JYX
JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Yehuda, Raz Ben; Leon, Roee; Zaidenberg, Nezer

Title: Arm security alternatives

Year: 2019

Version: Published version

Copyright: © The Author(s) 2019

Rights: In Copyright

Rights url: http://rightsstatements.org/page/InC/1.0/?language=en

Please cite the original version:

Yehuda, R. B., Leon, R., & Zaidenberg, N. (2019). Arm security alternatives. In T. Cruz, & P. Simoes (Eds.), ECCWS 2019 : Proceedings of the 18th European Conference on Cyber Warfare and Security (pp. 604-612). Academic Conferences International. Proceedings of the European conference on information warfare and security.

ARM Security Alternatives

Raz Ben Yehuda[1], Roee Leon[1] and Nezer Zaidenberg[2]
[1]University of Jyväskylä, Finland
[2]College of Management Academic Studies, Rishon LeZion, Israel
rabenyeh@student.jyu.fi
roee@trulyprotect.com
scipio@scipio.org

Abstract: Many real-world scenarios such as protecting DRM, online payments and usage in NFC payments in embedded devices require a trustworthy "trusted execution environment" (TEE) platform. The TEE should run on the ARM architecture. That is popular in embedded devices. Furthermore, past experience has proved that such TEE platform should be available in source code form. Without the source code 3rd parties and user cannot be conducted code review audit. Lack of review put doubt on the system as a trustworthy environment. The popular Android OS supports various TEE implementations. Each TEE OS implementation has its own unique way of deploying trusted applications(trustlets) and its own distinct features. Choosing a proper TEE operating system can be a problem for trust applications developers. When choosing TEE applications developers has many conflicting goals. The developers attempt to ensure that their apps work on as many different Android devices as possible. Furthermore, developers relay on the TEE for certain features and must ensure the suggested TEE provides all the features that they need. We survey multiple ARM TrustZone TEE operating systems that are commonly available and in use today. We wish to provide all the information for IoT vendors and SoC manufacturer to select a suitable TEE.

Keywords: virtualization, ARM architecture, TrustZone, trusted computing

1. Introduction

The proposed solutions for creating TEE on the ARM architecture are all using TrustZone™ feature. TrustZone™ is a unique privilege level on ARM (ARM 2009) whose purpose is to create a Trusted Execution Environment (TEE) (Zaidenberg 2018). Trustzone™ can be found on virtually all modern mobile phones. Additionally, TrustZone™ can be found in other ARM based systems on chip, such as AMD with their "Hiero falcon", AppleMicros X-Gene3, Cavium Thunder X and other systems.

Trust Execution Environment is required on many scenarios. TEE use cases include providing Digital right management (DRM)support. DRM requires a root of trust that can store decryption keys on the end point so that it will not be available to outsiders (Zaidenberg et al 2015b). Using virtualization to create root of Trust was attempted by SONY on PS4 and later shown on PC and MIPS by (Averbuch et al 2013). A complete system for reverse engineering protection by distribute keys for encrypted code execution after attestation (Resh et al 2017) introduced the concept of protecting code and registers by the hypervisor. (Kiperberg et al 2019) proposed creating buffers of protected code in the CPU case using memory addresses that only the hypervisor can address. Virtualization can also provide end point security (Resh et al 2017). Hypervisor can also be used for development aid by catching hypercalls (Khen et al 2011) connecting virtual debug hardware (Khen et al 2013) forensics (Zaidenberg et al 2015) and Forensic memory dump (Kiperberg et al 2019b). Last hypervisor can be used for end point attestation as shown by (Kiperberg et al 2013, Kiperberg et al 2015) etc.

Using virtualization as a tool for cyber security is also a common practice. Virtualization was initially designed for dynamic provisioning of computing resources. However, virtualization offers higher privileges and execution permissions that can used to detect threats as well as serve as TEE. ARM didn't offer virtualization or TrustZone™ support until the ARM7a hardware. ARM virtualization and TrustZone™ technologies were proposed as two optional additions. These technologies are available in some 32bit ARM7a models. ARM virtualization and TrustZone™ are now part of the 64bit ARM8a architecture and offered in all ARM8a devices.

