OULU UNIVERSITY OF
APPLIED SCIENCES

Srinivas Vijayanand Nayani

DESIGNING SECURE SOLUTIONS FOR EMBEDDED SYSTEMS

DESIGNING SECURE SOLUTIONS FOR EMBEDDED SYSTEMS

Srinivas Vijayanand Nayani
Master's Thesis
Autumn 2017
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology

Author: Nayani Srinivas Vijayanand
Title of the master’s thesis: Designing Secure Solutions for Embedded Systems
Supervisor: Teemu Korpela
Term and year of completion: Autumn 2017 Number of pages: 75

The aim of the thesis was to explore and develop an insight on various security aspects and practices that need to be considered while designing embedded products. The thesis was commissioned by Bittium Oy.

Design requirements were gathered from the customer of the project and they were further consolidated based on industry practices and standards. Solutions were developed in C language over Linux operating system using tools; Eclipse IDE, Hardware Debuggers, version control tools GIT and Gerrit for continuous integration methodology. The project was executed in the Iterative software development lifecycle model.

The project was completed successfully and delivered to the client by 2016. This thesis also extends the core topic a bit further by discussing some of the best practices for designing secure products and some of the future roadmaps and technologies in the area as part of conclusion.

Keywords:
Embedded security, Attacks and vulnerabilities, Trust execution environment, Multiple Independent Levels of Security, ARM TrustZone, Root of trust

TABLE OF CONTENTS

VOCABULARY

AppArmor	A Linux Security Module
BYOD	Bring Your Own Device
CAPEC	Common Attack Pattern Enumeration And Classification
CVE	Common Vulnerability Enumeration
CWE	Common Weakness Enumeration
DoS	Denial Of Service
eLibc	Embedded C library
Gerrit	Code collaboration tool
GIT	Distributed version control system for development
IoT	Internet of things
PLOVER	Primary list of vulnerabilities examples for researchers.
Meego	Linux distribution for mobile devices
MILS	Multiple Independent Levels Of Secu-rity
SDLC	Software Development Lifecycle
SE	Secure Element

SELinux	Security Enhanced Linux
SoC	System On Chip
SK	Separation Kernel
Stuxnet	Computer worm that targets Industrial networks
SUDO	Sudo access rights
TEE	Trusted Execution Environment
TPM	Trusted Platform Module

1 INTRODUCTION

Security in embedded systems has been gaining attention every passing day. With the advent of IoT and cloud technologies, security is ubiquitous and has found its way in most of the fields like telecom, aerospace, healthcare, smart, wearable devices and defence. Its presence is set to increase with the upcoming technologies, such as like Cloud, robotics, machine learning and AI, and it has not hit the peak.

Given its importance, security should be treated as an integral part of design which should be considered right from the product conception stage. Security should be built into the system across multiple levels, often referred as a layered approach. It is not feasible or often viable to design a totally fool-proof system. Thus, designers should rather focus on systems that can be difficult to compromise and can reduce the risk to an acceptable level. Amit and Barnum state (1.) that it is much easier to find vulnerabilities in soft-ware than it is to make software secure. This can be applied when designing the product end to end and it is not limited to just software.

2 DRIVERS FOR DEVICE SECURITY

In the current IoT (Internet of Things) age, it is possible to we pretty much can assume that almost every entity acts as a connected node in the network. The list includes but is not limited to, mobile phones, surveillance IP cameras, consumer appliances, parking spaces, power plants, cars and mass transportation vehicles. (2.) By the year 2020, analysts estimate that there could be 50 billion Internet connected devices which is only bound to grow. (2.)

This also means that this increasing number of nodes needs to be protected in the network: not only to establish a secure, reliable communication but also run a device without compromising the device’s integrity and authenticity. These are the primary non-functional requirements that are targeted to be achieved by the security solutions.

Devices can be compromised either by running unauthorized software or by software being subject to hacking. Compromised devices can result in exposing users or organization secrets, such as cryptographic keys, information stealing and assets loss. This can have wider financial implications. It comprises user privacy and creates a blow to confidentiality. Thus, it is extremely important to have appropriate security solutions that can detect and repel any such attacks, and perform a damage control and an incident response mechanism in the event of any such attacks.

3 SECURITY ATTACKS

The formal attack terminology widely used by industry consists of attack trees, attack windows and patterns that are used to describe any attacks.

Attacks can be classified based on certain parameters, such as attacker's objectives, impacts, Origin, phase of introduction, technology employed and exploitability. The purpose of any attacks is to achieve a desired goal that can range from data stealing, inflicting serious damage and crippling the system, hobby attacks and academic or institutional attacks as part of research to expose vulnerabilities. In the attack terminology, the purpose of an attack is viewed as a root. And by initiating an attack, attacker's target is to reach the root of the attack tree.

Attack trees also provide a formal and methodical way of describing the security of systems based on varying attacks. (3.) In an attack tree with only "or" branches, this consists of all paths from a leaf node to the root node. Such paths are also known as "attack paths." In a tree with some "and" branches, an attack pattern may be a sub-tree of the attack tree that includes the root node and at least one leaf node. (1.) Attack patterns provide a coherent way of teaching designers how their systems may be attacked and how they can effectively defend them. (3.) Different attack types are described below in detail

3.1 Attack types

3.1.1 Focused attack

Focused attacks are highly focused on particular or specific kinds of systems or environments or ecosystems. This kind of attack has no limitation on time, money and resources. The most practical examples of these kinds of attacks are to targeting to defence installations and penetrating enemy communication lines. The most recent example of this kind of attack is building of "stuxnet" which is targeted to attack at only Siemens systems in Iran.

3.1.2 Cryptanalytic attacks

Cryptanalytic is a study of techniques to unravel the meaning of an encrypted text without access to secret keys. Cryptanalysis techniques are used to decrypt the ciphered text without really accessing the encryption keys. Doing a cryptanalysis requires working knowledge of the system and knowing the internals of cryptography which in practice means uncovering the secret key. These attacks are briefly classified as plain and cipher text attacks and are explained below.

Known plain text attack
In this kind of attack, the attacker will have access to at least one pair of a plain text and corresponding cipher text which are not explicitly chosen and act as inputs for further analysis. (43.) These plain texts are usually obtained via eavesdropping or from parities who already possess the encryption key. The results are used to break the rest of the encryption in the system by tracing out the secret key.

Chosen plain text attack
A chosen-plaintext attack (CPA) is an attack model for a cryptanalysis which presumes that the attacker can obtain the cipher texts for arbitrary plaintexts. (4.) In this kind of attack, the attacker feeds a pre-chosen text into the cipher oracle after which they analyse the result and in the worst case they can find the secret key.

Two forms of chosen plain text attacks are a batch chosen plain text attack and an adaptive plain text attack. The batch chosen plain text chooses all the plain texts before it analyses the ciphered text where as in the adaptive chosen text attack, a cryptanalyst requests for additional cipher texts after analysing the results of previous cipher operations.

Known cipher text attacks
Under known cipher text attacks also referred as Ciphertext only attack (COA), the attacker has access to bunch of cipher texts mostly obtained either by eavesdropping or stealing. Also here the attacker will not have access to more

cipher texts or will not have the luxury of choosing a cipher text or producing more. In this attack, attacker has access only to the encrypted message, as the language is known a frequency analysis could be attempted. In this situation the attacker does not know anything about the contents of the message, and must work from ciphertext only. (44.) This is certainly one of the weakest attacks as the attacker will have nothing to work against other than a few cipher texts in hand.

Chosen cipher text attacks (CCA)

A chosen cipher text attack is a scenario in which the attacker has the ability to choose cipher texts and to view their corresponding decryption's plaintext. (5.) In this kind of attack, the attacker selects the cipher text, sends it to the victim, and is given in return the corresponding plaintext or some part thereof. (6.)

It is essentially the same scenario as a chosen plaintext attack but it is applied to a decryption function instead of the encryption function, which means that the attacker will be able to produce a clear text from a set of pre-selected ciphered text messages from the decryption oracle.

Chosen cipher text attacks can be adaptive or non-adaptive. Under non adaptive cipher text attacks, the attacker chooses certain cipher texts in advance for decrypting them. The clear texts obtained are not used for the next cipher operations. A chosen-plaintext attack is called adaptive if the attacker can choose the cipher texts depending on the previous outcomes of the attack. (6.) Servers using the Cipher Block Chaining (CBC) mode of operation and RSA PKCS1 are under certain circumstances vulnerable to adaptive chosen-cipher text attacks and such attacks allow an attacker to recover the encrypted data. (7.) This means that adaptive cipher text attacks are more context based and a result of outcome of previous operations. These attacks may be quite practical in the public-key setting. Bleichenbacher (8.) demonstrated that a plain RSA is vulnerable to a chosen cipher text attack and some implementations of RSA may also be vulnerable to an adaptive chosen cipher text attack.

The chances of finding the secret key with chosen cipher-text attacks is more than simple plain text attack.

Lunchtime Attack

Lunchtime attacks are also referred to as a midnight attack or a CCA1 attack. This kind of attack is targeted by attackers when the owner or user of the system is away or often when the system is not logged in. The idea is that the system is vulnerable and there is often less or no resistance when there is no active user who otherwise would be more challenging to penetrate. During this time, the attacker will generate pre-chosen cipher text quarries which are valid until the period of time after which penetrating will be difficult.

Adaptive cipher text attack

An adaptive cipher text attack is also referred as a CCA2 attack and it is stronger in nature compared to CCA1. This attack relies on selecting a cipher dynamically at runtime whenever the attacker is posed of challenge.

This is an interactive based attack where the attacker sends a stream of ciphered texts to be decrypted and subsequent ciphered texts are chosen depending on the responses from the system (7.).

In an increasing order from weakness to strength, the above attacks can be sorted as a known cipher text attack, a known plain text attack, a chosen plain text attack and a chosen cipher text attack.

3.1.3 Network attacks

In this world of ever increasing networking, systems have become very attractive targets for external attacks through the network. Networking attacks mostly rely on monitoring, spoofing or masquerading of network traffic.

