



PEOPLE
IAPP MARIE CURIE ACTION

Project acronym: CECRIS
Project full title: Certifications of Critical Systems
Grant Agreement no.: 324334

Partners:

1. Consorzio Interuniversitario Nazionale per l'Informatica
2. Resiltech S.r.l.
3. Universidade de Coimbra
4. Budapesti Muszaki es Gazdasagtudomanyi Egyetem
5. Prolan Iranyitastechnikai Zartkoruen Mukodo Reszvenytarsasag
6. Critical Software SA

***INTEGRATION OF SAFETY AND SECURITY ASPECTS INTO EXISTING
METHODOLOGIES***

Revision of the document:	01
Responsible partner:	BME
Contributing partner(s):	All
Reviewing partner(s):	
Document date:	21/10/14 (date of last revision)
Dissemination level:	PU/CO (Public/Confidential)

© Copyright 2013 CECRIS Project. All rights reserved

This document and its contents are the property of CECRIS Partners. All rights relevant to this document are determined by the applicable laws. This document is furnished on the following conditions: no right or license in respect to this document or its content is given or waived in supplying this document to you. This document or its contents are not be used or treated in any manner inconsistent with the rights or interests of CECRIS Partners or to its detriment and are not be disclosed to others without prior written consent from CECRIS Partners.

Each CECRIS Partners may use this document according to CECRIS Consortium Agreement.



Change Records

Revision	Date	Changes	Authors	Status ¹
01	21/10/14	Table of contents prepared	Ábel Hegedüs (BME), András Pataricza (BME)	NV
02	25/11/2014	First RT contribution to section 3.1, 4, 5 and 6	Davide Iacono (RT)	NV
03	09/12/2014	First CINI-FI Contribution to section 6	Nicola Nostro (CINI-FI)	NV
04	07/01/2015	UC contribution with secure development and detection of code vulnerabilities.	Nuno Antunes (UC), Cristiana Areias (UC), Marco Vieira (UC)	NV
05	12/01/2015	Prolan contribution to section 3.2	András Zentai (Prolan)	NV
06	20/01/2015	CSW contributions	Nuno Silva (CSW)	NV
07	26/01/2015	RT contributions	Arun Babu (RT)	NV
08	29/01/2015	Final version	Laszlo Gonczy (BME)	V

¹ V - Valid, NV - Not Valid, R - Revision

TABLE OF CONTENTS

1	INTRODUCTION.....	5
1.1	Aim of the document	5
1.2	Definitions and acronyms.....	5
2	SECURITY CHALLENGES IN THE DIFFERENT SAFETY-CRITICAL DOMAINS	6
2.1	Security challenges in embedded systems.....	6
2.1.1	Security Challenges in Aerospace Systems	7
2.2	Interaction between safety and security concerning certified systems.....	8
2.2.1	Security Threads Across Domains	8
2.2.2	Safety and Security Interaction.....	9
2.2.3	Security Issues Impacting Safety Certification	10
3	CO-MODELING SAFETY AND SECURITY	11
3.1	Analysis Goals	11
3.2	Modelling	12
4	DESIGN PRINCIPLES FOR OPEN EMBEDDED SYSTEMS.....	14
4.1	Security in the software development lifecycle	14
4.2	Challenges in designing secure embedded systems	15
4.3	CINI contribution	18
4.4	Space Systems Design Principles.....	19
4.4.1	Space Systems Telecommands and Telemetry	20
4.4.2	Housekeeping Telemetry	20
5	COMMON DESIGN REQUIREMENTS FOR SAFETY AND SECURITY	22
5.1	Developing secure code.....	22
5.2	Requirements for secure data transmission systems	23
5.3	Security requirements for embedded systems	25
5.4	Design Requirements for Safety and Security in aerospace Systems	26
6	TECHNIQUES FOR THE ASSESSMENT OF SECURITY IN SAFETY-CRITICAL SYSTEMS	28
6.1	Detecting code vulnerabilities.....	28
6.2	Threat Assessment & Remediation Analysis (TARA)	31
6.3	Smart Grid Information Security	34
6.4	Techniques for the assessment of security in aerospace systems	38
6.5	A Security Threats Library for Safety Critical Systems	38
6.5.1	The NIST Taxonomy	39
6.5.2	The Threats Library.....	40
6.5.3	Aerospace Case Study	40
6.6	Additional techniques for increasing the security of safety-critical systems	42
6.6.1	Secure OS hardening.....	42

6.6.2	Strong random bit generator	44
6.6.3	Privilege separation.....	45
7	REFERENCES.....	48

1 INTRODUCTION

1.1 AIM OF THE DOCUMENT

This deliverable aims at describing how the aspects of safety and security can be integrated when designing, developing or certifying critical embedded systems (ES).

The deliverable builds on the results of Task T1.2 that were described in Deliverable D1.2

In Section 3 we present a co-modeling approach where the aspects of safety and security are integrated into a joint error propagation modeling and analysis.

The challenges of security in safety-critical domains are elaborated in Section 2, with challenges listed from embedded systems and interaction between safety and security in the certification of ES.

To address the challenges, a collection of design principles for open ES are detailed in Section 3, while common requirements for safety and security are described in Section 5. Finally, techniques for assessing the security of safety-critical systems are described in Section 6.

The results of Task T2.3 will be used as input for Tasks T3.2, T3.3, T2.5.

1.2 DEFINITIONS AND ACRONYMS

Acronym	Meaning
ADCN	Avionics Data Communications Network
ADS-B	Automatic dependent surveillance-broadcast
D	Delivery
ES	Embedded systems
FAA	Federal Aviation Administration
FMEA	Failure Modes and Effects Analysis
HK	Housekeeping
NextGen	Next Generation Air Transportation System
SESAR	Single European Sky Air Traffic Management Research
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privileges
TC	Telecommand
TM	Telemetry

2 SECURITY CHALLENGES IN THE DIFFERENT SAFETY-CRITICAL DOMAINS

2.1 SECURITY CHALLENGES IN EMBEDDED SYSTEMS

Resp: industrial partners

RT contribution:

The most important threats that can be found across several domains (e.g. railway, airborne, automotive) mostly regard the communications among the subsystems which compose the systems [1]. For example, in the railway domain an attacker can send false train position information to the control centre, so that incorrect movement authorities are evaluated or, conversely, an attacker sends false movement authorities to the trains. In these cases a train accident can take place. In the airborne domain, instead, other kinds of introduction of fake messages can take place (e.g. ghost aircraft injection or virtual trajectory modification) in order to provoke accidents. In the automotive domain, fake messages manipulating traffic flow or simulating traffic jam can be performed in order to manipulate the cars routes. Anyway, in this domain more dangerous attacks wrt the people safety can be performed, for example attacking active brake function.

The scopes of security countermeasures are, therefore, to preserve data confidentiality, integrity and availability. Obviously, these three characteristics have a different priority with respect to other business IT application. Indeed, for Industrial Control System (ICS) integrity and availability of the information are more critical then confidentiality. For example, for train traffic control systems it is more important that trains positions are correct and available than the fact that an attacker discloses it.

The major challenges that it is worth to mention are that ICSs are mostly characterized by distributed architecture. Then, it is not cost effective to completely protect against physical access the overall infrastructure. This fact makes the system vulnerable to physical denial of service attacks leading to service interruptions.

A communication infrastructure is needed in order to link the components that compose the distributed architecture. The communication infrastructure is shared among multiple functionalities. The use of a shared communication infrastructure represents a vulnerability of the system, which can be exploited by attackers as a threat vector. In railway domain, the GSM network, which is used as a means of communication between the train and the ground system, can be exploited to perform false messages injections. In a similar way, the radio communications between ground systems and aircrafts can be forged.

Furthermore, systems are characterized by long life components with respect to business systems, which can be replaced in shorter replacement cycles, as in the automotive, airborne and railway domains. This characteristic has the drawback that long life components can become outdated in some technology aspects. Thus, an update to these systems is needed in order to increase their robustness against cyber-attack threats, but industrial control systems have more constraints with respect to other enterprise IT systems. Indeed, it is much more difficult to test and apply patches, because ICS affect life safety and involve systems that need to run uninterruptedly.

In addition, in case an attacker can have physical access to the devices, other kinds of threats have to be considered. Indeed, an attacker can replace parts of SW image compromising its integrity and forcing it to have unwanted behaviours. Another way to tamper the SW behaviours is to insert non-original peripherals. Alternatively an attacker can be interested in in the know-how and intellectual properties contained in the SW. Thus, the attacker can perform actions aimed at cloning or reverse engineering the SW. In other situations, instead, the attacker is more interested in disclosing information stored in the devices in order to perform some other attack steps.

As described above, there are several aspects to take into account concerning security and if not well defined, a designer can miss some of them leaving some vulnerability in the system. To this aim, some standardization effort in order to formalize methodologies aimed at tackling security issues has been done. One of these standards is the EN 50159:2010 [5].

European Standard EN 50159:2010 [5] is applicable to safety-related electronic systems in railway domain using for digital communication purposes a transmission system which was not necessarily designed for safety-related applications and which is:

- under the control of the designer and fixed during the lifetime, or
- partly unknown or not fixed, however unauthorised access can be excluded, or
- not under the control of the designer, and also unauthorised access has to be considered.

Both safety-related equipment and non safety-related equipment can be connected to the transmission system. The normative categorizes the transmission system in three categories according to the requirements that it satisfies:

- Category 1 (Closed Transmission System):
 - i. The number of pieces of connectable equipment, either safety-related or not, to the transmission system is known and fixed. As the safety-related communication depends on this parameter, the maximum number of participants allowed to communicate together shall be put into the safety requirement specification as a precondition.
 - ii. The characteristics of the transmission system (e.g. transmission media, environment under worst case conditions, etc.) are known and fixed. They shall be maintained during the life cycle of the system.
 - iii. The risk of unauthorised access to the transmission system shall be negligible
- Category 2 (Open Transmission System):
 - If a transmission system does not satisfy the preconditions (i) or (ii), but fulfils precondition (iii) it shall be considered as Category 2.
- Category 3 (Open Transmission System):
 - If a transmission system does not satisfy the precondition (iii) it shall be considered as Category 3.

The main hazard to safety-related communication is the failure to obtain a valid message in terms of authenticity, integrity, sequence and timeliness at the receiving end. Hazardous events identified may include the following systematic failure: broken wires; cabling errors; antenna misalignment; performance loss; HW random failure and ageing; human error; maintenance error; EMI; cross-talk; thermal noise; fading effects; overloading of transmission system; magnetic storm; fire; earthquake; lightning. In a same way also deliberately-caused events should be considered such as: wire-tapping; damage or unauthorised change to HW; unauthorised change to SW; monitoring of channels; transmission of unauthorised messages. However, although there can be a wide range of hazardous events, the basic message errors, which form the threats to the transmission system, are one of the following:

- repetition;
- deletion;
- insertion;
- re-sequencing;
- corruption;
- delay;
- masquerade.

2.1.1 Security Challenges in Aerospace Systems

Commercial aerospace is currently undergoing huge changes. There is increased complexity and interconnection between aerospace systems, which raises new challenges related to both safety and security. The main systems presented in a modern commercial aircraft include cockpit, flight

control, engines, fuel, power supply and landing gear systems. These different systems are interconnected by the Avionics Data Communications Network (ADCN). An ADCN can contain up to three network segments: control, crew and passenger networks, the security of which must be ensured for safe flight operations.

Programs such as the “Next Generation Air Transportation System” (NextGen) or the “Single European Sky Air Traffic Management Research” (SESAR) aim to optimise several aspects of the current airspace system, which will have an impact on both Airborne and Ground systems operating at all points of the flight, from take-off to landing. The more integrated functions of the new Programs, such as Integrated Flight Planning, Surface Traffic Management, Enhanced Pre-departure Clearances, Streamlined Departure Management and Efficient Cruise, will make the overall flight mission be more reliant on various software applications. The protection of these applications from any intentional abuse or attacks has been increasingly recognized as an important area.

It is an urgent and undeniable task to ensure the information confidentiality, data integrity and service availability of critical aerospace systems against malicious intents from potential adversary parties. The security challenges in aerospace systems lie in the following perspectives:

- Vulnerability of existing and future aeronautic and space systems. The potential weakness of the various complex aerospace systems that may be exploited or abused by attackers needs to be considered in depth. For example, weak signals may be easily jammed; unencrypted data can be accessed without authorization.
- Potential threat types and attack scenarios for aerospace systems. Effective security assessment starts from a comprehensive understanding of potential threats to systems. Specialized analysis of threat types and attack scenarios in aerospace sector is necessary for integrating existing security knowledge and experience in other domain to another application context and supporting security requirement specification in aerospace software development.
- Applicable security mechanisms or counter measures against attacks. Critical aerospace systems usually have more stringent requirements than ordinary web applications or enterprise information system, e.g. short response time, inadmissible jitter or delay, unacceptable power-down or reboot, strong fault-tolerance capability. It is a great challenge to have effective mechanisms adopted, e.g. monitoring techniques, segregation strategies, but with limited negative effects on the system performance introduced.
- Secure software development support. The development process and better programming language support should be studied in order to have security protection built-into the ways that aerospace programs are designed and written. More exploration on security testing techniques on aerospace software systems should be carried out, through which the vulnerable features of software may be detected before software is released and installed.
- Secured data communications between system interfaces. The data flowing over various routes on the airborne networks or space communication links should be protected so that the information cannot be diverted, monitored, or altered. Some of current protocols for directing data traffic can be exploited to make messages appear to come from someplace other than their true origin.

2.2 INTERACTION BETWEEN SAFETY AND SECURITY CONCERNING CERTIFIED SYSTEMS

This section should refer back to D1.2 and contain a shortened version of its conclusions.

The aim of the section is to give a concise view of the security related challenges as a motivation and background for the contents of subsequent sections.

2.2.1 Security Threats Across Domains

Based on the STRIDE approach a previous assessment of the security threats across all industrial domains considered in the CECRIS project was completed in the framework of the WP1. In framework of WP2 the data of the relevant security threats were processed in order to show which

domains are concerned for different type of attacks. This results are presented in Table 1. The results confirm that Denial of Service, Tampering and Spoofing are the most relevant type of attacks meanwhile information disclosure is not a serious issue in case of ICS.

STRIDE Classification	Domain	Threat Description
Denial of Service	Airborne, Space, Automotive, Railway	Jamming and flooding ground station, VLAN flooding attack, Flooding signals to satellite, Fake correspondent node addresses, Unauthorized Brake, Attacking Active Brake Function, Attacking E-Toll, Head Unit Attack, Flashing per OBD, WLAN Attack, Disturbing passenger Information system, GSM-R Attack (DoS - Denial of service)
Elevation of privileges	Airborne, Automotive, Railway	VLAN Tagging attack, Attacking E-Toll, Force Green Wave/Getting traffic lights green ahead of the attacker, Flashing per OBD, E-Call, Manipulate Speed Limits, Manipulate Traffic Flow, Database attack
Repudiation	Automotive	Engine DoS-Attack (Engine Refuse to Start)
Spoofing	Airborne, Space, Railway	Spoofing attacks on the Automatic Dependent Surveillance – Broadcast (ADS-B) system, Fake correspondent node addresses, Spoofed binding updates, Fake/Modified telecommands delivered to satellites, WLAN Attack
Tampering	Airborne, Automotive, Railway, Space	Interference in communications, Tampering GPS coordinates, Tampering attacks on ADS-B, Head Unit Attack, Simulate Traffic Jam, Tamper with Warning Message, Flashing per OBD, Manipulate physical components, Manipulate signalling components, GPS data falsification, Disturbing passenger Information system, Tampering satellite Software updates
Information disclosure	None	None

Table 1. Classification of threats accross domains

2.2.2 Safety and Security Interaction

Conception of safety and security could be interpreted in a very similar way as it is illustrated below [27].

Definition of Safety Engineering:

the engineering discipline within systems engineering concerned with lowering the risk of unintentional unauthorized harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, mishaps (i.e., accidents and incidents), hazards, and safety risks.

Definition of Security Engineering:

the engineering discipline within systems engineering concerned with lowering the risk of intentional unauthorized harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, misuses (i.e., attacks and incidents), threats, and security risks.

Differences between Safety and Security definitions:

- Unintentional vs. Intentional
- Accidental vs. Malicious Harm
- Mishaps vs. Misuses
- Hazards vs. Threats

Extending safety artifacts (e.g. fault tree) with faults caused by malicious intentional failures could be a feasible approach to assess the effects of security threats on the safety of the system.

2.2.3 Security Issues Impacting Safety Certification

One of the conclusions of the safety and security interaction assessment in D1.2 was that the most relevant causes of security and safety interaction are the followings:

- Conflicting safety and security requirements,
- the safety mechanism does “open” a vulnerability compromising the very same safety mechanism,
- the safety mechanism does “open” a vulnerability compromising other same safety mechanisms being part of the safety concepts.

