Thesis: Outline

Oberon Swings

April 27, 2022

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 2
- 1.2 Contributions . . . 2
- 1.3 Outline . . . 3

2 Background **4**
- 2.1 Remote Attestation . . . 4
- 2.2 Trusted Execution Environment . . . 6
- 2.3 ARM TrustZone . . . 7
- 2.4 PinePhone . . . 9
- 2.5 OP-TEE . . . 9
- 2.6 Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes [Lin+21] . . . 9

3 Method **13**
- 3.1 Detailed Problem . . . 13
- 3.2 System Model . . . 14
- 3.3 Attacker Model . . . 14
- 3.4 Solution . . . 14

4 Implementation **15**
- 4.1 Attestation TA . . . 15

5 Experiments **17**
- 5.1 Performance . . . 17
- 5.2 Performance Evaluation . . . 17
- 5.3 Security Properties . . . 18
- 5.4 Security Evaluation . . . 18

6 Discussion **19**
- 6.1 Related work . . . 19
- 6.2 Comparison of Approaches . . . 19
- 6.3 Future Improvements . . . 20

7 Conclusion **21**

Chapter 1

Introduction

Smartphones

Everyone is assumed to have a smartphone of their own because it has become an essential gadget for our day to day lives.

Everywhere these smartphones can be spotted, people use them at home, on the bus or even at work.

Personal Computers seem less and less important to some people because all their needs can be fullfilled with a smartphone.

Sensitive data

Making connections that is what these smartphone devices are good at but it is important to keep in mind that there are certain security implications with every connection that is made.

Personal data is what drives the interactions with the smartphone. People use them to do online banking, send mails or even consult health related reports.

Related to IoT

Hardware similarities between smartphones and IoT devices are more prominent than one might expect. This is due to the fact that smartphones actually stem from IoT devices and not from personal computers.

Security features for IoT devices is a hot research topic, this is because not many solutions or standards that are present today are found to be adequate in terms of protection against the existing threats.

1.1 Problem statement

IoT security

Performance is the main focus when IoT devices are designed, they only have a small number of tasks but these need to be executed as fast or as energy efficient as possible.

Dynamic and distributed groups of devices form an IoT network, this implies that the security of this network is as strong as the weakest link in the chain (which can be very weak in the IoT environment).

Progressive functionality

Banking, e-Health and mails are applications most people trust their smartphone with, even though personal computers weren't even trusted with these kinds of functionality a few decades ago.

Missmatch

Progressive functionality is what smartphones have been bringing for their users and is what makes that the industry keeps growing every year.

Lagging or lacking security is the downside of this push for better functionality because it is very hard to make certain functionality secure but companies want to be the first to bring their products to the market.

1.2 Contributions

Reproduction of existing work is the first goal of this thesis, the solution of the paper is replicated as closely as possible (no source code available). The experiments in the paper are redone to be able to make a comparison between the original and the implementation discussed here.

Open source code is very important in the field of computer science because it allows other researchers to reproduce the experiments and review the work that has been done in a detailed manner.

Extra experiment measurements are executed to expand on their work and give a more complete picture of the solution.

Comparison with similar solutions is of course also important to evaluate whether this way of trying to solve the problem is in the right direction or whether different solutions have achieved more promising results.

1.3 Outline

In the next chapter more background information about among other things Remote Attestation and ARM TrustZone will be given. In the third chapter the methods to solve the problem will be explained. In chapter four the implementation of the attestation program are elaborated upon. In the fifth chapter the goal and outcome of the experiments will be made clear. The sixth chapter will conclude this thesis informing the reader about related work and future improvements. The final chapter will discuss the completed work.

Chapter 2

Background

2.1 Remote Attestation

Goals

Supplying evidence about a target to a verifier is the most important goal for remote attestation. [Cok+11] defines attestation as follows:

> "Attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim."

Making a claim in this context refers to stating whether the target is in a secure state or not, this is often done implicitly. The appraiser can be seen as the verifier, it receives the evidence (and the claim) and will decide based on those whether the claim is valid and the target is still trusted. Remote attestation is achieved when the verifier is a remote service provider which is accessed through a network.

