DOI 10.5162/sensoren2016/6.3.4

Software in Measuring Instruments: Ways of Constructing Secure Systems

Daniel Peters[1], Florian Thiel[1]
[1]Physikalisch-Technische Bundesanstalt, Abbestr. 2 -12, Berlin, Germany
{daniel.peters, florian.thiel}@ptb.de

Abstract

In the era of the Internet of Things (IoT), the number of connected devices is expected to exceed 25 billion in the year 2020. This also concerns legal metrology. Legal metrology comprises measuring instruments that are employed for commercial or administrative purposes or for measurements which are of public interest. More than 100 million legally relevant meters are in use in Germany. The central concern of legal metrology is to protect and ensure trust. For software, this also means that applications must be stable and withstand attacks. These attacks increase in all areas where devices are connected via an open network, i.e. the internet. Additionally, measuring instruments have evolved into powerful universal devices with unsecure system architectures. Such IT systems, running conventional operating systems, can be hardly secured. One solution to enforce security is by creating a component-based architecture which modularizes the critical software parts and isolates them. In this paper some methods are described, which can be used to achieve software separation, and therefore enhance security and flexibility.

Keywords: **B3**: Informations- und Datenfusion; **B4**: Diagnose von Messgeräten, Selbstüberwachung, Zuverlässigkeit; **C1**: Sensoren für das Internet of Things; **C8**: Sicherheitstechnik, Safety and Security

1. Introduction

What does "*software separation*" in legal metrology mean? Generally, one can say that software separation describes technical measures that prevent non-legally relevant software functions, which have no measurement purpose, from influencing legally relevant ones, which are solely devoted to measurement.

If software separation has been implemented fully and correctly, non-legally relevant software components can and may be exchanged or modified by the manufacturer and even user, after the measuring instrument is in commission. No conformity assessment is necessary.

To achieve software separation for measuring instruments under legal control, the requirements of the WELMEC Software Guide 7.2 (W7.2) [14] should be followed; concretely, the sections *S1-S3*, which formulate the requirements for software separation. These requirements describe the application level and assume that the manufacturers control both pieces of software (legally relevant and non-legally relevant) and that they can ensure compliance with the requirements. But if the manufacturers have no full control of all the software parts, the legally relevant software needs to be protected against unknown influences by additional measures. These additional measures depend on the respective hardware and software platform, e.g., support for the separation of program and data areas, management of the common resources, and access to the system by the user, etc.

For manufacturers and notified bodies (NBs), the software separation as described in the W7.2 has also a second benefit : It helps to decide which depth of testing for the various software components of a measuring instrument needs to be applied, and therefore, reduces the expenses for modifications in software throughout the life cycle. In general, one can say that the primary aim of the modularization according to W7.2 / S1-S3 is to facilitate conformity assessment, and not necessary to hinder unknown manipulations of software parts.

Often the used platform of the measuring instrument does not have mechanisms to protect the legally relevant software part against interference from other software components, even if they are separated according to the rules of the W7.2 / S1-S3.

In this article different mechanisms to separate software are named and analyzed according to their strengths and benefits.

1.1 Outline

The paper is structured as follows: In Section 1, an introduction about the topic was given, outlining the importance of software separation. In Section 2, an overview of legal metrology and especially the software requirements for measuring instruments under legal control are supplied. Afterwards in Section 3, technical solutions on how to achieve software separation according to the WELMEC 7.2 Software Guide (W7.2) are discussed, before more sophisticated approaches are being described in Section 4. In Section 5, everything is rounded up by the conclusion.

2. Software in Legal Metrology

Measuring instruments that are employed for commercial, administrative purposes or are of public interest, fall under legal control. More than 100 million legally relevant meters are in use in Germany [6]. The majority of them are used for business purposes, in particular they are commodity meters for the supply of electricity, gas, water or heat. Other examples are counters in petrol pumps, scales in the food sector, speed and alcohol meters. The commonality of all these applications is that the person executing or being affected by an official measurement cannot check the determined result, the parties concerned must rather rely on the accuracy of the measurement. Hence, the central concern of legal metrology is to protect and ensure that trust.