Passive attacks

A passive attack monitors the unprotected or weekly encrypted communication between two nodes for capturing authentication information or passwords which can be passed to parties who would or have the ability to compromise the system.

Active attacks

Active attackers penetrate the system by circumventing the security and breaking the protection of existing systems. They can cause undesired effects by executing their malicious code and injecting viruses or Trojan horses. Active attacks can have varied effects right from minor ones to bringing down the whole system or network especially if the attack is on servers.

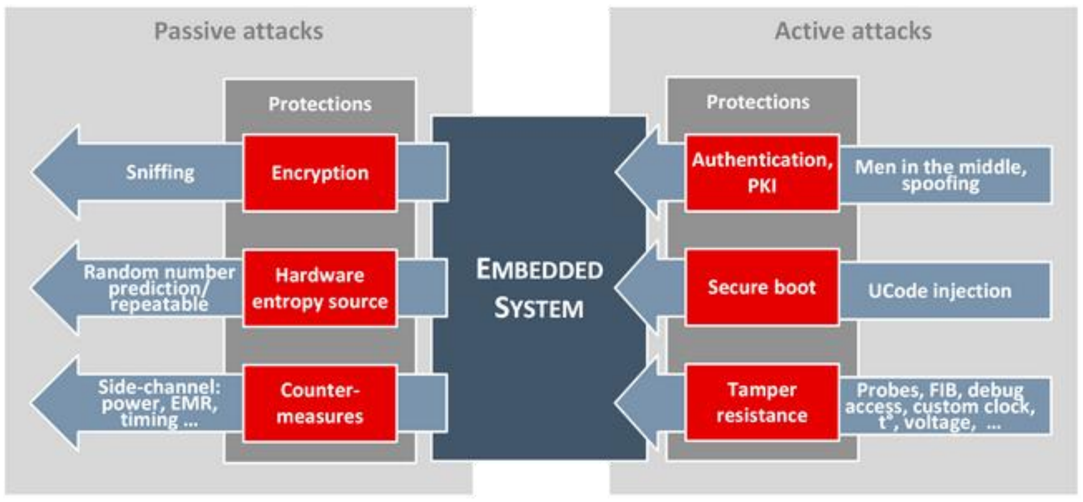


FIGURE 1. Example of attacks on embedded systems (11.)

Insider attack

These are the attacks that are perpetrated from inside the organization or persons who have genuine access to the system. Insider attacks come from disloyal persons, persons with malicious intent or dissatisfied employees inside from an organization.

Phishing attack

Phishing attacks are attacks where the attacker will design fake almost identical web sites through which they direct users to login with credentials. These credentials will be recorded and used by attackers to log into proper websites which can result in stealing of vital information or can even result in financial frauds if the target is banking sites.

Hijack and spoof attacks
An IP spoofing attack is one in which the source IP address of a packet is forged. There are generally two types of spoofing attacks: IP spoofing used in DoS attacks and man in the middle attacks (9.).

The attacker can hijack the communication sessions and disconnect the one of the node. Under a hijacked session, other connected party is still under the impression that they are communicating with the original party and can still pass some vital private or secret information.

3.2 Classification of attackers

Class1: Clever Outsiders
Outsiders are external attackers exploiting a certain weakness in the system, they have certain knowledge of the system.

Class2: Knowledgeable insiders
Insider attackers are the ones who have the needed technical expertise as well as access to specialized tools to break into the system.

Class3: Funded Organizations
Organizations fund attacks with focused objectives. Here, attackers have in-depth knowledge and no restriction on funding. They are also equipped with highly specialized tools to break open the systems.

TABLE 1: Classification of attacks calibrated against different parameters, demonstrated in black hat conference by Joe Grand (10.).

Resource	**Hacker class (Class1)**	**Academic (Class 2)**	**Organized (Class 3)**	**Government (Class 4)**
Time	Limited	Moderate	Large	Large

Budget	< $1000	$10K – $100k	>$100k	Unknown
Creativity	Varies	High	Varies	Varies
Detectability	High	High	Low	Low
Target	Challenge	Publicity	Money	Varies
Number	Many	Moderate	Few	Unknown
Organized	No	No	Yes	Yes
Release Info	Yes	Yes	Varies	No

3.3 Levels of difficult

TABLE 2. Attack difficulty (10.)

Level	Name	Description
1	None	Primitive ,No special tools or skills needed
2	Intent	Minimal skills needed to compromise the system
3	Common tools	Technically competent, Can be dealt with tools available in the market
4	Unusual tools	Can be compromised with tools, is available to most people
5	Special tools	With specialized tools and with expertize only available in universities or government
6	In Laboratory	Major effort needed, Only available to few facilities in the world.

3.4 Attack vectors

Attack vectors are typical routes via which an attacker can gain access and after that exploit the vulnerabilities in the system in order to achieve their objective. Typical attack vectors are eves dropping, brute force attacks, injecting crafted packets, reverse engineering and fabrication by counterfeit assets of a product.

4 CLASSIFIYING VULNERABILITIES

According to the CVE website, a vulnerability is a mistake in software code that provides an attacker with direct access to a system or network. For example, the vulnerability may allow an attacker to pose as a super user or system administrator to gain full access privileges. (12.)

An exposure, on the other hand, is defined as a mistake in software code or configuration that provides an attacker with indirect access to a system or network. (12.) Classification of attacks and vulnerabilities can help to understand the tools, remedies and approaches that can help us to contain, trace and overcome these vulnerabilities.

4.1 Classification by SDLC (Software Development Lifecycle)

Vulnerabilities can be classified based on the phase of the software development life-cycle they usually appear. Some of the popular lifecycle phases that a have higher chance to see system vulnerabilities are design, testing, deployment and maintenance phases. (45.)

Typical examples of vulnerabilities in a design phase can include a wrong choice of OSS components, protocol and algorithms, using of untested reusable libraries or code that may not be really a fool proof. Often vulnerabilities in design phases are easy to exploit and difficult to plug.

During a maintenance phase, systems can be prone to vulnerabilities that come along with improper bug fixes and components that are not updated on timely and priority basis. In case the system is using OSS or reusable closed sources, system integrators need to update the system with the latest fixes from upstream. Leaving them unmaintained can expose holes in the system makes the system an attractive target for exploit. Vulnerabilities that appear due to coding errors, relaxed compiler settings and bad design of APIs are few that evolve during the implementation phase. Some of these can be addressed by employing a stricter process approach towards using code analysis tools and fine tuning the compiler optimization options.

In a typical software development life-cycle, system shall be configured to relax certain hard rules to accommodate testing and debugging. Eventually, If these are not closed can act as potential attack points. E.g. opening debug ports, opening access points in hardware.

4.2 Classification by attackers objective

One of the most prominent approaches is to enumerate the attacks by attackers objectives, such as gaining root access and higher privileges, creating a denial of service, stealing confidential and sensitive data, executing malicious code and Integrity and security policy violation.

4.3 Classification by attacks by their location in OSI model and their origin

One approach of classifying the vulnerabilities is by deciding where exactly they appear in the 7 layered OSI reference model. In practice this means segregating the vulnerability into one of the seven baskets (application, presentation, session, network, link and physical layer). Attacks can also be enumerated depending on their original location in the network, such as a local system, intranet (Ethernet network), an internet, particular protocol (e.g. TCP/IP) and a wireless network. (46.)

Mitigations for addressing these exploits can be implemented on different objects at different layers in OSI model. A mitigation may be applied at the source code level that eliminates a security flaw and the associated vulnerability, or a work-around can be applied at a system or network level to prevent the security flaw from being exploited. (47.) But this kind of classification may not be appropriate at all times as many of the times vulnerabilities can fall between layers and hence, it is difficult to segregate on this layered approach. For example, it is not often easy to decide on whether the vulnerability is exactly in the OS in the application layer or both as contention results which is right and wrong.

4.4 Classification by effected technology

Systems get vulnerable though holes created by string exploits and buffer overflows which are not unusual in the C language. (47.) Likewise, systems can be prone to attacks that exploit meta characters vulnerabilities in the LDAP and SQL Injection with database languages.

Systems that run code with memory leaks and malicious code are vulnerable to a resource exhaustion and thus can block the system resources resulting in DoS attacks. This kind of technology classification may not suit for vulnerabilities that spawn across technologies, not just limited to one.

4.5 Classification by errors

System can be made vulnerable as a result of improper code design and programmatic errors that appear and during the implementation phase. A few of them are freeing the memory more than once and executing code from malicious memory locations or locations program either did not allocate or have any control on.

At times, unclosed holes that are left in production code to accommodate debugging for diagnostic purposes can spell trouble and can be exploited by attackers. (47.)

4.6 Classification by enabled attack scenario

Vulnerabilities can also be classified based on precise type of attack scenarios. As seen above, Denial of Service is an effect that can happen due to multiple reasons, such as memory leaks or buffer overflows. Thus, it is sensible to classify the vulnerabilities on the nature of an attack scenario rather than based on effects.

Examples of attack scenarios are cryptographic attacks, network attacks, secure storage attacks, Software attacks, entropy attacks and malicious code execution. Techniques employed to execute these attacks can be chosen-known/cipher text attacks, compromised boot sequence and string exploits.

4.7 CLASP Classification

CLASP (Comprehensive light weight application security process) enumerates vulnerabilities based on software events and conditions that are responsible for the vulnerability. (13.)

4.8 Range and type errors

Generic range type errors are errors due to buffer overflow , stack and heap overflow , integer overflow , truncation errors, signed and unsigned errors, integer coercion errors, unchecked indexing of arrays, NULL character misplacing for string buffers , NULL pointer dereferencing , usage of freed memory, format string and code injection into data areas of memory. (47.)

