D1.2

List of tools and techniques applicable for high and medium assurance for efficient assurance

Project number:	731456
Project acronym:	certMILS
Project title:	Compositional security certification for medium to high-assurance COTS-based systems in environments with emerging threats
Start date of the project:	1st January, 2017
Duration:	48 months
Programme:	H2020-DS-LEIT-2016

Deliverable type:	Report
Deliverable reference number:	DS-01-731456 / D1.2 / V1.0
Work package contributing to the deliverable:	WP1
Due date:	Dec 2017 – M12
Actual submission date:	20th December 2017

Responsible organisation:	SRO
Editor:	Jan Rollo
Dissemination level:	PU
Revision:	V1.0

Abstract:	Different techniques to achieve medium and high-assurance for security evaluation are described
Keywords:	medium assurance, high-assurance, applicable techniques, Common Criteria, IEC 62443, testing, modelling

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731456.

Editor

Jan Rollo (SRO)

Contributors (ordered according to beneficiary numbers)

Amelia Alvarez de Sotomayor, Benito Caracuel (SCHN)

Alvaro Ortega (E&E)

Reinhard Hametner (THA)

Sergey Tverdyshev, Holger Blasum (SYSGO)

Tomáš Kertis (UCO)

Jan Rollo, Oto Havle (SRO)

Thorsten Schulz (UROS)

Michal Hager (EZU)

Executive summary

This deliverable summarizes consortium experience and expectations for a number of tools that can support high-assurance development for embedded systems.

Contents

List of figures

List of tables

Chapter 1 Introduction

The certMILS project targets medium and high assurance security certification. Security assurance can be gained by a large number of analysis methods. This document serves to establish a baseline of state-of-the-art analysis techniques that can be considered during a later phase of the project to support establishing such assurance, for the MILS separation kernel, one or several of the prototypes.

This deliverable has a simple structure: the main part is the list of tools and technologies in Chapter 2; that is any subsection(s) of Chapter 2 can be read independently. Chapter 3 gives a (short) discussion how the results can be used further on. Chapter 4 contains a list of abbreviations.

Chapter 2 Tools and techniques

This section summarizes the experience of partners with certain tools and techniques.

The section has been originally based on a template that had been distributed to some certMILS partners, to encourage reporting on different tools. The template-based approach is the reason why each section of Chapter 2 has many subsections. We generally avoided merging subsections to maintain easy comparability, with the only exception that empty subsections have been deleted. The template itself is reproduced in the appendix, see Chapter 5. In order not to stifle creativity by a too formal approach, it was intentionally left optional, whether a description was more based on a specific tool or a specific technique. For each tool / technique, we asked, how used tool(s) have been acquired, which safety/security properties that can be asserted by the tool, what is the usage experience, and, if known, specific usage for Common Criteria and IEC 62443, and/or other safety/security standards, including, if applicable, compositional certification.

Tools and techniques have been grouped into:

- Static analysis (Section 2.1): Astrée (Section 2.1.1), complexity metrics (Section 2.1.2)
- Formal models (Section 2.2): Isabelle/HOL (Section 2.2.1)
- Security testing (Section 2.3): OpenVAS (Section 2.3.1), Achilles (Section 2.3.2), fuzzing (Section 2.3.3)
- Documentation and assurance creation (Section 2.4): DOORS (Section 2.4.1), medini (Section 2.4.2)

2.1 Static analysis

2.1.1 Taint analysis

Taint analysis is a static analysis that consists in tracking how information flows between different parts of a program or system. The goal is to show whether some untrusted inputs (so-called tainted inputs) may interact with and harm sensitive parts of the systems (the so-called sinks). Taint analysis can be expressed in an abstract interpretation framework, allowing for sound and efficient integration with existing abstract domains, such as those used by Astrée.

2.1.1.1 Tool acquisition

Astrée is commercial software developed by AbsInt Angewandte Informatik GmbH [1]. In the ARAMiS II project, AbsInt and SYSGO want to explore the use of taint analysis in a safety analysis of an embedded operating system. AbsInt will develop the taint analysis, whereas SYSGO will apply the analysis on the source code of a separation kernel.

2.1.1.2 Tool characterization

Astrée is a logically sound static analyser designed to prove the absence of runtime errors and further critical program defects. It is based on the abstract interpretation of the C programming language.

Astrée shares the front-end and user interface with RuleChecker, a static analysis tool that checks compliance to coding standards, such as MISRA C 2012 and SEI CERT C.

2.1.1.3 Properties that can be asserted

Properties that can be asserted are:

- Two processes running on the same system are isolated: that is they must not be able to write into each other’s memory at all.

- Two processes are separated under control of the operating system: That is the two processes can only interact with proper checks from the operating system. In that case, a variable may be a taint source for a process, but a sink for another one.

2.1.1.4 Usage experience

2.1.1.4.1 Base components

2.1.1.4.1.1 Input and its preparation

The inputs for Astrée are the C language source files of the main part of an embedded real-time separation kernel. Parts written in assembly, platform-dependent code, scheduler, initialization code, and kernel drivers are excluded and replaced with suitable stubs. The Astrée analysis entry point is a stub main function, which invokes all functions that implement system calls. The assert() statements are adapted, an assertion failure in the separation kernel source code leads to an alarm reported by Astrée.

2.1.1.4.1.2 Output and its interpretation

Astrée shows list of alarms after the analysis terminates. Alarms include C language undefined behaviour and assertion failures. Astrée contains RuleChecker, which is used to check coding standard compliance of the separation kernel's source code.

In addition, taint analysis is being developed. In its current form, that taint analysis is not able to analyse the whole separation kernel without false positives. AbsInt reported some success with analysis of the separation kernel initialization phase. The taint sources will be the system call arguments and the copy-from-user functions (i.e. functions that copy from a non-separation kernel user address space to an address space of the separation kernel), the taint sinks will be the system call return values and the copy-to-user functions. The data stored in the operating system's thread descriptors will be considered as taint sinks. Further development will be needed regarding abstraction of pointers to thread descriptors and a relational abstract domain for tainting.

2.1.1.4.2 MILS systems

Astrée taint analysis is used to prove separation of a Multiple Independent Levels of Security and Safety (MILS) separation kernel, i.e. that the separation kernel properly propagates the separation of the MILS system to its internals. This is the very heart of MILS systems.

2.1.1.5 Use in certification

Taint analysis is a new technique specifically valuable for MILS systems, and, to the best of our knowledge, in the form of "taint analysis" not yet specifically demanded by generic safety and security standards. However, taint analysis is an optional high-assurance means of confirming partitioning integrity, as required by e.g. DO-178 [2].

2.1.1.6 Use in certification: Common Criteria

Astrée is a useful tool that can be used under the vulnerability analysis scope of Common Criteria for finding implementation-type vulnerabilities such as buffer overflows.

2.1.2 Complexity metrics

A complexity metric is used to evaluate code understandability by evaluation of structural or syntactical code properties.

2.1.2.1 Tool acquisition

The pmccabe tool can be obtained from the tool's web site [3].

2.1.2.2 Tool characterization

A wide range of complexity measures exists. In this section, we describe the McCabe cyclomatic complexity measure [4], as it is conceptually simple to explain and widely used [5]. The pmccabe tool computes the number of code execution branches, v(G) = e - n + 2 p where:

- p: parts = connected components / functions [single entry + single exit]
- e: edges
- n: nodes

v(G) is 1 for a non-branching function. The number of code execution branch corresponds to the number of basic blocks produced by a compiler, as well as the number of test cases that have to be generated to achieve decision code coverage.

2.1.2.3 Properties that can be asserted

Complexity of code according to a metric.

2.1.2.4 Usage experience

The pmccabe tool is simple to install and use:

```
$cat branching.c
#include "stdio.h"

void nobranch() {
        return;
}

void branch(intprototype_id) { // line 8
        if (prototype_id == 1) { // line 9
                printf("smart grid"); // line 10
        } //line 11
        if (prototype_id == 2) { //line 12
                printf("subway"); //line 13
        } //line 14
        if (prototype_id == 3) { //line 15
                printf("railway"); //line 16
        } //line 17
} //line 18
$ sudo apt-get install pmccabe
$ pmccabe branching.c
1       1       1       3       3       branching.c(3): nobranch
4       4       6       7       11      branching.c(7): branch
```

Figure 1: Session with pmccabe

Figure 1 shows a session with pmccabe. The first column of the output shows the non-branching function `nobranch` has a McCabe complexity of 1 (n=1, e=0, p=1), whereas the function `branch`, which has three if statements, has a McCabe complexity of 4 (n=7, e=9, p=1; see Figure 2). In his

original paper McCabe suggested that the cyclomatic complexity should not exceed a value of 10, other tools, e.g. by Microsoft, recommend a maximum of 25.

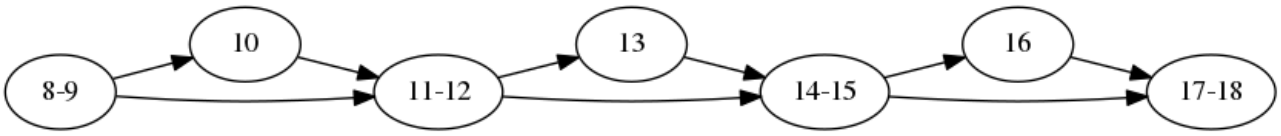

Figure 2: Branching diagram of function `branch`: 7 nodes (showing the line numbers given in Figure 1) and 9 edges

The McCabe complexity measure is easy to calculate, but caution is advised [6][7][8][9][10]. For instance, when we experimentally applied pmccabe to the source code of a separation kernel, a rather high value came out for the system initialization routine. However, system initialization is just a sequence of actions, with platform and configuration dependent parameters that are executed sequentially, and the code is not very hard to understand. For operating system code, Jbara [11] has observed that many operating systems (Linux, BSD, Windows) have high cyclomatic complexity e.g. due to long switch tables, and that "simplifying" away the cyclomatic complexity would probably not improve readability. For instance, structural code complexity is balanced by code regularity [12]. Structural code complexity strongly correlates with potentially even simpler parameters such as lines of code [8][9][13]. Moreover, in addition to complexity at function level there is also complexity at design level [14][15][16] and low complexity at function level may lead to high complexity at design level.

2.1.2.5 Use in certification: Safety certification (such as e.g. IEC 61508 [17], DO-178 [2], IEC 62290 [18], etc.)

2.1.2.5.1 IEC 61508

IEC 61508 Part 7 C.5.13 describes "Complexity metrics". According to IEC 61508 Part 3 Table B.9, the technique is recommended for SIL 1 and SIL 2 and highly recommended for SIL 3 and SIL 4.

