Chapter 1

Discussion

1.1 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.

SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE

The solution described in the paper about SecTEE [?] aims to implement a framework using ARM TrustZone to achieve similar security guarantees as a hardware-based secure enclave architecture. The writers provide the following contributions with their paper. First of all SecTEE the new secure enclave architecture which achieves 'the hightest level of security' for ARM platforms using ARM TrustZone. Secondly a locking mechanism is introduced which makes sure enclave pages cannot be accessed while the enclave is running to prevent cross-core side channel attacks. The TEE OS is also extended to provide functionality like identification, remote attestation and sealing sensitive data. Lastly an implementation of SecTEE based on OP-TEE is provided along with experiments showing the performance.

Their wording of 'the highest level of security' means that it should be resistant to privileged host software attacks, board-level physical attacks, page fault based side-channel attacks and cache based side-channel attacks. The hardware attacks on which is focused are cold boot attacks [?], bus monitoring attacks [?] and Direct Memory Access (DMA) attacks. Attacks against the internal state of the SoC are not considered because those are assumed to be very sophisticated and require expensive equipment. For these goals to be achieved certain requirements are listed. For one thing a Device Sealing Key (DSK) needs to be present, this is a symmetric key only known by the device itself and used to protect secrets related to the device. As follows a Device Root Key (DRK) which is an asymmetric key pair and is necessary to identify and authenticate the device. Lastly the Manufacturer's Public Key needs to be hard coded on the device to make sure it is able to verify the signature on software updates from the manufacturer.

An important aspect of the SecTEE architecture is the method that is applied for memory protection. SecTEE protects enclaves from physical attacks by using a similar approach as the OP-TEE pager. It is an on-demand paging system which runs the entire TEE system on On Chip Memory (OCM). Whenever a page leaves this OCM it is encrypted to ensure confidentiality and integrity of the data while it is stored on the DRAM. Another key feature of SecTEE is it's side-channel resistance. Side-channel attacks from the secure world are avoided by using a page coloring mechanism. Different enclaves can never share the same cache set which ensures that one enclave will never be able to evict cache lines of another one. Of course side-channel attacks can also come from the Normal World (NW) especially because the NW and Secure World (SW) share the same cache in the ARM TrustZone architecture. In the NW there are of course certain limitations to what is possible with the cache lines due to privileges, the prime and probe method is still relevant in this case though. SecTEE cleans and invalidates all cache levels when the CPU switches from the SW to the NW. This however is not sufficient because cross-core side-channel attacks can launch the attack while the CPU is still executing in the SW. To handle this problem the cache set of the enclave is locked which guarantees that the cache of the enclave cannot be probed or manipulated by the NW. Some final aspects that deserve some highlighting are enclave identification, measurement and remote attestation in SecTEE. Enclaves are published along with the public key of the author and the signed integrity value of the image. With this information the enclave can be verified and identified before it is run. During run-time the enclave keeps track of important aspects like enclave ID, integrity and a flag indicating whether it is privileged. For the remote attestation a specific Quoting Enclave is created, this is a privileged enclave which is able to make the system calls related to the management of attestation keys, other enclaves are not allowed to do this. These keys are used to sign the report data of the attestation together with the run-time measurement to provide proof to the verifier that it comes from the correct enclave.

First of all the number of lines of code are measured because these implementations extend the TEE kernel which is the Trusted Computing Base (TCB) and this needs to be kept minimal because bugs could easily introduce security vulnerabilities. Next the overhead of the trusted computing features is identified and discussed. This overhead is acceptable in case (Eliptic Curve Cryptography) ECC with keys of 256 bits are used, in case RSA keys are used of 2048 bits certain system calls take too long to be usable. The performance of certain enclaves was measured and the xtest benchmark of OP-TEE were executed. SecTEE was about 4 times slower than OP-TEE on the benchmark and 40 times slower than OP-TEE on the enclave execution. It is argued that the memory protection mechanism is the cause of the greatest performance overhead in SecTEE. Finally to test the effectiveness of the side channel defense the effectiveness of the attack on plain OP-TEE is compared to the case of SecTEE. In the case of a NW attack there is a clear difference between prime and probe timings in OP-TEE while both memory accesses take an equal amount of time in case of SecTEE which makes it impossible for an attacker to learn anything about the cache behavior of the victim enclave. For the SW attack an AES key could be recovered after 256,000 encryptions in OP-TEE while in SecTEE the attacker could not learn any information about the used key.

TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone

TrustShadow [**?**] is designed to protect existing applications from untrusted Operating Systems (OS) without the need to modify the applications themselves. This is achieved by using TrustShadow as a lightweight runtime system that shields the sensitive information of the application from the OS. This system does not provide system services itself but requests them from the OS, it does however guarantee security through encryption, context switching and verifying return values. To ensure that the OS cannot get hold of the encryption keys, sensitive data or interfere with the security verification TrustShadow runs in the Secure World of the ARM TrustZone processor.

The solution described in the paper about TrustShadow is mainly focused on protecting applications from an untrusted OS. This means that the attacker is able to launch Direct Memory Access (DMA) attacks, modify interrupted process states, change program execution control flow, hijack system services and control communication channels [**?**]. Denial of Service (DoS) attacks and side-channel attacks such as timing and power analysis are not taken into account in the solution.

To make sure the rich OS is not able to tamper with the memory of the applications and the runtime system the memory of the different stakeholders is isolated from each other. TrustShadow utilizes the Trust Zone Address Space Controller (TZASC) to make three partitions in memory, one unprotected for the applications, one protected for the applications and one for the runtime system itself. The virtual memory space is equally distributed between the OS and the runtime system TrustShadow, and the runtime system also maps the physical address space holding the OS to efficiently locate shared memory. Another aspect that needs to be taken into account is that the OS is used to handle exceptions. When an exception is thrown the state of the application is stored and the state for the OS is restored. The runtime system then reproduces the exception in this OS state to allow the OS to handle it without having to get sensitive details about the application. There are two exceptions that the runtime system implemented in the secure world and those are the random number generation and floating point operations. These two exceptions are security critical because the used cryptography relies on the security of these operations. Similar to exceptions are system calls, the OS needs some information about the application requesting the system call but TrustShadow sanitizes this information to make sure only the necessary data is received by the OS. After receiving the result from the system call the runtime system also checks this result on correctness before forwarding it to the application. Another example of sensitive information about an application that an untrusted OS should not have access to is the memory mapping from virtual to physical address space. This information is stored in the page tables, TrustShadow makes use of the OS page fault handler to update and retrieve the page table entries that are stored in the secure world. This ensures that the rich OS does not require the information because it is controlled by the runtime system. After the memory location of a page has been found this can be loaded for execution, before this loading is finished however the page is checked on it's integrity. TrustShadow verifies the executable pages of an application and the executable pages of the shared libraries based on a manifest file related to the application.

TrustShadow was evaluated with a variety of measuring methods. Microbenchmarks using LMBench, file operations using Sysbench, HTTP and HTTPS request handling using Nginx and finally the performance overhead on data analytic applications. The performance overhead caused by the runtime system was negligible across all tests. In terms of security the OS still has a lot of control, it could for instance invoke a system call with the original *execve* instead of the new *tz_execve* which largely bypasses TrustShadow. Another concern is the possibility to manipulate or forge manifest files which could allow an attacker to tamper with the code of an application without the runtime system being able to detect it.

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.

1.2 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.

1.3 Future Improvements

Tradeoffs

- a hash for every page vs one hash for the entire process

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.