DOI: 10.15439/2014F460
ACSIS, Vol. 3

Achieving Software Security for Measuring Instruments under Legal Control

Daniel Peters, Ulrich Grottker, Florian Thiel
Physikalisch-Technische Bundesanstalt,
Germany
Email: {daniel.peters, ulrich.grottker, florian.thiel}@ptb.de

Michael Peter, Jean-Pierre Seifert
Security in Telecommunications,
Technische Universität Berlin,
Germany
Email: {peter, jpseifert}@sec.t-labs.tu-berlin.de

***Abstract*—In recent years measuring instruments have adopted general-purpose operating systems to offer the user a broader functionality that is not necessarily restricted towards measurement alone. Additionally the trend to the internet of things from which measuring instruments are not immune, e.g. smart meters and traffic enforcement cameras just to name a few, brings forth security questions.**

In this paper, a flexible software system architecture that can be constructed out of freely available open source software is presented which addresses these challenges within the framework of essential requirements laid down in the Measuring Instruments Directive of the European Union. The system architecture is based on a modular design assuring correct collaboration between modules by encapsulating them in different virtual machines and supervising their communication.

I. Introduction

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 [16], e.g. electricity meters, gas meters, etc. and their related measurements. In Germany alone, this corresponds to an amount of 104 to 157 billion Euros each year [16]. Hence, manipulations of measuring instruments' software could have far-reaching financial consequences. Clearly, special measures should be considered to secure such instruments.

In the light that around 98% of the world's computer systems are embedded devices [4], 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.

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 2 000 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[1], 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 fuelled. Hence in Europe, member states denominate institutions, called *notified bodies*, which are responsible to review 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, 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. This assumption is too optimistic, considering that just one open interface, e.g. an USB-port or WLAN, can allow a sophisticated attacker to run arbitrary code by exploiting a single vulnerability.

The remainder of the paper is organized as follows. Section II provides an introductory part about legal metrology. Section III describes how virtualization can help to construct a suitably secure embedded system for measuring instruments. Section IV together with Section V describe our framework before it is analysed in Section VI. In Section VII, we give a conclusion and describe further work to be done for the final implementation.

II. Legal Metrology

Legal metrology comprises measuring instruments that are employed for commercial or administrative purposes or for

[1]Catalogue 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

measurements which are of public interest. More than 100 million legally relevant meters are in use in Germany [16]. 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 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 does 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 harmonising 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 [21].

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.

A. Measuring Instruments Directive

Directive 2014/32/EU of the European Parliament and of the Council [20], which is based on Directive 2004/22/EC [19], 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.

B. 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 concerns 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 [24], 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.

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 relevant* that 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* of received or stored data.

III. 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 perspective, 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[2] shown in Figure 1.

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 -

[2]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.

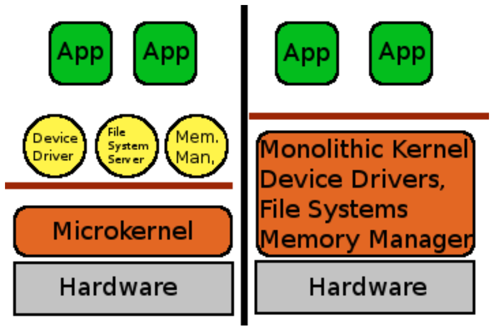

Fig. 1. Comparison between a microkernel (left) and a monolitic kernel design (right)

in Figure 1 the privileged mode is under the horizontal lines. 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 parts of a system (software and hardware) which are needed 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 yellow circles. 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 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. 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. Virtualization can be divided into two main approaches [13]. *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 [1]. 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 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 [5] 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.

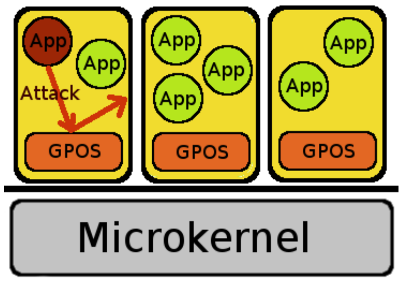

Fig. 2. Left VM's general-purpose operating system (GPOS) is compromised by a pernicious application. Due to isolation the other VMs are not vulnerable

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 *virtual machine monitor (VMM)* or *hypervisor*[3], is correctly implemented. A microkernel, because of its minimality principle, seems to be a good choice for implementing a hypervisor [9, 17, 15, 10, 8, 22].

B. 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

[3]In this paper we do not differentiate between a VMM and a hypervisor as sometimes done. Both refer to the underlying (micro-)kernel.

- implementing a limited set of critical functional security policies. This special kernel, often called *separation kernel* or *partition kernel*, implements the policies for *information flow control*, *data isolation*, *damage limitation*, and *periods processing* [2].

- *Information flow control* ensures that information cannot flow between partitions unless explicitly permitted by the system security policy
- *Data isolation* ensures 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
- *Periods processing* ensures that information from one component is not leaked into another one through resources, which may be reused across execution periods

As stated in Section III, modern monolithic kernels and 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 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.