Requirements

The target must adhere to certain constraints to provide the necessary abilities for correct attestation. First of all a trusted base is needed that can enforce an isolation mechanism to avoid it being tampered with. This isolation will make sure that the attestation can be executed even when the target is unreliable. Another important aspect of the attestation is the ability to measure useful aspects of the system, this means that there needs to be structure to these measurements to be able to understand them. When these measurements are requested, they need to be executed in a trusted manner and the results need to be sent to the verifier securely.

The verifier needs comprehensive and fresh information about the target to be able to correctly attest it. Based on this information the verifier makes

decisions on the reliability and trustworthiness of the target. These decisions are referred to as attestations, it should be possible to draw conclusions from multiple attestations or make predictions based on them. Besides a complete set of information about the target the verifier also needs proof of the trustworthiness of this information because it is used as evidence for the decision making process.

Techniques

Integrity Measurement Architecture implements attestation by measuring the code of programs before running them on the target. The verifier can check whether the program's code has been modified based on these measurements [YF08] [JSS06] [Dua+20] [KBC21]. According to [Ala+12] and many others Integrity Measurement Architecture (IMA) is inflexible and static because updates for programs are hard to take into account. The solution is also very limited because there are a variety of ways a program can misbehave without the code base being tampered with.

Attestation on Program Execution is an important step in the right direction, it measures the dynamic behavior of the program. [Gu+08] propose to observe the system calls the program makes to verify whether it adheres to the permitted control flow. Although it is a big improvement there are still weaknesses like the granularity of a system call might not give enough details and the behavior of a system is much more than the system calls alone. Many similar solutions that focus on the dynamic behavior of code have been proposed all with their weaknesses and shortcomings [Qin+20] [Ali+17] [SKU10] [Ba+17].

Combined strategies are being proposed more and more often due to them providing protection against a wider variety of vulnerabilities. Model-based Behaviour Attestation was proposed by [Ala+08] which is again mentioned by the same researchers in their analysis about existing techniques [Ala+12]. The platform is expected to enforce a certain security model and the attestation will verify whether the platform behaves accordingly. The behavior of the platform is monitored by a variety of techniques like IMA or Property Based Attestation (PBA) for a 'full picture' approach. PBA is focused on properties that the platform possesses, it is still very hard to map configurations of the platform to certain properties but it does provide lots of flexibility in terms of attestation. [MNP16] have also combined a variety of approaches to attest the trust of IoT devices and implemented multiple modules that attest certain platform properties based on different measurements.

2.2 Trusted Execution Environment

Definitions

A definition of a Trusted Execution Environment is given by [SAB15]:

> "Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting back door security flaws are not possible."

The requirements to achieve a secure and trusted execution environment are largely accomplished by the separation kernel. This kernel simulates a distributed system which divides the system into strongly isolated partitions with different security levels. For instance data in one partition cannot be leaked by shared resources because they are sanitized and cannot be read or modified by other partitions. A partition also needs to give explicit permission before others are able to communicate with it and a security breach in a partition cannot impact any other partitions.

Trust is a very important aspect of a TEE, there are multiple types of trust with different origins. Static trust is measured only once, before deployment in most cases and assumed to never change during the lifetime of the device. Dynamic trust on the other hand is based on the state of the system and this state changes continuously. In this latter case the trust needs to be measured periodically to have an up to date view on it. To be able to do these measurements a trusted entity is required, this is because trust cannot be created but needs to be transferred from the Root of Trust (RoT) to the component that is being measured. To reach this final target, a chain will be created which links intermediate components by having the next one build upon the trust of the previous component and this is how a Chain of Trust (CoT) is constructed.

Building blocks

The Root of Trust is the starting point for the secure boot process which assures that only code with certain properties is given control. For instance during the boot process checking the integrity of the succeeding component before loading it and giving it control will construct a Chain of Trust. This

CoT is necessary because that is the basis of the trust of the separation kernel. This RoT is often implemented using some hardware component that is trusted [MAT+21] [Fot+21] [Kin06] [Zha+21].

The separation kernel is a very important component of the architecture because it is responsible for the secure scheduling and information flow control. The secure scheduling makes sure that the TEE doesn't affect the rich OS too much to allow the latter to remain responsive and meet real-time requirements. Information flow control is tightly coupled with the inter-environment communication, this is an interface which allows communication between TEE and the rest of the system. To avoid security risks from this communication it needs to adhere to the following guidelines: reliable isolation, minimum overhead and protection of the communication structures. As shown in these papers [Van+19] [Guo+21] [KM20] [Mac+17] security risks in implementations of these components do exist and they need to be dealt with to make sure the TEE operates as it is expected to.