IEC 61508 Part 7 C.5.13 states that the aim of complexity metrics is "*to predict the attributes of programs from properties of the software itself or from its development or test history*" and goes on with the following description: "*These models evaluate some structural properties of the software and relate this to a desired attribute such as reliability or complexity. Software tools are required to evaluate most of the measures. Some of the metrics, which can be applied, are given below:*

- *Graph theoretic complexity – this measure can be applied early in the lifecycle to assess trade-offs, and is based on the complexity of the program control graph, represented by its cyclomatic number.*
- *Number of ways to activate a certain software module (accessibility) – the more a software module can be accessed, the more likely it is to be debugged.*
- *Halstead type metrics science – this measure computes the program length by counting the number of operators and operands; it provides a measure of complexity and size that forms a baseline for comparison when estimating future development resources.*
- *Number of entries and exits per software module – minimising the number of entry/exit points is a key feature of structured design and programming techniques.*"

For further background on metrics, IEC 61508 references a book by Kan on "metrics and models on software" [19].

2.1.2.5.2 DO-178C [2]

DO-178 Section 5.2.2 "Software Design Process Activities" states, "*the current state of software engineering does not permit a quantitative correlation between complexity and the attainment of system safety objectives. While no objective guidance can be provided, the software design process should avoid introducing complexity because as the complexity of software increases, it*

becomes more difficult to verify the design and to show that the safety-related requirements are satisfied"

Further mentions are Section 6.3.4 d: "*Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, for example, complexity restrictions and code constraints. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified.*" and 11.7 f. "*Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components.*"

2.1.2.6 Use in certification: Common Criteria [20]

CC, Part 3, Section A.3 "ADV_INT: Supplementary material on TSF internals" states that the "*wide variety of TOEs makes it impossible to codify anything more specific than "well-structured" or "minimum complexity".* The CC states also suggests to use a modular design and justifies this in the following way: "*Software written with a modular design aids in achieving understandability by clarifying what dependencies a module has on other modules (coupling) and by including in a module only tasks that are strongly related to each other (cohesion).*" [CC, Part 3, Section A.3.1]. At the same place, the CC caution [CC, Part 3, Section 3.2]: "*Modules that are mutually dependent may rely on one another to formulate a single result, which could result in a deadlock condition, or worse yet, a race condition (e.g., time of check vs. time of use concern), where the ultimate conclusion could be indeterminate and subject to the computing environment at the given instant in time.*"

2.1.2.7 Use in certification: IEC 62443 [21]

We have not found a discussion of complexity metrics in IEC 62443.

2.2 Formal models

2.2.1 Isabelle/HOL

Isabelle/HOL is an interactive theorem prover.

2.2.1.1 Tool acquisition

Isabelle/HOL is developed under a BSD license and can be downloaded free of charge [22].

2.2.1.2 Tool characterization

Isabelle/HOL is equipped with a GUI that simplifies writing proof scripts that are checked by the tool interactively. The user suggests proof steps for a theorem and Isabelle/HOL checks whether these proofs steps are correct and gives visual feedback. Isabelle/HOL also can be used non-interactively to check proofs. Moreover, Isabelle/HOL can export proof scripts to PDF documents.

2.2.1.3 Properties that can be asserted

The logic used by Isabelle/HOL (HOL, higher-order logic) is expressive enough so that it can be used as an axiomatic foundation for mathematics [23]. In the software domain, it is possible to express negative requirements and non-functional properties [24], Section 1.4.

Thus, in MILS systems, Isabelle/HOL is commonly used to formulate top-level properties such as

- data separation and non-interference,
- temporal separation [25].

Isabelle/HOL can also express low-level or intermediate properties, e.g. what a part of a separation kernel does, what invariants it maintains etc.

It is always possible to replace a higher-level description of properties by a lower-level description and to show (or attempt to show) that the lower-level description implies the higher-level description. This process is called refinement.

Code level-verification by a combination of frameworks to derive Isabelle from Haskell and to derive C from Haskell has been used in the verification of other operating system code (seL4 kernel, [26]).

2.2.1.4 Usage experience

2.2.1.4.1 Base components

2.2.1.4.1.1 Input and its preparation

In the FP7 project EURO-MILS, SYSGO and SRO prepared a top-level model of a separation kernel. The top-level model was based on previous work on separation kernels by Rushby [27]. The top-level model includes support for pre-emption and interrupts [28]; concrete proof obligations are given for pre-emption on single-core [29] and multi-core [30].

The second input was the formal specification of part of the API of the separation kernel. Partners have created proofs that the formal API specification of the API of the separation kernel implies the top-level model for any configuration provided by the system integrator, i.e. there is no illicit information flow, which is not allowed by the integrator.

The proof is structured along different API invocations ([28], Section 2.3), claiming invariants such as data separation (no infiltration = “locally respects”, no exfiltration = “view-partition equivalent” in [28], Section 2.3) and information flow on each. Reasoning on API invocations can have dependencies (e.g. the IPC API depends on the memory API), which are reflected in the proofs.

2.2.1.4.1.2 Output and its interpretation

As outlined in [28], in EURO-MILS the output was that

- the theorems describing the top-level property held within the top-level theory and
- that the functional specification of the separation kernel satisfied the assumptions of the top-level theory,

from which followed that the functional specification of the separation kernel satisfied the property for information flow control of the top-level theory, which implies also the property of separation of data.

2.2.1.4.2 MILS systems

2.2.1.4.2.1 Input and its preparation

In EURO-MILS, a firewall has been modelled on top of the separation kernel [31].

2.2.1.4.2.2 Output and its interpretation

In a study of a firewall based on separation kernel presented at the MILS workshop [31], a MILS system, which consisted of a firewall and untrusted applications was described. It was shown that the MILS system behaved according to the firewall’s configuration.

2.2.1.4.3 Usability/scalability/interoperability

A comfortable GUI of Isabelle/HOL is available for Linux, MacOS and Windows. However, the activity of establishing formal invariants is demanding, the effort for generating proofs is large, e.g. the generic multicore theory MCISK [29] consists of 9367 lines of code and the instantiation by the separation kernel is even larger than that ([28], Section 3.8).

A limitation of logical computer-based reasoning in general is that once a model is unsatisfiable (has a contradiction), any conclusion holds. A model can be shown satisfiable by providing a witness. As EURO-MILS has only used definitional reasoning (no axiomatizations), Isabelle/HOL allowed to do a relatively small consistency proof ([29], file Step_locale.thy).

A limitation of refinement approaches (in general) is that it is easy to overlook underlying properties, e.g. a memory controller shared between partitions can bypass partitioning provided by the separation kernel.

The security domains that were underlying the information flow analysis were that each partition was a security domain, and information flow between partitions either existed or not. This model of security domains and their interaction might be overly crude (many ways of separation with controlled interaction exist).

2.2.1.5 Use in Common Criteria [20]

Isabelle is explicitly mentioned in the CC [20], Part 3. Several published STs mention Isabelle [32], [33], [34], or, as similar tool, Coq [35]. The CC only requires formal models for levels of Evaluation Assurance Level (EAL) 6 and higher (the CC assurance family for this is called ADV_SPM where SPM means security policy modelling), thus, for the CC evaluation done in certMILS, it is not mandatory to have a formal security policy model for the EAL levels targeted by certMILS.

2.2.1.5.1 Base components

In the EURO-MILS project, rules for using Isabelle/HOL in a CC-conformant way have been formulated, including their direct application to the artefacts in the form of a compliance statement to a style guide ([24], Sections 1.7.2 and 3). In EURO-MILS, a self-assessment exercise, whether the EURO-MILS R&D effort would fully meet the “production” guidelines for ADV_SPM by ANSSI and BSI was done ([36], Section 2.3.2). On the positive side, most other requirements of the ANSSI and BSI guidelines were fully matched.

2.2.1.5.2 Compositional certification: MILS systems

The CCDB [37] demand to “determine whether the application uses services of the underlying platform within its own Composite-ST to provide domain separation, self-protection, non-bypassability and protected start-up; if no, there is no further composite activities for ADV_ARC” (paragraph 76); that “*the evaluator shall examine the statement of compatibility to determine that the Platform-TSF being used by the Composite-ST is complete and consistent for the current composite TOE*” (ASE_COMP.1-2) and that “*the evaluator shall examine the statement of compatibility to determine that the relevant organisational security policies of the Platform-ST are not contradictory to those of the Composite-ST*” (ASE_COMP.1-7).

A formal model can ease to make statements on consistency and completeness.

2.2.1.6 Use in IEC 62443 [21]

Surprisingly, there appears to be no direct mention of Isabelle, HOL, or even “formal methods” in IEC 62443 [21]. However, HOL is directly mentioned in IEC 62443’s parent standard, IEC 61508 [17], Part 7, C.2.4.4.

2.2.1.6.1 Base components

Isabelle may be used for a base component, giving credits for, e.g., IEC 62443 Part 4-1 [38] Section 7.3.1, SR-2 Threat model, where “*all products shall have an up-to-date threat model with the following characteristics: a) correct flow of categorized information throughout the system: x) trust boundaries; y) processes; z) data stores*” as the information flow can be directly modeled as a formal model.

2.2.1.6.2 MILS system

The MILS design eases to satisfy functional and assurance requirements for IEC 62443 [39].

Isabelle may be used for a MILS system, e.g., in the context of IEC 62443 Part 4-1 [38], Section 7.3.1, SD-2 “*Defence in Depth Design*”, Isabelle may give credits by giving high assurance that a composition of layers cannot be compromised, which backs *“A process shall be employed for including multiple layers of defense where each layer provides additional defense mechanisms. Each layer should assume that the layer in front of it may be compromised. Secure design principles are applied to each layer*”.

2.2.1.7 Related tools

Many other interactive theorem provers exist, an overview is given by Wiedijk [40]. Other interactive theorem provers used on operating system code are ACL2 and Coq. Together with ACL2, and Coq, Isabelle/HOL seems to be among the market leaders in interactive theorem proving.

2.3 Security testing

2.3.1 OpenVAS

OpenVAS is a framework to perform vulnerability scans and solution management.

2.3.1.1 Tool acquisition

The tool can be downloaded from OpenVAS official site: http://www.openvas.org/

2.3.1.2 Tool characterization

OpenVAS (Open Vulnerability Assessment System) is an open source framework composed of several services and tools to perform vulnerability scans and solution management.

The OpenVAS security scanner is continually updated with new Network Vulnerability Tests (NVTs). Currently, there are over 50,000 NVTs included in total.

All OpenVAS products are Free Software. Most components are licensed under the GNU General Public License (GNU GPL).

OpenVAS used the Common Vulnerability Scoring System (CVSS). CVSS is an industry standard for the classification and rating of vulnerabilities that assists in prioritizing the remediation measures.

2.3.1.3 Properties that can be asserted

OpenVAS assesses vulnerabilities considering three different perspective of an attacker:

- External: The system is attacks the network externally. This way it can identify badly configured or misconfigured firewalls.
- DMZ: Through this type of tests, OpenVAS can identify actual vulnerabilities of the system under assessment that could be exploited by attackers if they get past the firewall.
- Internal: OpenVAS also includes tests to evaluate vulnerabilities in the case that attacks are executed internally by insiders through methods of social engineering or a worm. This perspective is considered crucial for the security of the IT infrastructure.