4.9 Environmental errors

The vulnerabilities can be classified based on environment condition in which the application is deployed. The same program image can have different properties when installed in different environments (for e.g. as a result of dynamic runtime linkage). (47.) Environmental errors could also occur due to resources exhaustion (e.g. sockets, kernel objects, file descriptors, memory), execution of untrusted code and data, system variable manipulations (e.g. system paths, library paths), spoofing of system events, a failure to protect secure data and keys, a TRNG generation failure and insufficient entropy for PRNG.

4.10 Synchronization and timing errors

Situations leading to synchronization and timing errors are race conditions in code (unlocking code via kernel objects) , race condition in signal handlers, improper references for symbolic names which change at runtime , failure to drop user privileges at right times soon after task is accomplished, leaking sensitive information through error messages , time to check and time to use errors (for example , resources can change their state between a window of time lag between their validation and actual usage). Potential race conditions in the design could also bring up new vulnerabilities that attackers tend to exploit. At an even higher level of abstraction, these vulnerabilities could be classified as a logic or design error, since a resource (in this case, the file or socket) can be deleted while in use. (47.) Seacord & Householder in his work (47.) stated that, the lowest level of abstraction this vulnerability could be classified as an input validation problem, since the programmer fails to ensure that the object being validated is the same object the (potentially) insecure operation is performed on.

4.11 Protocol Errors (46.)

Protocols errors are the ones that usually arise out of protocol, algorithm errors that are as a result of improper use or wrong choices. (46.) Such vulnerabilities are from failure to check for certification expiration and revocation, key exchange without proper authentication, failure to encrypt communication, failure to do integrity check where ever needed, usage of hardcoded and stored passwords or keys, trusting certain IP address or range of IPs that can be spoofed easily, using of broken, week or risker cryptographic algorithms, improper usage of OSS components and failure to protect confidential and sensitive data.

4.12 Generic Errors

Errors that are enumerated based on their generic nature are improper error and exception handling, improper break and jump instructions in code, ignoring return values from functions ,uninitialized variables, failure to free unused resources and memory and unintentional assignment when comparison two values etc. (.47)

4.13 Popular Dictionaries for Attack Taxonomy

MITRE is a government funded non-profit organization which publishes and controls standards to be used by community. Below are the popular dictionaries of publicly known information security vulnerabilities and exposures maintained by MITRE. (14.)

4.13.1 PLOVER

PLOVER is a primary list of working examples intended for researchers. It lists over 1,400 real world vulnerabilities by their CVE IDs organised as a conceptual framework. This framework offers a platform for discussion for a further analysis and describing them in a further detailed manner. PLOVER is targeted at those who are engaged in analysing vulnerabilities to understand and communicate them in a more abstract level. (14.)

CVE

CVE stands for Common Venerability Enumeration, which precisely describes a certain pin-pointed instance in a system or a vendor through which exploits can happen. For example, CVE-2015-7547, points to a specific venerable instance in eglibc for an attack.

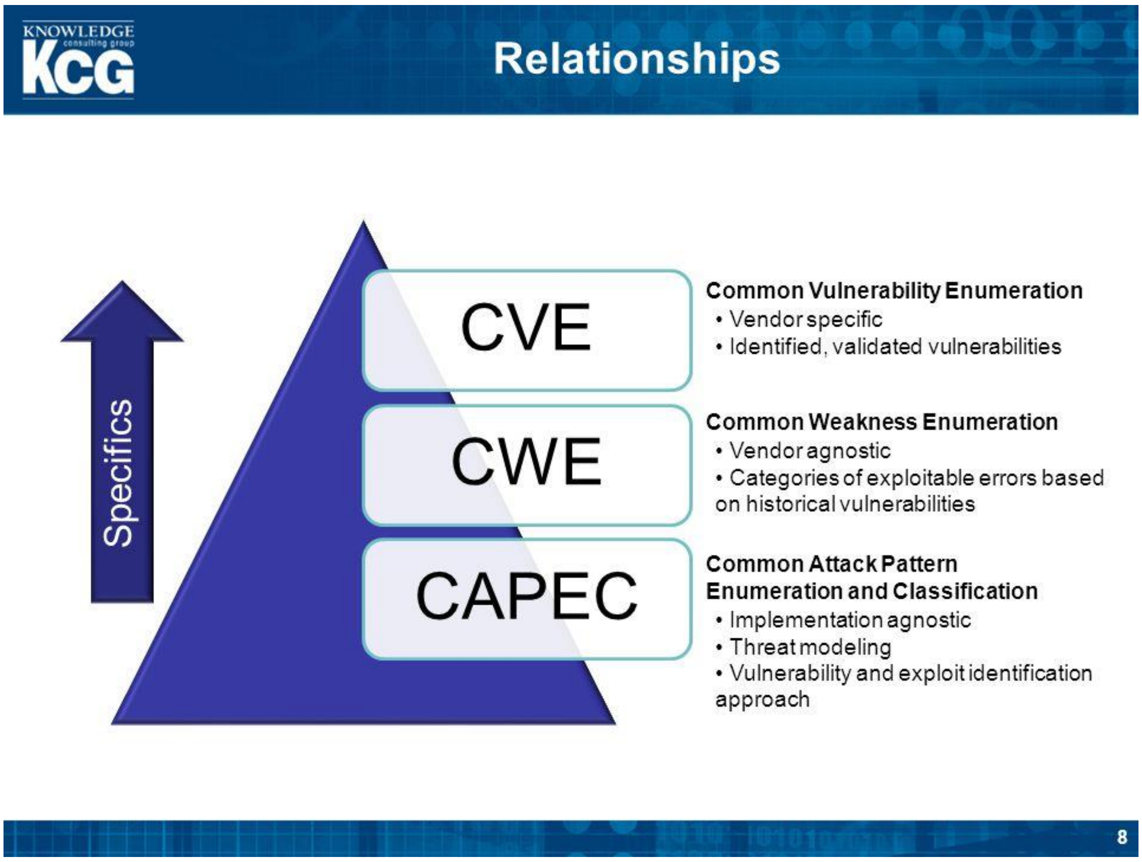
4.13.2 CWE

CWE stands for Common Weakness Enumeration, which essentially deals with underlying software weakness in general but not specific to a particular instance in a system. Vulnerabilities can emerge from weakness and they may have potential for getting targeted. The goal of CWE is to educate the programmers or system designers to For example, CWE-367 is time to check and time to use weakness spotted in software but it is not limited to any specific component or instance in any system.

4.13.3 CAPEC

To respond to attacks effectively, the community needs to think outside of the box and have a firm grasp of the attacker's perspective and the approaches used to exploit software systems. (15.) CAPEC provides this information to the community in order to help to enhance security throughout the software development lifecycle and to support the needs of developers, testers, and educators. (15.) Therefore, CAPEC is a publicly available catalogue of attack patterns along with a comprehensive schema and classification taxonomy created to assist in the building of secure software. (15.) While CWE is a list of software weakness types, Common Attack Pattern Enumeration and Classification (CAPEC™) is a list of the most common methods attackers use to exploit vulnerabilities resulting from CWEs. Used together, CWE and CAPEC provide understanding and guidance to software development personnel of all levels as to where and how their software is likely to be attacked, thereby equipping them with the information they need to help them build more secure software.

Below is the pyramid described by Knowledge Consulting group (16.) on how CWE- CVE- CAPEC stack each other.



Relationships

Common Vulnerability Enumeration
- Vendor specific
- Identified, validated vulnerabilities

Common Weakness Enumeration
- Vendor agnostic
- Categories of exploitable errors based on historical vulnerabilities

Common Attack Pattern Enumeration and Classification
- Implementation agnostic
- Threat modeling
- Vulnerability and exploit identification approach

FIGURE 2. Relationship between CWE, CVE, CAPEC (16)

5 PRINCIPLES OF SECURITY

Primary principles of security is to preserve the confidentiality, integrity and availability of the consumers, stakeholders and devices in the operating ecosystem.

"The principle of information security protection of confidentiality, integrity, and availability cannot be overemphasized: This is central to all studies and practices in IS. You'll often see the term CIA triad to illustrate the overall goals for IS throughout the research, guidance, and practices you encounter."—Three goals of security by Jim Breithaupt & Mark S. Merkow are shown in figure 3.

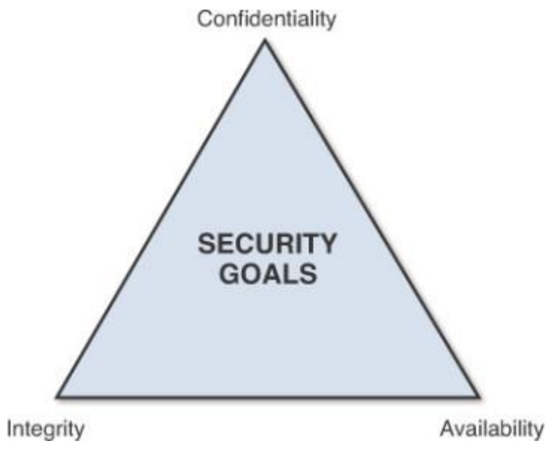


FIGURE 3. CIA Triad (17.)

5.1 Confidentiality

Confidentially refers to a restriction of access to sensitive data from unauthorized people. Loss of sensitive information, such as device or user passwords, credit card and banking credentials could result in identity theft, financial implications and compromised privacy.

According to Mathew and Stan (18.), designers should strongly consider measures to ensure confidentiality and prevent sensitive information from reaching unauthorized people, while simultaneously making sure that the right people can in fact get it: Access must be restricted only to those authorized to use the data in question. It is common, as well, for data to be categorized according to the amount and type of damage that could be done should it fall into unintended hands. (18.)

