Chapter 1

Method

1.1 Detailed Problem

Smartphones often store, use and transmit sensitive data of their owners. Transmission will often happen on a 4G network because smartphones are frequently used on the go. This form of mobile computing implies that lots of people in the near surroundings of the smartphone can pick up on these signals. The security of these transmission methods like 4G are already extensively analyzed [?] [?] [?] but there is still room for improvements because vulnerabilities like a paging storm attack [?] or session hijacking [?] are still possible. 4G is of course not the only communication channel available on a smartphone, Bluetooth is a very popular option to connect devices to a Personal Area Network (PAN). While very popular there are still many security risks and possible attacks against it like reflection attack [?], cross technology pivoting attack (forcing a different protocol to be used) [?], Bluejacking, Bluesmacking and Bluesnarfing [?] and finally entropy downgrade attacks which make brute force attacks on the keys possible [?]. Certain security risks related to data transmission do not have a direct impact on the user but could have future implications as [?] show, they found a way to identify the smartphone OS based on smartphone traffic which could give attackers insight in what vulnerabilities to use. Although very relevant, the transmission stage is not the only option for data to leak, it still needs to be protected during storage and when it is used at execution time.

Adversaries have many possibilities when it comes to stealing sensitive data from smartphone users while the data is being used during execution or stored on the device. [?] for instance proposed AlphaLogger which infers the letters being typed on a soft keyboard based on the vibrations and [?] we're able to achieve identity theft through data cloning of auto login credentials. Besides custom attacks, there are plenty of well known risks as well. [?] have executed a review of various malicious software threats and mitigations, [?] researched software attacks that take advantage of hardware resources to conduct fault injection or sidechannel analysis and [?] reviewed the possible threats that can be introduced by smartphone providers altering the Android OS to add their signature flavor. These attacks are possible due to the fact that the design of smartphones is based on that of IoT devices, which means that lots of focus lies on the performance of the final product. This performance is often hard to achieve because the resources want to be kept to a minimum to lower the price or keep the device small. Security is still seen as a performance killer, which is very unfortunate because it should have an essential role in the design and implementation of a system. The security of a system like smartphones is even more crucial because lots of people are unaware of the possible threats they face.

1.2 System Model

The system model is a smartphone that is owned by a user but for which multiple software providers offer programs. These providers don't necessarily trust eachother but they do want guarantees that the execution of their software will not be interfered with by software of other providers. The user of the device is also it's owner, they have full control over what software should be able to be installed and run on the platform. This is very different from large smartphone companies where the company has a signature key which is kept secret from the user of the smartphone. Programs that are not signed with this signature key will be rejected for installation on the device. The smartphone in this case is a PinePhone which is equipped with 4 ARM Cortex A53 Cores that are TrustZone enabled. Lots of smartphones have ARM processors as System on Chip (SoC) which means that ARM TrustZone is the ideal candidate for the implementation. It provides additional hardware security features which make sure that the attack surface of the smartphone becomes very narrow and well defined. While providing additional security it only has a minimal impact on performance due to it being implemented on hardware [?] [?]. ARM TrustZone makes sure that there is a Trusted Execution Environment (TEE) available with capabilities like secure memory, trusted I/O and many others. The Trusted kernel is responsible for making partitions of the memory only accessible to the secure world which should then be able to provide secure data storage services to the normal world applications. Secure data storage is necessary to make sure that data from one application cannot be read or modified by another one. Trusted I/O paths on the other hand allow the user application to request I/O features from the Secure World (SW) instead of the rich OS. Because these connection go through the SW the rich OS is not able to inspect or modify the data that is transmitted to the I/O peripheral, this is important in cases where an adversary has gained control over the rich OS. For this TEE to work correctly a trusted kernel is required, for this the implementation of OP-TEE is used which provides the interfaces to communicate with the TEE and call Trusted

Applications (TA) from the normal world. The normal world runs a Mobian Linux distribution as kernel which is specifically written for a smartphone device. The system can be put together using the source of the Mobian distribution [?], OP-TEE [?], U-boot [?] and the ARM Trusted Firmware [?]. Unfortunately this was not realized in this work so the QEMU emulator [?] was used to test the code and run the experiments. QEMU allows to run OP-TEE on a desktop while emulating the TEE, because OP-TEE can run on the PinePhone the code and experiments should be reproducible on the PinePhone if correctly configured with OP-TEE.