IV. Our Framework

Our proposed framework, shown in Figure 3, consists of three parts. The big block on the right is the VM for the non-legally relevant software (N) and the block next to it is the VM for the legally relevant software (L). In the L VM all the computations that are needed for the measurement procedure are executed, e.g. image processing in a traffic enforcement camera. The 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. On the left hand side the smaller 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 [6, 10]. 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.

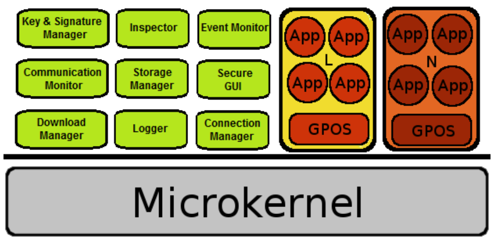

Fig. 3. Framework

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

- Displaying data: *Secure GUI*
- Protecting data: *Key & Signature Manager*
- Saving data: *Storage Manager*
- Identifying the software: *Inspector*
- Executing downloads: *Download Manager*
- Transferring data over network: *Connection Manager*
- Recording modifications: *Logger*

The *Communication Monitor* redirects queries from and to the I/O devices, e.g. sensors and keyboard. Finally the *Event Monitor* is a watchdog, running availability checks of the software to dynamically spot manipulations.

A. 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, the kernel can 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/writeable for the other system components. The first check then starts at the boot process where, for example, the hash value, e.g. 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, that 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 behaviour. 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 [11], 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, could be split into more than one window of execution. In this way, the system can react faster to input devices.

B. Distributed System View

As already mentioned, we have built a virtual distributed system in which the VMs communicate through a virtual network. The microkernel ensures that the network is reliable, hence for the transport layer protocol we can use *UDP*. For security reasons the network layer protocol used when a VM communicates with the Connection Manager should be *IPsec* in *Transport Mode*. To encrypt the packets IPsec uses the mechanism *Encapsulating Security Payload (ESP)*, encrypting the payload by the symmetric encryption algorithm AES-CBC with the keys, that are being managed by the Key & Signature Manager. 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 fulfil client and server duties because they redirect tasks to and process tasks for other VMs, which makes them so called *servents*. 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, the Event Monitor, and the Download Manager. Finally the *data* is managed by the Key & Signature Manager, the Logger, 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 Event Monitor, all the other modules can be outsourced to other machines.[4]

V. Description of the Individual 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 & Signature Manager, the Inspector, the Logger, the Communication Monitor, and the Event Monitor. A closer look at the individual modules is given below.

A. Event Monitor

The Event Monitor is 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. Additionally, the Event Monitor can advise the Inspector to automatically check the system for integrity, which then reports errors to the Logger or advises the Event Monitor to shut down the system. Every module announces to the Event Monitor in fixed intervals that it is sane. If a module is not responding, the Event Monitor can restart it. For restarting VMs or shutting down the system, the Event Monitor needs special access rights that no other module has.

B. Key & Signature Manager

The Key & 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 & 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

[4]When using an open network reliability is not ensured, therefore TLS should be used as the transport layer protocol, which the Connection Manager should enforce for network communication.

to the manager. Only an authorized entity should be able to command the Key & Signature Manager in this way, even a sealed hardware switch could be possible that needs to be broken to allow the reassigning of keys.

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 SoftwareIDs for integrity checks at boot and runtime.

C. 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, analysing 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 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. Non-legally relevant data can be send unencrypted, but firewall rules for outgoing packets should be obeyed.

D. Inspector

The Inspector serves as a remote attestation server for market surveillance. 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 module is indirectly accessible through the network. The Connection manager redirects the network packets after checking if an authorized person is connected to the device. After connection establishment 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 Logger to print out the logging, to check for enough storage capacity and if needed to check the other modules for malware. Finally, it can advise the Key & Signature Manager to invalidate and incur public keys.

E. 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 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 Logger and advises the Event Monitor to restart the legally and/or non-legally VM.

F. 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 IV-A.

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 Communication Monitor, which in turn translates it to a network package and sends it to the legally relevant VM.

G. Secure GUI

Non-legally relevant output must be unambiguously marked to distinguish it from 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 which 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.

H. 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 Event Monitor to shut down the system.

I. Logger

The Logger is responsible for tracking interventions. On interventions and errors the other modules inform the Logger, by sending him a network packet. This massage is then concentrated to an aligned format and transmitted to the Storage Manager. On demand the Logger sends its log-file to the Inspector, which in turn may send it to the Secure GUI, the Connection Manager or the Communication Manager to transmit it to other devices.

VI. Analysing the System

We analyse the system by showing that the policies that should be implemented according to MILS, to be specific, *data isolation*, *information flow control*, *damage limitation* and *periods processing*, are upheld.

A. 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 IV) have no direct access to the outside world. Their software must be checked by 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.

B. 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 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 IV-B, and checking access permissions enforcing the *least privileged* principle. Through unique network card numbers (MAC-address) the communication partners are known and cryptography protects the confidentiality and integrity of their communication.

