Trusted Virtual Domains on OKL4: Secure Information Sharing on Smartphones

Lucas Davi
System Security Lab/CASED
Technische Universität
Darmstadt, Germany
lucas.davi@trust.cased.de

Alexandra Dmitrienko
Fraunhofer SIT
Darmstadt, Germany
alexandra.dmitrienko@trust.cased.de

Christoph Kowalski
HGI System Security Lab
Ruhr-Universität Bochum,
Germany
christoph.kowalski@rub.de

Marcel Winandy
HGI System Security Lab
Ruhr-Universität Bochum,
Germany
marcel.winandy@trust.rub.de

ABSTRACT

The flexibility and computing power of modern smartphones to install and execute various applications allows for a rich user experience but also imposes several security concerns. Smartphones that are used both for private and corporate purposes do not separate the data and applications of different security domains, and users are usually too unskilled to deploy and configure extra security mechanisms. Hence, data leakage and unwanted information flow may occur.

In this paper we present the design and implementation of the Trusted Virtual Domain (TVD) security architecture for smartphones. The TVD concept separates data and applications of different security domains and automates the security configuration on devices. In particular, we build our solution on top of the OKL4 microkernel, which provides the basic isolation properties, and extend it with a framework that realizes the TVD policy enforcement for Android operating systems. Our results show that the TVD security architecture can be built and used on modern smartphones, but there are also limitations that current security kernels like OKL4 have to address to improve the user experience.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

TVD, smartphone, OKL4, microkernel

1. INTRODUCTION

Smartphones are increasingly used as general purpose computing devices with permanent Internet connection. Users can install and execute almost arbitrary applications on these devices. In particular, smartphones are often used for different tasks with different security requirements, e.g., for playing games, taking photos, running private online banking, or even accessing a corporate network. This imposes several threats as the phone operating system (OS) is typically derived from desktop counterparts and, hence, inherits the same or similar security shortcomings. Moreover, even though modern smartphone operating systems like Apple's iOS and Google's Android include application sandboxing and (in case of Android) advanced application permission control, these systems still lack support of grouping applications to different security domains and apply a corresponding separation mechanism. For instance, data belonging to an application of one domain (e.g., corporate data) should not be accessible in another domain, and it should hence be encrypted appropriately before transferred out of the domain (e.g., to the Internet), and decrypted again when imported back (e.g., to the corporate domain on the same or another device).

Today, the borders of private and corporate usage become indistinct due to convenient features of modern smartphones, which are often in private possession of users, but also used for some corporate tasks (email, todo lists, contacts, etc.). Currently, users have to take care themselves to separate or encrypt data and applications according to the security requirements of their enterprises. However, the average user is usually unskilled in deploying adequate security mechanisms or configuring them appropriately. Alternatively, enterprises may prohibit usage of corporate smartphones for private needs, forcing the users to have a dedicated device for private usage. This approach is secure because it provides physical isolation between corporate and private applications and their data. But it is also costly and user unfriendly, as it requires double costs for hardware and the user to carry two devices at the same time.

We aim to address those existing limitations and propose a trade-off solution which is (almost) as secure as physical separation but allows usage of a single device for business and private needs at the same time. We leverage the idea of Trusted Virtual Domains (TVDs) [16, 7, 8], known from data centers, which aims to tackle similar security risks. TVDs are a security framework for distributed systems that aims to provide verified containment, trust in components, simplicity in management and secure communication between its participants. Members of a TVD can communicate freely with each other, whereas communication to or from other systems is subject to explicit information flow control.

We deploy the TVD concept on smartphone devices with the goal to isolate corporate applications and data from untrusted software. The prominent advantage of TVD usage in this context is the transparent policy enforcement and simplified and scalable management based on TVD policies. Thus, security mechanisms that enforce the separation on TVD-enabled smartphones are transparent to the user and do not require any additional configuration overhead.

Contributions and Outline.

In this paper we present the realization of the TVD concept on smartphones. In particular, our contributions are:

- **TVD Framework for Smartphones:** We present the design (Section 3) and implementation (Section 5) of a TVD framework on top of the OKL4 microkernel, and we support to run Android operating systems and their applications in isolated environments to separate data and applications of corresponding security domains based on TVDs. Moreover, we provide an evaluation of our prototype with respect to usability and performance (Section 6).

- **Policy Mapping Tool:** To realize the TVD policy enforcement on OKL4, we have developed a policy mapping tool that translates the general TVD policy rules to specific security mechanisms of the OKL4 kernel (Section 4). We provide a full implementation and report on limitations of the current OKL4 which affect the user experience on a TVD-enabled system.

We choose OKL4 as underlying security and virtualization layer because it is a commercially available separation kernel that provides isolation of execution environments for two types of entities: Native applications and guest operating systems. As it is based on a microkernel architecture, OKL4 has a very small code base, which allows one to verify its correctness more easily than complex commodity operating systems [19]. In addition, the virtualization support of OKL4 allows execution of (several) full Android virtual machines on a single device. Android, being a popular smartphone OS, can run various user applications. Our system remains compatible with existing and future Android applications and does not impose extra efforts for software developers.

2. BACKGROUND

In this section we provide background information on the TVD concept, and summarize the existing security features of OKL4 that we use in our implementation.

2.1 Trusted Virtual Domain (TVD)

The concept of TVDs was introduced by researchers from IBM [16, 7, 18], and has since been extended and refined by a number of other works, e.g. [9, 3, 8]. The main goal of the TVD concept is to execute workloads of different domains in isolated computing environments.