For DMZ and internal scans, they can be differentiated between authenticated and non-authenticated scans. When performing an authenticated scan the OpenVAS uses credentials and can discover vulnerabilities in applications that are not running as a service but have a high-risk potential. This includes web browsers, office applications or document viewers.

2.3.1.4 Usage experience

2.3.1.4.1 Base components

2.3.1.4.1.1 Input and its preparation

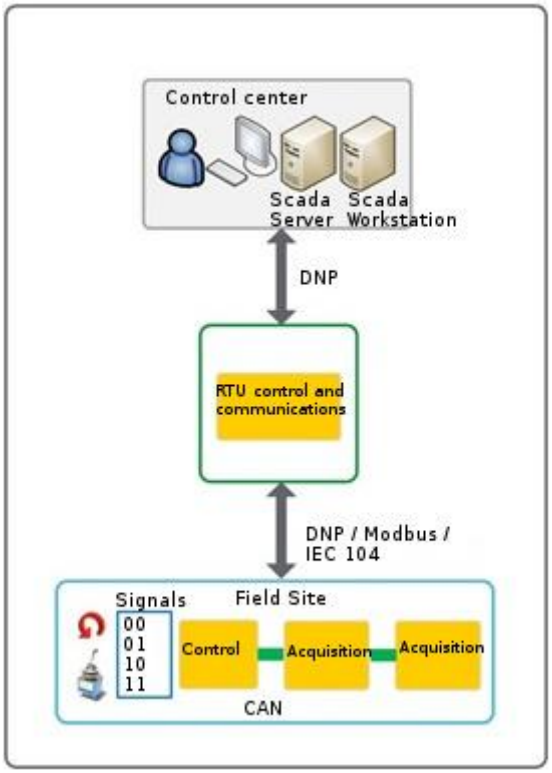

Figure 3: Test bed emulating an ICS

OpenVAS was tested on a test bed emulating an ICS (Industrial Control System) basic architecture with basic functionalities for substation monitoring and control [41] shown in Figure 3. This test bed was composed as follows:

- **Field Site Level:** This level was composed by the Acquisition System. This system implements the acquisition functions of the test bed. It consists of a set of devices that provide real-time control and automation applications. The system included the following elements:

  Control RTU: the CPU module performs the control functions, centralizing the information acquired by acquisition modules and executing the programmable logic control, the communication protocols and user specific applications. The CPU module provides a wider range of functionalities, especially in terms of communication protocols, serial and Ethernet communication ports and synchronization mechanisms.

  Acquisition RTU: it consists of I/O modules, which are connected to the CPU module and perform data acquisition and pre-process signals, control and execute commands to field devices.

- **Communication Centre Level:** The communication centre level of the test bed performed the communication interface between the Acquisition System and the SCADA System.

  It consisted of a control/communication RTU, with several communication protocols, serial and Ethernet communication ports and synchronization mechanisms.

- **Control Centre:** At the higher level, the test bed implemented the SCADA, which was responsible for real-time data gathering, interactive device control, alarm notification and response, historical data storing and automated reporting.

  The SCADA System included the following elements:

  SCADA server: it is a real-time database and program package that collects data from the Acquisition System, checks for alarm conditions, scales values, drives devices, provides storage space and enables the user to send out commands to field devices in the Acquisition System. SCADA server communicates with the RTU sending commands and gathering system information.

  HMI: it is the Graphic User Interface, used to interact with the SCADA. HMI lets operators and authorized users to interact with the other components of the SCADA System.

  Data Historian: it provides the storage space for historical data as well as the data mining capabilities to generate reports. Data Historian is connected to the SCADA Server to transfer information from the real-time into the historical system.

In addition to the components emulating the ICS, the test bed included the following components:

- **Laboratory Agent:** the role of this agent is to carry out the security tests on the test bed, so this is the component where the OpenVAS framework was deployed.

  This component was connected to the emulated ICS with a VPN connection.

- **RTU Configuration and Management Tools:** it consisted on a set of monitoring, maintenance and configuration tools for RTU devices.

The deployed ICS allowed Schneider Electric to test vulnerabilities for the operation of the integrated system as well as for the following specific targets:

- **Communications**: Ethernet, Serial
- **Protocols**: DNP 3.0, IEC 104, Modbus, Profibus, CAN

2.3.1.4.1.2 Output and its interpretation

After using OpenVAS scanner on the architecture described above, we obtained a vulnerability report. This report included a summary of the problem, the possible causes and risks. At the end of the report, solutions that can be adopted are suggested.

After following instructions provided by the report and performing a new scan, the vulnerability was solved.

2.3.1.5 Use in certification: Safety certification (IEC 61508 [17], DO-178 [2], IEC 62290 [18], etc.)

OpenVAS is not used for Safety certification

2.3.1.6 Use in certification: Common Criteria [20]

OpenVAS could be used to detect vulnerabilities of the product.

2.3.1.7 Use in certification: IEC 62443 [21]

OpenVAS could be used to detect security vulnerabilities of the product.

2.3.1.8 Use in certification: Other Security certification (IEC TS 62531 [42], DIN VDE V 0831-104 [43])

OpenVAS could be used to detect security vulnerabilities of the product.

2.3.1.9 Related tools

The following tools were used together with OpenVAS Scanner to be able to perform more tests:

- Nmap: Free and open source utility for network discovery and security auditing.
- Nikto: Web server scanner that tests web servers for dangerous files/CGIs, outdated server software and other problems.
- ike-scan: Command-line tool that uses the IKE protocol to discover, fingerprint and test IPSec VPN servers.
- Snmp: Protocol created to monitor network.
- Pnscan: Scanner for TCP network services.
- Amap: Application protocol detection tool.
- w3af-console: Web application security scanner.
- Ncrack: Network authentication cracking tool used to perform high speed parallel cracking using an engine adaptable to different network situations.
- ldap-utils: Utilities to access a local or remote LDAP server.
- Phrasendrescher: Modular and multi-processing pass phrase cracking tool.
- Smbclient: Ftp-like client to access SMB/CIFS resources on servers.
- wmi-client: Tool to remotely execute commands and query parameters on a Windows Host.
- Dirb: Web content scanner that looks for existing and/or hidden web objects.
- Arachni: Web Application Security Scanner Framework.
- Dsniff: Collection of tools for network auditing and penetration testing.
- Wapiti: Web application vulnerability scanner.

2.3.2 Achilles

Achilles Test Platform detects vulnerabilities on communication interfaces, including network monitoring and evaluation of operational parameters.

2.3.2.1 Tool acquisition

The tool can be bought at https://www.ge.com/digital/products/achilles-vulnerability-testing-platform. The product started as a bare software tool and has grown into a solution with an additional hardware platform.

2.3.2.2 Tool characterization

The Achilles Test Platform offers manufacturers of critical infrastructures an efficient tool to test their products for vulnerabilities to cyber-attacks.

The test platform is focused on communication robustness, being able to monitor both network and operation parameters. Thus, allowing vulnerabilities to be discovered, faults to be reproduced, isolated, identified and resolved before products are commercialized.

It is mainly used to evaluate products robustness according to ISASecure's EDSA criteria.

2.3.2.3 Properties that can be asserted

Achilles Test Platform is focused on detecting vulnerabilities on communication interfaces, including network monitoring and evaluation of operational parameters.

As the Achilles Test Platform is used for certification of products in accordance with Achilles Communication Certifications and ISASecure EDSA Communication Robustness Testing (CRT) component, it is provided with tests related to industry-recognized standards.

In addition, it covers a very extensive set of control protocols, with tests designed specifically for devices found in critical infrastructures and addressing real-life scenarios in the field.

The tests are classified as follows:

- **Achilles Grammars**: It is used to evaluate protocol boundary conditions in device communications. It consists of systematic iterations over each field and combinations of fields to produce repeatable, quantifiable tests of the common types of implementation errors. During these tests, invalid, malformed or unexpected packets are sent to the Device Under Test to detect vulnerabilities in specific layers of the protocol stack.
- **Achilles Storms**: The module generates packets at a high rate to evaluate the ability of the Device Under Tests to handle high traffic rates for different communication protocols. It also includes the ability to search for the denial-of-service threshold for a given type of storm traffic; this is the storm rate at which the device is no longer able to respond normally to other requests.
- **Known Vulnerabilities**: These test cases exploit traffic for categorized vulnerabilities with high probability of existence in control devices.

Figure 4 shows the communication protocols for critical infrastructures that can be evaluated with this platform:

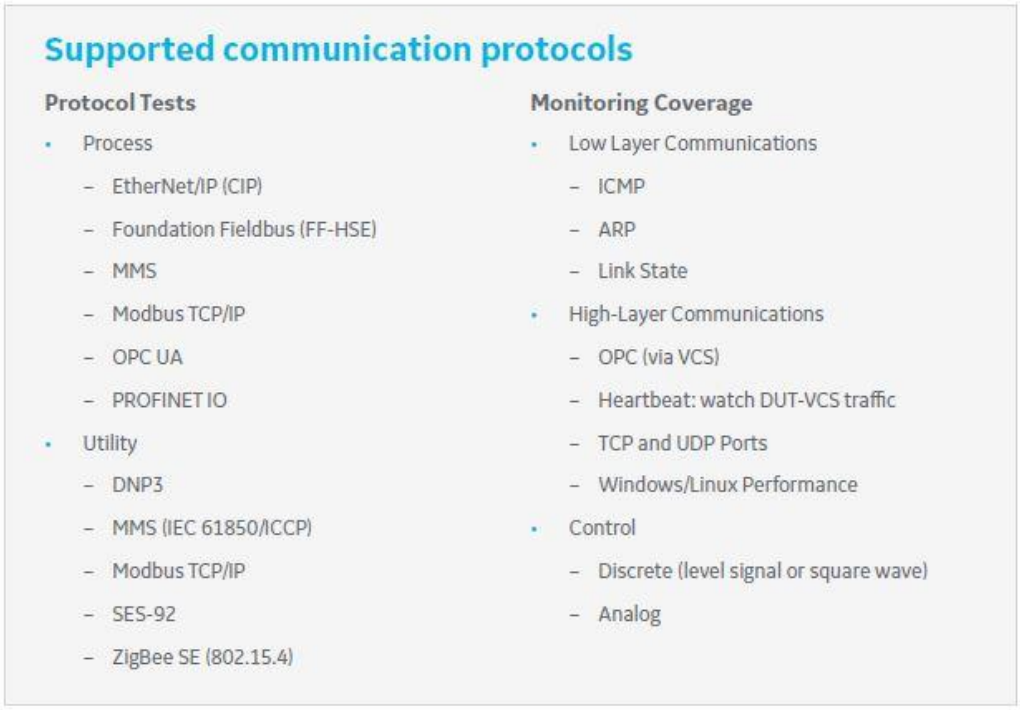

Figure 4: Communication protocols supported by the Achilles platform

2.3.2.4 Usage experience

2.3.2.4.1 Base components

2.3.2.4.1.1 Input and its preparation

While OpenVAS was used by Schneider Electric to evaluate vulnerabilities of operating systems and applications, the Achilles Testing Platform was used for evaluating robustness of Schneider Electric devices in accordance with level 2 of ISASecure EDSA Communication Robustness Testing (CRT).

