Android Fingerprint Sensor: Pitfalls And Challenges

Luminița Apostol, Ciprian Dobre
Department of Computer Science and Engineering
University *Politehnica* of Bucharest
apostol.luminita@outlook.com, ciprian.dobre@cs.pub.ro

***Abstract*—In the last four years, the number of smartphones capable to read user fingerprints in order to perform authorization and authentication has tripled. Also, after the introduction of biometric payment services, fingerprint scanner driven authentication became more popular and the variety of smartphones equipped with this kind of sensor multiplied. However, the implementation methods are directly handled by manufacturers, who are often making decisions under the pressure of a short time-to-market. Starting from these facts, scholars and industry leading experts started to research the documentation in order to analyze the privacy and security flaws of the existing implementations. the assessment of Android good practice advises has revealed that some devices were not compliant, due to the fact that architecture could be attacked by an adversary both from the external world and by using malicious applications. Most common types of attacks against fingerprint authentication which could be demonstrated were aiming either to confuse the user in order to perform a malicious operation without a proper context or by employing custom-made molds of the user thumb minutiae. This paper analyses former and current issues affecting the fingerprint authentication in mobile devices powered by the Android operating system.**

***Index Terms*—Android operating system, fingerprint sensor, security flaws.**

I. Introduction

Fingerprint authentication is a biometric process which implies capturing, analyzing and comparing specific biological features belonging to the surface of a human finger, to uniquely identify a person. It is a well-known fact that fingerprints do not change during the lifetime and, therefore, they are used by authorities to associate biometric data with personal information (for example passport data and social security number). Biometric features can be used both for identification, when the fingerprint is correlated by matching its owner with data belonging to the enrolled users, and verification, which implies the comparing process of a scanned fingerprint to other locally stored ones.

Figure 1 outlines the fingerprint recognition architecture. This system is capable of determining the user's identity by analyzing and comparing the data collected from the sensor and the value calculated during the initial setting up phase.

During the *enrollment operation*, the hardware sensor scans the biometric and transfers the digitally transformed input to the *Minutiae Extractor*, which will process the given fingerprint in order to determine the number, the location and the direction of the unique thumb details, known as *minutiae points*. This friction ridges can vary from 20 to 70 minutiae points, depending on the quality and the size of the hardware sensor. [1]

The *identification phase* implies that the same user scans its fingerprint in order to generate a new input image, known as *query print*. This digital image is processed in order to extract the visible minutiae points. The *Matcher mechanism* stores the temporary values and compares them with the ones existing in the *enrollment database*. In order to correctly evaluate the input and the existing data, the Matcher must align the two fingerprints

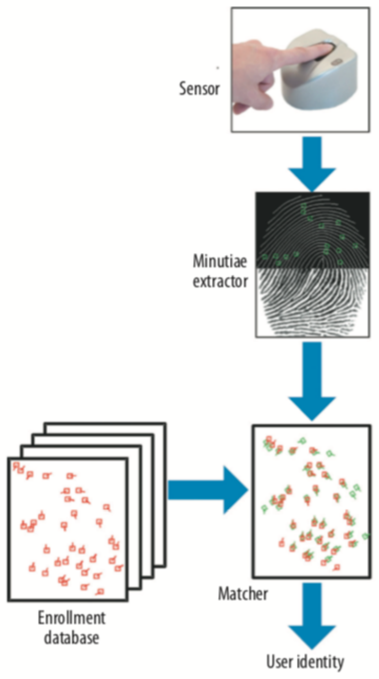

Fig. 1. Automated fingerprint recognition architecture

and determine if the minutiae points are located in the same place, the direction and the number are the same (figure 2).

However, in same cases (for example finger cuts or burns) the number of visible friction ridges may vary, but their location and direction will not change. Also, this approach cannot be applied to children fingerprints due to the fact that their thumbs are constantly growing in dimension, but the direction and the number of minutiae points remains the same during the lifetime.

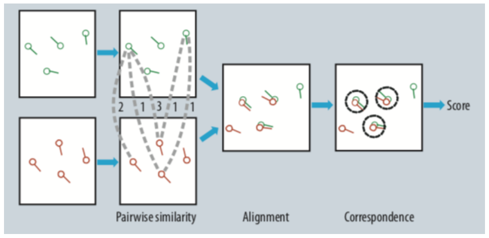

Fig. 2. Fingerprint matching processing

On a standard mobile device, a user can enroll one or more of its fingerprints in order to unlock the device or to authenticate itself with the help of an incorporated hardware sensor. According to a world-wide market study [2], the number of mobile terminals shipped with a fingerprint sensor has tripled between 2014 and 2018, reaching almost 60% of the market share. The most prevalent operating system used by smartphones between 2017 and 2018 is Android, which is installed on 76% of market available devices. [3].