1.3 Attacker Model

Physical access brings along lots of risk because the adversary has a variety of possible attacks they could launch from this position. The TEE can, when combined with some specific hardware make these attacks ineffective or a lot harder to execute. For instance reading from the hardware memory becomes ineffective because everything on there that is sensitive is encrypted by the TEE, only the TEE can decrypt this information with the help of a Trusted Platform Module (TPM). Tampering with memory can be made less effective by using secure boot, this ensures that the TEE is started up in a secure and known state from which a trusted base can be ensured. With this trusted base attestation could be run on the memory to check whether inconsistent memory pages can be found before they receive control in case of them being code pages. Another hardware attack is one where a physical back door is exploited, this is assumed to be impossible because the processor has been designed for security purposes and thus no back doors should be available. The main advantage defenders have in terms of hardware attacks is that they are often very hard, reading the physical memory is doable but this should be dealt with by the TEE combined with a TPM. There are still lots of hardware attacks for which TEE's are vulnerable, manipulation of RAM and eFuse bypassing secure boot [?], micro architectural structures leaking information [?] and electro magnetic analysis of side channels [?]. These attacks are very advanced and would be really hard to execute but they do exist and not many defense mechanisms are present to protect against them.

OS/Firmware attacks have the ability to compromise all user level applications because the OS is the 'trusted' layer on which these user level applications rely. This is the main area where ARM TrustZone and other TEE implementations make a very big difference in security. ARM TrustZone for instance is implemented below the rich OS which means that the trusted kernel has more privileges than the rich OS, the rich OS has of course more functionality but to achieve this functionality it will in some cases have to rely on the trusted kernel. A TEE increases the security when an OS attack has succeeded by shielding the user level applications from this OS, this is done by allowing the user level application to request services like trusted I/O and secure memory from the TEE itself without interference of the OS. Examples of this are a container using ARM TrustZone [?], securing camera and location peripherals [?] and checking whether the OS executes system services correctly [?]. It does not make claims about making OS attacks harder because the vulnerabilities in the rich OS are still there, the normal world could be attested which would allow the detection of an OS attack but the TEE doesn't have special protection mechanisms to avoid it from happening. This is not surprising of course, OS attacks are often a consequence of software bugs in the implementation that give rise to security vulnerabilities which are very hard (if not impossible) to avoid.

Software attacks try to tamper with the control flow of certain program executions or get hold of sensitive data through malicious code. Most of these attacks can be defended against by a TEE implementation or with the help of a TEE. Applications can be isolated from each other making it a lot more difficult for a malicious application to tamper with the execution or data of another one. This isolation can be enforced using virtualization or specialized Trusted Applications (TA), they make sure the application can only be interacted with using a very well defined interface. The virtualization needs to be implemented correctly to make sure no data is leaked in between the partitions, in the case of the TA it can use the TEE functionality to store its data securely. Software attack defense mechanisms based on ARM TrustZone come in many forms, isolate application and secure communication [?], control flow integrity scheme [?] and reverse engineering protection [?].

1.4 Solution

As discussed in the attacker model, the TEE (ARM TrustZone in this case) will provide a certain level of security on its own by providing secure and trusted services to user applications. These services can be utilized to achieve secret encryption, trusted I/O paths and secure data storage [?]. These services rely on the assumption that the TEE itself is secure and trusted, this can be achieved through secure boot [?]. Secure boot ensures that the device starts in a known secure state, to achieve this a Root of Trust (RoT) is needed from which a Chain of Trust (CoT) is constructed. The RoT is often implemented by using a Trusted Platform Module (TPM) along with hardware memory specifically designed for secure storage like an eFuse for instance. The CoT ensures that only code that is verified will be able to execute and get control over the device during boot time. When the device is successfully started up using secure boot it can be confidently assumed that the trusted kernel and TEE will work as intended. This of course is only the beginning, a TEE provides a framework which needs to be used to implement secure solutions and defense mechanisms against known attack strategies.

An important application that is built upon the TEE framework is one where the integrity of the control flow, code and data is guaranteed. Achieving this level of security is rather hard, weakening this constraint in the sense of making sure violations to this integrity are detected is achievable. Attestation can be used to check the integrity of an application or running system depending on what properties are looked at to measure the reliability of the target. Remote attestation seems like a weird solution in the context of a smartphone, but the user (which in our case is seen as the owner of the device) could be alerted about the attestation results. This attestation process can run within the TEE on the device and because of this will be tamper proof against software and OS attacks. Notifying the user will also need to be done in a secure manner, for this the trusted I/O paths that ARM TrustZone provides can be used.