A Trusted Virtual Domain (TVD) is a coalition of mutually trusting members, usually virtual machines. The TVD policy is a set of rules that state security requirements a TVD member must satisfy in order to be accepted as a TVD member (e.g., integrity measurements of the virtual machines and their software components). Members of a TVD are assigned a security label that identifies the TVD, which is also referred to as the "color" of the TVD. Basically, the TVD concept enforces a simple security model [2]: *Communication between two components is allowed if and only if they share the same color.*

From a logical point of view, TVDs are isolated (virtual) computer networks. However, their physical deployment can span over several computing platforms. On each platform, a trusted virtualization layer enforces the central TVD policies and provides the isolated execution environments for the TVD members. Figure 1 illustrates the logical and physical view of TVDs. As shown in Figure 1, the TVD infrastructure also provides separate virtual networks that are operated over one physical network, but isolated from each other (e.g., via encryption).

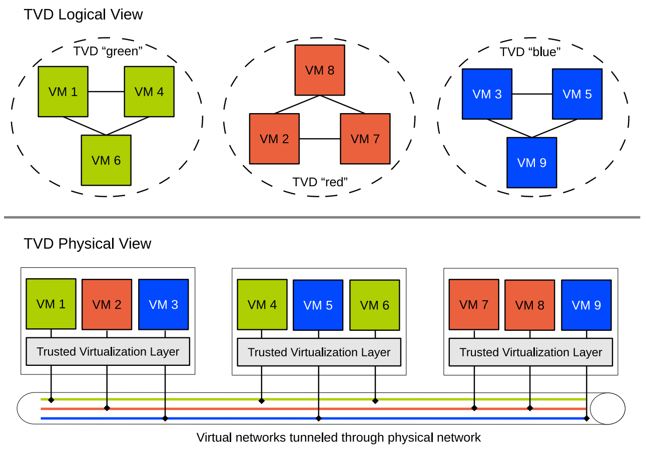

Figure 1: Logical and physical view of Trusted Virtual Domains (TVDs).

But the TVD concept goes beyond isolated execution and virtual networks. One important aspect is the information flow control. For instance, a strict TVD policy requires that all data belonging to a TVD must remain in that TVD. As a result, the TVD infrastructure ensures that data is automatically encrypted wherever it is stored or transmitted, including external storage devices like flash memory.

The TVD policy is configured centrally and enforced locally. The central management component is called *TVD Master*, which stores the TVD policy and controls admission of other physical platforms to the TVD infrastructure. On each platform, a *TVD Proxy* is created for each TVD, which is a local copy of the TVD Master and is responsible for admission of the VMs to the corresponding TVD. The TVD Proxy configures the local security services of a platform to realize the TVD policy enforcement.

Existing works on the realization of TVDs mainly have concentrated on data centers [3, 2, 8, 10] and desktop computers [14, 22]. Commercial TVD solutions are becoming available, e.g., Turaya[1]. However, the feasibility of the TVD concept for mobile platforms has not been investigated yet.

2.2 OKL4 Microvisor

The OKL4 Microvisor provides virtualization and compartmentalization in embedded system development and acts as a manager for hardware resources. It offers hardware abstraction (virtual CPUs, virtual MMU) and virtual device abstraction (virtual interrupts). The term Microvisor is a composition of the words microkernel and hypervisor and reflects that both technologies apply to OKL4 [17].

OKL4 builds on the top of HyperCell^{TM}-Technology and maintains isolated execution environments, so-called cells, that can run native applications or guest operating systems such as Android, Linux, and Symbian. OKL4 provides resource management, fast inter-process communication (IPC) among cells and enforces process isolation and mandatory access control on IPC calls based on capabilities – security tokens that protect access to protected resources.

Each cell is assigned a specific kernel memory pool and a set of physical memory address ranges (pre-defined by the system administrator). OKL4 ensures that address spaces of different cells are isolated from each other. Communication between cells is performed via IPC, which is the only means by which cells are allowed to communicate. The system administrator defines for each cell the available IPC channels, and specifies capabilities which other cells must possess in order to communicate over this channel. Note that the capabilities of each cell are defined once at system startup and cannot be changed afterwards.

OKL4 relies on a minimal trusted computing base (TCB) and has a small memory footprint.

3. ARCHITECTURE

Our TVD architecture is depicted in Figure 2. It includes a virtualization layer and isolated execution environments, so-called compartments, that can run guest operating systems or native applications. The virtualization layer provides process and memory management, interrupts, and device drivers. Further, it enforces isolation between execution environments and ensures that pre-defined communication policies between isolated partitions are applied. Our architecture contains compartments of two types: (1) *TVD compartments* that are assigned to a TVD and typically run a guest operating system such as para-virtualized Android, and (2) *system compartments* that do not belong to any TVD but provide security services to the rest of the system. The latter include a trusted graphical user interface system (Secure GUI), an attestation service, and a Mobile Trusted Module (MTM) [25].

The virtualization layer, separated execution environments and system services described above are similar to building blocks of Trusted Mobile Desktop (TMD) [13] – an architecture that uses virtualization to separate execution environment of the mobile platform into trusted and untrusted isolated worlds (e.g., to allow separation of private and corporate assets when a corporate smartphone is in use for both, business and private needs). We use the TMD architecture as basis for our solution. In addition, our architecture includes TVD-specific components: TVD Proxy, TVD Master and TVD Policy. The TVD Master acts as a central service that manages the TVD infrastructure and serves as place where administrators can specify the TVD Policy. TVD Proxy is an additional client security service, which deploys the TVD Policy on the platform and is responsible for platform configuration and local policy enforcement. The TVD Policy is delivered from TVD Master to TVD Proxy via a trusted channel [15, 1].