A proposed solution to the problem of conflicting requirements was to consider a trade off between the safety and security.

The last two causes came from the architecture definition level of the lifecycle. Proposed solutions were mainly to consider relevant type of attacks during the design phase of the system components and consider the appropriate tests during the assessment.

3 CO-MODELING SAFETY AND SECURITY

Resp: András Pataricza

This section describes a joint error propagation modelling approach for safety and security analysis (first described in [8]).

In the analysis, security hazards and attacks are integrated into the existing approach for software or hardware faults for safety analysis.

3.1 ANALYSIS GOALS

The analysis method ([9]) aims at finding cost-optimal solutions for services built on a combination of SW and HW elements. The method considers the structure/topology of the system under analysis, services/functionalities offered by the system, error modes of services and underlying resources, and preventive/corrective mechanisms which improve the fault tolerant characteristics of the system.

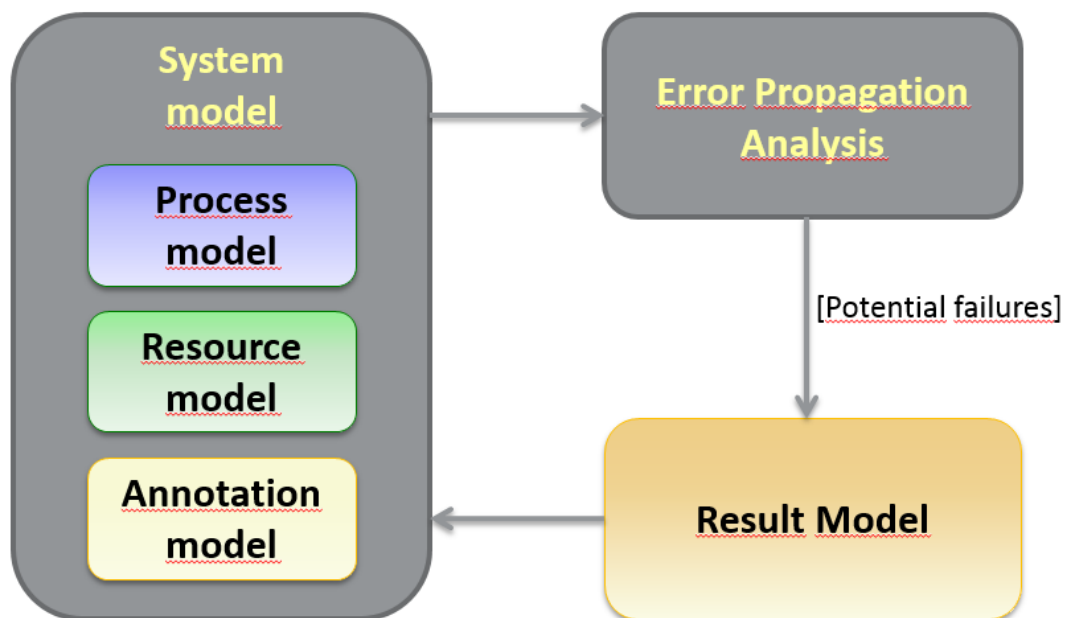


Figure 1 Models in SW FMEA

This analysis serves multiple purposes, depending on the level of details in the input:

- As a basic feature, it can detect SPOFs at the system level. The method can be considered as an architecture level FMEA (or FMECA) tool.
- Once cost/loss values are added to the model, the method can find a) cost-optimal solutions for strengthening a given system against a set of faults (fault modes), b) determine the maximum level of protection which can be achieved, or c) calculate the cost of protection/non-protection of a given system configuration.
- By adding extra information to the system model by annotations (like “this component should be treated as fault-free” or “this input may be erroneous”) partial FMEA can be executed. This supports early system design/integration of COTS/OSS components (by concentrating the analysis on a set of system components/functionalities) and determine the minimum required level of robustness for third-party provided elements. On the other hand, trivial/non-relevant faults can be eliminated. Annotations can also represent the result

of previous analysis rounds, therefore they enable iterative execution of SW FMEA (the automation of this is under development).

- The result of this analysis can be used in system synthesis as dependability bottlenecks are not only identified, but also suggestions are returned to reduce the effect at component level faults (also covering interface/human faults).

3.2 MODELLING

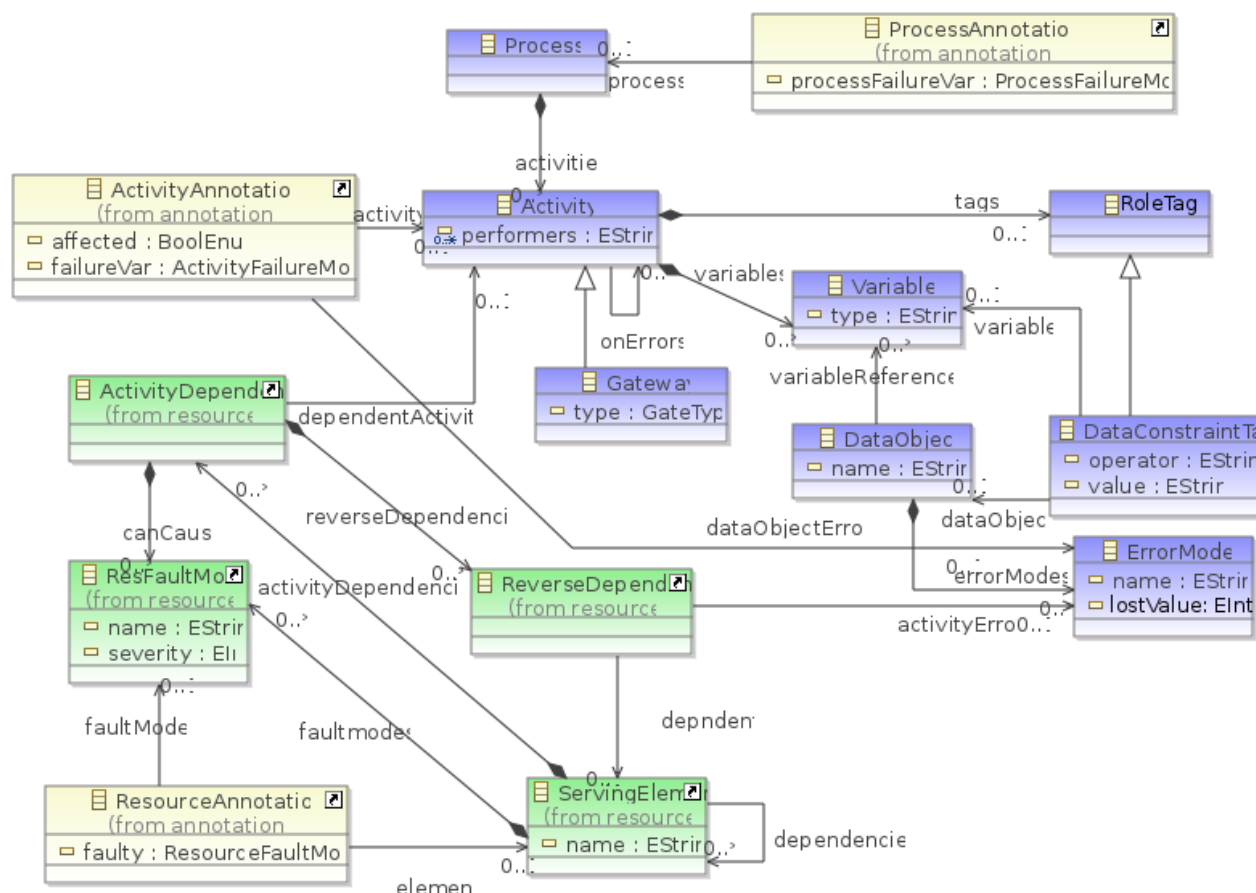


Figure 2 Metamodel of error propagation tool

Figure 2 shows the metamodel of the concepts used for software/hardware analysis in Eclipse Modeling Framework (EMF). We assumed that the operation of the system is captured by process models which rely upon a restricted number of BPMN ([10]) elements. Currently all control constructs are modeled as sequences or a combination of gateways.

Resource usage and dependency of SW services on HW elements is captured by assigning resources to each activity in such a process (where process is not limited to classic “business processes”). Both resources and activities can be annotated to model assumptions on their initial state, coming either from expert judgement or a result of a previous analysis. Fault tolerant behaviour is captured by a set of error propagation rules attached to each component.

The method is able to handle both safety and security aspects, as the underlying mechanisms can process security treats similarly to dependability issues.

Results of the analysis include a set of resources/activities with (error) states which in combination may lead to system level failure. This set/list can be used also to determine the potential loss value; on the other hand, corrective measures can be extended with a cost function which represents their initial/operation cost. The method can lower the verification cost and can be applied

in combination of standard FMEA techniques as it can determine input for those. As ongoing research, also the internal models of components can be co-analysed if these are described in an executable modelling formalism (e.g. fUML, [11]).

The current tool was implemented on the basis of constraint solving techniques implemented by the Choco Java library[12]. Modeling mechanism is based on EMF and can be easily adapted to process/resource modelling notations.

Applicability of the method was investigated by external researchers in Quanopt Ltd. an illustrative process of the automotive domain was described and analysis according to a given set of safety goals.

4 DESIGN PRINCIPLES FOR OPEN EMBEDDED SYSTEMS

Resp: ALL

4.1 SECURITY IN THE SOFTWARE DEVELOPMENT LIFECYCLE

Resp: UC (Marco Viera)

A software development process is composed of multiple phases [28]. To develop secure software it is important not only to focus on secure coding, but also to take a broader view by integrating existing approaches and tools in the development process, i.e. to use such approaches and tools in the points of the process where they can make the difference. Different authors divide the software process in different ways, but usually software development includes the following phases (which can be repeated in an iterative manner): initialization, design, implementation, testing, deployment and decommissioning. The process starts with requirements gathering (including security requirements), followed by specification and design, implementation (coding), testing and deployment. Decommission takes place when the product is not useful/used anymore.

In critical systems, the required software quality level and trustworthy are high, demanding the use of V&V techniques throughout the whole system development cycle. Since faults can be introduced at every stage, from inception to deployment, including operation and maintenance, V&V should be planned in parallel with development activities. V&V activities will check the product against the functional requirements, to assess the non-functional attributes, and also to reduce costs by diminishing rework on correcting faults at later stages [29].

The traditionally used V-Model [30] represents an extension of the traditional waterfall model and includes an additional set of explicit phases related to V&V as the outcome of each of the development activities. **Figure 3** provides a simplified representation of this model (again, different authors may consider more or less steps in each side of the V). Each development step also defines the V&V activities to be done, which may be assigned to programmers, team leaders, testers or users. With this integrated V&V process, it becomes easier to detect and prevent faults, thus increasing confidence on the final product. Basically, the software V&V activities analyze, review, demonstrate or test all software development outputs.

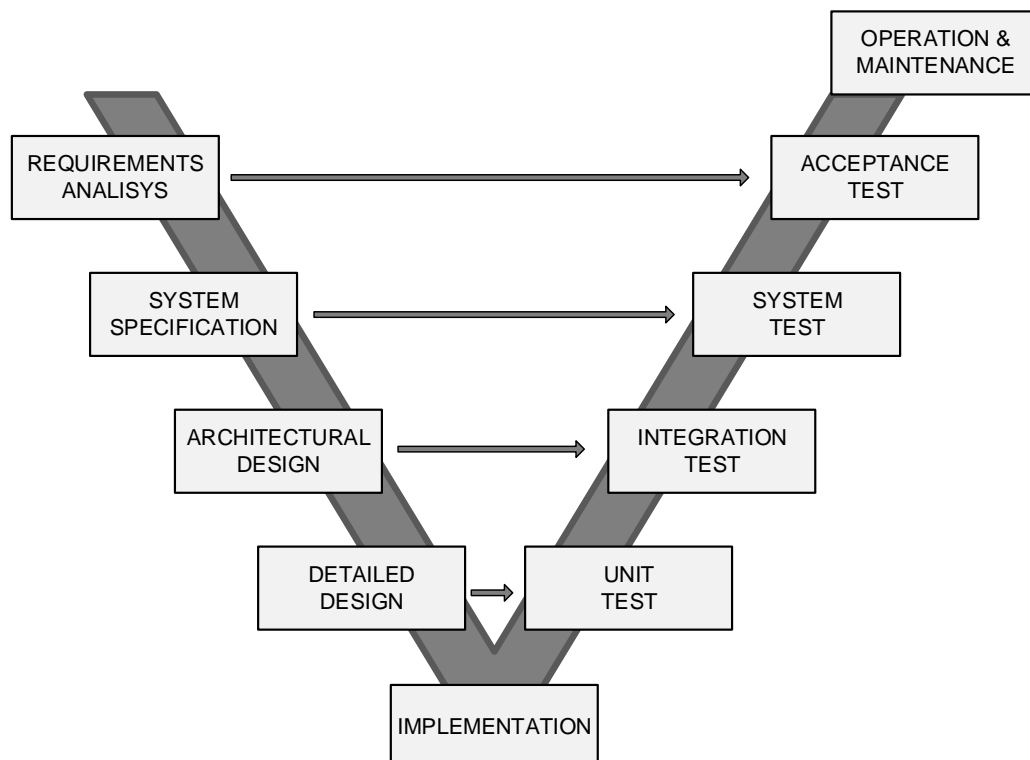


Figure 3. An example representation of the V-Model.

Although code security concerns should be addressed during the entire software product development lifecycle, as highlighted by [31] especial focus should be put in three key phases [32]: *implementation, testing, and deployment*. The next points summarize the main challenges and discuss these three phases in the context of the V-Model:

- **Implementation:** during coding we must use best practices that avoid the most critical vulnerabilities in the specific application domain. Examples of practices include input and output validation, the escaping of malicious characters, and the use of parameterized commands [33]. Vulnerability and attack injection techniques [34] have in this phase a very important job in the evaluation of the best security testing tools to use. Also, for the success of this phase, it is essential to adequately train the development teams. For instance, experience shows that the main reason for the vulnerabilities in application's code is related to training and education. First, there is a lack of courses/topics regarding secure design, secure coding, and security testing, in most computer science degrees [32]. Second, security is not usually among the developers' main skills as it is considered a boring and uninteresting topic (from the development point-of-view), and not as a way to develop new and exciting functionalities.
- **Testing:** the most used V&V technique and may be carried out at different granularity levels, from single units to the whole system (ascending part of the V-model, as shown in **Figure 3**). There are many security testing techniques available for the identification of vulnerabilities during the testing phase [33]. To mitigate vulnerabilities, it is necessary to have well-trained teams that adequately apply those techniques during the development of the application. The problem is that software quality assurance teams typically lack the knowledge required to effectively detect security problems. This way, it is necessary to devise approaches to quickly and effectively train security assurance teams in the context of applications development, by combining vulnerability injection with relevant guidance information about the most common security vulnerabilities.
- **Operation/maintenance:** it is possible to include in the environment different attack detection mechanisms, such as Intrusion Detection Systems (IDS), among others, that protect the application after deployment, i.e. during operation/maintenance phase (see **Figure 3**). These mechanisms can operate at different levels and use different detection approaches. The main problems preventing their use are related to the performance overheads and to the false positives that disrupt the normal behavior of the system. In this phase, security benchmarking plays a fundamental role in helping to select the best alternatives (in terms of servers, security mechanisms, etc.) to use, according to specific security requirements. Also, vulnerability and attack injection techniques represent in this phase an efficient way to evaluate the effectiveness of attack detections mechanism to be installed.

Although most of the security problems in computer systems are related to unknown attacks and vulnerabilities, there is a substantial share of the problems that are due to vulnerabilities and the attacks that widely known. In most of the cases, applying the best coding practices (see Section 5.1) and thorough security testing/vulnerability detection processes (see Section 6.1) can reduce the effectiveness of most of the known attacks. However, in the case of the unknown attacks, although some anomaly detection systems can catch one, often the only alternative is to follow best design and coding practices.

4.2 CHALLENGES IN DESIGNING SECURE EMBEDDED SYSTEMS

RT contribution:

One of the biggest difficulties in designing secure systems is related to the fact that security is, usually, much more dependent on what is unknown about the system properties (e.g., the existing vulnerabilities and threats) and its potential attackers than by what is known about them. [1].

Several challenges should be taken into account when designing a secure embedded system [2]. First of all there is a computational gap that should be taken into account. Existing embedded system architectures are not capable of keeping up with the computational demands of security processing, due to increasing data rates and complexity of security protocols. This is particularly critical in case of systems that need to process very high data rates.

