Project no: 269317

nSHIELD

new embedded Systems arcHItecturE for multi-Layer Dependable solutions

Instrument type: Collaborative Project, JTI-CP-ARTEMIS

Priority name: Embedded Systems

D2.5: Preliminary SPD metric specification

Due date of deliverable: M12 – 2012.08.31

Actual submission date: M12 – 2012.08.31

Start date of project: 01/09/2011 Duration: 36 months

Organisation name of lead contractor for this deliverable:

Fundación Tecnalia Research & Innovation, TECNALIA

Revision [Final 1.0]

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2012)		
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**		
Eider Iturbe	Tecnalia	20/12/11	
Inaki Eguia	Tecnalia	09/01/12	
Nastja Kuzmin	THYIA	30/03/12	
Ljiljana Mijic	THYIA	30/03/12	
Konstantinos Fysarakis	TUC	30/03/12	
Konstantinos Rantos	TUC	30/03/12	
Balázs Berkes	S-LAB	30/03/12	
Mariana Esposito	ASTS	30/03/12	
Renato Baldelli	SE	30/03/12	
Harry Manifavas	TUC	30/04/12	
Alexandros Papanikolaou	TUC	30/04/12	
Alfio Pappalardo	ASTS	30/04/12	
Luigi Trono	SG	30/04/12	
Andrea Morgagni	SE	31/05/12	
Reviewed by			
Name	**Company**		
Nikolaos Priggouris	HAI	15/08/12	
Balázs Berkes	S-LAB	30/08/12	
Approved by			
Name	**Company**		
Inaki Eguia	Tecnalia		

Applicable Documents		
ID	**Document**	**Description**
[01]	TA	nSHIELD Technical Annex
[D2.1]	Preliminary System Requirements	nSHIELD Preliminary System Requirements
[D2.3]	Preliminary System Architecture Design	nSHIELD Preliminary System Architecture that should be linked to SPD metrics
[D.2.2.1]	SPD Metrics for pSHIELD	SPD metrics identification for pSHIELD project
[D2.2]	Preliminary System Requirements and Specifications	nSHIELD preliminary specifications
[D2.4]	Reference system architecture design	nSHIELD document related to main outcome: reference architecture

Modification History		
Issue	**Date**	**Description**
Draft A	20.12.2011	First ToC
Draft A	3.30.2012	Contributions from main partners for metrics identification
Draft A	4.30.2012	Contribution from main partners for metrics composability
Draft A	5.30.2012	Refinement of metrics (I)
Draft A	6.30.2012	Refinement of metrics (II)
Draft A	7.30.2012	Document refinement with new methods contributions.
Revised: HAI	8.30.2012	Revision by HAI / Nikolaos Priggouris
Revised: S-LAB	30.08.2012	Revisions by S-LAB / Balázs Berkes
Final 1.0	31.08.2012	Final version

Contents

Figures

Tables

Glossary

Please refer to the Glossary document, which is common for all the deliverables in nSHIELD.

This Page is intentionally left blank

1 Executive Summary

The main objective of this task is to define nSHIELD metrics quantitatively. A subsequent objective is to design a methodology for measuring the composition or aggregation of metrics. This document will also present a description and a taxonomy of metrics that nSHIELD will utilise. These metrics will be measured upon security, privacy and dependability parameters.

For this, defined metrics have to converge with both nSHIELD scenarios and requirements, so that a joint effort will be followed in order to make it happen. The output of this document shall be used for architecture design and for demonstrations construction.

The first two chapters of the document refer to executive summary and introduction. These topics will describe the background of security, privacy and dependability metrics and their implication in nSHIELD project.

The third chapter “Terms and definitions” is about defining SPD Metrics context. This is of paramount importance in order to explicit it and describes the correct context and interfaces towards other documents.

The fourth chapter will describe the methodology that task 2.2, “Multi-technology SPD Metrics Specification” task, will follow. This task will be oriented to define the correct procedures to define metrics quantitatively through nSHIELD’s different layers being able to sensor and compose them, so that the system can obtain composed metrics. Therefore, the system might be able to measure security by different abstraction levels and hence, isolate rapidly those malfunctions that might occur due to some vulnerability.

The fifth chapter will define security, privacy and dependability metrics through different layers. Moreover, this chapter will define the range and domain of each particular metric quantitatively. The outcome of this chapter will be a metrics taxonomy divided in different already identified layers: node, network, middleware and overlay layers.

Chapter six will describe different methods, procedures and techniques for composing and aggregating metrics. The objective is to determine systems capacity while measuring elements (nodes, devices, subsystems, etc.) in different levels of abstraction even the highest one: one value that measures the whole system in terms of security, privacy and dependability. This chapter will be prioritised and enhanced in deliverable 2.8 – Final SPD Metrics Specification.

Chapter seven will serve as a first introduction for defining a methodology for metrics deployment and a first approach for mapping SPD requirements and scenarios through SPD metrics. This chapter will be the basis for next deliverable 2.8 – Final SPD Metrics Specification.

2 Introduction

Metrics complexity varies depending on the view (static or dynamic) and the composability criteria. In nSHIELD we have identified several views for formalizing metrics design and development. Actually in pSHIELD project a Common Criteria approach was taken into account (a brief description of this methodology will be defined also in this document). Making systems reliable and efficient is usually a continuous learning and improvement process that optimizes calibration of subsystems and the modes of operation within specific scenarios and settings. This implies, especially for complex, networked systems of systems, the ability to continuously improve the involved subsystems in different configurations and situations. Adaptation is the ability of a system to modify its own behaviour in response to changes in its operating environment or to changes within the system itself [1].

Spending on IT security has increased significantly in certain sectors. One of the main goals in the nSHIELD project was the impact that it will have on the European market. The SPD metrics [2] will contribute to the success of this project and represent the central part of the nSHIELD development process that is tailored for four main scenarios: Railroad Security scenario, Voice/Facial Verification scenario, Dependable Avionics System Scenario and Social Mobility and Networking Scenario.

A widely accepted system principle is that an activity cannot be managed if it cannot be measured. *Moreover measurement is critical for the future of software security*. Metrics can be an effective tool for security and dependability but these tools should be correctly chosen. The main objective of “Preliminary SPD metrics for nSHIELD” is to identify the appropriate metrics for nSHIELD reference architecture and apply them into the 4 scenarios selected and described at the technical annex. . These tools can be used to identify the effectiveness of various components of the security programs, the security of a specific system, product or process. Metrics can also help identify the level of risk in not taking a given action, and in that way provide guidance in prioritizing corrective actions.

The nSHIELD system will contain a large number of ***heterogeneous computing and communication infrastructures and devices*** that will provide new functionalities. These devices will hold a variety of data with different security, dependability and privacy requirements that will be supported with SPD metrics requirements. This information will be used in different ways in different applications and computing contexts and, therefore, different policies (possibly contradicting) will be applied. In such settings, securing the device or the information alone or even each individual application is not sufficient, and context information should be integrated in order to be able to choose appropriate security mechanism on-the-fly. Because of their complexity, and because elements will be under the control of different owners, security mechanisms will ***need to be supervised*** (monitored) in order to identify potential threats and attacks and decide on recovery actions, when possible. Some existing approaches can provide suitable solutions to support the dynamic evolution of security policies for specific security mechanisms. However, these approaches cannot be extended to support the **dynamic evolution of general security mechanisms** (as opposed to security policies for a single mechanism). Furthermore, their results are extremely complicated to integrate, monitor and dynamically evolve as would be required by nSHIELD systems. Heterogeneity and dynamicity of complex systems make very difficult to integrate an overall overview of systems measurement and hence the system situation for one particular moment (system photograph).

The static approach focus on aspects such as how the system was built and what types of vulnerabilities it may contain whereas the dynamic methods focus more on how the system is operated and how it behaves in a certain environment. Development of dependable applications in dynamic and adaptive systems is not trivial, since both dynamism and adaptability may compromise algorithm aliveness or may complicate the design of such algorithms, especially those best suited for static systems. Strategies for building adaptable and scalable dependable services (based on “cloud systems”) will be surveyed and improved. Moreover, an efficient support for dependable applications in dynamic systems will be provided, combining three different approaches: relaxed consistency models, interconnection protocols (for supporting both consistency and multicasting) and reconciliation strategies. Last but not least, also the usage and support for integrity constraints in replicated systems will be analysed and improved for dynamic systems.

Dependable software systems, like any other software system, are subject to change during their lifetimes. Traditional approaches to bringing about change require that the system be brought offline temporarily. However, this is often undesirable due to requirements for high availability. Furthermore, dramatically reducing availability may also cause a reduction in other dependability attributes such as security, safety and reliability. This document presents an overview of a framework for managing change in dependable systems dynamically with the aim of preserving a high degree of availability. The framework allows the dynamic change process, controlled by an open set of algorithms, to be monitored and the impact of particular algorithms on a system's dependability attributes to be revealed. Novel aspects of our work include support for making an informed decision about the algorithm used to control the change process and the means to assess the dependability of change control algorithms. nSHIELD includes within its structure the overlay layer. This layer will maintain also its capacity for composing SPD Core Services depending in SPD Metrics (links between Core Services)

Links to SPD Systems Requirements

This document is not working standalone within the nSHIELD project. It will be closely linked to (above all) the requirements documents (both 2.1 and 2.2). Indeed there is a whole sub-section in the requirements document dedicated to SPD metrics (8.1.5 nSHIELD SPD Metrics) such as:

Table 1.SPD Metrics Specific Requirements table

Code	Name of the requirement
REQ_D2.1.1_2043.A	Security, Privacy and Dependability Metric
REQ_D2.1.1_2044.A	Attributes of dependability
REQ_D2.1.1_2045	A Measuring attributes of dependability
REQ_D2.1.1_2046.A	SMART Metric
REQ_D2.1.1_2047.A	Static and Dynamic Metrics
REQ_D2.1.1_2048.A	SPD level at each layer and for the overall system
REQ_D2.1.1_2049.A	Situational-aware and context-aware capabilities
REQ_D2.1.1_2050.A	Validation of performance of nSHIELD framework
REQ_D2.1.1_2051.A	Human factors
REQ_D2.1.1_2052.A	Threats, attributes and means
REQ_D2.1.1_2053.A	Security Assurance (SA) Assessment
REQ_D2.1.1_2054.A	Aggregation of SA values
REQ_D2.1.1_2055.A	Emerging attributes
REQ_D2.1.1_2056.A	Composition and decomposition
REQ_D2.1.1_2057.A	Attribute-dependency graph
REQ_D2.1.1_2058.A	Aggregation operators
REQ_D2.1.1_2059.A	Attack graphs
REQ_D2.2.SH1	Availability
REQ_D2.2.SH2	Information transmission integrity
REQ_D2.2.SH3	Information transmission confidentiality
REQ_D2.2.SH4	Peer authentication
REQ_D2.2.SH5	User authentication
REQ_D2.2.SH6	Authorisation
REQ_D2.2.SH7	Secure Node Deployment
REQ_D2.2.SH8	Secure Node Upgrade
REQ_D2.2.SH9	Denial of service
REQ_D2.2.SH10	Secure Execution
REQ_D2.2.SH11	Secure boot
REQ_D2.2.SH12	SPD level assignment
REQ_D2.2.SH13	SPD level identification
REQ_D2.2.SH14	Software failure mitigation
REQ_D2.2.SH15	Hardware failure mitigation
REQ_D2.2.SH16	Data backup
REQ_D2.2.SH17	Data storage redundancy

REQ_D2.2.SH18	Data storage integrity
REQ_D2.2.SH19	Data storage confidentiality
REQ_D2.2.SH30	Vulnerability assessment

Other requirements that implicitly also encompass metrics management:

- **REQ_D2.1.1_2025.A SPD metrics**
- **REQ_D2.1.1_2037.A Static and dynamic composability**
- **REQ_D2.1.1_2039.A Replacement of modules**

Some other links to scenario requirements are analysed in **chapter 7**. Other layer-related requirements are linked in chapter 5 for each of the SPD metrics identified.

Links to SPD Systems Architecture

This document (and above all document D2.8 SPD Metrics Specification) aims to consider the architecture document (Preliminary system architecture design) as the main document of nSHIELD project. Indeed, nSHIELD aims at designing reference architecture for inserting security, privacy and dependability in embedded systems engineering.

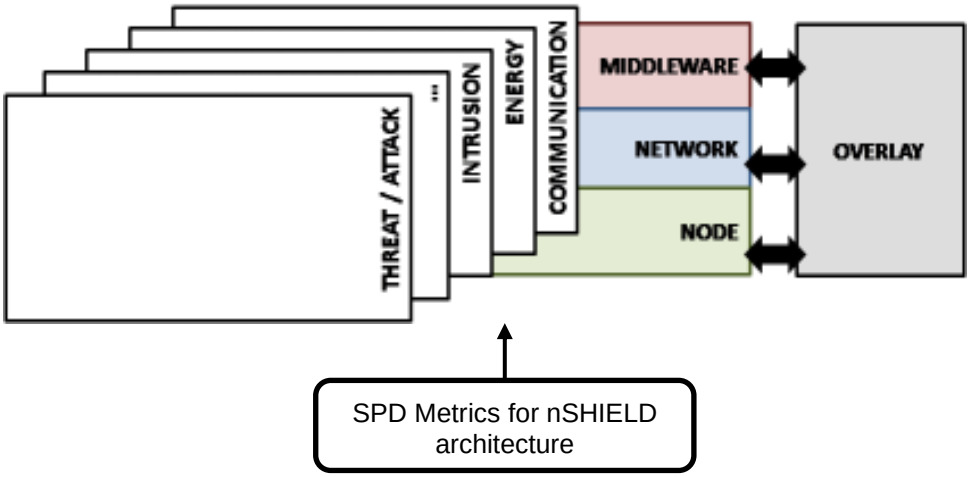


Figure 1: SPD Metrics measuring nSHIELD Architecture

SPD Metrics will be structured in this document according to this four layered architecture. It might happen that there will be some SPD metrics that might incur in more than one layer but we will locate each SPD metric in one unique layer. It is also imperative to know how the nSHIELD embedded device will gather the information for SPD metrics and measurements. Both ns-ESD and ns-SPD-ESD must develop the minimum requirements in order to guarantee SPD functions and core services. SPD metrics will be the instrument for enabling this and of delimiting the boundaries (this relation is depicted in D2.4 figure 6-5 “internal architecture of nSHIELD ESDs).

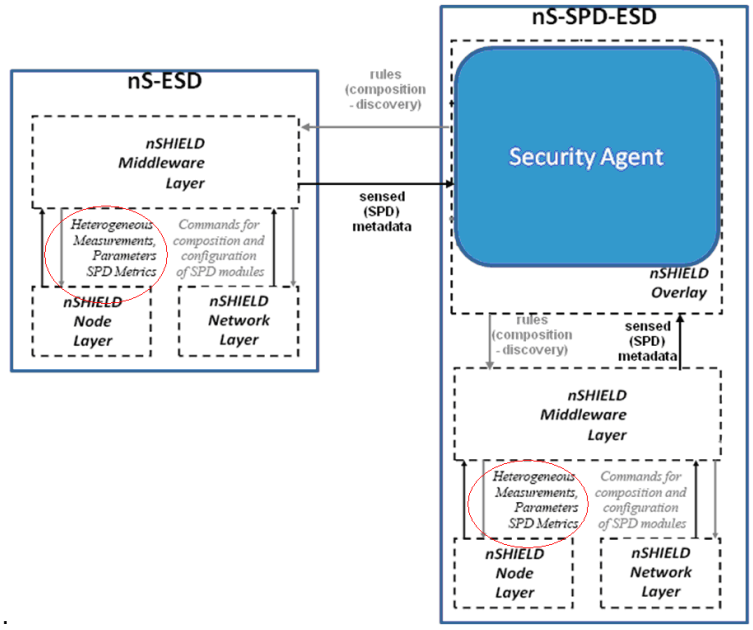


Figure 2: Relation with nS-ESD and nS-SPD-ESD

The architecture document specifies the input/output data that will be managed in nSHIELD systems regarding metrics. These metrics allow managing SPD core services in the correct way and support the innovative SPD functionalities to be implemented by the correct boundaries.

The security gateway nS-ESD-GW will be a main actor for SPD metrics. The gateway will be in contact with all L-ESD and unknown components that might be willing to interoperate with nSHIELD system. This is one of the main challenges that will have to be faced during the validation phase.

This document has structured metrics (chapter 5) according nSHIELD architecture. This is the most intrinsic link between SPD Metrics and nSHIELD architecture. However, document D2.8 (Final SPD metrics) will develop this view by the implementation of SPD metrics according to nSHIELD architectural components.

Links to scenarios

SPD metrics will be deployed into the scenarios described in the technical annex of nSHIELD. These scenarios are:

- Railroad Security scenario
- Voice/Facial Verification scenario
- Dependable Avionics System Scenario
- Social Mobility and Networking Scenario

The process is to validate and refine those metrics which are generic and specific for each scenario. This will be preliminarily identified in this document (Section 7.2: Convergence with nSHIELD Scenarios) and extensively developed in document D2.8 SPD Metrics Definition.

3 Terms and definitions

Security is the concurrent existence of:

- availability for authorized users only,
- confidentiality, and
- integrity with 'improper' taken as meaning 'unauthorized'.

Security has not been introduced as a single attribute of dependability. This is in agreement with the usual definitions of security, which view it as a composite notion, namely 'the combination of:

a) Confidentiality (the prevention of the unauthorized disclosure of information),
b) Integrity (the prevention of the unauthorized amendment or deletion of information), and
c) Availability (the prevention of the unauthorized withholding of information).

A single definition for **security** could be: the absence of unauthorized access to, or handling of, system state.

Dependability is the ability of s system/product to deliver required services during its life cycle that can justifiably be trusted. Dependability encompasses security attributes and some other such as availability and safety.

Finally **Privacy** is the ability of an individual or group to seclude them or information about themselves and thereby reveal them selectively.

Definition of Security, privacy and dependability Metrics

Security, privacy and dependability metrics is an area of IT security that has been receiving a good deal of attention lately. It is not a new topic, but one which receives focused interest sporadically. Much of what has been written about security metrics is definitional, aimed at providing guidelines for defining a security metric and specifying criteria for which to strive. However, relatively little has been reported on actual metrics that have been proven useful in practice. This is the research goal in nSHIELD to prove how the metrics are useful on the system and component levels.

Information SPD metrics are seen as an important factor in making sound decisions about various aspects of security (but also dependability), ranging from the design of security architectures and controls to the effectiveness and efficiency of security operations. **Security metrics strive to offer a quantitative and objective** basis for security assurance. The main uses fall into several broad classes:

- **Strategic support** - Assessments of security properties can be used to aid different kinds of decision making, such as program planning, resource allocation, and product and service selection.
- **Quality assurance** - Security, privacy and dependability metrics can be used during the software development lifecycle to eliminate vulnerabilities, particularly during code production, by performing functions such as measuring adherence to secure coding standards, identifying likely vulnerabilities that may exist, and tracking and analysing security flaws that are eventually discovered.
- **Tactical oversight** - Monitoring and reporting of the security, privacy and dependability status or posture of an IT system can be carried out to determine compliance with SPD requirements (e.g., policy, procedures, and regulations), gauge the effectiveness of security controls and manage risk, provide a basis for trend analysis, and identify specific areas for improvement.

