Computers **2015**, *4*, 61-86; doi:10.3390/computers4020061

OPEN ACCESS

computers

ISSN 2073-431X

www.mdpi.com/journal/computers

Article

A Secure System Architecture for Measuring Instruments in Legal Metrology †

Daniel Peters [1,*], Michael Peter [2], Jean-Pierre Seifert [2] and Florian Thiel [1]

[1] Physikalisch-Technische Bundesanstalt (PTB), Abbestrasse 2–12, 10587 Berlin, Germany; E-Mail: florian.thiel@ptb.de

[2] Security in Telecommunications, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany; E-Mails: peter@sec.t-labs.tu-berlin.de (M.P.); jpseifert@sec.t-labs.tu-berlin.de (J.-P.S.)

† This paper is an extended version of our paper published in Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014 (EAIS), Warsaw, Poland, 7–10 September, 2014.

* Author to whom correspondence should be addressed; E-Mail: daniel.peters@ptb.de; Tel.: +49-30-3481-7916; Fax: +49-30-3481-7506.

Academic Editor: Aaron Quigley

Received: 29 December 2014 / Accepted: 16 March 2015 / Published: 24 March 2015

Abstract: Embedded systems show the tendency of becoming more and more connected. This fact combined with the trend towards the Internet of Things, from which measuring instruments are not immune (e.g., smart meters), lets one assume that security in measuring instruments will inevitably play an important role soon. Additionally, measuring instruments have adopted general-purpose operating systems to offer the user a broader functionality that is not necessarily restricted towards measurement alone. In this paper, a flexible software system architecture is presented that addresses these challenges within the framework of essential requirements laid down in the Measuring Instruments Directive of the European Union. This system architecture tries to eliminate the risks general-purpose operating systems have by wrapping them, together with dedicated applications, in secure sandboxes, while supervising the communication between the essential parts and the outside world.

Keywords: legal metrology; secure system architecture; WELMEC; Measuring Instruments Directive (MID); microkernel; virtualization; hypervisor; software integrity

---

1. Introduction

In light of the fact that around 98% of the world's computer systems are embedded devices [1], security in embedded systems will inevitably play an important role in computer science. This fact is further supported by the tendency of these systems to become more and more connected over insecure networks, e.g., the Internet. Challenges in constructing a secure embedded system arise due to the increase in complexity, which is driven by the growing demand for improved capabilities, the digitization of manual and mechanical functions and interconnectability.

According to estimations, about four to six percent of the gross national income in industrial countries is accounted for by measuring instruments, which are subject to legal control [2], e.g., electricity meters, gas meters and their related measurements. In Germany alone, this corresponds to an amount of 104 to 157 billion Euros each year [2]. Hence, manipulations of measuring instruments' software could have far-reaching financial consequences. Clearly, special measures should be considered to secure such instruments.

Nowadays, most of the manufacturers of measuring instruments prefer building their software stacks on general-purpose operating systems (GPOSs), like Linux and Windows, due to the wide availability of device drivers, software infrastructure and applications. These systems were not created with high security awareness in mind. Their total embedded software content often exceeds 10 million source lines of code (SLOC). The alarming danger becomes clear when knowing that tests place the number of severe bugs in well-written open source software code at the rate of about one per 2000 SLOC [3]. Such a high amount of erroneous code consequentially increases the vulnerability of these systems to attackers, because just one bug could be so severe that a sophisticated attacker is able to run arbitrary code on the measuring instrument. The National Vulnerability Database (a catalog of software bugs published by the U.S. National Institute of Standards and Technology and the U.S. Department of Homeland Security's National Security Cyber Division), for example, reveals such bugs weekly.

First of all, one should have a look at the basic lawful requirements a measuring instrument in legal metrology has to meet with respect to security. Here, consumer protection and the certainty of a correct measurement are most important. A consumer must be sure that, for example, a fuel dispenser is not manipulated to charge more than what was fueled. Hence, in Europe, member states denominate institutions, called notified bodies, which are responsible for reviewing the measuring instruments before commissioning. Current scrutiny in the laboratory concentrates on the validation of correct measurements from the hardware parts, e.g., physical sensors. Software analysis is hampered by obstacles, like proprietary software, e.g., the operating system, where source code cannot be checked. The approval for commission is often given after a "black box" validation of the sensors, by application of some test vectors at the user interface and the sealing of as many interfaces as possible. The aforementioned fuel dispenser is stated to be not manipulated as long as no seal is broken. As mentioned before, this assumption is too optimistic, considering that just one open interface, e.g., an USB-port or WiFi, can allow a sophisticated attacker to run arbitrary code by exploiting a single vulnerability.

The framework presented in this paper addresses powerful measuring instruments that are able to run the newest general-purpose operating systems. Such measuring instruments are very common nowadays and are, for example, traffic enforcement meters, moisture meters, fuel dispensers, and many more.

However, even small measuring instruments, like dosimeters, often present their measuring results on a separate display device, which can be a tablet running Android. For these powerful measuring instruments we present a new secure software framework, which uses known and sound techniques, like a microkernel/hypervisor and virtualization, and combines them with the requirements of directives and guides for measuring instruments in Europe, which are under legal control.

Outline

The remainder of the paper is organized as follows:

- Section 2 provides an introductory part about legal metrology, especially legal requirements for measuring instruments in Europe. Additionally, these requirements are compared with an international standard for IT security, *i.e.*, Common Criteria.
- Section 3 talks about the basics that help to construct a secure software system, like microkernels and virtualization. Afterwards, policies of secure systems and measures to achieve them are named.
- Section 4 shows our framework and describes its distributed system nature.
- In Section 5, the duties of the individual modules (virtual machines) of our system architecture are described.
- Section 6 goes into more detail describing a special virtual machine, named the Inspector, which is a monitoring virtual machine. It is responsible for system integrity checking and software-based attestation.
- In Section 7, we show the feasibility of our approach by an implementation on a demonstrator. We start all of the VMs on this demonstrator and measure the times that are necessary to transmit information from one VM to another through a virtual network.
- Section 8 analyzes the system by showing that the policy data isolation, information flow control, damage limitation and period processing are upheld.
- Section 9 sums up related work about microkernel and hypervisor architectures.
- In Section 10, we end the paper with a conclusion.

2. Legal Metrology

Legal metrology is comprised of measuring instruments that are employed for commercial or administrative purposes or for measurements that are of public interest. More than 100 million legally relevant meters are in use in Germany [2]. 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 classical measuring instruments, with which the end user comes into contact, are, e.g., counters in petrol pumps or scales in the food sector. Measuring instruments are to a large extent also required in the public traffic system. Examples are speed or alcohol meters. The commonality of all of 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. In this context, legal metrology makes a lasting contribution to a functioning economic system by simultaneously protecting the consumers.

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 [4].

WELMEC is the European committee to promote cooperation in the field of legal metrology, for example by establishing guides to help notified bodies (responsible for checking the measuring instruments) and manufacturers implement the Measuring Instruments Directive described below.

2.1. Measuring Instruments Directive

Directive 2014/32/EU of the European Parliament and of the Council [5], which is based on Directive 2004/22/EC [6], 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 analyzers. 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 with 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. 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 of interest and concern to WELMEC Members and Associate Members. 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. As of this writing, its current version is WELMEC 7.2 Issue 5 [7], 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 a 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.

Out of the MID and the assessment of hazards, the WELMEC 7.2 Software Guide 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 (no measuring instrument is classified that low) and risk Class F instruments need the highest (There is also no measuring instrument, included in the regulations of the MID, specified at risk Class F. Still, there are nationally variable ranked measuring instruments, e.g., in Germany, traffic enforcement cameras are viewed as F measuring instruments). Specific groups of measuring instruments are then, in all conscience, assigned to one risk class, e.g., taximeters are assigned to risk Class D. For our purposes, the best way to start is to construct a system architecture for risk Class F, because a system architecture for risk Class F conforms to all requirements of the other classes. For risk Class F measuring instruments, the following three main points must hold true:

(1) Software protection against deliberate modifications with sophisticated software tools;
(2) Ambitious software assessment with examination of source code has taken place;
(3) Software conformity: the software implemented in the individual instruments is completely identical to the approved one.