Another gap that should be considered is the battery one. The energy consumption overheads of supporting security on battery-constrained embedded systems are very high.

The design should be flexible. An embedded system is often required to execute multiple and diverse security protocols and standards in order to support multiple security objectives (e.g., secure communications, etc.), interoperability in different environments (e.g., a handset that needs to work in both 3G cellular and wireless LAN environments), and security processing in different layers of the network protocol stack (e.g., a wireless LAN enabled PDA that needs to connect to a virtual private network, and support secure web browsing may need to execute WEP, IPSec, and SSL). Furthermore, security protocols are continuously evolving, since they are constantly targeted by hackers. It is, therefore, desirable to allow the security architecture to be flexible (programmable) enough to adapt easily to changing requirements. However, flexibility may also make it more difficult to gain assurance of a design's security.

Embedded systems should be designed to be tamper resistant. An attack can be HW, SW or both. Several countermeasures can be used to increase the robustness of the system against attacks. Indeed, since we are dealing with open embedded system, the design of such systems should foresee countermeasures against attackers which are interested in injecting modified SW in order to tamper its behaviour. In this case, SW authentication can be used. In this case the design should foresee an architecture able to sign the SW and verify the signature of the SW.

The example reported in [3] and shown in Figure 4 shows a possible solution for SW authentication using asymmetric encryption. In the example the SW developed in a secure off-line environment is firstly hashed and then signed with a private key. Once signed, the SW and its signature are uploaded on the flash memory. The SW, before being run, is thus verified using the public key which is stored in the systems.

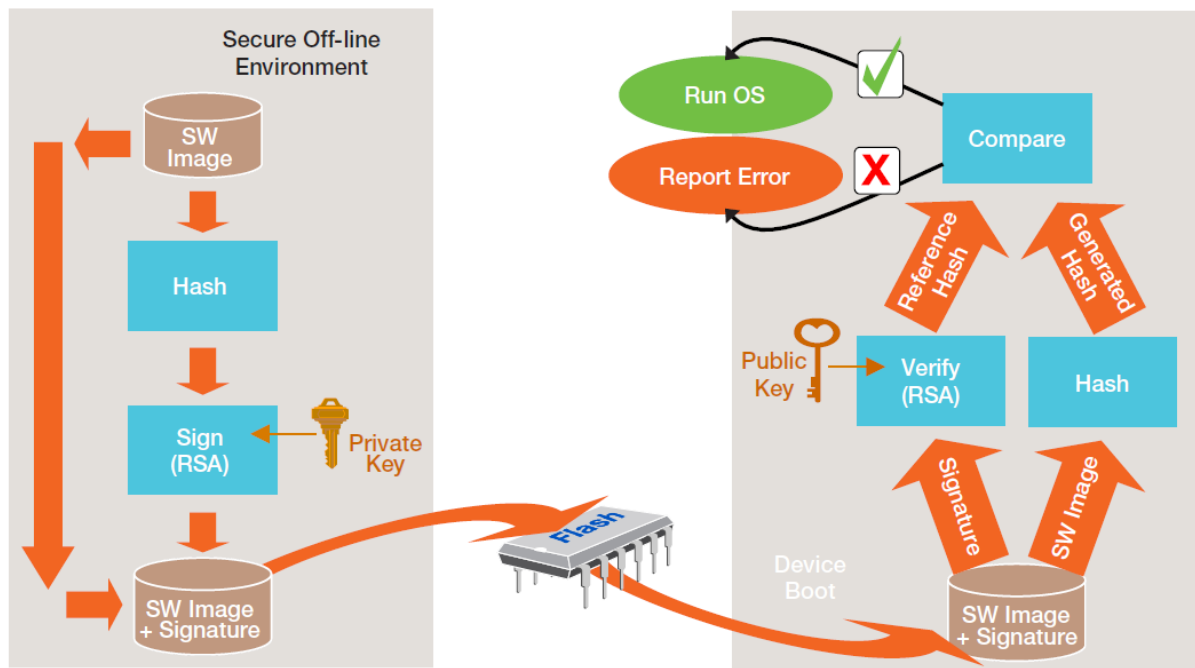


Figure 4 Flash Programming Protocol

It is worth noting that in this solution both code and public key are not encrypted. In some cases, anyway, it is also required to protect the know-how and intellectual property against cloning and reverse engineering [4]. A starting point for protecting the software image is by means of code encryption, such that the attacker doesn't have access to the encryption key. Access must also be prevented to the plain code itself once decrypted, when it is located in on-chip RAM.

The encryption key used to encrypt the software image must be stored using a Secure Key Storage mechanism. At the basic level, key storage can be with on-chip ROM (less flexible) or One-Time-Programmable (OTP) bits.

Once the image is ready encrypted on flash, there is a need for fast code decryption at runtime. This refers to the capability to decrypt the code as it is being read from external storage and into on-chip RAM.

An attacker might also try to extract IP by probing internal RAM, where the code resides in plain form, using the chip JTAG interface. In this case a protection of the JTAG debug interface is needed. Debug port protection is carried out by a dedicated Secure JTAG unit, which resides on the JTAG entry point. Whenever a technician is interested in debugging the chip, the technician uses the debug tool to ask the on-chip Secure JTAG unit to send a challenge (based on a cryptographic scheme). The challenge is received by the debug tool and if it is an authorized debug tool then the debug tool is able to create the appropriate response to authenticate itself. The secret for the authentication protocol must be provisioned to the chip at an earlier stage, similar to key provisioning described above. Once authenticated, the Secure JTAG unit opens the JTAG chains and enables debugging of the chip. An attacker would have a hard time accessing JTAG chains and internal RAM when such a mechanism is in place.

Attackers can be also interested in disclosing confidential data which are stored in the embedded systems. Some pieces of information require only protection against modification (e.g. identity), while other items in the database also require protection against reading (e.g. private keys). Attacks on the database can occur off-line or on-line. An off-line attack involves reading or modifying the database on the external storage device (most likely flash) while the

system is powered-down. In order to prevent reading database items, the database should employ encryption methods to provide data confidentiality. In order to prevent modifications of database items, the database should employ digest methods to provide data integrity. Both the confidentiality and the integrity of the database are based on encryption keys and therefore require Secure Key Storage capability.

Having in mind that in most cases this kind of systems are characterized by a distributed architecture, then each of the components/subsystems needs to communicate with the others. The communication channel among the components represents a threat vector that can be exploited by attackers in order to perform some malicious action. Attacks can be passive, listening to messages that are sent into the communication network trying to disclose their content, or active, inserting fraudulent messages (e.g. masquerade attack, man in the middle attacks, etc.). Therefore, the embedded system primarily needs to ascertain the identity of the other peer. This requires the peer to have a non-forgeable identity. Utilizing public key cryptography methods would provide the most effective way to address this requirement. The target peer holds a private key and the system is provisioned with the matching public key. Using the public key, the system is then able to authenticate the target peer. The public key must be kept in a Secure Key Storage, otherwise the attacker might replace it and lead the system to believe it is communicating with a different peer. In order also to preserve confidentiality of the information contained in the messages, the message can be encrypted through symmetric cryptography. In this case, both the sender and receiver share the same key used to encrypt the message. Thus, this key must be kept in a Secure Key Storage, otherwise an attacker could be able to disclose message content or even send fraudulent messages that would be decrypted by the receiver.

4.3 CINI CONTRIBUTION

Safety-critical systems design is subjected to strict regulations and guide lines, well defined by specific standards in the domain of interest, as described in Deliverable 1.2 [19].

Safety in the context of embedded systems deals with minimizing the frequency of mishaps (especially loss of life, injuries, and damage to property). Reliability in embedded systems has been studied for many years, and has to do with ensuring that once an embedded system starts a "mission", it has a high probability of completing that mission without experiencing a failure.

Networking connections makes these systems unsecure, thus exposing them to potential intrusions and intentional malicious attacks. When these systems become vulnerable from the security point of view, thus liable to malicious attacks (both from inside and outside), we have seen that current design principles, especially the reference standards, are actually lacking in addressing these security issues. The same considerations holds true for open embedded systems, which increasingly pervade our lives.

Therefore, today security, together to safety, represents one of the major issue to be tackled during the design of such kind of systems. Nevertheless, there are not approaches, techniques and/or methodologies able to face in combination both security and safety. At the same time trying to directly apply security techniques to safety-critical systems is not so simple. Indeed, just to give an example, embedded system in safety contexts have often strict timing constraints, which represent a critical requirement, thus implementing crypto mechanisms to messages in order to improve the security of the system, can lead to a violation of the timing requirements, due to the computational effort needed of the encryption and decryption operations. Consequently, it is of paramount importance to be able to identify a proper tradeoff between safety and security throughout embedded systems' design.

Specifically, techniques of threats analysis could be necessary to understand proper security countermeasures to be implemented on the system and whether/how they could impact on safety aspects. Such an approach allows thus to have a much clear insight of the system with respect to security and safety requirements. A preliminary methodology that goes in this direction is proposed in Section 6.

Although so far there is not a wide number of security violations in this kind of systems, in contrast to big companies, especially ICT ones that every day have to deal with security problems, it is clear that security aspects cannot be neglected when designing embedded systems. At the same time it is important to understand how to deal with security at design time and the challenges to face with.

The real issues that need to be tackled are described in the following and are related to the characteristics of embedded systems:

- **Development Environment.** Many embedded systems are created by small development teams. They are usually composed by domain experts, which have the proper skill and knowledge in that specific domain, thus allowing a successful realization of the product. At the same time such domain experts do not have an adequate expertise in dependability and security. Specifically, this problem could be identified within small companies that cannot afford dependability and security specialists or sometimes they do not realize they need this role within the company.
- **Costs reduction.** In order to be competitive in the market, producers pose often attention on embedded systems costs. Even some cents can lead to a significant impact when building millions of unit per year. For example many companies use 4- and 8-bit microcontrollers, which are not able to store big cryptographic key, thus avoiding to implement proper security solutions.
- **Energy constraints.** As mentioned embedded systems have often strong limitation with respect to energy and usually they are battery powered. Particular problem can arise for battery powered systems, especially for those system where battery is not expected to be replaced daily. In this last case, attackers can lead to a system failure (e.g. DoS) simply by continually querying the device thus draining away the battery.

Timing constraints. Embedded systems, such as in cars, use a network to coordinate control actions in real time. Timing constraints represent a critical aspect of similar systems, hence implementation of security countermeasures may have a high negative impact on time.

4.4 SPACE SYSTEMS DESIGN PRINCIPLES

CSW contribution

Although these are not commonly open system, the design principles are common and transversal, as well as applicable to open systems, depending on the requirements.

Space systems need to fulfil a set of safety and survivability related requirements that are implemented in generic design principles, including at least:

- Real-Time systems (forbidden usage of dynamic memory allocation, resources constrained, low power consumption, strict timing constraints, needs to deals with synchronous and asynchronous events and outputs, predictability is important, usually based on well known scheduling algorithms, hardware and software are tightly coupled);
- A certain level of autonomy (error detection, correction, recovery);
- Planned actions with or without human intervention;
- Regular housekeeping reports (provide status of health of the on-board systems);
- Highly reliable communications (Telecommands, Telemetry);
- Redundancy (Hardware, voting, etc.);

- Watchdogs (Hardware or Software, to avoid the system from crashing or becoming irresponsible at any time);
- Dump and Patch capabilities (to investigate problems and to correct software or hardware issues while the system is operating);
- Energy/battery management.

For the purpose of this subsection we will only focus on one of the previous design principles that can clearly be influenced by safety and security issues, the housekeeping principle.

4.4.1 Space Systems Telecommands and Telemetry

The importance of communications is undeniable for space or remote space systems. These are based on the delivery of telecommands and the reception of telemetry. In one direction (ground to space system) the communication link is used either for routine programming (commercial imaging requested by Spot Image) or for sending commands to carry out specific actions to handle events as required (orbital manoeuvres, equipment tests, anomalies, failures, etc.); this is the telecommand (TC) link. Although modern satellites operate automatically, they still need to receive commands from the ground. This need is particularly obvious during the satellite attitude acquisition phase. During this critical phase, the satellite needs to be very closely controlled from the operations control centre. Once the solar arrays have been automatically deployed, telecommands sent by the control centre switch on the equipment that was off during the launch: recorders, payloads and passengers.

On the other direction (space system to ground), the communication link is used to monitor the satellite through status reports and anomalies detected by the on-board computer; this is called the telemetry (TM). Telemetry is a set of measurements taken on board the space system and then sent to the operations control centre (ground). The measurements can describe the satellite, subsystem by subsystem. Measurements concern magnitudes as varied as temperatures, voltages, currents, etc. For example, if we consider the solar array subsystem, we need to know the output voltage and current at all times. The telemetry is also used to dump memory areas of the on board system and scientific data, although it has limited throughput for this purpose.

Data transmitted from a satellite during satellite's ground contact involve not only image data but also housekeeping telemetry data including information about satellite health which comprises a set of satellite status indicators and sensors read outs such as electric current of the on-board equipment and temperature, etc.

4.4.2 Housekeeping Telemetry

Commonly, a selection of parameters is defined in the space systems requirements to be inserted into the cyclic housekeeping (HK) or diagnostic TM packets. These HK parameters are inserted in streams of bits that will compose the TM packets. The frequency, number and contents of the HK TM packets are configurable and modified from ground through telecommands.

The HK telemetry is used to monitor the status of the on board system, and then to help in detecting and correcting any problems or deviations. Both the TC and TM are essential to maintain the safety and functioning of the systems, any security breaches can bring quite severe impacts on the decisions made and on the system itself.

Example of a housekeeping telemetry from ESTCUBE (<http://www.estcube.eu/en/telemetry-packet-description>):

COM housekeeping data

01 06 00 19 00 05 00 15 0E 00 00 00 00 00 AF 00 00 E6 1A 00 00 E0 1A 00 00 26 03
00 00

Frame header:

01 06 00 19

- Source: COM, 01
- Destination: GS, 06
- Length: 25 Bytes, 19

Command header:

00 05 00 15

- Command ID: COM housekeeping data, 05
- Length: 21 Bytes, 15

Command parameters:

0E 00 00 00 00 00 00 AF 00 00 E6 1A 00 00 E0 1A 00 00 26 03 00 00

- Number of reboots (uint16): 14, 0E 00
- Downlink temperature (int16): 0, 00 00 (unimplemented)
- MCU temperature (int16): 0, 00 00 (unimplemented)
- RSSI (int8): -80, AF
- AFC (int8): 0, 00 (unimplemented)
- Packets sent (uint32): 6886, E6 1A 00 00
- Correct packets received (uint32): 6880, E0 1A 00 00
- Broken packets dropped (uint32): 806, 26 03 00 00

5 COMMON DESIGN REQUIREMENTS FOR SAFETY AND SECURITY

5.1 DEVELOPING SECURE CODE

UC contribution (Marco Viera)

To mitigate security threats it is necessary to apply the best coding practices and to perform specialized security testing in order to develop non-vulnerable code [33]. Before applying vulnerability detection techniques, developers should follow the coding practices that are widely accepted as suitable to produce code without vulnerabilities. The recommendation is more often that not a *defense-in-depth approach* [32]. This approach assumes that each security precaution can fail and so, security depends on several layers of mechanisms that cover the failures of each other. Developers are expected to apply the effort needed to put in place adequate security precautions that minimize the probability of successful attacks. Usually it is necessary to consider at least three distinct lines of defense that can be used against threats.

The **first line of defense** consists in reducing the input domain of the application as a whole, acting directly on the values provided by the users. This is frequently called **input validation** [35] and consists in forcing the input parameters of an application to be within the correspondent valid domain or to interrupt the execution when a value outside of the domain is provided. Basically, every external input must be considered as malicious until proven otherwise, even the ones coming from other modules/components of the same application/system. This starts with the normalization of the inputs, for instance in terms of character set and encoding. Then, filtering strategies must be applied over the normalized inputs, rejecting the ones that contain values outside the valid domain. This is considered to be a good practice that may avoid many problems in applications' code. Input Validation can also be performed using **positive pattern matching** or **positive validation**. In this case, the developers establish input validation routines that identify the inputs to be accepted, contrarily to the previous case. This technique may be advantageous in some cases as developers might not be able to predict every type of attack that could be launched against their application, but should be able to specify all the forms of legal input.

A key issue is that input validation is frequently not enough as the data domain of an input parameter may allow the existence of vulnerabilities, independently of the validation performed. Examples of this are injection vulnerabilities, in which a quote is the character used as a string delimiter and can be used to perform Injection attacks. But, in some cases the domain of a string input must allow the presence of quotes. This way, we cannot exclude all the values that contain quotes. This means that, in this case, additional security mechanisms are necessary.