ARM vendors offer their own TEE implementations. Some TEE implementations such as Trustonic's TSP and Qualcomm's QSEE are closed source while other are open source or offer providing the TEE source code for a fee. We survey Trusted computing alternatives for implementations. We mainly consider alternatives with available source code. All the surveryed solutions offers a complete solution for the TrustZone™ environment. We also survey some ARM virtualization (not TrustZone™) based alternatives.

2. Background

2.1 Trusted Execution Environment

The ARM architecture design allows both a Trusted Execution Environment (TEE) and a Rich Execution Environment (REE, i.e. normal ARM OS e.g. Android or iOS) to run simultaneously. The Trusted Execution Environment is a secure "area" inside a main processor not effected by the REE operating system. The Trusted Execution Environment runs its own operating system and uses its own set of register and its own memory management unit (MMU). The TrustZone™ operating system is a separate operating system that is running in parallel to the main operating system in an isolated environment. The Trusted Execution Environment guarantees that the code and the data loaded in the TEE are protected with respect to confidentiality and integrity from all application that run on the REE OS.

The Rich Execution Environment is a separate privilege level inside the main processor. The Rich Execution Environment runs its own separate operating system (compared to TEE). The Rich Execution Environment is the standard operating system that the device is running, usually the REE is Google's Android or Apple's iOS. The Rich Execution Environment offers significantly more features and applications than the TEE. This is by design and application are supposed to run on the REE and not on the TEE. As a result of offering more features, the attack surface against the REE is much larger and therefore, the REE is more vulnerable to attacks. The Rich Execution Environment receive services such as decryptions and storing decryption keys from the Trusted Execution Environment. According to ARM design the TEE acts as a monitor service for the REE.

The TEE has higher permissions than the REE as well as access to the REE memory, MMU, registers and data structures. The REE should not have access to the TEE memory and data structures. In ARM terminology, the two execution environments are called worlds, the secure world (TEE) and the non-secure world(REE). Context can be switch between secure and insecure worlds through the supervision the "Secure Monitor" running in monitor mode(TrustZone). This switch from secure to insecure world is performed through a special architecture specific assembler instruction called "secure monitor call" or smc. In order to communicate between the secure and non-secure world the user creates a shared memory segment. TrustZone™ splits the SoC device to the secure and non-secure worlds. TrustZone™ controls all the device hardware interrupts. TrustZone™ can route any interrupt to the secure world or to the non-secure world. Like in the memory case, I/O and interrupts routing may change dynamically. TrustZone™ uses its own MMU. Operating systems and Processes that execute in TrustZone™ do not share the same address space with their non-secure world counterparts. Thus, there is no need to have distinct TrustZone™ for each processor. A single TrustZone™ OS can run across multiple ARM processors/cores and manage all the device trusted computing needs. The ARM architecture cryptographic keys are accessible only in TrustZone™, The manufacturer can provide each CPU or platform with device specific keys using e-fuses. These keys are device specific, thus enabling protection in the end unit granularity level.

(For example distributing video that only specific device can decode etc.)

Booting a Trusted Execution Environment must form a chain of trust in which a trust nexus verifies the next component on the boot chain. Each component verifies the next component until the system.

2.2 ARM permission model

The ARMv8 architecture has a unique approach to privilege levels. The ARM platform normally has 4 exception (permission) levels.

ARM also has secure world (TrustZone™) and normal world (non TrustZone™)

ARM Exception levels are described in Table 1 Each of the exception levels provide its own state of special purpose registers and can access these registers of the lower levels but not higher levels. The general-purpose registers are shared.

Thus, moving to a different exception level on the ARM architecture, does not require the expensive context switch that is associated with the x86 architecture.