The WELMEC Guide 7.2 goes into more detail on achieving these goals. The different recommendations are listed in Section 4 and collated to their appropriate modules in the system architecture.

Before constructing a secure measuring instrument software architecture, it is important to clarify legally relevant parts, because only these parts are critical, while of course ensuring that non-legally relevant parts do not effect legal ones. According to WELMEC 7.2, all modules are legally relevantthat make a contribution to or influence measurement results. These modules facilitate auxiliary functions, like displaying data, protecting data, saving data, identifying the software, executing downloads, transferring data and checking received or stored data.

2.3. Common Criteria and WELMEC

The ISO/IEC 15408, more commonly known as the Common Criteria for Information Technology Security Evaluation, is, with 26 participating countries, the international standard for IT security. The Common Criteria provide guidance for security evaluation by describing generic requirements for major security functionalities of IT products and assurance measures to be applied to these functionalities. Under Common Criteria, products are evaluated against Protection Profiles—PP (more specifically the Security Target—ST), which are the means to adapt the generic requirements to particular application areas. In this context, functional requirements are the security policies or protections a product claims to implement, and assurance requirements are the controls that the developer has to follow to ensure the realization of these functional requirements.

Basically, the development process of a Protection Profile starts with an analysis of the threats to which the target of evaluation (TOE) is exposed. Furthermore, assurance requirements are collectively

tagged with an Evaluation Assurance Level (EAL) that represents the overall confidence stakeholders can have in the security functional requirements, *i.e.*, the confidence that these are actually met by the conforming product. Common Criteria assurance levels range from EAL1 to EAL7, and their meanings are shown in Table 1.

Table 1. Comparison between Common Criteria (CC) Evaluated Assurance Levels (EALs) and WELMEC risk classes (WRC).

CC	Meaning	WRC
EAL1	Functionally tested	B
EAL2	Structurally tested	C
EAL3	Methodically tested and checked	D
EAL4	Methodically designed, tested and reviewed	E
EAL5	Semiformally designed and tested	F
EAL6	Semiformally verified design and tested	-
EAL7	Formally verified design and tested	-

Table 1 also tries to compare the requirements of the EALs with the risk classes of the WELMEC 7.2 Software Guide. The juxtaposition in Table 1 is not a fixed mapping; it should be more seen as a guidance for the future. That is to say if a product meets EAL4 and was designed, tested and reviewed for measurement purposes, it satisfies the requirements of risk Class E. This comparison justifies the use of Windows or Linux operating systems for measuring instruments that need to be tested for risk Class E requirements, because the latter mentioned GPOSs are partially certified at EAL4. In our opinion, software for measuring instruments for risk Class F should be semiformally designed and tested. Still, it should be noted that measuring instruments classified as F instruments, e.g., traffic enforcement cameras running GPOSs, are frequently approved by notified bodies. Another example are smart meter gateways in Germany, for which the the German Federal Bureau (BSI) demands an EAL4+ certification, hence a direct correlation between legal metrology directives and Common Criteria is apparent, in this case.

3. Background

Before going into the technical details of our system architecture, it needs to be clarified what software security, especially confidentiality, integrity and availability, stand for in this document, because the literature does not always give a consistent definition.

Security is the ability of a component to protect resources for which it announces protection responsibility, and the component's security policies are its security-enforcing properties and requirements. Generally, security policies try to enforce confidentiality, integrity and/or availability.

- Confidentiality ensures no unauthorized disclosure of information;
- Integrity prevents modifications or corruptions of a resource without authorization;
- Availability ensures that authorized users have access to resources whenever needed.

For every one of these three points, authorization plays a key role and so does authentication. Authentication ensures that an entity is who it claims to be, and authenticity means that the data and

communication partners are genuine. After authenticity is determined, authorization either grants the right to use a resource or denies that privilege.

3.1. Creating a Secure System

An operating system is the essential component for hardware abstraction in a software system. It acts as an intermediary between programs and the hardware and, from the security point of view, plays the most important part in constructing a secure system. Generally, operating system architectures are subdivided into two main designs, the monolithic kernel and the microkernel system architecture shown in Figure 1. It should be noted, that often, another architecture is mentioned, e.g., Windows is sold as a hybrid kernel architecture, combining aspects of a microkernel and a monolithic kernel architecture. We consider this kernel to be a "smaller" monolithic kernel, because in contrast to a microkernel, many (nearly all) operating system services are in kernel space, like in a monolithic kernel.

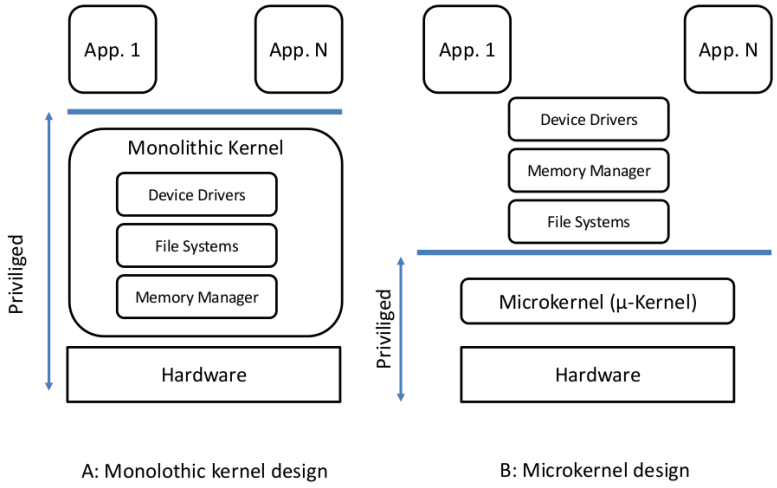

Figure 1. Comparison between a monolithic kernel design (**A**) and a microkernel (**B**).

In a monolithic kernel system architecture, the entire operating 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 Figure 1, the privileged mode modules have rounded corners and are depicted in red. The advantage of this architecture is performance, 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 the resulting large trusted computing base (TCB). The TCB refers to those portions of a system (software and hardware) that are needed in a system to ensure that it works as expected. Therefore, it must be trustworthy.

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; in Figure 1, represented as blue components with sharp edges. The motivation to place as much functionality as possible in separate protection domains, not running in privileged mode, is to gain stability, because, for example, a crash in the network stack that would have been fatal for a monolithic system is now survivable. Consequently, the goal of this architecture is to keep the TCB small and under control, as even well-engineered code can have several defects per thousand SLOC [3]. Hence, a bigger system has

inherently more bugs than a small system and often a bigger attack surface. For comparison, modern microkernels have around 15K SLOC or less, and the monolithic kernel of Linux (version 3.6) at least 300K SLOC to a maximum of 16M SLOC, depending on the configuration.

3.2. Virtualization

A major drawback in constructing a new software system on a microkernel is that drivers and software libraries available for known GPOSs, e.g., Linux, which uses a monolithic kernel design, need to be ported, or in the worst case, completely rewritten. Virtualization seems to be the right solution to incorporate the best implementations of both architectures in a single system. Through virtualization, device drivers and software libraries can be reused, which makes the implementation of our system architecture for a manufacturer as easy as possible.

Virtualization can be divided into two main approaches [8]. Pure virtualization, sometimes also referred to as faithful or full virtualization, supports unmodified guest operating systems, running atop another kernel, safely encapsulated. The advantage of this approach is that closed-source operating systems are directly executable. Commodity processors often do not have adequate support for pure virtualization, requiring complex technologies, such as binary translation, to be used [9]. For the second approach, called para-virtualization, the guest operating system is presented with an interface that is similar, but not identical to the underlying hardware to make virtualization possible. To improve performance and allow virtualization at all, the guest operating system is often modified. The approach used depends on the guest operating system that should be employed and the hardware features available.