The biometric authentication features supported by the Android operating system are implemented by each manufacturer, using proprietary algorithms and hardware components. However, in order to integrate with the current security architecture, they must implement a specific open-source hardware abstraction layer interface, which is managed by a service running as "*fingerprintd*" [4]. On most platforms, this service sends the requests to the hardware subsystem to enroll or authenticate a user and receive a status message through a callback function. The acquisition, processing, comparison and storage of the fingerprints is handled entirely by dedicated hardware and software, outside the operating system. The design of this flow aims that no sensitive biometric data can be leaked by an adversary.

Because this data is associated with the owner's identity, through social security number, immigration record or medical history, the fingerprint data internally stored on the mobile device must be secured against internal and external attacks. Unfortunately, previous work highlighted the fact that this implementation can allow some electrically conductive materials, molded in the victim's fingerprint, to fool the sensor and permit the access. Beside this, other pitfalls were proven, like:

- Confused Authorization Attack,
- Fingerprint Sensor Spying Attack,
- Trusted fingerprint sensors exposed to the untrustworthy world,
- Pre-embedding of some malicious fingerprints.

II. Flaws and challenges

Some of the first public research papers [5] [6] [7] [8] aimed to answer one question: "*Are fingerprint sensors as secure as passwords?*". Starting from there, the authors tried to determine if the hardware equipment can be fooled by home-made real copies of the human fingerprint. The main issue with biometric authentication is the fact that once your "*credentials*" are stolen, is impossible to change them. Also, in real life we leave many sets of fingerprints on different kinds of surfaces due to the natural oils which can be found on the surface of the human skin. In fact, the methods used by police authorities in order to determine if any traces (partial or full fingerprint) were left behind can be copied by an attacker in order to obtain a valid set of fingerprints.

There are two methods which can be used in order to create a dummy fingerprint (see figure 3):

- *With user's cooperation*. The human thumb is pressed into a plaster that will be used as a mold in which waterproof cement or liquid silicon will be poured. After the cement or silicon is hardened enough, the dummy thumb can be used.
- *Without user's cooperation*. The fingerprint image is collected from some hard surface, by enhancing its particularities with fine powder and moving it using scotch tape. After that, the print will be transferred to a photo sensitive copper plated circuit board (PCB) in order to obtain the mold. If the negative is not deepened enough, it can be enhanced using a Dremel multi-tool.

Most of these researches are focused on proof-of-concepts in order to demonstrate the success of "*dummy fingerprint attacks*" on the available types of sensors and architectures, testing their implementations on:

- *Personal Computers and Servers* which were equipped with external fingerprint sensors.
- *Mobile smartphones and tablets* with built-in biometric scanners. This category can include the laptops which have the fingerprint sensor installed by the manufacturer.

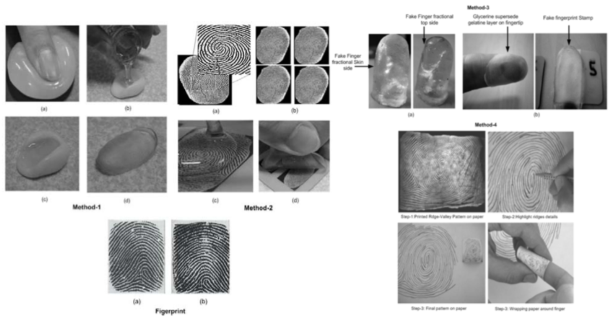
Method-1
Method-2
Method-3
Fake Finger fractional top side
Glycerine supersede gelatine layer on fingertip
Fake fingerprint Stamp
Fake Finger fractional Skin side
Method-4
Step-1 Printed Ridge-Valley Pattern on paper
Step-2 Highlight ridges details
Step-3: Final pattern on paper
Step-3: Wrapping paper around finger
Figerprint
(a)
(b)
(c)
(d)

Fig. 3. **Method-1**: The creation of fake fingerprint from plastic or silicon mould a) Pressing of finger on soft plastic silicon or rubber mold b) Dripping of liquid gelatin or rubber over mold c) Solidification d) Artificial fingerprint stamp. **Method-2**: a) Imaging of fingerprint from residual fingerprint b) Making and printing of fingerprint c) Masking and printing of fingerprint d) Detaching of fake fingerprint stamp. **Method-3**: GSG based fake finger and fingerprint (a) Glycerin based 3-D fake finger (b) Finger sample. **Method-4**: Four step fake fingerprint preparation to spoof the touch-less surrounded biometric system. Fingerprint: a) Live Finger b) Fake/Gummy Finger. [9]

- *IOT smart locks* used for access control systems.