SPD metrics can be categorized various ways. One simple classification is to consider metrics that denote the maturity level of processes believed to contribute to the security of a system, versus those that denote the extent to which some SPD characteristic is present in a system. The former apply to SPD processes, procedures, and training used when designing, configuring, maintaining, and operating a system. The latter apply to the security posture of a system and the inherent level of risk involved.

Note that the term "**metrics**" when used in the context of Information Technology (IT) is a bit of misnomer. It implies that traditional concepts in metrology, as used in physics and other areas of science and technology, apply equally to IT. That is not the case, however. For example, the concepts of fundamental units, scales, and uncertainty prevalent in scientific metrics have not traditionally been applied to IT or have been applied less rigorously. It is also important to recognize that compared with more mature scientific fields, IT metrology is still emerging. Many physical properties began as a qualitative comparison (e.g., "warmer" and "colder") before becoming a formally defined quantity (e.g., "temperature"), which holds promise for IT metrics in general, and IT system security metrics in particular.

While some movement toward quantitative metrics for IT system security and dependability exists, in practice, **qualitative measures that reflect reasoned estimates of security, privacy and dependability by an evaluator are the norm**. **That is, measures of information system security properties are often based on an evaluator's expertise, intuition**, and insight to induce an ordering, which is then quantified (e.g., 1=low, 2=medium, 3=high). Because of the subjectivity involved, some of the attributes sought in a good metric are not readily obtainable. For example, results in penetration testing or other methods of assessment that involve specialized skills are sometimes not repeatable, since they rely on the knowledge, talent, and experience of an individual evaluator, which can differ from other evaluators with respect to a property being measured.

It helps to understand what metrics are by drawing a distinction between metrics and measurements. **Measurements** provide single-point-in-time views of specific, discrete factors, while **metrics** are derived by comparing to a predetermined baseline two or more measurements taken over time. Measurements are generated by counting; metrics are generated from analysis. In other words, **measurements are objective raw data and metrics are either objective or subjective human interpretations of those data**.

The main difference between metrics and measurements is that metrics are measurements that are compared to a baseline. You must establish the normal operating level for each of your metrics in order to tell when something is abnormal.

You must establish a trend for your SPD status by creating a new baseline whenever you have systemic changes. If your company hires 200 more people or you implement a new security procedure or tool, the normal operation of your system will change and thus your baseline will change. At first this change is compared to the present baseline but as a new trend emerges, it then becomes your new baseline. If your new procedure or tool includes new metrics, then this is included as part of the new baseline.

Good metrics are those that are SMART[1]:

- **Specific:** well-defined, using unambiguous wording**,**
- **Measurable:** quantitative when feasible**,**
- **Attainable:** within budgetary and technical limitations**,**
- **Repeatable:** measurements from which metric is derived do not vary depending on the person taking them, and
- **Time-dependent:** takes into consideration measurements from multiple time slices**.**

Truly useful metrics indicate the degree to which security goals, such as data confidentiality, are being met, and they drive actions taken to improve an organization's overall security program. Distinguishing metrics meaningful primarily to those with direct responsibility for security management from those that speak directly to executive management interests and issues is critical to development of an effective security metrics program.

SPD metrics have evolved from early attempts to compare cryptographic protocols (e.g., the length of cryptographic keys) to modern approaches that try to evaluate and compare the security, resiliency and dependability of entire organizations. The problem with security metrics is the difficulty to make

[1] Look into smart metrics in Requirements document D2.1

predictions. Indirect metrics based on market mechanisms are a possible approach towards prediction of security.

Cryptographic Attack Metrics

When evaluating systems containing cryptographic components, the question arises how to measure the security provided by the cryptography included in the system. While measuring cryptographic security is desirable, the opposite point of view, i.e. measuring cryptographic **insecurity** using **attack metrics**, yields useful results and behaves reasonably under composition of cryptographic components. Metrics for the *insecurity* of systems are safer to use. We call such metrics **CAM [3].**

nSHIELD SPD Metrics Terminology

This taxonomy is an aggregation of definition of section 5.1.1 of Requirements document D2.1 and D2.2.

Table 2: SPD Metrics Terminology [4]

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. 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 is 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.
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.
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 behaviour perceived by the user.

4 Methodology

4.1 Formalization of metrics insertion

One of the main challenges of nSHIELD will be the **formalization [5] of SPD metrics insertion**. nSHIELD aims to insert metrics formally and hence have a formal estimation of nSHIELD derived systems measurements. In pSHIELD[2] a first attempt was carried out which takes inspiration from Common Criteria [6], [7] (CC) standard, to ensure a consistently measured and expressed as a cardinal number SPD metric for embedded systems [8]. This concept is integrated with the notion of a system's attack surface and we present a systematic way to measure it. Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage. Hence the "smaller" the attack surface, the more secure the system. This method is more detailed in chapter 6.

There will be new approaches (new methods) in nSHIELD that partners are analysing as formal methods for security systematisation within embedded systems engineering. The aim of this document is to analyse those which are the most interesting for nSHIELD taking into account:

- Adaptation (static and dynamic view)
- Quantitative approach
- Composability
- Formalisation

These new methods are considered in chapter 6 (Quantitative solutions for Metrics composition) and will be developed within this document and the final SPD Metrics Specifications (D2.8).

4.2 nSHIELD approach: quantitative solution

nSHIELD metrics gathering approach will follow a quantitative focus. The aim of this document is to find correct metrics and to be able to quantify them. This document will provide a metric template for metrics identification and gathering process. Table 3 provides a template which is used throughout this section for the definition of the metrics to present metrics in a formalised and consistent manner.

Table 3: Metric Template

Metric	*//Name together with an optional code name*
Description	*//Provide a short description about the metric*
System component(s)	*//It should define the system component(s) where the metric is applicable, i.e. [node | network | middleware | overlay | all], where all also denotes a single nSHIELD node. The field also has the meaning of the data source.*
Formula	*//Type of value and how it is calculated*
Target	*//It defines the target value if available*
Frequency	*//How often should measurements be collected or value checked*
Applicability	*//It should define whether it is a global metric or bound to a specific scenario*
Requirements	*//It lists the requirements that this metric satisfies*
CC Functional requirements /Classes	*//It does the mapping between this metric and the applicable CC functional requirements to facilitate assessment against protection profiles' requirements.*

[2] http://www.pshield.eu/index.php?option=com_docman&task=doc_download&gid=241&Itemid=37

This metrics table will be a very useful table for managing and deploying metrics.

In this table above, a link to requirements can be found. The row “Requirements” is the specific link to a set of requirements identified in document D2.1 Preliminary System Requirements and D2.2 Systems Requirements and Specification. The aim is to include the requirements that the SPD Metrics is satisfying or accomplishing for its fulfilment.

There is another specific row related to Common Criteria requirements. This is important issue in order to maintain the formalisation focus of nSHIELD project as it was done in pSHIELD project. However, this formalisation comes since the first step of the metric identification.

In chapter 7, a mapping between scenario requirements and SPD metrics will be performed.

4.3 SPD Metrics gathering process

The SPD metrics gathering process will be done through a systematised and formalised manner. This includes observing the requirements description in order to evaluate **what** to measure. This includes also a look to other documents such as document of SPD Cores Services from WP5 and related node and network technical documents from WP3 and WP4.

Once we selected the *what* criteria we need to go through the **how** challenge. For doing this, we considered in the previous section the quantitative approach [9]. We need to have a quantitative solution as a final result of the measurement.

The final step is the **integration** to nSHIELD reference architecture. A tireless joint task will be performed with other WPs. Indeed, nSHIELD SPD metrics will be much linked to the SPD Taxonomy and semantic framework that will be managed by the overlay layer. The overlay layer will also have to receive some results from SPD Metrics module in order to dynamically take decisions towards the correct status of the nSHIELD system. Thus the proposed process for SPD metrics collection in nSHIELD system is as follows:

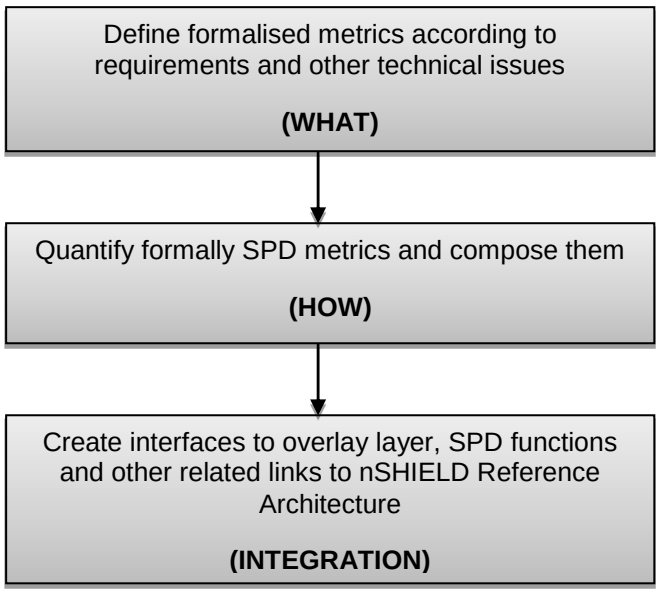


Figure 3: Process that will be exploited into 7 steps methodology for SPD Metrics deployment

The process above is defining a generic overview of how to collect the SPD metrics for nSHIELD. The overall SPD metrics deployment phases are seen in chapter 7.1: SPD Metrics Design steps

5 Metrics in nSHIELD multi-layer scheme (Convergence with Requirements)

This section will specify security, dependability and privacy metrics for each nSHIELD functional layer. Each nSHIELD functional layer has its own specificities, therefore it is important to collect all the **parameters in this multi-layer approach. This will be linked to the requirements collected in documents D2.1 and D2.2**. This link is necessary in order to maintain a formal specification and facilitate traceability during nSHIELD architecture design. In order to maintain this link a reference has been inserted as one row of the table template for requirements.

5.1 Node Layer

This section presents metrics applicable to an nSHIELD node as opposed to a specific layer.

Table 4: nSHIELD node availability

Metric	Availability
Description	Measures the availability of the system through an operational period. Although availability of the whole system can be a very complex function which depends on the corresponding deployed components and redundancy measures, the approach taken for a single nSHIELD node considers it as a system with high-availability requirements which is not only subject to system failures but also to attacks and environmental conditions such as network signal strengths. Therefore it is not considering only MTTF (Mean Time To Failure) and MTTR (Mean Time To Recover) periods but a more generic time operation formula.
System component(s)	All. Whole system. Node layer as basic point.
Formula	(Time the node has been available (Uptime) / Total operational period (Uptime + Downtime)) *100
Target	Set by the system owner/administrator
Frequency	Measurements can be combined with resource access requests or can be the result of a sequence of beacons sent to the node to check availability.
Applicability	Global
Requirements	REQ_D2.1.1_2168.A Availability REQ_D2.2_SH1
CC Functional requirements /Classes	FPT_ITA.1 FRU[3].Resource utilisation

5.1.1 Security

Table 5: Code Execution

Metric	Code Execution
Description	Verification that only authorized code (booting, kernel, application) runs on the system

[3] Abbreviations from CC.

System component(s)	Node
Formula	Boolean value depending on whether code execution checks are present on the node.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per application execution.
Applicability	Global
Requirements	REQ_D2.1.1_2130.A Node – Code execution REQ_D2.2_SH7/8 REQ_D2.2_ND01
CC Functional requirements /Classes	FMT.Security management

Table 6: Data Freshness[4]

Metric	Data Freshness
Description	Data Freshness checks to avoid replay attacks
System component(s)	Node
Formula	Ensure that received data follows a monotonically increasing timeline and that the data freshness of each packet lies within acceptable, application-specific limits.
Target	True (1)
Frequency	Per packet received unless scenario requires otherwise (e.g. every nth packet when in low-power operation states)/ Scenario specific.
Applicability	Global
Requirements	REQ_D2.1.1_2131.A Node – Data Freshness REQ_D2.2_ND02
CC Functional requirements /Classes	FMT.Security management

Table 7: Digital Signatures

Metric	Digital Signatures
Description	Ability of the node to verify digital signatures even in cases where a trusted third party is not available
System component(s)	Node
Formula	Boolean value depending on whether digital signature verification functionality is present on node.
Target	True (1)
Frequency	Scenario specific
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2132/2134.A Node – Digital Signatures

[4] This metric was initially placed in a different section which included metrics for the whole nSHIELD system, as opposed to a specific layer. Deliverable 2.8 should group metrics that are tackling different layers and schemes.

	REQ_D2.2_SH18/19 (Enabler) REQ_D2.2_ND03
CC Functional requirements /Classes	FDP. User data protection, FCS. Cryptographic support, FIA.Identification and authentication

Table 8: Policy Updates

Metric	Policy Updates
Description	Security policy updates only accept by authorized (white-listed) sources
System component(s)	Node
Formula	Boolean value depending on whether policy-update requests are cross-checked against white-list of authorized entities.
Target	True (1)
Frequency	Per policy update request
Applicability	Global
Requirements	REQ_D2.1.1_2134/2136.A Node – Policy updates REQ_D2.2_ND04
CC Functional requirements /Classes	FMT. Security management, FPR. Privacy, FRU. Resource utilisation

Table 9: TPM Low Power mode

Metric	TPM Low Power mode
Description	Node able to enter into low power state without compromising its security.
System component(s)	Node
Formula	Boolean value depending on whether a secure low-power feature is implemented on the node.
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Global
Requirements	REQ_D2.1.1_2139.A Node – TPM Low Power mode REQ_D2.2_ND05
CC Functional requirements /Classes	FRU. Resource utilisation, FTA. TOE access, FMT. Security management

Table 10: TPM Context-based encryption keys

Metric	TPM Context-based encryption keys
Description	Extension of TPM key generation functionality to include generators and parameters dependent on the context available.
System component(s)	Node
Formula	Boolean value depending on whether context-based encryption keys functionality is implemented on the node.
Target	True (1)

Frequency	Upon initialization / network enrolment.
Applicability	Application specific
Requirements	REQ_D2.1.1_2140.A Node – TPM Context-based encryption keys
CC Functional requirements /Classes	FCS.Cryptographic support, FMT. Security management

Table 11: Key parameterization

Metric	Key parameterization
Description	Key size parameterization options mapping application requirements (SHORT, MEDIUM or LONG-TERM security)
System component(s)	Node
Formula	Boolean value depending on whether key size parameterization is supported by the node.
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Global
Requirements	REQ_D2.1.1_2142.A Node – Key parameterisation
CC Functional requirements /Classes	FCS.Cryptographic support, FMT. Security management

Table 12: Secure key distribution

Metric	Secure key distribution
Description	Secure low-cost key distribution mechanism with parameters (e.g. algorithm, key length, usage etc.) defined by the security policy requirements, taking into consideration any restrictions that participating parties impose
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per key initialization/re-keying.
Applicability	Global
Requirements	REQ_D2.1.1_2143.A Node – Secure key distribution mechanism REQ_D2.2_SH18/19 (Enabler)
CC Functional requirements /Classes	FCS.Cryptographic support, FMT. Security management, FCO. Communication

Table 13: Third-party key management

Metric	Third-party key management
Description	Support for third-party key management services
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	Scenario specific

Frequency	Upon initialization / network enrolment. Per key initialization/re-keying.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2144.A Node – Third-party key management REQ_D2.2_SH18/19 (Enabler) REQ_D2.2_ND06
CC Functional requirements /Classes	FCS.Cryptographic support, FMT. Security management, FCO. Communication, FTP. Trusted Path/Channels

Table 14: Security context establishment

Metric	Security context establishment
Description	Node should allow security context establishment and sharing, allowing more efficient keys or key material to be exchanged, thereby increasing the overall performance and security of subsequent communications
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	Scenario specific
Frequency	Upon initialization / network enrolment. Per key initialization/re-keying.
Applicability	Global
Requirements	REQ_D2.1.1_2146.A Node – Security context establishment REQ_D2.2_SH12/13 (Enabler) REQ_D2.2_ND07
CC Functional requirements /Classes	FMT. Security management, FDP. User data protection, FPR. Privacy, FIA. Identification and authentication

Table 15: Physical/tamper resilience

Metric	Physical/tamper resilience
Description	Resilience to tampering, micro-probing and reverse-engineering. Aim of this specific metric is to “detect” and report the presence of tamper-resistance provisions on the node.
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Global
Requirements	REQ_D2.1.1_2148.A Node – Physical/tamper resilience REQ_D2.2_ND11
CC Functional requirements /Classes	FPR. Privacy, FRU. Resource utilisation, FTA. TOE access

5.1.2 Dependability

Table 16: TPM Remote Attestation

Metric	TPM Remote Attestation
Description	TPM Remote Attestation functionality to ensure integrity of node prior to resource allocation
System component(s)	Node
Formula	Boolean value depending on the result of the execution of Remote Attestation.
Target	True (1) – Successful TPM_RA integrity check
Frequency	Per resource allocation.
Applicability	Global
Requirements	REQ_D2.1.1_2153/54.A Node – TPM Remote attestation REQ_D2.2_ND08
CC Functional requirements /Classes	FRU. Resource utilisation, FPT. Protection of the TSF, FIA. Identification and authentication

Table 17: Virtualization

Metric	Virtualization
Description	Virtualized hardware redundancy mechanism
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Scenario specific. Power node.
Requirements	REQ_D2.1.1_2155.A Node – TPM Virtualization REQ_D2.2_ND09
CC Functional requirements /Classes	FRU. Resource utilisation, FMT. Security management

Table 18: Runtime Reconfiguration

Metric	Runtime Reconfiguration
Description	Runtime modification of the device's functionality during both normal operation or fault conditions, through either hardware or software changes
System component(s)	Node
Formula	Boolean value depending on node's runtime reconfiguration support
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Global
Requirements	REQ_D2.1.1_2157.A Node –Runtime reconfiguration REQ_D2.2_ND21 (Dynamic behaviour)

CC Functional requirements /Classes	FAU. Security audit, FIA. Identification and authentication, FMT. Security management

Table 19: Alternative power supply sources

Metric	Alternative power supply sources
Description	Support for alternative power modes, depending on the specific application and environmental conditions (e.g. vibration generator, micro-solar cells)
System component(s)	Node
Formula	Boolean value depending on node's support for alternative power modes. Cannot be represented by a single measurement, if there are more than two alternative power modes.
Target	Application specific
Frequency	Upon initialization / network enrolment. When device reports low power levels
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2158.A Node – Alternative power supply sources REQ_D2.2_ND27 REQ_D2.2_ND28/29
CC Functional requirements /Classes	FRU. Resource utilisation