Table 1: Arm exception levels

Exception level	Meaning	Notes
Exception Level 0 (EL0)	Refers to user space code.	Exists in both secure and normal world This is analogous to "ring 3" in x86 platform.
Exception Level 1 (EL1)	Refers to operating system code.	Exists in both secure and normal world This is analogous to "ring 0" in x86 platform.
Exception Level 2 (EL2)	Refers to HYP mode. ARM hypervisor privilege level	Exists in both secure and normal world This is analogous to "ring -1" or "real mode" on the x86 platform.
Exception Level 3 (EL3)	TrustZone™	Refers to TrustZone™ as a special security mode that can monitor the ARM processor and may run a security real time OS. There is no direct analogous modes but related concepts in x86 are Intel's ME or SMM. Naturally it exists only on secure world

ARM7 architecture is similar to ARM8. ARM7 offers virtualization as an extension that is only available to some late ARM7 models. ARM7 also offer TrustZone™ as separate extension. Furthermore, ARM7 is 32bit architecture while ARM8v is 64bit (and 32bit) architecture.

3. Virtualization vs. TrustZone™ mode

The first question we must address is how the operating system should be verified. The REE operating system can be verified using HYP mode or TrustZone™. ARM has designed the TrustZone™ mode specifically for attesting and monitoring the Rich operating system. The benefit of using TrustZone™ is that it reserves the HYP mode for real hypervisor without the need to use features such as nested virtualization. Furthermore, only the vendor can install software on the TrustZone™ mode. In some cases, even the vendor (i.e. the manufacturer of the device or phone, not the CPU vendor) has limited access and cannot install software in TrustZone™ mode. However, no such limitation exists on HYP mode. Everybody can install software in HYP mode with no special limitations. This usually makes hypervisor code easier to install. From the device vendor standpoint therefor it is assumed that TrustZone™ is available. (or possibly available) From software vendor standpoint TrustZone is not available but virtualization may be.

The two main drawbacks of using virtualization are that virtualization mode is no longer available for other software that may want to run there. Also, TrustZone™ is monitored on boot by the BSP (Board Support Package) it cannot be modified or replaced as easy as the hypervisor boot loader or driver.

Resh et al (2017) and Seshadri et al (2007) both provide examples of using hypervisor for end point security.

We examine several hypervisor implementations for completion However it is assumed that a TrustZone™ solution is preferable whenever TrustZone™ is available.

3.1 Virtualization classification

Virtualization is the process of running multiple Operating system on a single hardware or running microkernel to manage single operating system. Hypervisor is the software that provides virtualization. Hypervisors are classified to two modes:

- 1. Full virtualization - The guests operating system is not modified in any way.
- 2. Para-virtualization – The guest operating system is aware it run as guest. The guest's operating system code is modified. The guest operating system does not attempt to communicate directly with the hardware. Instead the guest operating systems uses hypercalls to communicate with the host hypervisor. When the guest's operating system needs the host for example for I/O access and sometimes in critical sections. The hypercalls trap to the hypervisor to perform a service on behalf of the guest. Using para-virtualization and hypercalls usually yields better performance.

In the taxonomy of virtualization environment, virtualization environments are categorized by their design.

- 1. Complete monolithic - A single software responsible to provide access to the hardware to the guests. For example, VMware ESXi server.

- 2. Partially monolithic - The technology is an extension to the general-purpose operating system, such as KVM in Linux and VMWare Desktop or Microsoft Hyper-V
- 3. MicroKernel These are light weight micro kernels that a minimal set of services to the guests, mainly CPU virtualization and hardware access. Xen and seL4 are examples for such micro hypervisors.

Last virtualization pioneers Popek and Goldberg (Popek et al 1974) classified hypervisors to type I and type II

Type I hypervisors (or boot hypervisors) – are hypervisors that start at boot and start various guest operating systems. Examples include VMWare ESXI, Xen and IBM S/390 VM.