Although the type of system which uses this kind of sensor might not seem important, its dimensions and resource constraints can have a huge impact over the security enhancements that can be applied in order to patch the dummy fingerprint vulnerability. For example, a smart lock or a computer can be upgraded with another scanner which is capable to detect the pulse, the blood vessels or other biometric features. Unfortunately, the size of a mobile device is around 4 to 9 inch, therefore the hardware which could be integrated is limited. Also, the battery consumption can grow proportionally with the number of authentication attempts. Due to these facts, the hardware upgrade approach is not suitable for smartphone or tablets.

In the past years, the authentication mechanism based on biometric analysis technology evolved, becoming more affordable, robust and easier to integrate into smaller spaces. Due to these factors, the technology has spread worldwide, being adopted by governments, public institutions and various private companies, which are currently relying on it to perform specific tasks. Since more and more people became familiar and even comfortable with fingerprint readers, the obvious step to widespread adoption was integration with existing hand-held devices. The major players in the market have begun to embed small fingerprint scanners into their mobile terminals. Currently, the only system which is available for security and implementation flaws analysis is the Android Operating System, due to its open source nature.

An Android application will use the Fingerprint Manager API in order to access the fingerprint authentication subsystem. A FingerprintManager instance can be obtained by every application which requests it, if it meets the required Android specific permissions. This instance further wraps native calls which communicates with an Android framework service, called Fingerprint Service. This service lives as a single entity in the context of the system process and forwards requests to the native daemon *fingerprintd*, which further communicates with the fingerprint hardware abstraction layer, a vendor-specific library which can dispatch, forward and translate proprietary messages to hardware devices (see figure 4). Even if manufacturers have proprietary implementations, they must follow an open-source specification for proper integration with the existing open-source code-base. This specification states that proprietary code must implement methods to manage biometric data without exposing the actual fingerprints outside the sensor.

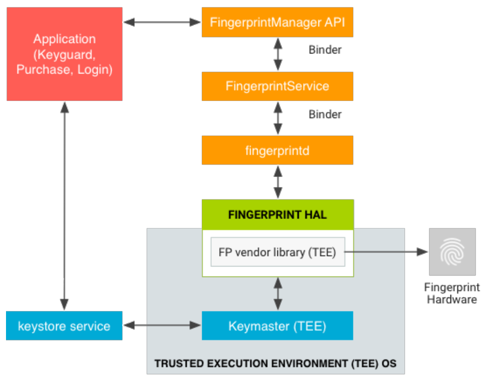

Fig. 4. Android fingerprint authorization data flow

The fingerprint reading subsystem produces an output, often referred as a template, which is essentially a collection of features uniquely identifying the finger of a user. To protect the confidentiality and integrity of the template, vendors use a technology called Trusted Execution Environment, which is employed to prevent unauthorized access to sensitive data from both user and kernel space. In our case, sensitive data refers to biometric information and cryptographic keys. Sometimes, due to strict space requirements, biometric information is stored on device memory, in a ciphered form, and is decrypted for various operations only when needed, using keys, algorithms and memory buffers managed exclusively by the Trusted Execution Environment.

The communication between the Trusted Execution Environment and kernel/user space is managed by a special middleware acting as a message broker between the two zones, to enforce security. This component is critical from a security standpoint; thus, it has a separated memory area and some implementations [11] can even switch the processor state from secure to non-secure mode, in order to prevent side-channel attacks.

Another security mechanism designed to guard native services on the Android platform, activated in enforcing mode beginning with the 4.4 version is SELINUX (Security Enhanced Linux), which is a mandatory access control system consisting of multiple components implemented in user-space and kernel space [12]. This system has been released on 22 December 2000 by NSA and is a part of the streamline Linux kernel since 2003. On Android, each native daemon has his own set of access rules, in order to prevent exploitation. Every per-service collection of rules is structured as a security context, specifying only the allowed actions. When an access rule is broken by a process, which is possible when an adversary manages to take control of the execution flow, SELINUX terminates the process, in order to prevent further damage.