[1]Developed by Sirrix security technologies http://www.sirrix.com

3.1 Building Blocks

In the following we provide a more detailed description on the building blocks of our architecture.

Virtualization layer.

As our virtualization layer we chose the OKL4 Microvisor [17] (in contrast to TMD [13] which has been build on top of PikeOS [5]), because it is available under open source license for research and educational purposes. OKL4 accurately fits into the TVD architecture since it provides high-performance inter-process communication (IPC) among different cells and enforces process isolation and mandatory access control on IPC calls based on capabilities.

Mobile Trusted Module.

The TCG *Mobile Trusted Module* (MTM) [25] is a security extension for embedded devices, a lightweight counterpart of the *Trusted Platform Module* (TPM) [26], which is used for PCs. Similarly to TPM, MTM provides means to record platform configuration in extendable *Platform Configuration Registers* (PCRs). We use a software MTM in our architecture (i) to establish trusted channels, (ii) to support remote attestation, and (iii) to manage cryptographic keys.

Attestation Service.

The Attestation Service provides the remote attestation functionality that allows a remote party to verify if a platform configuration is in a trusted state. It attests the security kernel and application-level compartments and is used in a process of a trusted channel establishment between TVD Proxy and TVD Master.

Compartment Manager.

The compartment manager is an application that provides a graphical user interface for local compartment management. It delegates the commands received from the user to be executed by the TVD Proxy. Further, it shows the graphical output of the started command and its result.

Secure GUI.

Any user input/output on the device goes through a secure GUI. The Secure GUI catches all user inputs and offers an unique framebuffer for graphical outputs of each compartment. It includes a *trusted bar* – a reserved area on the screen that shows information about the compartment the user is currently interacting with. This includes the name and the color of the compartment, i.e., its TVD membership. Further, it ensures that the user's input is delivered to the indicated compartment, and not to any other one.

TVD Compartments.

Each TVD compartment runs a single guest operating sys-

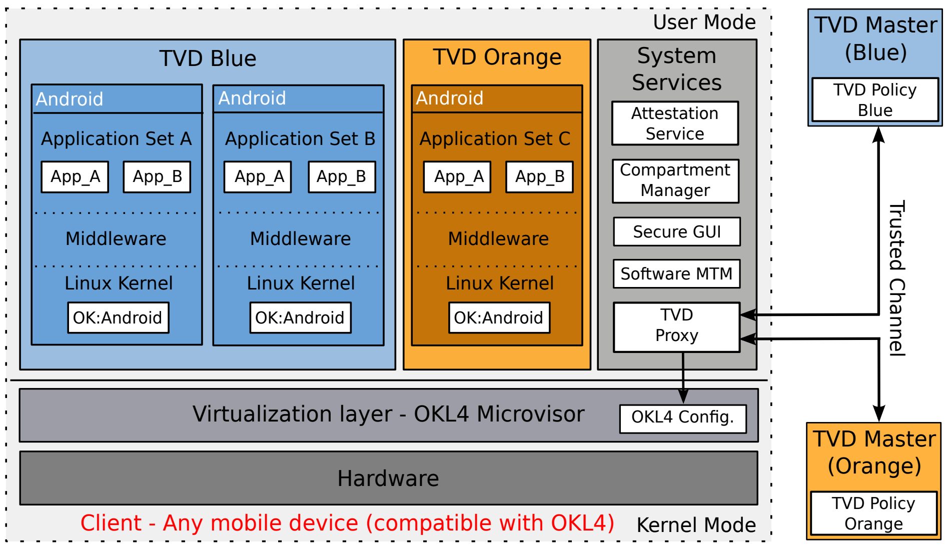

Figure 2: Architecture Overview

tem. For instance, the platform depicted in Figure 2 has three TVD compartments that run Android OSes. Each Android instance consists of a Linux kernel, the middleware and an application layer. The Android Linux kernel is modified with the OK:Android [24] package, which provides microkernel support for Android. Each TVD compartment is assigned to a domain. For instance, in Figure 2, two TVD compartments are assigned to TVD Blue and one compartment belongs to TVD Orange. Compartments from a single domain can freely communicate, while communication between compartments of different domains is prohibited.

TVD Master.

TVD Master is a remote TVD administrator. Each TVD has its own TVD Master, which is responsible for defining a TVD security policy.

TVD Proxy.

The TVD Proxy is responsible for establishing the trusted channel to a TVD Master and TVD Policy delivery to a local platform. When several domains are deployed on the platform (e.g., TVD Blue and TVD Orange), TVD Proxy communicates with the corresponding TVD Masters, receives the TVD Policies, and maps these policies to an OKL4 configuration.

TVD Policy.

The TVD Policy specifies the TVD compartments to be installed on the platform and the available resources on the device to be assigned to compartments. Further, inter-TVD communication may also be specified in the policy. By default, communication between different TVDs is not allowed. We elaborate on the TVD Policy in more detail in Section 3.3.

3.2 Design Challenges

One of the challenges we have to deal with is the static configuration of OKL4. The number of cells on the platform, IPC channels among them, and capabilities to control access to these channels are specified by the OKL4 configuration. This configuration cannot be changed at runtime, but can be updated only upon reboot. The challenge here is to build a usable system despite this static limitation. The straightforward approach would be to change OKL4 or use another system, but we want to stick with a commercially available and deployed system to provide a solution that has more chance of adoption in practice. Hence, our objective is to build a system that lives with these limitations but is still efficient and usable enough.