5.2 Integrity

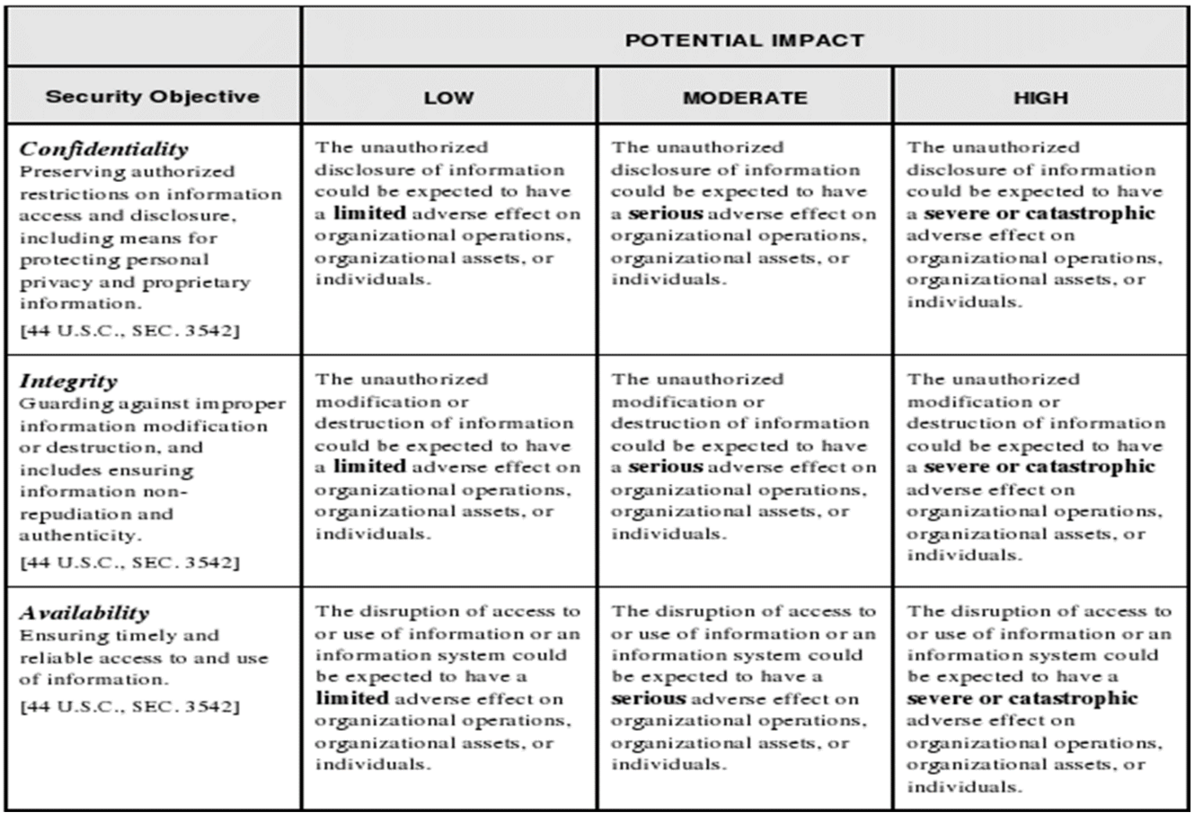
According to study published in a security blog by Mathew and Stan (18.), integrity involves maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle. Data generated at a source must not be changed in transit, and steps must be taken to ensure that data cannot be altered by unauthorized people (for example, in a breach of confidentiality). For ensuring data integrity designers can employ the measures like, consider file permissions, user access controls, employing cryptographic checksums for verification. Also, designers need to plan for backups in the event of any damage to data to restore it to its correct state. (18.)

5.3 Availability

Terry refers to availability as one of the pillars in the CIA traid (19.), According to a report published by her, availability of information refers to ensuring that authorized parties are able to access the information when needed. This can also be applied just not only to information, but also to systems that cater services. DoS attacks on such systems will deprive and deny users to access systems, thus hurting the systems availability. If critical services (systems) are attacked and brought down, it could cripple services and caste impact on economy and people’s lives. DoS attacks on mission critical systems such as aircraft and space ships can hurt the availability and can bring down the whole system threating the lives.

Terry (19) advocates the importance of taking regular backups to ensure the data availability in the event of any natural disasters. Redundancy might be a solution for information services that are highly critical. These redundant copies can be stored in different geographical areas to mitigate against natural disasters and intentional attacks by mala fide people. Designers, while designing embedded products should ensure that the systems can repel or filter such Dos attacks, have proper intrusion detections mechanisms, can raise an alarm, have damage control mechanisms in place in the event of any such attacks.

Extra security equipment or software such as firewalls and proxy servers, can guard against downtime and unreachable data due to malicious actions, such as denial-of-service (DoS) attacks and network intrusions. (19.)



	POTENTIAL IMPACT		
Security Objective	**LOW**	**MODERATE**	**HIGH**
Confidentiality Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. [44 U.S.C., SEC. 3542]	The unauthorized disclosure of information could be expected to have a **limited** adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized disclosure of information could be expected to have a **serious** adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized disclosure of information could be expected to have a **severe or catastrophic** adverse effect on organizational operations, organizational assets, or individuals.
Integrity Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. [44 U.S.C., SEC. 3542]	The unauthorized modification or destruction of information could be expected to have a **limited** adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized modification or destruction of information could be expected to have a **serious** adverse effect on organizational operations, organizational assets, or individuals.	The unauthorized modification or destruction of information could be expected to have a **severe or catastrophic** adverse effect on organizational operations, organizational assets, or individuals.
Availability Ensuring timely and reliable access to and use of information. [44 U.S.C., SEC. 3542]	The disruption of access to or use of information or an information system could be expected to have a **limited** adverse effect on organizational operations, organizational assets, or individuals.	The disruption of access to or use of information or an information system could be expected to have a **serious** adverse effect on organizational operations, organizational assets, or individuals.	The disruption of access to or use of information or an information system could be expected to have a **severe or catastrophic** adverse effect on organizational operations, organizational assets, or individuals.

FIGURE 4. Potential Impacts for CIA (20.)

6 DESIGN CHALLENGES

6.1 Resource limitations

Embedded software has traditionally evolved as "software on devices with limited resources". In this traditional view, the principal problem is resource limitations (small memory, small data word sizes, and relatively slow clocks). (21.) Given their size, portability and cost sensitiveness, they also often come with other resource limitations, such as computational power, absence of protective theft and shielding technologies when compared with a larger system. Among the examples, the Mission Critical system has much more stringent size and weight requirements than the others because of its use in a flight vehicle. (22.) For consumer devices, such as mobile phones and IoT devices, apart from size and computational power limitations, it is extremely important for these kinds of devices to be designed keeping the security of the user data and software attacks in mind.

While designing embedded systems, designers have to work with those above mentioned design challenges and at the same time they have to ensure users' integrity of data and secrets, and damage limitation in case a system or a device is compromised due to a multitude of reasons.

6.2 Reliability

Embedded software systems are generally held to a much higher reliability standard than any general purpose software. (21.) They are also known to work in extreme and tough conditions, therefore they are prone to more failures. Sometimes these the failures associated with embedded systems can be quite risky and threating. They can cause substantial damage in terms of monitory or personal loss. In mission-critical applications, such as an aircraft flight control, a severe personal injury or an equipment damage could result from a failure of the embedded computer. (22.) A failure to safeguard the user data in a mobile phone can have financial implications. Traditionally, such systems have employed multiply-redundant computers or distributed consensus protocols in order to ensure a continued operation after an equipment failure. (22.) Thus, it is important that, designers create alternate solutions for mission critical systems. But IoT devices designers should consider damage limiting functionalities for consumers in case of any software attack or physical tampering.

6.3 Cost constraints

Most consumer segment devices are designed keeping the cost factor in mind. Achieving a reliable system under controlled costs can be challenging for system designers. While designing cost consensus devices, designers keep the overall system costs controlled by keeping system complexity down, which should not affect the integrity or reliability of the overall system.

6.4 Functionality creep

Designers tend to improve the functionality of the products to compete in the market with their competitors. This is partly because consumers prefer to choose products offering the better functionality against the ones that offer more secure solutions. Therfore, in this ball game, it is often challenging for designers to balance the security features with the functionalities.

6.5 Knowledge Gap

A challenge in the area of designing secure solutions arises from the fact that attackers have been learning how to exploit the products for several decades, but the general solution designers have not kept up with the knowledge that attackers have gained. (3.) Therefore, it will be challenging for designers to get ahead of the attackers to come up with good solution.

7 MECHANICAL DESIGN ASPECTS

This chapter introduces mechanical design approaches that should be considered to bring out a secure product.

7.1 Product housing

A typical attack hard point in this case would be a product enclosure and an attack vector where attackers will make an effort to open the casings and access to internal circuitry and components. Product designers need to prevent an easy access to the product internals by concealing the access points through which the device can be opened.

Some of the safe enclosure techniques currently in use are

- designing a product with a single piece outer shell.
- using high melting point glues wherever needed.
- designing in such a way that the opening of a device needs a complete destruction.
- in addition to the above mentioned, leaving the surface points of device accessibility for service personnel who can open the device with specialized tools.

7.1.1 External Interfaces

External interfaces, such as proprietary connectors, Ethernet, RS232, USB, Firewall, JTAG, Wireless (802.11) and Bluetooth are a vital lifeline for the product to the outside world so that the device functions properly. In addition to carrying the usual device operations, these interfaces also aid in regular maintenance tasks, on-field diagnostic procedures and field programming.

External interfaces are always attractive targets for hackers as the root of the attack tree can originate from here. Attackers indulge in probing, sniffing, flooding and pushing malformed packets through these interfaces. Product designers should be able to defeat all possible attacks that originate from these external interfaces. Designers should remember the below mentioned issues while they are designing interfaces.

- A device must transmit only non-sensitive and public information in clear text format.
- All sensitive and confidential information that has to be exchanged over interfaces, must be encrypted.
- Adequate protection must be added inside the device to defeat spoofing, malformed packets. e.g, hardening of OS, strong firewall.
- Strict no obfuscation policy. Attackers are always ahead of designers and can uncover them easily.
- All the diagnostic ports, backdoor interfaces and JTAG connectors must be removed from production software images. Closing them by employing techniques, such as blowing resisters and fuses, is not the best approach as they can be reopened by attackers.