Devices that were tested with the Achilles Testing Platform are: one RTU of Saitel DR family (HU_A model) with VxWorks operating system and one RTU of Saitel DP family (SM_CPU866e) with Linux operating system.

Both RTU were configured with Schneider Electric configuration tools to have enabled the following communication interfaces and protocols:

- Process: EtherNet/IP
- Low layer communications: ARP, ICMP
- High layers communication: TCP and UDP ports
- Utility: DNP3 and Modbus/TCP

2.3.2.4.1.2 Output and its interpretation

Taken into account that the Achilles Testing Platform performs security tests focused on network protocols (ARP, TCP, UDP, DNP3.0, etc.) and not on services, the obtained results were not able to detect vulnerabilities on services. Nevertheless, it operated adequately to detect weaknesses in protocol implementations, at both software and hardware level.

The fault reporting generated with the tool allowed us to identify some minor faults. The tool shows packet captures and reports with the required information to be able to reproduce problems and solve them.

2.3.2.5 Use in certification: Safety certification (IEC 61508 [17], DO-178 [2], IEC 62290 [18], etc.)

Achilles is not used for Safety certification.

2.3.2.6 Use in certification: Common Criteria [20]

Achilles can be used under the vulnerability analysis scope of Common Criteria for TOEs implementing communication security.

2.3.2.7 Use in certification: IEC 62443 [21]

The Achilles Test Platform provides the exact test suites utilized in the ISA Secure EDSA Communications Robustness Testing Component, in addition to ACC Level 1 and ACC Level 2 test suites (Achilles Communication Certification).

2.3.2.8 Use in certification: Other Security certification (IEC TS 62531 [42], DIN VDE V 0831-104 [43])

GE Digital Cyber Security offers two certification programs: Achilles Practices Certification (APC) and Achilles Communications Certification (ACC). APC verifies that an organization employs industry standard best practices for security, while ACC verifies the network robustness of industrial devices. Both help address security and robustness in the development lifecycle.

Achilles Communications Certification (ACC) from GE Digital is designed to assess the network robustness of industrial devices and certify that they remain operational when subjected to network tests. ACC provides device manufacturers with an independently verified result to communicate product robustness to customers while providing confidence to control system operators in the products and systems they deploy and use.

Besides ISA Secure EDSA CRT test suites, Achilles test platform also performs test suites utilized in ACC Level 1, ACC Level 2.

2.3.2.9 Related tools

No additional tools are mentioned.

2.3.3 Fuzzing

This section covers tools related to the dynamic analysis aspect of security testing, the vulnerability testing (SV-3), in particular fuzz-testing. Fuzzing is an advanced testing technique that has received rising attention in recent years by identifying security vulnerabilities overlooked by other techniques. A fuzzer artificially generates randomly deviated data and feeds this into the test target in consecutive iterations. The data deviation methods distinguish the fuzzer type. Multiple methods of fuzzing are in active development. They require different input-data preparation techniques (e.g. templates), code instrumentation (e.g. to retrieve code coverage information) and result interpretation (e.g. false positive detection). Currently there is no established limit on the number of different tools to employ and the iterations to run them. This chapter examines one of multiple tools.

2.3.3.1 Tool acquisition

AFL (“American Fuzzy Lop”) is available as Open Source under the Apache License 2.0. Google Inc. owns the copyright. Currently, the source code is downloadable through http://lcamtuf.coredump.cx/afl/, and also packaged in most Linux distributions.

2.3.3.2 Tool characterization

The tool author gives a characterization as follows: “American fuzzy lop is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary. This improves the functional coverage for the fuzzed code. The compact synthesized corpora produced by the tool are also useful for seeding other, more labour- or resource-intensive testing regimes” [44].

2.3.3.3 Properties that can be asserted

- AFL uses input deviation methods by deterministic mutation based on feedback from code-coverage instrumentation of the binary.
- Code instrumentation integrates with (but is not limited to) compilers Clang and GCC.
- There also exists a QEMU-run-mode for binaries without access to source that does the instrumentation through the virtual machine
- AFL stores input corpuses that led to a crash to later also test for regressions or for use in other analysers or as a start point for fuzzers that are more TOE specific.
- For performance reasons, AFL is bound to Linux and BSD systems (quicker process fork).
- The AFL framework also provides integrated analysis tools to spot-light a crash result
- There exist derivatives for kernel system calls [45] and other programming languages than C.

2.3.3.4 Usage experience

There exists some experience by partner SRO.

2.3.3.4.1 Usability/scalability/interoperability

Fuzz-testing is computation intense and requires large resources in terms of RAM and CPU speed. It can be efficiently parallelized, both on multi-core CPUs and on computation clusters.

The output of AFL can be fed into further analysis tools, tracking frameworks and implanted into test frameworks.

2.3.3.5 Use in certification: Common Criteria [20]

Fuzzing is an acceptable technique to be used under the vulnerability analysis scope of Common Criteria from AVA_VAN.2 to AVA_VAN.5.

2.3.3.6 Use in certification: IEC 62443 [21]

AFL is currently not officially recognized as part of a tool for communication robustness testing. AFL is not directly applicable as a fuzz-test tool for network traffic. According to [46], network traffic can be redirected through file-pipes of the target. Concerning ISA-62443-4-1 §10.4.1-2 AFL can provide functionality to run test cases for “abuse case or malformed or unexpected input testing focused on uncovering security issues.”

AFL can focus on existing test vectors to probe publically known vulnerabilities and regressions.

Furthermore §10.4.2 suggests manual preparation of input corpuses or running a different fuzzing tool. Both are typical use cases for AFL, e.g. input initialization files and running in parallel with e.g. LibFuzzer [47].

2.3.3.7 Use in certification: Other Security certification (IEC TS 62531 [42], DIN VDE V 0831-104 [43])

Fuzzing is explicitly mentioned in ISASecure Security Development Lifecycle Assurance (SDLA) [48].

2.4 Tools for documentation and assurance case creation

2.4.1 DOORS

DOORS [49] is a requirement management tool.

2.4.1.1 Tool acquisition

DOORS has to be licensed from IBM. The list price is currently starting at 5300 USD per user. [50]

2.4.1.2 Tool characterization

DOORS comes with a GUI that maintains requirements. From the view of DOORS, each requirement is a short block of text, usually one or a few paragraphs. A document consists of requirements (DOORS calls a document “module”). DOORS adds unique identifiers to requirements, and allows to define arbitrary attributes, baselines and maintains a history of changes. DOORS allows to insert directed links between requirements within the same or different modules. Amongst others, DOORS allows export to/import from Microsoft Office and export to CSV and HTML. DOORS allows to run scripts in a scripting language (DXL). DOORS can maintain links to test cases and source code.

2.4.1.3 Properties that can be asserted

DOORS uses natural language, i.e., any property can be expressed.

2.4.1.4 Usage experience

A high-level property is usually asserted by refinement, i.e. it is linked to lower-level requirements. The consistency of linkage has to be established and reviewed manually.

2.4.1.4.1 Input and its preparation

The input consists of thoughts about what a product shall do and how this functionality shall be implemented. This input is reviewed by testers at an early stage.

2.4.1.4.2 Output and its interpretation

The output is a hierarchy of requirements: high-level requirements, interface requirements, design requirements, and linkage to test cases, analysis documents. This linkage can be used to make an assurance case that the implementation satisfies the requirements, i.e. traceability matrix.

2.4.1.4.3 Usability/scalability/interoperability

The GUI is user-friendly and DXL scripting is used to automate more tedious analyses and operations. DOORS allows concurrent editing of different requirements at the same time, i.e., an editor can choose to lock only small portions of a document. Microsoft Office, ReqIF [51] or other requirement exchange formats can be used.

2.4.1.5 Use in certification: Safety certification (IEC 61508 [17], DO-178 [2], IEC 62290 [18], etc.)

MILS has strong roots in avionics. For the avionics as well as the railway domain, DOORS is mentioned as requirement engineering and configuration management tool [52].

2.4.1.6 Use in certification: Common Criteria [20]

2.4.1.6.1 Base components

In product Security Targets (STs) and certification reports [53] downloaded [54] from the Common Criteria Portal, “DOORS” is mentioned in a certification report of a tachograph [55] where it is used to keep functional requirements and test cases together.

At SYSGO, for a separation kernel, DOORS is used to provide documentation about the functional specification (in CC terms: ADV_FSP), the separation kernel design (ADV_TDS) and linkage to test cases in the form of DOORS modules. Also, the security architecture (ADV_ARC) is kept in DOORS. Moreover, the DOORS modules contain links to a proxy document representing the Security Target.

2.4.1.7 Use in certification: IEC 62443 [21]

Because DOORS and requirements are a comparatively non-formal approach, they might be a good tool to fulfil IEC 62443 Part 4-1, SR-5, security requirements review*: “A process shall be employed to ensure that security requirements are reviewed, updated as necessary and approved to ensure their clarity, validity and their ability to be verified. At least one person from each of the following shall participate in this process:*

- *Developers (those who will implement the requirements);*
- *Testers (those who will validate that the requirements have been met); and*
- *Customer advocate (such as sales, marketing, or customer support).”*

The good traceability that can be achieved using DOORS, can also be used to fulfil [21], Part 4-1, SD-1, “*A process shall be employed for developing and documenting a secure design that identifies and characterizes each exposed interface of the product, including physical and logical interfaces.*”

2.4.1.8 Related tools

Other requirement engineering tools exist, e.g., also from IBM, Jazz. In the Trusted Architecture for Securely Shared Services (TAS^3) research project [56], Graphviz was used to visualize traceability [57]. In the DO-178 guide by Hilderman [52], Synergy and (for small projects) Excel are mentioned as alternatives to DOORS. There is also support from the eclipse development platform for requirements engineering such as the eclipse Requirements Management Framework (RMF) / ProR [58].

2.4.2 Medini analyze

2.4.2.1 Tool acquisition

medini analyze is part of the ANSYS product family.

Most of information about the product, licence policy and distributors is available at the following address: http://www.medini.eu.

2.4.2.2 Tool characterization

medini analyze is an integrated tool which implements efficiently core activities of the functional safety analysis and integrates them with the existing processes. Target users are safety managers and experts as well as development engineers and quality managers involved in the development of electronic and software based components mainly in the automotive industry [59].