In the past, embedded systems used to be relatively simple devices, and their software was dominated by hardware constraints, which made virtualization unattractive. Nowadays, most embedded systems have the characteristics of general purpose systems and increasingly have the power to actually run applications built for PCs. This power together with the low development costs for a board combined with virtualization technologies like ARM TrustZone [10] makes virtualization in the embedded world, and therefore, in measuring instruments, attractive.

A strong motivation for virtualization is security. By running an operating system in its own environment safely encapsulated—a so-called virtual machine (VM)—the damage of an attack is restricted to this virtual machine, because access to the rest of the system can be prohibited, as shown in Figure 2.

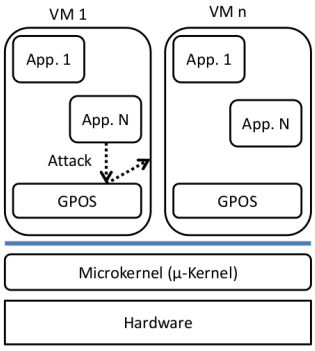

Figure 2. The general-purpose operating system (GPOS) of the VM in the middle is compromised by a pernicious application. Due to isolation, the other VMs are not vulnerable.

The access is not just restricted to the VM; the whole hardware access can be redirected through the underlying kernel through virtualized device drivers, preventing direct communication with the hardware or any hardware access at all. This isolation can only be assumed if the privileged software managing the virtual machines, called the virtual machine monitor (VMM) or hypervisor (in this paper, we do not differentiate between a VMM and a hypervisor, as is sometimes done. Both refer to the underlying microkernel) is correctly implemented. For example, in Xen—a popular VMM—all VMs depend not only on the hypervisor, but on a full operating system, because the first VM—called dom0—is launched by default with direct access to the hardware. From the dom0, the hypervisor can be managed, and unprivileged VMs can be launched. This makes the TCB very large and the construct vulnerable; actually, fully exploitable vulnerabilities to subvert the hypervisor and take over the whole system have been found in the past [11]. In our opinion a microkernel, because of its minimality principle, seems to be a good choice for implementing a hypervisor [12–17].

3.3. Policies of a Secure System

Multiple Independent Levels of Security/Safety (MILS) is a high-assurance security architecture based on the concepts of separation and controlled information flow. The foundation of the system is a small kernel, as used in our design, implementing a limited set of critical functional security policies. This special kernel, often called the separation kernel or partition kernel, implements the policies for information flow control, data isolation, damage limitation and period processing [18].

- Information flow control ensures that information cannot flow between partitions unless explicitly permitted by the system security policy.
- Data isolation ensures that a partition is provided with mechanisms, whereby isolation within, it can be enforced.
- Damage limitation ensures that a bug or attack damaging a partitioned application cannot spread to other applications.
- Period processing ensures that information from one component is not leaked into another one through resources, which may be reused across execution periods.

To guarantee these policies, the separation kernel must ensure the following properties:

- Non-by-passable: a component cannot use any communication path, including lower level mechanisms, to bypass the security functions.
- Evaluable: security assurance must be given by evaluable (even mathematically verifiable) security claims.
- Always-invoked: the security functions are invoked each and every time.
- Tamper proof: applications must not be able to tamper with the security policy or its enforcement mechanisms (e.g., by exhausting resources or overrunning buffers).

As stated in Section 3.1, modern monolithic kernels and, of course, whole GPOSs consist of tens (sometimes hundreds) of millions of SLOC. Hence, they are too difficult and expensive to evaluate. The separation kernel, which, with sometimes no more than 10K SLOC, is small enough to be thoroughly evaluated and mathematically verified for the highest assurance level. The applications managing

sensitive data are then built on top of the secure separation kernel. An advantage of this approach is its modularity, allowing software of varying security demands to run on the same microprocessor by means of software partitioning through the separation kernel. This way, the MILS security policies are also stacked, meaning that a module layered atop the separation kernel cannot circumvent the enforced restrictions, which the separation kernel defines for it.

3.4. Principles to Satisfy Security Properties

In our opinion the MILS system architecture, especially the separation kernel, together with VMs as its components, furnishes a great base for high-assurance software combined with the ability to run GPOS applications. An example of a well-designed separation kernel is Integrity. The Integrity operating system from Green Hills is the world's first software to achieve an EAL 6+/high robustness certification. Integrity was constructed with the PHASE (Principles of High Assurance Software Engineering) methodology in mind [19]. PHASE is a methodology that is not restricted to separation kernels; it can be used to create reliable software in general. In our case, reliability means to achieve the assurance level required for lawfully-used measuring instruments, which is indisputably high. As an aid to fulfil this goal, PHASE names five principles to be upheld:

(1) Minimal implementation;
(2) Component architecture;
(3) Least privilege;
(4) Secure development process;
(5) Independent expert validation;

As previously highlighted, the security of a system highly depends on the amount of code used to solve a problem. Therefore, finding a minimal implementation is one of the most important tasks in secure software development. In our proposed architecture, this also implies using the minimal configuration for the GPOS running in the virtual machine, e.g., while compiling Linux, only needed modules should be appended to the kernel.

A component architecture follows the principle of damage limitation: critical operations should be protected from non-critical ones by placing them in different components. Damage limitation is not the only benefit of a component-based architecture; other points to be named are the facility to swap modules for costumer-specific changes and the overall improved maintainability. A downside is the need for well-defined interfaces for communication between the components, in our case VMs. The principle of the MILS system architecture goes hand in hand with the component architecture principle.

Every component in the system should be designed with the least privilege concept in mind. The VMs should be given access only to those resources, e.g., I/O devices, that they absolutely require. In our proposed system, direct access to I/O devices is omitted as much as possible.

A secure development process is the basis for high-assurance software. It covers, for example, coding, testing, formal design and formal proof of a system and paves the way for an EAL6/7 certification. Our proposed framework should use all tools available to safeguard the development of the system from the beginning, like model-driven design, coding standards and static and dynamic code analysis.

In legal metrology, independent expert validation is stipulated and conducted by notified bodies. The manufacturer must provide its design documentation together with user manuals for testing purposes, e.g., to check if all input possibilities are documented and permitted. Additionally, complete access to the source code must be granted for risk Classes E and F measuring instruments.

4. Our Framework

Our proposed framework, shown in Figure 3, consists of three approaches.

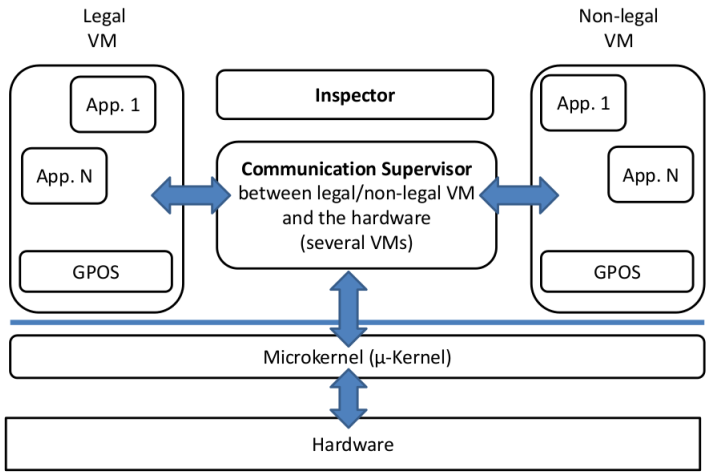

Figure 3. The framework.

Firstly, we separated the legally relevant parts from the irrelevant ones by putting them in different virtual machines. Secondly, we made sure that their virtual machines have no direct access to I/O devices and that they are scanned and integrity checked by a dedicated VM (the Inspector). Lastly, we constructed a secure framework (the Communication Supervisor) which provides services to these VMs. This framework, also consisting of separated VMs, monitors the information flow and correctly delegates requests from and to I/O devices. It is shown in more detail in Figure 4.

