ACTA IMEKO
ISSN: 2221-870X
March 2018, Volume 7, Number 1, 13-19

Software Separation in Measuring Instruments through Security Concepts and Separation Kernels

Daniel Peters, Patrick Scholz, Florian Thiel

Physikalisch-Technische Bundesanstalt, Abbestrasse 2 - 12, 10587 Berlin, Germany

ABSTRACT
In the age of the Internet of Things and Industry 4.0, more and more embedded systems are connected through open networks, which also concerns measuring instruments under legal control (e.g. smart meters). Therefore, cyber-security for measuring instruments is becoming increasingly important. In this paper, possibilities to design secure measuring software running on general-purpose operating systems are analyzed according to legal requirements set up by European Directives, e.g., the Measuring Instruments Directive (2014/32/EU), which define the mandatory security level. Technical interpretations for the security concepts described in this paper are derived from these legal requirements with the aim to provide manufacturers the architectural guidance to construct systems which easily pass a conformity assessment at a Notified Body. In this paper security concepts, i.e., SELinux, AppArmor and Mandatory Integrity Control (MIC) are being described, which are based on Mandatory Access Control (MAC) strategies. Additionally, high-security methodologies and concepts, e.g., Multiple Independent Layers of Security (MILS) and security kernels, are highlighted. In the examples given, software separation, which enhances overall security, is achieved by using SELinux mechanisms in the modules (virtual machines) atop a separation kernel.

Section: RESEARCH PAPER

Keywords: SELinux; AppArmor; MIC; MAC; MILS; legal metrology; WELMEC 7.2 Software Guide; separation kernel; hypervisor; MID

Citation: Daniel Peters, Patrick Scholz, Florian Thiel, Software Separation in Measuring Instruments through Security Concepts and Separation Kernels, Acta IMEKO, vol. 7, no. 1, article 4, March 2018, identifier: IMEKO-ACTA-07 (2018)-01-04

Section Editor: Lorenzo Ciani, University of Florence, Italy

Received September 25, 2017; **In final form** January 14, 2018; **Published** March 2018

Corresponding author: Daniel Peters, e-mail: daniel.peters@ptb.de

1. INTRODUCTION

Embedded systems play an increasingly important role in industry and economy and, in Europe, currently have an annual growth rate of 12% [1]. Software security is becoming a major focus of research, because today's embedded systems are often connected to insecure networks such as the Internet. Nevertheless, software architectures are becoming more and more complex, making it increasingly difficult to design secure systems.

Focusing on measuring instruments in Europe, in Germany alone, more than 100 million legally relevant measuring instruments are in use, mainly electricity, gas, water and heat meters which in total lead to an annual gain between €104 billion and €157 billion [2]. A malfunction or manipulation of these measuring devices could cause considerable damage. Therefore, it is particularly important to focus on the security of such systems and their software.

Manufacturers of measuring instruments prefer to use general-purpose operating systems (GPOSs) such as Windows or Linux due to the comfortable software infrastructure and the broad availability of device drivers. These operating systems include several million source lines of code (SLOC). Studies have revealed that on average a software bug can be found in every 2300 lines of code [3]. The presence of only one bug can be sufficient for an attacker to gain unauthorized access, which leads to the assumption, that measuring devices running on GPOSs are under great risk of manipulation.

The focus of the basic requirements for measuring instruments under legal control is the protection of consumers and the correctness of the measurements. In Europe that trust is ensured by institutions, so-called *notified bodies*, which devote themselves to the validation of measuring instruments before commissioning and the assessment of conformity. The tests in laboratories primarily focus on the hardware, such as the physical sensors. Testing the software proves to be more difficult, since operating systems and their additional software have too many lines of code and are often not open source. To support this challenge, GPOSs enhance their systems with security concepts. In legal metrology most of the times through malfunction or manipulation financial loss or gain can be caused. Still in other domains, which are also concerned about the right measuring of sensors, loss of life can be the result of a malfunction. An example is railway level crossings [18].