Type II Hypervisors (or hosted hypervisors) – are hypervisors that starts under a host operating system that already booted and took control of the machine. Examples include VMWare desktop.

For security and trusted computing purposes only Type I hypervisors are of interest. Type II hypervisors can be disabled by the host OS and thus serve no security purpose.

4. Alternatives review

4.1 GlobalPlatfrom

GlobalPlatform is a non-profit organization that consist of an alliance of many mobile device manufacturers. GlobalPlatform defines and publishes the standards for mobile devices including a standard for secure digital services for mobile devices. GlobalPlatform (2011) is the current industry standard for TEE platform under ARM.

4.2 General Dynamics OKL4

OKl4 is a microkernel that was originally developed, maintained and distributed by Open Kernel Labs.

The OKL4 operating system was based on the L4 operating system by Liedtke (Liedtke 1996).

The L4 microkernels family in its earlier form was called L3. L3 was a microkernel that was developed Liedke in the 1980's on an i386 system and was deployed in few thousands' installations, mainly education institutes. L3 suffered from a high overhead of Inter-process communication (IPC) communication which was over 100us. Liedtke, trying to reduce the IPC overhead problem, had re-implemented L3 completely and reduced significantly the IPC overhead to 5us, on i486. This new design was referred as L4.

L4 had evolved over the years and become a family of L4 microkernels, to name a few, L4-embedded, Codezero, NICTA, sel4 etcetera. NICTA was maintained by OpenLabs which renamed it to OKL4 microkernel and stopped the open source development.

OKL4 (Heiser et al 2010) is deeply discussed in peer reviews press. The OKL4 micro-visor supports both paravirtualization and pure virtualization. It is designed for the IoT industry, and supports, ARMv5, ARMv6, ARMv7ve and ARM8va. OKL4 is focused on embedded devices. OKL4 was originally open source software.

On 2012 general dynamics acquired the Open Kernel Labs. After being acquired General Dynamics changed OKL4 source code policy from open to closed source project. The latest available open source OKL4 is from May 2013 and is still available to download from archive.org (and other sources). OKL4 has a sister open source project (supported by GeneralDynamics) called seL4 which is described later.

Installing OKL4 and running it is a challenging task that requires expertise. Open source OKL4 code must also be adapted to modern hardware. Today OKL4 is still under development and support of General Dynamics.

One can also obtain the current OKL4 source code under suitable license and NDAs.

We refer to the latest available open source OKL4 from 2013 and not to current releases (for which source code is not available) thus we are not up to date with current releases (compared to other Trusted Execution Environment alternatives).

4.3 Google Trusty TEE

Trusty is a secure Operating System (OS) that was developed by Google.

Trusty provides a Trusted Execution Environment (TEE) for The Android (only) Operating system.

The Trusty OS doesn't require security specific hardware. Instead, Trusty TEE runs on the same processor as the normal Android OS, However, despite running on the same CPU, Trusty is isolated from the rest of the system. This is done using ARM TrustZone™ features that enable separate MMU for trusty (in TrustZone™) and the normal world OS. TrustZone™ allows Trusty to create an isolated secure execution environment and provide certain services to the non-secure (i.e. Android) OS.

Trusty consists of:

- A small operating system kernel. The trusty Kernel is derived from Little Kernel. Little Kernel is a small operating system that is also used as Android boot loader.
- A Linux kernel driver that acts as mediator between the TEE (Trusty) and REE (Android) environments
- An Android user space library that provide to communication between the REE (Android) and TEE (Trusty) applications using the kernel driver

Trusty is compatible with ARM and Intel processors. On ARM systems, Trusty uses ARM's Trustzone™ to virtualize the main processor and create a secure trusted execution environment. Similar support is also available on Intel x86 platforms using Intel's Virtualization Technology.

4.4 Linaro OP-TEE

Linaro security working group and STMicroelectronics have teamed to create OP-TEE.