Table 20: Dependable key distribution mechanisms

Metric	Dependable key distribution mechanisms
Description	Support of secure and dependable low-cost key distribution mechanisms for initialization or re-keying
System component(s)	Node
Formula	Cannot be represented by a single measurement.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per key initialization / re-keying.
Applicability	Global
Requirements	REQ_D2.1.1_2160.A Node – Dependable authentic key distribution mechanisms REQ_D2.2_SH18/19 (Enabler) REQ_D2.2_ND10
CC Functional requirements /Classes	FCS. Cryptographic support, FTP. Trusted Path/Channels, FMT. Security management

5.1.3 Privacy

Table 21: Privacy-centric physical/tamper resilience

Metric	Privacy-centric physical/tamper resilience
Description	No compromise in the privacy of contained sensitive information in the case of a malicious user gaining physical possession of the device.
System component(s)	Node
Formula	Cannot be represented by a single measurement, until the tamper resilience mechanisms are finalized.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per triggering of fault/critical condition.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2161.A Node – Dependable authentic key distribution mechanisms REQ_D2.2_ND11
CC Functional requirements /Classes	FPR. Privacy

Table 22: ECC Authentication

Metric	ECC Authentication
Description	Optimized hardware implementation of an ECC-based public-key authentication algorithm
System component(s)	Node
Formula	Boolean value depending on node's ECC authentication support.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per resource allocation.
Applicability	Global
Requirements	REQ_D2.1.1_2162.A Node – ECC Authentication REQ_D2.2_N4/5 REQ_D2.2_ND12
CC Functional requirements /Classes	FCS. Cryptographic support, FIA. Identification and authentication

Table 23: Storage of private information

Metric	Storage of private information
Description	Provisions to ensure the long term storage of private information, guaranteeing confidentiality of said information even under fault conditions
System component(s)	Node
Formula	Boolean value depending on node's provisions for long-term/fault-condition secure storage of private information.
Target	True (1)

Frequency	Upon initialization / network enrolment. Per triggering of fault/critical condition.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2164.A Node – Storage of private information REQ_D2.1.1_2166.A µNode / Wearable personal node - Storage of private information REQ_D2.2_ND14
CC Functional requirements /Classes	FPR. Privacy, FCS. Cryptographic support, FIA. Identification and authentication

Table 24: Anonymity and location privacy

Metric	Anonymity and location privacy
Description	Privacy-aware management of location and other sensitive personal information, utilizing secure storage and sanitization mechanisms to be applied to such information prior to transmission
System component(s)	Node
Formula	Boolean value depending on node's enforcement of anonymity and location privacy mechanisms prior to transmission of sensitive personal information.
Target	True (1), scenario specific
Frequency	Upon initialization / network enrolment. Per exchange of sensitive information.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2163.A Node – Location Privacy REQ_D2.1.1_2165.A µNode / Wearable personal node – Anonymity & Location Privacy REQ_D2.2_ND15 (Wearable personal node – anonymity and location)
CC Functional requirements /Classes	FPR. Privacy

Table 25: Privacy across trust domains

Metric	Privacy across trust domains
Description	Security token exchange to enable the issuance and dissemination of credentials within different trust domains.
System component(s)	Node
Formula	Boolean value depending on support for security tokens to be exchanged within different trust domains
Target	Scenario specific
Frequency	Upon initialization / network enrolment. Per exchange of sensitive information. Per communication across different trust domain.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2167.A Node – Privacy in different trust domains REQ_D2.2_ND17 (Privacy in trust domains)

CC Functional requirements /Classes	FPR. Privacy, FTP. Trusted Path/Channels

5.1.4 Composability

Table 26: eNetwork/Hybrid Network Compatibility

Metric	eNetwork/Hybrid Network Compatibility
Description	Support for switching between infrastructure-centric and ad-hoc networks on demand, in order to adapt continually to changes in the physical environment
System component(s)	Node
Formula	Boolean value depending on support of both infrastructure-centric and ad-hoc networks.
Target	Scenario specific
Frequency	Upon initialization / network enrolment.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2169.A Node – eNetwork / Hybrid network compatibility REQ_D2.2_ND18
CC Functional requirements /Classes	-

Table 27: Flexible key distribution mechanisms

Metric	Flexible key distribution mechanisms
Description	Flexible and secure low-cost key distribution mechanism for initialization or re-keying, allowing the distribution of authentic public keys via insecure channels, either according to a pre-defined schedule or ad-hoc, while adhering to policy requirements as well as requirements participating parties impose
System component(s)	Node
Formula	Cannot be represented by a single measurement, until the key distribution mechanisms are finalized.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per key-initialization / re-keying.
Applicability	Global
Requirements	REQ_D2.1.1_2170.A Node – Flexible key distribution mechanisms REQ_D2.2_SH18/19 (Enabler) REQ_D2.2_ND19
CC Functional requirements /Classes	FCS. Cryptographic support, FTP. Trusted Path/Channels, FMT. Security management

Table 28: Conflict resolution between policy domains

Metric	Conflict resolution between policy domains

Description	In case of nodes' communication between different policy domains, mechanisms facilitate the communication and resolve this conflict with the minimum processing and communication overhead
System component(s)	Node
Formula	Cannot be represented by a single measurement
Target	True (1)
Frequency	Upon initialization / network enrolment.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2171.A Node - Conflict resolution between policy domains REQ_D2.2_ND20
CC Functional requirements /Classes	FMT. Security Management, FCO. Communication

Table 29: Dynamic security behaviour

Metric	Dynamic security behaviour
Description	Support for security behaviour changes based on the dynamic change of policy requirements without requiring to reprogram or shut down the node
System component(s)	Node
Formula	Cannot be represented by a single measurement, until the dynamic security mechanisms are finalized.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per policy change.
Applicability	Global
Requirements	REQ_D2.1.1_2172.A Node – Dynamic security behaviour REQ_D2.2_N12/13 (Enabler) REQ_D2.2_ND21 (Dynamic behaviour)
CC Functional requirements /Classes	FMT. Security Management, FCO. Communication

5.1.5 Performance

Table 30: Lightweight embedded operating system

Metric	Lightweight embedded operating system
Description	Low resource requirements Operating System. Refers to the presence/installation of a lightweight O/S running on the device.
System component(s)	Node
Formula	Value based on the OS installed on the device.
Target	Scenario specific. Cannot be finalised, until the Operating Systems are finalized.
Frequency	Upon initialization / network enrolment.
Applicability	Scenario specific

Requirements	REQ_D2.1.1_2173.A Node – Lightweight embedded operating system REQ_D2.2_ND22
CC Functional requirements /Classes	FMT. Security Management, FCO. Communication

5.1.6 Interfaces

Table 31: SCA protection based on EM emissions

Metric	SCA protection based on EM emissions
Description	Inclusion of interfaces to monitor the nodes own EM emissions and modify its functionality accordingly, to protect its assets against SCA.
System component(s)	Node
Formula	Boolean value based on whether node features SCA protection/EM monitoring interfaces and whether these are functioning properly.
Target	True (1)
Frequency	Constantly on. If device enters low power state, SCA protection is disabled.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2177.A Node - SCA protection based on EM emissions REQ_D2.2_ND25
CC Functional requirements /Classes	FCO. Communication, FMT. Security Management

Table 32: Location Awareness

Metric	Location Awareness
Description	Inclusion of interfaces that enable location-based functionality
System component(s)	Node
Formula	Boolean value based on whether node features location awareness.
Target	Scenario specific
Frequency	Upon initialization / network enrolment.
Applicability	Scenario specific
Requirements	REQ_D2.1.1_2178.A Node – Location awareness REQ_D2.2_ND26
CC Functional requirements /Classes	FCO. Communication, FMT. Security Management

Table 33: Situation and context awareness

Metric	Situation and context awareness[5]
Description	Inclusion of interfaces that enable the provision of situational-aware and context-aware services and SPD.
System component(s)	Node
Formula	Cannot be represented by a single measurement, until the situation and context mechanisms are finalized.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per service request / resource allocation.
Applicability	Global
Requirements	REQ_D2.1.1_2179.A Node - Situational-aware and context-aware SPD REQ_D2.2_ND07
CC Functional requirements /Classes	FCO. Communication, FMT. Security Management

5.2 Network layer

5.2.1 Security

Table 34: Metric – Data exchange confidentiality

Metric	Data Confidentiality
Description	Checks whether data exchange confidentiality is enforced and not only provided, so that to be in line with the CC requirements.
System component(s)	Network
Formula	Boolean value depending on whether encryption functionality is enforced
Target	True (1)
Frequency	Per data transmission
Applicability	Global
Requirements	REQ_D2.1.1_2181.A Confidentiality REQ_D2.2_SH3 (Enabler) REQ_D2.2_NW01
CC Functional requirements /Classes	FDP.UCT.1

[5] Indirectly the presence of context awareness is of course linked to security (e.g. enforces the use of stronger crypto mechanisms in untrusted areas) and one of the important features we try to implement in nSHIELD. In fact, not all requirements are measurable and this metric is one example.

Table 35: Metric – Data exchange integrity

Metric	Data Integrity
Description	Checks whether data exchange integrity is enforced
System component(s)	Network
Formula	Boolean value depending on whether integrity functionality is applied
Target	True (1)
Frequency	Per packet
Applicability	Global
Requirements	REQ_D2.1.1_2182.A Integrity REQ_D2.2_SH2 (Enabler) REQ_D2.2_NW02
CC Functional requirements /Classes	FDP.UCT.1

Table 36: Secure routing

Metric	Secure routing
Description	Checks whether secure routing is in operation (as required by the system) and verifies routing information on each packet.
System component(s)	Network
Formula	Boolean value based on whether the network features secure routing and whether this is functioning properly.
Target	True (1)
Frequency	Per packet
Applicability	Global
Requirements	REQ_D2.1.1_2185.A Secure Routing REQ_D2.1.1_2189.A Reputation-Based Secure Routing REQ_D2.1.1_2190.A Reputation-Based Intrusion Detection REQ_D2.2_NW03 (Secure Routing) REQ_D2.2_NW11 (Reputation-based secure routing)
CC Functional requirements /Classes	FCO. Communication, FMT. Security Management

5.2.2 Dependability

Table 37: Dependable authentic key distribution mechanisms

Metric	Dependable authentic key distribution mechanisms
Description	Support of secure and dependable low-cost key distribution mechanisms for initialization or re-keying
System component(s)	Network
Formula	Cannot be represented by a single measurement.
Target	True (1)
Frequency	Upon initialization / network enrolment. Per key initialization / re-keying.
Applicability	Global

Requirements	REQ_D2.1.1_2194.A Dependable Authentic Key Distribution Mechanisms REQ_D2.2_SH2/3 (Enabler) REQ_D2.2_NW15
CC Functional requirements /Classes	FCS. Cryptographic support, FTP. Trusted Path/Channels, FMT. Security management

5.2.3 Privacy

Table 38: k-anonymity

Metric	k-anonymity
Description	The goal of this metric is to make a user's identify information in-distinguish with other k-1 users
System component(s)	ALL
Formula	An integer value greater than one that represents the number of users
Target	Level of location privacy and anonymity
Frequency	When location-based services must be provided
Applicability	The metric is used in scenario 4 – Social Mobility and Networking Scenario
Requirements	REQ_D2.1.1_2197.A Anonymity REQ_D2.2_NW17 (Anonymity)
CC Functional requirements /Classes	FPR. Privacy, FIA. Identification and authentication

Table 39: L-diversity

Metric	L-diversity
Description	This metric is used in cooperation with k-anonymity. The metric adds another dimension L to incorporate with k. L is a set of distinct locations. Thus, we consider the k users in L distinct locations.
System component(s)	ALL
Formula	An integer value greater than one that represents the number of distinct locations
Target	Level of location privacy and anonymity
Frequency	When location-based services must be provided
Applicability	The metric is used in scenario 4 – Social Mobility and Networking Scenario
Requirements	REQ_D2.1.1_2197.A Anonymity REQ_D2.2_NW17 (Anonymity)
CC Functional requirements /Classes	FPR. Privacy, FIA. Identification and authentication

Table 40: Time of confusion

Metric	Time of confusion
Description	Measures how long it will take until an attacker will become confused about a subjects track as the subject seeks to obfuscate its location by omitting measured samples
System component(s)	ALL
Formula	Time duration
Target	Level of location privacy and anonymity
Frequency	When location-based services must be provided
Applicability	The metric is used in scenario 4 – Social Mobility and Networking Scenario
Requirements	REQ_D2.1.1_2197.A Anonymity REQ_D2.2_NW17 (Anonymity) REQ_D2.2_NW18 (Location) REQ_D2.2_ND15 (Anonymity and Location privacy)
CC Functional requirements /Classes	FMT. Security management

5.2.4 Performance

Table 41: Metric – Network Delay (per node)

Metric	Network Delay (per nSHIELD node)
Description	This is a performance metric used for measuring the delay induced by a node in retransmitting incoming data. This metric is important in cases where the node acts as a relaying node, where retransmission of incoming packets without extra delays is important for the overall network performance. Message relaying increases network latency and therefore network topology, e.g. a mesh or cluster-tree topology of a WSN, and the number of nodes significantly affects it.
System component(s)	Network
Formula	Can calculate average delay ratio, for nodes A, B and C that are directly linked between them, thus forming a triangle [10]. This parameter is defined as the ratio of the transmission delay in the cooperative path (through the relay node), over the transmission delay in the direct (non-cooperative) path. The delay ratio sensed by node C is defined as: $DR_{acb} = (D_{ac} + D_{bc}) / D_{ab}$ where, the D_{ab}, D_{ac} and D_{bc} are the corresponding delay between the three nodes as calculated from the inverse of the current transmit rate ($D_i=1/R_i$).
Target	Scenario specific
Frequency	Per message transmission
Applicability	Global
Requirements	REQ_D2.1.1_21102.A Low Network Delay REQ_D2.2_SH1 (Enabler) REQ_D2.2_NW20 (Low Network delay)
CC Functional requirements /Classes	FMT. Security management

Table 42: Metric – Network Latency

Metric	Network Latency
Description	This is a performance metric used for measuring the network latency which shall be kept low. This metric applies to the nSHIELD architecture as opposed to a single node where relaying delays due to routing decisions (static or dynamic) are induced. It quantifies the mean time needed for reaching a node. Network latency is typically the average delay from the time of data production to the time of its reception by the peer [11], [12]. There are several approaches in measuring latency performance in wireless sensor networks [13], including those that consider first order statistics, i.e. mean and variance, and schemes on probabilistic analysis of delay, which can be utilized in the design process.
System component(s)	Network
Formula	Scenario specific
Target	Scenario specific
Frequency	Scenario specific
Applicability	Global
Requirements	REQ_D2.1.1_21102.A Low Network Delay REQ_D2.2_SH1 (Enabler) REQ_D2.2_NW20
CC Functional requirements /Classes	FMT. Security management

Table 43: Metric – Network Information Capacity

Metric	Information Capacity
Description	This is a performance metric used for measuring the network's capacity, which shall be large enough to allow the necessary traffic to go through. As a rule of thumb, at normal operation, the traffic should be about 60-70% of the network's capacity, so as to avoid bottlenecks when there will be traffic peaks. Measuring capacity is quite a complex task [14]. However, the approach of "IP-type-P Path Capacity" seems to be rather straightforward to implement in the nSHIELD network.
System component(s)	Network
Formula	C(P,T,I) = min {1..n} {C(Ln,T,I)}, P: Packet type, T: Time, I: Interval, Ln: Link AP: RFC 5136,
Target	
Frequency	Scenario specific (It depends on whether the network topology is dynamic or not).
Applicability	Global
Requirements	REQ_D2.1.1_21103.A Information Capacity REQ_D2.2_SH1 (Enabler) REQ_D2.2_NW21

CC Functional requirements /Classes	FMT. Security management

5.3 Middleware layer

Table 44: Metric – Discovery frequency

Metric	Discovery frequency
Description	Amount of discovery events per protocol per unit time
System component(s)	Middleware (DoS protection core SPD service / Secure Discovery core SPD service)
Formula	N_{disc} / sec for each discovery interface available (per protocol)
Target	
Frequency	Per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_21107.A Discovery REQ_D2.2_MW1
CC Functional requirements /Classes	-

Table 45: Metric – Discovery statistics

Metric	Discovery service statistics
Description	Discovery service related availability statistics: • Queue length (momentarily count of outstanding discovery requests) • Thread count (momentarily count of threads serving discovery requests)\ • Used memory (amount of heap memory reserved by the service • Wait time (time from arrival to processing, averaged for all requests per interval) • Processing time (time from start of processing to sending response, averaged for all requests per interval)
System component(s)	Middleware (DoS protection core SPD service / Secure Discovery core SPD service)
Formula	Five integer fields: • QL: queue length [-] • TC: thread count [-] • UM: used memory [bytes] • WT: wait time [us] = sum(wait time(i))/count(i) for i in requests • PT: processing time [us] = sum(processing time(i))/count(i) for i in requests
Target	
Frequency	Per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_21107.A Discovery REQ_D2.2_MW1

CC Functional requirements /Classes	-

Table 46: Metric – Composition statistics

Metric	Composition service statistics
Description	Composition service related availability statistics: • Queue length (momentarily count of outstanding composition requests) • Thread count (momentarily count of threads serving composition requests) • Used memory (amount of heap memory reserved by the service • Wait time (time from arrival to sending, averaged for all requests per interval) • Response time (time from start of sending to receiving response, averaged for all requests per interval)
System component(s)	Middleware (DoS protection core SPD service / Trusted Composition core SPD service)
Formula	Five integer fields • QL: queue length [-] • TC: thread count [-] • UM: used memory [bytes] • WT: wait time [us] = sum(wait time(i))/count(i) for i in requests • RT: response time [us] = sum(response time(i))/count(i) for i in requests
Target	
Frequency	Per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_21108.A (Composition) REQ_D2.2_MW2 (Composition) REQ_D2.2_MW10 (Trusted composition)
CC Functional requirements /Classes	-

Table 47: Metric – Failed discovery requests

Metric	Failed discovery requests
Description	Accumulated count of failed discovery requests per protocol
System component(s)	Middleware (DoS protection core SPD service / Secure Discovery core SPD service)
Formula	N_{fail} (accumulated per protocol)
Target	
Frequency	Updated per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_1506.A N-Repudiation for discovery REQ_D2.2_SH14 (Enabler)

	REQ_D2.2_MW11 (non repudiation service discovery)
CC Functional requirements /Classes	

Table 48: Metric – Rejected discovery requests

Metric	Rejected discovery requests
Description	Accumulated count of rejected discovery requests per protocol
System component(s)	Middleware (DoS protection core SPD service / Secure Discovery core SPD service)
Formula	N_{reject} (accumulated per protocol)
Target	
Frequency	Updated per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_1507.A N-Repudiation for discovery REQ_D2.2_MW12 (non repudiation service discovery)
CC Functional requirements /Classes	-