The analysis this paper provides focuses on how *"Mandatory Integrity Control"* (Windows), *SELinux* and *AppArmor* (Linux) can be used to fulfil the requirements in legal metrology. Especially, security frameworks for measuring instruments under legal control are described that can be used to fulfil the requirements of legal metrology documents. Furthermore, high-security methodologies and concepts, e.g., Multiple Independent Levels of Security/Safety (MILS), are highlighted and a more sophisticated framework based on a separation kernel design is analysed. It enables the use of drivers and network stacks of GPOSs with the help of virtualization. In this framework, the legally relevant parts are separated to allow higher degrees of freedom for the unregulated software. Afterwards, a communication framework, which provides services to these VMs is created. This framework, also consisting of separated VMs, monitors the information flow, correctly delegates requests from and to I/O devices and helps control agencies (such as the market surveillance) to verify system integrity. The latter security framework does not hinder the additional MAC control mechanisms described in this paper, which can still be applied additionally in the VMs, as shown in Section 6, on an ARMv8 board.

2. LEGAL METROLOGY

To help consumers to rely on the correctness of measurement results without having to check them themselves, legal regulations at national and international level are necessary. Legal metrology creates the prerequisite for producing high-quality products and contributes significantly to the functioning of an economy. It is estimated that in industrialized countries, 4% to 6% of the gross national income is settled by measuring instruments and the related measurements [2].

At the international level, the "Organisation Internationale de Métrologie Légale" (OIML) is an intergovernmental organization that has set up a world-wide technical set of proposals to help with the harmonization of legal metrology. For example, software requirements for measuring instruments are formulated in the *OIML D 31* [4] guide.

In Europe, the *Measuring Instruments Directive* (MID) [5] of the European Union formulates concrete legal requirements. In addition, there is *WELMEC* a committee at the European level in legal metrology, which includes the member states of the EU and EFTA. The guidelines for the conformity assessment of measuring instruments, the so-called WELMEC Guides, help the manufacturers and the notified bodies to build or check measuring instruments, respectively. Here, *WELMEC 7.2* [6] is the Software Guide for measuring instruments. It states that the software-related requirements of the MID are fulfilled if the requirements of the guide are upheld [6].

3. BACKGROUND

In legal metrology, the risks of manipulating measuring instruments and the theft of sensitive data are increasing, because of the integration of general-purpose operating systems such as Windows or Linux, and because many devices are connected to the Internet. So nowadays, security is becoming an important focus.

In general, **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 three points:

- **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.

By using access control strategies and a secure boot mechanism, it is possible to ensure adequate security (upholding the three points from above) for connected devices. In addition, separation of functionalities in IT systems can provide even more security. For example, a further concept for high security IT systems is the "**Multiple Independent Levels of Security**" (**MILS**). *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 implementing a limited set of critical functional security policies [7], described in Section 3.3.

3.1. Legal Requirements

The *WELMEC 7.2* Software Guide tries to break down the requirements for legal metrology software of the *MID* to special technical examples and recommendations. One important point is verifying measuring instruments in commission. Validating the software identification ensures that software was not switched or manipulated. Out of the *MID* and the assessment of hazards, the *WELMEC 7.2* Software Guide defines six risk classes from A to F, evaluating the need for **software protection**, **software examination** and **software conformity**. The risk classes are ascending (from A to F) in their demand for security. For our purposes, the best way to start is to construct a system for risk Class F measuring instruments, because it conforms to all requirements of the other classes. Hence, the solution provided here can be downscaled to lower risk classes.

Furthermore, the *WELMEC 7.2* clarifies what **legally relevant parts are**. 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 received or stored data.*** Based on these seven points, we construct our security concepts in the next section.

Additionally, the WELMEC 7.2 differentiates between measuring instruments that are built solely for the measuring purpose and the ones that run universal software. The two classes are called P and U. Normally one can say, if a measuring instrument has an operating system installed, it is a type U instrument, else it belongs to the P class. For both classes, four subclasses are defined which deal with following IT functions:

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

To be more specific, the WELMEC 7.2 Software Guide tries to break down the requirements for legal metrology software of the MID to technical examples and recommendations, looking more closely at the four points from above. The important MID software requirements, we spotted, are:

