Chapter 1

Background

1.1 Remote Attestation

Goals

Supplying evidence about a target to a verifier is the most important goal for remote attestation. [**?**] 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 [?] [?] [?] [?]. According to [?] 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. [?] 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 [?] [?] [?] [?].

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 [?] which is again mentioned by the same researchers in their analysis about existing techniques [?]. 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. [?] 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.

1.2 Trusted Execution Environment

Definitions

A definition of a Trusted Execution Environment is given by [?]: "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 [**?**] [**?**] [**?**] [**?**].

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 [**?**] [**?**] [**?**] [**?**] 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.

1.3 ARM TrustZone

Core principles

ARM TrustZone [**?**] implements the TEE on the processor level which means it runs below the hypervisor or OS [**?**]. 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) [**?**] [**?**] [**?**] [**?**]. 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.

1.4 PinePhone

ARM TrustZone is used in the System on Chip (SoC) of the PinePhone which is a popular solution amongst smartphone suppliers, Samsung KNOX [**?**] and Android KeyStore [**?**] 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 [**?**].

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 [**?**]. Not only the hardware that is used is made 'open source' but the main operating system is Linux [**?**] which enables the user to control every nook and cranny of the hardware.

1.5 OP-TEE

OP-TEE stands for Open Portable TEE [**?**], 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.

1.6 Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes [?]

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.