Hence, the static nature of OKL4 influences our design decisions on the integration and enforcement of TVD Policies. We transform the TVD Policy into an OKL4 Microvisor configuration, which defines the system layout and access rules (capabilities). Next, a new configuration is applied to build a new executable OKL4 system image. In contrast to existing TVD solutions for desktop platforms [10, 9, 22], our TVD Policies can be updated only by means of building and installing a new system image on the platform.

Further, we are not able to reuse TVD Policies that are used for desktop systems due to the different types of resources in both environments. In a desktop computer environment, the typical resources are logical networks or file disks. In an embedded environment, especially OKL4 Microvisor, the focus is on more profound resources like interprocess communication channels, interrupt handling or initial RAM file systems and is bound to the targeted platform. In addition, mobile systems have static resources like physical devices or defined physical/virtual memory pools that are platform-dependent.

Finally, TVD deployment on the platform typically involves execution of two protocols [23, 10]: First, the *TVD Deploy* runs to verify the trustworthiness of the platform, and second, *TVD Join* protocol is executed by the TVD Proxy to determine if local virtual machines can participate in the TVD. However, this approach is not applicable to the OKL4 architecture due to its static nature. Since joining a TVD by a compartment would require changes in the underlying system configuration, this operation cannot be performed at runtime. Thus, we merge the protocols *TVD Deploy* and *TVD Join* into a single protocol.

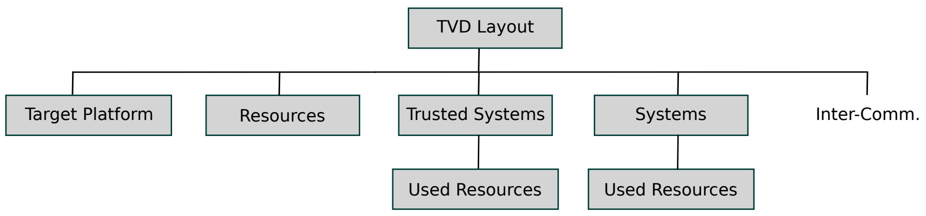

Figure 3: Conceptual structure of an OKL4 oriented TVD Policy

3.3 TVD Policy

The TVD Policy is expressed in XML language. The policy structure is represented in Figure 3. The figure shows a tree view of the XML structure and contains five groups of elements which we describe in the following:

- Target Platform: This group specifies the platform the TVD Policy is defined for. Note, that due to platform-dependent properties such as available hardware buttons and corresponding interrupt interfaces, the mobile version of the TVD Policy is platform dependent.
- Trusted Systems: This group of elements describes a trusted computing base (TCB) of the platform. Particularly, these include system services (depicted in Figure 2) and the OKL4 kernel.
- Systems: This group represents the set of TVD Compartments. The OKL4-based platform supports compartments that run either native applications, or guest operating systems like Android.
- Resources: This group specifies resources on the device. These are mainly IPC channels to trusted systems or initial ramdisk files for systems that can be assigned to compartments.
- Inter-Comm: This group defines inter-TVD communication rules. It specifies IPC channels between systems belonging to different TVDs. Elements of this group have to be synchronized in TVD Policies of all corresponding TVDs.

In addition, Trusted Systems and Systems may have child elements called ”Used Resources”. They assign allocated resources from the Resources group to the system elements.

4. SYSTEM MANAGEMENT

In this section we describe the mapping of the TVD Policy to the OKL4 configuration and the realization of the commands *Install Compartment* and *Delete Compartment*.

4.1 TVD Policy Mapping

In order to map multiple TVD Policies to a single OKL4 configuration, our system has to provide an $n : 1$ mapping technique. We rely on the fact that the TVD Policy and the OKL4 configuration share several characteristics, in particular, we exploit that both organize their elements (e.g., resources and virtual machines) into groups. The idea is to select elements of these groups and put them into sets, which are related to TVD policies. Newly formed sets are merged with mapping rules into one output set (OKL4 configuration), which contains OKL4 related elements.

The general workflow for the policy mapping is as follows (see Figure 4): We receive TVD policies as input and extract elements (e.g., resources, trusted systems) from the policy (step 1); then we merge them to output cells (step 2), and finally we map them into a corresponding OKL4 configuration (step 3).

In the *Extract* phase our system processes the TVD policies and parses the individual XML fields. In particular, it extracts the systems the user selected, and includes resources associated with the selected systems. The trusted systems are identical for all TVD policies. Extracted information is provided as the input for the model to generate sets. The set generation is performed as follows: First, the selected systems are associated with sets, based on the corresponding TVD Policy. Second, the trusted systems are associated with one dedicated set. Finally, every system and the trusted systems are associated with a resource set.

The *Merge* phase is responsible to copy previously generated sets into one output set, denoted as Cells. This process is performed under the following mapping rules:

1. For each element of all System Sets a new element in the set Cells is created.
2. For each Trusted System element one element in the set Cells is built.
3. Each associated Resource set is copied to an E-Resource Set and is linked to the corresponding Cells element.
4. For all Cells elements that belong to the same TVD, add an element from type “IPC” to the E-Resource Sets associated with these Cells elements. (This essentially creates IPC channels between members of the same TVD). This is repeated for all TVDs.
5. If an element in the set Inter-Comm exists, an element of type “IPC” is added to the associated E-Resource Sets of the Cells that map to the Inter-Comm element.

In the *Write* phase, we create the OKL4 configuration file in XML format, based on the Cells set and E-Resource sets.