Table 49: Metric – Composition response time

Metric	Composition response time
Description	Average delay of response from an entity when answering composition requests
System component(s)	Middleware (DoS protection core SPD service / Trusted Composition core SPD service)
Formula	$Sum(T_{comp}) / N_{comp}$ for each composition interface available (per protocol)
Target	
Frequency	Per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_21108.A (Composition) REQ_D2.2_MW12 (No response for composition)
CC Functional requirements /Classes	-

Table 50: Metric – Failed composition requests

Metric	Failed composition requests
Description	Accumulated count of failed composition requests per protocol
System component(s)	Middleware (DoS protection core SPD service / Trusted Composition core SPD service)
Formula	N_{fail} (accumulated per protocol)
Target	

Frequency	Updated per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_1507.A N-Repudiation for composition REQ_D2.2_MW12 (No response for composition)
CC Functional requirements /Classes	

Table 51: Metric – Blacklist/whitelist additions and removals

Metric	Blacklist/whitelist additions and removals for DOS protection
Description	Accumulated count of the following attributes of the DOS / DDOS protection scheme: • Blacklist additions • Blacklist removals • Whitelist additions • Whitelist removals
System component(s)	Middleware (DoS protection core SPD service)
Formula	Four integer fields for the following (without dimensions – count of events): • Blacklist additions • Blacklist removals • Whitelist additions • Whitelist removals
Target	0 / 0 / 0 / 0
Frequency	Updated per time interval (settable by configuration)
Applicability	Global
Requirements	REQ_D2.1.1_1508.A Access control for lists REQ_D2.2_MW13 (Access control for lists) REQ_D2.2_MW14 (Access control for components)
CC Functional requirements /Classes	

5.4 Overlay Layer

5.4.1 Reputation Metrics

Table 52: Average reputation

Metric	Average reputation
Description	The mean value of all reputations of agents from a specific type. It is the correct rating that an agent should receive for each transaction
System component(s)	Node, Network
Formula	$\frac{\sum_{i=1}^{n} R_i}{n}$

	Where n is the number of agents, Ri the reputation of agent i
Target	The average reputation
Frequency	It is updated when there are changes in individual agent reputations and is distributed to the whole system
Applicability	Global metric
Requirements	REQ_D2.1.1_2149.A Node – Dynamic security behaviour REQ_D2.1.1_2189.A Reputation-Based Secure Routing REQ_D2.1.1_2190.A Reputation-Based Intrusion Detection REQ_D2.2_NW12 (Reputation-Based Intrusion Detection) REQ_D2.2_NW11 (Reputation-Based secure routing)
CC Functional requirements /Classes	FTP. Trusted Path/Channels

Table 53: Reputation bias

Metric	Reputation bias
Description	The difference between the average reputation and the actual ratings
System component(s)	Network
Formula	| Average_reputation – Ri | Where Ri the reputation of agent i
Target	The reputation bias
Frequency	It is updated when there are changes in the agent's reputation or in the average reputation
Applicability	Global metric
Requirements	REQ_D2.1.1_2149.A Node – Dynamic security behaviour REQ_D2.1.1_2189.A Reputation-Based Secure Routing REQ_D2.1.1_2190.A Reputation-Based Intrusion Detection REQ_D2.2_NW12 (Reputation-Based Intrusion Detection) REQ_D2.2_NW11 (Reputation-Based secure routing) REQ_D2.2_ND21 (Dynamic security behaviour)
CC Functional requirements /Classes	FTP. Trusted Path/Channels

Table 54: Transaction rate

Metric	Transaction rate[6]
Description	Is the portion of transactions completed successfully compared with all initiated transactions
System component(s)	Network
Formula	$\frac{\text{successful completed transactions}}{\text{initiated transactions}}$

[6] A "transaction" can be whatever function the reputation-based mechanisms are using to calculate reputation levels. It might be a packet exchange or something more complex.

Target	The percentage of successful transactions
Frequency	It is updated when a new transaction is initiated
Applicability	Global metric
Requirements	REQ_D2.1.1_2149.A Node – Dynamic security behaviour REQ_D2.1.1_2189.A Reputation-Based Secure Routing REQ_D2.1.1_2190.A Reputation-Based Intrusion Detection REQ_D2.2_NW12 (Reputation-Based Intrusion Detection) REQ_D2.2_NW11 (Reputation-Based secure routing) REQ_D2.2_ND21 (Dynamic security behaviour)
CC Functional requirements /Classes	FTP. Trusted Path/Channels

Table 55: The profit of each agent type

Metric	The profit of each agent[7] type
Description	The types of agents include: • Evil or malicious • Disturbing • Selfish • Honest It is the profit that each agent type can gain according to its behaviour. The profit is higher if an agent cheats on another agent.
System component(s)	Network
Formula	$$p = \frac{1}{o+2} v$$ Where p is the profit of a transaction, v is the transaction value and $o \in [-1, \dots, 1]$ is the real outcome of this transaction.
Target	The percentage of successful transactions
Frequency	It can be calculated in advance for every pair of agent & transaction type
Applicability	Global metric
Requirements	REQ_D2.1.1_2149.A Node – Dynamic security behaviour REQ_D2.1.1_2189.A Reputation-Based Secure Routing REQ_D2.1.1_2190.A Reputation-Based Intrusion Detection REQ_D2.2_NW12 (Reputation-Based Intrusion Detection) REQ_D2.2_NW11 (Reputation-Based secure routing) REQ_D2.2_ND21 (Dynamic security behaviour)
CC Functional requirements /Classes	FTP. Trusted Path/Channels

[7] Refers to entities is evaluated by the reputation-based system. If it is set to evaluate an agent, then it refers to agents. If it evaluates network nodes, it refers to nodes.

5.4.2 Miscellaneous Metrics

Table 56: Uptime

Metric	Uptime
Description	This metric provides uninterrupted system availability
System component(s)	Overlay
Formula	Uptime = (Monitoring_Time - DownTime) / Monitoring_Time
Target	
Frequency	
Applicability	Global
Requirements	REQ_D2.1.1_2068/2085.A Nodes Availability REQ_D2.1.1_20122.A Availability and dependability REQ_D2.1.1_2183.A Availability for network REQ_D2.1.1_21117.A Availability for middleware REQ_D2.1.1_21135.A Availability for agents in overlay REQ_D2.2_SH1
CC Functional requirements /Classes	FRU. Resource utilisation

Table 57: Attack surface

Metric	Attack Surface
Description	Used to count the number of data inputs to an overlay node
System component(s)	Overlay
Formula	Attack surface = Number of incoming connection / Number of monitoring nodes
Target	
Frequency	
Applicability	Global
Requirements	REQ_D2.1.1_20190.A Overlay – Overlay interfaces REQ_D2.1.1_20191.A Overlay – Internal interfaces REQ_D2.1.1_20192.A Overlay – Overlay/Node external interfaces REQ_D2.2_ND29 (Interface to node) REQ_D2.2_NW22 (Interfaces to SCA) REQ_D2.2_NW23 (Interfaces to key exchange)
CC Functional requirements /Classes	FCO. Communication.

Table 58: Failed authentication

Metric	Failed authentication
Description	Measure the percentage of fail authentication attempts
System component(s)	Overlay, Middleware
Formula	Failed authentication = Authentication failures / Total number of authentications

Target	
Frequency	
Applicability	Global
Requirements	REQ_D2.1.1_21132.A Overlay data authenticity REQ_D2.1.1_21131.A Overlay data integrity REQ_D2.1.1_21129.A Agent Authenticity REQ_D2.1.1_21128.A Identity Validity REQ_D2.2_SH14/15 REQ_D2.2_SH4/5
CC Functional requirements /Classes	FIA. Identification and authentication, FCS. Cryptographic support

Table 59: Detection accuracy

Metric	Detection Accuracy
Description	Measure of the detection accuracy
System component(s)	Overlay
Formula	Accuracy = Undetected attacks / Total number of detected attacks
Target	
Frequency	
Applicability	Global
Requirements	REQ_D2.1.1_21136.A Intrusion Detection and Prevention REQ_D2.2_NW12 (Intrusion detection) REQ_D2.2_NW05 (Self management and self coordination)
CC Functional requirements /Classes	FMT. Security Management, FCO. Communication

Table 60: Total attack impact

Metric	Total attack impact
Description	Measure attacks' impact
System component(s)	Overlay
Formula	Total Attack Impact = Nodes attacked / Total number of nodes
Target	
Frequency	
Applicability	Global
Requirements	REQ_D2.1.1_21136.A Intrusion Detection and Prevention REQ_D2.2_NW12 (Intrusion detection) REQ_D2.2_NW05 (Self management and self coordination)
CC Functional requirements /Classes	FMT. Security Management, FCO. Communication

6 Quantitative solutions for Metrics composition

6.1 Introduction

Nowadays it is necessary to measure security efficiency in different embedded systems. That is why a study in metrics allowing quantifying concepts such as security, privacy and dependability is required. These metrics are called “SPD metrics” and, thanks to them, some attributes such as safety, integrity, reliability, availability, confidentiality and maintainability may be assessed.

In order to achieve this aim, SPD metrics must be applied in all nSHIELD system functional layers:

- Node
- Network
- Middleware
- Overlay

This way, the optimization of all resources involved in the system will be taken into account. It is necessary to add SPD metrics to each of the layers so that the result of the general system is a combination of the results obtained for each layer.

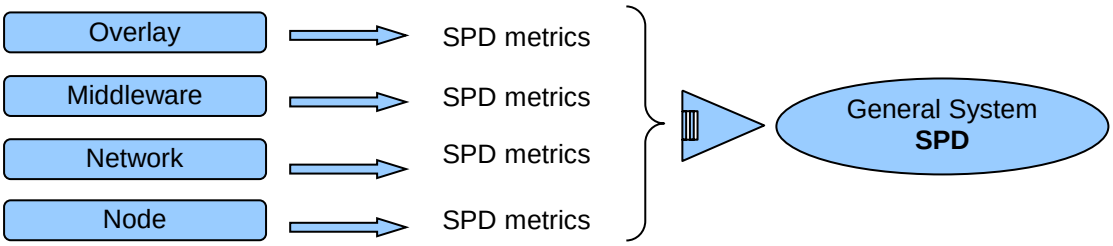


Figure 4: From layers to General System’s SPD Metric achievement

There are different methodologies to quantify all the concepts mentioned above. Below five possible methodologies for quantifying the SPD of composed SPD functions are listed:

- Method 1: SPD formalised through Common Criteria and composed by medieval castle.
- Method 2: A System for Security Assurance Assessment
- Method 3: Formalisation through metrics patterns (Derived from method 2)
- Method 4: SPD based on human immunologic system.
- Method 5: SPD on intrusion tolerant systems.

6.2 Solutions of metrics compositions for SHIELD

Composition of metrics is of paramount importance and one of the main goals of nSHIELD. The reason is that the Human being tends to have a single point(s) in order to know if one system is working correctly or not. In the following section, 5 methods for composing metrics will be analysed.

6.2.1 Method: SPD formalised through Common Criteria and composed by medieval castle Metrics

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 fulfil 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 nSHIELD technical annex in addition to this approach we consider the attack surface metric concept. In particular we define:

- the SPD metric of each nSHIELD component implementing a SPD function as it's described in CC vulnerability assessment taking into account that we know that many attacks on a system take place either by sending data from the system's operating environment into the system (e.g., buffer over-flow exploitation) or by receiving data from the system (e.g., Simulink attacks). In both these types of attacks, an attacker connects to a system using the system's channels (e.g., sockets), invokes the system's methods (e.g., API), and sends (receives) data items (e.g., input strings) into (from) the system. An attacker can also send (receive) data indirectly into (from) a system by using persistent data items (e.g., files); an attacker can send data into a system by writing to a file that the system later reads. Hence an attacker uses a system's methods, channels, and data items present in the environment to attack the system. We collectively refer to the methods, channels, and data items as the resources and thus define a system's attack surface in terms of the system's resources. In this way we derive a SPD metric following the approach of a rigorous international standard.
- The formal description of nSHIELD SPD functions according to CC security functional requirements which are a standard way of expressing the SPD functional requirements.

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 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 [15]. We start from the point that the nSHIELD 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 modelling method has 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 nSHIELD 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.

6.2.1.1 Applying CC concepts

The formal description of nSHIELD SPD functions is according to CC security functional requirements which are a standard way of expressing the SPD functional requirements.

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.

6.2.1.1.1 Class structure

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

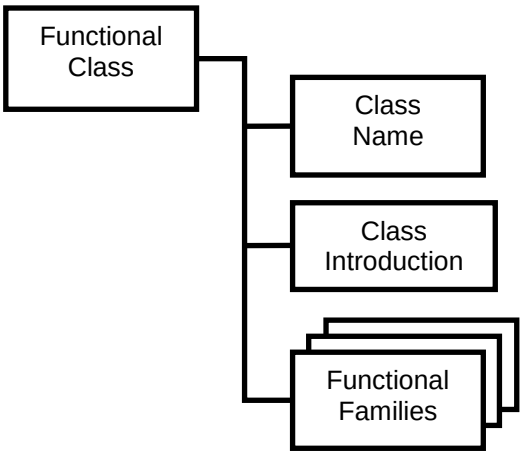


Figure 5: Functional Class Structure

6.2.1.1.2 Family structure

Next figure illustrates the functional family structure in diagrammatic form.

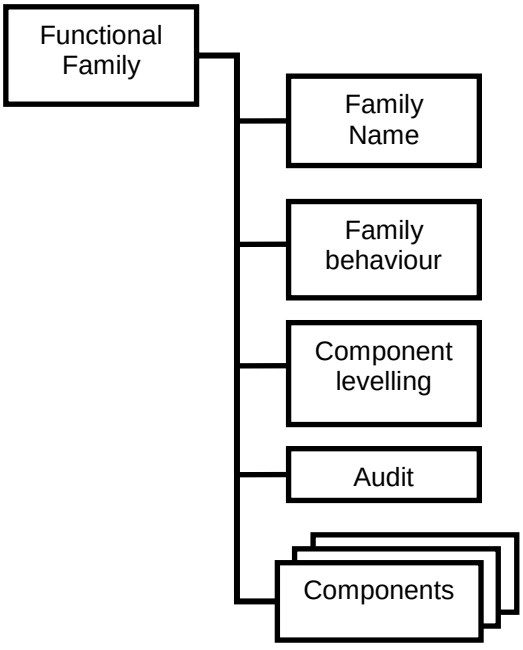


Figure 6: Family Structure

6.2.1.1.3 Component structure

Next figure illustrates the functional component structure.

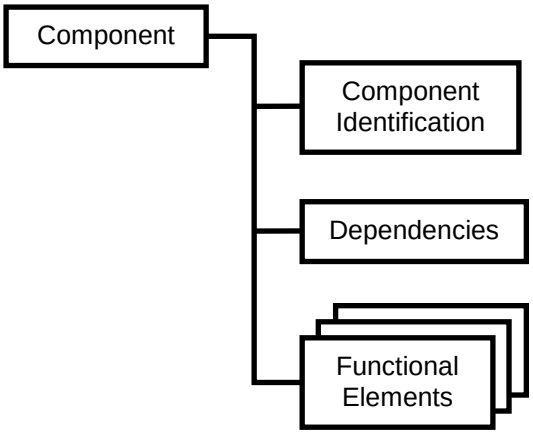


Figure 7: Component Structure

6.2.1.1.4 SPD function components catalogue example

6.2.1.1.4.1 Class AU - SPD audit

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

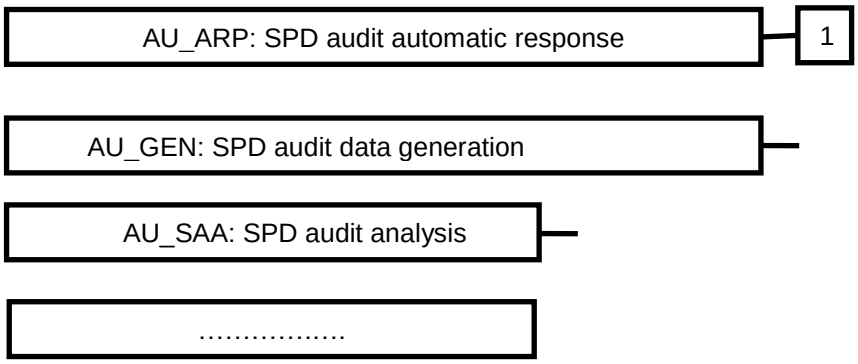


Figure 8: SPD function components catalogue example (Class AU)

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 levelling

Audit: AU_ARP.1

The following actions should be auditable if AU_GEN SPD audit data generation functionality is included in the nSHIELD 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 nSHIELD shall take [assignment: list of actions] upon detection of a potential [assignment: Security, Privacy Dependability] violation.

6.2.1.2 Solving Composability problem

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 the following section we solve how to compose metrics for measuring whole systems [16].

6.2.1.2.1 SPD for medieval castle

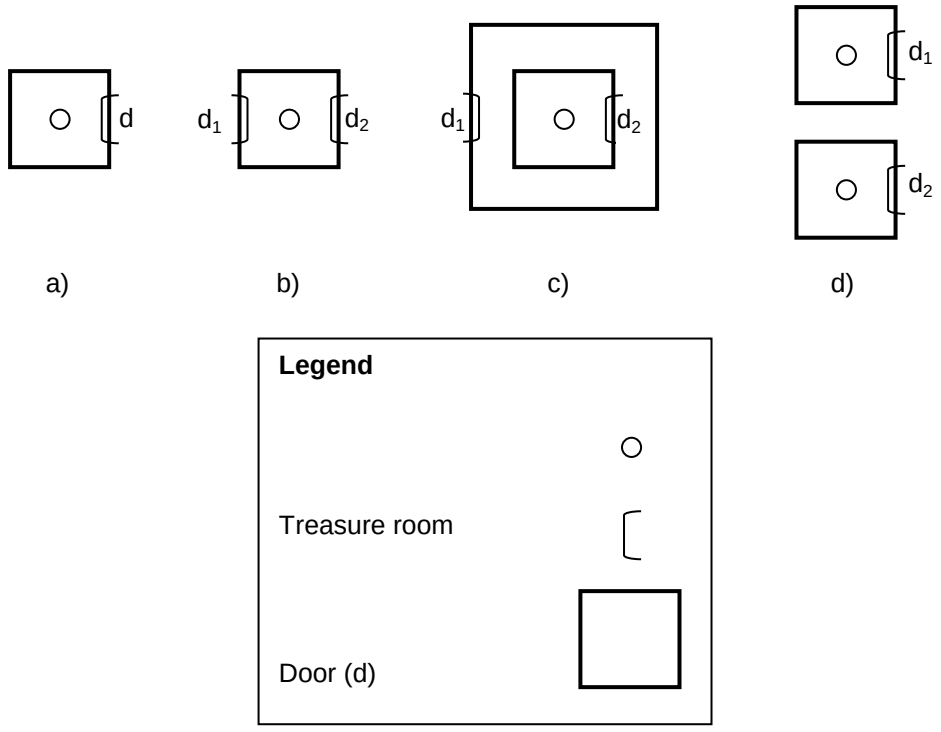


Figure 9: Medieval castle terminology

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 9: Medieval castle terminology, 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 d1 and d2, and we assume that d1 is weaker than d2. 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 d1 (we assume that d1 and d2 are totally independent so castle's SPD measure will be equal to the SPD measure of d1).