The TEE is where Trusted Applications (TA) run, it also has a trusted kernel which is kept as minimalistic as possible. The minimalism of the kernel is to avoid software bugs which could introduce security vulnerabilities. The trusted kernel provides services like secure memory and trusted I/O which are important features to allow the system to be used in a secure manner. Secure memory ensures confidentiality, integrity and freshness of stored data. The trusted I/O protects authenticity and confidentiality of communication between TEE and peripherals.

2.3 ARM TrustZone

Core principles

ARM TrustZone [ARM22] implements the TEE on the processor level which means it runs below the hypervisor or OS [PS19]. This approach divides the system into two main partitions namely the Normal World (NW) and the Secure World (SW). The processor executes either in the NW or in the SW, these environments are completely isolated in terms of hardware and the SW is more privileged to make sure the NW doesn't behave in a malicious way. The partition between these worlds gives rise to new and better security solutions for applications running on these types of System on Chips (SoC) [Len+18] [Esk+18] [Cha+17] [Fer+17]. The SW can provide services like data storage, I/O and virtualization all with hardened security guarantees because of these hardware features.

Implementation

The Normal and Secure World are the two main environments in which the processor will be executing code. The Normal World shelters the rich OS (like Linux), it is mainly due to the size of these operating systems that they cannot be trusted to run in the secure world. The risk of there being implementation bugs that introduce security risks is too high. Also user level applications run in the NW, for peripherals for instance they rely on the rich OS and depending on the service the rich OS relies on the Secure World, some services can also be requested from the user application directly to the SW. The Secure World is where the trusted kernel runs, the implementation of this component is kept very minimal and needs to be designed and implemented very securely to avoid vulnerabilities.

The NS-bit is the 33rd bit (in a 32-bit architecture) that flows through the entire pipeline and can be read from the Secure Configuration Register (SCR) to identify the world in which the operation is being executed. The processor has a third state which is the monitor state, this is necessary to preserve and sanitize the processor state when making transitions between NW and SW states. The new privileged instruction Secure Monitor Call (SMC) allows both worlds to request a world switch and the monitor state will make sure this is handled correctly. The only other way of getting into the monitor state is with exceptions or interrupts from the Secure World.

TZ Address Space Controller (TZASC) can be used to configure specific memory regions as secure or non-secure, such that applications running in the secure world can access memory regions associated with the normal world, but not the other way around. Making these partitions is also performed by the TZASC which is made available through a programming interface only available from within the secure world. A similar approach is taken for off-chip ROM and SRAM, this is implemented using the TrustZone Memory Adapter (TZMA). Whether these components are available or not and how fine grained the memory can be partitioned depends on the SoC because they are optional and configurable.

TZ Protection Controller (TZPC) is in the first place used to restrict certain peripherals from worlds, for instance to only allow the secure world to access them. It also extends the Generic Interrupt Controller (GIC) with support for prioritized interrupts from secure and non-secure sources. This prioritization is important to avoid Denial of Service (DoS) attacks on the secure world.

2.4 PinePhone

ARM TrustZone is used in the System on Chip (SoC) of the PinePhone which is a popular solution amongst smartphone suppliers, Samsung KNOX [Sam22] and Android KeyStore [Goo22] are two examples of this. It is evident that this solution has a lot of potential and the industry believes in it's potential but often the technical details are not disclosed which gives the academic world little opportunities to build upon this technology [PS19].

Open source smartphone is the best way to describe the PinePhone, this is a powerful tool for researchers and developers to learn how to use the ARM TrustZone framework. The PinePhone is a pioneer in this aspect because their hardware developments are all open to inspect and they are take into account the development ideas of their community [PIN21]. Not only the hardware that is used is made 'open source' but the main operating system is Linux [PIN22] which enables the user to control every nook and cranny of the hardware.