The International Organization of Legal Metrology (OIML) was set up to assist in harmonizing such regulations across national boundaries to ensure that legal requirements do not lead to barriers in trade. Software requirements for this purpose are formulated in the OIML D 31 document [7]. In Europe, WELMEC is the committee to promote cooperation in the field of legal metrology, for example by establishing guides like the WELMEC 7.2 Software Guide [14].

2.1 MID

Directive 2014/32/EU of the European Parliament and of the Council [9] that is based on *Directive 2004/22/EC* [8], known as the Measuring Instruments Directive (MID), are directives by the European Union to establish a harmonized European market for measuring instruments, which are used in different member states. The aim of the MID is to protect the consumer and to create a basis for fair trade and trust in the public interest. The directive is limited to ten types of measuring instruments that have a special economic importance because of their number or their cross-border use. These are:

- water meters,
- gas meters and volume conversion devices,
- active electrical energy meters,
- heat meters,
- measuring systems for the continuous and dynamic measurement of quantities of liquids other than water,
- automatic weighing instruments,
- taximeters,
- material measures,
- dimensional measuring instruments,
- and exhaust gas analysers.

The MID defines basic requirements for these measuring instruments, e.g. the protection against tampering and the display of billing-related readings.

Each measuring instrument manufacturer themselves decide which technical solutions they want to apply. Nevertheless, they must prove to a notified body that their instrument complies to the MID requirements. The notified bodies that must be embraced by the manufacturers are denominated by the member states. In Germany, for example, the Physikalisch-Technische Bundesanstalt (PTB) is such a notified body. The PTB is furthermore the German national metrology institute providing additional scientific and technical services, which is why it achieves the demanded technical expertise needed. In general, the combination of technical expertise related to the measuring instruments, competence for the assessment, monitoring of product related quality assurance systems, and experience with European regulations, are required. Additionally it is of particular importance that the notified body is independent and impartial.

2.2 MID Software Requirements

The WELMEC 7.2 Software Guide tries to break down the requirements for legal metrology software of the MID to specific technical examples and recommendations. Actually, the last chapter of the guide is solely devoted to document how the proposed

guidelines are mapped to these requirements. The important MID software requirements are:

- Reproducibility of measurement results must be guaranteed, even if handled by different users.

Reproducibility implies that a measurement result should not depend on the user/consumer employing the instrument. From the software point of view, different processes with varying access rights performing the same measurement should yield the same result.

- Durability of the measuring instrument's software over a period of time must be guaranteed. A measuring instrument shall be designed to reduce as far as possible the effect of a defect (bug) that would lead to an inaccurate measurement result, unless the presence of such a defect is obvious.
- A measuring instrument shall have no feature likely to facilitate fraudulent use, while possibilities for unintentional misuse shall be minimal.

The latter points describe the measures to be complied to, for reducing the impact of manipulations and bugs as far as possible.

- A measuring instrument shall be designed so as to allow the control of the measuring tasks after the instrument has been placed on the market and put into use. Software for this control must be available.
- Software identification shall be easily provided by the measuring instrument.
- Evidence of an intervention shall be available for a reasonable period of time.

The former points directly address software requirements for verifying measuring instruments in commission. Validating the software identification ensures that software was not switched or manipulated. Ancillary, an audit trail is needed to log interventions.

- If a measuring instrument has associated software which provides other functions besides the measuring function, the software that is critical for the metrological characteristics shall be identifiable and shall not be inadmissibly influenced by the associated software.
- The metrological characteristics of a measuring instrument shall not be influenced in any inadmissible way by the connection to it of another device, by any feature of the connected device itself or by any remote device that communicates with the measuring instrument.
- Software that is critical for metrological characteristics shall be identified as such and shall be secured.
- Measurement data, software that is critical for measurement characteristics and metrologically important parameters stored or transmitted shall be adequately protected against accidental or intentional corruption.