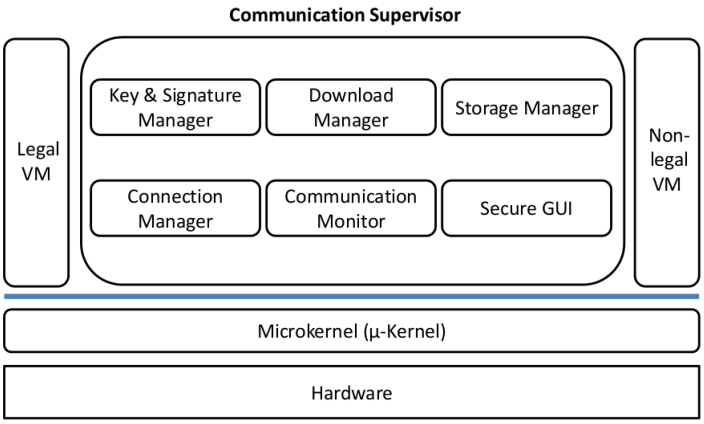

Figure 4. Communication Supervisor.

In the legal VM (L VM), all of the computations that are needed for the measurement procedure are executed, e.g., image processing in a traffic enforcement camera. The non-legal VM (N VM) is only

allowed to run software that has no measurement purpose, e.g., like showing the manual or starting a calculator. This strict separation ensures that non-legally relevant software has no effect on the legally relevant one, as postulated in the MID. The other blocks form our framework. Their general purpose is to supervise the communication between the L/N VM and the hardware. These modules can be native microkernel processes or VMs themselves. In our opinion, the VM approach seems to be the better one, because communication between the individual VMs can take place through network protocols already implemented in the GPOS, and the GPOS device drivers can be used [15,20]. Through the implemented network stacks, virtual private networks (VPNs) can be created to encrypt communication. The VM approach even allows modules to be transferred out to different computers, creating a distributed system over a network. A disadvantage of this concept is the invalidation of the minimal implementation principle, which should be counteracted by using minimal configurations for the GPOS.

Our framework consists of inclusively legally relevant modules, fulfilling legally relevant functions, as demanded in the WELMEC 7.2 Guide and mentioned in Section 2.2. The mapping of the functions to the modules is as follows:

- Secure GUI: displaying data;
- Key & Signature Manager: protecting data;
- Storage Manager: saving data, recording modifications;
- Inspector: identifying the software, software integrity checking;
- Download Manager: executing downloads;
- Connection Manager: transferring data over the network;
- Communication Monitor: redirecting queries from and to the I/O devices, e.g., sensors and keyboard.

4.1. System Requirements

The confidence of a user in a modular system is based on their confidence in the individual system components. Thus, an important aspect of the system is the necessity of a secure boot mechanism, where all modules are checked for authenticity and integrity. Only starting from an untampered kernel on an untampered central unit can the kernel check the other modules for authenticity. In the case of measuring instruments, where seals are used to detect hardware manipulation, the boot-loader should lie on a separate tamper-proof sealed storage unit that is not readable/writable for the other system components. The first check then starts at the boot process, where, for example, the hash value, e.g., the secure hash algorithm *SHA-2*, of the kernel binary is calculated and checked with the pre-calculated value stored in the storage unit of the boot-loader. If the two values are identical, the kernel can be loaded and starts with similar tests on the individual modules.

After a successful boot process, confidential conversation channels between the VMs must be established. The most important system requirement is a correct microkernel/VMM, which enforces isolation of the VMs and has no covert communication channels. It must be impossible to subvert the VMM or to attack other VMs through a compromised VM. The microkernel needs to assign unique unchangeable identifiers to the virtual network interface controllers of the VMs and must create buffers

for every virtual connection the system needs. The buffers are not directly accessible by the VMs; they only simulate a network transmission.

There must be a mechanism to switch from legally relevant to non-legally relevant mode. A hardware switch accessible by the VMs would be an example. If the switch is set to legal mode, every input from devices that can be used for non-legally relevant tasks, e.g., a keyboard, would be redirected to the non-legally relevant VM and to the legally relevant VM, the other way around. Another method would be to use a touch screen that is divided into legally relevant parts and non-legally relevant ones.

The scheduler implemented must ensure fixed runtimes for every VM to ensure worst-case response times and to minimize the potential for denial-of-service attacks. A measuring instrument is a real-time system, meaning that, if within a maximum time frame, measuring tasks are not completed, failure has occurred. Therefore, absolute worst-case execution times (WCET) for the VMs must be guaranteed. An advantage of our system and of embedded systems in general is their static behavior. The amount of needed VMs remains constant over the whole execution time; therefore, a static schedule can be declared from the beginning. A temporal partition schedule should be applied [21], assuring that VMs do not starve, *i.e.*, do not get execution time. Each VM is provided a window of execution within the repeating timeline.

In our framework, some VMs, e.g., the Connection Manager that is responsible for redirecting interrupts, can be split into more than one window of execution. In this way, the system can react faster to input devices. Other VMs that need to execute their calculation as fast as possible, e.g., the L VM, could get a bigger window of execution than the rest of the VMs or could run with fewer VMs on a separate core in a multicore system, as can be seen in Section 7.

4.2. Distributed System View

As already mentioned, we have built a virtual distributed system in which the VMs communicate through a virtual network. The microkernel/hypervisor ensures that the network is reliable; hence, for the transport layer protocol, we can use the User Datagram Protocol (UDP). The virtual network is shown in Figure 5.

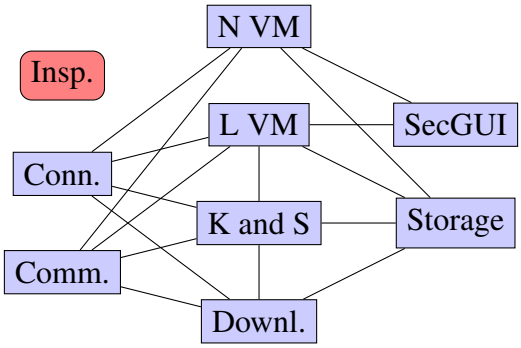

Figure 5. Connected VMs: Inspector is monitoring the VM.

For security reasons, the network layer protocol used when a VM communicates with the Connection Manager should be encrypted, e.g., Internet Protocol Securiy (IPsec) in Transport Mode. To encrypt the

packets, IPsec uses the mechanism Encapsulating Security Payload (ESP), encrypting the payload by the symmetric encryption algorithm like Advanced Encryption Standard in Cipher Block Chaining mode (AES-CBC) with the keys, that are being managed by the Key and Signature Manager. By encrypting the transmission with a key unknown to the Connection Manager, we make sure that if it is compromised, e.g., because it is accessible by the real network, it is not able to modify messages that are unnoticed. Communication between other VMs is not that critical and, therefore, does not need encryption.

For the application layer, a protocol must be defined that encapsulates the commands and data. Thereby, every VM could use its own protocol or interpretation of payload, e.g., the Storage Manager accepts requests, like storing data to disk or getting data from disk, which other VMs do not need. Hence, every VM needs to know the structure of the protocols, which other VMs use, if they want to communicate with each other.

Each VM has a server application that listens to a predefined port to receive and afterwards respond to the queries. In our architecture, the VMs fulfill client and server duties, because they redirect tasks to and process tasks for other VMs, which makes them so-called servants. The VMs can be divided into three core layers: the user interface layer, the processing layer and the data layer. Tasks for the user interface are executed by the Communication Manager that redirects I/O from and to devices, the Secure GUI, which displays the graphical user interface (GUI), and the Connection Manager communicating to the outer world. The processing is done by the L/N VMs, the Inspector and the Download Manager. Finally the data are managed by the Key and Signature Manager and the Storage Manager. If the scope is to construct a real distributed system, the only modules needed on the measuring instrument are the user interface layer VMs and the Inspector, which does integrity checking on the other VMs (described in detail in Section 6); all of the other modules can be outsourced to other machines. When using an open network, reliability is not ensured. Therefore the cryptographic protocol Transport Layer Security (TLS) should be used as the transport layer protocol, which the Connection Manager should enforce for network communication.

