail in a manner that causes catastrophic damage during a specified period of time.
- *Testability*: The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

14 References

[1] Hu J, et al. seamless integration of dependability and security concepts in SOA: A feedback control system based framework and taxonomy. J Network Comput Appl (2011), doi:10.1016/j.jnca.2010.11.013

[2] Avizienis A, Laprie JC, Randell B, Landwhehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 2004;1(1):11-33

[3] pSHIELD – System Requirements and Specification for the pSHIELD-project – D2.1.1

[4] Jonsson E. An integrated framework for security and dependability. Proceedings of the 1998 workshop on new security paradigms. ACM Press; 1998

[5] Common Criteria for Information Technology Security Evaluation – Part 2: Security functional components – July 2009 – Version 3.1 – Revision 3 – Final – CCMB-2009-07-002

[6] Common Criteria for Information Technology Security Evaluation – Part 3: Security assurance components – July 2009 – Version 3.1 – Revision 3 – Final – CCMB-2009-07-003

[7] M. Walter and C. Trinitis, Quantifying the security of composed systems. In Proc. of PPAM-05, 2005

[8] Security Metrics, Replacing Fear, Uncertainty, and Doubt - Andrew Jaquitti

[9] E. Kushilevitz, Y. Lindell, T. Rabin, Information-Theoretically Secure Protocols and Security Under Composition, December 20, Department of Computer Science, Technion Israel Institute of Technology, 2009

[10] L. Krautsevich, F. Martinelli, A. Yautsiukhin, Formal approach to security metrics. What does “more secure” mean for you?, Copenhagen, Denmark, ECSA 2010 August 23-26, 2010

[11] V. Verendel, Quantified Security is a Weak Hypothesis, A critical survey of results and assumptions, NSPW’09, September 8–11, 2009, Oxford, United Kingdom

[12] N. Pham, L. Baud, P. Bellot, M. Riguidel, A Near Real-time System for Security Assurance Assessment, Computer Science and Networking Department (INFRES), Institute TELECOM, Telecom ParisTech (ENST), Paris, France

[13] D. Lie, M. Satyanarayanan, Quantifying the Strength of Security Systems, Department of Electrical and Computer Engineering, University of Toronto, School of Computer Science, Carnegie Mellon University

[14] W. Jansen, Directions in Security Metrics Research, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, April 2009

[15] N. Pham, M. Riguidel, Security Assurance Aggregation for IT Infrastructures, Computer Science and Networking Department, GET/Télécom Paris (ENST), Paris, France, April 12, 2007

[16] N. Pham, M. Riguidel, S. Naqvi, Security Assurances for Intelligent Complex Systems, École Nationale Supérieure des Télécommunications (ENST), Paris, France; Centre d’Excellence en Technologies de l’Information et de la Communication (CETIC), Belgium, 2004

[17] M. Al-Kuwaiti, N. Kyriakopoulos, S. Hussein, A Comparative Analysis of Network Dependability, Fault-tolerance, Reliability, Security, and Survivability, IEEE Communications Surveys & Tutorials, Vol. 11, No. 2, Second Quarter 2009

[18] M0.2: Proposal for the aggregation of SPD metrics during composition, p-SHIELD project

[19] Ingols, K., Lippmann, R., Piwowarski, K. (2006) Practical Attack Graph Generation for Network Defense, 22nd Annual Computer Security Applications Conference, Florida, 2006

[20] Jajodia, S., Nodel, S., O'Berry, B., (2003) Topological Analysis of Network Attack Vulnerability, Chapter 5, Kluwer Academic Publisher, 2003
[21] Lippmann, R. et al (2005) Evaluating and Strengthening Enterprise Network Security Using Attack Graph, Technical Report, MIT Lincoln Laboratory, Lexington, MA, 2005.
[22] Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R., Validating and Restoring Defense in Depth Using Attack Graphs, Military Communications Conference, Washington, 2006
[23] D2.2.1: Preliminary SPD Metrics Specification, pSHIELD project, 2011
[24] D2.3.1: Preliminary System Architecture Design, pSHIELD project, 2011
[25] James McCabe, Practical Computer Network Analysis and Design, Morgan Kaufmann Publishers, Inc., CA, 1998, pp. 1-9