C. 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, ...). 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 III-A, 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 Event Monitor. If a VM does not conform to its predefined rules, the Event Monitor notices the misbehaviour and takes action, e.g. reloading and restarting the module. In the worst case, the Event Monitor can shut down the system.

D. 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 [18]. 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 analysed by the ensuing VM to gain information.

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 IV-A, 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.

VII. Conclusion and Future Work

In this paper, we presented a framework for a new secure system architecture for measuring instruments in legal metrology. We constructed our architecture by analysing 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. To harness device drivers and network stacks of general purpose operating systems we came to the conclusion that virtualization is the right solution to combine security with usability. We took a three-pronged approach. First, we separated the legally relevant parts from the irrelevant ones by putting them in different virtual machines. Second, 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 instruments in commission.

A. Future Work

To show the feasibility of our approach, we have started to build a system atop a 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 (seL4) is even fully verified [12] inferring that classical security threats against operating systems, like buffer overflows, null pointer dereferencing, 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 (PandaBoard Rev. A3) we used L4Linux running atop the open source Fiasco.OC L4-microkernel which yields good results even for real-time applications [14].

Our main goal is to construct a configurable framework, applicable for every measuring instrument under legal control.

References

[1] K. Adams and O. Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. *SIGARCH Comput. Archit. News*, 34 (5):2–13, Oct. 2006. ISSN 0163-5964. doi: 10.1145/1168919.1168860.

[2] R. W. Beckwith, W. M. Vanfleet, and L. MacLaren. High Assurance Security/Safety for Deeply Embedded, Real-time Systems. *Embedded Systems Conference*, 2004.

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

[4] C. Ebert and C. Jones. Embedded Software: Facts, Figures, and Future. *Computer*, 42(4), April 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.118.

[5] T. Frenzel, A. Lackorzynski, A. Warg, and H. Härtig. ARM TrustZone as a Virtualization Technique in Embedded Systems. *Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi*, 2010.

[6] H. Härtig, J. Loeser, F. Mehnert, L. Reuther, M. Pohlack, and A. Warg. An I/O Architecture for Mikrokernel-Based Operating Systems. *TU Dresden technical report TUD-FI03-08, Dresden*, July 2003.

[7] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter. The Nizza secure-system architecture. *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2005. doi: http://doi.ieeecomputersociety.org/10.1109/COLCOM.2005.1651218.

[8] G. Heiser. The Role of Virtualization in Embedded Systems. In *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, IIES '08, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-126-2. doi: 10.1145/1435458.1435461.

[9] G. Heiser, V. Uhlig, and J. LeVasseur. Are Virtual-machine Monitors Microkernels Done Right? *SIGOPS Oper. Syst. Rev.*, 40(1):95–99, Jan. 2006. ISSN 0163-5980. doi: 10.1145/1113361.1113363. URL http://doi.acm.org/10.1145/1113361.1113363.

[10] M. Hohmuth, M. Peter, H. Härtig, and J. S. Shapiro. Reducing TCB Size by Using Untrusted Components: Small Kernels Versus Virtual-machine Monitors. In *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW 11, New York, NY, USA, 2004. ACM. doi: 10.1145/1133572.1133615.

[11] T. Kerstan, D. Baldin, and S. Groesbrink. Full virtualization of real-time systems by temporal partitioning. *Proceedings of the Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Brussels*, 2010.

[12] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: 10.1145/1629575.1629596. URL http://doi.acm.org/10.1145/1629575.1629596.

[13] A. Lackorzynski, A. Warg, and M. Peter. Virtual Processors as Kernel Interface. *Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi*, 2010.

[14] A. Lackorzynski, J. Danisevskis, J. Nordholz, and M. Peter. Real-Time Performance of L4Linux. *Proceedings of the Thirteenth Real-Time Linux Workshop, Prague*, 2011.

[15] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter. 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*, SPSM '11, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1000-0. doi: 10.1145/2046614.2046623.

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

[17] A. S. Liebergeld, M. Peter, and A. Lackorzynski. Towards Modular Security-Conscious Virtual Machines. *Proceedings of the Twelfth Real-Time Linux Workshop, Nairobi*, 2010.

[18] M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. In *Proceedings of the 13th International Conference on Selected Areas in Cryptography*, SAC'06, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74461-0. URL http://dl.acm.org/citation.cfm?id=1756516.1756531.

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

[20] *Directive 2014/32/EU of the European Parliament and of the Council*, February 2014. Official Journal of the European Union. doi: 10.3000/19770677.L_2014.096.eng.

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

[22] M. Peter, H. Schild, A. Lackorzynski, and A. Warg. Virtual Machines Jailed: Virtualization in Systems with Small Trusted Computing Bases. In *Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems*, VDTS '09, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-473-7. doi: 10.1145/1518684.1518688.

[23] F. Thiel, U. Grottker, and D. Richter. The challenge for legal metrology of operating systems embedded in measuring instruments. In *OIML Bulletin*, 52 (LII). OIML Bulletin, 2011.

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