5. Description of the Modules

By dividing the framework into modules, it can be tailored individually for every measuring instrument. For example, if a measuring instrument does not need a download mechanism, the Download Manager should be removed, and if it does not need network access, the Connection Manager is unnecessary. Besides the L VM, the modules every measuring instrument needs are the Key and Signature Manager, the Storage Manager, the Communication Monitor and the Inspector. A closer look at the individual modules is given below.

5.1. Key and Signature Manager

The Key and Signature Manager is responsible for assuring confidentiality and integrity by managing the public keys of the VMs, which want to communicate to the outside world over the Connection Manager. The Key and Signature Manager serves as a certification authority (CA), assigning and dispensing public keys to the corresponding VMs, which are, in turn, used to negotiate a symmetric key. The manager should have, as every module has, exclusive rights to a portion of the storage device

to hold sensitive information. Every VM has its own key database holding the symmetric keys. If a public key gets changed, the manager informs the other VMs to renegotiate a symmetric key. If a key is compromised, the certificate can be invalidated, and the respective VM can be prompted to regenerate a key pair and to transmit the new public key to the manager. Only an authorized entity should be able to command the Key and Signature Manager in this way. Even a sealed hardware switch that needs to be broken to allow the reassigning of keys could be a possible solution.

Furthermore, the manager incurs the protection of legally relevant data by holding the keys for file system data encryption and the hash values of the Software IDs for integrity checks at boot and runtime.

5.2. Connection Manager

The Connection Manager is the only VM with physical network access. All data transmitted from and into the network goes through this module. Hence, the Connection Manager is critical from a security point of view. The manager is a firewall for the system, analyzing received data and, according to its rules, redirecting the packets to the appropriate VM. For this purpose, a well-defined protocol must be established. If a packet does not conform to the protocol or the firewall rules, it is discarded.

Another one of its duties is the encryption of legally relevant data in transit, by building up a VPN to only trusted end-points. Data coming from other VMs is itself already encrypted by the symmetric keys that the individual VMs have pre-negotiated with the end-points and cannot be read out by the Connection Manager. It can only be redirected. In this way, a compromised Connection Manager is not able to modify data that are undetected. Non-legally relevant data can be sent unencrypted, but firewall rules for outgoing packets should be obeyed.

5.3. Inspector

A measuring instrument shall be designed to allow surveillance control by software after the instrument has been placed on the market and put into use. Hereby, software identification shall be easily provided by the measuring instrument. To achieve these requirements, the Inspector is the monitoring VM that can be advised by authorized entities to show the unique ID of the device. The Connection Manager (or Communication Monitor) redirects the requests after checking if an authorized person is connected to the device.

The Inspector itself automatically checks the system after a specified time interval and after failure has occurred. The Inspector module can advise the Storage Manager to check the file-system and the individual measurements for integrity, to transmit and check the identifications of all modules, to advise the Storage Manager to print out the logging, to check for enough storage capacity and, if needed, to check the other VMs for malware. The Inspector receives a snapshot of every VM from the hypervisor to do this; a detailed description is given in Section 6.

The Inspector is also an autonomous watchdog timer that ensures correctness in the event of a delay that could harm the measurement. In extreme cases, it even deletes or marks measurements as invalid or shuts the system down. If a VM is not responding, the Inspector can restart it. For restarting VMs, shutting down the system and getting snapshots from the hypervisor, the Inspector needs special access rights that no other VM has.

5.4. Download Manager

In legal metrology, legally relevant software can only be updated if the software update was checked prior to download on the device and afterwards by breaking a seal. Downloads for non-legally relevant parts are allowed without new checking. In our system architecture, this is no problem, due to the strict isolation. Before legally relevant software is updated, the Download Manager checks if the sealed hardware switch for downloading legally relevant software is set. If this hardware switch is set, measuring must be disabled. For non-legally relevant updates, it only must be ensured that the computational time that the download mechanism needs does not disturb correct measuring. Therefore, the Download Manager should get a minimum running time, if a non-legally relevant software download is performed.

An upload can take place through different interfaces. If the download is started through the Ethernet interface, the Connection Manager receives the request and redirects the download to the partition of the Download Manager through the Storage Manager. If another interface is used, e.g., USB, the data goes through the Communication Monitor. After the download is finished, the respective module informs the Download Manager that checks the update on its partition for authenticity and integrity through hash values. If everything is correct, the Download Manager advises the Storage Manager to copy the update to the legally relevant or non-legally relevant partition, respectively. Afterwards, the Download Manager reports the download to the Storage Manager for the audit trail and advises the Inspector to restart the legal and/or non-legal VM, respectively.

5.5. Communication Monitor

The Communication Monitor supervises the queries from and to the I/O devices. This module ensures that user input and software cannot influence measurement data in an unwanted way, that peripheral devices can only be accessed by legally relevant software and that the transmission of data from the legally relevant VM to the non-legally relevant one is licit. If an input device is allowed to send data to the non-legally relevant VM, a switch must be set as described in Section 4.1.

This monitor serves as a kind of firewall for internal communication of the legally relevant VM, blocking packets that do not conform to well-defined rules and checking that confidential data is not transmitted to the non-legally relevant VM. The communication monitor is the only VM that has direct access to the peripheral devices besides the physical network card (accessible by the Connection Manager), the display (accessible by the Secure GUI) and the storage device (accessible by the Storage Manager). Other modules can just communicate with each other through their virtual network cards. An interrupt from an input device is first directed to the driver in the Communication Monitor, which, in turn, translates it to a network package and sends it to the legally relevant VM.

5.6. Secure GUI

Non-legally relevant output must be unambiguously marked to distinguish it from the legal one. The Secure GUI supervises this output to the screen and is the only VM that can write directly to the screen. One possible way is to define selected parts of the screen to the legally relevant software that cannot be

changed by the non-legally relevant VM. Another solution is to change the running mode via a hardware switch. Hence, when the switch is set, non-legally relevant software cannot write to the screen. In either case, the Secure GUI module conducts the buffering for both VMs. For that purpose, the legally relevant and the non-legally relevant VMs could each have their own virtual video card driver that redirects requests to the Secure GUI, which, in turn, visibly separates the output for the user. Another solution is to communicate the screen output through the virtual network cards, by using a well-defined protocol to winnow the screen output data from other message data. By using this solution, the Secure GUI communicates the same way with the VMs as every other module, and no extra driver must be written, respecting the minimality principle.

5.7. Storage Manager

The Storage Manager is the only VM with access to the storage device. Every other module that wants access to its storage partition must send the read and write commands through network packets. The Storage Manager assures strict isolation of the individual module's stored data in use. It checks the module's file permissions, making sure that no illicit manipulation of data takes place. It is also responsible for the encryption of data-at-rest, making the file-system data unreadable if the key is unknown. In the case of errors, the Storage Manager informs the Logger and, in extreme cases, e.g., no storage capacity and nothing can be deleted, the Inspector to shut down the system.

Additionally, The Storage Manager is responsible for tracking interventions. The other modules inform the Storage Manager about interventions and errors by sending it a network packet. This message is then concentrated to an aligned format and saved in a log-file for the audit trail. On demand, the Storage Manager sends its log-file to the L VM, which, in turn, can send it to the Secure GUI, the Connection Manager or the Communication Manager to transmit it to other devices.