This additional security represents the **second line of defense** and it is necessary to complement the limitations of a general input validation strategy. In practice, each type of attack to an application targets a specific set of statements of code of the application that are prone to specific types of vulnerabilities. The second line of defense focuses on protecting these lines, for instance by guaranteeing that the values actually used lie within their input domain.

Coming back to the example of Injection vulnerabilities, which frequently use single and double quote characters exist in the majority of attacks. Thus, some programming languages provide mechanisms for **escaping** [36] this type of characters in such way that they can be used within an expression rather than delimiting values in the statement. However, this kind of techniques has two main problems. First, it can be circumvented by some more elaborated attacks, like combining quotes (') and slashes (\). Second, the introduction of characters for escaping increases the length of the string and thus can introduce later problems like unexpected data format or even data truncation. An alternative technique to implement the second line of defense is using correctly **prepared statements** (also named parameterized queries) [36]. When a prepared statement is created (or prepared) its structure is sent to the resource responsible by its execution. The variable parts of the query are marked using placeholders. Afterwards, each time that the query needs to be executed, the values are binded to the corresponding placeholder. No matter what is the content of the data, the expression will always be used as a value and not as part of the command. To help ensuring the correct usage of the data, many languages allow typed bindings.

A **third line of defense** is the **output validation** [35], which refers to the process of validating the output of a process before it is sent to some recipient, preventing it from receiving information that should not be received, like information about exception inside the application that can help conducting other attacks. Another example is searching the output of an application for critical data (e.g. credit card numbers) and replacing them with asterisks (*) before sending to the recipient. Output encoding is a type of output validation and it is a mandatory precaution to avoid XSS vulnerabilities [36]. If the data sent to the browser are to be echoed in a web page, then that data should be correctly encoded (depending on the destination in the page, either in HTML encoding or percent encoding). This way, even the malicious characters used in XSS attacks become innocuous while preserving its meaning.

5.2 REQUIREMENTS FOR SECURE DATA TRANSMISSION SYSTEMS

RT contribution:

Design requirements that should be taken into account in order to increase the security and safety regards several aspects. First of all, it is worth mentioning that security threats (possible dangers that might exploit a vulnerability to breach security and thus cause possible harm) may impact safety requirements in safety critical systems [1]. Although a security vulnerability by itself does not pose any significant safety risk (e.g., not using encryption in data communication is not a safety issue), by exploiting it an attacker may cause the system to fail (e.g., by tampering not encrypted messages the attacker may cause the system to perform invalid operations). On the other hand, implementing security mechanisms, like the ones mentioned above, may also impact the safety functions (e.g., forcing authentication may delay the access to some safety function in a critical situation). Therefore, a compromise between safety and security requirements should be reached.

Safety requirements mainly depend on the scope of the system. For example, in the railway domain, a safety requirement could be the correct and timely evaluation of the movement authorities of the trains. Security requirements, instead, are quite more difficult to define, since they are related to what is unknown about the system properties (e.g., the existing vulnerabilities and threats) and its potential attackers.

Requirements imply the adoption of certain techniques in data transmission systems (non-safety-related, safety-related). These techniques form a “library” of possible methods accessible to the control and protection system designer, to provide protection against each threat identified. EN50159 starts from this expertise.

To reduce the risk a set of fundamental safety services shall be considered and provided to the extent needed for the application, for both open and closed transmission systems:

- message authenticity;
- message integrity;
- message timeliness;
- message sequence.

The following set of known defenses has been outlined:

- a) sequence number;
- b) time stamp;
- c) time-out;
- d) source and destination identifiers;
- e) feedback message;
- f) identification procedure;
- g) safety code;
- h) cryptographic techniques.

Sequence Number

Sequence numbering consists of adding a running number (called sequence number) to each message exchanged between a transmitter and a receiver. This allows the receiver to check the sequence of messages provided by the transmitter.

Time stamp

Concerning the time stamp, it is worth mentioning that when an entity receives information, the meaning of the information is often time-related. The degree of dependence between information and time can differ between applications. In certain cases old information can be useless and harmless and in other cases the information could be a potential danger for the user. Depending on the behavior in time of the processes which interchange information (cyclic, event controlled etc.) the solution may differ. One solution which covers time-information relationships is to add time stamps to the information. This kind of information can be used in place of or combined with sequence numbers depending on application requirements.

Timeout

In transmission (typically cyclic) the receiver can check if the delay between two messages exceeds a predefined allowed maximum time. If this is the case, an error shall be assumed. If a back channel is available, supervision can be performed by the sender. The sender starts a timer when sending a message. The receiver of this message responds with an acknowledge message related to the received one. If the sender does not receive the corresponding acknowledge message within a predefined time, an error shall be assumed.

Source and destination identifiers

Multi-party communication processes need adequate means for checking the source of all information received, before it is used. Messages shall include additional identifiers to permit this.

Feedback message

Where an appropriate transmission channel is available, a feedback message may be sent from the receiver of safety-critical information to the sender. The contents of this feedback message may include:

- data derived from the contents of the original message, in identical or altered form,
- data added by the receiver, derived from its own local information,
- additional data for safety or security purposes.

The use of such a feedback message can contribute to the safety of the process in a variety of ways:

- by providing positive confirmation of reception of valid and timely messages;
- by providing positive confirmation of reception of corrupted messages, to enable appropriate action to be taken;
- by confirming the identity of the receiving equipment;
- by facilitating synchronization of clocks in sending and receiving equipment;
- by facilitating dynamic checking procedures between parties.

Identification Procedure

Open transmission systems can additionally introduce the risk of messages from other (unknown) users being confused with information originating from an intended source (a form of masquerade).

A suitably designed identification procedure within the safety-related process can provide a defense against this threat.

Two types of identification procedure can be distinguished.

- Bi-directional identification: Where a return communication channel is available, exchange of entity identifiers between senders and receivers of information can provide additional assurance that the communication is actually between the intended parties.
- Dynamic identification procedures: Dynamic exchange of information between senders and receivers, including transformation and feedback of received information to the sender, can provide assurance that the communicating parties not only claim to possess the correct identity, but also behave in the manner expected. This type of dynamic identification procedure can be used to preface the transmission of information between communicating safety-related processes and/or it can be used during the information transmission itself.

Safety Code

In transmission systems, in general, transmission codes are used to detect bit and/or burst errors, and/or to improve the transmission quality by error-correction techniques. Even though these transmission codes can be very efficient, they can fail because of hardware faults, external influences or systematic errors.

The safety-related process shall not trust those transmission codes from the point of view of safety. Therefore a safety code under the control of the safety-related process is required additionally to detect message corruption.

Cryptographic techniques

Cryptographic techniques can be used if malicious attacks within the open transmission network cannot be ruled out. This is usually the case when safety-related communication uses:

- a public network;
- a radio transmission system;
- a transmission system with connections to public networks.

Against intentional attacks by means of messages to safety-related applications, the safety-related messages shall be protected with cryptographic techniques.

This requirement, aimed at avoiding masquerade from unauthorized senders, can be met by one of the following solutions:

- use a safety code able to provide cryptographic protection;
- encipher the messages after the safety code has been applied;
- add a cryptographic code to the safety code.

These techniques can be combined with the safety encoding mechanism or provided separately.

Cryptographic techniques imply the use of keys and algorithms. The degree of effectiveness depends on the strength of the algorithms and the secrecy of the keys. The secrecy of a key depends on its length and its management.

5.3 SECURITY REQUIREMENTS FOR EMBEDDED SYSTEMS

A security solution for embedded devices must ensure the device firmware has not been tampered with, secure the data stored by the device, secure communication and protect the device from cyber-attacks. This can only be achieved by including security in the early stages of design. Security requirements must take into consideration the cost of a security failure (economic, environmental, social, etc.), the risk of attack, available attack vectors, and the cost of implementing a security solution.

Development of secure embedded systems shall involve use of proper development processes, software tools and hardware platforms, and it has to consider specific features, as described in the following.

Secure code. It can be achieved through cryptographically signed code from the manufacturer along with hardware support to verify that the code is authenticated. This solution may ensure the firmware has not been tampered.

Secure code updates. A method for secure code updates shall be implemented in order to install security patches, to fix potential bugs, etc.

Authentication. All communications with the device shall be authenticated.

Access control.

- Access to the embedded system shall be restricted to a selected set of authorized users (user identification);
- Access to a network or a service has to be provided only if the device is authorized (secure network access).

Use an appropriate team.

- Development and design teams of embedded systems shall include people with relevant experience and qualifications.
- Development teams shall include by domain experts as well as security experts.

Use appropriate software tools.

Creating high-quality source code is not sufficient. Proper tests on the executable code, as well as on the source code shall be carried out.

Control power consumption.

Identification of factors useful to minimize the power consumption:

- Low power elements; CPU, for instance, runs on reduced clock rate with less on-chip functionality;
- Operational regime: sleep/wakeup mode – transmission only if the value of a measured physical quantity is larger than the predetermined bound;
- Communication protocol – dictates a lower bound on the power consumption.

Security Assessment.

In order to prove that a product actually satisfies security requirements, thus giving confidence on system security to end users, a quantitative measure of security shall be provided before its deployment.

5.4 DESIGN REQUIREMENTS FOR SAFETY AND SECURITY IN AEROSPACE SYSTEMS

CSW contribution

Aerospace systems must comply with international standards such as DO-178[ref] and ECSS standards [ref. www.ecss.nl]. These standards are well described and provide good guidance for the safety and dependability properties of the systems. However, these standards have not been thought with security concerns in mind, this is why most of the systems designed based on the mentioned standards are generally safety and dependable but lack some security design and security concerns.

As already stated in section 4.4, the need for safety and dependability of aerospace systems leads to some common requirements that guide specific design constraints, including real-time systems (timings), redundancy, autonomy, availability and performance.

However, these systems are generally not designed with confidentiality, integrity and availability (from a security perspective) in mind or as main properties (section 2.1.1). In order to join the current safety and dependability design-based systems with security concerns it is important to promote a security analysis (hazard analysis or FMEA analysis-based) in-line with the threats that exist and that will evolve, as for example:

- Vulnerability of existing and future aeronautic and space systems. The potential weakness of the various complex aerospace systems that may be exploited or abused by attackers needs to be considered in depth. For example, weak signals may be easily jammed; unencrypted data can be accessed without authorization. Requirements for communications and signal exchange protection must be defined, but the impact on real-time and performance shall also be taken into account.
- Applicable security mechanisms or counter measures against attacks. Critical aerospace systems usually have more stringent requirements than ordinary web applications or enterprise information system, e.g. short response time, inadmissible jitter or delay, unacceptable power-down or reboot, strong fault-tolerance capability. It is a great challenge to have effective mechanisms adopted, e.g. monitoring techniques, segregation strategies, but with limited negative effects on the system performance introduced. Requirement to take into account these security mechanisms shall be derived from a threat analysis activity.
- Secure software development support. The development process and better programming language support should be studied in order to have security protection built-into the ways that aerospace programs are designed and written. More exploration on security testing techniques on aerospace software systems should be carried out, through which the vulnerable features of software may be detected before software is released and installed. Requirements from other domains that already implement secure software development support shall be analysed and integrated in the safety-critical systems development processes.
- Secured data communications between system interfaces. The data flowing over various routes on the airborne networks or space communication links should be protected so that the information cannot be diverted, monitored, or altered. Some of current protocols for directing data traffic can be exploited to make messages appear to come from someplace other than their true origin. Requirements to prevent data modification and communication attacks must be defined and implemented.

6 TECHNIQUES FOR THE ASSESSMENT OF SECURITY IN SAFETY-CRITICAL SYSTEMS

6.1 DETECTING CODE VULNERABILITIES

UC contribution (Marco Viera)

To minimize security issues, developers should search applications for vulnerabilities, activity for which there are two main approaches: white-box analysis and black-box testing. Other techniques, generically named as gray-box, combine black-box testing and white-box analysis to achieve better results. The following sections introduce these approaches and present some of the existing techniques and tools.

6.1.1 Black-box Testing

Black-box testing is based on the analysis of the program execution from an external point-of-view [37]. In short, it consists of exercising the software and comparing the execution outcome with the expected result. Testing is the most used technique for verification and validation of software. There are several levels for applying black-box testing, ranging from unit testing to integration testing and system testing. The testing approach can also be more formalized (based on models and well defined tests specifications) or less formalized (e.g. when considering informal “smoke testing”). The tests specification should define the coverage criteria (i.e. the criteria that guides the definition of the tests in terms of what is expected to be covered) and should be elaborated before development. The idea is that the test specification can help developers during the coding process (e.g. tests can be executed during development) and that, by designing tests a priori, it is possible to avoid biasing the tests due to knowledge about the code developed.

Robustness testing is a specific form of black-box testing [37]. The goal is to characterize the behavior of a system in presence of erroneous input conditions. Although it is not directly related to benchmarking (as there is no standard procedure meant to compare different systems/components concerning robustness), authors usually refer to robustness testing as robustness benchmarking. This way, as proposed by [38], a robustness benchmark is essentially a suite of robustness tests or stimuli. A robustness benchmark stimulates the system in a way that triggers internal errors, and in that way exposes both programming and design errors in the error detection or recovery mechanisms (systems can be differentiated according to the number of errors uncovered).

Penetration testing, a specialization of robustness testing, consists of the analysis of the program execution in the presence of malicious inputs, searching for potential vulnerabilities [33]. In this approach the tester does not know the internals of the application and it uses fuzzing techniques in its requests [33]. The tester needs no knowledge of the implementation details and tests the inputs of the application from the user’s point of view. The number of tests can reach hundreds or even thousands for each vulnerability type. Penetration testing tools provide an automatic way to search for vulnerabilities avoiding the repetitive and tedious task of doing hundreds or even thousands of tests by hand for each vulnerability type.

Despite the use of automated tools, in many situations it is not possible to test all possible input streams, as that would take too much time. So, as soon as software specifications are complete, test cases should be designed to have the biggest coverage and representativeness possible. The most common automated security testing tools used are generally referred to as **security scanners** (or vulnerability scanners). Security scanners are often regarded as an easy way to test applications against vulnerabilities. These scanners have a predefined set of tests cases that are adapted to the application to be tested, saving the user from defining all the tests to be done. In practice, the user only needs to configure the scanner and let it test the application. Once the test is completed the scanner reports existing vulnerabilities (if any detected).

A similar concept is the one of attack injection [39] techniques are frequently used to assess both the security of application (by searching for vulnerabilities) and that can also be useful to evaluate the effectiveness of security protection mechanisms (by testing their ability to detect attacks).

Basically, attack injection methodology adapts and extends fault injection techniques to look for security vulnerabilities. These techniques use predefined test classes of attacks, fuzzing techniques and mutation testing to attack the target system under evaluation. It can be used in the form of penetration testing (*black box*), but the evaluation process is much more effective if there are available sources of information on the effects of the attacks.

An improved variation of this technique, which represents one of the prominent techniques to evaluate security mechanisms, is usually named vulnerability and attack injection. The technique, proposed in [34] for web applications, is also inspired on fault injection techniques that has been used for decades in the dependability area to evaluate fault tolerant mechanisms. Conceptually, this technique allows injecting realistic vulnerabilities and automatically attacking them, and finally evaluating the result of the attack. The technique is particularly interesting because it can be used for the assessment of specific security mechanisms and tools in custom setup scenarios, thus providing results that are more representative for the user. The authors demonstrated the usability of the technique by evaluating several web application IDSs and penetration testers [34].

6.1.2 White-box Analysis

White-box analysis consists in the examination of the code of the web service without executing it [33]. This can be done in one of two ways: manually during inspections and reviews or automatically by using automated analysis tools.

Inspections, initially proposed by Michael Fagan in the mid 1970's [40], are a technique that consists on the manual analysis of documents, including source code, searching for problems. It is a formal technique based on a well-defined set of steps that have to be carefully undertaken. The main advantage of inspections is that they allow uncovering problems in the early phases of development (where the cost of fixing the problem is lower).

An inspection requires several experts, each one having a well-defined role, namely: author (author of the document under inspection), moderator (in charge of coordinating the inspection process), reader (responsible for reading and presenting his interpretation of the document during the inspection meeting), note keeper (in charge of taking notes during the inspection meeting), and inspectors (all the members of the team, including the ones mentioned before). During the inspection process, this team has to perform the following generic steps:

- 1) **Planning:** starts when the author delivers the artifact to be inspected to the moderator. The moderator analyses that artifact and decides if it is ready to undergo the inspection process. If not, then the artifact is immediately returned to the author for improvement. This step includes also selecting the remaining members of the inspection team.
- 2) **Overview:** after delivering the artifact (and other artifacts needed to understand it) to the experts, the author presents in detail the goal and structure of the artifact to be inspected. By the end of this meeting the inspection team must be familiar with the job to be performed.
- 3) **Preparation:** the inspectors analyze individually the artifact in order to prepare themselves for the inspection meeting.
- 4) **Inspection meeting:** in this meeting the reader reads and explains his interpretation of the artifact to the remaining inspectors. Discussion is allowed to clarify the interpretation and to disclose existing issues. The outcome of the inspection meeting is a list of issues that need to be fixed and one of three verdicts: accept (the document does not present any problem), minor corrections (the document presents minor issues that need to be fixed), and re-inspection (the document presents major issues that need to be fixed and the resulting artifact must be inspected).
- 5) **Revision:** the author modifies the artifact following the recommendations of the inspection team.
- 6) **Follow-up:** the moderator checks if all the problems detected by the inspectors were adequately fixed. The moderator may decide to conduct another inspection meeting if there

were considerable changes in the artifact or if the changes made by the author differ from the recommendations of the experts.