2.4.2.3 Properties that can be asserted

Main features of the tool are [60]:

- quality analysis for product design and related processes according to SAE J1739, VDA quality handbook etc.,
- safety analysis and design according to ISO 26262 for software controlled safety related functions,
- integration of architectural/functional design with quality, reliability and functional safety analysis methods,
- support of driving situation analysis, hazard and risk analysis, Fault Tree Analysis (FTA), Failure Mode and Effects Analysis (FMEA), probabilistic analysis and hardware failure metrics,
- complete end-to-end traceability, e.g. of
    - requirements to the design or other object which fulfil them (UML/SysML blocks and connections),
    - modelled objects to their implementation (UML/SysML blocks, connections and external links),
    - requirements and modelled objects to the verification method that verifies them and links to the internal and external evidence (requirements, UML/SysML blocks, connections, and external links).
- customizable work product/documentation generation,
- teamwork with detailed compare and merge,
- integration with IBM® Rational® DOORS, IBM® Rational® Rhapsody, Enterprise Architect, MATLAB®/Simulink®, Stateflow®, PTC Integrity, MS® Office, TortoiseSVN, IBM® Rational® ClearCase, IBM® Rational® Team Concert and others.

2.4.2.4 Usage experience

UCO has become familiar with the tool within the SESAMO project (Security and Safety Modelling). *Medini analyze* was used for system description and analysis of system requirements, it also allowed performing FMEA and FTA analysis. [61]

Although the tool is following mainly needs of the automotive domain (according to ISO 26262), it is useful tool for initial (i.e., development) phases of the railway lifecycle and others using customization functions allowed by UML/SysML based meta-model structure, i.e., profiling. The profiling allows to create additional properties (such as safety and security aspects) for each model component (system components, requirements, etc.). Specific UML connectors serve to create interdependencies among the model items at different levels, which are used for ensuring traceability. Besides the very useful functions of the tool described above, the tool has implemented OCL (Object Constraint Language) and JavaScript engines. This provides a versatile

tool with many possibilities for customization and adaptation to various domains. It also provides transformation of created models into other formats for external tools (using certain transformation rules).

UCO has created a railway profile for the tool that contains among others the following properties and functions (the profile is a product of UCO, intended for internal business needs):

- project structure according to railway system life cycle,
- checklists for verification during development phases,
- requirements of railway standards
  EN50126, EN50128, EN50129, EN 50159, EN 62290,
- libraries of UCO products and products' specific features (e.g. partitioning),
- profiling; customization of model block properties to railway terminology (possibility to assign a safety integrity level (SIL); risk table; railway specific stakeholders, influencing factors; specific operation conditions – stations, tunnel, track, depots, etc.)
- JavaScript library to export a system model with all components and interdependencies into a collection of web pages for to operation and maintenance purposes (e.g. maintenance manual).

The following three figures show three screenshot of the tool customization. The first is an example of Hazard analysis and Risk assessment (HARA) according to the railway domain.

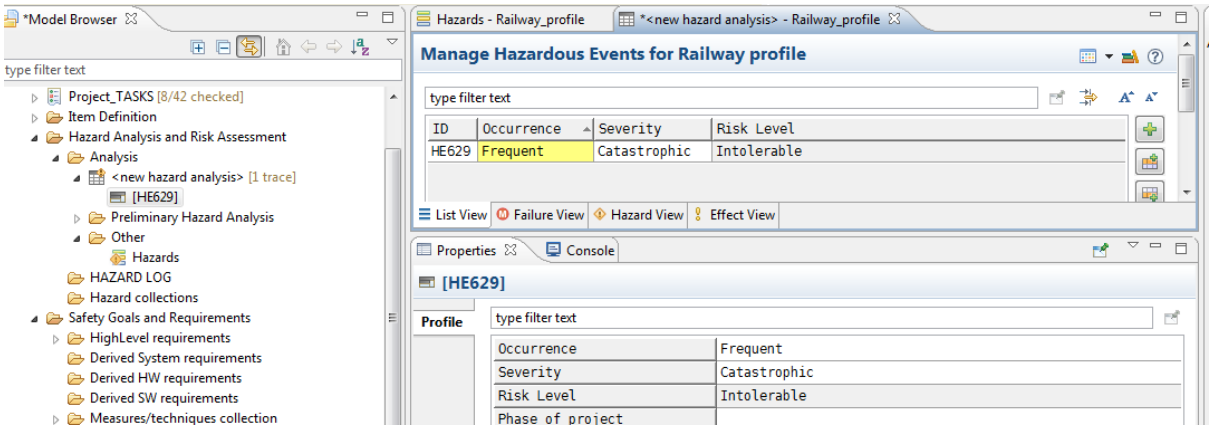

Figure 5: medini analyze – HARA in the railway profile.

Figure 6 captures a set of selected railway standards' requirements.

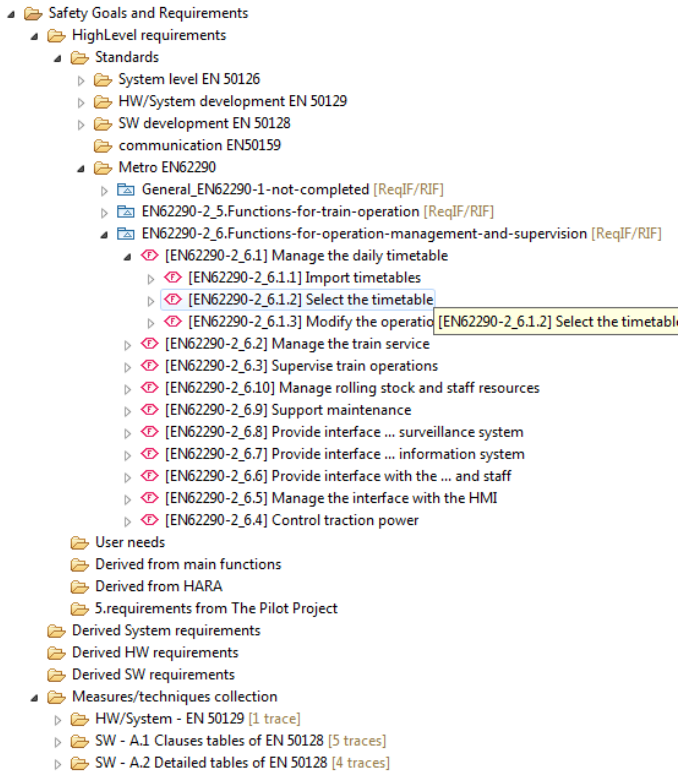

Figure 6: medini analyze – requirements of EN 62290.

Figure 7 is an example of a checklist on the railway system life cycle phases. The figure also visualizes the traceability (the “trace” mark with the number of connected items), where each verification question is traced to an appropriate resource in the project (it could be an external resource, an analysis, a model, standard’s requirement that produces the question etc.).

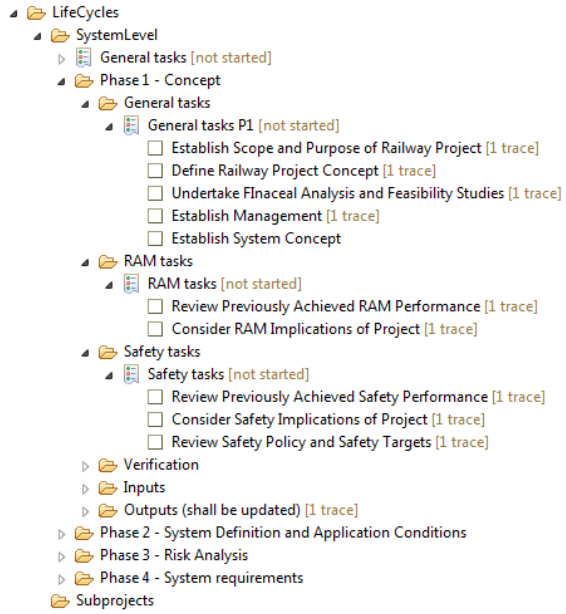

Figure 7: medini analyze – verification checklist of railway development phases.

UCO is going to use the tool also in the certMILS project for the demonstrator development.

2.4.2.5 Use in certification

Within any certification process, the tool and its outputs (in any form, such as analysis reports, filled checklists, structured requirements and their satisfactions, system models) are helpful for providing evidence of quality and safety assurance. The tool specifically provides evidence based

on requirement traceability, verification using checklists and references to the appropriate parts of the model or other external resources – many design aspects needed both in safety and security.

In the case of the certMILS project, UCO is going to enhance the railway profile with requirements of security standards. It will then be analysed and decided about the right set of requirements for the intended use case. Then the profile will be applied for system modelling, architecture, design description, requirement management, part of verification and traceability to provide the best evidences for the certification.

2.4.2.6 Related tools

The ANSYS medini Technologies AG guarantees integration with:

- IBM® Rational® DOORS,
- IBM® Rational® Rhapsody,
- Enterprise Architect,
- MATLAB®/Simulink®,
- Stateflow®,
- PTC Integrity,
- MS® Office,
- TortoiseSVN,
- IBM® Rational® ClearCase,
- IBM® Rational® Team Concert.

The integration depends on the selected license of the tool. UCO has implemented integration with TortoiseSVN; MS Office; Redmine server for management of development tasks; and it features model transformation to/from Enterprise Architect.

Beside these integration possibilities, there is the option to create own transformation rules using e.g. JavaScript to transform the model to or from other tools.

Chapter 3 Discussion

3.1 Security properties / assurance that can be asserted

Some tools directly show a separation property, like Astrée taint analysis or Isabelle/HOL modelling. Other tools target general robustness, such as Achilles, OpenVAS, and fuzzing where robustness is defined that certain attacks are mitigated/denied and that the system is still operational after these attacks. A third category targets to ease development processes such as DOORS or medini.

3.2 Efforts of tool use

Some tools can be used out of the box on existing source code like Astrée or pmccabe and ensure conformance to syntactic (Astrée RuleChecker), or structural (pmccabe showing number of branches within each function). Achilles and OpenVAS operate in a networked environment with deployed components, and investigate the vulnerability of components by executing known attacks and weaknesses.

Customized methods are more labour-intensive, e.g. fuzzing is a means of showing non-compliance of an implementation or non-robustness whereas modelling by Isabelle/HOL had been used to show separation. Also, the use of Astrée taint analysis needs customization to match the separation kernel under test.

3.3 Compositional aspects

None of the tools specifically addressed compositional certification. However, by formalizing one specific certification layer, the higher-level argument can be made in an easier way. Moreover, we realize that at the specification level, the scope of medini analyze is quite broad, so that ideas and workflow taken from medini could be used for further thinking about how to formulate compositional aspects of certification.

3.4 Tool scope for standards

Tool	Safety	CC	IEC 62443	Other security
taint	n	y	n	p
complexity	y	p	n	p
Isabelle/HOL	y	y	n	p
OpenVAS	n	y	y	y
Achilles	n	y	y	y
fuzzing	n	y	p	y
DOORS	y	y	y	y
medini	y	y	y	y

Table 1: Reported tool scope. “y” means applicable, “p” means potentially applicable, and “n” means “not commonly used”

Table 1 summarizes the reported tool scope for standards. We can see that documentation tools are universally usable, whereas some more specialized tools like OpenVAS or Achilles are more

specifically tailored to security. Most investigated standards are cautious on complexity metrics. The CC does not explicitly demand fuzzing or network penetration testing, but these are acceptable techniques to be used under the vulnerability analysis scope of the Common Criteria. On the other hand, IEC 62443 does not have much focus on low-level methods (such as complexity or formal methods). Overall, the results show that the certification of MILS base components may benefit from a different tool set than tools used for networked MILS system.