However, researchers have managed to discover that various vendors did not to follow the public guidelines and implemented pure software biometric authentication systems, guarded by weak SELINUX policies, storing plain user fingerprints on the internal memory and thus making sensitive information prone to leakage [10]. Such vulnerabilities have been discovered by analyzing a mobile hand held produced by HTC, which was saving unencrypted fingerprint images in a world-readable location accidentally. These misconfigurations allowed the access to plain biometric data for every non-privileged application running in the operating system.

Other types of adversarial authentication bypass techniques are also feasible, like:

- *Confused Authorization Attack*. This type of attack is a technique use by malware-backed applications to trigger a biometric authentication prompt on the device screen. This prompt can be confusing due to the lack of a proper message and context, and, if the request is fulfilled by the user, it can trigger an adversarial action which previously needed authorization. In order to trigger the vulnerability, the victim must be tricked to install a malicious application. Such an application might determine the user to authorize a malicious financial transaction, by triggering an authentication prompt or by replacing a legitimate transaction.
- *Fingerprint Sensor Spying Attack*. On some implementation, the fingerprint sensor is not fully locked down, and is handled through the Linux Kernel, thus enabling attackers to sniff data passing on the Serial Parallel Interface (SPI) bus. This type of attack is very dangerous, allowing mass fingerprint harvesting. The impact of this attack is huge because the adversaries can keep and use this data for many years. Even if the sensor is locked down, a motivated attacker can replace the kernel in order to enable an event listener. This is particular dangerous in the case of a home-button mounted sensor, enabling the harvesting of fingerprints every time a user wants to perform an action involving it.
- *Pre-Embedding of the Fingerprints*. This one is a theoretical attack, and a subject of many disputes. Scholars and security experts are revealing the fact that it is possible that a vendor might embed a non-observable master fingerprint in the system, allowing thus access to the device for a certain group. Practically, the visible count of the enrolled fingerprints can be altered by modifying open-source components (the Settings application), in order to report a smaller number. Thus, the existence of pre-embedded fingerprints would be concealed to a user who is not willing to invest precious time doing reverse-engineering on that application.

Even if the system design follows best-practices, it is as secure as the trusted execution environment, a subsystem designed as a special area in a processor, not accessible to the operating system. In the past, there were some attacks targeting the Trusted Execution Environment (TEE), which were employing a downgrade procedure in order to install an older, vulnerable version [13] [14].

Starting from the TrustZone driver architecture (see figure 5), [13] research paper demonstrated that this secure execution environment can be altered and replace with a vulnerable one. This exploit is also known as "*the Trustzone downgrade attack*". Due to the fact that the boot image has access both to secure and untrusted areas, the downgrade vulnerability can be exploited if the following prerequisites are met:

- *Root the mobile device*. This step is necessary in order to gain elevated execution privileges.
- *Remount the file system*. Because the file system is mounted with read-only capabilities, the remount command will allow the attacker to write in any section of the file system (for example can be replaced the boot or recovery image, can read and write from or into data file system).
- *Replace the current trust environment image with the vulnerable ones*. Security patches are continuously installed into the mobile device operating system in order to resolve known vulnerability issues, so the current version of TrustZone does not allows access to its resources. Using older implementations, the attacker can exploit vulnerabilities which were patched since the trust environment was first developed.

The secure booting procedure is based on a chain of trust type of implementation, which means that every time the boot sequence is starting, each software image which will be executed is authenticated by another tool that was previously verified and trusted by the ecosystem. This design aims to prevent unauthorized or malicious code from being run. Unfortunately, since the boot image can be replaced with older ones, this approach is still susceptible to downgrade type attacks due to the fact that the previous images have well known and documented vulnerabilities which can be exploited in order to gain access to the system resources.

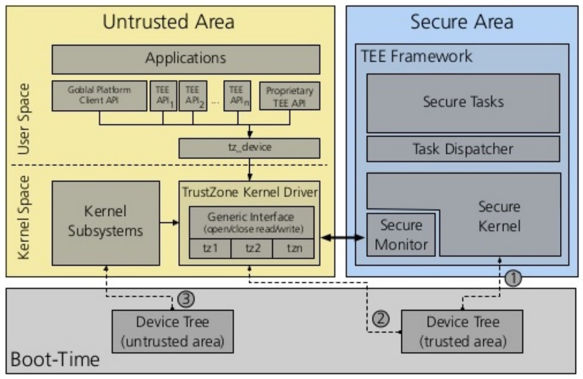

Fig. 5. TrustZone Driver Architecture [14]

To defeat the downgrade attack, manufacturers are using different key pairs for different versions. However, this approach is difficult to implement and maintain, due to the high complexity of the chain of trust, mobile device distribution methods and compatibility issues.