6. System Integrity Checking

In legal metrology, measurement devices need to be able to prove the integrity to a remote party, e.g., the control agent or the consumer, which is called remote attestation. The problem that arises with general purpose operating systems is that they are complex programs with millions of SLOC. Therefore, building high-assurance applications on these systems seems impossible, because they depend on the OS as part of their trusted computing base (TCB). In our system, we try to keep the TCB as small as possible by using a small hypervisor as the only underlying program that runs in privileged mode. This hypervisor enforces isolation: every VM needs to be securely isolated to provide confidentiality and integrity. Because the reliability of the applications still depends on the guest operating system, the VMs as a whole need to be integrity checked to make remote attestation possible. Furthermore, we need a trusted path, which ensures that the remote party knows that it communicates with an untampered VM and also makes sure that the VM notices if it communicates with malicious software. Hence, the trusted path ensures the integrity and privacy of the communication.

Attestation authenticates what software was started at each layer of the software stack, from the firmware up to the VM. For this purpose, a certificate chain is needed. It is being built from tamper-resistant hardware, which stores the embedded cryptographic key signed by the vendor to the

application. The tamper-resistant chip certifies the firmware, and the firmware certifies the boot loader. Afterwards, the boot loader certifies the VMM, which, in turn, certifies the VMs. The applications get their certificates by fist generating a private/public key pair and then by calling the VMM with a special call consigning the public key. The VMM generates and signs the certificate containing a hash of the attestable component and its public key with application data. This certificate binds the public key to a component whose hash is given in the certificate. In this attestation scenario, integrity checking is only done at boot time, and no runtime protection is given.

Runtime integrity measurement requires more attention. First of all, the continuity of the integrity beyond the measurement time should be guaranteed. An attacker could, for example, change a program after it is measured and before it is executed. This problem is known as time of check to time of use (TOCTTOU) consistency. To overcome this problem, the applications need to be actively monitored by intercepting critical system calls and by guest memory protection. To intercept system calls, interrupts and exceptions, the VMM needs to be modified and enhanced by integrity measurement functions that calculate the hash of the effected memory area. In our architecture, this hash combined with its ID, e.g., file name and VM, is then transmitted to the Inspector VM. The Inspector keeps track of the memory layout of all of the processes running in the other VMs, and it stores the hash image of each of the pages that form a measured memory area. With this information, the Inspector can compare the correct hash values in the database with the ones that will be loaded into memory. If they do not match, one can assume that the process is not trustworthy. Until the Inspector has not finished its comparisons, the inspected VM is not allowed to execute or modify the memory area to guarantee TOCTTOU. This can be achieved by hardware support. For example, in HIMA (Hypervisor-based Integrity Measurement Agent) [22], this is done by the No-eXecute (NX) bit page protection flag, which is available on most hardware platforms. Executing an instruction from a page that is protected by the NX flag will cause an exception that is trapped in the hypervisor. After a page was measured and verified by the Inspector, the hypervisor marks this page as executable, but not writable.

7. Experimental Evaluation

To show the feasibility of our approach, we have started to build a system atop an L4-microkernel. In our opinion, the L4-microkernel family is a good choice, because it is widely used and consists of third generation microkernels. One of these microkernels, called seL4, is even fully verified [23], inferring that classical security threats against operating systems, like buffer overflows, null pointer de-referencing, arithmetic overflows, arithmetic exceptions, pointer errors and memory leaks, are not possible. Another positive aspect of many L4-microkernels is that a binary compatible para-virtualized Linux is available.

For our demonstrator, we used L4Linux running on top of the open source Fiasco.OC L4-microkernel, which yields good results even for real-time applications [24]. Our test board was the PandaBoard Revision B3 equipped with the OMAP4460 SoC running a dual-core 1.2 GHz ARM Cortex-A9 MPCore with 1 GiB of DDR2 SDRAM.

We compiled Fiasco.OC with multicore support and used the open source tool Buildroot to generate our eleven RAM disks for the VMs. First, we compiled the L4Linux with a virtual network driver and simulated cross-over cable connections between the VMs that need to communicate with each

other, adding a virtual network interface controller (NIC) for every point-to-point connection to the VM. Hereby, 15 connections were established, which are shown in Figure 5 (Section 4.2). Afterwards, we included the Dropbear SSH server in the RAM disks, to remotely connect from one VM to another for test purposes (Allowing to remotely connect to one VM through another is a security risk and should not be permitted in a real system).

We start our VMs with a predefined memory and RAM disk size and assign the nine VMs to one of the two cores, with five VMs running on the first core and four on the second one, as also shown in Figure 5. The VMs are all scheduled with the round robin algorithm, where each VM has the same priority and a 21 microsecond time slice. The configurations are shown in Table 2.

Table 2. Configuration of the individual VMs.

VM	Core	Memory	RAM Disk Size
Key & Signature	1	30 MB	5 MB
Inspector	1	30 MB	5 MB
Storage.	1	110 MB	40 MB
Download.	1	30 MB	5 MB
Non-legal VM	1	30 MB	5 MB
Communication	2	100 MB	40 MB
Legal VM	2	300 MB	65 MB
Connection	2	30 MB	5 MB
Secure GUI	2	30 MB	5 MB
Σ	2	690 MB	175 MB

We focused our test on the transmission of data between three VMs, the Communication Monitor, the legally relevant VM (L VM) and the Storage Manager. For the remaining VMs, we simulated a constant workload by an auto-start shell script, which traversed the filesystem in an infinite while loop, adding files and afterwards deleting them. In this way, no VM became idle. In total, we tried to emulate a speed camera where the Communication Manager receives the pictures from the camera. In our demonstrator, we put seven different grayscale pictures in PNG format on the RAM disk of the Communication Manager. These were transmitted to the L VM with the Netcat program using a TCP connection. As the sizes of the pictures were scattered and too small for meaningful tests (from 100 kB to 4.6 MB), we additionally created dump files between 100 kB and 20 MB to get wider test results. Table 3 shows their sizes and the measured transmission times of the file transfers between two VMs (every test was repeated 10 times; the times shown in the tables in this section are the averages of these 10 repetitions).

Table 3. Measured times for TCP connections using Netcat.

File Size	Time
10 kB	0.289 s
100 kB	1.026 s
500 kB	1.358 s
1 MB	1.724 s
5 MB	2.585 s
10 MB	3.707 s
15 MB	4.913 s
20 MB	6.307 s

After the L VM has received the pictures, it converts the images with the program ImageMagick (convert command) by adding a label with a black background of 60 pixels in height on top of the pictures, which in national German law is postulated to show the velocity and other information corresponding to the picture. In another test scenario, the pictures are scaled to a quarter of their original resolution with auxiliary programs from the JPEG library (djpeg and cjpeg). The measured times for these operations are shown in Table 4. Additionally, we compiled a normal Linux kernel (Version 3.14, which corresponds to the version of L4Linux) for the PandaBoard, which loads the L VM RAM disk. Afterwards, the same convert commands were executed on the images to compare the times, also shown in Table 4.

Table 4. Comparison of image processing times on normal Linux and L4Linux in our architecture.

	L4Linux		Linux	
Resolution	**Convert**	**djpeg**	**Convert**	**djpeg**
1216 × 817 (1 MP)	4.034 s	0.774 s	1.562 s	0.138 s
1920 × 1080 (2 MP)	6.761 s	0.631 s	2.638 s	0.108 s
2580 × 1600 (4 MP)	13.955 s	1.567 s	5.776 s	0.480 s
2832 × 2236 (6 MP)	18.115 s	1.795 s	7.422 s	0.550 s
3264 × 2448 (8 MP)	30.814 s	2.061 s	10.656 s	0.615 s
3835 × 2855 (10 MP)	45.696 s	1.804 s	15.324 s	0.523 s

Lastly, the converted image is transmitted to the Storage Manager, where the data are encrypted by the OpenSSL program with AES 256-bit CBC. The encryption times are shown in Table 5 with the corresponding times needed on the normal Linux kernel.

Table 5. Comparison of OpenSSL execution times.