1) The measuring instrument must guarantee reproducibility, i.e., measurement results of the same input must yield the same output, even if handled by different users. This implies that a measurement result should not depend on the user/consumer employing the instrument. From the software point of view, different processes with varying access rights performing the same measurement should yield the same result.
2) A measuring instrument shall be designed to reduce as far as possible the effect of a defect (e.g. software bug) that would lead to an inaccurate measurement result, unless the presence of such a defect is obvious.
3) A measuring instrument shall have no feature to facilitate fraudulent use, and possibilities for unintentional misuse shall be minimal.
4) Software identification shall be easily provided by the measuring instrument. Additionally, evidence of an intervention shall be logged to verify measuring instruments in commission.
5) If a measuring instrument has associated software which provides other functions besides the measuring function, the software that is critical for the measurement purpose shall be identifiable.
6) Measurement data and software that is critical for measurement characteristics and stores or transmits metrologically important parameters, shall be adequately protected against accidental or intentional corruption.
7) For utility measuring instruments, the display of the total quantity supplied or the displays from which the total supplied quantity can be derived, which form the basis for payment, shall not be resettable during use.
8) The indication of any result shall be clear and unambiguous, i.e., easy reading of the presented result shall be permitted under normal conditions. Still, additional information may be shown, if they cannot be confused with the metrologically important results.
9) A durable proof of the measurement result and the information to identify the transaction shall be available.

Additionally, there are specific metrological requirements for the measuring instruments analysed in the MID. To give an example, we look at taximeters, which measure the time, the distance and calculates the fare for a trip based on the applicable tariffs. Hereby it is stated that in case of a reduction of the voltage supply to a value below the lower operating limit as specified by the manufacturer, the taximeter shall:

- continue to work correctly or resume its correct functioning without loss of data available before the voltage drop if the voltage drop is temporary, i.e. due to restarting the engine;
- abort an existing measurement and return to the position "For Hire" if the voltage drop is for a longer period;
- the taximeter needs to have a long-term storage, the data shall be available in the taximeter for at least one year.

An acceptable solution, for example, would be that the voltage level detector fires an interrupt when the voltage level drops for a time of 15s. The assigned interrupt routine collects measurement values, state values, and other relevant data and stores them in a non-volatile storage e.g. EEPROM. After the voltage level rises again the data is restored and the functioning continues or is stopped.

3.2. Access Control Strategies

In an Information Technology (IT) system, access control strategies determine which access type (e.g., read, write, or execute) a subject (e.g., user, process, or device) may have to an object (e.g., file, table, or subject). The *Discretionary Access Control* (DAC) strategy, for example, is a strategy in which access rights are defined only by the identity of a subject.

In legal metrology, where the focus is the integrity of measurement data and the confidence in the correctness of a measurement, this access control strategy may not be sufficient. Here, one should use the *Mandatory Access Control* (MAC) strategy, in which an access decision is additionally determined by means of object properties and rules. One of the MAC concepts is the *Multilevel Security* (MLS). In this concept, all subjects and objects are classified into four categories (security level): unclassified, confidential, secret and top secret. For these categories, rules for read and write access are defined in different models. Examples are given bellow.

The first example is the *Bell-LaPadula* model, which deals with the confidentiality of data. It is constructed with the purpose to protect information from unauthorized access. Following, security rules are being formulated:

- Simple-security rule (No read up policy): A subject can read the information of an object if its security level is equal to or greater than that of the object.
- *-Rule (No write down policy): A subject is allowed to write to an object if its security level is equal to or less than that of the object.

Another example is the *Biba* model, which deals with the integrity of data. Information shall be protected against manipulation of unauthorized persons. The following security rules are defined:

- Integrity-*rule (No read down policy): A subject can read the information of an object if its security level is equal to or less than that of the object.
- Simple-integrity policy (No write up policy): A subject can write to an object if its security level is equal to or greater than that of the object.

In a *Role Based Access Control* (RBAC) strategy the users have no direct access to objects. Instead, permissions are directly linked to tasks by assigning roles. Each user is initially assigned a role and can have access to several of these roles. In addition, at least one group is assigned to each role. By dividing everything into users, roles, and groups, an access decision is based on the roles assigned to the user and the groups that are accessible to these roles.