Another approach is to use version control for rollback prevention. Some TEE vendors have already implemented this mechanism, but this solution is not always practical. First of all, it may involve too many resources in order to fix the issue, especially on old mobile devices. Secondly, the success of a given exploit may depend on other unpatched vulnerabilities. So, if all the other vulnerabilities were fixed, the attack cannot be conducted successfully. In the end, the users may want to downgrade certain system images in order to meet their own requirements, which implies that this solution is not suited for any type of smartphone.

Confused Authorization Attack does not require any type of privileged rights in order to be conducted from a regular Android application. Due to its design, the authentication system does not offer any proof regarding what is actually happening under the biometric prompt. This behavior might confuse a non-technical user, and may succeed in defeating the whole purpose of authorization. If the phone is rooted, an adversary might replace financial transaction data, in order to perform a fraudulent payment. In the latest versions, Android Pay system is not working if any critical security component was altered.

In contrast, the *Fingerprint Sensor Spying Attack* is a very complex type of attack conducted by a highly motivated attacker, with deep knowledge regarding hardware and kernel internals. This type of attack is highly improbable in the context of the latest Android security patches.

Even if it is just a theoretical attack, the *Pre-Embedding of the Fingerprints* is a highly feasible type of attack which could be conducted by a fingerprint scanner manufacturer. Since these scanners are closed-source, and sometimes can store the fingerprints independently, they are the perfect candidate for such suppositions. However, storing fingerprints inside the sensors without making them accessible to the outside world might be considered a security enhancement, preventing biometric harvesting. Also, the pre-embedding of fingerprints might be considered useful for law enforcement.

Does and Maarse research paper [15] provided a small proof-of-concept that can demonstrate that FingerprintService can be modified in order to fake a successful authentication response by running a modified *fingerprintd* service. In their experiment, the source code of the binary (*fingerprintd* is provided by the Android Open Source Project (AOSP) source code) was modified in order to always return a fingerprint ID value which is not equal to "*0*". In consequent, all authentication attempts started by the FingerprintService assumes the user authentication process is successfully executed and forwards the result to the application which requested that user's identity verification (see figure 6). The service modified by Does and Maarse [15] always returns the value "*42*" for the fingerprint ID to the FingerprintService. Because this value is greater then "*0*", the authentication process appears to always succeed even though the scanned fingerprint was not enrolled.

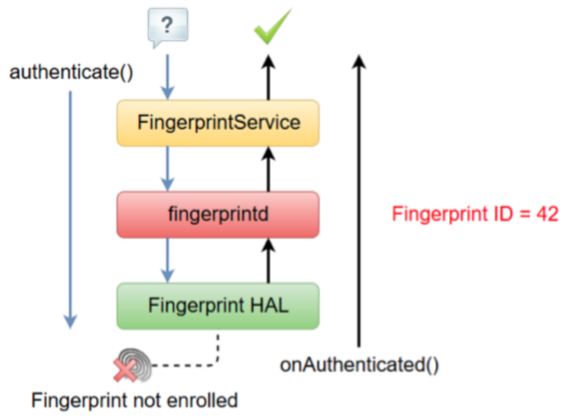

Fig. 6. Modified callback response work-flow from *fingerprintd* service [15]

After compiling the *fingerprintd* daemon source code, the original binary was replaced with the modified one. In order to execute the malicious service, the AOSP image was compiled to assign it the expected privileges of the genuine daemon. Unfortunately, even if the user does not modifies the source code of the *fingerprintd* daemon, the problem associated with the custom ROM image installation relies on the fact that the compiled product cannot be reviewed, so that product may contain malicious applications or backdoor which could be exploited in order to gain root permissions. Also, the elevated execution privileges could allow an attacker to target and replace the *fingerprintd* service through social engineering or other physical access attack.

III. Technological enhancements

A individual human being can be identified based on the information provided by answering the following questions: "*What do you have?*" and "*What do you know?*". So, any person has unique features which can be correlated with the context and knowledge about him or her, in order to authenticate and match the digital information with the real human being. In order to ensure a higher degree of trust and security for the biometric data, the current implementation can incorporate some software and hardware enhancements, like:

- Vendors should improve the spoof rejection rate, in order to avoid successful authentication with plaster molds. This can be achieved by incorporating hardware equipment capable of detecting the vein configuration or other particularities which can uniquely define a user.