A **code inspection** is the process by which a programmer delivers the code to his peers and they systematically examine it, searching for programming mistakes that can introduce bugs. A security inspection is an inspection that is specially targeted to find security vulnerabilities. Inspections are the most effective way of making sure that a service has a minimum number of vulnerabilities [41] and are a crucial procedure when developing software to critical systems. Nevertheless, they are usually very long, expensive and require inspectors to have a deep knowledge on web security.

A less expensive alternative to code inspections is **code reviews** [42]. Code reviews are a simplified version of code inspections that can be considered when analyzing less critical services. Reviews are also a manual approach, but they do not include the formal inspection meeting. The reviewers perform the code review individually and the moderator is in charge of filtering and merging the outcomes from the several experts. In what concerns the roles and the remaining steps reviews are very similar to inspections. Although also a very effective approach, it is still quite expensive.

Code walkthroughs are an informal approach that consists of manually analyzing the code by following the code paths as determined by predefined input conditions [42]. In practice, the developer, in conjunction with other experts, simulate the code execution, in a way similar to debugging. Although less formal, walkthroughs are also effective on detecting security issues, as far as the input conditions are adequately chosen. However, they still impose the cost of having more than one expert manually analyzing the code.

The solution to reduce the cost of white-box analysis is to rely on automated tools, such as static code analyzers. In fact, the use of automated code analysis tools is seen as an easy and fast way for finding bugs and vulnerabilities in web applications.

Static code analysis tools vet software code, either in source or binary form, in an attempt to identify common implementation-level bugs [33]. The analysis performed by existing tools varies depending on their sophistication, ranging from tools that consider only individual statements and declarations to others that consider dependencies between lines of code. Among other usages (e.g. model checking and data flow analysis), these tools provide an automatic way for highlighting possible coding errors. The main problem of this approach is that exhaustive source code analysis may be difficult and cannot find many security flaws due to the complexity of the code and the lack of a dynamic (runtime) view.

FindBugs is an open source tool that *“uses static analysis to look for bugs in Java code”* [43]. Findbugs is composed of various detectors each one specialized in a specific pattern of bugs. The detectors use heuristics to search in the bytecode of Java applications for these patterns and classify it according to categories and priorities. Some of the highest levels of priorities are usually, among other problems, security issues.

Fortify 360 is a suite of tools for vulnerability detection commercialized by Fortify Software [44]. The module Fortify Source Code Analyzer performs static code analysis. According to Fortify, it is able to identify the root-cause of the potentially exploitable security vulnerabilities in source code. It supports scanning of a wide variety of programming languages, platforms, and integrated development environments.

An example of research work is [45], which presented an approach for the detection of vulnerabilities related to input validation. This approach relies on developer-provided annotations, which limits the practical applicability of the approach, and assumes that preconditions for all sensitive functions can be accurately expressed ahead of time, which is not always the case.

6.1.3 Gray-box Testing

The main limitation of black-box approaches is that the vulnerability detection is restricted by the output of the application. On the other hand, white-box analysis does not take into account the

runtime view of the code. Gray-box approaches combine black-box and white-box techniques in order to overcome their limitations and can be used for both vulnerability and attack detection.

Dynamic program analysis consists of the analysis of the behavior of the software while executing it. The idea is that by analyzing the internal behavior of the code in the presence of realistic inputs it is possible to identify bugs and vulnerabilities. Obviously, the effectiveness of dynamic analysis depends strongly on the input values (similarly to testing), but it takes advantage of the observation of the source code (similarly to static analysis). For improving the effectiveness of dynamic program analysis, the program must be executed with sufficient test inputs. Code coverage analyzers help guaranteeing an adequate coverage of the source code [46][47].

Two techniques that combine static and dynamic analysis have been proposed to perform automated test generation to find SQL Injection vulnerabilities. SQLUnitGen, presented in [48], is a tool that combines static analysis with unit testing to detect SQL injection vulnerabilities. The tool uses a third-party test case generator and then modifies the test cases to introduce SQL injection attacks. These concrete attacks are obtained by using static analysis to trace the flow of user input values to the point of query generation.

While other works focused on identifying vulnerabilities related to the use of external inputs without sanitizations, the work presented in [49] introduces an approach that combines static and dynamic analysis to analyze the correctness of sanitization processes. First, a technique based on static analysis models the modifications that the inputs suffer along the code paths. This approach uses a conservative model of string operations, which might lead to false positives. Then, a second technique based on dynamic analysis works bottom-up from the sinks and reconstructs the code used by the application to modify the inputs. The code is then executed, using a large set of malicious input values to identify exploitable flaws in the sanitization process.

RT contribution:

Some works exist in literature intended to merge security risk assessment with safety one. IN particular we report two of them which we consider more complete with respect to the other: the former is the “Threat Assessment & Remediation Analysis (TARA)” developed by MITRE [6] and the second one is the “Smart Grid Information Security” developed by CENELEC/ETSI [7].

6.2 THREAT ASSESSMENT & REMEDIATION ANALYSIS (TARA)

TARA methodology is shown in Figure 5. TARA methodology includes three activities: Cyber Threat Susceptibility Analysis (CTSA), Cyber Risk Remediation Analysis (CRRRA) and Data and Tools development.

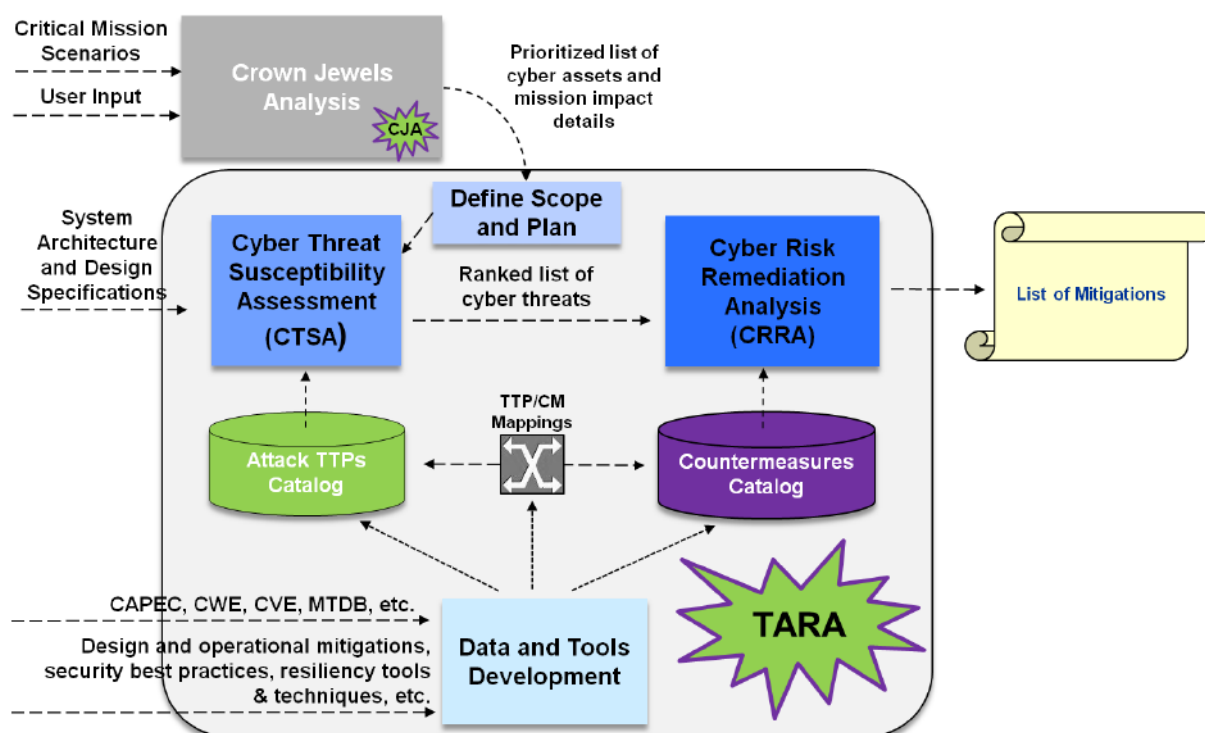


Figure 5 TARA Methodology

CTSA quantitatively assesses a system's inability to resist to cyber-attack over a range of adversary Tactics, Techniques, and Procedures (TTPs) and produces a Threat Matrix, which provides a ranked list of TTPs that each cyber asset is susceptible to. This matrix is used in CRRA to select the range of TTPs to mitigate.

The first step of CTSA is to establish the scope of the evaluation, which can be characterized in terms of:

- The set of system assets being evaluated.
- The range of attack TTPs being considered.
- The Types of adversaries.

TARA foresees that this evaluation can be done as a follow on of a Crown Jewels Analysis (CJA) [13]. CJA is a process for identifying those cyber assets that are most critical to the accomplishment of an organization's mission. CJA is also an informal name for Mission-Based Critical Information Technology (IT) Asset Identification. It is a subset of broader analyses that identify all types of mission-critical assets.

Once the scope of CTSA is established, the next step is to identify candidate TTPs and then eliminate the implausible ones. Many TTPs have prerequisites or conditions that must hold true in order to be effective, otherwise they can be considered implausible. For example, a SQL injection attack has a prerequisite that the system must include a SQL database. Use of weak passwords is one condition that must hold true in order for an adversary to successfully conduct brute force password attacks.

Finally, candidate TTPs that cannot be eliminated are ranked using scoring model. The TTP scoring model assesses the risk associated with each TTP relative to other plausible TTPs considered in the assessment. This ranking helps set priorities on where to apply security measures to reduce the system's susceptibility to cyber-attack. A default TTP scoring spreadsheet is illustrated in Figure 6. This list of criteria has evolved over time and organization are encouraged to tailor the scoring model to reflect their needs.


Factor Range	1	2	3	4	5
Proximity: What proximity would an adversary need in order to apply this TTP?	no physical or network access required	protocol access through DMZ and firewall	user account to target system (no admin access)	admin access to target system	physical access to target system
Locality:  How localized are the effects posed by this TTP?	isolated to single unit	single unit and supporting network	external networks potentially impacted	all units in theater or region	all units globally and associated infrastructure
Recovery Time: How long would it take to recover from this TTP once the attack was detected?	< 10 hours	20 hours	30 hours	40 hours	> 50 hours
Restoration Costs: What is the estimated cost to restore or replace affected cyber asset?	< \$10K	\$25K	\$50K	\$75K	> \$100K
Impact: How serious an impact is loss of data confidentiality resulting from successful application of this TTP?	no impact from TTP	minimal impact	limited impact requiring some remediation	remediation activities detailed in COOP	COOP remediation activities routinely exercised
Impact: How serious an impact is loss of data integrity resulting from successful application of this TTP?	no impact from TTP	minimal impact	limited impact requiring some remediation	remediation activities detailed in COOP	COOP remediation activities routinely exercised
Impact: How serious an impact is loss of system availability resulting from successful application of this TTP?	no impact from TTP	minimal impact	limited impact requiring some remediation	remediation activities detailed in COOP	COOP remediation activities routinely exercised
Prior Use: Is there evidence of this TTP in the MITRE Threat DB?	no evidence of TTP use in MTDB	evidence of TTP use possible	confirmed evidence of TTP use in MTDB	frequent use of TTP reported in MTDB	widespread use of TTP reported in MTDB
Required Skills: What level of skill or specific knowledge is required by the adversary to apply this TTP?	no specific skills required	generic technical skills	some knowledge of targeted system	detail knowledge of targeted system	knowledge of both mission and targeted system
Required Resources: Would resources be required or consumed in order to apply this TTP?	no resources required	minimal resources required	some resources required	significant resources required	resources required and consumed
Stealth: How detectable is this TTP when it is applied?	not detectable	detection possible with specialized monitoring	detection likely with specialized monitoring	detection likely with routine monitoring	TTP obvious without monitoring
Attribution: Would residual evidence left behind by this TTP lead to attribution?	no residual evidence	some residual evidence, attribution unlikely	attribution possible from characteristics of the TTP	same or similar TTPs previously attributed	signature attack TTP used by adversary

Figure 6 Default TTP Risk Scoring Spreadsheet

As said before, the Threat Matrix produced by CTSA is used as input by the CRRA. CRRA is an approach for selecting countermeasures (CMs) to reduce a cyber asset's susceptibility to attack over a range of TTPs.

The first step is to select a list of TTPs to mitigate. There are several strategies to do this. One strategy is to focus only on the highest scoring TTPs in the Threat Matrix for each cyber asset. Another strategy is to focus on the cyber asset(s) that have the highest aggregate susceptibility. A third strategy is to focus exclusively on crown jewel cyber assets. A hybrid strategy might select high scoring TTPs for the crown jewel cyber assets with the highest aggregate susceptibility. Whatever strategy is used, the result will be a list of TTPs for each cyber asset being assessed.

Then CRRA employs a mapping table to represent the many-to-many mapping between TTPs and countermeasures (CMs). Each mapping of the CM to TTP is characterized by the mitigation effectiveness of the CM over a range of criteria: detect, neutralize, limit and recover. Detect CMs serve to identify or uncover the action or presence of a TTP. Neutralize CMs stop or prevent execution of a TTP. Limit CMs serve to reduce or constrain the risk associated with a TTP, either by lessening the severity or likelihood. Recovery CMs facilitate recovery from attack. A given CM may be highly effective at detecting a certain TTP, moderately effective at neutralizing or limiting its impact, but provide no mitigation value in recovering from its effect.

After identifying the CMs, then an iteration to evaluate their merit is performed. It mainly consists in an estimation of utility and cost per each CM. The utility depends on the mitigation effectiveness of the CM (e.g. detection, neutralization etc.). Differently, cost can be estimated considering cost to develop, integrate and maintain the CM over the operational life of system.

An optimal CM solution is the set of CMs that provides effective mitigation over a specified range of TTPs at the lowest cost. What constitutes "effective mitigation" is determined by a CM selection strategy. The total cost per each solution is evaluated by summing the costs of its constituent CMs. The lowest cost solution will be optimal over the range of solution identified.

6.3 SMART GRID INFORMATION SECURITY

SGIS is a methodology proposed by CENELEC/ETSI in order to face with the ever growing cyber-attacks which are performed against electrical grid. With the introduction of the concept of smart grids, power systems became more ICT dependent because of new control mechanisms that are needed to manage renewable energy resources and flexible loads. The increase of the pervasiveness of ICT components introduced new vulnerabilities of electrical grid wrt cyber-attacks. This is the reason why standardization committees started to introduce more accurate security risk analysis in the design of smart grid. The SGIS methodology is a proposed way to be used in order to evaluate the risk of a security attack.

Figure 7 shows a step-by step guide in order to perform the risk analysis process.

In order to assure the best and reliable results executing the SGIS security risk analysis, it is critical to gather detailed and complete use cases identifying clearly the domain and zone (one or more) whose belongs. Each domain covers a set of particular processes and has different actors and owners. Drilling down in domains, zones are sub-processes within each domain representing hierarchical levels of power system management where different actors, technologies and specific activities are involved. It is important to describe the related process in terms of goals, main activities, expected times for the execution of the whole process and activities if it makes sense, involved regulations or laws determining how it has to be executed. Then we should identify the owner of the process as its maximum responsible (It would be very useful if the owner of the process or use case could participate in the risk impact assessment). The purpose of its information assets should be described as well as the identity of the owner of every information asset and the rest of actors involved in its processing (administrators, operators, developers, users, ...) explaining briefly their capabilities processing every information asset.