3.3. Multiple Independent Levels of Security/Safety (MILS)

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 implementing a limited set of critical functional security policies. This special kernel, often called a separation kernel (see Section 5), implements the policies for information flow control, data isolation, damage limitation and period processing [14], which are defined as follows:

- 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.

A separation kernel should be small enough to be thoroughly evaluated. The applications managing sensitive data are then built atop the secure separation kernel. An advantage of this approach is its modularity. Software of varying security demands runs on the same microprocessor, by means of software partitioning through the separation kernel.

4. SECURITY CONCEPTS

4.1. SELinux

In Linux, the Discretionary Access Control (DAC) is used by default. Nevertheless, Mandatory Access Control (MAC) and Role Based Access Control (RBAC) strategies can additionally be used. One example is *Security-Enhanced Linux* (SELinux), a Linux kernel module.

SELinux is based on an advanced security kernel called Flask, which was developed by a research group at the university of Utah named Flux [8]. In the Flask framework, a *Security Server*, which contains the security policies, communicates via interfaces to a so-called *Object Manager*. At each request of a subject to an object the Object Manger checks the policies, with the help of the Security Server, to grant or deny access.

In SELinux, every user is assigned to exactly one role at any given moment in time. If allowed, the user can him/herself switch into another role. Each role has access to a set of domains and types. With the construct of user, roles and domains (or types), well-defined security policies can be constructed.

The *Strict Policy*, which is available in SELinux, is well suited to build a secure framework, because it upholds the least privilege principle, which states that every component should only have the access permissions it really needs to have to function properly, no more. Hence, each subject and each object are controlled by their own domain and have just the access rights they need. In Figure 1, our framework of user, roles and domains based on the legal requirements listed in Section 3, is depicted.

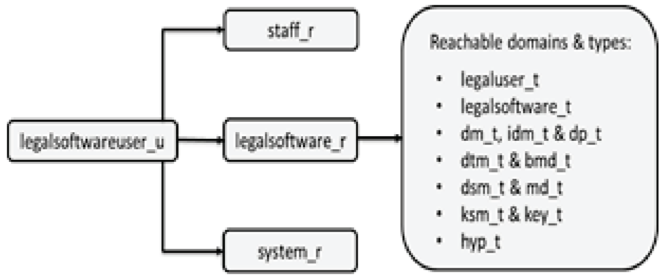

Figure 1. Framework of the user, roles and types in SELinux for legal and non-legal tasks.

As can be seen, we created a user called legalsoftwareuser_u in SELinux that can be in one of three roles: staff_r, system_r and legalsoftware_r. With the roles staff_r and system_r the user can have access to non-legally relevant objects, which are not needed for the measuring purpose. In our construct, only the legalsoftwareuser_u user has access to the role legalsoftware_r and only in this role the legally relevant domains can be used, which represent software processes that are needed for the measuring task. The legally relevant software parts can interact with each other in the way shown in Figure 2, and explained in the text above the figure. Users without the role legalsoftware_r cannot influence the legally relevant tasks, because they have no access to them.

4.2. AppArmor

A second way to use *Mandatory Access Control* (MAC) in Linux is the **ApplicationArmor** (**AppArmor**) framework. AppArmor is like SELinux a Linux kernel extension. It does not use *Role Based Access Control* (RBAC) strategies.

In AppArmor, *profiles* [9] and individual *security rules* are used to implement the MAC and to determine whether the access of a subject to an object is allowed or not. Each profile is created for just one process. In AppArmor only the processes with profiles are supervised. Because only newly started processes are monitored, AppArmor must be started at the beginning. In general, AppArmor divides processes into trustworthy and untrustworthy, and just guards the untrustworthy processes, because it is assumed that the others are not a danger to the IT system.

AppArmor can operate with much less rules then SELinux, making it easier to use. A disadvantage is that rules in AppArmor are based on absolute file-names. By renaming a file, the protection of AppArmor can be subverted. It should also be noted that AppArmor and SELinux cannot be used at the same time. If both modules are compiled into the Linux kernel, one must be deactivated. The framework we use to fulfill the legal requirements of the *WELMEC 7.2 Software Guide* is shown in Figure 2.