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 d1 is weaker than d2, the castle's SPD is at least as good as the SPD of d2.

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 [d1; d2].

6.2.1.2.1.1 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 doors8): d1, d2, . . . , dn
- a function: d : Mn → 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.

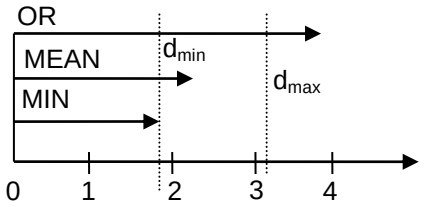


Figure 10: Operators and values

Figure 10 shows three operations for the unbounded continuous case with interval [0; +∞). If we call dmin the smaller operand and dmax the larger operand, we can classify these operations into:

- MIN-Operations, if d = dmin
- MEAN-Operations, if dmin ≤ d ≤ dmax
- OR-Operations, if dmax ≤ d ≤ +∞

where both operands are unbounded deterministic variables and values are computed by the measure of SPD metrics of basic components.

MIN operation

A MIN-Operation should be used for a system which resembles castle b) from Figure 9: Medieval castle terminology) In this case, the defenders have to defend both doors. Thus, the system is as weak as dmin.

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

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

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

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

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

OR operation

For systems corresponding to the situation named castle c) in Figure 9, OR-operations can be used.

To defend the castle, either dmin or dmax has to be defended. A corresponding system is at least as strong as dmax.

This kind of system models the concept of “defence in depth” where it is used multiple defence mechanisms in layers across the system to protect the assets. We use multiple defences 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 ORn, 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 ORn 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 9: Medieval castle terminology) as depicted in the next figure just to underline that the system is protected by the redundancy of the same SPD function.

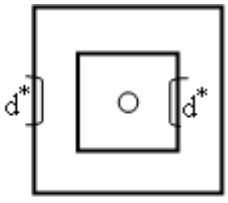


Figure 11: Redundant SPD function castle

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 9. 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 [dmin; dmax].

Therefore, it is straightforward to apply a mathematical mean operation to model these kinds of systems, which has the same property (dmin ≤ d ≤ dmax).

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 dmin and dmax. 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 Mp 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, Mp 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. $p \rightarrow +\infty$, 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 $p \rightarrow 0$, Mp 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. $p \rightarrow -\infty$, 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. In the presence of n elements as castle d) in Figure 9, where we can consider with d1, d2,…dn 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 d1, …, dn.

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 Mp 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.

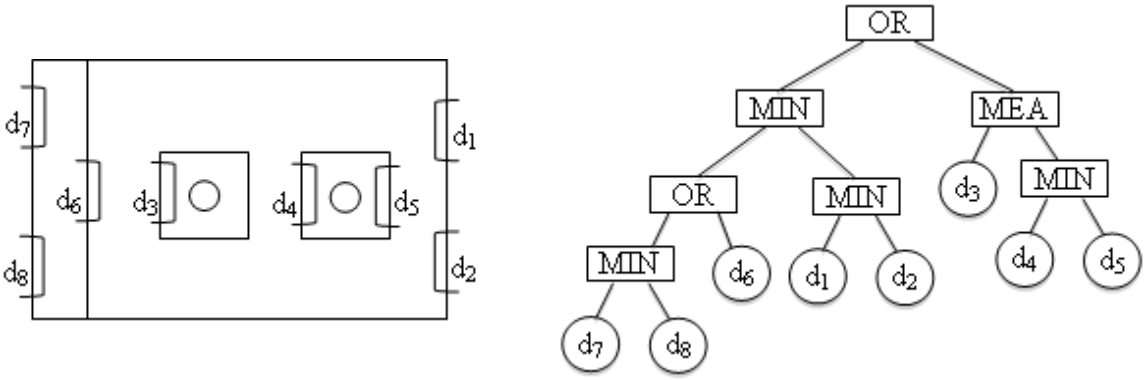
In the extreme case, i.e. $p \rightarrow +\infty$, the attackers always choose the strong door and thus:

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

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

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

6.2.1.2.1.2 SPD measure of systems with n-SPD functions



a) b)

Figure 12: Tree approach for its composition

If we consider a castle with eight doors and two treasure rooms, as shown in Figure 12 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 12 b). This representation can be easily implemented by the proposed ontology for the SPD functions modelling.

Figure 12 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:

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

6.2.1.2.1.3 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 dLC. The total value is:

$$dTOT = d * dLC$$

Because the dLC 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.

6.2.1.2.1.4 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

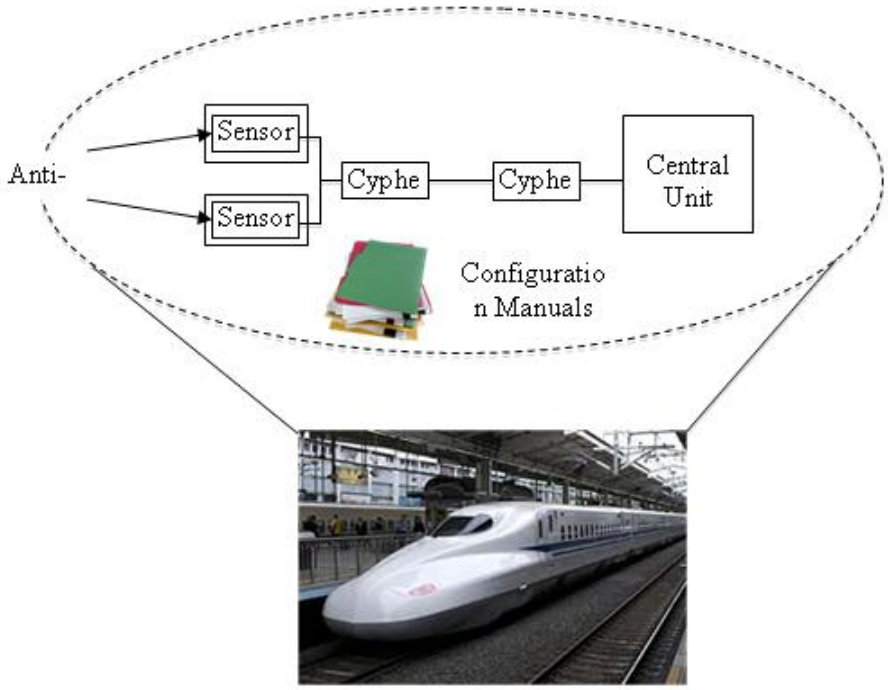


Figure 13: Real example for medieval castle approach

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 modelled as shown in the following graphic:

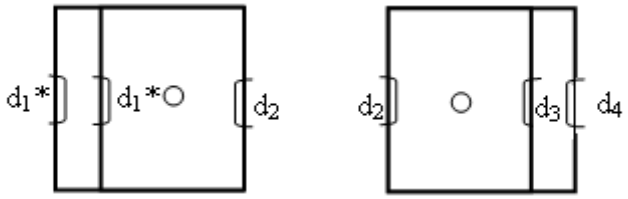


Figure 14: Representative graphic for described scenario

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:

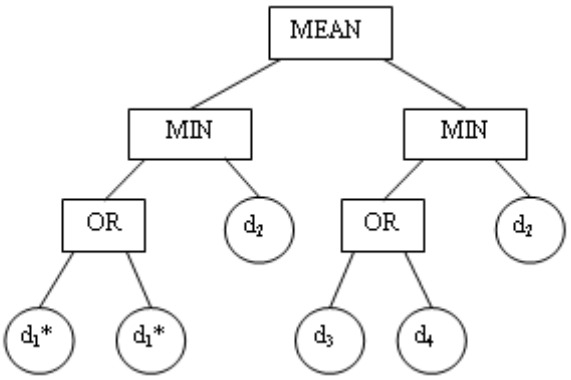


Figure 15: Representative tree for modelled scenario

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 dLC = SPD measure of life-cycle documentation

6.2.2 Method: A System for Security Assurance Assessment

Security assurance (SA) is the **objective confidence that an entity meets its security requirements [17]**. 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 behaviour 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 nSHIELD project.

6.2.2.1 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.

With the knowledge of security assurance values of all security assurance relevant entities and all security assurance relevant relations between these entities, the overall security assurance value of the system can be determined [17].

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.

In the cases of adding new entities to the existing system or combining different standard entities (i.e. patterns) is useful another approach, i.e. to assess the SA levels of the system entities separately before they are placed in the context of the system under study. With this assessment, the resulting values are the independent security assurance values for each entity. The effects of the interactions between connected patterns are then taken into account to obtain their operational SA values.

Operational SA Value or ***OSAV*** describe the SA level of a system entity or an attribute obtained when the system of evaluation is in operation. The *OSAV* of a system entity and an attribute are then respectively written $OSAV_e$ and $OSAV_a$. These values are the target of the SA assessment process.

SA metrics are necessary for adequate SA assessment in the above five-step process. SA metric is a system of SA relevant parameters or a standard for measuring SA, along with the procedures to carry out such measurement, with appropriate magnitude and scale, and an explanation of interpretation of the assessment

6.2.2.2 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.

6.2.2.3 SA Relations

Real physical connections are modelled as a pair of unidirectional **logical relations**, one in each direction. Figure 16 shows the abstraction of real connections from logical interactions between the elementary SA attributes of the corresponding entities.

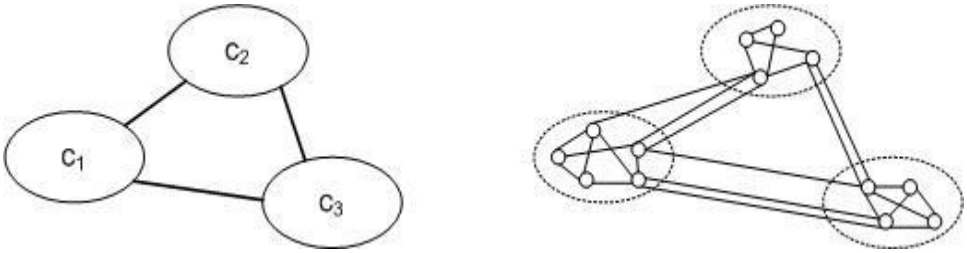


Figure 16: Actual relations between entities are represented as dependencies of their attributes.

Identifying the relations between elementary attributes of system entities is the goal of the modelling phase. It is assumed that the relations between the entities are known since aggregation takes place after the system under study has been modelled.

6.2.2.4 System Aggregation Process

The **aggregation problem** is a process of aggregating *n*-tuples, $n \geq 1$, 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 $Aggreg: \mathbb{R}^n \longrightarrow \mathbb{R}$, which assigns a real number *y* to any *n*-tuple $(x_1, x_2, \dots, x_n)$:

$$y = Aggreg(x_1, x_2, \dots, x_n) \quad (1)$$

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.

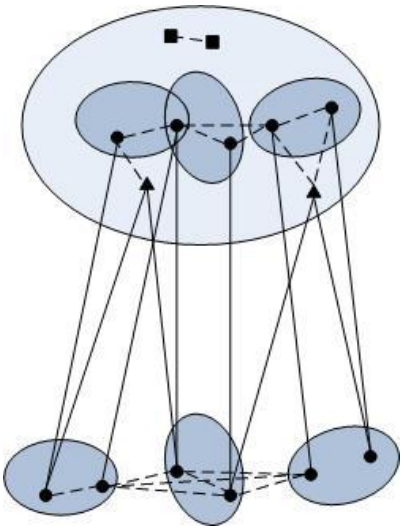


Figure 17: Emergent attributes can appear as a consequence of connecting system entities.

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**.

6.2.2.4.1 Aggregation with emergent attributes

The **attributes-dependency graph** is a hierarchical graph illustrated in Figure 18, 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).

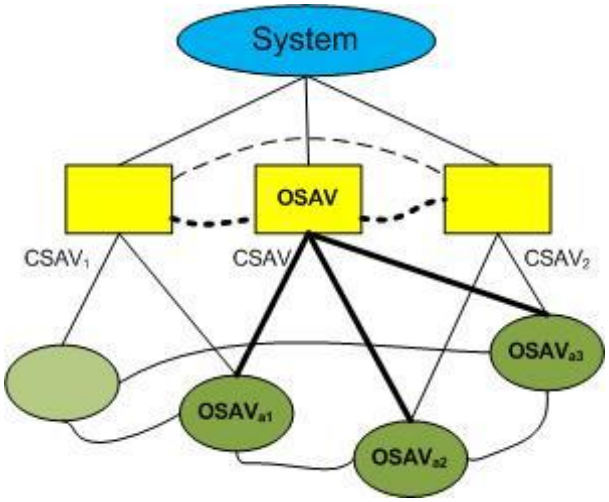


Figure 18: The attributes-dependency graph obtained from system decomposition.

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 $\boldsymbol{CSAV_a}$ 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 ($\boldsymbol{CSAV}$)** in the attributes-dependency graph is calculated from the operational values ($OSAVs$) of its child nodes by applying a function $Aggreg_{com}: \mathbb{R}^n \longrightarrow \mathbb{R}$ on these values depending on the decomposition relations between the node and its child nodes:

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

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 $\boldsymbol{OSAV_a}$ **as the operational SA value of the node**, $CSAV_a$ as the composition value of the node, $CSAV_{ai}$ ($1 \leq i \leq m$) as the composition values of the nodes having the emergent relations with the node under consideration. We define a function $Aggreg_{emer}: \mathbb{R}^n \longrightarrow \mathbb{R}$ 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:

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

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.

6.2.2.4.2 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.

6.2.2.4.2.1 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.

6.2.2.4.2.2 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:

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

6.2.2.4.2.3 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*).

6.2.2.4.2.4 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:

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

6.2.2.4.2.5 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:

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

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:

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

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

6.2.2.4.3 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.

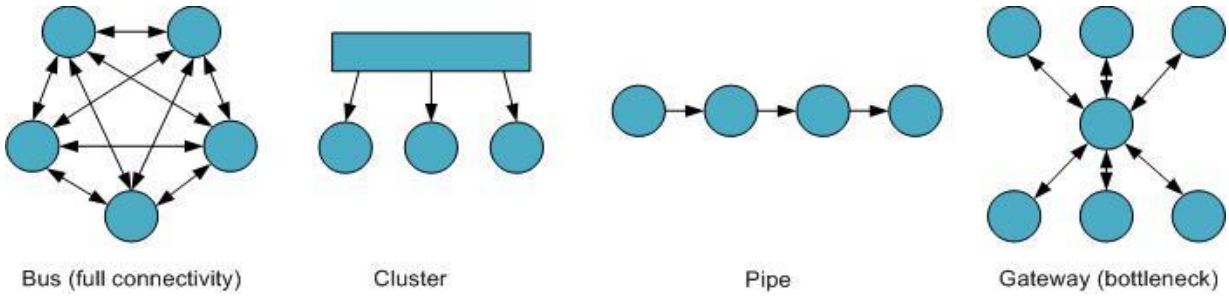


Figure 19: Examples of canonical architectures.

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

6.2.2.4.4 Aggregation

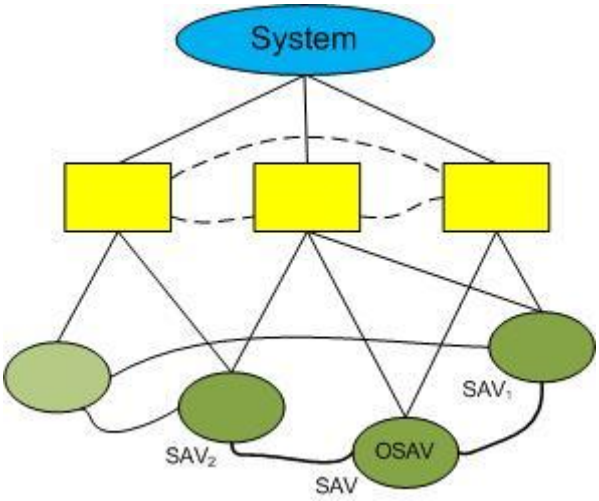


Figure 20: Attributes-dependency graph obtained when combining a number of evaluated entities.

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 20, 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 $\boldsymbol{SAV_i}$ ($1 \leq i \leq k$) 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 $Aggreg_{con}: \mathbb{R}^n \longrightarrow \mathbb{R}$ represents the **effect of the connection relations**. The $OSAV_a$ of every leaf in the attributes-dependency graph illustrated in Figure 20 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.

6.2.2.5 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 graph 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.

6.2.2.5.1 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.

6.2.2.5.2 Building Attack Graph

Example: **Sample network** (Figure 21)

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.

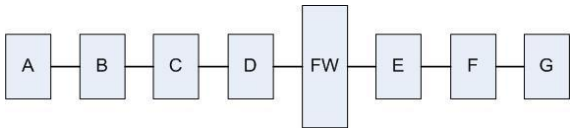


Figure 21: Sample network.

Figure 22 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 22 (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 22.

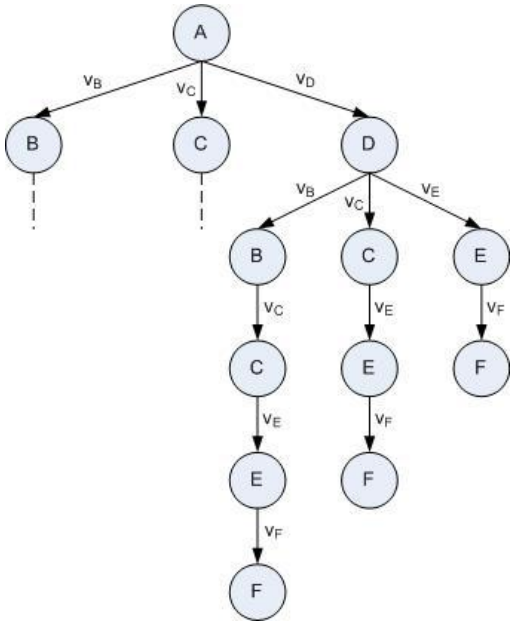


Figure 22. Full graph for sample network.