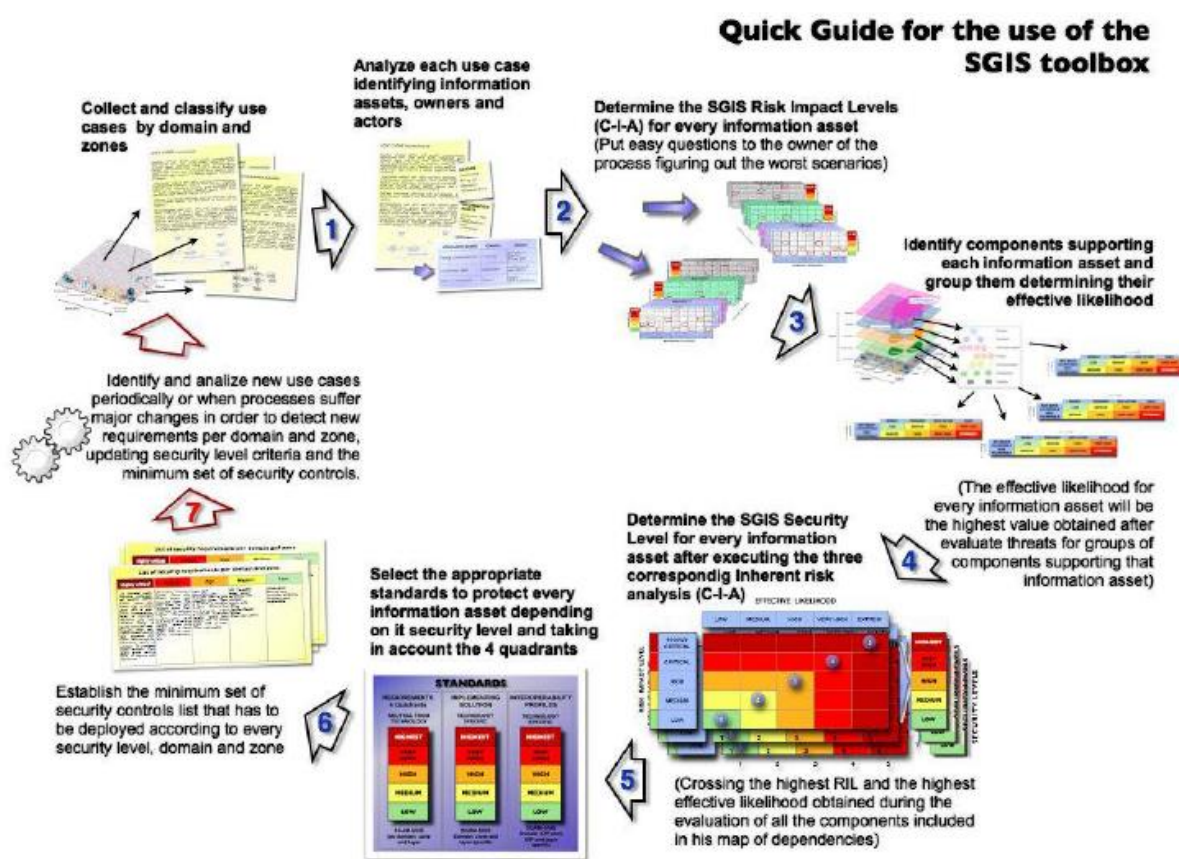


Figure 7 Step by step guide for inherent risk analysis process

Then a risk impact is established analysing how incidents into a particular information asset affects to the process where it is involved. Different incidents produce different impacts. So the highest impact identified in all the possible scenarios determines the Risk impact level for the analysed information asset. The impact scale ranks from 1 (low impact) to 5 (highly critical impact).

Security incidents against information assets affect to their involved processes in different ways. SGIS Impact Analysis methodology identifies six different types of affectation or categories. Categories should be evaluated independently: operational, legal, human, reputation, environmental, financial.

Operational category measures how security incidents impacts in service availability. This category starts offering four different aspects to assess service availability: energy supply, energy flow, population and infrastructure. Other measurable subcategories could be added here if they demonstrate to be useful when analysing use cases.

Energy Supply determines the risk impact level based on the size and type of grid affected by an information security incident (in terms of MW). Energy flow determines the risk impact level based on the flow affected (in terms of Watts/hour). Population determines the risk impact level based on the percentage of people affected in one or more countries (population size and population density were discarded because their variability from one country to another). This scale measures scenarios producing power or access disruptions to critical consumer oriented Smart Grid services. Infrastructures determine the risk impact level based on how many critical, essential or complimentary infrastructures could be affected by a security incident. Usually, at every country, Ministry of Defense defines and determines three types of infrastructures depending of their impact for a nation in case of disruption. The resultant catalogue is secret and it is only communicated to the affected companies who operate the mentioned infrastructures. Critical Infrastructures scope determines the risk impact level based on the type of infrastructures affected in case of unavailability.

Legal category measures the impact of security incidents deriving in both, legal or regulatory non-fulfilments. Because the tight relation between smart grid and citizens, this category is divided in two subcategories: data protection and other law regulation.

Data protection determines the risk impact level in accordance to the definitions to Directive 95/46/EC. Following that, there are two levels of relevant data related to privacy: personal data and sensitive data. Personal data (cf art. 2) are any information relating to an identified or identifiable natural person. Sensitive data (cf art. 8) are particular risky personal data that includes: revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade-union membership, the processing of data concerning health or sex life, biometric and genetic data (judicial interpretation).

Other laws and regulations determine the risk impact level based on legal and regulation punishments. Every country has different laws and regulations in addition to European directives. All of them have to be taken in account for this assessment.

Human category measures how security incidents impact directly or indirectly on people's health (e.g. minor accidents, direct death, etc.).

Reputation category is one of the most difficult aspects to measure when security incidents affect information assets and processes, because people reactions are always unpredictable. Security incidents damaging an organization's reputation for confidentiality, safety or availability may cause serious damage to finances.

Environmental category measures how a security incident on a particular information asset could produce negative changes to the land or ecosystem.

Financial category measures risk impact based on direct monetary loss derived from information security incidents. This category is quantitative and not necessary applies to all use cases. This scale uses as criteria the annual organization's EBITDA (acronym for Earning Before Interest, Taxes, Depreciation and Amortization).

Figure 8 summarizes the Risk Impact Levels (RILs) and the Measurement categories.

RISK IMPACT LEVELS	HIGHLY CRITICAL	regional grids from 10GW	from 10 GW/h	from 50% population in a country or from 25% in several countries	international critical infrastructures affected	not defined	company closure or collateral disruptions	direct and collateral deaths	permanent loss of trust affecting all corporation	>50% EBITDA
	CRITICAL	national grids from 1 GW to 10GW	from 1 GW/h to 10GW/h	from 25% to 50% population size affected	national critical infrastructures affected	not defined	temporary disruption of activities	collateral deaths	permanent loss of trust in a country	<50% EBITDA
	HIGH	city grids from 100MW to 1GW	from 100MW/h to 1GW/h	from 10% to 25% population size affected	essential infrastructures affected	unauthorized disclosure or modification of sensitive data	finances from 10% of EBITDA	direct deaths	temporary loss of trust in a country	<33% EBITDA
	MEDIUM	neighborhood grids from 1MW to 100MW	from 1MW/h to 100MW/h	from 2% to 10% population size affected	complimentary infrastructures affected	unauthorized disclosure or modification of personal data	finances up to 10% of EBITDA	seriously injured or incapacity	temporary and local loss or trust	<10% EBITDA
	LOW	home or building networks under 1 MW	under 1MW/h	under 2% population size affected in a country	no complimentary infrastructures	no personal nor sensitive data involved	warnings	minor accidents	short time & scope (warnings)	<1% EBITDA
MEASUREMENT CATEGORIES										
		Energy supply (Watt)	Energy flow (Watt/hour)	Population	Infrastructures	Data protection	other laws & regulations	HUMAN	REPUTATION	FINANCIAL
OPERATIONAL (availability)					LEGAL					

Figure 8 Risk Impact Level (RIL) and Measurement Categories.

It's important to understand how different types of incidents have different effects over an information asset and over the all process belonging to. SGIS risk impact methodology defines a reduced set of scenarios grouping all possible security incidents to make easier and faster the risk impact analysis process without loss reliability. These scenarios are grouped following the three essential requirements (confidentiality, integrity, availability) and a couple of additional requirements grouped under the term of accounting and assessed within the legal category.

Confidentiality scenarios summarize all security incidents and vector attacks that could be exploited gaining access to internal information and disclosing it to unauthorized people. The lack of legitimacy and authenticity or access of all actors and roles, the lack of encryption and authentication when transmitting control information to smart grid devices and the existence of pathways from outside to smart grid energy transport control systems usually derive in confidentiality scenarios.

Integrity scenarios include all security incidents, accidents and vector attacks that could be exploited with the intermediate goal of altering information for multiple malicious purposes (i.e. altering consumptions to reduce bills, or causing incorrect decisions for the generation and distribution of energy, etc.). Integrity scenarios, therefore, mainly regards data loss or alteration by error of the information asset. It summarizes security incidents where the analysed information asset would be accidentally altered or lost during the process. Manipulation is considered as well. It includes all possible incidents where authorized actors could alter information assets with malicious purposes producing different results from what was expected for the defined process. Then also authenticity scenarios are considered, which are similar to the manipulation scenario, but it takes in account when unauthorized users or agents alters information or inject fake information producing different results from what was expected for the defined process.

Availability scenarios refer to unavailability of required information for particular services due information security incidents against any component supporting the analysed information asset or even directly to the asset.

To determine the risk impact level for a specific information asset implies to evaluate every category and subcategory risk impact level at every SGIS predefined incident scenario affecting the involved process (use case). At the end of the Risk Impact analysis, every information asset will get three risk impact levels. Every specific sector has different interests and prioritizes

confidentiality, integrity and availability in a different way. An example of risk impact assessment is reported in Figure 9.

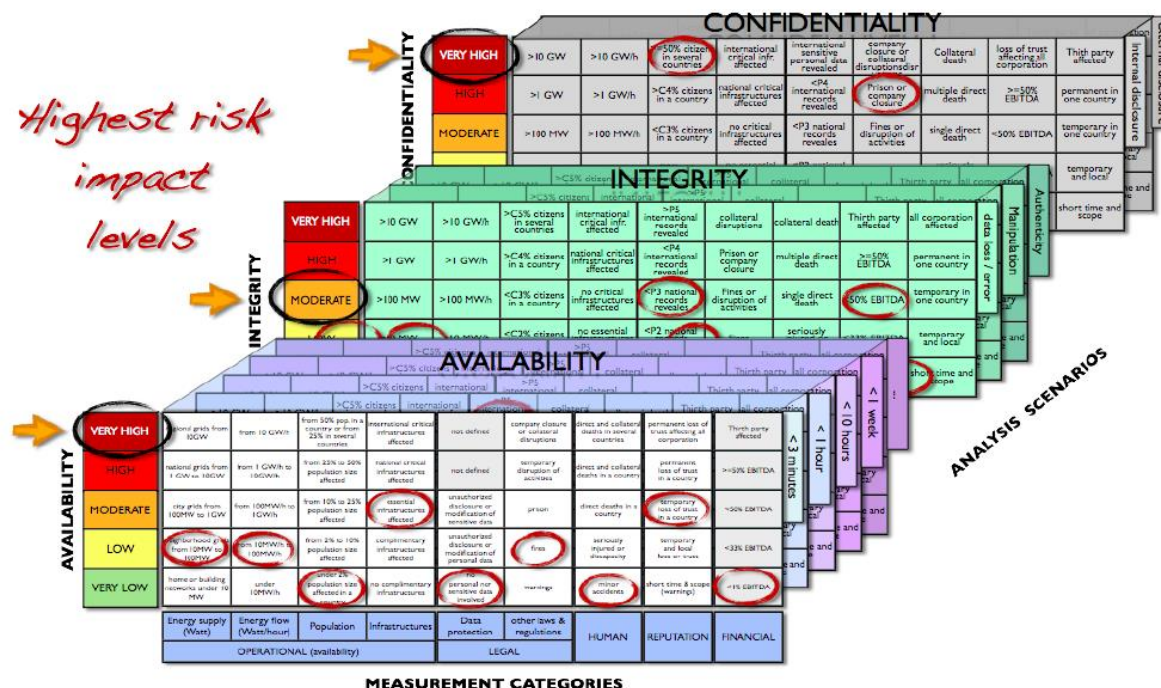


Figure 9 Example of full risk impact assessment

Then, in order to evaluate the risk, also the likelihood of success of an attack should be evaluated. At the moment SGIS methodology provides a preliminary way to evaluate this factor, but further work is needed. SGIS provides tables for identifying threat sources, threat actors, possible interests and capability levels requested in order to perform an attack. At the end of the likelihood analysis, the combination of threat sources, threat actors, interest, capabilities and possible compromise methods of attack will determine the level of likelihood for every information asset. Every combination of those elements will get an effective likelihood within a scale from 1 to 5 (see [7] for details):

- L1: Low: The likelihood of success of an attack is low.
- L2: Medium: The likelihood of success of an attack is medium.
- L3: High: The likelihood of success of an attack is high.
- L4: Very high: The likelihood of success of an attack is very high.
- L5: Extremely: The likelihood of success of an attack is extremely high.

The Inherent risk scale is established crossing the five risk impact levels determined for Confidentiality, Availability and Integrity separately, and the five effective likelihood levels. The result will bring three grids (Confidentiality, Integrity, Availability), each one of them composed by 25 cells grouped in five levels. Each group fits in one of the five security levels as shown in Figure 10.

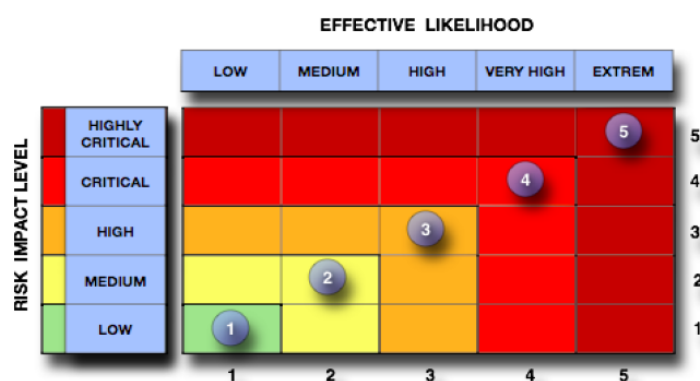


Figure 10 Security Level mapping.

Different security levels will establish different degrees of security requirements. Establishing the adequate set of countermeasures/requirements as standards covering these requirements will allow determine security baselines and identify gaps between smart grid implementations and their adequate deployments. As the protective measures are defined in the SGIS requirement standards (see Annex A in [7]), the implementation of those standards varies for specific products, services and organizations. Therefore all specific implementation also needs to assure that the remaining risk is acceptable for all participating actors in the smart grid.

6.4 TECHNIQUES FOR THE ASSESSMENT OF SECURITY IN AEROSPACE SYSTEMS

CSW contribution

As an example present some of the techniques identified for aerospace systems based on the threats library (Nicola Nostro do we need to complement this with the examples?).

6.5 A SECURITY THREATS LIBRARY FOR SAFETY CRITICAL SYSTEMS

CINI contribution

In literature there exists several attempts to combine both the security and safety techniques for the assessment of critical systems. Nevertheless, many of these works are only theoretical and have not been applied in practice, due to the fact safety and security belong to two different communities, with their own traditions, techniques, terminologies and standards, and also because safety and security specifications may sometimes conflict, i.e., when safety requires fail safe (such as degraded operation) and security requires fail secure (such as shut-down), thus leading security to impact on safety and vice versa. To give an example, Stoneburner [14] proposes a unified risk framework trying to harmonize a generally accepted risk taxonomy for security [15] and a common safety taxonomy from FAA [16]. The result is an idea of a potential combination of the two framework. However, both disciplines, security and safety, remain independent and continue to use the respective concept, and methodology where applicable.

The ongoing and preliminary work presented in [17] tries to set the basis for a general methodology to support the assessment of safety-critical systems with respect to security aspects. The proposed approach is based on a library of generic security threats regardless the specific application domain. The general idea behind the work is to understand the criticality of the issue and the difficulties in finding a proper solution able to deal with interdependencies between safety and security. The main objective is to provide a support to the analysis and the evaluation of safety-critical system with respect to the security issues. Figure 11 shows the logical schema of the interrelations between the activities considered relevant in order to achieve the objective described. The first step is to define a generic security threats library, independent from the system to be assessed. After the definition of the library, which requires a continuous update during the

time, a mapping between the single threats of the library and the NIST security controls [18] is provided. After that, the idea is to consider safety and security critical examples in order to exercise the library, then to identify existing V&V techniques [20] and avoidance techniques in safety domain, that can be used to detect the security threat(s) and provide efficient countermeasures to eliminate the effects. Finally, the last step is to provide proper recommendations (if needed) to adapt the existing techniques, and possibly try to verify whether there is a match with respect to the current safety critical standard. The preliminary results of the work are mainly related to the first two steps, namely the definition of the threats library and the identification of the mapping with the NIST security controls, which are described in detail in the next subsections.