Here, only a user (**User**), which can execute the

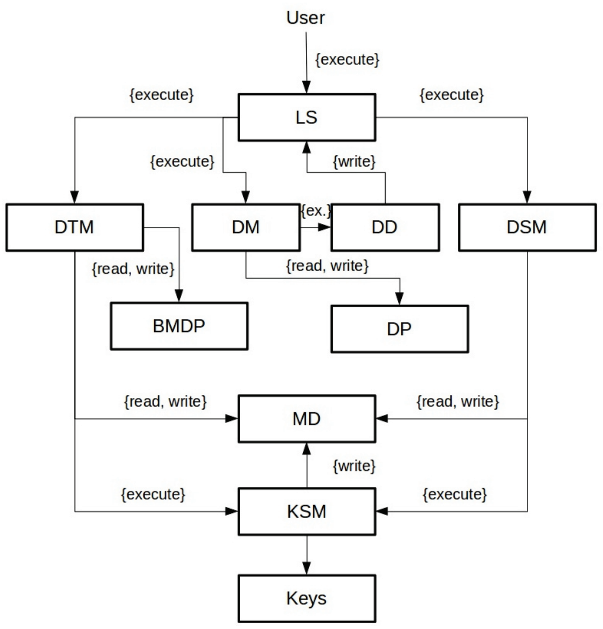

Figure 2. Legal software parts and their access permissions in SELinux and AppArmor.

measurement task, has access to the legally relevant software (**LS**) and through this, access to the Legally Relevant software modules, like the Download Manager (**DM**), the Data Transfer Module (**DTM**) and the Data Storage Manager (**DSM**). The DTM and the DSM are responsible for the transmission of measurement data and for their long-term storage. Both modules have access to the measurement data (**MD**) and can use a key & signature manager (**KSM**) for de- and encryption of data. In the broken measurement data protocol (**BMDP**), errors, like damaged measurement data can be recorded. The DM is used for the transfer of downloaded data (**DD**), the subsequent verification procedure and the following installation. To verify this data, the DM uses hash values to ensure authentication. Also, a download protocol (**DP**) is being created for each download process.

4.3. Mandatory Integrity Control

Since Windows Vista, MAC is available by a framework called Mandatory Integrity Control (MIC). It can be used additionally to the DAC, which Windows implements by default, like Linux. Hereby, an access allowed from the DAC can still be prevented by the MIC.

In MIC each object is considered a securable object and is classified into one of five classifications, called the *Integrity Level* (IL) that is part of the *Security ID* (SID) stored in a Mandatory Label of an object [10]. These five classifications are: Untrusted, Low, Medium, High and System. With these Integrity Levels and Mandatory Labels an access decision is made. MIC enforces system-wide no-up policies: no-write up, no-read up and no-execute up, meaning that a user with a lower IL is not allowed to access an object with a higher level.

The IL System is preserved for system-related processes such as kernel processes. No user can acquire this IL [10]. High is, e.g., for the administration of a system, but can be acquired from normal processes if needed. Medium is the default setting of most processes. The IL Untrusted is for unidentified processes and Low for low-trusted processes with higher attack-risk.

The Mandatory Label of an object which includes the SIDs, can be found in the Security Descriptor. The SIDs of subjects are stored in an Access Token alongside a set of privileges and a list of group memberships.

Figure 3 gives an overview of the ILs we use for our software framework, seen in Figure 2. Here, the legally non-relevant software (NLS), which is not used for the measurement, is classified as IL Medium. The legally relevant modules, that are used for measuring processes, are classified

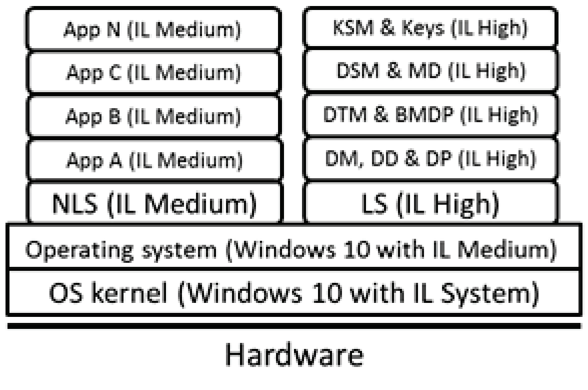

Figure 3. Framework of the user, roles and types in SELinux for legal and non-legal tasks.

with IL High. Because of the no-up policy in MIC, a manipulation of the legal relevant modules is not possible.