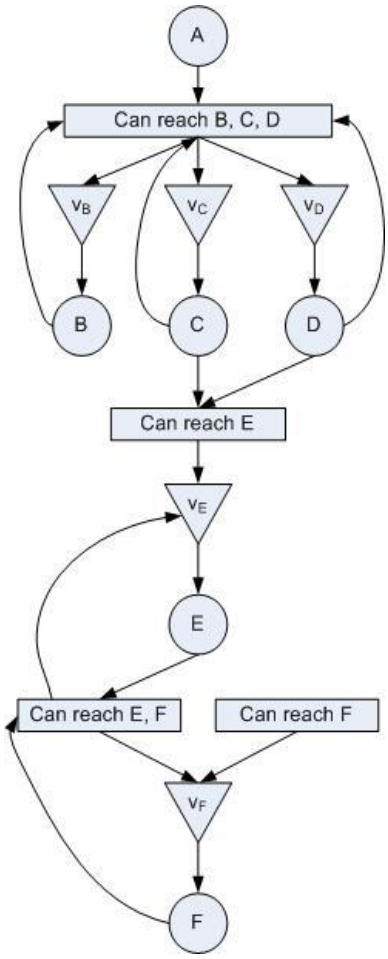


Figure 23. MP graph for sample network.

The **multi prerequisite graph (MP graph)** for example network is shown in Figure 23. 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.

The **multi prerequisite graph (MP graph)** for example network is shown in Figure 23. The MP graph explicitly represents the prerequisites of an attack. It is able to model portable credentials, such as

passwords, which an attacker can use to compromise a target. MP graph is fast to build and has great expressive power.

The MP graph's cycles embed the information contained in full graph without the redundant structure. If we do not have credentials, we can build a full graph from the MP graph by exploring the MP graph in a depth-first manner, stopping the exploration when we reach a vulnerability instance already used on the path from the root to the current node.

During graph construction full graphs must attempt actions which are not shown on the graph. In the Figure 22 the bottom node F of the full graph must be explored and vulnerabilities that could be reached from host F must be cut back. In the MP graph this problem is avoided by evaluating a prerequisite, such as the ability to reach hosts E and F, only once. Full graphs must evaluate a prerequisite once for each state that provides the prerequisite. The MP graph also shows all hosts which can be compromised from any host the attacker has compromised.

The MP graph has contentless edges and three types of nodes:

- **State nodes** (circles) represent an attacker's level of access on a particular host. Outbound edges from state nodes point to the prerequisites they are able to provide to an attacker.
- **Prerequisite nodes** or **pre-condition nodes** (rectangles) represent either a reachability group or a credential. Outbound edges from prerequisite nodes point to the vulnerability instances that require the prerequisite for successful exploitation.
- **Vulnerability instance nodes** (triangles) represent a particular vulnerability on a specific port. Outbound edges from vulnerability instance nodes point to the single state that the attacker can reach by exploiting the vulnerability.

These three node types in turn define the sole ordering of paths in the graph: a *state* provides *prerequisites*, which allow exploitation of *vulnerability instances*, which provide more *states* to the attacker. The maximum number of nodes in an MP graph is linearly related to the source data. There is at most one node for each vulnerability instance, state, reachability group, and credential.

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.

The MP graphs are built using a breadth-first technique where no node is explored more than once, and a node only appears on the graphs if the attacker can successfully obtain it. The pseudo code for graph generation is shown in Figure 24.

```
1  BFSQueue starts with the root node(s) representing the attacker`s starting state
2  While (BFSQueue is nonempty)
3  CurrNode = BFSQueue.dequeue()
4  DestSet = all nodes that can be reached from CurrNode
5  For each DestNode in DestSet
6  Add an edge from CurrNode to DestNode
7  If DestNode is brand-new
8  BFSQueue.enqueue(DestNode)
```

Figure 24: Pseudo code for MP graph generation

A weak point of the MP graph is that it cannot represent all possible positions of the attackers. G is not accounted in the constructed graph since the algorithm described in Figure 24 considers only the vulnerable nodes that can be reached from a current evaluated node. It will ignore the case when an internal attacker starts with G and can compromises E.

MP graph is an intermediate step to build the **host-based attack graph** shown in Figure 25 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.

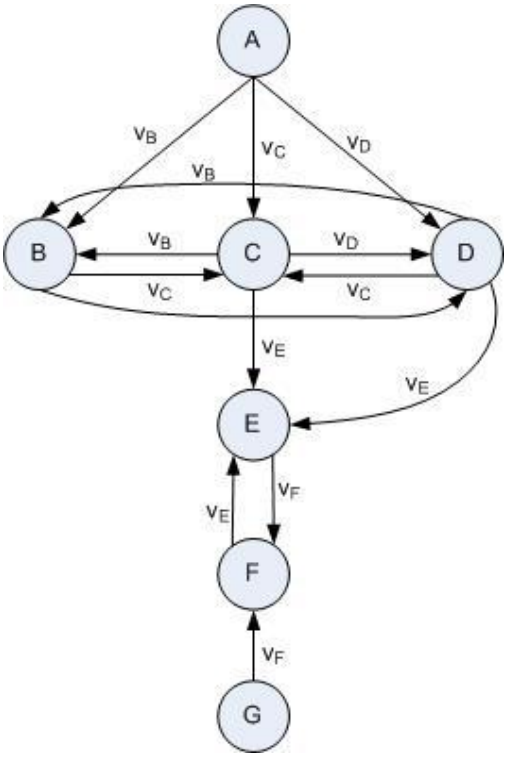


Figure 25: Host-based attack graph for sample network.

6.2.2.5.3 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:

$$SA_{Attackability} = (1 - \frac{nb\_inbound\_edges}{\sum nb\_inbound\_edges})$$

6.2.2.5.4 Examples

6.2.2.5.5 Generic Approach

Let us assume we 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.

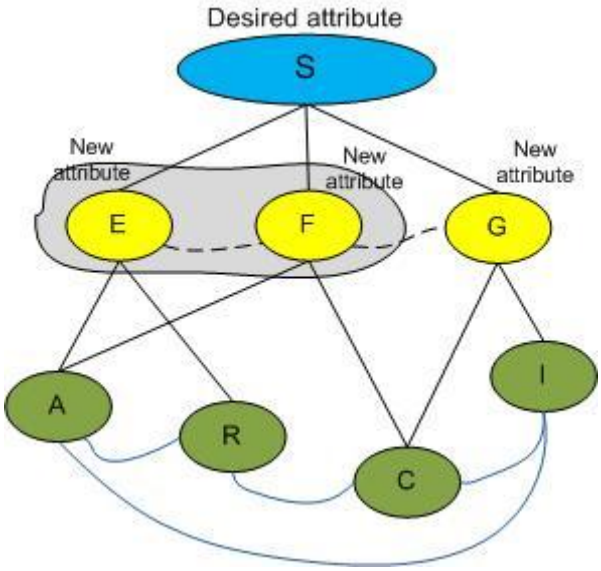


Figure 26: Attributes-dependency graph for an example

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:

$$OSAV_A = Aggreg_{con}(SAV_A, SAV_I, SAV_R)$$

$$OSAV_R = Aggreg_{con}(SAV_A, SAV_C, SAV_R)$$

$$OSAV_C = Aggreg_{con}(SAV_C, SAV_I, SAV_R)$$

$$OSAV_I = Aggreg_{con}(SAV_A, SAV_I, SAV_C)$$

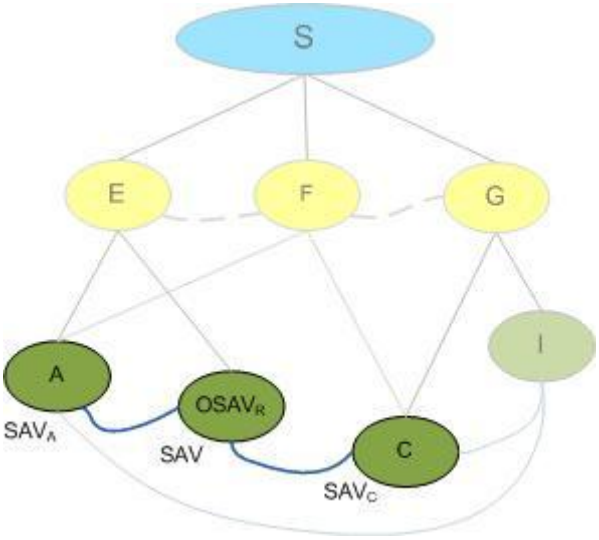


Figure 27: Pre-step for the node R

First step:

In this step we calculate the composition value of the node under composition from its child nodes.

$$CSAV_E = Aggreg_{com}(OSAV_A, OSAV_R)$$

$$CSAV_F = Aggreg_{com}(OSAV_A, OSAV_C)$$

$$CSAV_G = Aggreg_{com}(OSAV_C, OSAV_I)$$

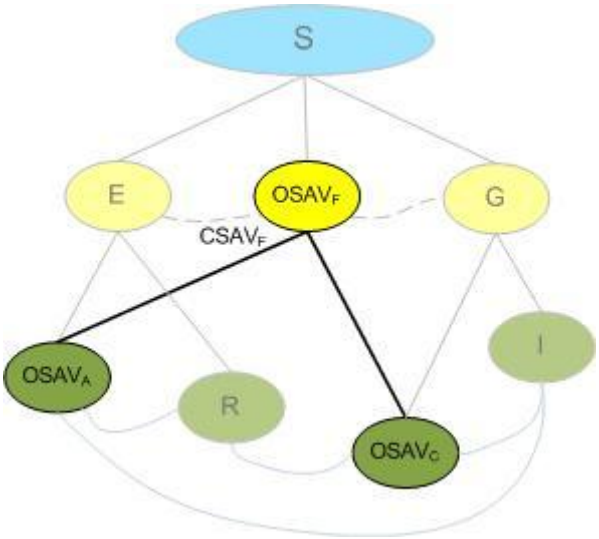


Figure 28: First step for the node F

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.

$$OSAV_E = Aggreg_{emer}(CSAV_E, CSAV_F)$$

$$OSAV_F = Aggreg_{emer}(CSAV_E, CSAV_F, CSAV_G)$$

$$OSAV_G = Aggreg_{emer}(CSAV_G, CSAV_F)$$

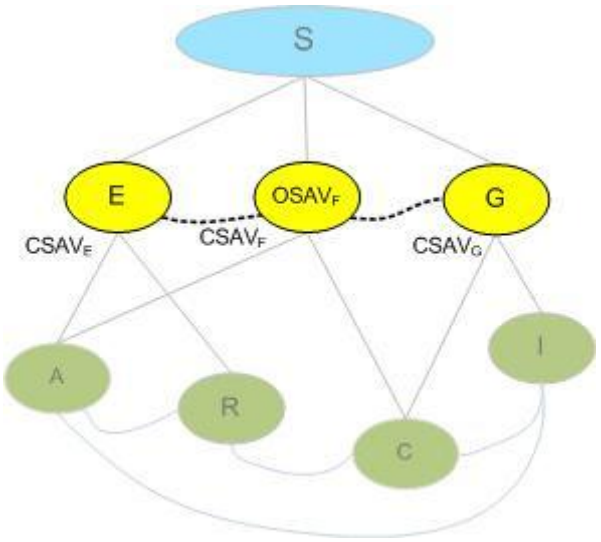


Figure 29: Second step for the node F

Last step:

The last step is the calculation of SA value under composition of the system.

$$CSAV_S = Aggreg_{com}(OSAV_E, OSAV_F, OSAV_G)$$

$$OSAV_S = CSAV_S$$

6.2.2.5.6 Example: Brute-force password attack

Consider the example of host-based mechanisms meant to defeat brute-force attempts at guessing passwords. When an attacker tries to access a machine, he often guesses passwords from a set of commonly used passwords and character sequences. After a few unsuccessful attempts, the machine typically blocks the attacked account. In the case of a cluster of machines that share a common set of users, the attacker can walk through the hosts in the cluster, trying brute-force attack against the same user-name on each host but specifying different ranges of attempted passwords per node. By keeping the number of login attempts below a threshold, the attacker may be able to successfully access to that cluster without violating the authentication mechanisms of each machine.

The SA levels of authentication attributes of each machine (for the attacked user-name) are represented as *AuthHost 1, AuthHost 2,* and so on (Figure 30). We can then decompose the authentication of each host into four attributes: requirements on passwords (key length, expiration), effectiveness and correctness of password verification algorithm, maximum number of permitted attempts, and timeout limit. Note that we cannot express the weakening effect of authentication attributes described above through the relations between lower-level attributes (*requirements of passwords, nbAttempts*, etc.). There are therefore emergent relations representing that weakening effect.

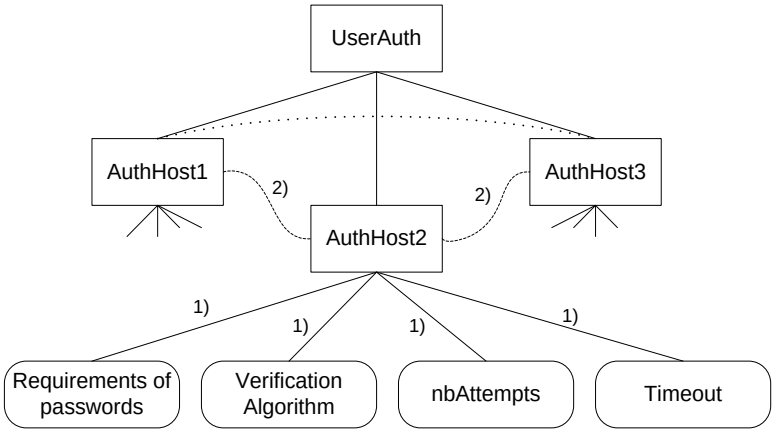


Figure 30: Cluster authentication decomposition.

Suppose that the $OSAV$ of *requirements of password*, *verification algorithm, nbAttempts,* and *timeout* have been determined. The SA level of the authentication attribute of each machine is calculated as follows:

1. Calculate the composition value $CSAV_{AuthHost}$: in this case, this can be done by applying the weighted-sum operator on the $OSAV$ values of its decomposed attributes.

$$CSAV_{AuthHost} = \sum_{i=1}^{4} w_i \, OSAV_i \quad with \sum_{i=1}^{4} w_i = 1$$

where *i* goes through elementary nodes, $OSAV_i$ is the SA value of elementary node *i*, w_i is the corresponding weight percentile.

$$CSAV_{AuthHost1} = w_{ReqPass} \, OSAV_{ReqPass} + w_{VerAlg} \, OSAV_{VerAlg} + w_{nbAtt} \, OSAV_{nbAtt} + w_{Timeout} \, OSAV_{Timeout} = 0{,}3 * OSAV_{ReqPass} + 0{,}4 * OSAV_{VerifAlg} + 0{,}2 * OSAV_{nbAttempts} + 0{,}1 * OSAV_{Timeout}$$

$$CSAV_{AuthHost2} = w_{ReqPass} \, OSAV_{ReqPass} + w_{VerAlg} \, OSAV_{VerAlg} + w_{nbAtt} \, OSAV_{nbAtt} + w_{Timeout} \, OSAV_{Timeout} = 0{,}3 * OSAV_{ReqPass} + 0{,}4 * OSAV_{VerifAlg} + 0{,}2 * OSAV_{nbAttempts} + 0{,}1 * OSAV_{Timeout}$$

$$CSAV_{AuthHost3} = w_{ReqPass} \, OSAV_{ReqPass} + w_{VerAlg} \, OSAV_{VerAlg} + w_{nbAtt} \, OSAV_{nbAtt} + w_{Timeout} \, OSAV_{Timeout} = 0{,}3 * OSAV_{ReqPass} + 0{,}4 * OSAV_{VerifAlg} + 0{,}2 * OSAV_{nbAttempts} + 0{,}1 * OSAV_{Timeout}$$

2. When the previous step has been performed in every machine of the cluster, the obtained results are then used to determine the actual operational value $OSAV_{AuthHost}$ by taking into account the

emergent relations with other nodes. With this example, the emergent relations can be represented by using the multiply operator applying on the $CSAV_{AuthHost}$ of the hosts.

$$OSAV_{AuthHost1} = CSAV_{AuthHost2} \times CSAV_{AuthHost3}$$

$$OSAV_{AuthHost2} = CSAV_{AuthHost1} \times CSAV_{AuthHost3}$$

$$OSAV_{AuthHost3} = CSAV_{AuthHost1} \times CSAV_{AuthHost2}$$

That is, the obtained $OSAV_{AuthHost}$ of each host is smaller than the composition value $CSAV_{AuthHost}$ because of the effect of emergent relations. Finally, the overall operational authentication SA value of the cluster (*UserAuth,* Figure 30) can be determined, for example, by using the weighted-sum operator applying on the $OSAV_{AuthHost}$ of every machine.

$$CSAV_{UserAuth} = w_{AuthHost1}\, OSAV_{AuthHost1} + w_{AuthHost2}\, OSAV_{AuthHost2} + w_{AuthHost3}\, OSAV_{AuthHost3}$$

A weak point of the approach described above is that the amount of nodes in the dependency graph can grow into a huge number after some steps of decomposition. To get the balance between the complexity and the time of evaluation, one can therefore assess the high-level nodes separately; the overall system SA value is then obtained by combining the SA values these nodes according to original system topology.

6.2.3 Method – Formalisation through Metrics Patterns (Derivation of 6.2.2)

Measurement is critical to the future of software security and especially in the field of Security engineering starts with requirements specification. However this specification is usually domain specific. This document will define Security Objectives instead of Security requirements in order to be domain independent. This will allow deploying and activating easier security metrics in order to help monitoring the system as a whole.

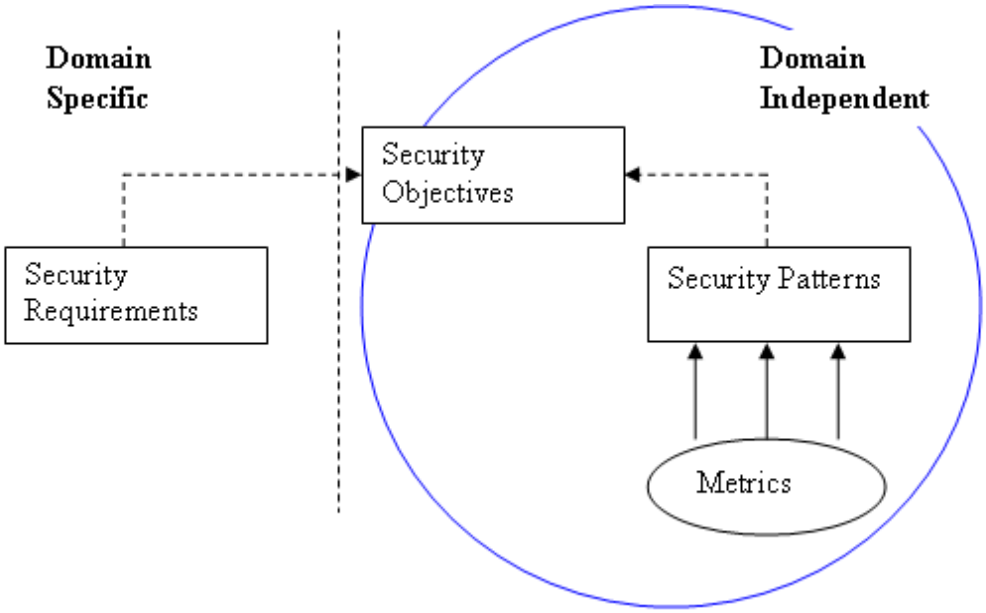


Figure 31: From SPD Requirements to Metrics through Security Objectives

This section aims to formalise the collection process of metrics. For doing this, nSHIELD will define SPD objectives, which are independent from the scenario chosen. Actually, SPD objectives differ from SPD requirements in this area: while requirements are domain specific, objectives and goals are domain

independent. This approach [18] is similar to “out of context” concept which is a very useful concept for certifying SPD.