OP-TEE follows GlobalPlatfrom specification (2011) and implements version 1.1 of GlobalPlatform TEE client API and TEE internal API. OP-TEE is an open source project and is widely available under BSD 2-clause license and (Kernel parts) GPLv2 license. According to (Bech 2014) and our testing OP-TEE has a small foot print and minimal effect on the running system. OP-TEE has a large community support. Like Trusty above OP-TEE consists of 3 main components.

- A light weight secure operating system. The OP-TEE operating system consists of several modules such as memory management, interrupt handling, etc. In addition, OP-TEE implements a hardware abstraction layer as it supports various processors and hardware. The OPTEE operating system also provides a capability to run user-space applications (typically referred to as Trustlets) in the secure world. These applications are provided with the GlobalPlatform TEE Internal API which allows them to ask for internal, secure-only, OS services
- A non-secure user-space client that is composed of two components: (1) a user-space/kernel-space mediator and (2) libraries that implement the GlobalPlatform TEE Client API.
- A kernel driver that simply performs the transitions between the secure and non-secure worlds

5. Other alternatives

There are other TEE options that the vendor can use for both EL2 and EL3. These alternatives are not as common or as strong as the alternatives mentioned above and fail in atleast one pre-requisite.

5.1 Jailhouse

Jailhouse was announced by Siemens in November 2013. Baryshnikov (Baryshnikov 2016) has analyzed the Jailhouse system. Jailhouse is a type 2 partitioning microvisor for Linux hosts.

A partitioning microvisor is a microvisor that controls the OS access to resources and isolates the resources from the general-purpose operating system or other guests. Partitioning in microvisor context means the microvisor performs strict allocation of the system resources. The hosting Linux is referred as the Root cell, and the guests are called inmates. Jailhouse itself is not an operating system, it is a resource access controller. Jailhouse is

controlled from the Linux host, and reveals information stored to the Linux host (the root cell), but not the guests (the inmates).

Jailhouse is a bare metal hypervisor, and in most cases, it is pure virtualization hypervisor, and as such can run many types of operating systems, such as FreeRTOS (Barry 2008), Erika3 (Evidence 2019)``, Linux and Zephyr. Jailhouse supports ARMv8, ARMv7a, and x86_64 architectures. Jailhouse requires the machine to have at least two processors. One processor is used to run the hosting Linux, and the other processors may be assigned to Jailhouse. Jailhouse requires the Linux kernel to provide a contiguous memory at boot time. It requires a memory footprint of few tens of megabytes, usually 50 megabytes.

The Jailhouse configuration is performed through a tool provided by Jailhouse. This tool scan sysfs and procfs, and generates a device tree that describes the hardware as seen by the Linux host. This Jailhouse device tree is referred as the cell configuration file. The user can edit the cell configuration file to create a correct guest configuration. Jailhouse targets the automation, robotics and IoT industries.

5.2 QSEE

QSEE is Qualcomm Secure Execution Enviornment. In the past it was based on OKL4 until GeneralDynamics and Qualcomm failed to reach agreement regarding licensing. Since 2015-6 Qualcomm has developed QSEE from scratch with no (direct) connection to GeneralDynamics.

QSEE is closed source (and Qualcomm does not provide source code licenses) Therefore QSEE fails our precondition of source availability and is not part of this survey. Prior releases of QSEE suffered from several well documented security problems.

(Beniamini 2016)

5.3 seL4

sel4 like OKL4 is also a based on the L4 microkernel. seL4 (Klein et al 2009) is a microvisor that was implemented by Open Kernels labs (and later GeneralDynamics). seL4 is not as popular as OKL4. One of the strong features of seL4 is the fact that it has been formally verified to be correct. The method of verification is definition of seL4 exact functional specification and proving its operations using rigorous logical means. Later the seL4 codebase was reimplemented in C to make it efficient. Despite the rigorous testing seL4 is not necessarily bug free. The implementation has some assumptions of correctness about the compiler, architecture and C reimplementation.

seL4 compared to OKL4, is an open source kernel. seL4 is the sole kernel that is mathematically proven secured and safe. L4-embedded, or NICTA embedded, was adopted by Qualcomm as a real time operating system for their wireless modem processors firmware.