To automate system image installations, we have developed an algorithm based on the above model. Listing 4.1 shows briefly the algorithm for the mapping. The inputs are n TVD policies, the hardware configuration of the targeted platform, a list of necessary hardware resources for a cell, and the selection of systems. We require that the TVD policies are specified for the same hardware, that the trusted system specification is identical, and that the user's selection of compartments is defined in the TVD policies. The algorithm starts with the set building for the trusted systems. Next, the set of systems is built, based on the user's selection. Both sets then are transformed into the cell Set

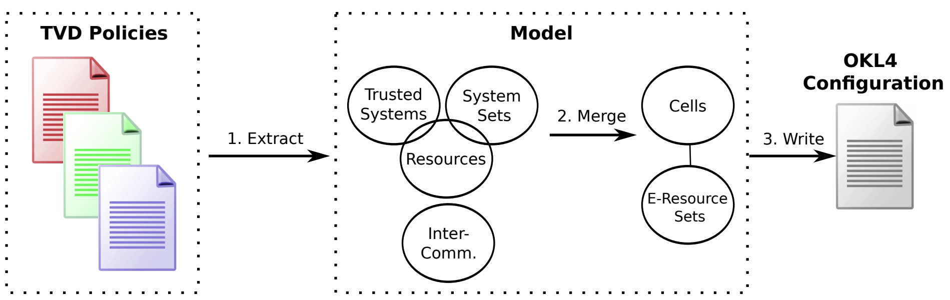

Figure 4: Policy mapping from TVD to OKL4

```
Algorithm 4.1 Merge TVD Policies to OKL4
Input: TVD policies, HW-Config, List-HW-Cell, selection
Ensure: Targeted platform is identical
Ensure: Trusted systems are identical
Ensure: selection is in TVD Policies
 1: Build set Inter-Comm. for all TVD policies
 2: Build set Trusted Systems and corresponding Resource Sets
 3: if Systems of selection are in one TVD policy then
 4:   Build Systems set and the Resource Sets for each Systems element
 5: else
 6:   Build for each TVD a Systems set based on selection. For each element assign a Resource Set.
 7: end if
 8: for all Elements in Trusted Systems do
 9:   Create element in Cells and the E-Resource set.
10: end for
11: for all Elements of Systems sets do
12:   Create element in Cells for current Systems set's element
13:   for Resources sets do
14:     Create E-Resource Sets and link them to the corresponding element on Cells
15:     if Element exists in Inter-Comm. set then
16:       Insert element into the E-Resources Set of the current Cells element
17:     end if
18:   end for
19: end for
20: Set IPC channels for each element in Cells of one TVD to another Cells element of the same TVD
Output: Microvisor Configuration = (HW-Config, microkernel resources, cell entries for all elements of $C$)
```

C. After this, the configuration file can be created. Firstly, the machine configuration (`HW-Config`) is included. Next, the microkernel resources like physical and virtual pool for the microkernel itself are written. Based on the computed memory addresses the physical pools for the cells are written as well. Based on the set C the cells are written into the configuration file.

4.2 Management Commands

The typical TVD management includes execution of such commands as install/remove/update a TVD compartment or install/remove a TVD. As we discussed in Section 3.2, due to the static property of the OKL4 configuration, these commands can be realized only via an update of OKL4 image configuration and building and installation of a complete new OKL4 system image on the mobile device. In the following, we provide description of two alternative solutions to perform this task. In the first scenario, the image is built on the mobile platform locally. However, this approach has some drawbacks as the compilation of a whole system image is a computationally expensive task. This would require significant time and, more importantly, would have negative impact on the battery life of a mobile device. Thus, we also provide an alternative solution, where the task to build the OKL4 image is delegated to a TVD Master. Generally, TVD Masters of different TVDs do not trust each other. So one TVD Master cannot trust another TVD Master to build a new image correctly. However, in use cases where all TVDs are deployed on the device are controlled by a single administrator, e.g., in case when all TVDs belong to a single company (but, e.g., handle information of different trust levels), the task of building of a new image can be delegated to a company server which represents a kind of multi-TVD Master and handles TVD Policies of different domains.

4.2.1 Image Building on Mobile Device

Figure 5 gives an overview of the general procedure of the installation of a new OKL4 image by the TVD Proxy. The user communicates with a Compartment Manager and sends a command to be executed (e.g., a command Install compartment, as depicted in Figure 5).

The Compartment Manager passes user request to the TVD Proxy, which connects via the trusted channel to the TVD Master and downloads new compartment data. After the successful download, the TVD Proxy creates a new Microvisor configuration, builds a new OKL4 image, measures it (creates reference values for subsequent measurements) and stores the image on an SD card. Next, device is rebooted. During reboot, the bootloader recognizes the new image on the SD card and loads it. When the new image is successfully loaded, the old image is deleted.

4.2.2 Image Building by TVD Master

The procedure of building the OKL4 image by the TVD Master and its delivery to the platform is shown in Figure 6. The first part of the protocol is similar to the previous one: The procedure is started by the user requiring installation of a new compartment, then user request is transferred to the TVD Proxy, and TVD Proxy tunnels the request to TVD Master via the trusted channel.