7.1.2 Anti-tamper mechanisms

Attackers try to gain physical access to a device by tampering to know confidential information and working internals device. Some of the tamper mechanisms worth considering by product designers are described below:

Tamper resistance

Tamper resistance relies on restricting physical access to devices. Devices must shall be housed and constructed with specialized tamper resistance materials and mechanisms, such as hardened steel enclosures, a usage of one way screws and epoxy coating materials, tight airflow channels, a usage of security bits, encapsulating the circuit and critical components to prevent intentional probing and tampering due to environment hazards.

Products with a well-protected housing would require a partial if not a complete destruction of device to open up. It is also essential for designers to lay emphasis on such a design that will leave a visible and clear evidence if a tamper attempt is made.

Tamper evidence

Tamper evidence mechanisms are designed to ensure that a visible evidence or trails are left of an attempted break in. These mechanisms do not protect the device or the confidential data in it but they raise awareness that there has been an attack on the system. Some of the widely used mechanisms are brittle packages, crazed aluminum and polished packages, bleeding paints and holographic tapes. All the above mentioned mechanisms will leave damages on the surface which would be hard to be unnoticed.

Tamper detection

Tamper detection can be done by installing relevant sensors which detect the tamper and trigger the relevant response mechanisms. Some of the widely used tamper detection mechanisms that are built into devices are quoted below (23.)

- Switches that detect a mechanical movement.
- Sensors that can detect and inform external environmental changes.
- Closed circuits and cables that are wrapped in and around the device for detecting an attempted break, tamper or modification.
- Monitoring of for the changes in voltages and clock frequencies from and to the chips.

Tamper response

Systems should trigger shutdown and disable themselves in response to tamper detection. They should be designed with a capability to log and generate forensic data for a further analysis of post tamper detection that can be accessed by service personnel. Designers should also consider techniques, such as erasing the sensitive data, in response to tamper detection. In products, which houses confidential data, keys should protect the data by resorting to erasing methods.

Confidential data is either stored in RAM or ROM. Erasing the random access memory is relatively easier and it is accomplished by dropping the voltage levels which will effectively clear the memory contents. In case of ROM, a ROM overwrite may be needed. There are even cases where a system employs an ultimate mechanism of physical destruction by shorting the circuits and rendering the device inoperable.

Further, it should be noted that these tamper response mechanisms do not trigger in case of unintentional actions, accidentally or by environmental factors.

7.2 PCB design and routing

Enough precautions should be taken while designing circuitry boards for not letting an easy access to components, such as FPGA, processors and memories. An easy access to these components and circuitry can help attackers in reverse engineering the product. Designers should look into advanced chip packaging technologies while they design PCBs, e.g. COB (Chip on Board) packaging, CIB (Chip In Board) packaging and Ball Grid Array packaging. The product should demark or erase all the chip markings either by black topping or using other methods, such as small sandering or etching. A failure to remove the chip markings will leave potential hints to attackers to learn about the chips behavior and their usage by referring their datasheets.

Sensitive components and circuitry lines should be concealed by using Epoxy material around them. Care should be taken during the PCB production not to leave test points visibly open and every attempt should be made to obfuscate the trace paths and critical lines into inner layers of PCB. Even electromagnetic emissions from the product can act as potential attack points which attackers can monitor to determine secret information. Thus, every step needs to be taken to minimize the emissions by installing an appropriate shielding and taking care of unprotected I/O busses from ESDs. Well-designed power lines and grounding can reduce noise levels and emissions. All unused GPIOs should be either disabled or fixed to a predetermined state.

7.3 Memory and bus protection

Designers should be aware that address and control bus lines are prone to probing and traffic is not secure. Designers need to design their systems to perform secure operations either inside SoC or components that will not use the unsecure bus in the system. Memory devices, such as RAM and ROM, are known to be unsecure. Even though systems are designed with a proper tamper detection and response mechanisms, such as a complete wiping out or erasing of data, there is a high chance that traces of data can still be left out. Although the product supports security fuses, boot-block protection mechanisms, these can be bypassed by die-attacks and chip decapping as attackers can reproduce and recreate cryptographic keys or remove security bits. Designers should also take steps to limit the time secure data can be stored in memory. The secure data should be erased if it is no longer needed.

8 HARDWARE ASSISTED SECURITY

Devices can be made more secure if security use cases utilize hardware solutions or extensions in conjunction with software rather than just as a software only solution. In the below sections the focus is on the role of hardware based solutions touching associated software functionalities that exploit these specialized hardware wherever needed. Security solutions that utilize an underlying platform and hardware assistance are more effective than purely software based ones.

8.1 Smart Card

A smart card is an embedded integrated circuit card with a CPU, memory, and at least one peripheral interface to communicate with a host device. (24.)

Every component in the smart card is deemed secure, basic in execution with very less computational power. Smart cards also have a higher level of temper resistance because of their size, a complete physical isolation of the trusted area and narrow interfaces .The isolated trusted area plus a higher level of temper resistance enables that the trusted services leveraged by a smart card are very secure, but very limited in scope due to lower computational capabilities. (24.)

8.2 TPM

Trusted platform modules (TPM) are dedicated secure crypto-processors which are designed to secure hardware or software by integrating cryptographic keys into a device. TPM chips are passive and execute commands from the CPU. They by no means will decide or control the execution flow. Generally, there are additional software components operating from the CPU. They that will interact with TPM chips, which would take action upon validating the content from the TPM.

Global standards of specification for the TPM are controlled by a Trusted computing group (TCG) alliance formed by HP, Intel, AMD and Microsoft. The objective of the TCG is to develop, define and promote open standards for hardware enabled trust computing and standardize software interfaces across platforms. These specifications aim to provide a platform independent functionality that must be provided by any trusted platform by facilitating a common interface for a secure computing environment. Thus, protecting and securing the platform against hardware and software attacks. There are also criticisms against TPM specifications. They are said to cripple the user rights by moving rights management from software to computing platform hardware.

TPM chips come with an internal hardware logic for RSA encryption and decryption, a random number key generator, a tamper proof non-volatile secure storage and a hash functionality. Although TCG specifications do not mandate the above to be hard wired, chip vendors prefer them as it can be difficult to fulfil the TCG specifications by just implementing these via software.

Simplified Architecture of the TPM

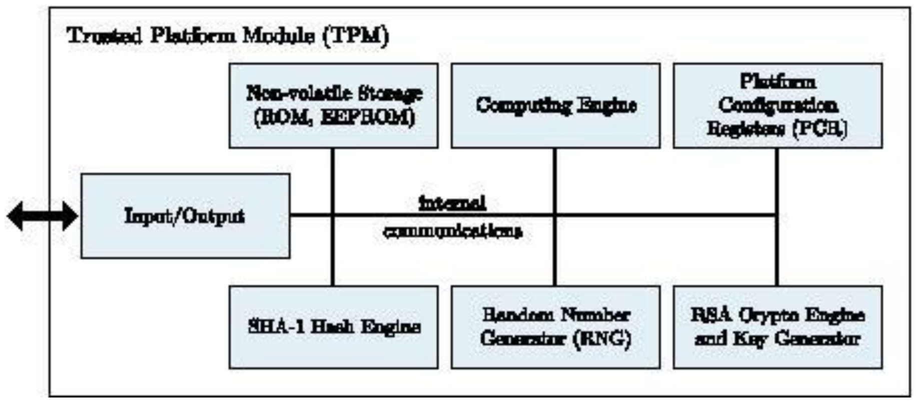

Trusted Platform Module (TPM)
Non-volatile Storage (ROM, EEPROM)
Computing Engine
Platform Configuration Registers (PCR)
Input/Output
internal communications
SHA-1 Hash Engine
Random Number Generator (RNG)
RSA Crypto Engine and Key Generator

FIGURE 5. High-level TPM Architecture (25.)

The first generation chips were physically separate which made the platform less secure. Later on they were either mounted in the mother board or embedded into the SoC. Mounting the TPM chips in the motherboard is prone to physical attacks and it is less secure, too.

Mandatory objectives of TPM chips are

- protecting the encryption and public keys from external stealing or getting misused by untrusted components in the system.
- preventing malicious code access to secret keys inside the TPM.

TPM chips achieve the above goals by implementing the functionality listed below

- Authorization: Public key authentication functions that provide on-chip key generation using random number generation hardware, verification, encryption and decryption functionality. Keys inside the TPM can be made to never leave or be visible outside the chip to avoid any kinds of phishing attacks and to prevent the key from being copied and used without the TPM. They can also be configured to be used by providing right authorization values when only they are unsealed. This in practice means that the keys are accessed only when the platform is in a known state.
- Integrity measurement functionality provides the capability to protect the private keys to untrusted code. In a trusted boot scenario, the platform state (hash of configuration) is stored in PCM registers. The private keys are sealed under the specific PCM registers and they get unsealed only if the TPM is provided with the same values by which the integrity of the system is verified. When an attempt is made to boot the system with an unauthenticated image or with a image that has malware or suspicious code built in will result in different PCM values. This kind of boot will fail the integrity check and will render the keys inaccessible.
- Attestation functionality: A mechanism where the TPM can certify the platform as trustworthy and not breached. By using the attestation, trusted clients can prove to the third party that their software is guanine

and not compromised. Attestation functions keep a list of software measurements committed to PCRs and can then carry out a signature using private keys known only by the TPM. During the attestation operation, the evaluating entity will request a TPM quote, which is essentially a signed composite hash of selected PCM data and its signature. The generated TPM quote is compared with the values that are generated during provisioning.

8.2.1 Root of trust for storage management

The root of trust for storage is a trusted component that lies inside the TPM that provides integrity and confidentiality checking for secure storage information in the chip. This is achieved using the RSA encryption and integrity checking by validating the platform state to a known value.

8.2.2 Root of trust for reporting

The root of trust for reporting is a trusted component that resides inside the TPM and responsible for reporting the platform state and RSA signed data to external parties who ever requests them.