These points demand a strict separation of legally relevant parts and legally not relevant ones. Furthermore, legally relevant parts should be protected from any malicious intrusion.

- For utility measuring instruments the display of the total quantity supplied or the displays from which the total quantity supplied can be derived, whole or partial reference to which is the basis for payment, shall not be able to be reset during use.
- The indication of any result shall be clear and unambiguous and accompanied by such marks and inscriptions necessary to inform the user of the significance of the result.
- Easy reading of the presented result shall be permitted under normal conditions of use.
- Additional indications may be shown provided they cannot be confused with the metrologically controlled indications.
- A durable proof of the measurement result and the information to identify the transaction shall be available on request at the time the measurement is concluded.

Finally, there shall be no confusion between data generated from legally relevant modules and data from irrelevant ones, by marking them distinguishable on the screen and on prints. Additionally, relevant data, which is the basis for payment, shall not be deleted or resettable until the payment is conducted.

2.3 WELMEC

WELMEC is the European cooperation responsible for legal metrology in the European Union and the European Free Trade Association (EFTA). Currently, representative

national authorities from 37 countries are part of the WELMEC Committee.

WELMEC Working Groups (WG) are established by the WELMEC Committee for the detailed discussion of issues. Currently, there are eight active Working Groups and one of them (WG7) is solely responsible for software questions and issues the WELMEC 7.2 Software Guide (W7.2). As of this writing its current version is WELMEC 7.2 Issue 5 [14], with Issue 6 near its completion. The WELMEC 7.2 Software Guide provides guidance to manufacturers and to notified bodies, on how to construct or check secure software for measuring instruments. Although it is based on the MID and its addressed instruments, its solutions are of general nature and may be applied beyond. The document states that by following this guide, a compliance with the software-related requirements contained in the MID can be assumed.

The W7.2 defines six risk classes from A - F, evaluating the need for software protection, software examination and software conformity. The risk classes are ascending in their demand for security, meaning that risk class A instruments do not need any security-awareness mechanisms and risk class F instruments need the highest. Specific groups of measuring instruments are then assigned to one risk class, e.g. petrol pumps are assigned to risk class C.

Additionally, the W7.2 differentiates between measuring instruments that are built solely for the measuring purpose and the ones that run universal software. The two classes are called P and U. Normally one can say, if a measuring instrument has an operating system installed, it is a U type, else it is situated in the P class. For both classes, four subclasses are defined which deal with following IT functions:

- L: long-term storage of measurement data,
- T: transmission of measurement data,
- D: software download,
- S: software separation.

This document focuses on the software separation part (S) which is discussed in detail in the following sections.

3. Simple Software Separation

As mentioned before the W7.2 formulates three requirements for software separation. These are:

S1: Realisation of software separation
"There shall be a part of the software that contains all legally relevant software and parameters that is clearly separated from other parts of software."

S2: Mixed indication
"Additional information generated by the software, which is not legally relevant, may only be shown on a display or printout, if it cannot be confused with the information that originates from the legally relevant part."

S3: Protective software interface
"The data exchange between the legally relevant and legally non-relevant software must be performed via a protective software interface, which comprises the interactions and data flow."

Additionally the W7.2 also differentiates between low-level and high-level separation. These points are explained below and analyzed for their conformity to S1-S3.

3.1 Low-Level Separation

According to W7.2, low-level separation means that software separation is realized independently from the operating system within an application domain, i.e., at the programming language level.