- **Open-sourcing the fingerprint matching algorithms implementation in order to allow peer and community review. The compiled firmware can be cryptographically signed in order to provide integrity checks and non-repudiation, limiting thus malicious updates. The firmware update should not be encrypted because the non-readable output can hide backdoors.**
- **The fingerprint authentication mechanism should not work when the operating system is altered (for example custom ROM is installed), root binary is present or when the TrustZone version is lower than the one with downgrade attack patch.**
- **The fingerprint authentication user interface (UI) should display the name and the package name of the process which requested this action. Also, the user should be able, through an administrative UI, to overview all the past authorization and authentication access which were made. This way, the owner of mobile device will be able to discern if the requested fingerprint scanning request is a legitimate one or if the process is malicious.**
- **The data transmission between the secure zone to the untrusted outside world should be done using secure implementation which should not be susceptible to Inter-Process Communication (IPC) interception, due to the fact that during the information transfer between the processes, the sensitive values could be modified by a malicious binding daemon.**

IV. Conclusion and Future work

The biometric authentication and authorization process using fingerprint scanners has been widely adopted due to multiple advantages over traditional password authentication. However, it is a well-known fact that adversarial techniques are constantly evolving with technology, and therefore, every authentication system should be the subject of future improvements. Even if vendors choose to enhance their technology, "*a chain is as strong as its weakest link*". Starting from this statement, we should regard security as a whole, and not ignore the adjacent mobile terminal building blocks. Research for future improvements and active hardening of existing software and hardware against malicious actions should be taken in consideration by every manufacturer.

From the user's point of view, it is important to check for updates and patches and install them in order to actively improve data confidentiality, integrity and availability. Also, modifying the OS or hardware components can conduct to major issues, due to the fact that extended execution privileges can be used both by the legit user and by any malicious application exiting in the ecosystem.

Even though software updates can help preventing or stopping biometric related vulnerabilities from further spreading, the best approach in order to overcome the current implementation flaws and challenges is to enforce the security of the data transfer from the trusted hardware environment to modifiable software processes and services.

References

[1] *Fingerprint Matching* http://biometrics.cse.msu.edu/Publications/Fingerprint/JainFpMatching_IEEEComp10.pdf

[2] *Global penetration of smartphones with fingerprint sensors, 2014-2018* https://www.statista.com/statistics/522058/global-smartphone-fingerprint-penetration

[3] *Mobile Operating System Market Share Worldwide* http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201708-201808

[4] *Fingerprint HAL* https://source.android.com/security/authentication/fingerprint-hal

[5] Lisa Thalheim, Jan Krissler & Peter-Michael Ziegler *Biometric access protection devices and their programs put to the test* http: //www.heise.de/ct/english/02/11/114/

[6] Matsumoto T., Matsumoto H., Yamada K. & Hoshino S. *Impact of artificial" gummy" fingers on fingerprint systems* Optical Security and Counterfeit Deterrence Techniques IV (Vol. 4677, pp. 275-290). International Society for Optics and Photonics.

[7] Marie Sandstrom *Liveness detection in fingerprint recognition systems* Master's thesis, Linkoping Tekniska Hogskola, 2004

[8] Wiehe A., Søndrol T., Olsen O. K. & Skarderud F. *Attacking fingerprint sensors* Gjøvik University College, December 2004.

[9] International Frequency Sensors Association (IFSA) *Sensors & Transducers Journal - Volume 1/2012* http://www.sensorsportal.com/HTML/DIGEST/january_2012/P_907.pdf

[10] Wei, T. & Zhang, Y. *Fingerprints on Mobile Devices: Abusing and Leaking* Blackhat USA, 2015

[11] *ARM Security Technology Building a Secure System using TrustZone Technology* http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/CACJBHJA.html

[12] Bill McCarty *Selinux: NSA's Open Source Security Enhanced Linux* O'Reilly, 2005

[13] Chen, Y., Zhang, Y., Wang, Z. & Wei, T. *Downgrade Attack on TrustZone* arXiv preprint arXiv:1707.05082, 2017

[14] Javier González *Operating System Support for Run-Time Security with a Trusted Execution Environment (Ph.D Thesis)* https://www.slideshare.net/JavierGonzlez49/operating-system-support-for-runtime-security-with-a-trusted-execution-environment-phd-thesis

[15] Does T. & Maarse M. *Subverting Android 6.0 fingerprint authentication* Doctoral dissertation, Master Thesis at University of Amsterdam, 2016