8.2.3 Root of trust for measurement

The RTM (root of trust for measurement) is a piece of code external to the TPM which is responsible for triggering the measurement of platform state. RTM sits inside the core root of trust (CRTM) which is immutable and acts as a root of trust for the measuring environment. This is a small bit of immutable an component that gets executed upon a device reset and which is never changed in its lifetime.

8.3 Secure Element

Secure elements are temper-resistant hardware elements which hold device secrets and a limited isolated execution environment. They can also be added to any device with a micro SD or a Universal Integrated Circuit Card (UICC). (24.) They house cryptographic keys which are not exposed to outside world but would execute crypto operations inside the isolated environment created inside the chip. Evicting the secrets out of the chip for illicit use is either hard or impossible to do.

The main benefits offered by a secure element are

- Data Protection: Protection of data from unauthorized access (24.)
- Hardware assisted cryptographic operations: Dedicated inbuilt hardware logic for cryptographic functionalities, such as encryption, decryption, hashing etc. (24.)
- Isolated execution environment: Small defined tasks (applets) which perform cryptographic operations with keys and data stored inside the SE, can be run in an isolated execution environment without the data leaving the TEE. (24.)

This technology is widely used in the EMV chip on payment cards. Security offered by the SE is considered to be on a greater degree of evaluation level (EVL5) compared to a device that offers solutions over the TEE.

8.4 Hardware assisted boot process

As part of a hardware assisted boot process, the boot sofware in the device, which is supposed to be executed to bring up the device at each stage, is verified by a preceeding stage with the help of a hardware mechanism either the TPM or SE. This happens in two phases.

Figure 6 below depicts the interaction between hardware and software components to achieve the two-phase boot verification. As part of a boot process, image signatures of both the traditional OS and its boot loader which is often referred as the normal world are verified by the trusted execution environment (TEE). The trusted execution environment comprises of its TEE boot loader and TEE OS with the SE acting as root of trust.

Components that are deemed to be unaltered during the lifecycle and immutable are considered as a root of trust components. They are usually ROM programed in factory and they are bound to never change. On the contrary to the applications running in rich a OS, the secure tasks running in the TEE can be trusted. Also in this case, the SE is configured as a trusted peripheral and it is only accessible from secure tasks that execute in the TEE. In this way, applications running in traditional OSs make use of secure tasks to access the SE. As a consequence, if the TEE becomes unavailable, rich applications would be unable to perform secure use cases, such as device identification, device management, decryption and encryption.

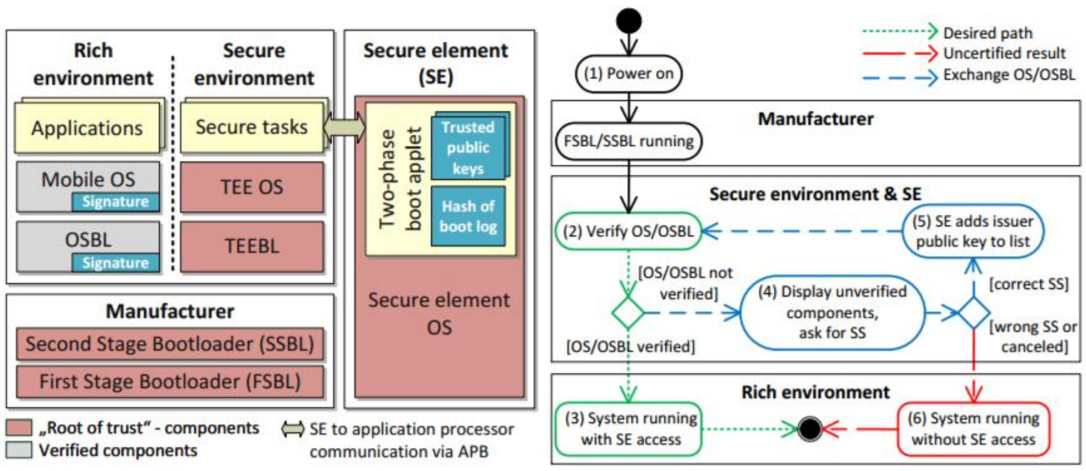


FIGURE 6. Architecture of secure execution environment (26.)

The above figure show that the root of trust components considered here are manufacturer's boot loaders and hash of a trusted public key (usually OEMs) in the Secure Element. If not in secure memory, trusted keys can also be stored in hardware one time fuse memory which is only accessible to the secure OS. These OEM keys are fused as part of series of steps in the assembly factory line when the device is manufactured. Under the "hardware assisted secure boot", images that form a chain of verification are pre-signed by the OEMs private key corresponding to the fused public key hash in a secure element or fuse memory. During the secure boot process, manufacturer's bootloder verifies the signed images, which are signed with a corresponding private key, whose hash of public key is stored in the Secure element. It is purely prerogative of the OEM to consider whether the TEE OS is under the root of trust. In case not, even the TEE OS and its bootloader need to be verified. If the verification at any stage fails, the device assumes to have been tampered and exits the boot phase.

Until now have been discussed the role of secure element or the TEE OS in a secure boot process have been discussed. Alternate solutions also exist, such as using TPMs instead of SEs as shown in figure 7.

8.5 Hardware assisted isolated/trusted execution environments

As described in figure 7, there are more than one way to realize isolation execution environments. Critical applications or part of use cases, which need an isolated execution environment to execute to protect confidentiality and integrity can be executed in these environments.

- External security co-processor can be used for executing sensitive operations and its results can be consumed by the primary processor.
- Using Embedded Secure Elements (ESE), which will be embedded inside the same System On Chip for secure operations.
- Advent of bus and memory isolation hardware extension logic blocks made it possible for a single processor (SoC) to be split into untrusted and trusted execution modes.

Secure peripherals can be accessed only while device is operating in secure mode. Untrusted entities cannot gain access to secure peripherals as they would run only on non-secure mode.

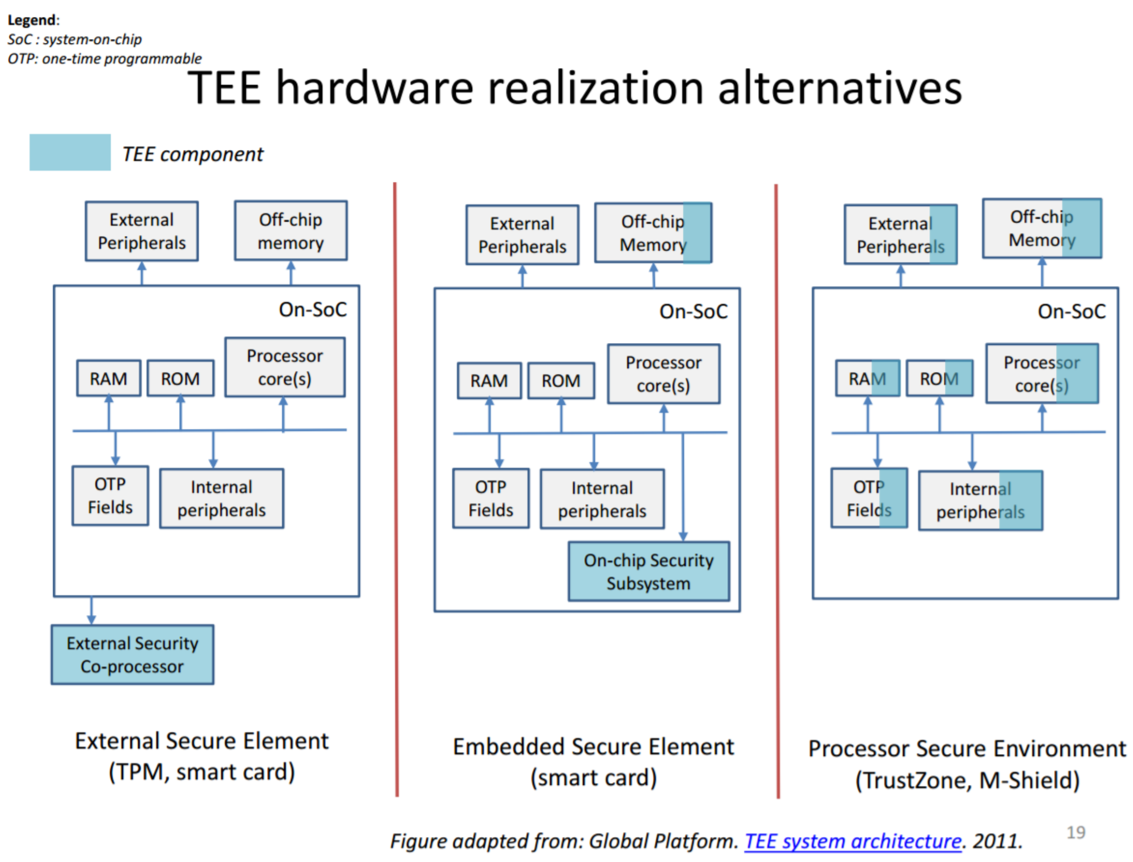


FIGURE 7. TEE Solution Realizations (27.)

Although a secure element offers a higher degree of security, the TEE is gaining momentum in the Industry as it addresses the most of the industry needs for secure applications by offering a higher level of security than any traditional OS, without the constraints associated with the secure element.

Recent trends show that TEE environments are widely used in mobile payments, enterprise applications, such as BYOD, and in content protection and in government electronic ID solutions.

ARM has come out with a trust zone architecture to facilitate the isolated execution environment (often referred as secure mode) against the rich operating systems execution environment (referred as non-secure or normal world mode) by maintaining a hardware separation between these two worlds.

The trust zone technology (see figure 8) will ensure that data and operations on the device remains secure, protecting consumer privacy and enabling a range of services, such as mobile banking, payment use-cases and playing multimedia entertainment which is meant for consumers and service providers.

TrustZone is incorporated within the same microprocessor core, enabling the protection of on- and off-chip memory. Since the security elements of the system are designed into the core hardware, security issues surrounding proprietary, non-portable solutions outside the core, are removed.

There is a minimal impact on the core area or performance while trustzone enables developers to build any additional security, for example cryptography, onto the secure hardware foundation.