Size	L4Linux	Linux
712 kB	0.550 s	0.103 s
904 kB	0.671 s	0.107 s
4.3 MB	1.303 s	0.359 s
4.5 MB	1.583 s	0.379 s
5.5 MB	1.782 s	0.461 s
6 MB	1.967 s	0.517 s

To combine all of the steps into one, we measured the time of the transmission and converting of a 10-MP image, which first was sent from the Communication Monitor to the L VM. Here, it was scaled down by a factor of sixteen and afterwards edited with ImageMagick and sent to the Storage Manager. Finally, the Storage Manager used OpenSSL to decode the picture. All of these steps were finished in around 12 s. Normal Linux achieved the same result in around 2 s, omitting the Netcat transfers, which were not necessary.

In our opinion, the slower processing time is justifiable considering the security gain. Furthermore, the test results seem to be promising, because it must be considered that eleven VMs run in parallel with a constant workload, whereas in the normal Linux architecture, just one operating system runs directly on the hardware, which leads to a relatively small speed gain of about 600%. It should be also noted that all of the tests were achieved by using Linux shell scripts (bash), which invoked standard Linux processes.

8. Analyzing the System

We analyze the system by showing how PHASE is followed and that this way, the policies that should be implemented according to MILS, to be specific, data isolation, information flow control, damage limitation and periods processing, are upheld. Afterwards, the basic framework is evaluated by time measurements.

8.1. Data Isolation

Data isolation is generally enforced by the component architecture principle and by using a theoretical verified separation kernel as our VMM. Hence, data isolation between the VMs can be presumed. As we also enforce the least privileged principle, our untrusted VMs running the variable software, *i.e.*, the L and N VMs (see Section 4), have no direct access to the outside world. Their software must be checked in an independent expert validation, as done by the notified bodies, to ensure data isolation inside the VMs. Data-at-rest security is managed by the Storage Manager, which is responsible for the isolation and encryption of data on the storage device.

8.2. Information Flow Control

Every VM is provided with a virtual network interface for communication. The separation kernel ensures that no other communication is possible. Information from the outside world, *i.e.*, from

peripheral devices and the network, go through the Communication Monitor and the Connection Manager, respectively. These VMs, in turn, communicate with the other VMs after carefully checking conformity to a well-defined protocol, as mentioned in Section 4.2, and checking access permissions enforcing the least privileged principle. Through unique network card numbers (MAC address) and predefined static connections, the communication partners are known. Additionally, sensitive data can be encrypted to protect the confidentiality and integrity of the communication.

8.3. Damage Limitation

The directly exposed VMs are the Communication Monitor and the Connection Manager, because they are the only VMs accessible through peripherals from which attacks could be mounted (NIC, USB, SD, *etc*.). These, in turn, have no direct access to the storage device and no access to measurement data, as the least privileged principle is applied. To prevent damage, the minimal implementation principle must be followed; hence, the modules, which are not just processes, but VMs with GPOSs, must have minimal configurations. A verified network stack and network interface card driver would drastically reduce the attack vectors.

As described in Section 3.2, virtualization adheres to the component architecture principle, limiting the damage to the respective VM in which it occurs. Additionally, independent expert validation by the notified bodies ensures damage limitation in the legally relevant VM.

Another measure taken is monitoring performed by the Inspector. If a VM does not conform to its predefined rules, the Inspector notices the misbehavior and takes action, e.g., reloading and restarting the module. In the worst case, the Inspector can shut down the system.

8.4. Periods Processing

The separation kernel must ensure that no hidden channels, through which information could leak out, are present. These could arise due to scheduling, because the VMs run consecutively, often using the same resources, e.g., shared caches.

Another way of getting secret information is to exploit variations for covert channels or side channel attacks. In general, side channel attacks benefit from variations, e.g., in timing, power consumption, electromagnetic emanation and temperature, to gain information of a cryptosystem. A high-awareness security system should be tested for side channel attacks and should take sophisticated attacks into account, e.g., cache-based side channel analysis [25]. A covert channel exploits the same variations as a side channel attack, but is used by malicious processes to exchange information. For example, a VM could try to reduce or extend its execution time, which then can be noticed by the ensuing VM and analyzed. Hereby, a reduced execution time could stand for a zero and an extended one for a one.

Most of the counter-measures are taken by the separation kernel or can be taken care of by special hardware. Covert channels that arise due to the scheduling policy must be considered separately. As mentioned in Section 4.1, a partition schedule should be employed to eradicate timing variations in the scheduling process, because every VM has its fixed running window, which cannot be released. To ensure that a VM cannot delay the beginning of the execution time of the ensuing VM, e.g., by a system call before the end of its execution window, a time buffer is needed between the VMs.

9. Related Work

For measuring instruments under legal control, there are papers that point out the risk general purpose operating systems have on these devices, e.g., [26]. Other documents talk more generally about Linux system configurations for general-purpose computers, e.g., the Guide to the Secure Configuration of Red Hat Enterprise Linux 5 [27]. These documents try to secure the system within the system by restricting the user, e.g., restricted access rights. Vulnerabilities that lie in the operating system or in user programs can be used to subvert these settings, making these protections too weak.

There are some papers that stress that the microkernel approach for virtualization is recommended to construct secure systems [12–17]. Their approaches are similar; they try to create a secure system by modularizing the system architecture, but of course, lack the concrete analysis of legal metrology aspects, because they talk about general purpose systems. NOVA (short for NOVA OS Virtualization Architecture) [28], for example, describes a microhypervisor architecture, which is written with only around 40K source lines of code, which is a magnitude smaller than popular virtualization architectures.

A modular system architectures for general purpose systems that is similar to ours is Terra [29]. It is based on virtualization by dividing critical parts of software from non-critical ones through virtual machines. Here, software-based attestation is realized by building a certification chain that begins at the hardware with its private key embedded in a tamper-resistant chip. Afterwards, the hardware certifies the firmware, which, in turn, certifies the system boot loader, which certifies the virtual machine monitor (VMM). The VMM at last certifies the virtual machines. The individual application can then be certified by constructing a private/public key pair and advising the VMM by a special call to sign their public key. Hence, Terra provides integrity checking only at boot time and no runtime protection. A better approach is given by [22], called HIMA (Hypervisor-based Integrity Measurement Agent), which can be also combined with Terra; a similar approach is described in Section 6. In HIMA, "out-of-the-box" measurement is done by active monitoring of critical guest events (e.g., creation of a process) and guest memory protection (by using hardware support, e.g., NX-bit).

In the measuring instruments domain, there are some papers that talk more specifically about smart meter devices and point out that software-based attestation is critical [30,31]. To our knowledge, most of the research done in software security for legal metrology concerns smart grids (e.g., also [32] and [33]). Here, threats, like network attacks, that exist for these devices are analyzed to point out challenges and solutions.

With our approach, a general and detailed solution for a secure system architecture is given. A bridge between academia and industry is created that makes sure that all of the requirements from European directives for measuring instruments are upheld by using secure and sound techniques. We tailor our architecture to fulfil legal requirements for measuring instruments in Europe and keep system integrity checking in mind.

10. Conclusions

In this paper, we presented a framework for a new secure system architecture for measuring instruments in legal metrology. We constructed our architecture by analyzing the requirements for

measuring instruments demanded in the MID and the WELMEC 7.2 Software Guide, combined with methodologies and concepts from high-assurance software systems, *i.e.*, MILS and PHASE. To harness device drivers and network stacks of general purpose operating systems, we came to the conclusion that virtualization through a small microkernel is the right solution to combine security with usability.

We took a three-pronged approach. Firstly, we separated the legally relevant parts from the irrelevant ones by putting them in different virtual machines. Secondly, we made sure that their virtual machines have no direct access to I/O devices. Lastly, we constructed a secure framework, which provides services to these VMs. This framework, also consisting of separated VMs, monitors the information flow, correctly delegates requests from and to I/O devices and helps control agencies to verify system integrity.

Furthermore, the framework was evaluated by building a system atop an L4-microkernel. For our demonstrator (PandaBoard Revision B3 equipped with the OMAP4460 SoC running a dual-core 1.2 GHz ARM Cortex-A9 MPCore with 1 GiB of DDR2 SDRAM), we used L4Linux running atop the open source Fiasco.OC L4-microkernel, which yields good results, even for real-time applications. These tests show that it is practically feasible to construct a configurable framework using our architecture, which is applicable for powerful measuring instruments under legal control.