2.5 OP-TEE

OP-TEE stands for Open Portable TEE [OPT22], it is an open source implementation of the API's that are exposed to Trusted Applications (TA) and that communicate with the TEE. It was designed with ARM TrustZone in mind but is applicable to other realizations of TEEs as well. The main design principles applied when creating OP-TEE were isolation, small footprint and portability. While the two first principles seem logical and have to do a lot with the security of the final product the portability is not straight forward but a nice feature to allow a very diverse community to have a related framework which encourages collaborations.

2.6 Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes [Lin+21]

System overview

The threat model assumes that attackers have physical access to the IoT device and are able to launch a wide variety of attacks. The attackers are assumed to be able to tamper with the images of the secure and normal world (including that of the OS) before the device is booted up. Another assumed attack is one where the adversary injects malware into the normal world during runtime and tamper with the normal world applications. Only the security of the text section of a program is considered but on the other hand it is assumed that this code is on ROM that is protected from modification.

Lastly the secure world and remote attestation server are assumed to be trustworthy and secure.

The solution that is proposed uses a hybrid booting method to ensure the load-time integrity and remote attestation to ensure the runtime integrity of the system. Secure boot is used to load the kernel of the secure world, this provides strong guarantees that the secure world starts in a secure and known state. The normal world is booted with what is called trusted boot which uses attestation to provide proof of the integrity of the image that is being started. Before the control is given to the rich OS it's image is measured and after it has started it should send this to the remote attestation server to verify the measurement. The remote attestation service is implemented in the secure world. The memory pages of the rich OS are periodically measured, encrypted by an attestation key and sent to the remote attestation server for verification.

Hybrid booting

Secure boot starts with a Root of Trust (RoT), which in this case is achieved by using the OCROM and eFuse of the IoT device. The first-stage bootloader is encrypted with a private key and stored on the OCROM to verify the integrity during the boot phase. The hash of the public key is stored in the eFuse to verify it's integrity during the boot phase. The images of the second-stage bootloader and secure kernel are measured and signed before deploying the device. These measurements and signatures are stored in the flash memory while the hash of the public key of the secondary bootloader is stored in the eFuse. The hash of the public key of the trusted kernel is stored in the secondary bootloader to achieve an incremental chain.

The secure boot phase starts with the first-stage bootloader which locates the second-stage bootloader, the public key and its signature. Secondly the first-stage bootloader calculates the hash of the public key and verifies the integrity of the public key. After successful verification it uses the public key to obtain the measurement result for the second-stage bootloader. Finally the first-stage bootloader calculates the hash of the second-stage bootloader and verifies its correctness. The second-stage bootloader does this entire process for the secure kernel to complete the secure boot phase.

Trusted boot is setup by producing a hash chain, the image of the rich OS is hashed first and concatenated with the image of the file system and hashed again. This final hash value is stored in the remote attestation server, during run-time this hash value will need to be sent to the remote attestation server in a secure way. To achieve secrecy a symmetric key is used, storing this in the IoT device is not trivial so this is solved with the following method. The Cryptographic Acceleration and Assurance Module (CAAM) is used to execute cryptographic functions in a secure environment. This module is used

to generate a 256-bit blob key, this key is used to encrypt the attestation key. A MAC is calculated from the attestation key to ensure its integrity. The blobkey is itself encrypted with a Blob Key Encryption Key (BKEK) which is derived from the master key (MK) by the CAAM. The MK is stored in Secure Non-Volatile Storage (SNVS) which is assumed to be secure by default.

The trusted boot phase starts with the NW attestation client application establishing a TLS connection with the remote attestation server and requesting a nonce. The measurement Trusted Application (TA) restores the attestation key from the Blob using the CAAM. The TA measures the rich OS and filesystem images, appends the nonce to the final hash value and encrypts this combination with the attestation key. The encrypted text is put in shared memory to allow the client application to send it to the remote attestation server. On the remote attestation server the cyphertext is decrypted and verified to check for integrity violations and replay attacks.

Page-based attestation

The idea is to measure the code segments of the programs in the normal world on the IoT device, it is assumed that this code base does not change in the lifetime of the device. The secure world is trusted but the normal world is still vulnerable to attackers, that is why attesting the code in the normal world would increase the security for the applications running in the normal world. The measurement is done on pages of 4KB at a time so programs will have multiple tuples of the form {*process-name, page-hash*} which will later be used to verify the integrity of the process. The first measurement is done before deploying the IoT device and the results are stored on the remote attestation server to be able to compare the future measurements with.