However, the second part of the protocol is different: The TVD Master does not transfer new compartment data to

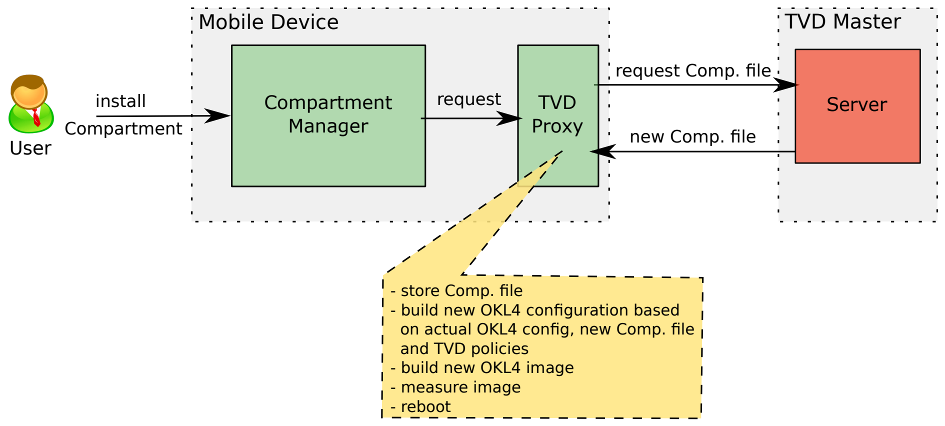

Figure 5: System Installation: OKL4 Image built by TVD Proxy

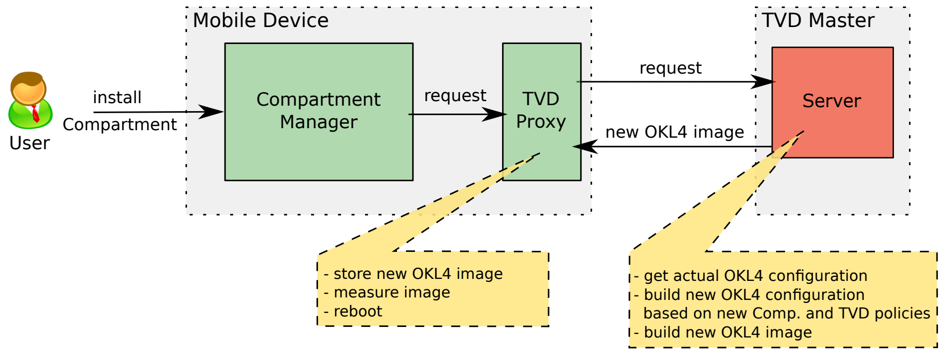

Figure 6: System Installation: OKL4 Image built by TVD Master

the TVD Proxy, but builds the new OKL4 image itself. It is assumed that TVD Master maintains TVD Policies of all domains deployed on the device, and also knows the current device configuration. Thus, it possesses all the necessary information for building a new OKL4 image. Then, the whole image is transferred to the TVD Proxy via the trusted channel. The TVD Proxy stores the image, measures it (to create new reference values) and reboots the system. The TVD Master can verify correctness of the new image via the remote attestation protocol.

In both scenarios TVD Proxy and TVD Master communicate via the trusted channel. We integrated a trusted channel establishment protocol proposed in [10] into our design. Note, in contrast to desktop solutions, a mobile device is not necessarily connected to the network for a long time. This implies that the trusted channel connection can be terminated before data is successfully transmitted from the TVD Master to the TVD Proxy. When connection is re-established, file transmission should start from the beginning, if no support is provided for resuming of the previously terminated session. In this case, the approach of building OKL4 images on the device might be preferable. It would require only to transmit smaller amount of data, particularly, a compartment file vs. a whole OKL4 image.

5. IMPLEMENTATION

In the following, we describe implementation of particular building blocks of our architecture. First, we elaborate on OKL4 development techniques, next, shortly describe the implementation of security services, and finally describe our implementation of the Policy Mapper tool.

5.1 OKL4 Development

For the development on the OKL4 Microvisor platform, we used the Open Kernel Labs SDK. It includes all necessary binaries of the OKL4 microkernel and includes a tool (called ok tool) to create bootable images. This tool takes a configuration file in XML format as input. This configuration file includes important definitions of the system like cell configurations or allocated capabilities. The ok tool merges this configuration with the corresponding binaries for each cell into a single bootable system image. This image can be booted on any supported hardware platform. The microkernel starts all specified cells and it is not possible to start or stop cells at runtime. Further, it is not possible to change the microkernel configuration at runtime.

The OKL4 configuration file (also called system configuration) consists of three parts. The first part contains the elements that are used to specify resources in the system. The second part is an inclusion of the hardware configuration file (also in XML format), which defines various aspects of the targeted platform like CPU, board, virtual and physical memory, and physical devices. The third part contains the elements describing the OKL4 cells and their assigned resources or capabilities, e.g., for IPC channels.

5.2 Security Services

We have implemented security services partly as native applications that run directly on the top of OKL4 and partly as Linux applications. Particularly, a Mobile Trusted Module (MTM), the TVD Proxy, and an Attestation Service are implemented as Linux applications that run in a small Linux OS instance, while Compartment Manager and Secure GUI are native C applications which run each in their own OKL4 cell. Communication among security services residing in different cells is performed via IPC calls. We reused the implementation of the software MTM and Attestation Service from TMD [13].

5.3 Policy Mapper

The Policy Mapper is implemented as a part of a TVD Proxy security service. It maps the TVD Policy to a valid OKL4 configuration and allows to automate the image building process. The tool realizes a mapping model presented in Section 4. The algorithm is written in Java to offer platform independence. Hence, it can be run on the mobile device or by a TVD Master. We structured the Policy Mapper in two packages called `model` and `policymapper`. The package `model` contains eight classes that represent the `model` in Figure 4. The package `policymapper` includes two classes that implement the mapping algorithm described in Section 4.1.

6. EVALUATION

In this section we evaluate the performance of our prototype implementation and discuss its current limitations.