The basic rules of the L4 kernel design are minimalism. Leidtke (1996) formulated the rule of minimization as follows:

> *"A concept is tolerated inside the u-microkernel only of moving it outside the kernel, i.e. permitting competing implementation would prevent the implementation of system required functionality".*

This principle, known also as the no-policy in the kernel is the core of the L4 microkernels design. Though operating systems tends to grow in size over the years, L4 footprint is considerably low. seL4 footprint is 9600 lines of code. As a side effect of the minimization and performance, L4 microkernels, do not strive to hardware abstraction. Half of the seL4 microkernel is agnostic to the underlying hardware.

L4 also demonstrates a new resource management scheme where all memory allocations are user space driven. Another interesting feature of the L4 microkernels, is the fact that interrupts are disabled while executing in kernel mode. This approach simplifies the implementation, increases the performance and eases the kernel verification. Direct process switch, which in general means that sel4 tries to avoid from using the scheduler, is another interesting facet of L4. When a thread reaches a pre-emption point, the kernel switches to the first runnable thread, which in turn, executes on the time slice of the pre-empted thread.

seL4 runs on ARM, supports SMP and Uniprocessor. Like OKL4 seL4 also provides real-time support.

seL4 was recently ported to ARM8 architecture (Heiser 2019)

5.4 TrustTonic

TrustTonic is known for its TrustZone technology in the mobile world, mainly Android. TrustTonic operating system, Kinibi, is a closed source operating system. Kinibi is wide spread in the Android cellphone world. Kinibi provides data encryption and device authentication. It also gives safe access to peripherals, such as the touch screen, NFC and finger print reader, through its TEE. Since peripheral I/O is provided using the TEE no malware in the REE will affect the I/O. In addition, Kinibi can isolate sensitive code execution and secure data.

Kinibi is verified by the chain of trust, i.e; it is verified by the bootloader each time the device boots.

Furthermore, TEE application has access to the network. This way, a trusted application can access remote services securely.
TrustTonic can also be found in the automotive industry. In this area, TrustTonic approaches data leakage, application overlapping and application re-packaging attacks. Application overlapping attack is an interception technique for stealing sensitive I/O, such as when a user enters its password. A repacking of an application is a method of modifying a program to steal sensitive data. For example, adding a log entry that prints sensitive information.

Trustonic offers an SDK, compliant with GlobalPlatform API standards, to help build Trusted Applications.

Since no open source version of Kinibi exists (not even older version) we left it out of this survey.

5.5 Xen

Xen was announced in 2003. Xen was developed initially at the university of Cambridge, by Ian Pratt. Xen is a micro kernel hypervisor. Xen provides CPU virtualization using virtual interrupts,MMU and under-guest communication. In Xen, a virtual machine is referred as Domain. Domain0, also known as Dom0, is the first domain. Dom0 must run before any other virtual machine. Dom0 is usually Linux or BSD, and DomU is a virtual machine on top of the other domains. Domain0 requires access to the entire machine's hardware. Domain0 responsibility is management through the Linux kernel. Xen's event channel provides communication between Dom0 and DomU. Whenever DomU issues a virtualized event it uses this event channel. The event channel is used for para-virtualized guests. For a full virtualized guest Xen uses QEMU. Xen's tool stack is the management tool used to control guests. The fact that Xen uses Linux as Dom0 provides Xen with abundant of hardware support and Linux software. Xen boots from the bootloader and is then loads the a para-virtualized host.