Process integrity measurement starts with the measurement Trusted Application which resides in the Secure World, it requests the memory address of the initial process. The client application translates the virtual address of the *init_proc* into a physical address which is later translated to the virtual address in the secure world memory address space. With this address the measurement TA iterates over all processes and measures their code pages. This measuring method uses the *task_struct* which has a doubly-linked-list structure so it enables the TA to find all processes.

Process integrity attestation uses the measurement of the process integrity measurement stage. First the IoT device requests a nonce from the remote attestation server with which a TLS connection is established. The measurement TA encrypts the measurement results concatenated with the nonce using the attestation key. The measurements of the processes are encrypted individually meaning that a set of cyphertexts is sent to the remote attestation server. The remote attestation server decrypts the measurements

of the processes and checks whether integrity violations can be found, this means new software or old software that has been modified.

Evaluation

The effectiveness of the secure boot process is measured by whether the secure boot phase is able to detect any violations against the integrity of the images, the signatures or the public key which it does correctly. For the trusted boot the focus lies on whether the remote attestation server is able to identify an abnormal system status, this is the case because NW programs can still be executed but the remote attestation server will verify the system state. The process integrity attestation is tested by tampering with existing programs and inserting additional programs, both these cases are also picked up on by the attestation.

Performance of the boot procedures is measured by comparing the mean time of 30 iterations with secure and trusted boot and 30 iterations without it. The secure boot adds little overhead on the second-stage bootloader while the trusted boot almost doubles the time it takes for the secure kernel to boot. The main reason why the secure kernel takes this long is because the image of the filesystem and rich OS is rather large and takes some time to measure. The overhead of the measurement TA and the attestation CA is measured by calling rich OS services while these modules are running and with them disabled. The overhead these modules introduce is between -0.55% and $+0.67\%$.

Security analysis is executed on the hybrid booting approach and the page-based process attestation. In the booting method a Chain of Trust (CoT) is constructed from the Root of Trust (RoT) residing in the eFuse and OCROM. A successful secure boot ensures that the secure world can be seen as the secure base from which the normal world can be booted. If the normal world image is tampered with the remote attestation server will pick up on this threat. The execution of the measurement TA and the results it generates are both secure because of the isolation in the secure world. The results pass through the normal world but they are encrypted at this stage, the encryption key is also securely stored in the secure world giving the normal world no opportunity to get hold of the information. The main drawback of this approach is that the method relies on the rich OS to access the paging structure and process management kernel objects.

Chapter 3

Method

3.1 Detailed Problem

Context

Sensitive data is often stored on smartphone devices or being used and transmitted.

Mobile computing is the main usage scenario when talking about smartphones but this means that the signals it sends can be picked up by lots of people that are nearby.

Performance driven , that is still the slogan lots of designers or implementers for smartphones have in mind.

Security is needed in regards of the functionality these devices provide, yet it's nowhere near what personal computers can provide.

Solution

Hardware security features could be the solutions or at least part of the solution, these features make sure that the attack surface of the smartphone becomes very narrow and well defined.

Correct implementation is of course still necessary, this is why research in this area is of utmost importance and experience needs to be shared to educate engineers in how to use these frameworks.

3.2 System Model

Open platform

Multiple software providers with no mutual trust will want to be certain about their software running without it being interfered with by software of other providers.

The platform owner is the user themselves, with phones from large companies the company is still the actual owner because only software with a correct signature can be installed on the device.

Secure software execution

Software isolation should be used to ensure the software providers integrity of execution.

Secure data storage is necessary to make sure that data from one application cannot be read or modified by another one.

3.3 Attacker Model

Physical access brings along lots of risk because the adversary has a variety of possible attacks they could launch from this position.

OS/Firmware attacks have the risk of compromising all user level applications because the OS is the 'trusted' layer on which these user level applications rely.

Software attacks try to tamper with the control flow of certain program executions or get hold of sensitive data through malicious code.

3.4 Solution

Secure boot ensures that the device starts in a known secure state, to achieve this a Root of Trust is needed from which a Chain of Trust is constructed.

User attestation can be used to check the integrity of control flow, data structures etc. it should also be able to check authenticity of the code that is running.