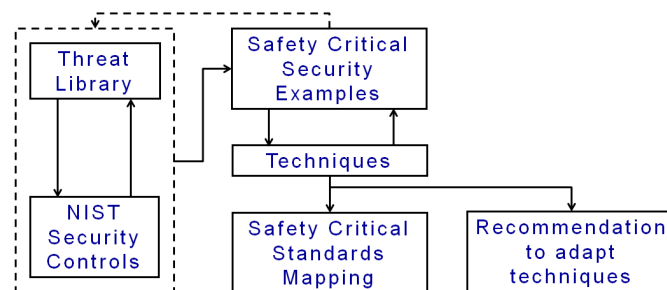


Figure 11: Logical Schema.

6.5.1 The NIST Taxonomy

NIST SP 800-53 [18] provides a catalog of security controls to be applied to federal information systems. The controls are divided into 18 families, as shown in Table 2. Each family contains security controls related to the general security topic of the family. An ID identifies the families, e.g., **SI** corresponds to *System and Information Integrity*. Moreover, the family is associated with a specific class: *Operational*, *Technical*, and *Management* (in round brackets in Table 2), based on the characteristics of the controls of the family. The security controls prescribe specific security-related activities or actions to be carried out by organizations or by information systems, involving aspects of policy, oversight, supervision, manual processes, actions by individuals, or automated mechanisms implemented by information systems/devices. The security controls are identified by the ID of the family they belong to, followed by a numeric identifier; e.g., **SI-4**, whose control name is "*Information System Monitoring*", is the fourth control in the *System and Information Integrity* family. The general security control structure consists of the following sections: *i*) control; *ii*) supplemental guidance; *iii*) control enhancements; *iv*) references; and *v*) a priority and baseline allocation. More details regarding the NIST security controls can be found in [18].

Table 2. NIST SP 800-53 SECURITY CONTROLS FAMILIES

ID	FAMILY	ID	FAMILY
AC	Access Control (Technical)	MP	Media Protection (Operational)
AT	Awareness and Training (Operational)	PE	Physical and Environmental Protection (Operational)
AU	Audit and Accountability (Technical)	PL	Planning (Management)
CA	Certification, Accreditation and Security Assessments (Management)	PS	Personnel Security (Operational)
CM	Configuration Management (Operational)	RA	Risk Assessment (Management)
CP	Contingency Planning (Operational)	SA	System and Services Acquisition (Management)
IA	Identification and Authentication (Technical)	SC	System and Communications Protection (Technical)
IR	Incident Response (Operational)	SI	System and Information Integrity (Operational)
MA	System Maintenance (Operational)	PM	Program Management (Management)

Furthermore, the NIST SP 800-53 provides a direct mapping between the security controls with respect to ISO/IEC 27001 [21], and to ISO/IEC 15408 (*Common Criteria*) [22], which are the main security reference standards. Thus, such a mapping can provide evidence that certain security controls are implemented correctly, operating as intended, and producing the desired effect in satisfying stated security requirements.

6.5.2 The Threats Library

This section presents the preliminary threats library, which, for its nature, may be not exhaustive. Indeed, not all of the threats may have been considered during the definition of the library, and new threats, previously not plausible, may become realistic at a later time. Thus requiring a continuous update over the time. The library is partitioned into four levels: *Network/Communication*, *Device/Hardware/Host*, *Application/Software*, and *Human*. For each threat it provides: *i)* a brief description; *ii)* the potential system vulnerability that can be exploited to realize the threat, since the threats are not referred to a specific domain, the corresponding vulnerabilities are generic as well; *iii)* the proper countermeasures, useful to tackle the realization of the threat; *iv)* the affected assets or security properties (*Confidentiality, Integrity, Availability*). Moreover, each threat of the library is associated to a number of NIST security controls, according to the activity of mapping threats to security controls. Such an activity turns out to be useful by providing a complete view of the controls, and consequently a mapping to the security standards, thus allowing an absolute assessment of the system under analysis.

Figure 12 shows an extract from the threats library developed in this work, in order to provide an overview of the structure and contents. As mentioned it is composed by four main levels (Figure 12 shows two of them in the first column: *Communication/Network* and *Device/Hardware/Host* levels). The second column reports the identified threats for all the levels. We want to point out that the threats do not refer to a specific system, but they are as general as possible.

LEVEL	THREAT	VULNERABILITY	DESCRIPTION	COUNTERMEASURES	PROPERTIES/ASSET AFFECTED	NIST 800-53 SECURITY CONTROLS
COMMUNICATION NETWORK Level						
	Network Scanning	Configuration	Network devices can be discovered and profiled in much	1) Configure network devices (e.g., routers) to restrict their	Confidentiality	CM-7: LEAST FUNCTIONALITY SI-4: INFORMATION SYSTEM MONITORING
	Sniffing	Cryptography	Also known as packet analyzer or protocol analyzer , is a program	1) Use strong physical security and proper segmenting of the	Confidentiality	CM-6: CONFIGURATION SETTINGS AC-4: INFORMATION FLOW ENFORCEMENT
	Spoofing	Authentication	Spoofing is attempting to gain access to a system by using a false identity. This can be	1) Use strong authentication. 2) Do not store secrets (for example, passwords) in	Availability Integrity Confidentiality	AC-18: WIRELESS ACCESS CA-7: CONTINUOUS MONITORING IA-3: DEVICE IDENTIFICATION AND
	Message Modification (Hijacking)	Authentication	An attacker uses network monitoring software to capture the authentication token (e.g., a	1) Use encrypted session negotiation. 2) Use encrypted	Integrity Availability	AC-2: ACCOUNT MANAGEMENT AC-3: ACCESS ENFORCEMENT AC-12: SESSION TERMINATION
DEVICE/HARDWARE/HOST Level						
	Malware (Generic)	Configuration	Designed to take control of internal organizational	1) Stay current with the latest operating system service packs	Confidentiality Availability	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES
	Viruses - Malware	Configuration Awareness	A virus is a program that is designed to perform malicious	1) Stay current with the latest operating system service packs	Availability Integrity	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES
	Remote access Trojan (RAT) - Malware	Configuration Awareness	Monitoring user behavior through keyloggers or other	Keep antivirus software up to date and refrain from	Confidentiality Availability	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES
	Trojan horses - Malware	Configuration	A Trojan horse resembles a virus except that the malicious code is	1) Stay current with the latest operating system service packs	Availability Integrity	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES
	Keylogger or form-grabber	Configuration	keylogger is a software able to capture every keystroke and	1) Stay current with the latest operating system service packs	Confidentiality Availability	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES
	Worms - Malware	Configuration	A worm is similar to a Trojan horse except that it self-	1) Stay current with the latest operating system service packs	Availability	AT-1: SECURITY AWARENESS AND TRAINING POLICY AND PROCEDURES

Figure 12. Extract from the Threats Library.

6.5.3 Aerospace Case Study

The aerospace domain considered involves civil airborne safety critical systems, developed under the DO178B/C standards ([23], [24]), and composed by complex embedded on-board and ground control systems. There are some examples of detailed attacks for specific aerospace and space

systems that might be the targets of malicious bodies. A first insight has been provided in [19]. In the following is provided a preliminary application of the threats library for a specific system in the aerospace domain, originally taken from survey performed on real problems that occurred and that are hereby simplified.

Potential attacks to aerospace systems may be identified on the Automatic Dependent Surveillance-Broadcast (ADS-B) system. The ADS-B is a cooperative surveillance technology for tracking aircrafts. The aircraft determines its own position via GPS and periodically broadcasts it, in order to be tracked. Such a data can be received by any airborne or ground based receivers. ADS-B is one of the technologies selected as part of the Next Generation Air Transportation System (NextGen) and the Single European Sky ATM Research (SESAR), and it will be mandatory for all new aircraft in the European airspace by 2015, and in the US by 2020. ADS-B provides an improvement to safety and decreases the likelihood for incidents, unless the system's weak security is exploited by malicious users. The ADS-B has been selected because of its critical role nowadays and in the future, but also because it has been found that it is vulnerable from the point of view of security [25], [26]. There are also common attacks at Virtual Local Area Networks (VLANs) that can be used for separation of aviation data networks between flight subsystems. Based on the work compiled in [19], a mapping has been performed between the general threats identified in the library and the specific attacks to the ADS-B and VLANs. The focus in this study was only on the threats related to the Communication and Network level of the defined library, because at this level is more likely to identify security vulnerabilities. In the following there is a list of the general threats (from the library) and the specific threats (identified in the case study), with a brief description.

1. Spoofing

- **Aircraft Spoofing** (ADS-B). ICAO 24 bit address that is used by transponders for aircraft identification may be spoofed or reprogrammed.
- **ARP spoofing** (VLAN). Sending spoofed Address Resolution Protocol (ARP) messages. ARP spoofing may allow an attacker to intercept data frames on a VLAN, modify the traffic, or stop the traffic altogether.

2. Message Modification (Hijacking)

- **Virtual Trajectory Modification** (ADS-B). Modification of messages on the trajectory of an existing aircraft.
- **False Alarm** (ADS-B). Modification of messages of a real aircraft to indicate a fake alarm.

3. Message Deletion (Tampering)

- **False Alarm** (ADS-B). Deletion of messages of a real aircraft to indicate a fake alarm.
- **Aircraft Disappearance** (ADS-B). Deletion of all messages of a target aircraft, thus leading to failure of collision avoidance systems and confusion at ground sensors.

4. Denial of Service

- **Ghost Aircraft Flooding** (ADS-B). The message injection of multiple aircrafts simultaneously.
- **Ground Station Flooding** (ADS-B). Continuous jamming signals on a ground sensor or aircraft result in high losses and deletion of ADS-B messages.
- **MAC Flooding** (ADS-B). The limited memory of the switch between airborne data networks is flooded.
- **Multicast Brute Force** (VLAN). Exploitation of switches' potential vulnerabilities against a storm of L2 multicast frames.

5. Network eavesdropping

- **Air traffic eavesdropping** (ADS-B). Anyone can easily capture clear text data of air traffic.

6. Jamming

- **Jamming the Air Traffic Control (ATC) reception of ADS-B** (ADS-B). Interfere with communications between ATC and ADS-B.

7. Message Injection

- **Ghost Aircraft Injection** (ADS-B). Messages of a non-existing aircraft are broadcasted on the ADS-B communication channel.
- **False Alarm** (ADS-B). Injection of messages of a real aircraft to indicate a fake alarm.
- **Double-Encapsulated 802.1 Q/Nested VLAN** (VLAN). Transmission of packets from one VLAN to another VLAN without proper authorization.

6.6 ADDITIONAL TECHNIQUES FOR INCREASING THE SECURITY OF SAFETY-CRITICAL SYSTEMS**RT contribution****6.6.1 Secure OS hardening**

Security attacks [50][51] on certain critical systems has raised the importance of security in safety-critical systems. Hence, safety-critical embedded systems must use a security focused, and usually a security evaluated OS for its operation.

6.6.1.1 OS SECURITY FEATURES

Some of the OSs use exploit mitigation techniques by default, however some require hardening which involves reducing the attack surface of the OS. Hardening may be achieved by installing third party security modules or may be done manually.

OS security features for an application depends on the amount of security needed, regulatory requirement, performance, and the cost associated with the system. Table - 1 lists some of the security features available in security focused OS.

Security Features	Description/Benefits	Reference
Formal verification of security properties	<p>“Enforces a number of strong security properties: integrity, confidentiality, and availability” [FAQ]</p> <p>“If the proof assumptions are met, the seL4 kernel implementation has no deviations from its specification” [FAQ]</p> <p>However:</p> <p>“There may still be unexpected features in the specification and one or more of the assumptions may not apply.” [FAQ]</p>	[52][53]
Semi-formally Verified Design and Tested (NSA: EAL 6+ High Robustness Common Criteria SKPP)	EAL6+ rating means that the OS was designed and certified to defend against well-funded and sophisticated attackers.	[54][55]
SIL – 4 Certified OS	OS certified to the level of EN 50128 SIL 4.	[56]
Privilege separation, Privilege revocation, Chroot jailing, Sandboxing of applications.	Based on principle of least privilege, the OS and its programs are divided into modules each having separate set of	OpenBSD

	<p>privileges. Once a privileged module completes its privileged task, It drops its privileges.</p> <p>In case of a security attack/compromise of a module, Its privilege may not be enough to do much damage.</p>	
<p><u>Disk protection:</u></p> <ol style="list-style-type: none"> 1. Full Disk Encryption (FDE) 2. Filesystem-level encryption 	<p>Some operating systems support full disk encryption and file level encryption. Also, some support disk encryption hardware.</p> <p>In general most of the embedded applications are diskless. However, these features could be useful in some applications such as in medical sector. The Health Industry Portability and Accounting Act (HIPAA) requires that patient data stored within medical devices should be protected.</p>	<p>BitLocker for Windows; FileVault for Apple OS/X.</p>
<p><u>Memory protection:</u></p> <ol style="list-style-type: none"> 1. Stack protector 2. W^X (W xor X) 3. Guard pages 4. Non Executable Stack 	<p>MISRA standards do not allow use of dynamic memory though. Allows early detection of incorrect use of memory in application and OS.</p> <p>Even if a application/OS is vulnerable, these features make the exploitation difficult.</p>	[57]
Randomization	<p>Randomization of:</p> <ol style="list-style-type: none"> 1. malloc() addresses 2. mmap() blocks 3. stack gap 4. Key data areas of a process in its address space. 5. Process ids. 6. TCP ISN, TCP Timestamp, Ephemeral Source Port, IPID and the IPv6 Fragment ID. <p>This reduces the predictability of various values that may be of use to an attacker.</p>	[57][58]
Mandatory access control	<p>MAC is a OS controlled access control mechanism based on a defined policy. The policy determines which accessing entity can access which resources.</p> <p>MAC requires careful planning to define a policy. And the policy is strictly enforced by the OS and cannot be altered by normal users or programs.</p>	[59][60][61][62]

7. Table OS Security Features

A secure OS is the base of a secure system, and is a must.

6.6.1.2 OS DETECTION AND PORT SCANNING

The first steps an attacker takes to know if a system is vulnerable is by knowing if the OS is vulnerable. This is done by gaining knowledge of the OS name and its version. This helps in running specific attacks.

Gaining the knowledge of OS through network is possible as different OS respond differently to specially crafted network packets. Network scanners such as Nmap [63] are able to guess the remote OS with version; and one can run such network scanners from outside to know if the outside world can know the true name and version of the OS remotely. A good firewall may be used to reject these specifically crafted/maligned packets as a defense mechanism.

It is also necessary to check if the OS is vulnerable, which can be achieved by thinking like an attacker and using a network vulnerability scanner (such as Metasploit[64] and Nessus [65]) to check for possible vulnerabilities before deployment. These scanners scan for vulnerabilities reported for various operating systems in forums such as CVEDetails for each OS [66], Vulnerability Notes Database [67], and BugTraq mailing list [68].

6.6.2 Strong random bit generator

Random bits (numbers or strings) may be generated in a system to generate: session ids, Initial sequence numbers, port numbers, process ids, new username/passwords, salts, cryptographic keys, etc.

There are three types of random number generators:

1. **Pseudo Random Number Generator (PRNG):**

PRNGs are random bits generated through algorithms. Unfortunately computers are not known to produce true random bits. Thus random bits generated algorithmically are known as pseudo-random bits. These algorithms are deterministic in nature and depend on initial seed value. The seed value is usually given by the programmer and could be a simple constant or may include current time and process id [69].

PRNGs may also be used with periodically refreshed random seed.

Advantages: The main advantage is speed of random bits generation.

Disadvantages:

1. They can be guessed if seed is known.
2. They are known to be repeatable.
3. Easily prone to power analysis.

Due the above reasons PRNGs must not be used in critical systems.

2. **Cryptographically secure pseudorandom number generator (CSPRNG):**

A cryptographically secure pseudo-random number generator (CSPRNG) is a pseudo-random number generator (PRNG) which generates strong random bits using cryptographic algorithms.

Advantages: The pseudo random bits provided by a good CSPRNGs are stronger than simple PRNGs, and the bits are difficult to guess before they are generated.

Disadvantages:

1. Even though they use cryptographic techniques, they are still pseudo random in nature. Their main strength lies in the cryptographic algorithm. Hence the

algorithms must be reviewed periodically and must be changed if they are found to be weak.

2. The randomness is dependent on the OS and its security.

Example: Several operating systems include APIs like **arc4random** for generating random bits. The **arc4random** API for some OSs is based on RC4 algorithm, and due to reports on possible attacks on RC4, some OSs have modified **arc4random** to use **ChaCha20** cipher and some have not.