Author Contributions

Florian Thiel and Jean-Pierre Seifert saw the challenges in security for measuring instruments under legal control. They formulated the initial problematic and contributed to this article by their expertise in legal metrology and software system security, respectively. Daniel Peters contributed to the initial design of the architecture, implemented the system architecture on the demonstrator, analyzed the data and contributed to the writing of this article. Michael Peter contributed with his expertise on microkernels and virtualization to the initial design and the implementation of the architecture. All authors have read and approved the final manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Ebert, C.; Jones, C. Embedded Software: Facts, Figures, and Future. *IEEE Comput.* **2009**, *42*, 42–52.
2. Leffler, N.; Thiel, F. *Im Geschäftsverkehr das richtige Maß—Das neue Mess und Eichgesetz, Schlaglichter der Wirtschaftspolitik, November 2013, Monatsbericht*; Bundesministerium für Wirtschaft und Technologie (BMWi): Berlin, Germany, 2013. (In German)
3. Chelf, B. *Measuring Software Quality—A Study of Open Source Software*; Coverity: San Francisco, CA, USA, 2011.
4. OIML D 31. *General Requirements for Software Controlled Measuring Instruments*; Bureau International de Métrologie Légale: Paris, France, 2008.
5. Official Journal of the European Union. *Directive 2014/32/EU of the European Parliament and of the Council*; European Commission: Brussels, Belgium, 2014.

6. Official Journal of the European Union. *Directive 2004/22/EC of the European Parliament and of the Council*; European Commission: Brussels, Belgium, 2004.
7. *WELMEC 7.2, Issue 5, Software Guide*; WELMEC Secretariat: JA Delft, The Netherlands, 2012.
8. Lackorzynski, A.; Warg, A.; Peter, M. Virtual Processors as Kernel Interface. In Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi, Kenya, 25–27 October 2010.
9. Adams, K.; Agesen, O. A Comparison of Software and Hardware Techniques for x86 Virtualization. *SIGARCH Comput. Archit. News* **2006**, *34*, 2–13.
10. Frenzel, T.; Lackorzynski, A.; Warg, A.; Härtig, H. ARM TrustZone as a Virtualization Technique in Embedded Systems. In Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi, Kenya, 25–27 October 2010.
11. Wojtczuk, R. Subverting the Xen hypervisor. In *Black Hat USA*; Invisible Things Lab: Las Vegas, NV, USA, 2008.
12. Liebergeld, A.S.; Peter, M.; Lackorzynski, A. Towards Modular Security-Conscious Virtual Machines. In Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi, Kenya, 25–27 October 2010.
13. Lange, M.; Liebergeld, S.; Lackorzynski, A.; Warg, A.; Peter, M. L4Android: A Generic Operating System Framework for Secure Smartphones. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Chicago, IL, USA, 17–21 October 2011; pp. 39–50.
14. Heiser, G.; Uhlig, V.; LeVasseur, J. Are Virtual-machine Monitors Microkernels Done Right? *SIGOPS Oper. Syst. Rev.* **2006**, *40*, doi:10.1145/1113361.1113363.
15. Hohmuth, M.; Peter, M.; Härtig, H.; Shapiro, J.S. Reducing TCB Size by Using Untrusted Components: Small Kernels Versus Virtual-machine Monitors. In Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, Leuven, Belgium, 19–22 September 2004.
16. Heiser, G. The Role of Virtualization in Embedded Systems. In Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems, Glasgow, UK, 31 March–4 April 2008; pp. 11–16.
17. Peter, M.; Schild, H.; Lackorzynski, A.; Warg, A. Virtual Machines Jailed: Virtualization in Systems with Small Trusted Computing Bases. In Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems, Nuremberg, Germany, 31 March 2009.
18. Beckwith, R.W.; Vanfleet, W.M.; MacLaren, L. High Assurance Security/Safety for Deeply Embedded, Real-time Systems. In Proceedings of the Embedded Systems Conference, San Francisco, CA, USA, 29 March–1 April 2004; pp. 247–267.
19. O'Dowd, D. RTC Magazine: Green Hills Software. Available online: http://www.rtcmagazine.com/articles/view/101494 (accessed on 29 March 2014).
20. Härtig, H.; Loeser, J.; Mehnert, F.; Reuther, L.; Pohlack, M.; Warg, A. An I/O Architecture for Mikrokernel-Based Operating Systems; In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004), San Francisco, CA, USA, 6–8 December 2004.
21. Kerstan, T.; Baldin, D.; Groesbrink, S. Full virtualization of real-time systems by temporal partitioning. In Proceedings of the Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Brussels, Belgium, 7–9 July 2010.

22. Azab, A.M.; Ning, P.; Sezer, E.C.; Zhang, X. HIMA: A Hypervisor-Based Integrity Measurement Agent. In Proceedings of the 2009 Annual Computer Security Applications Conference, Honolulu, HI, USA, 7–11 December 2009; pp. 461–470.
23. Klein, G.; Elphinstone, K.; Heiser, G.; Andronick, J.; Cock, D.; Derrin, P.; Elkaduwe, D.; Engelhardt, K.; Kolanski, R.; Norrish, M.; *et al*. seL4: Formal Verification of an OS Kernel. In Proceedings of the 22nd ACM Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009.
24. Lackorzynski, A.; Danisevskis, J.; Nordholz, J.; Peter, M. Real-Time Performance of L4Linux. In Proceedings of the Thirteenth Real-Time Linux Workshop, Prague, Czech, 20–22 October 2011.
25. Neve, M.; Seifert, J.-P. Advances on Access-driven Cache Attacks on AES. In Proceedings of the 13th International Conference on Selected Areas in Cryptography, Montreal, QC, Canada, 17–18 August 2006; pp. 147–162.
26. Thiel, F.; Grottker, U.; Richter, D. The challenge for legal metrology of operating systems embedded in measuring instruments. *OIML Bull.* **2011**, *52*, 5–14.
27. Guide to the Secure Configuration of Red Hat Enterprise Linux 5. Available online: https://www.nsa.gov/ia/_files/os/redhat/rhel5-guide-i731.pdf (accessed on 27 December 2014).
28. Steinberg, U.; Kauer, B. NOVA: A Microhypervisor-based Secure Virtualization Architecture. In Proceedings of the 5th European Conference on Computer Systems (EuroSys '10), Paris, France, 13–16 April 2010; pp. 209–222,
29. Garfinkel, T.; Pfaff, B.; Chow, J.; Rosenblum, M.; Boneh, D. Terra: A Virtual Machine-based Platform for Trusted Computing. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003; pp. 193–206.
30. LeMay, M.; Gunter, C.A. Cumulative Attestation Kernels for Embedded Systems. *IEEE Trans. Smart Grid* **2012**, *3*, 744–760.
31. Doeornemann, K.; von Gernler, A. Cybergateways for Securing Critical Infrastructures. In Proceedings of International ETG-Congress 2013, Symposium 1: Security in Critical Infrastructures Today, Berlin, Germany, 5–6 November 2013.
32. Boccardo, D.R.; da Costa Carmo, L.F.R.; Dezan, M.H.; Machado, R.C.S.; de Aguiar Portugal, S. Software evaluation of smart meters within a Legal Metrology perspective: A Brazilian case. In Proceedings of the Innovative Smart Grid Technologies Conference Europe (ISGT Europe), Gothenburg, Sweden, 11–13 October 2010; pp. 1–7.
33. Do Prado, C.B.; Boccardo, D.R.; Machado, R.C.S.; da Costa Carmo, L.F.R.; do Nascimento, T.M.; Bento, L.M.S.; Costa, R.O.; de Castro, C.G.; Camara, S.M.; Pirmez, L.; *et al*. Software Analysis and Protection for Smart Metering. *NCSLI Meas.* **2014**, *9*, 22–29.