Trusted Execution Environment will provide the desired characteristics of execution isolation and runtime integrity.

Chapter 4

Implementation

4.1 Attestation TA

Trusted application

Hashing the memory pages of the text section of a program is the very first step the TA does when attesting a program.

Storing reference values needs to be done when the device is in a known secure state, the memory pages will be hashed and these hashes are stored for later comparison.

Comparing the hash of a memory page with its reference value is the actual attesting step, from this comparison it should be clear whether the integrity of the memory page has been violated.

Notifying the user is the final step to inform them about the problem to allow them to take action, this could be rebooting to ensure a secure known state again or ask for help from a specialist.

NW OS dependencies

Retrieving address needs to be done from within the NW OS at the moment, this is because the datastructures that contain this information are owned by the NW OS.

Translating address is another functionality for which the rich OS is used, this is also due to the fact that these translations are easily determined from the NW OS datastructures.

Extensions

Trusted IO could be used to inform the user of the problem (which program has been tampered with for instance), it could also take on a more coarse grained form that an led licht signals the user that some piece of software has failed the attestation which is easier but less usefull.

Becoming independent from the rich OS in the normal world seems like a very important step because otherwise OS/Firmware attacks are still a threat.

Detailed attestation is necessary, lots of software attacks are based on the used datastructures and don't impact the text section of code of programs.

Chapter 5

Experiments

5.1 Performance

Reproduction

Trusted boot is based on the attestation of the normal world before giving it control. In the paper they talk about 107 MB of filesystem image that is being measured so this could be a valuable starting point to compare their performance with the performance achieved in this thesis.

Overhead is measured in the paper by executing system services from the linux kernel and running this experiment with and without the attestation.

Additional

Attestation time should be measured for a program with a certain size to be able to evaluate how often this attestation should be executed to have a balance between performance overhead and security assurance.

5.2 Performance Evaluation

Comparison between the performance achieved in the paper and the performance measured in this thesis is important to be able to interpret the results correctly.

Balance between performance overhead and security assurance depends on the usecase but in the context of the smartphone some statements could be made.

5.3 Security Properties

Integrity of the measurement execution is of utmost importance when it comes to attestation, in remote attestation this is achieved because of a hardened server but here the trusted execution environment needs to take care of this.

Secure storage of results is also important, first of all to make sure the reference values are not tampered with and second of all to correctly react to results that indicate a violation.

5.4 Security Evaluation

Security guarantees that can be made are the integrity of the measurement execution and the security of the results that are being stored. These are achieved due to secure boot enabling the trusted execution environment but are key assumptions in the field of remote attestation.

Shortcomings in terms of security are the OS/firmware attacks because the solution still relies on the rich OS rather heavily.

Extensions in terms of additional aspects of the system that can be attested are necessary to protect against software attacks.

Chapter 6

Discussion

6.1 Related work

Secure boot, Trusted boot and remote attestation for ARM TrustZone-based IoT Nodes is the paper on which the implementation and experiments are based.

DAA-TZ: An Efficient DAA Scheme for Mobile Devices Using ARM TrustZone implements Direct Anonymous Attestation on a mobile ARM TrustZone device.

SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE implements enclaves on a CPU with ARM TrustZone technology.

TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone uses ARM TrustZone to protect the attestation service on the mobile device from being tampered with.

TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone utilizes the functionality of the secure world to shield applications from untrusted OSes.

6.2 Comparison of Approaches

Effectiveness

The goal of these papers are all a little different but it is important to evaluate which ones actually realized their goal and how this compares to the goal set out by this thesis.

Most variety of attacks that the solution defends against is a clear measure on how effective the solution is in the field.

The strongest security guarantees that were made and achieved also indicate how well the solution works.

Assumptions

The least assumptions that were made by the authors of the paper the more widely applicable the solution is because there is enormous hetergeneity among devices and assumptions put restrictions on the devices for which the paper is usefull.

The most realistic assumptions are of course also important to look at, if the assumptions are not realistic they are not practical to adhere to and the solution will be worthless if it can't be applied to the real world.

6.3 Future Improvements

Weaknesses

Rich OS dependency is very undesirable, it is thus important to look at different solutions that achieve similar outcomes to avoid this aspect of the current solution.