3.5 Check against tools listed in security tool registries

While this was not in the original scope of the planned deliverable, for additional verification, we checked some online tool registries for inspiration. These mostly address static code analysis and network attacks. E.g., we could find OpenVAS and Achilles to be recommended by ISASecure. However, the intersection of the online registries of ISASecure, CWE-compatible and SCAP is completely empty as of June 2017, which was a non-anticipated result. For more details, see the appendix “Comparison with other tool lists” (Chapter 5).

3.6 Further research / interaction between project partners

A first result of creating this deliverable is that partners understand better each other’s experience. For example, we plan to present the medini tool within the consortium. OpenVAS is also used by THA. Fuzz-testing and the Achilles Platform is of high interest to THA to learn and integrate the approach.

For code metrics, we have learned that the application of code metrics has to be closely monitored in order not to create unwanted distortions, and that most standards are also cautious about this. Something to keep an eye on in the future, and not yet covered in this deliverable, are metrics for measuring code coupling and cohesion.

Beyond directly learning about new tools, one result of this tool comparison is that we get a better grasp of the breadth of what state-of-the-art tools can do and what is still beyond the state of the art. From experience, even when a tool output is not directly used in certification, adoption of any tool allows to verify an informal working hypothesis, and by requiring machine-readable input, per se enforces semi-formal architectural thinking. For instance, when Isabelle/HOL was used for a formal model of the separation kernel, even if that model was not directly used for an EAL 6 certification (which, for operating systems, would be very ambitious), it gave us a better feeling on how to structure an argument on separation, e.g., the different assets to be considered. Discussions on medini analyze also motivate us to keep UML/SysML-formalizability in mind when working on future MILS guidance in the certMILS work package WP2 context.

It is important to avoid raising too high expectations: when we plan to further outline our strategy, specifically for compositional systems in D1.3 (“Compositional security certification methodology”), it is not expected that each tool discussed here will be reflected in D1.3. However, this deliverable D1.2 has clarified what security properties exist and offered a better understanding what the effort is to ascertain them. Tools targeting to ease development processes also will be discussed the scope of the certMILS Protection Profile (PP) standardization reach-out in D2.4 (“Guidelines to use and apply PP for all involved stakeholders”).

Chapter 4 List of abbreviations

Abbreviation	Translation
ACC	Achilles Communications Certification
ACS	Authenticated Configuration Scanner
AFL	American Fuzzy Lop
APC	Achilles Practices Certification
API	Application Programming Interface
ARP	Address Resolution Protocol
BSD	Berkeley Software Distribution
CAN	Controller Area Network
CC	Common Criteria
CERT	Computer Emergency Response Team
CIFS	Common Internet File System
CPU	Central Processing Unit
CRT	Communication Robustness Testing
CSV	Comma-separated values
CVSS	Common Vulnerability Scoring System
CWE	Common Weaknesses Enumeration
DM	Security Defect Management
DMZ	Demilitarized Zone
DNP	Distributed Network Protocol
DOORS	Dynamic Object Oriented Requirements System
DXL	DOORS eXtension Language
EAL	Evaluation Assuance Level
EDSA	Embedded Device Security Assurance
FMEA	Failure Mode and Effects Analysis
FP7	7th Framework Programme
FTA	Fault Tree Analysis
GCC	GNU C Compiler
GE	General Electrics
GUI	Graphical User Interface
GNU	GNU's not Unix
GPL	General Public License
HARA	Hazard Assessment and Risk Analysis
HMI	Human Machine Interface
HOL	Higher-Order Logic
HTML	Hypertext Markup Language
IBM	International Bureau Machines
ICMP	Internet Control Message Protocol
ICS	Industrial Control System
IEC	International Electrotechnical Commission
IKE	Internet Key Exchange
IP	Internet Protocol

Abbreviation	Translation
IPC	Inter-process Communication
ISA	International Society of Automation
ISO	International Organization for Standardization
LDAP	Lightweight Directory Access Protocol
MCISK	Multicore Controlled Interruptible Separation Kernel
MILS	Multiple Independent Levels of Security and Safety
MISRA	Motor Industry Software Reliability Association
NVT	Network Vulnerability Tests
OCIL	Open Checklist Interactive Language
OCL	Object Constraint Language
OpenVAS	Open Vulnerability Assessment System
PDF	Portable Document Format
PM	Security update management
PP	Protection Profile
QEMU	Quick Emulator
RAM	Random Access Memory
RMF	Requirements Management Framework
RTU	Remote Terminal Unit
SAE	Society of Automotive Engineers
SCADA	Supervisory Control and Data Acquisition
SCAP	Security Content Automation Protocol
SD	Secure by Design
SDLA	Security Development Lifecycle Assurance
SEI	Software Engineering Institute
SESAMO	Security and Safety Modelling
SG	Security Guidelines
SI	Secure Implementation
SIL	Safety Integrity Level
SM	Security Management
SMB	Server Message Block
SR	Specification of Security Requirements
ST	Security Target
SV	Secure Verification and Validation Testing
TAS^3	Trusted Architecture for Securely Shared Services
TCP	Transport Control Protocol
UDP	Universal Datagram Protocol
UML	Universal Modeling Language
USD	US Dollars
VDA	Verband der Automobilindustrie
VITT	Vulnerability Identification Testing Tool
VPN	Virtual Private Network
XML	Extensible Markup Language

Chapter 5 Appendix: Comparison with other tool lists

We will take a quick look at different types of assurance based on stages of the life cycle, as well as how available tools fit into those stages.

5.1 Types of Assurance

Assurance can and needs to be obtained at every stage of the life-cycle, as illustrated for example by Microsoft's well-known security development life-cycle [62]. Table 2 shows the life cycle outlined by IEC 62443-4-1 as of the March 2016 draft [38] as it is a standard relevant for the certMILS industrial control domains.

Practice	Name	Acronym
Practice 1	Security Management	SM
Practice 2	Specification of Security Requirements	SR
Practice 3	Secure by Design	SD
Practice 4	Secure Implementation	SI
Practice 5	Secure verification and validation testing	SV
Practice 6	Security defect management	DM
Practice 7	Security update management	PM
Practice 8	Security guidelines	SG

Table 2: Security Development Life Cycle outlined in IEC 62443-4-1

In 62443-4-1, these life cycle practices are further broken down into elements as shown in Table 3.

Number	Name
Practice 1	Security management
SM-1	Development process
SM-2	Identification of responsibilities
SM-3	Identification of applicability
SM-4	Security expertise
SM-5	Process tailoring
SM-6	Code signing
SM-7	Development environment security
SM-8	Third-party embedded component security
SM-9	Special purpose third-party components
SM-10	Addressing of security-related issues
SM-11	Process verification
Practice 2	Specification of security requirements
SR-1	Product security context

Number	Name
SR-2	Threat model
SR-3	Product security requirements
SR-4	Product security requirements content
SR-5	Security requirements review
Practice 3	Secure by design
SD-1	Secure design principles
SD-2	Defense in depth design
SD-3	Security design review
SD-4	Assessing security-related issues
SD-5	Addressing security-related issues
SD-6	Secure design industry recommended practices
Practice 4	Secure implementation
SI-1	Security implementation review
SI-2	Assessing security-related implementation issues
SI-3	Addressing security-related issues
SI-4	Secure implementation recommended practices
Practice 5	Security verification and validation testing
SV-1	Security requirements testing
SV-2	Threat mitigation testing
SV-3	Vulnerability testing
SV-4	Penetration testing
SV-5	Independence of testers
Practice 6	Security defect management
DM-1	Receiving notifications of security-related issues
DM-2	Reviewing security-related issues
DM-3	Assessing security-related issues
DM-4	Addressing security-related issues
DM-5	Disclosing security-related issues
DM-6	Periodic review of security defect management practice
Practice 7	Security update management
PM-1	Security update qualification
PM-2	Security update documentation
PM-3	Dependent component or operating system security update
PM-4	Security update delivery
PM-5	Timely delivery of security patches
Practice 8	Security guidelines
SG-1	Product defense in depth
SG-2	Defense in depth measures expected in the environment
SG-3	Security hardening guidelines
SG-4	Secure disposal guidelines
SG-5	Secure operation guidelines
SG-6	Account management guidelines
SG-7	Documentation review

Table 3: Security Development Life Cycle elements in IEC 62443-4-1

As example, and because SV is a dominant category in Table 5 below, it is also worth to present the IEC 62443-4-1 descriptions in the field of verification (SV), where the following four types of verification given in Table 4 are distinguished.

Name	Description
SV-1: Security requirements testing	This testing focuses on verifying all the security requirements in the security requirements specification (SecRS) have been met. Functional, negative, boundary, performance and other types of standard testing will be performed on the security capabilities in the SecRS.
SV-2: Threat mitigation testing	This testing is derived from creating threat trees from the threats identified in the threat model and ensures that the mitigations designed and implemented in the product are effective in stopping the proposed threat. Testers will design their tests to attempt to thwart the mitigation using the type of threat identified.
SV-3: General vulnerability testing	This testing focuses on using standard tools or published instructions for discovering potential security vulnerabilities. No attempt is made to exploit the vulnerability or assess the ability to exploit the potential vulnerability and the product is tested without consideration to the implementation or its defense in depth design.
SV-4: Penetration testing	This testing focuses specifically on compromising the confidentiality, integrity or availability of the product. It can involve defeating multiple aspects of the defense in depth design. This is an unstructured test that depends on the skills and knowledge of the attacker. In this case, the tester tries to play the role of an attacker. This testing is not based on an analysis of the design or threat model, rather it encompasses the tester trying to defeat the security of the system using any technique that he chooses. This testing often will identify types of vulnerabilities that need to be fixed rather than single vulnerabilities. This testing will often detect problems that are not detected in threat model driven testing because there may be errors or omissions in the threat model itself.

Table 4: IEC 62443-4-1 testing approaches, from IEC 62443-4-1, Section 10.1

5.2 Other security tools

The results of a search of several websites and partner knowledge for security tools is shown in Table 5. We tentatively have assigned each tool to a practice of IEC 62443-4-1. This analysis showed that the bulk of tools marked for security are heading towards secure implementation (SI) and secure verification and validation testing (SV).