5. SEPARATION KERNEL SYSTEM ARCHITECTURE

As mentioned in Section 3.3, a separation kernel is the foundation of a MILS architecture, it needs to fulfil following properties:

- *Non-bypassable*: the security monitor is not bypassable by a component through a hidden communication path, including lower level mechanisms.
- *Evaluatable*: any component that is critical for the trustworthiness of the system can be evaluated to the level of high assurance.
- *Always-invoked*: each and every access/message is checked by the appropriate security monitors.
- *Tamperproof*: the system controls rights to the security monitor code, e.g., configuration and data; preventing unauthorized changes.

An acronym, which is often used, for these characteristics is NEAT.

5.1. Virtualization

The MILS system architecture, especially the separation kernel, together with virtual machines (VMs) as its components, furnishes a great base for high-assurance software. Through virtualization GPOS applications can be directly executable. In general, virtualization is divided into two main approaches [14]. Pure virtualization supports unmodified guest operating systems, running on top of a kernel, which is called hypervisor. Hereby, closed-source operating systems are directly executable without the need for software changes. Unfortunately, many processors do not have adequate support for pure virtualization. Therefore, the second approach, called para-virtualization, presents the guest operating system with an interface that is nearly identical to the underlying hardware to make virtualization possible. To improve performance and allow virtualization at all, the guest operating system is often modified.

In the past, embedded systems and also measuring instruments used to be relatively simple devices, which did not support virtualization. Nowadays, many measuring devices have the characteristics of general purpose systems. This power together with the low development costs for a board combined with hardware virtualization technologies makes virtualization in measuring instruments, attractive. The motivation for virtualization is in our case also security. By running an operating system in its virtual machine (VM) safely encapsulated, the damage of an attack cannot propagate to other VMs.

5.2. System Architecture

A separation kernel architecture well suited for measuring instruments is described for example in [16]. Here, the legally relevant parts, which are needed for the measuring purpose, are separated from the irrelevant ones by putting them in different virtual machines. Afterwards, it is made sure that their virtual machines have no direct access to I/O devices. Figure 4 depicts such a stripped system architecture.

The framework consists of three virtual machines that run atop the separation kernel. In the L (egally relevant) VM, all the programs are running that are important for the measuring purpose. In the N(on-legally relevant) VM everything else is

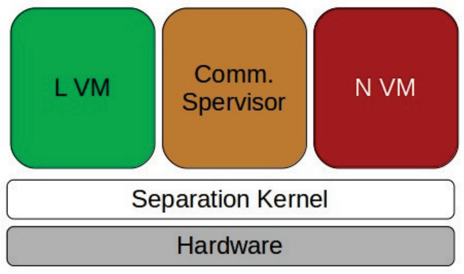

Figure 4. Framework with a separation kernel running three VMs.

running. To communicate with the separation kernel and through it with sensors and perhaps the internet, the communication supervisor is used, the third VM. It can be used to monitor the information flow, correctly delegate requests from and to I/O devices and help control agencies (such as the market surveillance) to verify system integrity.

With the VM approach communication between the individual VMs can take place through network protocols already implemented in the GPOS, and the GPOS device drivers can be used, creating virtual networks connections. The disadvantage of this concept is that attacks or bugs in the individual VMs are the same as in the normal GPOSs, the separation kernel only makes sure that an attack/bug in one VM cannot propagate into the others. The disadvantage should be counteracted by using minimal configurations for the GPOSs and by configuring the individual VMs with the security frameworks described in this paper. An example is given in the next section, showing how a separation kernel can execute under a Linux with activated SELinux.

6. PRACTICAL TESTS

To show the practicality of our approach, we have built two test systems.

With the minimal implementation principle in mind, we constructed the first system with a minimal Linux and SELinux support, to see how much space is needed. For demonstration purposes we used QEMU, which is a generic and open source machine emulator and virtualizer.