Uncomplete attestation introduces a fake sense of security because not all possible attacks are checked, for instance modified data structures that influence the control flow of a program. (inspiration from Lightweight and Flexible Trust Assessment Modules for the Internet of Things)

Additional features

- Based on the related work papers some additional features could be stated or solutions could be combined to achieve protection against a wider variety of attacks.

Chapter 7

Conclusion

Solution overview

Reproduction of the solution provided in the paper is the starting point of this thesis.

Extensions on this reproduced solution are necessary because the original solution does not achieve all goals.

Shortcomings

Attacker possibilities that are not accounted for are still present, the security measures taken in the solution are not adequate for the wide variety of attacks that it claimed to protect against.

Desired guarantees the solution should be able to provide are not entirely met.

Positives

Code for this thesis is made available as open source code to make it easier to reproduce the experiments and continue work on this research topic by other researchers.

Thorough comparison has been executed on the provided solution and solutions of related work to give an overview of what direction is most promising to achieve the security goals.

Bibliography

[Ala+08] Masoom Alam et al. “Model-based behavioral attestation”. eng. In: *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*. SACMAT ’08. ACM, 2008, pp. 175–184. ISBN: 9781605581293.

[Ala+12] Masoom Alam et al. “Analysis of existing remote attestation techniques”. eng. In: *Security and communication networks* 5.9 (2012), pp. 1062–1082. ISSN: 1939-0114.

[Ali+17] Toqeer Ali et al. “Design and implementation of an attestation protocol for measured dynamic behavior”. eng. In: *The Journal of supercomputing* 74.11 (2017), pp. 5746–5773. ISSN: 0920-8542.

[ARM22] ARM Ltd. *TrustZone TrustZone for cortex-a*. 2022. URL: `https://www.arm.com/technologies/trustzone-for-cortex-a` (visited on 04/26/2022).

[Ba+17] Haihe Ba et al. “Runtime Measurement Architecture for Bytecode Integrity in JVM-Based Cloud”. eng. In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. Vol. 2017-. IEEE, 2017, pp. 262–263. ISBN: 9781538616796.

[Cha+17] Rui Chang et al. “MIPE: a practical memory integrity protection method in a trusted execution environment”. eng. In: *Cluster computing* 20.2 (2017), pp. 1075–1087. ISSN: 1386-7857.

[Cok+11] George Coker et al. “Principles of remote attestation”. eng. In: *International journal of information security* 10.2 (2011), pp. 63–81. ISSN: 1615-5262.

[Dua+20] Jialiang Duan et al. “Integrity Measurement Based on TEE Virtualization Architecture”. eng. In: *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*. IEEE, 2020, pp. 370–376. ISBN: 9780738105208.

[Esk+18] Saba Eskandarian et al. “Fidelius: Protecting User Secrets from Compromised Browsers”. eng. In: (2018).

[Fer+17] Andrew Ferraiuolo et al. “Komodo: Using verification to disentangle secure-enclave hardware from software”. eng. In: *Proceedings of the 26th Symposium on operating systems principles*. SOSP ’17. ACM, 2017, pp. 287–305. ISBN: 9781450350853.

[Fot+21] Georgios Fotiadis et al. “Root-of-Trust Abstractions for Symbolic Analysis: Application to Attestation Protocols”. eng. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 13075. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 163–184. ISBN: 9783030918583.

[Goo22] Google. *Android keystore system*. 2022. URL: `https://developer.android.com/training/articles/keystore` (visited on 04/26/2022).

[Gu+08] Liang Gu et al. “Remote attestation on program execution”. eng. In: *Proceedings of the 3rd ACM workshop on scalable trusted computing*. STC ’08. ACM, 2008, pp. 11–20. ISBN: 9781605582955.

[Guo+21] Pengfei Guo et al. “Research on Arm TrustZone and Understanding the Security Vulnerability in Its Cache Architecture”. eng. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12382. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 200–213. ISBN: 9783030688509.

[JSS06] Trent Jaeger, Reiner Sailer, and Umesh Shankar. “PRIMA: policy-reduced integrity measurement architecture”. eng. In: *Proceedings of the eleventh ACM symposium on access control models and technologies*. SACMAT ’06. ACM, 2006, pp. 19–28. ISBN: 1595933530.