ISA-62243-4-1 practice elements	Company	Tool	IsaSecure VITT [63]	IsaSecure CRT [64]	CWE-compatible [65]	SCAP ACS [66]	SCAP CVE [66]	SCAP OCIL [66]	Covered in this deliverable
SD-1	ANSYS	medini							Section 2.4.2
SD-1, SR-5	IBM	DOORS							Section 2.4.1
SD-2	Cambridge/Mu	Isabelle/HOL							Se

ISA-62243-4-1 practice elements	Company	Tool	IsaSecure VITT [63]	IsaSecure CRT [64]	CWE-compatible [65]	SCAP ACS [66]	SCAP CVE [66]	SCAP OCIL [66]	Covered in this deliverable
	nich/Paris								ction 2.2.1
SI-1	AbsInt	Astrée							Section 2.1.1
SI-1	AdaCore	CodePeer			y				
SI-1	Bame	PMCCABE							Section 2.1.2
SI-1	Beijing Beida Software Engineering Development Co., Ltd.	COBOT			y				
SI-1	CAST	CAST Application Intelligence Platform			y				
SI-1	Conviso Application Security	Conviso Security Compliance (CSC)			y				
SI-1	Coverity, Inc.	Coverity Quality Advisor			y				
SI-1	Coverity, Inc.	Coverity Security Advisor			y				
SI-1	David A. Wheeler	Flawfinder			y				
SI-1	Denim Group, Ltd	ThreadFix			y				
SI-1	Evenstar	BigLook			y				
SI-1	Fasoo.com, Inc.	SPARROW			y				
SI-1	GrammaTech, Inc.	CodeSonar			y				
SI-1	GTONE Co., Ltd.	SecurityPrism			y				
SI-1	Hewlett-Packard Development Company, L.P.	HP Fortify On Demand			y				
SI-1	Hewlett-Packard Development Company, L.P.	HP Fortify Software Security Center			y				

ISA-62243-4-1 practice elements	Company	Tool	IsaSecure VITT [63]	IsaSecure CRT [64]	CWE-compatible [65]	SCAP ACS [66]	SCAP CVE [66]	SCAP OCIL [66]	Covered in this deliverable
SI-1	Hewlett-Packard Development Company, L.P.	HP Fortify Static Code Analyzer			y				
SI-1	IBM Security Systems	IBM Security AppScan Standard			y				
SI-1	Julia S.R.L.	Julia			y				
SI-1	Klocwork, Inc.	Klocwork Insight			y				
SI-1	LDRA	LDRA Testbed			y				
SI-1	LDRA	LDRArules			y				
SI-1	LDRA	Tbvision			y				
SI-1	Lucent Sky Corporation	Lucent Sky Application Vulnerability Mitigation (AVM)			y				
SI-1	MathWorks, Inc.	Polyspace Bug Finder			y				
SI-1	Programming Research, Inc.	QA*C – CWE Compliance Module for C Programming Language			y				
SI-1	Soft4Soft Co., Ltd.	RESORT Code Analysis			y				
SI-1	SonarSource SA	SonarQube platform with C/C++ plugin			y				
SI-1	SonarSource SA	SonarQube platform with Java plugin			y				
SI-1	SonarSource SA	SonarQube platform with Objective-C plugin			y				
SI-1	Suresoft Technologies Inc.	CodeScroll Code Inspector			y				
SI-1	Suresoft Technologies Inc.	CodeScroll SNIPER			y				
SI-1	Veracode, Inc.	Veracode Dynamic Analysis			y				
SI-1	Veracode, Inc.	Veracode Static Analysis			y				
SV-1	Veracode, Inc.	Veracode Manual Testing			y				
SV-3	Beyond Security	beSTORM		y					
SV-3	bmc	Client Management 12.0.0				y	y		
SV-3	bmc	Server Automation 8.6				y	y	y	
SV-3	CIS Center for Internet Security	CIS-CAT Pro Assessor				y	y		
SV-3	CNCERT	Acheron		y					
SV-3	Hewlett-Packard	HP Assessment Management Platform (ASP)			y				

ISA-62243-4-1 practice elements	Company	Tool	IsaSecure VITT [63]	IsaSecure CRT [64]	CWE-compatible [65]	SCAP ACS [66]	SCAP CVE [66]	SCAP OCIL [66]	Covered in this deliverable
	Development Company, L.P.								
SV-3	Hewlett-Packard Development Company, L.P.	HP Fortify Real-Time Analyzer			y				
SV-3	Hewlett-Packard Development Company, L.P.	HP WebInspect			y				
SV-3	High-Tech Bridge SA	ImmuniWeb			y				
SV-3	Hitachi	Raven ES		y					
SV-3	IBM	Endpoint Manager 9				y			
SV-3	IBM	BigFix				y	y		
SV-3	intel	Policy Auditor 6.2				y	y		
SV-3	Microsoft	SCAP Extensions for Microsoft System Center Configuration Manager				y	y		
SV-3	Qualys	SCAP Auditor 1.2				y	y		
SV-3	Rapid	Nexpose 6				y	y		
SV-3	Red Hat, Inc.	openSCAP 1				y	y		
SV-3	Red Hat, Inc.	openSCAP 1.0				y	y		
SV-3	SAINT	SAINT Security Suite 8				y	y		
SV-3	Security-Database	Security-Database Web Services			y				
SV-3	SPAWAR	SCAP Compliance Checker				y	Y	y	
SV-3	Synopsis	Defensics X		y					
SV-3	tenable	nessus	y						Section 2.3.1(OpenVAS)
SV-3	tenable	SecurityCenter				y	y		
SV-3	ThreatGuard	Secutor Prime (S-CAT) 5				y	y		
SV-3	ThreatGuard	Secutor Prime 5				y	y	y	
SV-3	tripwire	Enterprise 8				y	y		
SV-3	Veracode, Inc.	Veracode Analytics			y				
SV-3	Wurldtech	Achilles		y					Section

ISA-62243-4-1 practice elements	Company	Tool	IsaSecure VITT [63]	IsaSecure CRT [64]	CWE-compatible [65]	SCAP ACS [66]	SCAP CVE [66]	SCAP OCIL [66]	Covered in this deliverable
									2.3.2
x (N.A.)	Cr0security	Cr0security Certified Security Testing			y				
x (N.A.)	Cr0security	Cr0security Penetration Testing and Consultant Services			y				
x (N.A.)	CXSecurity	World Laboratory of Bugtraq (WLB) 2			y				
x (N.A.)	High-Tech Bridge SA	High-Tech Bridge Security Advisories			y				
x (N.A.)	National Institute of Standards and Technology (NIST)	Software Assurance Reference Dataset (SARD)			y				
x (N.A.)	Red Hat, Inc.	Red Hat Customer Portal			y				
x (N.A.)	ToolsWatch	vFeed API and Vulnerability Database Community			y				
x (N.A.)	WebLayers, Inc.	WebLayers Center Security Policy Library			y				

Table 5: Tools collected from partner experience and security websites

5.3 Results

Interestingly, in Table 5, the intersection of the online registries of ISASecure, CWE-compatible and SCAP is completely empty as of June 2017, which was a non-anticipated result.

If we apply this classification, then most commercially available generic tools from Table 5 fall into category SV-3, as they are not customized for a particular product. Developing a customized fuzzing strategy would probably fall into category SV-4, giving highest credit. While this deliverable simply collects current state of the art, in particular, the approach of developing a customized fuzzing and robustness testing strategy will be followed in certMILS deliverables D4.1 and D4.4.

Chapter 6 Appendix: Used Template

This is the template that had been used for Chapter 2.

6.1.1 Tool name

6.1.1.1 Tool acquisition

<Describe where to download or buy the tool.>

6.1.1.2 Tool characterization

<Describe what the tool does (one paragraph).>

6.1.1.3 Properties that can be asserted

<Describe what properties of a system can be asserted.>

6.1.1.4 Usage experience

<Describe your usage experience with the tool. Usage can be both industrial and R&D contexts. If you wish you can split it along base components and MILS systems, like in the structure below. (I.e., it is optional to have entries for both. You also can opt for a simpler structure that does not split between base components and MILS systems.)>

6.1.1.4.1 Base components

6.1.1.4.1.1 Input and its preparation

6.1.1.4.1.2 Output and its interpretation

6.1.1.4.2 MILS systems

6.1.1.4.2.1 Input and its preparation

6.1.1.4.2.2 Output and its interpretation

6.1.1.4.3 Usability/scalability/interoperability

6.1.1.5 Use in certification: Safety Certification (IEC 61508 [17], DO-178 [2], IEC 62290 [18], etc.)

<If possible, describe your knowledge and/or expectations for where the tool gives credit for safety certification. This section is optional.>

6.1.1.5.1 Base components

6.1.1.5.2 Compositional certification: MILS systems

6.1.1.6 Use in certification: Common Criteria [20]

<If possible, describe your knowledge and/or expectations for where the tool gives credit for Common Criteria. This section is optional.>

6.1.1.6.1 Base components

6.1.1.6.2 Compositional certification: MILS systems

6.1.1.7 Use in certification: IEC 62443 [21]

<If possible, describe your knowledge and/or expectations for where the tool gives credit for IEC 62443. Again, this section is optional.>

6.1.1.7.1 Base components

6.1.1.7.2 Compositional certification: MILS system

6.1.1.8 Use in certification: Other Security Certification (such as e.g. IEC TS 62531 [42], DIN VDE V 0831-104 [43], etc.)

6.1.1.8.1 Base components

6.1.1.8.2 Compositional certification: MILS systems

<If possible, describe your knowledge and/or expectations for where the tool gives credit for other, e.g. domain specific security certification. This section is optional.>

6.1.1.9 Related tools

<Mention related tools>.

Chapter 7 References

[1] AbsInt GmbH, “Fast and sound runtime error analysis,” 2017. [Online]. Available: https://www.absint.com/astree/index.htm.

[2] RTCA SC-205 / EUROCAE WG-71, DO-178C: Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics (RTCA), Inc., 1150 18th NW, Suite 910, Washington, D.C. 20036, 2011.

[3] P. Bame, “PMCCABE Overview,” 2017. [Online]. Available: https://people.debian.org/~bame/pmccabe/overview.html.

[4] T. J. McCabe, “A complexity measure,” *IEEE Transactions on software Engineering,* no. 4, pp. 308-320, 1976.

[5] N. Saini, S. Kharwar and A. Agrawal, “A study of significant software metrics,” 2014. [Online]. Available: http://www.ijeijournal.com/papers/Vol.3-Iss.12/A030120107.pdf.

[6] J. Delange, J. J. Hudak, W. R. Nichols, J. McHale and M.-Y. Nam, “Evaluating and Mitigating the Impact of Complexity in Software Models,” 2015. [Online].

[7] S. Burger and O. Hummel, “Applying Maintainability Oriented Software Metrics to Cabin Software of a Commercial Airliner,” mar 2012. [Online]. Available: https://doi.org/10.1109/CSMR.2012.58.

[8] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” *IEEE Transactions on software engineering,* vol. 25, no. 5, pp. 675-689, 1999.

[9] M. Shepperd, “A critique of cyclomatic complexity as a software metric,” *Software Engineering Journal,* vol. 3, no. 2, pp. 30-36, 1988.

[10] A. Muslija, “On the Complexity Measurement of Industrial Control Software,” 2017. [Online]. Available: http://www.idt.mdh.se/utbildning/exjobb/files/TR1984.pdf.

