Chapter 1

Implementation

The goal for the implementation is to guarantee that the executable memory pages that are loaded into the memory are not tampered with. This integrity check is achieved by measuring the code pages of the processes in advance, for instance during the installation of the particular code. This measurement can be done in a variety of ways, in this work hashing was chosen because of it's common use and it being well understood by the community. After these initial measurements have been taken they need to be stored in a secure place, the secure memory of the PinePhone was chosen because it can guarantee the integrity and confidentiality of the data. Last but not least these measurements need to be retaken during execution time and compared with the initial measurements to verify whether the running code has been tampered with. These run time checks need to happen periodically to guarantee no modifications are introduced, or that the changes are observed before they can damage the system too much.

1.1 NW application

The Normal World (NW) application begins with looking up the processes that are currently running by examining the */proc* directory. In this directory there is a collection of directories with numbers as names, these numbers represent the process identifiers (pid). The pid is used within Linux to allow the operating system to differentiate between the different processes and manage them. It is of course necessary to attest all these processes, however the explanation will focus on just one for now to keep things clear. Take for instance the directory */proc/1*, this one will always exist due to the fact that pid 1 is associated with *proc_init* which is the initial process the Linux operating system forks. Within this directory there are two important files *pagemap* and *maps*. The *maps* file gives an overview on the different memory regions associated with the process. Besides the virtual address range other information is also present for instance whether the pages are executable or

writable and where their symbolic origin is in the file structure. The *pagemap* file on the other hand is necessary to do the translation from virtual address to physical address.

The translation of the virtual addresses that are found in the *proc/pid/maps* file into physical addresses is based on a solution from [**?**]. The solution investigates the *proc/pid/pagemap* file and generates a *pagemap_entry* from it. This *pagemap_entry* is used to structure the information associated with one entry in the *maps* file. Based on this *pagemap_entry* the physical address can be derived from the virtual address found in the *maps* file. Do note, the *maps* file provides a contiguous virtual memory region, it is not guaranteed that the physical memory will also be contiguous so the first virtual address of every page is translated into a physical address. Pages are 4kB large so lots of virtual addresses will have to be translated into physical ones. After the physical addresses are obtained they are put together into a list and a memory reference of this list is sent to the secure world for further investigation.

1.2 Attestation PTA

The attestation Pseudo Trusted Application (PTA) receives a memory reference with inside the buffer all the physical memory addresses of the pages that need to be attested. Before the Secure World (SW) is able to access these memory pages they need to be mapped into its virtual memory space. The mapping is done using the *core_mmu_add_mapping* function from the OP-TEE kernel, when this mapping is successful the virtual address can be obtained using *phys_to_virt*. The function *phys_to_virt* returns the virtual address in the PTA where the memory can be accessed. With the TA having access to the memory pages, the actual measurements can start. The hashing algorithm used is SHA-256, this can easily be substituted by another algorithm provided in the OP-TEE library if necessary. The hashing algorithms are provided by the OP-TEE framework and easily usably from within the PTA.

In the initialization phase the hash digests are stored in the secure memory of the device. The files written from the PTA to secure memory are only accessible by this PTA and are protected against everything in the Normal World. In this file the hashes are sorted based on which *pagemap_entry* they come from and the page number within this region, this allows to later uniquely identify the hash value with which will need to be compared. After the initial hash values have been stored the initialization phase is complete. This implies that the security of the stored measurement values is guaranteed under the assumption that the secure world does indeed protect the secure memory against access (read and write) from outside the Secure World.

During the attestation phase, all the same steps will be taken as the initialization phase has taken up to this point except for storing the calculated

hash values. Instead during this phase the newly calculated hash values will be compared with the hash values that are stored in the protected file where the initialization phase has written the measurement results. For the comparison the hash value and the initial value are put into 4 *uint64_t* data types each, the first one from the hash is then compared with the first one of the initial value, the second one with the second one and so on, using the standard comparison functionality. The number of comparisons that fails is saved and printed in the debugging output stream of the secure world. Ideally the user should be notified about these faults and possibly which processes may experience an impact from this based on whether the process uses the code for which the attestation has failed. Based on the information the user receives from the attestation they can decide what action to take as the owner of the device.