Xen's I/O virtualization comes with a performance cost. Virtualized I/O accesses and virtualized interrupts from DomU Xen guests are delegated to Dom0. In addition, if a host interrupt occurs while DomU runs, then this interrupt would be served only when Dom0 is gets the processor. Thus, interrupts and events have a performance overhead.

Xen is available in ARM and x86, runs on SMP and UP. Xen is licensed GPL.

5.6 Xvisor

Xvisor was announced in Apr 2012. Xvisor (Patel et al 2015), is a monolithic, type 1 hypervisor that is independent of Linux. Xvisor is a monolithic hypervisor that controls the hardware peripherals. Xvisor provides a minimal operating system, thus Xvisor is not a microkernel. Xvisor can emulate devices and provides a path-through access to real devices. As an operating system, Xvisor has a memory management, scheduler, load balancer and threads. Xvisor does not support processes and is not POSIX compliant. Xvisor supports SMP, so that a guest can use two or more processors. There are no restrictions on the number of processors, and XVisor can also execute on a single processor.

Xvisor provides an IPC between two guests through the use of aliased guest region, which is

a GPA (guest physical address) shared between two guests. In the Xvisor taxonomy, a processor can be Normal VCPU or an Orphan VCPU.

Normal VCPUs serve guests OSes, and Orphan VCPUs belongs to the hypervisor. Xvisior support ARM 32bit and 64bit and x86. Its footprint is less than 10MB, however, since it is a type 1 hypervisor, it is required to change the boot loader. Xvisor is widely targets the infotainment market in the automobile world, and automation in general. It is licensed GPL.

6. Discussion

We surveyed most well known TrustZone and HYP alternatives. If the user requires open source and community review as security requirements then both trust-tee and OP-TEE provide good alternatives.

OP-Tee provides the benefit of following GlobalPlatform standard and is more light weight and provide less features than google Trust-Tee. OP-Tee has been an open source OS but the source is only available to paying customers. However, OP-Tee (and seL4) offer real time support that are required for some systems. We also reviewed seL4 which is also open source. seL4 is an option but is not as widely used as OP-Tee and Trust Tee (or OKL4).

Furthermore, TrustZone™ and other extensions can be used for other means such as real-time processing (Ben Yehuda et al 2018) and control flow analysis (Abera et al 2016). Using the TrustZone™ architecture for other purposes is an interesting future research area

7. Conclusion

We surveyed the popular TEE alternatives available today. Each alternative has its own benefits and drawbacks. SoC vendors and Platform manufacturers can choose the desired implementation based on their requirements and preferences. Out of the popular alternatives we believe that the free alternatives Google TrusTEE and OP-TEE offer sufficient features and fair replacement for OKL4. We believe OKL4 has none security benefits in strict real time environments that are beyond the scope of this review.

References

Abera, T., Asokan, N., Davi, L., Ekberg, J. E., Nyman, T., Paverd, A., Sadeghi A.R & Tsudik, G. (Abera et al 2016). C-FLAT: control-flow attestation for embedded systems software. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 743-754). ACM.

Averbuch, A., Kiperberg, M., & Zaidenberg, N. J. (Averbuch et al 2013). Truly-protect: An efficient vm-based software protection. *IEEE Systems Journal*, *7*(3), 455-466.

ARM, Architecure (ARM 2009). Security technology building a secure system using TrustZone™ technology (white paper). ARM Limited.

Barry, R. (Barry 2008). FreeRTOS. *Internet, Oct*

Baryshnikov, M. (Baryshnikov 2016). Jailhouse hypervisor (Bachelor's thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum.).

Bech, J. (Bech 2014). OP-TEE, open-source security for the mass-market. Core Dump. https://www.linaro.org/blog/op-tee-open-source-security-mass-market/

Ben Yehuda R. and Zaidenberg N. J (Ben Yehuda et al 2018) Hyplets - Multi Exception Level Kernel towards Linux RTOS In the Proceedings of the 11th ACM International Systems and Storage Conference Systor 2018 pp 116-117

Beniamini G. (Beniamini 2016) "Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption". https://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html

Evidence Srl (Evidence 2019) "ERIKA Enterprise 3 source code" https://github.com/evidence/erika3

Heiser, G., & Leslie, B. (Heiser et al 2010). The OKL4 Microvisor: Convergence point of microkernels and hypervisors. In Proceedings of the first ACM asia-pacific workshop on Workshop on systems (pp. 19-24). ACM.