6.1 Performance

We measured the performance for building the image on the client and the TVD Master. The client system is a Beaglebord (rev. C4)[2] and for the TVD Master we used a Dell OptiPlex 980. The Beagleboard is based on the common chip set TI OMAP3530 and the ARM Cortex-A8 CPU, which is a part of various available smartphones like Nokia N900 or Motorola Milestone. The measurement includes the following steps: (i) running the Policy Mapper tool to create OKL4 configuration file and (ii) building the system image. We tested both approaches as described in Section 4.2. We repeated the test 25 times on both systems.

The input for the Policy Mapper tool is a native application as a TVD compartment and a trusted system. The native application is a C program and has a file size of 377 kByte. The trusted system is a small Linux VM and contains the Policy Mapper tool and is 18 MByte in size, representing the TVD Proxy. For both systems we calculated the maximal theoretical time for downloading the necessary files by an Enhanced Data Rates for GSM Evolution (EDGE) connection, which is a technology to improve data transfer rates of standard GSM connection and is available in more than 170 countries. The test does not include the time that is needed for rebooting the device and running the remote attestation protocol.

Table 1 shows the performance results for building the image by the TVD Proxy. The overall time for installation of a new OKL4 image is 1130,51 seconds. As it can be seen, the most time is spent in downloading the compartment files and in compiling the new OKL4 image.

The performance results for the case when OKL4 image is built by the TVD Master are shown in Table 2. Generally, this approach is more time consuming, namely the overall required time is 2115,74 seconds. This is due to the fact that the download of the complete OKL4 image takes more time than the download of compartment files. With a faster communication channel, e.g. UMTS or WiFi, the overall performance can be improved.

[2]See: http://www.beagleboard.org

task	time required
download (EDGE)	720 sec.
create configuration	0,73 sec.
build image	409,78 sec.
overall	1130,51 sec.

Table 1: Performance results for OKL4 system installation by TVD Proxy

task	time required
download (EDGE)	2100 sec.
create configuration	0,04 sec.
build image	15,70 sec.
overall	2115,74 sec.

Table 2: Performance results for OKL4 system installation by TVD Master

Beside the time consumption, another important criterion is the power that is consumed to download the larger files. While running these tests, we noticed that 99% of computing power and just 1,6% of RAM were used. This feature was noticed for both tetsted use-cases. Better performance could be achieved by faster mobile chips like the Snapdragon or the dual core processor Cortex A9 or by optimization of the OKLabs tool, respectively the development of a tool that is designed for usage on mobile devices.

6.2 Limitations

The OKL4 Microvisor properties limit usability of the OKL4-based implementation of the presented architecture. It is not possible to add or delete new compartments during runtime, rather we have to perform it at install time. This means that all TVD commands like *Install Compartment* or *TVD Policy Update* involve the deletion of the whole system image. Hence, backup mechanism are necessary to preserve user data. Further, it is not possible to start or stop compartments on demand. All compartments are started on device upon boot up. This requires power supply and limits the realization of the TVD use cases. Lastly, building the system image costs a lot of time and computing power. This restricts the usability of the system. An optimization of the software that builds OKL4 images, or faster CPUs like Snapdragon would clearly improve the performance.

7. RELATED WORK

Trusted Mobile Desktop (TMD) [13] is a research prototype that uses virtualization to separate execution environments of a mobile platform into trusted and untrusted worlds. The implementation of TMD uses PikeOS [5], a microkernel derived from the L4 [21] microkernel family, which also supports virtualization. While the TMD provides basic functionality necessary to build a TVD infrastructure (i.e., isolated execution environments, and a secure GUI), a full

TVD realization is not provided. In contrast, our solution includes the TVD-specific policy enforcement and automatic configuration based on a policy deployment from a central management server. Instead of PikeOS, our implementation uses OKL4 [20], a microkernel-based hypervisor. OKL4 is derived from seL4 [19]. Though this does not automatically imply that OKL4 has the same security properties as seL4, we believe that a comparable effort would only be needed to verify the correctness and security of its derivations.

TrustDroid [6] is a very recent security extension for Android that targets the same security goals as TMD, however, it does not rely on a virtualization approach to achieve them. TrustDroid extends Android OS to provide isolation at different layers of the Android software stack: Android middleware and the Linux kernel. This approach is lightweight, as it does not require to run a full operating system for each domain, and also scalable, as resources required for each new domain to be created are very modest. However, it has a large TCB that includes Android middleware and the Linux kernel. Although static integrity of TCB can be guaranteed by means of secure boot, the TCB is still vulnerable to runtime attacks and thus can be compromised, e.g., though the kernel-level attacks that exploit vulnerabilities of the underlying Linux kernel[3].

Enterproid Devide [12] is also focusing on the separation of enterprise and private data on Android. It uses different profiles on an Android device to control the access to differently classified data. Moreover, each profile contains its own set of applications, and information flow between different profiles is not allowed. Since Devide is still in beta phase, it is unclear how the profiling is technically solved. In contrast of modifying Android, we enforce the isolation policy for TVDs on a lower level, i.e., at the hypervisor, and we run several instances of Android on the same device.

The Enterprise Mobility Management (EMM) [11] is a software solution for enterprises that provides data encryption, secure communication to the enterprise's network and management for mobile devices. EMM allows to define a policy that states which applications can be installed on a device and which are blacklisted. However, this approach just targets the configuration and management of mobile devices. It is not possible to prevent the leakage of information and to separate the access to differently classified data.

8. CONCLUSION

In this paper we have presented a TVD solution for mobile devices, based on the OKL4 microkernel hypervisor. Our implementation extends an existing Trusted Mobile Desktop architecture with TVD components, in particular with a TVD Proxy component that manages the TVD policy enforcement locally on a device and with a policy mapping tool that maps TVD policies to the static OKL4 configuration. Despite the static nature of OKL4 we provide a working end-user system for separating private and enterprise domains for data and applications on smartphones. However, we also showed the limitations of using OKL4 in such systems. Future work includes the integration of backup strategies for user data and the optimization of the system building software.