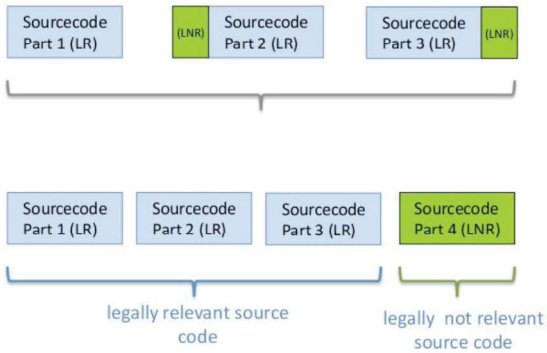

Fig. 1: Low-level separation according to W7.2

Figure 1 shows such a separation. Hereby, the source code is modularized into separate files. This is a first step to achieve software separation and helps in the coding process because a clean modularized environment makes locating and emending bugs easier. Still, this is not enough if one executable is generated, as can be seen in Figure 2.

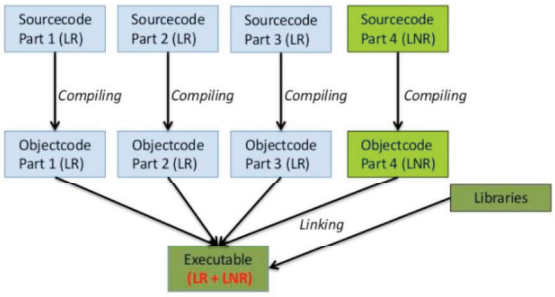

Fig. 2: Software separation that still leads to one executable

At execution time a single binary is running on the device. In this binary legally relevant functions are again mixed together with non-legally ones, hence S1 and S3 is not satisfied. An example where two separate binaries are compiled can be seen in Figure 3.

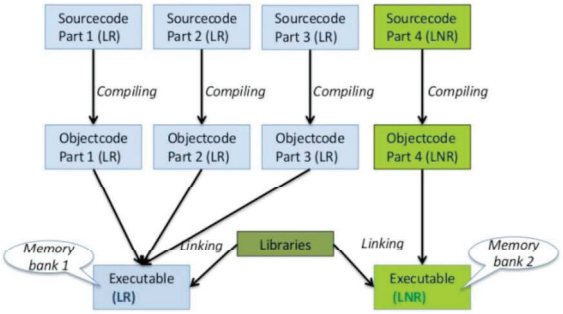

Fig. 3: Software separation leading to two executables

Hereby, the separation of the two executables is achieved by copying the executables in separate memory banks. The data transfer between the executables can be managed through shared libraries. S1 is being satisfied, still S2 and S3 must be checked. Especially S3 states that the shared libraries must be protective software interfaces, i.e. legally-non relevant software is not allowed to effect legally relevant one in an unwanted way. Hence the libraries are legally relevant software.

3.2 High-Level Separation

High-level separation means that the software modules to be separated are realized as independent objects with the help of the operating system. An example which is similar to the last one is shown in Figure 4.

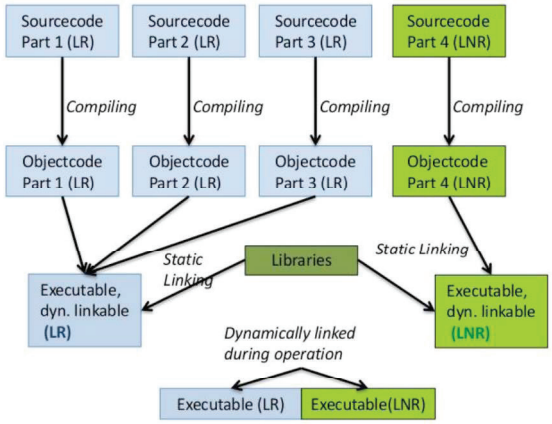

Fig. 4: Difference between statically linked in libraries and dynamic ones

The different source code files generate different executables. Here again, libraries are the parts of code that are used by both executables. If the libraries are statically linked into the executables, the binaries are independent of each other, and the operating system makes sure that the applications are isolated. For this purpose, the operating system needs mechanisms to make isolation possible, like the use of separate address spaces for different processes. For general purpose operating system like windows and Linux this is the case. Still, many known bugs generate vulnerabilities in these operating systems that are used to subvert the isolation. Hence, it is important to do a risk analysis of the specific measuring instrument to check if the needed security mechanisms are upheld by the respective operating system.