Security patterns represent a well-known technique to package domain-independent knowledge and expertise in a reusable way and will contribute to achieve security objectives defined in this document.

Security metrics compose a particular security pattern and determine if a security pattern is being compliant or not with respect to desired security objective. Security objectives have to be distilled from security requirements. For doing this, an analysis of document 2.1 and 2.2 about nSHIELD SPD Requirements have to be done.

The association of SPD metrics to SPD patterns (in nSHIELD SPD functions) allows the metric to easily be instantiated in the applications we have in nSHIELD (Railroad Security scenario, Voice/Facial Verification scenario, Dependable Avionics System Scenario and Social Mobility and Networking Scenario): metrics are selected implicitly by selecting the SPD patterns which reduces the problem of selecting the correct metrics. The challenge should be adopted by selecting the correct patterns and this is on challenge faced together with the overlay layer (overlay layer commands the SPD functions and SPD core services).

In the following picture (an example for nSHIELD), contains a hierarchical graph where the security objective correspond to the security requirements that needs to be fulfilled. High level objectives are recursively redefined into lower level objectives (or SPD functions) by the usage of AND-decomposition. The decomposition depicts the interrelationship between different objectives.

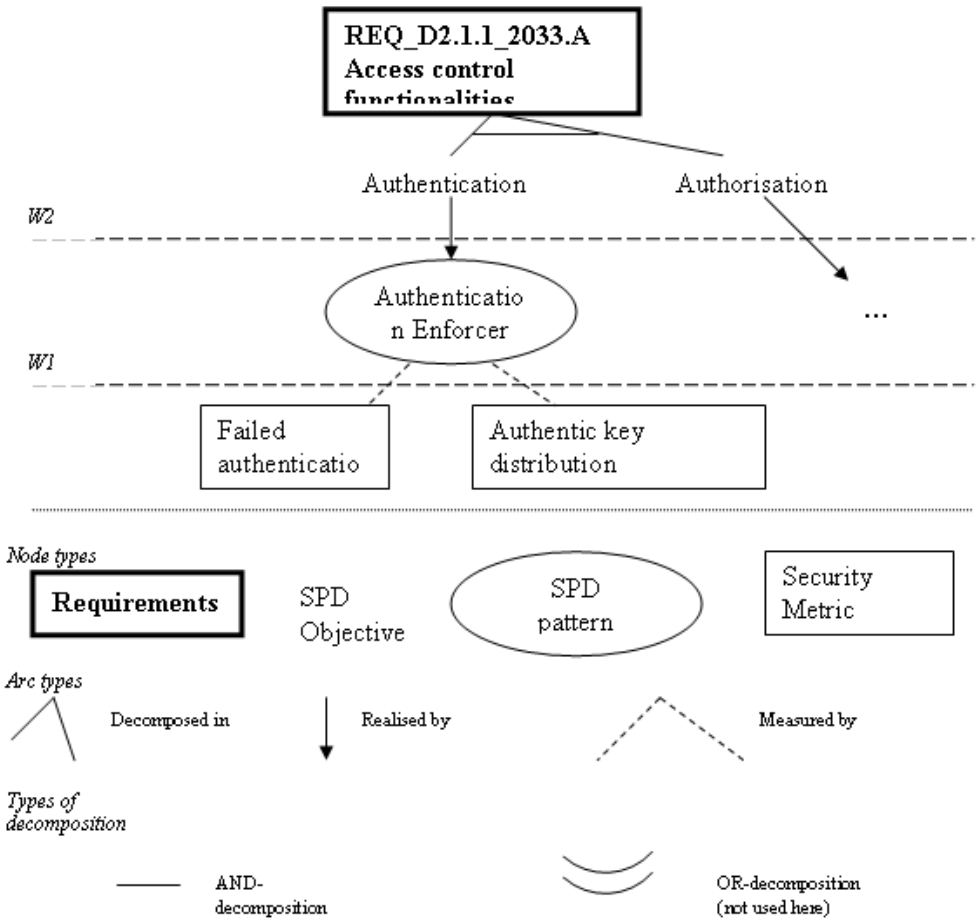


Figure 32: Dependency graph extrapolated to nSHIELD (from nSHIELD requirements to nSHIELD Metrics)

The figure depicts a formal method that begins at the distillation of nSHIELD requirements and ends with the definition of nSHIELD SPD Metrics.

The measurements obtained by instantiating these metrics give an indication of how well authentication is realised. *W2* defines the transformation from security objective to security pattern. This is a mapping job which should be easily done by the SPD process expert in one particular scenario. Previously this expert has to extract the SPD objectives from the SPD Requirements. This is the **most valuable** issue to point out from this method: **requirements and SPD Metrics are linked and can be traced**. The "Access Control Functionalities" requirement is divided into 2 security objectives (Authentication and Authorisation – which is not used in this example). One of the "Authentication" SPD patterns has been identified as "Authentication Enforcer" and this, in turn, has several metrics below w1 level. The main challenge is to enable the correct SPD patterns in order to fulfil SPD objectives and thus accomplishing SPD Requirements.

In *W1* level, 2 metrics have been identified (Failed authentication and Authentic Key Distribution Management). Once we enable the SPD pattern (father node: Authentication Enforcer), these metrics will be activated. But how will we activate the correct node (pattern)? In the following section a recursive algorithm is proposed in order to activate an "aggregation" algorithm that will assign values to each node of the dependency graph.

Proposed Algorithm to activate and aggregate the activation of SPD metrics

```
Algorithm SPD_Composition_Measurement_Metrics_Patterns_Objectives_Req (Obj)

Require: AND-Nodes initialised with value 1; OR-Nodes initialised with value 0.

If Obj is not leaf objective then
        For all Obj_s in Obj.subobjectives do
                SPD_Composition_Measurement_Metrics_Patterns_Objectives_Req(Obj_s)
                If Obj is and AND-node then
                        value ← min {value, Obj_s.Value}
                Else
                        Value ← max{value, Obj_s.value}
                End if
        End for
Else if Obj has patterns assigned to it then
        For all p in Obj.patterns do
                Calculate_metrics(p)
                If Obj is an AND-node then
                        value ← min {value,p.value}
                Else
                        Value ← max {value, p.value}
               End if
        End for
End if
Obj.value ← value
```

Figure 33: Metrics aggregation algorithm.

6.2.4 Method: Metrics Human Immunologic system

This methodology compares our system with the human immunologic system to detect possible intruders. Thus, as in human immunologic system we can find antigens or intruders, the agents that detect them, the system itself and recognition mechanisms, the architecture of our system can be modelled by control and reaction agents (C/R agents), network and maintenance functions and distributed databases.

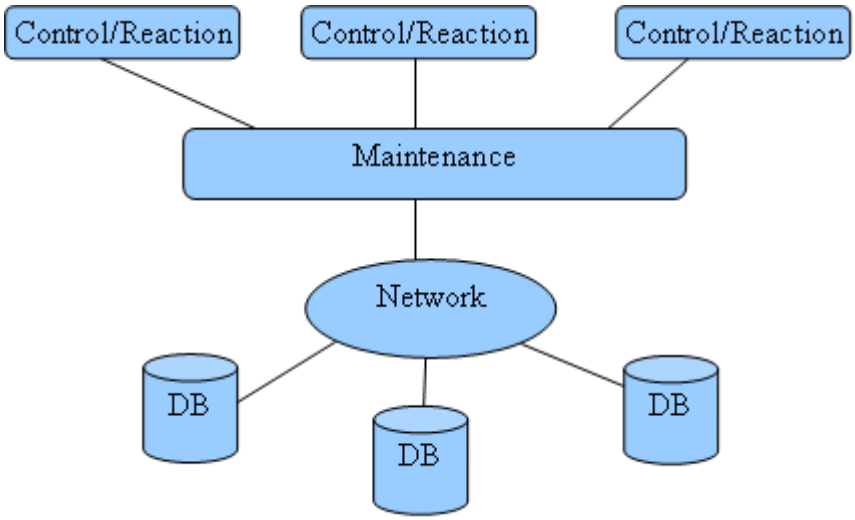


Figure 34: System architecture

- **Control and reaction agents:** monitor users' (*) actions and the use given to the system resources in order to react when an anomaly is found.
  (*) The user will usually be a device, so the interaction of a device in the system input should be measured.
- **Maintenance agents:** group control and reaction agents, handle data encryption and maintenance of databases, returning results to C/R agents based on their requests.
- **Network agent:** group maintenance agents and is responsible for packet filtering and detection of other attacks (multi-host, DoS...)

Metrics used to quantify users' behaviour are based on a “Behaviour Vector” which takes into account effort, memory, dependability and other requirements:

Behaviour Vector = EffortW + MemoryX + DependabilityY + SpecialRequirementsZ

- *Effort:* takes into account aspects such as CPU, disk read/writes, the duration of the session, the amount of data transferred to the network.
- *Memory:* number of invalid logins, incorrectly typed commands and the number of times that system information commands are executed, are checked.
- *Dependability:* is calculated from password complexity, the number of invalid actions per session and encrypted information.
- *Special requirements:* the type of connection to the system, the type of authentication, and the presence of accessibility requirements are checked.

In summary, the “Behaviour Vector”, allow us to quantify a user's behaviour at a particular time, so it is necessary to observe changes in behaviour over time. Statistics and a “decision tree” may be used to extract results.

This method is used as a metric aggregation. To obtain SPD metrics for the entire system, the interrelationship and dependence of different metrics is taken into account. Below is an example of possible decision tree based on different metrics:

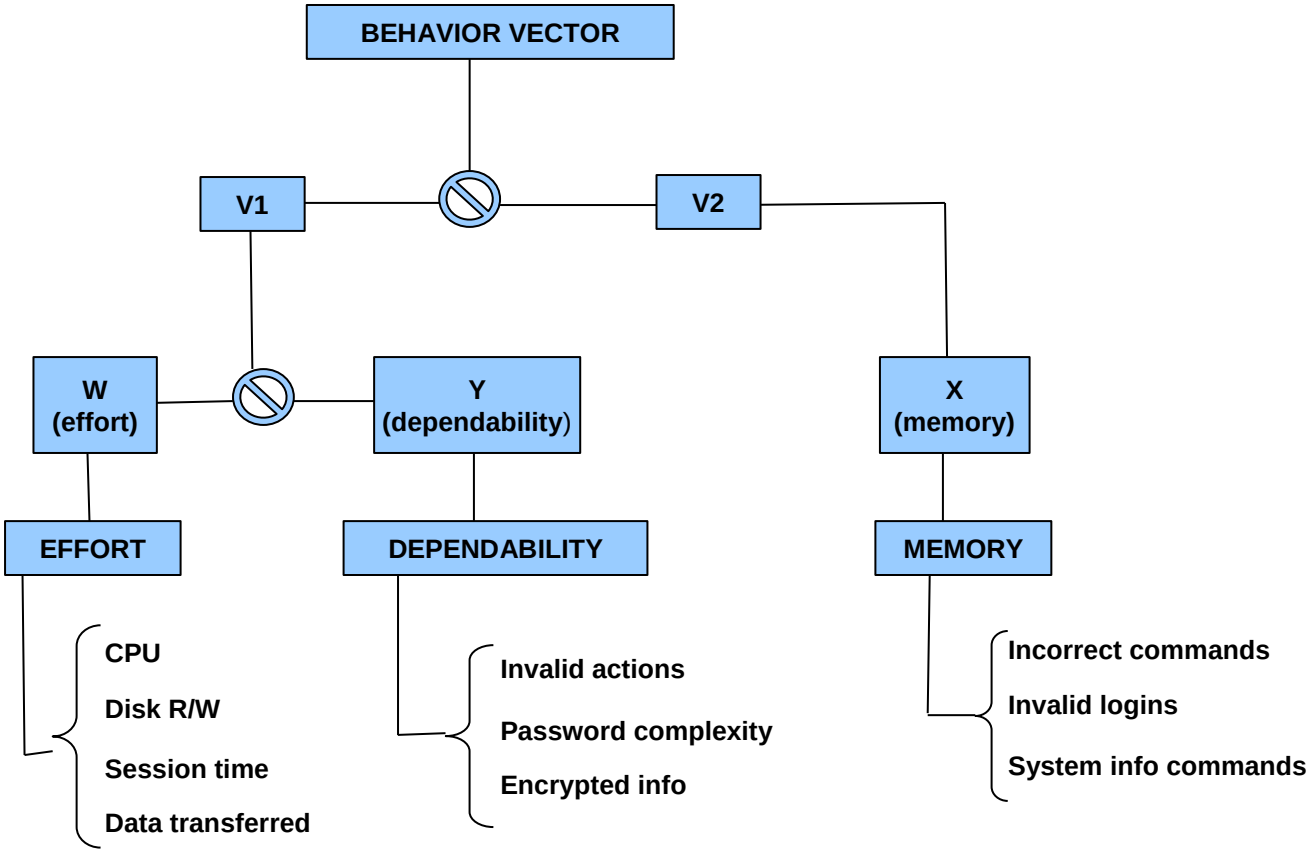


Figure 35: Behaviour vector tree

6.2.5 Method: Metrics for Intrusion tolerant system

This methodology is based on a precondition: tolerating security intrusion and reconfiguring, regenerating or rejuvenating a system after a security intrusion has occurred.

Complex systems may content errors and, sometimes, these errors are caused deliberately by security intruders. It is important to prevent security intrusions and treat security as a QoS attribute, along with availability and performance.

An intrusion tolerant system must be able to reorganize itself from a security attack in order to reduce the effects that may be caused. A state transition diagram is shown below:

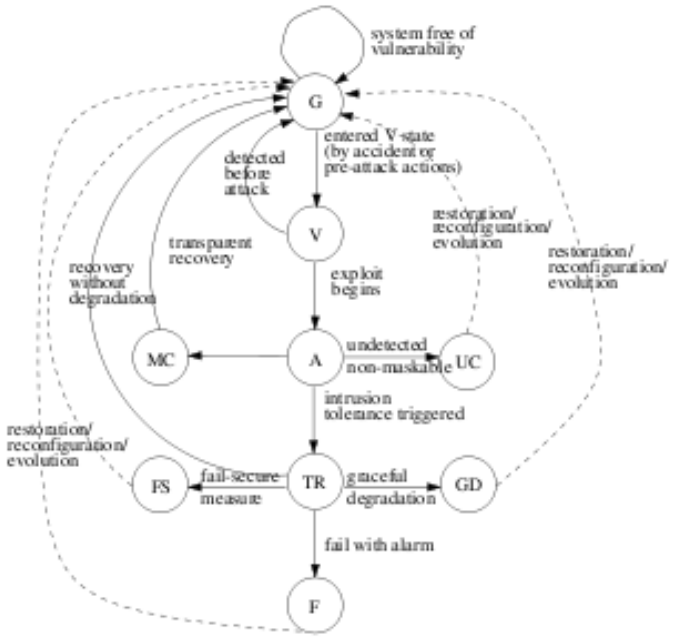


Figure 36: State machine for intrusion tolerant system

If security strategies fail the system changes from a good state G to a vulnerable V state. If the attack is detected it is possible to bring the system from the vulnerable state back to the good state, but if the vulnerability goes on, then the system changes into an active attack state A.

If there is enough redundancy to enable the delivery of error-free service it is possible to mask the attack impact MC and bring the system back to a good state, but if the system fails recognizing the attack, the damage may go on leading to an undetected compromised state UC, where the service is not assured.

When an active attack is detected, the system enters the triage state TR, where it decides whether to recover or limit the possible effects of the attack. If the aim is to protect the system from DoS the system enters a graceful degradation state GD providing just essential services, but if the aim is to protect data integrity or confidentiality, the system enters the fail-secure state FS and it stops.

Finally, if all mentioned strategies fail, the system enters the fail state F and signals an alarm.

This model allows quantifying dependability attributes, such as safety, reliability, availability, maintainability, integrity and confidentiality. As an aggregation of SPD metrics, this methodology requires the knowledge of some parameters, such as mean sojourn time in each state and the transition probabilities. Thereby, SPD metrics are calculated taking into account the different states and other parameters involved.

For example, to calculate steady state availability the states where the system is unavailable must be considered:

$$A = 1 - (\pi_{FS} + \pi_F + \pi_{UC})$$

where π_i $i \in \{FS, F, UC\}$ denotes the steady state probability that the process is in state .

7 nSHIELD Metrics system deployment

In this section, we are going to provide a first brief analysis of how nSHIELD should deploy SPD Metrics into its 4 use cases. The whole analysis will be in deliverable *D2.8. Final SPD Metrics Specification*, where we will develop a common or specific strategy for the deployment in each of the scenarios.

Particularly, in this section we will focus on the convergence between SPD metrics and related requirements for the nSHIELD use cases.

7.1 SPD Metrics Design steps

The What, How and Integration model described in previous sections can be exploited into 7 steps (Aligned with section 4) that lead this section and will also lead deliverable D2.8 Final SPD Metrics Specification.

Regardless of the underlying framework, the seven key steps below could be used to guide the process of establishing a security metrics deployment process.

Figure 37: SPD metrics design process

This seven-step methodology should yield a firm understanding of the purpose of the security metrics, its specific deliverables, and how, by whom, and when these deliverables will be provided.

Step 1: Define the metrics program goal(s) and objectives (already done in chapter 5) - WHAT

Because developing and maintaining a security metrics program could take considerable effort and divert resources away from other security activities, it is critical that the goal(s) and objectives of the program be well-defined and agreed upon up front. Although there is no hard and fast rule about this, a single goal that **clearly states the end toward which all measurement and metrics gathering efforts should be directed** is a good approach. A goal statement might be, for example:

"Provide metrics that clearly and simply communicate how efficiently and effectively our company is balancing security risks and preventive measures, so that investments in our security program can be appropriately sized and targeted to meet our overall security objectives."