We compiled the Linux kernel (version 4.11.0-rc4) for x86 systems, which most desktop computers (e.g. based on Intel chips) are nowadays. Additionally, we used Buildroot to create our root filesystem. Buildroot is a tool that helps to generate embedded Linux systems. With this tool, it is also simple to cross-compile everything, constructing for example also binaries for ARM platforms. The compilation of BusyBox is also supported that is often used in embedded systems to replace more than 300 common Linux commands, e.g., ls, sh, rmdir, etc. into one executable. In the root filesystem of our Linux system, we added many additional packages to the core BusyBox-based system: the bash shell, a GUI (X.org display server, matchbox window manager), mdev as device manager, and of course the necessary infrastructure to activate SELinux (Linux kernel modules, SELinux support options in Busybox). Afterwards, we downloaded example policies (from [11]) and added them to our root filesystem. Finally, we activated SELinux by adding selinux=1 to the kernel commandline and created a test file /etc/selinux/config with following parameters: SELINUX=permissive and SELINUXTYPE=targeted, to set permissive mode, which does not automatically block unauthorised access, but just logs it for our demonstration purposes. By setting SELINUX=enforce, access can be if the device is in real operation mode. The resulting root filesystem including the kernel had a total size of 45.8 MB.

For our second test, we used a new hypervisor called Phidias ("Provable Hypervisor with Integrated Deployment Information and Allocated Structures"). This hypervisor was developed at the chair for Security in Telecomunications (SecT) of the TU Berlin and has been designed to exhibit a maximum of runtime staticity, i. e. all dynamic elements usually found in system software are either omitted altogether or carefully altered to behave statically. To obtain a bootable image, the hypervisor implementation requires an accompanying configuration framework centred around a simple, human-readable XML configuration file, which is integrated into the build process and generates the data structures for a deployment setting. This technique pushes all dynamic decisions (e. g. the placement of memory allocations) from runtime to compile-time and thus enables Phidias to be a static, yet very versatile hypervisor.

The goal of this design principle is the ease of automated verification against malfunctions and bugs. It was mainly developed in C, with some assembler code for low-level operations. It is kept small (6,134 lines of C + 1,305 lines of assembler) which of course also increases the tractability of automated verification. Due to this small size and the inclusion of all data structures for a given configuration, the bootable image can be verified to satisfy certain integrity and non-interference properties through symbolic execution.

Our test board was the HiKey (LeMaker version) board which is equipped with an ARM Cortex-A53 consisting of 8 cores, 1.2 GHz and 2GB of LPDDR3 RAM. The ARM Cortex-A53 is a 64-bit-multicore-processor with an ARMv8-A instruction set. With ARMv8-A, an additional exception level was implemented, the HYP mode for hypervisor support on which the Phidias hypervisor runs on. So, pure virtualization is easily possible through hardware support.

We compiled Phidias with multicore support and again used the open source tool Buildroot to generate RAM disks for two VMs. First, we added a virtual network driver to the Linux kernel (version 4.11.0-rc4) for inter-VM communication which uses the communication buffer system and the signalling mechanism that the Phidias configuration framework offers. No further modifications to the Linux source code were necessary.

In the root filesystem of the first Linux VM we added many additional packages to the core busybox-based system, like in our first test: the bash shell, a GUI (X.org display server, matchbox window manager), mdev as device manager, and of course the necessary infrastructure to activate SELinux (Linux kernel modules, SELinux support options in Busybox). Afterwards, we used the same example policies (https://rpmfind.net/linux/rpm2html/search.php?query=selinux-policy-targeted) and added them to our root filesystem. Finally, we also activated SELinux by adding selinux=1 to the kernel commandline and created a test file /etc/selinux/config with following parameters: SELINUX=permissive and SELINUXTYPE=targeted. The resulting root filesystem of our first VM including the kernel had a total size of 44.6 MB. The second VM was a minimal Linux (without SELinux), but still constructed with Buildroot with a size of 11.7 MB, which led to a total size of 56.3 MB for both VMs. Phidias itself had a final total size (compiled hypervisor binary, data structures, pregenerated pagetables) of less than 250 kB.