If the libraries are dynamically linked into the binaries, they represent shared code and need to be checked accordingly to S3 as mentioned before. These libraries can then be used for communication purposes between legally relevant and non-legally relevant software.

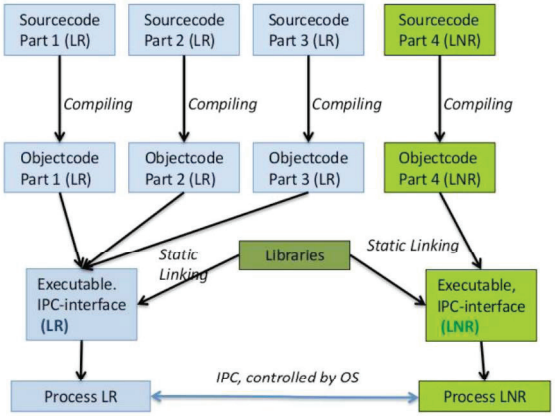

Fig. 5: Inter Process Communication (IPC) controlled by the operating system

Lastly, Figure 5 shows a completely controlled communication by operating system mechanisms. Here, the used libraries are statically linked into the binaries, constructing separate isolated processes. These processes can than communicate through Inter-Process Communication (IPC) with each other, which is regulated by the operating system. Nevertheless, one has to trust the operating system, as mentioned before the level of trust needed can be checked by a risk analysis.

4. Stronger Software Separation

4.1 Security Kernels

A security kernel in a system ensures that subjects have access only to objects that are given to them by a security policy. A common way of expressing these requirements is given by the acronym NEAT, which defines the following criteria for security kernels:

- *Nonbypassable*: The safety concept of the system cannot be bypassed. Components cannot create communication paths, different from the determined ones to bypass the safety concept.
- *Evaluatable*: The security architecture is small and has a low level of complexity, in order to make a formal verification possible. The components must be small and modular, to facilitate verification.
- *Always-invoked*: The safety concept is always active. Every access and communication must be checked and accepted by the security architecture (the security architecture normally verifies only the first access to an object, all other requests are forwarded without a recheck to speed up the process).
- *Tamper-proof*: The system has a strict access control management, specifically handling the modification of data or code. The security architecture strictly controls which components can modify the system to prevent unauthorized changes.

A security kernel is not necessary an operating system kernel. Rather, it refers to those components that perform the functions of a reference monitor within an operating system kernel. If access to a sensitive object is requested, the system first asks this reference monitor for permission. The reference monitor itself then checks the access rights by some kind of policy table. Hereby, the objects can be hardware, e.g. CPUs, memory segments, hard-disk blocks; or software, e.g. processes or files.

Usually general purpose operating systems use a **Discretionary Access Control (DAC)** model, in which the individual users are able to decide who can have what access to their documents. In systems were stronger security is needed, the systems themselves should have additional rules that decide what objects can be accessed by whom, e.g. in a hospital, the doctor should not be able to give the janitor reading or writing rights to his patience database. These systems should have so called **Mandatory Access Control (MAC)** models in place, for example the *Bell-La Padula* [5] model or the *Biba* [3] model.

A well-known example of a security kernel is **SELinux** [10], an implementation of the **Flux Advanced Security Kernel (FLASK)** [11] for Linux. SELinux replaces the normally used Discretionary Access Control (DAC) by the more restrictive one, the Mandatory Access Control (MAC). In general, Flask utilizes a security server to make access decisions and object managers that enforce those decisions. This separation of access control decision from enforcement, allows the support of flexible mandatory access control.

From a legal metrology point of view, a security kernel, if correctly set up, fulfills all separation requirements of W7.2: S1 (Realisation of software separation), S2 (Mixed indication), and S3 (Protective software interface).