3. Hardware based random number generators (TRNG, True Random Number Generator):

Hardware random number generator (TRNG, True Random Number Generator) is a device that generates random numbers from a physical process such as thermal noise, photoelectric effect, and other quantum phenomena (example [70]).

Advantages:

1. The bits generated are expected to be true random in nature.
2. The random bits generated is independent of OS.
3. They can be made less prone to power analysis attacks.

Disadvantages:

1. The number of random bits/sec of the device may depend on the nature of the hardware. Hence a proper device must be chosen to suit the application.
2. The hardware also needs regular health monitoring to see if the numbers being produced are really random [71].
3. Cost and maintenance.

For critical systems, a good random number generator is a must to provide security against guessing and brute force attacks. Hence, depending on the application a strong source of random bits must be selected.

6.6.3 Privilege separation

Privilege separation is a technique in which programs are divided in to modules (sometimes executables), each having separate set of privileges. A module is expected to drop its privileges once its privileged task is completed. The technique is useful in cases where during a security attack/compromise of a module, Its privilege may not be enough to do serious damage.

There are few well known libraries available for fine-grained privilege separation such as Capsicum [72] for few operating systems. However for other operating systems and for applications running without an operating system, a technique given by Provos [73] may be applied:

“The goal of privilege separation is to reduce the amount of code that runs with special privilege. We achieve this by splitting an application into parts. One part runs with privileges and the others run without them. We call the privileged part the monitor and the unprivileged parts the slaves. While there is usually only one slave, it is not a requirement. A slave must ask the monitor to perform any operation that requires privileges. Before serving a request from the slave, the monitor first validates it. If the request is currently permitted, the monitor executes it and communicates the results back to the slave”.

The allowable request from a slave at any particular point is decided by a finite state machine maintained by the monitor. Any invalid request by slave is responded either by closing or restarting the connection.

A privilege separated slave can be in one of two states (Figure 13):

1. Pre-Authentication Phase: A slave has contacted the monitor but is not yet authenticated. In this case, the slave is chrooted in a empty directory, and the unprivileged slave has no process privileges and no rights to access the system.
2. Post-Authentication Phase: The slave has successfully authenticated to the monitor. The slave has the privileges of the user including file system access, but does not hold any other special privilege. However, special privilege are required to perform other privileged operations. For those operations, the slave must request an action from the privileged monitor.

If by some means if the attacker is able to compromise the slave, then the attacker is left with only two options:

1. Attempt to exploit OS bugs (if any) to gain higher privileges.
2. Attempt denial of services.

Mitigation techniques such as: (i) Do not spawns sub-processes before authentication, (ii) Do not allow unlimited authentication attempts; (iii) Use of a state machine to validate slave requests; can help limit the damage [74].

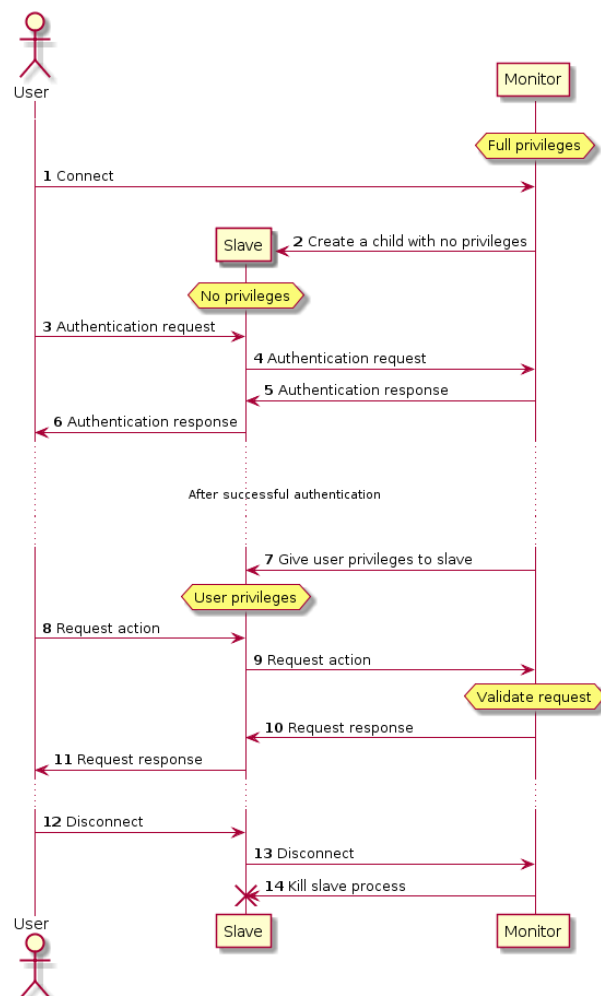


Figure 13 Privilege separation using monitor and slave.

Privilege separation provides security by reducing the attack surface; and the separation of privileged and non-privileged code lets reviewers focus on privileged part of the code. Also, if the monitor's state machine code is generated from a high-level description (a model) will make audits easier [74].

The flip side of this technique is that it requires the programmer/designer to separate the code into privileged and non-privileged modules and may add little performance penalty due to message passing.

7 REFERENCES

To number references use the Reference Style:

- [1] CECRIS, Deliverable D1.2, "Analysis of the interaction between safety and security concerning certified systems"
- [2] Kocher, Paul, et al. "Security as a new dimension in embedded system design." Proceedings of the 41st annual Design Automation Conference. ACM, 2004.
- [3] Richard Soja, "Automotive Security: From Standards to Implementation", Freescale white paper, 2014.
- [4] Eran Rippel, "Security Challenges in Embedded Designs", Discretix white paper, 2009.
- [5] CENELEC, EN 50159:2010: "Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems", 2010.
- [6] MITRE, "Threat Assessment & Remediation Analysis (TARA)", 2011.
- [7] CENELEC, "Smart Grid Information Security", 2012.
- [8] András Pataricza. "Model-based dependability analysis." DSc Thesis. Hungarian Academy of Sciences (2006).
- [9] Gábor Urbanics, László Gönczy, Balázs Urbán, János Hartwig, Imre Kocsis. (2014). Combined Error Propagation Analysis and Runtime Event Detection in Process-Driven Systems. In Software Engineering for Resilient Systems (pp. 169-183). Springer International Publishing.
- [10] BPMN 2.0 Standard. Object Management Group, 2011. <http://www.omg.org/spec/BPMN/2.0/>
- [11] Semantics Of A Foundational Subset For Executable UML Models (fUML), version 1.1. Object Management Group, 2013. <http://www.omg.org/spec/FUML/>
- [12] Narendra Jussien, G Rochart, and X Lorca. "The CHOCO constraint programming solver". In: CPAIOR'08 workshop on Open-Source Software for Integer and Constraint Programming (OSSICP'08). 2008.
- [13] G. Hastings, L. Montella, J. Watters, "MITRE Crown Jewels Analysis Process", MITRE Technical Report , 2009.
- [14] Gary Stoneburner, "Toward a Unified Security-Safety Model", IEEE Computer, Vol. 39, pp. 96-97, 2006.
- [15] NIST Special Publication 800-30, "Guide for Conducting Risk Assessments", September 17, 2012. http://www.nist.gov/manuscript-publication-search.cfm?pub_id=912091 (accessed Dec. 2014).
- [16] Federal Aviation Administration (FAA) "System Safety Handbook". http://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/ (accessed Dec. 2014).
- [17] Nicola Nostro, Andrea Bondavalli and Nuno Silva. "Adding Security Concerns to Safety Critical Certification". To appear in the Proceedings of the 4th edition of the IEEE International Workshop on Software Certification (WoSoCer2014). 2014.
- [18] Joint Task Force Transformation Initiative, "Security and privacy controls for federal information systems and organizations," National Institute of Standards and Technology, NIST SP 800-53r4, Apr. 2013.

- [19] CECRIS - Certifications of Critical Systems, "Deliverable D1.2: Analysis of the interaction between safety and security concerning certified systems," <http://www.cecris-project.eu>. 2014.
- [20] CECRIS - Certifications of Critical Systems, "Deliverable D1.3: Analysis and Assessment of Verification Techniques", <http://www.cecris-project.eu>. 2014.
- [21] International Organization for Standardization, <http://www.iso.org/iso/home/standards/management-standards/iso27001.htm>, (Accessed: Dec. 2014).
- [22] International Organization for Standardization, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50341, (Accessed: Dec. 2014).
- [23] RTCA DO-178B/EUROCAE ED-12B - Software Considerations in Airborne Systems and Equipment Certification, Dec.1992.
- [24] RTCA DO-178C/EUROCAE ED-12C - Software Considerations in Airborne Systems and Equipment Certification, Dec. 2011.
- [25] M. Schäfer, V. Lenders, and I. Martinovic, "Experimental analysis of attacks on next generation air traffic communication." In Proc. of the 11th Int. Conf. on Applied Cryptography and Network Security, Springer-Verlag, Berlin, Heidelberg, 253-271, 2013.
- [26] A. Costin and A. Francillon, "Ghost in the Air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices". Black Hat USA, 2012.
- [27] Donald G. Firesmith, Engineering Safety and Security Related Requirements for Software Intensive Systems, full day tutorial at the 29th International Conference on Software Engineering (ICSE) in Minneapolis, Minnesota on 21 May 2007
- [28] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of software engineering. Prentice Hall PTR Upper Saddle River, NJ, USA, 2002.
- [29] B. Boehm, "A view of 20th and 21st century software engineering," in Proceedings of the 28th international conference on Software engineering, New York, NY, USA, 2006, pp. 12–29.
- [30] R. S. Pressman, Software engineering: a practitioner's approach, 7th ed. New York: McGraw-Hill Higher Education, 2010.
- [31] G. McGraw, Software Security: Building Security In. Addison-Wesley Professional, 2006.
- [32] M. Howard and D. E. Leblanc, Writing Secure Code, 2nd ed. Redmond, Washington: Microsoft Press, 2002.
- [33] D. Stuttard and M. Pinto, The web application hacker's handbook: discovering and exploiting security flaws. Wiley Publishing, Inc., 2007.
- [34] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection," IEEE Trans. Dependable Secure Comput., vol. 11, no. 5, pp. 440–453, Sep. 2014.
- [35] OWASP Foundation, "Open Web Application Security Project," OWASP Foundation, 2001. [Online]. Available: <http://www.owasp.org/>.
- [36] M. Shema, Seven deadliest web application attacks. Burlington, MA: Syngress, 2010.
- [37] G. J. Myers, C. Sandler, and T. Badgett, The art of software testing. John Wiley & Sons, 2011.

- [38] A. Mukherjee and D. P. Siewiorek, "Measuring software dependability by robustness benchmarking," *IEEE Trans. Softw. Eng.*, vol. 23, no. 6, pp. 366–378, 1997.
- [39] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using Attack Injection to Discover New Vulnerabilities," in *International Conference on Dependable Systems and Networks*, 2006. DSN 2006, 2006, pp. 457–466.
- [40] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, 1976.
- [41] M. Curphey, D. Endler, W. Hau, S. Taylor, T. Smith, A. Russell, G. McKenna, R. Parke, K. McLaughlin, and N. Tranter, "A guide to building secure Web applications," *Open Web Appl. Secur. Proj.*, vol. 1, 2002.
- [42] D. P. Freedman and G. M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*. Dorset House Publishing Co., Inc., 2000.
- [43] University of Maryland, "FindBugsTM - Find Bugs in Java Programs," 2009. [Online]. Available: <http://findbugs.sourceforge.net/>. [Accessed: 12-Mar-2009].
- [44] Fortify Software, "Fortify 360 Software Security Assurance," 2008. [Online]. Available: <http://www.fortify.com/products/fortify-360/>. [Accessed: 08-Jun-2010].
- [45] Y. W. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo, "Securing web application code by static analysis and runtime protection," in *Proceedings of the 13th international conference on World Wide Web*, 2004, pp. 40–52.
- [46] M. Doliner, "Cobertura," 2006. [Online]. Available: <http://cobertura.sourceforge.net/>. [Accessed: 15-Jun-2010].
- [47] Atlassian, "Clover - Code Coverage for Java," 2010. [Online]. Available: <http://www.atlassian.com/software/clover/>. [Accessed: 15-Jun-2010].
- [48] Y. Shin, L. Williams, and T. Xie, "SQLUnitGen: SQL Injection Testing Using Static and Dynamic Analysis," in *Proceedings of the 17th IEEE International Conference on Software Reliability Engineering (ISSRE 2006)*, Raleigh, NC, USA, 2006.
- [49] D. Balzarotti, M. Cova, V. Felmetzger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," in *IEEE Symposium on Security and Privacy*, 2008. SP 2008, 2008, vol. 66, pp. 387–401.
- [50] Langner, Ralph (November 2013). ["To Kill a Centrifuge: A Technical Analysis of What Stuxnet's Creators Tried to Achieve"](http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf). (<http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf>)
- [51] ["Duqu: A Stuxnet-like malware found in the wild, technical report"](http://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf). Laboratory of Cryptography of Systems Security (CrySyS). 14 October 2011. (<http://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf>)
- [52] Klein, Gerwin, et al. "seL4: Formal verification of an OS kernel." *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009.
- [53] Potts, Daniel, et al. "Mathematically verified software kernels: Raising the bar for high assurance implementations." (2014). (<http://sel4.systems/Docs/GD-NICTA-whitepaper.pdf>)
- [54] Greenhills software. INTEGRITY Homepage <http://www.ghs.com/products/rtos/integrity.html>
- [55] INTEGRITY Global Security. "The Gold Standard for Operating System Security: SKPP" (http://www.ghs.com/download/whitepapers/GHS_SKPP_Gold_Std.pdf)

- [56] SYSGO. PikeOS™ SIL 4 certification on multi-core platform. October 23, 2013 (<http://www.sysgo.com/news-events/press/press/details/article/pikeosTM-sil-4-certification-on-multi-core-platform/>)
- [57] De Raadt, Theo. "Exploit mitigation techniques." (2005). (<http://www.openbsd.org/papers/ven05-deraadt/>)
- [58] Matthieu Herrb, et al. Protection measures in OpenBSD. November 2009 (<http://homepages.laas.fr/matthieu/talks/openbsd-h2k9.pdf>)
- [59] McCarty, Bill. SELinux. No. s 16. O'Reilly Media Inc, 2004.
- [60] Fox, Michael, et al. Selinux and grsecurity: a side-by-side comparison of mandatory access control and access control list implementations. Tech. Rep. URL <http://www.cs.virginia.edu/~jcg8f/GrsecuritySELinuxCaseStudy.pdf>, 2008.
- [61] Watson, Robert, et al. "Design and implementation of the trusted BSD MAC framework." DARPA Information Survivability Conference and Exposition, 2003. Proceedings. Vol. 1. IEEE, 2003.
- [62] Schreuders, Z. Cliffe, Tanya McGill, and Christian Payne. "Empowering end users to confine their own applications: The results of a usability study comparing SELinux, AppArmor, and FBAC-LSM." *ACM Transactions on Information and System Security (TISSEC)* 14.2 (2011): 19.
- [63] Lyon, Gordon Fyodor. Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure, 2009.
- [64] Maynor, David. Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research. Elsevier, 2011.
- [65] Deraison, Renaud. "Nessus Security Scanner. The Nessus Homepage." (<http://www.Nessus.Org>)
- [66] CVE Details for operating systems (http://www.cvedetails.com/product-list/product_type-o/vendor_id-0/firstchar-A/Operating-Systems.html)
- [67] Center, CERT Coordination. "Vulnerability notes database." *CERT Coordination Center*. (<http://www.kb.cert.org/vuls>)
- [68] BugTraq archives homepage (<http://www.securityfocus.com/archive/1>)
- [69] Ted Unangst. *Random in the wild* (<http://www.tedunangst.com/flak/post/random-in-the-wild>) Accessed on 26th January 2015.
- [70] Haahr, Mads. "Random. org: True random number service." School of Computer Science and Statistics, Trinity College, Dublin, Ireland. Website (<http://www.random.org>).
- [71] [RFC 4086](http://tools.ietf.org/html/rfc4086) (<http://tools.ietf.org/html/rfc4086>)
- [72] Watson, Robert NM, et al. "Capsicum: Practical Capabilities for UNIX." *USENIX Security Symposium*. 2010. (http://static.usenix.org/legacy/events/sec10/tech/full_papers/Watson.pdf)
- [73] Niels Provos, Markus Friedl and Peter Honeyman, *12th USENIX Security Symposium*, Washington, DC, August 2003. (<http://www.citi.umich.edu/u/provos/papers/privsep.pdf>)
- [74] Miller, Damien. "Security measures in OpenSSH." (2007). (<http://openbsd.ukbsd.org/papers/openssh-measures-asiabsdcon2007.pdf>, <http://www.openbsd.org/papers/openssh-measures-asiabsdcon2007-slides.pdf>)