[11] A. Jbara, “High-MCC Functions in the Linux Kernel,” 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6240512/.

[12] A. Jbara and D. G. Feitelson, “How programmers read regular code: a controlled experiment using eye tracking,” jun 2017. [Online]. Available: http://link.springer.com/10.1007/s10664-016-9477-x.

[13] Y. Gil and G. Lalouche, “On the correlation between size and metric validity,” oct 2017. [Online]. Available: http://link.springer.com/10.1007/s10664-017-9513-5.

[14] T. J. McCabe and C. W. Butler, “Design Complexity Measurement and Testing,” dec 1989. [Online]. Available: http://doi.acm.org/10.1145/76380.76382.

[15] D. J. Sturtevant, “System design and the cost of architectural complexity,” 2013. [Online].

[16] U. Tiwari and S. Kumar, “Cyclomatic complexity metric for component based software,” feb 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2557833.2557853.

[17] International Electrotechnical Commission, Technical Committee 65: Industrial-process measurement and control, Subcommittee 65A: System aspects, “IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems,” 2010. [Online]. Available: http://www.iec.ch/.

[18] International Electrotechnical Commission, “IEC 62290: Railway applications - Urban guided transport management and command/control systems - Part 1: System principles and fundamental concepts,” 2014. [Online].

[19] S. Kan, Metrics and Models in Software Quality Engineering, Addison Wesley, 2003.

[20] Common Criteria Sponsoring Organizations, “Common Criteria for Information Technology Security Evaluation. Version 3.1, revision 4,” September 2012. [Online]. Available: http://www.commoncriteriaportal.org/cc/.

[21] International Electrotechnical Commission, Technical Committee 65: Industrial-process measurement and control, “IEC 62443: Security for industrial automation and control systems,” 2017. [Online]. Available: http://www.isa99.org/.

[22] L. Paulson, T. Nipkow and M. Wenzel, “Isabelle,” 2017. [Online]. Available: http://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[23] F. Wiedijk, “Is ZF a hack? Comparing the complexity of some (formalist interpretations of) foundational systems for mathematics,” 2006. [Online]. Available: http://www.cs.ru.nl/F.Wiedijk/zfc-etc/zfc-etc.pdf.

[24] H. Blasum, O. Havle, S. Tverdyshev, B. Langenstein, W. Stephan, A. Feliachi, Y. Nemouchi, B. Wolff, C. Proch, F. Verbeek and J. Schmaltz, “Used Formal Methods,” 13 Oct 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.47297.

[25] Y. Zhao, D. Sanan, F. Zhang and Y. Liu, “High-Assurance Separation Kernels: A Survey on Formal Methods,” 2017. [Online]. Available: https://arxiv.org/abs/1701.01535.

[26] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood, “seL4: Formal Verification of an OS Kernel,” Jun 2010. [Online]. Available: http://ertos.nicta.com.au/publications/papers/Klein_EHACDEEKNSTW_10.pdf.

[27] J. Rushby, “Design and verification of secure systems,” 1981. [Online]. Available: http://www.sdl.sri.com/papers/sosp81/sosp81.pdf.

[28] F. Verbeek, O. Havle, J. Schmaltz, S. Tverdyshev, H. Blasum, B. Langenstein, W. Stephan, B. Wolff and Y. Nemouchi, “Formal API Specification of the PikeOS Separation Kernel,” 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17524-9_26.

[29] F. Verbeek, J. Schmaltz, O. Havle, B. Wolff and B. Langenstein, “MCISK,” 30 Mar 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.48658.

[30] F. Verbeek, S. Tverdyshev, O. Havle, H. Blasum, B. Langenstein, W. Stephan, Y. Nemouchi, A. Feliachi, B. Wolff and J. Schmaltz, “Formal Specification of a Generic Separation Kernel,” 2014. [Online]. Available: http://afp.sourceforge.net/entries/CISC-Kernel.shtml.

[31] J. Schmaltz, H. Blasum, B. Langenstein, B. Leconte, K. Müller, F. Verbeek and R. Koolen, “Formal Framework for MILS Integration,” 03 Feb 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.57413.

[32] BAE SYSTEMS DATAGATE (formerly Tenix Datagate), “Tenix Interactive Link Data Diode Device Version 2.1,” 2005. [Online]. Available: http://www.commoncriteriaportal.org:80/files/epfiles/st_vid9512-st.pdf.

[33] BAE SYSTEMS DATAGATE (formerly Tenix Datagate), “Tenix Interactive Link Data Diode Device, Gigabit Variant, Version 3.0,” 2006. [Online]. Available: http://www.commoncriteriaportal.org:80/files/epfiles/st_vid9513-st.pdf.

[34] BAE SYSTEMS DATAGATE (formerly Tenix Datagate), “Tenix Interactive Link Version 5.1,” 2005. [Online]. Available: http://www.commoncriteriaportal.org:80/files/epfiles/st_vid1021-st.pdf.

[35] Gemalto, “Java Card Virtual Machine of LinqUs USIM 128k platform on SC33F640E,” 2012. [Online]. Available: http://www.commoncriteriaportal.org:80/files/epfiles/ANSSI-CC-cible_2012-18en.pdf.

[36] EURO-MILS, “Guide to EURO-MILS results,” 31 Aug 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.61251.

[37] NLNCSA, “Composite product evaluation for Smart Cards and similar devices,” 2012. [Online]. Available: https://www.commoncriteriaportal.org/files/supdocs/CCDB-2012-04-001.pdf.

[38] International Society of Automation, Working Group 04, Task Group 06, “IEC 62443: Security for industrial automation and control systems: Part 4-1: Secure product development life-cycle requirements,” March 2016. [Online]. Available: http://isa99.isa.org/Public/Series/Documents/ISA-62443-4-1-Public.pdf.

[39] S. Nordhoff and H. Blasum, “Ease Standard Compliance by Technical Means via MILS,” 2017. [Online]. Available: https://doi.org/10.5281/zenodo.571176.

[40] F. Wiedijk and D. Scott, “The Seventeen Provers of the World,” 2005. [Online]. Available: http://www.cs.ru.nl/~freek/comparison/comparison.pdf.

[41] Schneider Electric, “SCADALAB (2012 – 2014),” 2017. [Online]. Available: http://www.schneider-electric.com/b2b/en/products/medium-voltage-switchgear-and-energy-automation/r-and-d-projects/scadalab.jsp.

[42] International Electrotechnical Commission, Technical Committee 57, “IEC TS 62351 Power systems management and associated information exchange - Data and communications security,” 2007. [Online]. Available: http://www.iec.ch/.

[43] DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE, UK 351.3 Bahn-Signalanlagen, “DIN VDE V 0831-104: Elektronische Bahn-Signalanlagen -

Teil 104: Leitfaden für die IT-Sicherheit auf Grundlage IEC 62443," Oct 2015. [Online].

[44] M. Zalewski, "american fuzzy lop (2.52b)," 2017. [Online]. Available: http://lcamtuf.coredump.cx/afl/.

[45] V. Nossum and Q. Casasnovas, "Filesystem Fuzzing with Americal Fuzzy Lop," 2018. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/slides/AFL%20filesystem%20fuzzing%2C%20Vault%202016_0.pdf.

[46] Lolware, "Fuzzing nginx - Hunting vulnerabilities with afl-fuzz," 2015. [Online]. Available: https://lolware.net/2015/04/28/nginx-fuzzing.html .

[47] LLVM, "libFuzzer – a library for coverage-guided fuzz testing," [Online]. Available: http://llvm.org/docs/LibFuzzer.html#afl-compatibility.

[48] ISA Security Compliance Institute, "SDLA-312 Security Development Lifecycle Assessment Version 3.0," 2014. [Online]. Available: http://www.isasecure.org/en-US/Certification/IEC-62443-SDLA-Certification.

[49] IBM, "Rational DOORS," 2017. [Online]. Available: http://www-03.ibm.com/software/products/en/ratidoor/.

[50] IBM, "Rational DOORS: Purchase," 2017. [Online]. Available: https://www.ibm.com/us-en/marketplace/requirements-management/purchase.

[51] Object Management Group, "Requirements Interchange Format," 2016. [Online]. Available: http://www.omg.org/spec/ReqIF/1.2/PDF/.

[52] V. Hilderman and T. Baghi, Avionics certification: A Complete Guide to DO-178 (Software) DO-254 (Hardware), Leesburg, VA: Avionics Communications Inc., 2007.

[53] Common Criteria Sponsoring Organizations, "Certified Products," 2017. [Online]. Available: http://www.commoncriteriaportal.org/products/.

[54] H. Blasum, "ccportaldump," 2017. [Online]. Available: https://github.com/hblasum/ccportaldump.

[55] TNO Certification, "NSCIB-CC-09-11192 Certification Report STARCOS 3.4 ID Tachograph C2," 2010. [Online]. Available: http://www.commoncriteriaportal.org/files/epfiles/Certification_Report_NSCIB-CC-09-11192-CR2.pdf.

[56] Trusted Architecture for Securely Shared Services, "Trusted Architecture for Securely Shared Services: Overview," 2017. [Online]. Available: http://tas3.eu/.

[57] S. Gürses, M. Seguran and N. Zannone, "Requirements engineering within a large-scale security-oriented research project: lessons learned," 2013. [Online]. Available: http://dx.doi.org/10.1007/s00766-011-0139-7.

[58] eclipse Foundation, "ProR Requirements Engineering Platform," 2017. [Online]. Available: http://www.eclipse.org/rmf/pror/.

[59] ANSYS, “medini analyze - functional safety,” ANSYS medini Technologies AG, 2017. [Online]. Available: http://www.medini.eu/index.php/en/products/functional-safety. [Accessed 18. May 2017].

[60] ANSYS, “medini analyze - Quality, Safety and Reliability Engineering,” [Online]. Available: http://www.medini.eu/images/stories/ikv/pdf/medinianalyzekeyfacts.pdf. [Accessed 18. May 2017].

[61] SESAMO, “Security and Safety Modelling,” ARTEMIS Joint Undertaking (Grant Agreement No. 295354), [Online]. Available: http://sesamo-project.eu/. [Accessed 18. May 2017].

[62] M. Howard and S. Lipner, “The Security Development Lifecycle,” 2006. [Online]. Available: https://aka.ms/SDL/PDF.

[63] ISASecure, “Vulnerability Identification Testing Tool,” 2017. [Online]. Available: http://isasecure.org/en-US/Test-Tools/Vulnerability-Identification-Testing-Tool.

[64] ISASecure, “CRT Test Tools,” 2017. [Online]. Available: http://isasecure.org/en-US/Test-Tools/Recognized-CRT-Test-Tools.

[65] MITRE, “CWE-Compatible Products and Services,” 2017. [Online]. Available: https://cwe.mitre.org/compatible/compatible.html.

[66] NIST Computer Security Resource Center, “NVD - SCAP Validated Tools,” 2017. [Online]. Available: https://nvd.nist.gov/scap/validated-tools#scap_labs.