4.2 Separation Kernels

Generally, operating system architectures are subdivided into two main designs, the monolithic kernel and the microkernel system architecture. The main difference between those two is that a monolithic kernel system is working in privileged mode sharing a single memory space with the system software, such as file systems and complex device drivers with direct access to the hardware. In former years this was great for performance reasons, because user applications are able to access most services, e.g. I/O devices and TCP/IP networking, with a simple and efficient system call. The disadvantage of this approach is that the system parts are not secured from each other and one bug in a component can affect all other components in the system.

In the microkernel design, the microkernel is the only software executed at the most privileged level. Hence in contrast to a monolithic design, services are implemented in

separate processes and secured against each other. Hereby, stability is gained because, for example, a crash in the network stack that would have been fatal for a monolithic system is now survivable. Even well-engineered code can have several defects per thousand SLOC [4], which leads to the conclusion that a bigger system should have inherently more bugs than a small system. For comparison, modern microkernels have around 15K SLOC and less, the monolithic kernel of Linux (version 3.6) at least 300K SLOC to a maximum of 16M SLOC, depending on the configuration.

A separation kernel [15] is a special microkernel, which divides the system into partitions - sometimes also called domains. This software component ensures complete separation of the partitions from each other, both in time and space. Partitions can only communicate with each other through strictly controlled channels. The term separation kernel originates from the field of embedded systems, where isolation of individual components often plays an important role. Accordingly, the requirements for separation kernels are very high. The ARINC653 [13] specification defines requirements that need to be fulfilled by operating systems, to be approved in areas where functional safety must be guaranteed. There are four requirements for the operating systems, which must be met by their separation kernel:

1. Temporal separation
2. Spatial separation
3. Information flow control
4. Fault-isolation

The term separation kernel is often used in conjunction with “Multiple Independent Levels of Security / Safety” (MILS) [1, 2]. Hereby, the separation kernel represents the lowest layer of the architecture. In the partitions, a middleware layer is running as a connecting plane to the applications. This is needed because the provided interfaces of the separation kernel are often very rudimentary and provide only a minimum of functionality, to keep the complexity in the kernel low. Therefore, the middleware implements missing functionality, often in the form of libraries to offer applications a standardized interface (e.g. POSIX). These libraries contain a variety of functions such as memory management, threading or just mathematical functions. However, the middleware can also offer a virtualization layer, which enables the partitions to run operating systems with a wider range of functions, for example, Linux or Windows.

4.3 Virtualization

With virtualization, standard operating systems that offer great functionality, a familiar user interface and many working drivers can be used. Still, security is ensured due to the encapsulation and modularization of the software.

In [12], for example, a software reference architecture for measuring instruments is described, which is based on a microkernel and virtualization. The microkernel runs on the lowest level, under the actual operating systems. These operating systems in turn, are encapsulated into modules, so-called virtual machine (VM). The operating systems can continue to load their usual programs and drivers, but are obligated to communicate via the microkernel with each other and the hardware. The system architecture is based on a modular design that fulfils the requirements of the Measuring Instruments Directive of the European Union (MID) and the WELMEC 7.2 Software Guides (W7.2). These can be seen in Figure 6 and are as follows:

- Displaying data (Secure GUI),
- Data protection (Key & Signature Manager),
- Storing data (Storage Manager),
- executing downloads (Download Manager),
- Transferring data (Connection Manager),
- Internal data processing (Communication Monitor).

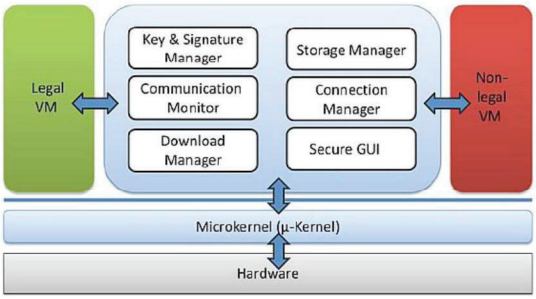

Fig. 6: Communication between the individual modules within the system architecture

Hence, the reference architecture ensures that all legally relevant measurement functions can be monitored safely. In addition, the architecture separates non-legally relevant software (N) and legally relevant software (L). All calculations that fall under legal control are carried out in the L-VM, everything else in the N-VM. This strict separation ensures that legally relevant software is not irregularly affected.