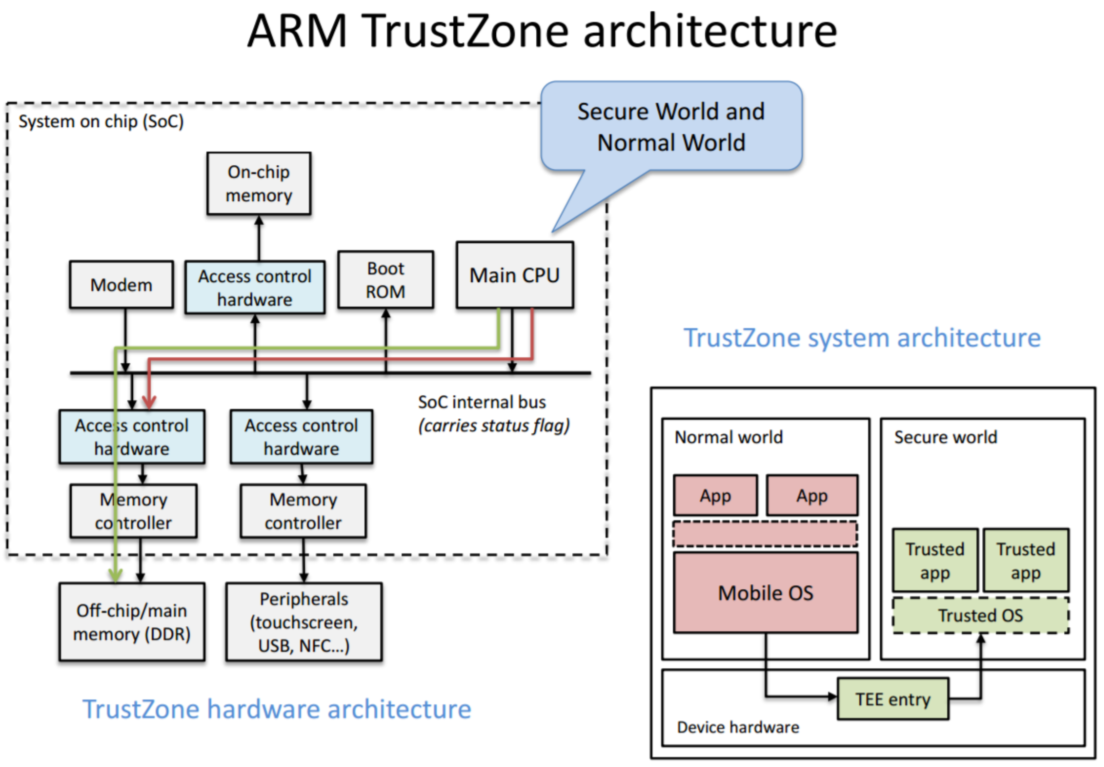


FIGURE 8.ARM Trustzone Architecture (27.)

Having introduced by ARM 10 years ago, Trustonic and Xilinx were the first ones to propose frameworks and solutions utilizing these hardware platforms.

This enables the research community (28.) as well as the industry to experiment and develop innovative solutions on it.

9 SECURITY ARCHITECTURES

In this chapter it will be discussed the ways software can be designed to exploit the specialized hardware modules (discussed in the earlier chapter) that lay foundation to secure solutions. This chapter also discusses some profound architectures to design software solutions.

9.1 Static root of trust measurement

As implied by name, the static root of trust begins with a piece of immutable code which has not been changed for the lifetime of platform. This piece of code forms the root of trust for the measurement (CRTM). This fundamental entity is referred as a trusted building block (TBB) from where the chair of trust originates.

In any embedded system the first executable is usually the ROM BIOS code, but to enable the static root of trust, there needs to be an additional immutable component, which would verify the bios before it starts to execute, and which is the Core root of trust for measurement (CRTM). The CRTM is either stored in the chip burned in the factory or found in the BIOS as block in the BIOS boot block. All that TCG mandates is to have the CRTM to be present in immutable and it is a true trusted building block.

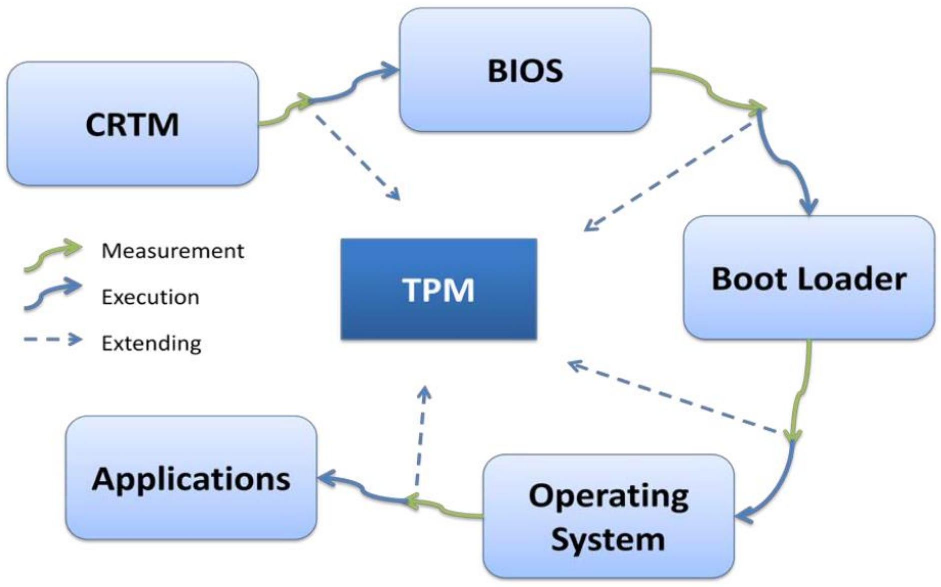


FIGURE 9.Chain of trust (29.)

The CRTM entity is responsible for measuring the integrity of the next component that follows the boot sequence and for extending the measurement values to predefined Platform Configuration Registers (PCRs). Each of the component that executes will need to verify the next one in the boot sequence and to extend the measurement values to PCRs before transferring the control to the next component

Even though there is not any compromised component in the execution chain, it cannot bypass the chain of trust verification. Also, the comprised components cannot technically extend its measurement values back to PCRs as shown in figure 10 below.

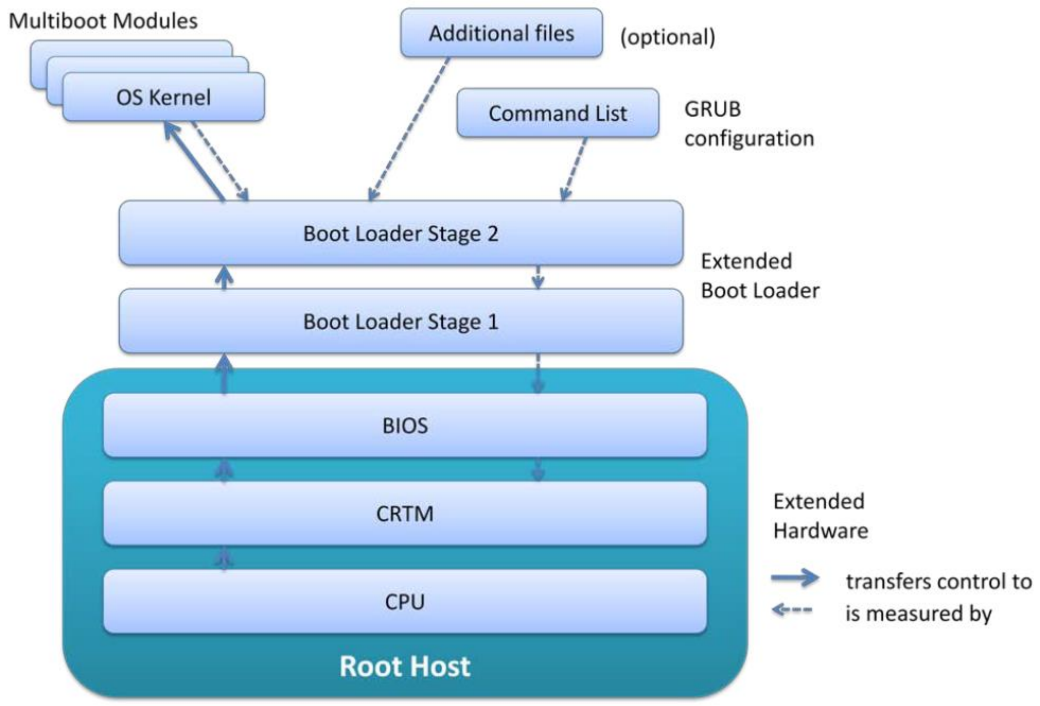


FIGURE 10.Chain of Trust, Measured boot in use (29.)

Of course, this static method to verify is good for load time, It also comes with few drawbacks, such as scalability and time to check and time to use (TOC-TOU) issues.

Scalability is the issue in case that all the executables, libraries and scripts will need to be part of a verified chain as there is no clear definition for a TCB in all the major operating systems used today. This is partly because all these components are subjected to patching, fixes and a varied order of execution, which can potentially change the PCR measurement values.

The time of measurement plays an important role due to the fact that even though the system can be booted in a known state, it is not guaranteed to be in the same state at the runtime. Potentially, during the execution of code, attackers can look for vulnerabilities and they can put the system to a unsafe state. It is essential to be aware that the Static Root of Trust for Measurement only gives a load-time guarantee not a run-time guarantee, i.e that that is it gives assurance of what program has been loaded, not necessarily what is running. (29.)

9.2 Dynamic root of trust for measurement

The above mentioned shortcoming related to the static root of trust has been addressed by the TCG group as part of 1.2 specifications .A dynamic root of trust measurement (DRTM) essentially deals with measuring the system state not just at the boot time but whenever there is a need for verifying the platform arises. The DTRM brings a substantial contribution from hardware side as well. Most semiconductor vendors have introduced special instructions (SKINIT, SENTER for AMD and Intel) which trigger the creation of controlled and attested execution environments in runtime.