[3]Current reports show that Android's underlying Linux kernel suffers from various (precisely 88) bugs [4]

9. REFERENCES

[1] F. Armknecht, Y. Gasmi, A. R. Sadeghi, P. Stewin, M. Unger, G. Ramunno, and D. Vernizzi. An efficient implementation of trusted channels based on OpenSSL. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 41–50, New York, NY, USA, 2008. ACM.

[2] S. Berger, R. Cáceres, K. Goldman, D. Pendarakis, R. Perez, J. R. Rao, E. Rom, R. Sailer, W. Schildhauer, D. Srinivasan, S. Tal, and E. Valdez. Security for the cloud infrastructure: Trusted virtual data center implementation. *IBM Journal of Research and Development*, 53(4):6:1–6:12, July-August 2009.

[3] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. TVDc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, 2008.

[4] M. Broersma. Serious security bugs found in Android kernel. `http://www.eweekeurope.co.uk/news/serious-security-bugs-found-in-android-kernel-11040`, Nov. 2010.

[5] J. Brygier, R. Fuchsen, and H. Blasum. PikeOS: Safe and secure virtualization in a separation microkernel. Technical report, SYSGO, September 2009.

[6] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Scalable and lightweight domain isolation on Android. In *SPSM'11: Proceedings of the 2011 ACM workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011.

[7] A. Bussani, J. L. Griffin, B. Jansen, K. Julisch, G. Karjoth, H. Maruyama, M. Nakamura, R. Perez, M. Schunter, A. Tanner, L. V. Doorn, E. A. V. Herreweghen, M. Waidner, and S. Yoshihama. Trusted Virtual Domains: Secure foundations for business and IT services. Technical Report RC23792, IBM Research, November 2005.

[8] S. Cabuk, C. I. Dalton, K. Eriksson, D. Kuhlmann, H. G. V. Ramasamy, G. Ramunno, A.-R. Sadeghi, M. Schunter, and C. Stüble. Towards automated security policy enforcement in multi-tenant virtual data centers. *Journal of Computer Security*, 18(1):89–121, 2010.

[9] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 235–245. ACM, 2007.

[10] L. Catuogno, A. Dmitrienko, K. Eriksson, D. Kuhlmann, G. Ramunno, A.-R. Sadeghi, S. Schulz, M. Schunter, M. Winandy, and J. Zhan. Trusted virtual domains – design, implementation and lessons learned. In *Trusted Systems, First International Conference, INTRUST 2009*, volume 6163/2010 of *Lecture Notes in Computer Science*, pages 156–179. Springer, 2010.

[11] T. Digital. Enterprise Mobility Management. `http://trustdigital.com/`.

[12] Enterproid. Enterproid Devide. `http://www.enterproid.com`.

[13] Florian Feldmann Utz Gnaida, Christian Stüble, and Marcel Selhorst. Towards a trusted mobile desktop. In

TRUST2010: 3rd International Conference on Trust and Trustworthy Computing, 2010.

[14] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, M. Winandy, R. Husseiki, and C. Stüble. Flexible and secure enterprise rights management based on trusted virtual domains. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 71–80. ACM, 2008.

[15] K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *STC '06: Proceedings of the First ACM Workshop on Scalable Trusted Computing*, pages 21–24, 2006.

[16] J. L. Griffin, T. Jaeger, R. Perez, R. Sailer, L. van Doorn, and R. Cáceres. Trusted Virtual Domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability (HotDep'05)*, June 2005.

[17] G. Heiser and B. Leslie. The OKL4 microvisor: convergence point of microkernels and hypervisors. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*, APSys '10, pages 19–24, New York, NY, USA, 2010. ACM.

[18] Y. Katsuno, M. Kudo, P. Perez, and R. Sailer. Towards Multi-Layer Trusted Virtual Domains. In *The Second Workshop on Advances in Trusted Computing (WATC 2006 Fall)*, pages 133–138, Tokyo, Japan, Nov. 2006. Japanese Ministry of Economy, Trade and Industry (METI).

[19] 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 *SOSP '09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pages 207–220. ACM, 2009.

[20] O. K. Labs. OKL4 Microvisor. http://www.ok-labs.com/products/okl4-microvisor.

[21] J. Liedtke. On micro-kernel construction. In *SOSP '95: Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 237–250. ACM, 1995.

[22] H. Löhr, T. Pöppelmann, J. Rave, M. Steegmanns, and M. Winandy. Trusted virtual domains on OpenSolaris: Usable secure desktop environments. In *STC '10: Proceedings of the 5th Annual Workshop on Scalable Trusted Computing*, pages 91–96. ACM, 2010.

[23] H. Löhr, A.-R. Sadeghi, C. Vishik, and M. Winandy. Trusted privacy domains – challenges for trusted computing in privacy-protecting information sharing. In *Information Security Practice and Experience, 5th International Conference, ISPEC 2009*, volume 5451 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2009.

[24] J. Matthews. Android Migration at the Speed of Light. http://www.ok-labs.com/_assets/image_library/Android%20Migration%20at%20the%20Speed%20of%20Light.pdf, June 2009.

[25] Trusted Computing Group. Mobile Trusted Module (MTM) Specification. Version 1.0 Revision 6, 26 June 2008.

[26] Trusted Computing Group. Trusted Platform Module (TPM) Main Specification. Version 1.2 Revision 103, 9 July 2007.