4.4 Hardware Separation

A more secure way than virtualization to separate software is by directly using separate hardware, e.g. two central processing units (CPUs). One is solely devoted to calculate legally relevant task and one is doing only non-relevant calculations, as can be seen in Figure 7.

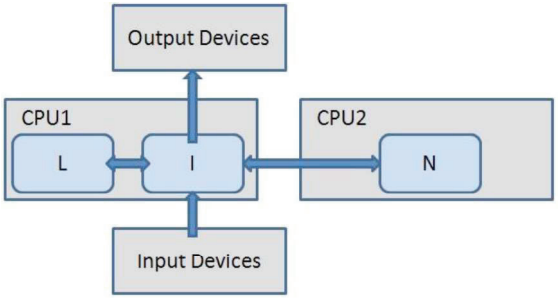

Fig. 7: Software separation through hardware

Again, the communication interface is legally relevant and needs to be checked to fulfill S3, if the two CPUs are communicating data to each other or use the display together. This method is the most expensive one, because it needs additional hardware, which the other methods do not need.

5. Conclusion

In this paper a concrete description of many known methods to achieve software security by separation was given. First, normal methods were described, which the WELMEC 7.2 Software Guide calls low and high level separation. Afterwards, more sophisticated approaches were disscussed, like security and separation kernels. The advancement beyond the state of the art in this paper consists of both the detailed description of these methods and their validation for security aspects in the context of existing documents that are used to validate measuring instruments in legal metrology.

References

[1] J. Alves-foss, W. Scott Harrison, P. Oman, and C. Taylor. The mils architecture for high-assurance embedded systems. Journal of Embedded Systems, 2:239-247, 2006.

[2] R. Beckwith, W. Mark Vanfleet, and L. MacLaren. High assurance security/safety for deeply embedded, real-time systems. Systems Conference . Citeseer, 2004.

[3] K. J. Biba. "Integrity Considerations for Secure Computer Systems", MTR-3153, Mitre Corporation, Juni 1975.

[4] B. Chelf. Measuring Software Quality - A Study of Open Source Software. Coverity, 2011.

[5] D. Elliott Bell, and Leonard J. LaPadula: Secure Computer Systems: Mathematical Foundations. MITRE Corporation, 1973

[6] N. Leffler, and F. Thiel. Im Geschäftsverkehr das richtige Maß. In Schlaglichter der Wirtschaftspolitik, Monatsbericht November, 2013.

[7] OIML D 31. General requirements for software controlled measuring instruments, 2008.

[8] Official Journal of the European Union. DIRECTIVE 2004/22/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, March 2004.

[9] Official Journal of the European Union. DIRECTIVE 2014/32/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, February 2014.

[10] online at https://www.nsa.gov/research/selinux/index.shtml, accessed 26.02.2016

[11] online at http://www.cs.utah.edu/flux/fluke/html/flask.html, accessed 26.02.2016

[12] D. Peters, M. Peter , J.-P. Seifert, and F. Thiel: A Secure System Architecture for Measuring Instruments in Legal Metrology. Computers - Open Access Journal 4(2), 61-86, 2015

[13] Slawomir Samolej. Arinc specification 653 based real-time software engineering. E-Informatica , 5(1):39-49, 2011.

[14] WELMEC European cooperation in legal metrology. WELMEC 7.2 Issue 5 Software Guide, March 2012.

[15] Y. ZHAO, M. Dianfu, and Y. Zhibin. A survey on formal specification and verification of separation kernels. Technical report, Tech. rep., National Key Laboratory of Software Development Environment (NLSDE), Beihang Univerisity, 2014.