[KBC21] Michal Kucab, Piotr Borylo, and Piotr Cholda. “Remote attestation and integrity measurements with Intel SGX for virtual machines”. eng. In: *Computers & security* 106 (2021), p. 102300. ISSN: 0167-4048.

[Kin06] Steven Kinney. *Trusted platform module basics: using TPM in embedded systems*. Mbedded technology series. Amsterdam ; Boston: Elsevier Newnes, 2006. ISBN: 1280637005.

[KM20] Fatima Khalid and Ammar Masood. “Hardware-Assisted Isolation Technologies: Security Architecture and Vulnerability Analysis”. eng. In: *1st Annual International Conference on Cyber Warfare and Security, ICCWS 2020 - Proceedings*. IEEE, 2020, pp. 1–8. ISBN: 9781728168401.

[Len+18] Matthew Lentz et al. “SeCloak: ARM Trustzone-based Mobile Peripheral Control”. eng. In: *MobiSys 2018 - Proceedings of the 16th ACM International Conference on Mobile Systems, Applications, and Services*. MobiSys ’18. ACM, 2018, pp. 1–13. ISBN: 9781450357203.

[Lin+21] Zhen Ling et al. “Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes”. eng. In: *Journal of systems architecture* 119 (2021), p. 102240. ISSN: 1383-7621.

[Mac+17] Aravind Machiry et al. “BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments”. eng. In: *NDSS*. ndss symposium, 2017.

[MAT+21] Tsutomu MATSUMOTO et al. “Secure Cryptographic Unit as Root-of-Trust for IoT Era”. eng. In: *IEICE transactions on electronics* E104.C.7 (2021), pp. 262–271. ISSN: 0916-8524.

[MNP16] Jan Tobias Mühlberg, Job Noorman, and Frank Piessens. “Lightweight and Flexible Trust Assessment Modules for the Internet of Things”. eng. In: *Computer Security – ESORICS 2015*. Vol. 9326. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 503–520. ISBN: 9783319241739.

[OPT22] OP-TEE. *OP-TEE About OP-TEE*. 2022. URL: `https://optee.readthedocs.io/en/latest/general/about.html` (visited on 04/26/2022).

[PIN21] PINE64. *PINE64 PinePhone*. 2021. URL: `https://www.pine64.org/pinephone/` (visited on 04/26/2022).

[PIN22] PINE64. *PinePhone Software Releases*. 2022. URL: `https://wiki.pine64.org/wiki/PinePhone_Software_Releases` (visited on 04/26/2022).

[PS19] Sandro Pinto and Nuno Santos. “Demystifying Arm TrustZone: A Comprehensive Survey”. eng. In: *ACM computing surveys* 51.6 (2019), pp. 1–36. ISSN: 0360-0300.

[Qin+20] Yu Qin et al. “RIPTE: Runtime Integrity Protection Based on Trusted Execution for IoT Device”. eng. In: *Security and communication networks* 2020 (2020). ISSN: 1939-0114.

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. eng. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE, 2015, pp. 57–64. ISBN: 9781467379526.

[Sam22] Samsung. *KNOX Stay connected, protected, and productive*. 2022. URL: `https://www.samsungknox.com/en` (visited on 04/26/2022).

[SKU10] B Stelte, R Koch, and M Ullmann. “Towards integrity measurement in virtualized environments - A hypervisor based sensory integrity measurement architecture (SIMA)”. eng. In: *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 2010, pp. 106–112. ISBN: 1424460476.

[Van+19] Jo Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”. eng. In: *Proceedings of the 2019 ACM SIGSAC Conference on computer and communications security*. CCS ’19. ACM, 2019, pp. 1741–1758. ISBN: 9781450367479.

[YF08] Aimin Yu and Dengguo Feng. “BBACIMA: A trustworthy integrity measurement architecture through behavior-based TPM access control”. eng. In: *Wuhan University journal of natural sciences* 13.5 (2008), pp. 513–518. ISSN: 1007-1202.

[Zha+21] Shijun Zhao et al. “Research on Root of Trust for Embedded Devices based on On-Chip Memory”. eng. In: *2021 International Conference on Computer Engineering and Application (ICCEA)*. Piscataway: IEEE, 2021, pp. 501–505. ISBN: 9781665426169.