Heiser, G (Heiser 2019) "Whats new in the world of seL4" FOSDEM 2019 https://www.youtube.com/watch?v=6s5FDX5PkZI

Khen, E., Zaidenberg, N. J., & Averbuch, A. (Khen et al 2011). Using virtualization for online kernel profiling, code coverage and instrumentation. In *2011 International Symposium on Performance Evaluation of Computer & Telecommunication Systems* (pp. 104-110). IEEE.

Khen, E., Zaidenberg, N. J., Averbuch, A., & Fraimovitch, E. (Khen et al 2013). Lgdb 2.0: Using lguest for kernel profiling, code coverage and simulation. In *2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)* (pp. 78-85). IEEE.

Kiperberg, M., & Zaidenberg, N. (Kiperberg et al 2013) Efficient Remote Authentication. In *Proceedings of the 12th European Conference on Information Warfare and Security: ECIW 2013*(p. 144). Academic Conferences Limited.

Kiperberg, M., Resh, A., & Zaidenberg, N. J. (Kiperberg et al 2015). Remote Attestation of Software and Execution-Environment in Modern Machines. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 335-341). IEEE.

Kiperberg M, Algawi A, Leon R, Resh A. & Zaidenberg N. J. Hypervisor-assisted Atomic Memory Acquisition in Modern Systems (Kiperberg et al 2019b) in Proceedings of 5th international conference on information system security and privacy ICISSP 2019

Kiperberg, M., Leon, R., Resh, A., Algawi, A., & Zaidenberg, N. J. (Kiperberg et al 2019). Hypervisor-based Protection of Code. *IEEE Transactions on Information Forensics and Security*.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe D., Engelhardt K., Kolanski R. ,Norrish M., Sewell, T., Tuch H. & Winwood S. (Klein et al 2009) . seL4: Formal verification of an OS kernel. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (pp. 207-220). ACM.

Liedtke, J. (Liedtke 1996). Toward real microkernels. Communications of the ACM, 39(9), 70-77.

Patel, A., Daftedar, M., Shalan, M., & El-Kharashi, M. W. (Patel 2015). Embedded hypervisor xvisor: A comparative analysis. In Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on (pp. 682-691). IEEE.

Popek, G. J., & Goldberg, R. P. (Popek et al 1974). Formal requirements for virtualizable third generation architectures. Communications of the ACM, 17(7), 412-421

Resh, A., Kiperberg, M., Leon, R., & Zaidenberg, N. (Resh et al 2017b). System for Executing Encrypted Native Programs. *International Journal of Digital Content Technology and its Applications*, *11*

Resh, A., Kiperberg, M., Leon, R., & Zaidenberg, N. J. (Resh et al 2017). Preventing Execution of Unauthorized Native-Code Software. International Journal of Digital Content Technology and its Applications, 11.

Zaidenberg, N. J. (Zaidenberg 2018). Hardware Rooted Security in Industry 4.0 Systems. *Cyber Defence in Industry 4.0 Systems and Related Logistics and IT Infrastructures*, *51*, (pp 135-151).

Zaidenberg, N. J., & Khen, E. (Zaidenberg et al 2015). Detecting Kernel Vulnerabilities During the Development Phase. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 224-230). IEEE

Zaidenberg, N., Neittaanmäki, P., Kiperberg, M., & Resh, A. (Zaidenberg et al 2015b). Trusted Computing and DRM. In *Cyber Security: Analytics, Technology and Automation* (pp. 205-212). Springer, Cham.

View publication stats