In our opinion, the ground framework of around 57 MB is no restriction for measuring instruments as our small board (costs of around 100 $) has already 2GB of RAM and an 8GB of NAND flash storage. Therefore, we think that our framework which combines MACs with a separation kernel is not only feasible but should be the next step to enhance security in measuring instruments, which are connected to the internet.

7. CONCLUSIONS

In this paper, secure software constructions for general purpose operating systems, i.e., Linux and Windows are analysed. Concretely, we look at measuring instruments under legal control, which have become powerful devices running under such operating systems. The frameworks presented here, are constructed to fulfil the requirements of legal metrology such as the Measuring Instruments Directive MID 2014/32/EU and the WELMEC 7.2 Software Guide.

The security methods shown in this paper are SELinux, AppArmor and MIC, which are based on Mandatory Access Control and/or Role Based Access Control strategies (MAC and RBAC). With these methods well-defined rules for roles, classifications for subjects and objects, and access rights are being defined for legally relevant software parts that fulfil measuring tasks. Additionally, a component-based architecture is described, in which every component receives only least privilege rights. Together with the before mentioned well-defined rules, all requirements of legal metrology directives for software are fulfilled. Our framework ensures that non-legal processes, which run on measuring instruments, have no effect on the legally-relevant processes. This is further enhanced using a hypervisor with separation kernel properties. The tests in this paper describe how an installation of such a framework is easily possible. The goal is to make current measuring instruments more resistant against software vulnerabilities and attacks from open networks like the Internet.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for several helpful suggestions that improved the quality of this article.

REFERENCES

[1] IDC France (L'Institut Du Comportement), “Special Study - Final Study Report: Design of Future Embedded Systems (SMART 2009/0063)”, IDC - Analyze the Future (International Data Corporation), Final interim Study Report 2, Deliverable D4, April 2012.

[2] N. Leffler, F. Thiel, “Im Geschäftsverkehr das richtige Maß - Das neue Mess- und Eichgesetz”, Monatsbericht, 11-2013.

[3] B. Chelf, “Measuring software quality - A Study of Open Source Software”, Chief Technology Officer, San Francisco, CA, USA, 2011.

[4] OIML D 31, “General requirements for software controlled measuring instruments”, Organisation Internationale de Métrologie Légale, Edition 2008 (E).

[5] Official Journal of the European Union, “Directive 2014/32/EU of the Eurpean Parliament and of the Council”, L 96/149, European Commission: Brussels, Belgium, 2014.

[6] WELMEC (European Cooperation in Legal Metrology), “Softwareleitfaden (Europäische Messgeräterichtlinie 2014/32/EU)”, WELMEC 7.2, 2015.

[7] D. Kleidermacher, M. Kleidermacher, “Embedded Systems Security - Practical methods for safe and secure software and systems development”, Newnes - Imprint of Elsevier, 2012, pp. 27-31.

[8] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, J. Lepreau, “The Flask Security Architecture: System Support for Diverse Security Policies”, Secure Computing Corporation, National Security Agency, University of Utah, USA, August 1999.

[9] R. Spenneberg, “SELinux & AppArmor - Mandatory Access Control für Linux einsetzen und verwalten”, 2008, Addison-Wesley Verlag.

[10] F. Kuhn, “Mandatory Integrity Control”, ERNW Newaletter 17, Juli 2007.

[11] SELinux policies, accessed: 28.04.2017, link: https://rpmfind.net/linux/rpm2html/search.php?query=selinux-policy-targeted.

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

[13] J. Barr, "The Flask Security Architecture", In Computer Science 574, 2002.

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

[15] K. Doeornemann, and A. von Gernler, "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.

[16] D. Peters and F. Thiel, "Software in Measuring Instruments: Ways of Constructing Secure Systems", SENSOR 2016, Nuremberg; 05/2016.

[17] F. Thiel, U. Grottker, and D. Richter, "The challenge for legal metrology of operating systems embedded in measuring instruments", OIML Bull. 2011

[18] T. Addabbo, A. Fort, C. D. Giovampaola, M. Mugnaini, A. Toccafondi and V. Vignoli, “On the safety design of radar based railway level crossing surveillance systems”, In ACTA IMEKO, Vol. 6, No. 4, 2017.