Statements of objective should **indicate high-level actions that must be collectively accomplished** to meet the goal(s). An action plan should be directly derivable from these statements. A few objectives for the goal above, for example, might be:

a. *To base the security metrics program on process improvement best practices within our company.*
b. *To leverage any relevant measurements currently being collected.*
c. *To communicate metrics in formats custom-tailored to various audiences.*
d. *To involve stakeholders in determining what metrics to produce.*

Step 2: Decide which metrics to generate[9]

Any **underlying corporate framework for process improvement could dictate what metrics are needed**. For example, a "Six Sigma" approach would focus on security processes for which defects could be detected and managed. Step 2 of building a metrics program would, therefore, be to identify those specific security processes. A compliance-based approach would assess how closely established security standards are being followed. In this case, Step 2 would identify those standards for which compliance should be tracked.

In the absence of any pre-existing framework, a top-down or a bottom-up approach for determining which metrics might be desirable could be used. The top-down approach starts with the objectives of the security program, and then works backward to identify specific metrics that would help determine if those objectives are being met, and lastly to determine measurements needed to generate those metrics. For example:

Table 61: Top-down approach

Top-down Approach	
Steps	**Examples**
a. Define/list objectives of the overall security program	*To reduce the number of virus infections within the institution by 30% by 2015*
b. Identify metrics that would indicate progress toward each objective	*Current ratio of viruses in the wild to actual infections as compared to the baseline 2012 figure*
c. Determine measurements needed for each metric	*Number of virus in the wild as reported by abc external source* *Number of virus infections detected*

The bottom-up approach entails first defining which security processes, products, services, etc. are in place that can be or already are measured, then considering which meaningful metrics could be derived from those measurements, and finally assessing how well those metrics link to objectives for the overall security program. To illustrate:

[9] Done in chapter 5 but will be corroborated in D2.8

Table 62: Bottom-up approach

Bottom-up Approach	
Steps	**Examples**
a. Identify measurements that are/could be collected for this process	*Monthly number of critical vulnerabilities detected in servers using xyz scanning tool*
b. Determine metrics that could be generated from the measurements	*Change in number of critical vulnerabilities detected in servers since xyz scanning tool implemented*
c. Determine the association between the derived metrics and established objectives of the overall security program	*To reduce the number of detectable vulnerabilities on servers by 95% by 2011*

The top-down approach will more readily identify the metrics that should be in place given the objectives of the overall security program, while the bottom-up approach yields the most easily obtainable metrics. Both approaches assume that overall security program objectives have already been established. If they have not been, defining these high-level objectives is obviously important and a prerequisite.

Step 3: Develop strategies for generating the metrics[10] - HOW

Now that what is to be measured is well understood, **strategies for collecting data needed and deriving the metrics must be developed**. These strategies should specify the **source of the data, the frequency of data collection, and who is responsible for raw data accuracy, data compilation into measurements, and the generation of the metric**.

Step 4: Establish benchmarks and targets (Current use cases as benchmark)

In this step **appropriate benchmarks would be identified and improvement targets set**. Benchmarking is the process of comparing one's own performance and practices against peers within the industry or noted "best practice" organizations outside the industry. Not only does this process provide fresh ideas for managing an activity, but also can provide comparative data needed to make metrics more meaningful. Benchmarks also help establish achievable targets for driving improvements in existing practices. A security manager should consult industry-specific data resources for possible benchmarks and best practices.

Step 5: Determine how the metrics will be reported[11]

Obviously, no security metrics efforts are worthwhile if the results are not effectively communicated. So, **effective communication of metrics is obviously key matter and should not be over-simplified, but be presented clearly**. Executives are accustomed to dealing with financial and other trend lines, so complex security-related data can be valuable to this group if presented well. Graphic representations are particularly effective.

[10] Referred to chapter Quantitative solutions for Metrics composition

[11] Plan to do in deliverable D2.8

Some metrics may be meaningful only to the security manager and staff and should not be distributed further. Security managers may, however, use other metrics to help trigger needed remedial actions with the organization.

Since overlay mechanisms in nSHIELD also collect and act upon composite values of metrics collected from other components, a machine-readable and validatable format (such as XML) describing each metrics is also needed to be established, as well as means to represent, validate and transfer the metrics values.

In any case, **the context, format, frequency, distribution method, and responsibility for reporting metrics should be defined up front**, so that the end product can be visualized early on by those who will be involved in producing the metrics and those who will be using them for decision-making.

Step 6: Create an action integrated plan and act on it[12] - INTEGRATION

Now it is time to get the real work done. **The action plan should contain all tasks that need to be accomplished to launch the SPD metrics program**, along with expected completion dates and assignments. As mentioned in Step 1, action items should be directly derivable from the objectives. Documenting the linkage of actions in the plan to these objectives is useful, so that no one will lose sight of why a given action is important.

In the same manner that software should be developed, it is critical to include a testing process in the plan. Deficiencies in collected data may, for example, prove some metrics unusable and require reexamination of what is to be measured and how.

Step 7: Establish a formal review/refinement cycle[13]

Formal, regular reexamination of the entire security metrics program should be built into the overall process. This step will serve as an input for Task *6.3 Lifecycle SPD Support* for nSHIELD.

Through these steps it should be possible to answer questions such as: Are the metrics useful in determining new courses of action for the overall SPD functionalities? How much effort is it taking to generate the metrics? Is the value derived worth that effort? These and other similar questions will be important to answer during the review process. A fresh review of security metrics standards and best practices within and outside the industry should also be conducted to help identify new developments and opportunities to fine-tune the program.

7.2 Convergence with nSHIELD Scenarios

During the 7 steps described above, in this document we have identified some metrics and we need to select them according to the particularities of each of the use cases. **Hence we can say that we have developed most of the work done in the first 4 steps (leaving the 3 final ones for deliverable D2.8.)** However this section establishes a common dependency between SPD metrics – Requirements – Scenarios.

In chapter 7 of *D2.2 Preliminary System Requirements and Specifications*, generic requirements for nSHIELD scenarios can be found. The applicability of SPD metrics identified in chapter 5 into the scenarios will be the main challenge of this section and deliverable D2.8.

The following table shows the dependencies found among the high level requirements of scenarios and the SPD Metrics identified in chapter *5 Metrics in nSHIELD multi-layer scheme* (Convergence with Requirements).

[12] Planned to do in deliverable D2.8

[13] Planned to do in deliverable D2.8

Table 63: Mapping between scenario HL Requirements and SPD metrics identified

Scenario name	Scenario High-Level requirement	SPD Metric
Railway Scenario	REQ_D2.1.1_0201.A Application scenario – Information availability	- Availability - Uptime
	REQ_D2.1.1_0202.A Application scenario – Information integrity	- Digital Signatures
	REQ_D2.1.1_0205.A Application scenario – Information privacy	- Policy updates - Privacy-centric physical/tamper resilience - ECC Authentication - Storage of private information - Anonymity and location privacy - Privacy across trust domains
	REQ_D2.1.1_0208.A Application scenario - SPD parameters assurance/evaluation	- Code execution - Digital signatures - Policy updates - Security context establishment - Conflict resolution between policy domains - Dynamic security behaviour - ES Certification - HW/SW co-design - Detection accuracy
	REQ_D2.1.1_0209.A Application scenario – Mechanisms for failure mitigation	- Availability - Uptime - Code execution - Data freshness - Digital signatures - Policy Updates - Secure key distribution - Physical tamper resilience - Virtualisation - Alternative power supply sources - Dependable key distribution mechanisms - Network Latency - Security metrics (confidentiality, integrity, routing) - Detection accuracy
	REQ_D2.1.1_0211.A Application scenario – Dynamic adaptively to the available resources	- Policy Updates - Dynamic security behaviour - Runtime reconfiguration
	REQ_D2.2_RW1. Real Time Availability	- Availability
	REQ_D2.2_RW16. Data back up	
	REQ_D2.2_RW2 Information	- Security metrics (Integrity)

	transmission integrity	
	REQ_D2.2_RW3 Information transmission confidentiality	- Security metrics (confidentiality)
	REQ_D2.2_RW4 and RW5. Peer and user authentication	- Security metrics (confidentiality, integrity, routing) - -Privacy-centric physical/tamper resilience - Secure key distribution
	REQ_D2.2_RW6. Authorisation	- Policy updates - Dynamic security behaviour - Runtime reconfiguration
	REQ_D2.2_RW7.Secure Node Deployment	- Identification through digital signature
	REQ_D2.2_RW8. Secure Software Upgrade	- Policy updates
	REQ_D2.2_RW9. DoS	- Availability
	REQ_D2.2_RW10.Secure execution	- Code execution
	REQ_D2.2_RW11. Secure Booting	
	REQ_D2.2.RW12 SPD Level Assignment REQ_D2.2.RW13 SPD Level Identification	- Composition of different metrics - Security context - Situation and awareness
	REQ_D2.2.RW14 Software Failure Mitigation	- Availability - Policy updates
	REQ_D2.2.RW15 Hardware Failure Mitigation	
Voice/facial recognition scenario	REQ_D2.1.1_0801.A Voice/Facial Recognition Scenario: Biometric data security	- Security metrics (confidentiality, integrity, routing)
	REQ_D2.1.1_0802.A Voice/Facial Recognition Scenario: Biometric data privacy REQ_D2.2_VF4 Biometric data privacy	- Security metrics (confidentiality, integrity, routing) - -Privacy-centric physical/tamper resilience - Secure key distribution
	REQ_D2.1.1_0803.A Voice/Facial Recognition Scenario: Data storage REQ_D2.2_VF3 Protection of biometric data at storage	- Security metrics (confidentiality, integrity, routing) - Privacy-centric physical/tamper resilience

	REQ_D2.1.1_0805.A Voice/Facial Recognition Scenario: Data encrypting REQ_D2.2_VF5. Biometric data and metadata confidentiality during transmission	- Security metrics (confidentiality, integrity, routing) - Privacy-centric physical/tamper resilience - Digital signatures - Secure key distribution
	REQ_D2.1.1_0807.A Voice/Facial Recognition Scenario: thresholds	- Selected SPD metrics maintain and update thresholds.
	REQ_D2.2_VF1 Secure identification though face recognition	- Security metrics (confidentiality, integrity, routing) - -Privacy-centric physical/tamper resilience - Secure key distribution
	REQ_D2.2_VF7. Biometric Data Privacy REQ_D2.2_VF8. Biometric Metadata privacy	- Privacy across trust domain - Storage of private information - Privacy-centric physical/tamper resilience
	REQ_D2.2_VF12. System Fault Tolerance	- Availability - Performance related metrics - Uptime
	REQ_D2.2_VF13. Transmitted biometric information integrity	- Data exchange integrity
	REQ_D2.2_VF14. Secure execution	- Code execution
Avionics scenario	REQ_D2.1.1_1201.A Avionics Scenario – Data Availability REQ_D2.2_AV1 Real-time availability	- Availability - Uptime
	REQ_D2.1.1_1202.A Avionics Scenario – Data Acquisition Integrity REQ_D2.2_AV2 Transmitted information integrity	- Privacy-centric physical/tamper resilience - Digital signatures - Secure key distribution
	REQ_D2.1.1_1203.A Avionics Scenario – Data Storing Integrity REQ_D2.2_AV17 Data storage integrity (also linked to REQ_D2.2_AV19 Data acquisition integrity	- Privacy-centric physical/tamper resilience - Digital signatures - Secure key distribution
	REQ_D2.1.1_1207.A Avionics Scenario – Information privacy REQ_D2.2_AV3 Transmitted information confidentiality	- Privacy-centric physical/tamper resilience - Digital signatures - Secure key distribution - Tamper resiliency

REQ_D2.2_AV18_ Data storage confidentiality	
REQ_D2.1.1_1210.A Avionics Scenario – nSHIELD compliant	- Security context establishment - Conflict resolution between policy domains - Situation and context awareness
REQ_D2.1.1_1211.A Avionics Scenario – Procedure against SW failure	- Code execution - Policy updates
REQ_D2.1.1_1212.A Avionics Scenario – Procedure against HW failure	- HW Code execution - Policy updates - Privacy-centric physical/tamper resilience
REQ_D2.1.1_1214.A Avionics Scenario – SPD Static configuration	- Situation and context awareness - Security context establishment
REQ_D2.1.1_1215.A Avionics Scenario – SPD dynamic configuration	- Security context establishment - Policy updates - Runtime reconfiguration
REQ_D2.1.1_1216.A Avionics Scenario – Dynamic adaptively to the available resources	- Runtime reconfiguration - Policy updates
REQ_D2.2_AV4/AV5 User and peer authentication	- Digital signatures - Secure key distribution
REQ_D2.2_AV6. Authorisation	- Policy updates - Dynamic security behaviour - Runtime reconfiguration
REQ_D2.2_AV7.Secure Node Deployment	- Identification through digital signature
REQ_D2.2_AV8. Secure Software Upgrade	- Policy updates
REQ_D2.2_AV9.Secure execution	- Code execution
REQ_D2.2_AV10. Secure Boot	
REQ_D2.2.AV11 SPD Level Assignment REQ_D2.2.AV12 SPD Level Identification	- Composition of different metrics - Security context - Situation and context awareness
REQ_D2.2.AV13 Software Failure Mitigation	- Availability - Policy updates
REQ_D2.2.AV14 Hardware	

	Failure Mitigation	
	REQ_D2.2.AV15 Data backup	- Availability (within this metric)
	REQ_D2.2.AV16 Data storage redundancy	
	REQ_D2.2.AV20 Data preservation	- Availability - Privacy-centric physical/tamper resilience - Policy updates
	REQ_D2.2.AV22 Data access control	- ECC Authentication - Digital signatures - Secure key distribution
	REQ_D2.2AV26	- Dynamic security behaviour - Conflict resolution between policy domains
Social Mobility and Networking Scenario	REQ_D2.1.1_2001.A Permanently reachability and location awareness of data	- Location Awareness
	REQ_D2.1.1_2005.A Security of data, REQ_D2.1.1_2006.A Privacy of data, REQ_D2.1.1_2007.A Dependability of data	- Data integrity - Data confidentiality
	REQ_D2.1.1_2010.A Communication through interconnected Networks	- Average reputation of network - information capacity - Network latency - Network delay

Deliverable *D2.8 Final SPD Metrics Specification* will detail and specify the real convergence of the binomial requirements-SPD metric. This table aims to link first requirements and specifications from D2.1 and D2.2 and preliminary SPD Metrics. As stated, refinement and implementation of this table will be developed in deliverable D2.8 Final SPD Metrics Specification.

8 Conclusion

Within this document, these main objectives have been achieved:

- Identification of main metrics for nSHIELD reference model
- Quantification for their measurement
- Classification of them (somehow a kind of prioritisation)
- Identification of main composition methods
- Links to scenarios and requirements

In the following paragraphs we list a number of comments concerning D2.5 as a whole.

Group of metrics

There are reported several groups or families of metrics that have to be evaluated together to correctly evaluate a functionality. The CC field is used for this purpose but in some cases it has been ignored.

Qualitative Metrics

The qualitative metrics remain an open issue. Usually, we denote them with '0' if they aren't used and with '1' if they are used. In most cases, the qualitative metrics can form a group or family of metrics to evaluate the level of security/dependability/privacy of functionality (like low, medium, high level). We could concentrate in forming a total quantitative group metric for such functionalities, based on these metrics, than considering them as individual qualitative metrics. For example, instead of using a standalone metric for node availability, it could be used in a group of metrics that determine the overall level of security/dependability.

Moreover, we have a few metrics which are linked to specific requirements but are not directly indicative of the nodes security levels (e.g. metric on the presence of a lightweight O/S). Likewise the presence of context awareness is only indirectly linked to security (e.g. enforces the use of stronger crypto mechanisms in untrusted areas) but also remains one of the important features we try to implement in nSHIELD. Metrics of this type (mostly defined as Boolean) obviously arose from the respective requirements and the whole "requirements should have a match in metrics" concept we tried to follow, but that does not guarantee the correct trace and implementation unless we continue iterating it in D2.8.

Attack on metric establishment mechanism

In nSHIELD we will use the metrics defined in D2.5 to configure the system at runtime. There would be thresholds for several metrics that will determine the normal or abnormal state of the system. Also, there will be metrics that will determine if the system is secure / dependable or not. Eventually, there have to be a mechanism to determine how the metrics are installed in a device at deployment and how it could be changed at runtime. Moreover, there have to be a way to take decisions on what technological features will be used based on the current level of security / dependability / privacy.

Except from determining the metrics themselves, attacks to these features should be taken into account. An attacker could compromise the system by controlling the threshold of a metric or by requesting the network nodes to use lower security components. For example in the first case, an attacker could change the threshold that is used by a reputation-based scheme and determine which users are legitimate or malicious. The attacker could increase the reputation threshold in order to keep malicious entities as legitimates or decrease the threshold to make all legitimate entities look like malicious and expelled from the network. In the second case the attacker could compromise the runtime configuration mechanism and make the nodes of a network to use smaller cryptographic keys.

These objectives will be extended and satisfied for the final deliverable D2.8 SPD Metrics Specification. The main research and working topics for this deliverable should focus on:

- Links to the architecture in order to define the interfaces to it
- Analysis in depth of the deployment of SPD metrics in the 4 scenarios of nSHIELD.
- A detailed design of interfaces with respect to the overlay layer.
- A refinement of SPD metric measurement formula taking into account particular scenarios specifications and technical requirements

We conclude that the document “Preliminary SPD Metrics” is a valid starting point that will become a bridge between requirements, architecture and scenarios. We also conclude that this is a not ended task that will continue with document D.2.8 by mainly defining integration mechanism towards nSHIELD architecture.

9 References

[1] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wolf, „**An Architecture-Based Approach to Self-Adaptive Software**“, IEEE Intelligent Systems, 14(3), p. 54-62, 1999

[2] 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

[3] Lippmann, R. *et al* (2005) **Evaluating and Strengthening Enterprise Network Security Using Attack Graph**, Technical Report, MIT Lincoln Laboratory, Lexington, MA, 2005.

[4] 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

[5] 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.

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

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

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

[9] 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.

[10] R. Sadeghi, “Metrics for optimal relay selection in cooperative wireless networks”, IEEE 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), 11-14 September, 2011, pp. 278–283.

[11] R. Serna Oliver, G. Fohler. Timeliness in Wireless Sensor Networks: Common Misconceptions In Proceedings of the 9th International Workshop on Real-Time Networks (RTN2010), Brussels, Belgium, July 2010.

[12] Z. Chen, K. G. Shin, “Post-deployment performance debugging in wireless sensor networks,” Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS), 2009.

[13] Y. Wang, M. Vuran, and S. Goddard, “Cross-layer analysis of the end-to-end delay distribution in wireless sensor networks,” in Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE, Dec. 2009, pp. 138–147.

[14] P. Chimento, J. Ishac, “Defining Network Capacity”, RFC 5136, http://www.rfc-editor.org/rfc/rfc5136.txt, 2008.

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

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

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

[18] Thomas Heyman, Riccardo Scandariato, Christophe Huygens, Wouter Joosen, Using Security patterns to combine security metrics, Availability, Reliability and Security, 2008. ARES 08. 1156-1163. IEEE