1.3 Improvements and extensions

The first and probably most important improvement has to do with the dependency on the Rich OS. Looking up the virtual addresses and translating them into physical addresses needs to be done from within the NW OS at the moment, this is because the data structures that contain this information are owned by the NW OS. On top of that this functionality is only possible with root privilege because the necessary files are not accessible otherwise. If possible this should be improved because an attacker in control of the rich OS could alter these data structures to hide the changed processes from the attestation PTA. The adversary could for instance keep an unchanged process hierarchy loaded in memory while the device is actually running on a different malicious process hierarchy. As long as it cannot be guaranteed that the addresses the secure world receives are the addresses of the actual processes that are running, the attestation can be bypassed.

Secondly, the attestation currently attests the executable pages present in RAM which is sufficient but for the initialization phase all the possible executable pages need to be measured to have a reference value to compare with. To achieve that the entire code base of the process is measured during the initialization phase the binary files of the modules can be looked up. If it is possible to access these rich OS files from within the secure world they could also be attested as if they were loaded in memory. By measuring from these files the initialization phase is not bound by the executable memory pages present in RAM anymore and can attest all pages. Having an initial value for all code pages is important because measuring a memory page for the first time while the device is already deployed, there is no guarantee that the memory page hasn't been tampered with already so the initial value does not provide strong additional security guarantees.

A significant extension to allow this proof of concept to be turned into a complete solution is to notify the user using trusted I/O. When informing the

user about the results of the attestation it is important to keep in mind the security of the channel on which this is achieved. ARM TrustZone provides trusted I/O paths which can be used for this goal. Trusted I/O could be used to inform the user of the problem (which program has been tampered with for instance) or it could also take on a more coarse grained approach like an LED light signal that some piece of software has failed the attestation which is easier but less useful. What details can be derived from this communication is not necessarily of the greatest importance, it is important how this communication works. To make sure the NW cannot interfere with the communication from the attestation PTA to the user it should be implemented using the provided APIs of ARM Trusted Firmware-A (TFA) [**?**]. Secure I/O paths is an extensively researched topic in the field of Trusted Execution Environments, for instance to protect user data from compromised browsers [**?**] or to have a general trusted I/O path between the user and trusted services [**?**]. Using Secure I/O is necessary to build a fully functional solution based on the proof of concept presented in this chapter. The Secure I/O was not included in this work because no new functionality would be showcased and it would divert too much from the main focus of the implementation namely the attestation of the NW processes.

The second major extension focuses on the added security guarantees the attestation process provides. To achieve great security guarantees it is necessary to do more than just code attestation, lots of software attacks are based on the used data structures and don't impact the text section of programs. As discussed in the background on attestation there are a variety of methods to achieve this and a great example of an extensive attestation method is [**?**]. These forms of attestation are of course more complicated to execute and take a great amount of implementation effort. Due to the limited time and experience the attestation for this work was kept relatively simple to show that this form of attestation is achievable on the PinePhone. When designing a final product it is of course encouraged to use more advanced techniques to increase the security guarantees of the code execution on the device.

Lastly it was found out that in a recent framework update of OP-TEE an attestation PTA was added to the kernel [**?**]. The big difference between that PTA and the implementation described above is that it attests the secure world processes (TA's) while the implementation presented here is focused on attesting the processes of the Normal World. Even though there are certain differences, it is definitely possible to change the implementation 'slightly' to make it fit into this provided attestation PTA and reuse some of it's already present functionality. Before this could ever happen the first weakness of this implementation should probably be tackled namely the fact that this method still relies on the rich OS.