The DRTM relies on small trusted piece of code which is loaded from a trusted source whose responsibility is to measure and execute a predefined piece of software. The DRTM comes with a short chain of trust when compared to the static root of trust measurement. This piece of software, which is launched and untainted and not modified right from the device boot time, puts the system into the secure state directly. This in practice means that the device gets into the secure state without a need to reboot or a further need to the static root of trust management.

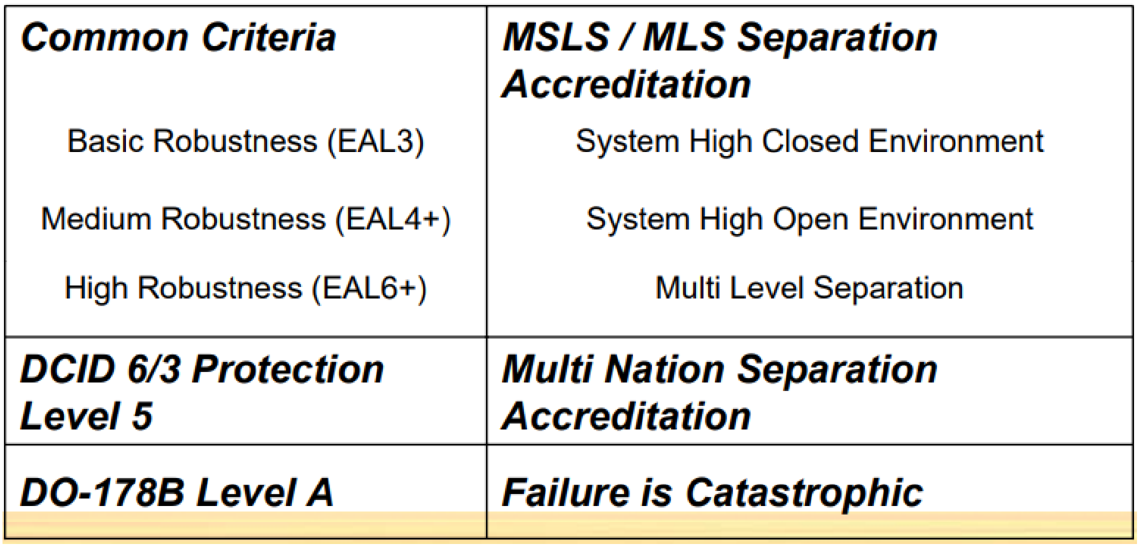
9.3 Multiple Independent layered security (MILS)

Legacy secure systems were designed around a secure kernel and a trusted computing base, with the idea that all the security decisions and the security enforcement mechanisms are an integral part of the TCB. (30.) This methodology can increase the complexity of the TCB as designers tend to add pump in more and more logic there thus creating issues for maintainability.

From the design paradigm, the MILS architecture stands better compared to the legacy one as it forces designers to follow a structured and modular approach.

The Multiple Independent Levels of Security (MILS) is a security architecture based on the concepts of separation and a controlled information flow implemented by the separation mechanisms, which supports both untrusted and trusted code where each level is responsible for its own security domain. Limiting the scope and complexity of the security mechanisms provides users with manageable and, more importantly, evaluable implementations. (31.)

This approach provides applications with mechanisms to control, manage and force their security policies in a manner that enforcement mechanisms are always invoked, mechanisms are non-by-passable, evaluable, and tamperproof. (30.) Under the MILS architecture, privileged mode processing is isolated from under privileged user data or applications. The MILS architecture is targeted at mission critical operations where a failure cannot be even thought of. The MILS architecture is intended to be used at the highest levels of security. Consequently, an EAL 7 will be targeted. (31.)



Common Criteria	***MSLS / MLS Separation Accreditation***
Basic Robustness (EAL3)	System High Closed Environment
Medium Robustness (EAL4+)	System High Open Environment
High Robustness (EAL6+)	Multi Level Separation
DCID 6/3 Protection Level 5	***Multi Nation Separation Accreditation***
DO-178B Level A	***Failure is Catastrophic***

FIGURE 11.Assurance certification goals (32.)

The advantages of the MILS architecture (shown in figure 11 above) are:

- Reducing the footprint of secure critical code dramatically, the lesser is better.

- Isolating the critical components and operations in the system.
- One processor can host multiple applications at different security levels. (31.)
- Certification costs are reduced since individual functions within non-security critical layers can be certified separately. Non-critical functions can be certified at a lower level. (31.)

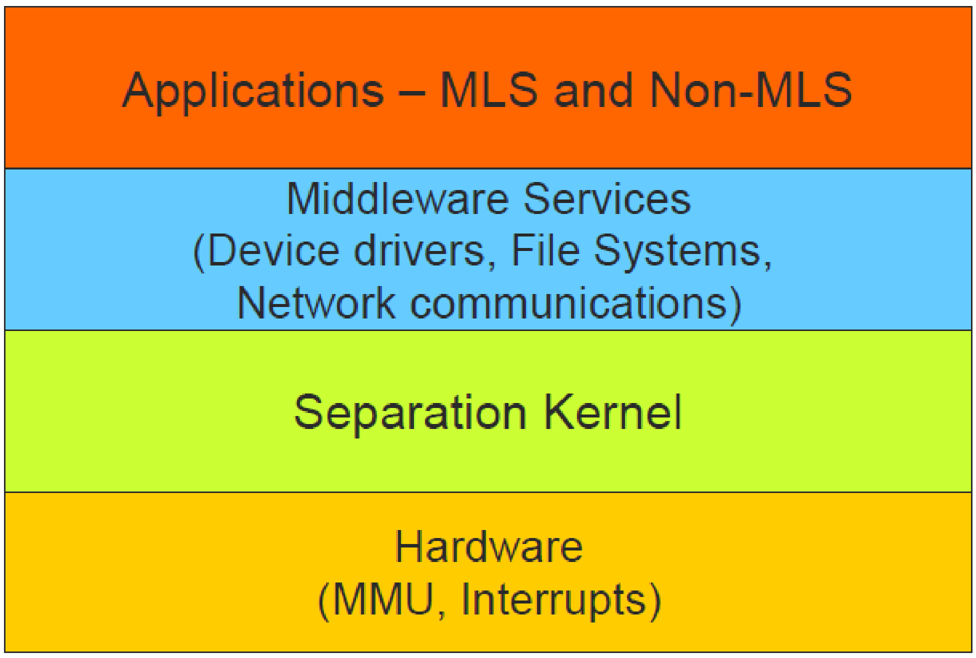

Applications – MLS and Non-MLS
Middleware Services (Device drivers, File Systems, Network communications)
Separation Kernel
Hardware (MMU, Interrupts)

FIGURE 12. Conceptual VIEW OF MILS layers (33.)

9.3.1 Separation kernel (SK)

Separation kernels are usually small code bases (4k) which execute close to hardware with the primary objective to isolate execution environments for all partitions. They act as a base layer interacting with hardware, effectively enforcing complete separation for data and controlling the flow control in a SoC by providing time and storage partition between secure and unsecure operations.

The main objectives of the secure kernel are data isolation, highly controlled information flow, data sanitization, damage limitation. (31.)

The kernel conceals and denies access to application data and memory contents between partitions. The state of executions in the current partition will not affect state of executions in other partitions and vice versa. In a practical scenario, it might not be desirable to achieve a total isolation between partitions as minimal communication across them might be needed. For such situations, the secure kernel defines some secure authorized channels for inter-partition communications through which the data sharing can happen. The kernel takes responsibility for relinquishing shared resources and data clean up (buffers, processor register, memory allocations) once they expire on longevity. In the event of security breach or an errant behaviour the secure kernel (SK) limits the damage limitation in the event of a security breach or an errant behaviour from any application by separating the address spaces between partitions. The secure kernel also caters to all partitions by enforcing bounds on shared resources and guaranteeing minimum processing times, interrupting servicing times and accessing to resources.

9.3.2 MILS Device Drivers

The MILS architecture requires the kernel to be small, having a minimum footprint in order to be fully evaluated. In any typical microkernel architecture, other needed OS services are typically included in the operating system running either as separate threads or processes but not in the kernel space, such as any other monolithic kernel (30.). But in the MILS architecture, these services are confined to play in address spaces of individual partitions or relegated to separate shared partitions if they service shared resources, such as communication across partitions, but are mediated by the separation kernel policy. When the device is categorized as private and un-sharable, access to the device is restricted to other partitions other than to partition where the device is configured to access. Their general access via MMUs needs to be curtailed down (memory mapped I/Os to communicate with devices) for private configured devices. (See figure 13, p 54)

9.3.3 Hardware support

The Secure kernel needs to support the underlying hardware for effectively executing partitioning, controlling information flow and isolating system resources. The SKPP (secure kernel protection profile) mandates certain functional requirements from hardware in which the secure kernel will operate on.

The support from the hardware MMU (Memory Management Unit) is needed for the separation kernel to isolate memory address spaces from various partitions. Processors should provide an ability for the separation kernel to grant the configuration access for memory layouts partitions in the system. The processor SDK needs to provide a privileged instruction set that can only be executed by the separation kernel along with the mechanism to transfer the execution control to the separation kernel in the event of execution of any high privileged operation or invalid instruction from any partitions. The underlying hardware needs to provide the atomicity support for critical operations, such as partition swapping and memory layout changes made by the separation kernel. The processor should also provide the separation kernel with options to restrict or configure access of i/o peripherals to specific partitions.

9.3.4 Middleware services

Middleware layers sit on top of the separation kernel providing services to applications and services in a particular partition. Some of the examples are inter-core communications, event and diagnostic logging, resource sharing and allocation. In addition to the above mentioned, the middleware layer is also responsible for the end to end security and communication policy by labeling, filtering and controlling the message/information flow. Middleware services can also be designed for authorized communication channels between specific partitions dictated by use case requirements. These services should address solutions from end to end across multiple partitions and cores rather than confine to a single process, partition or core.

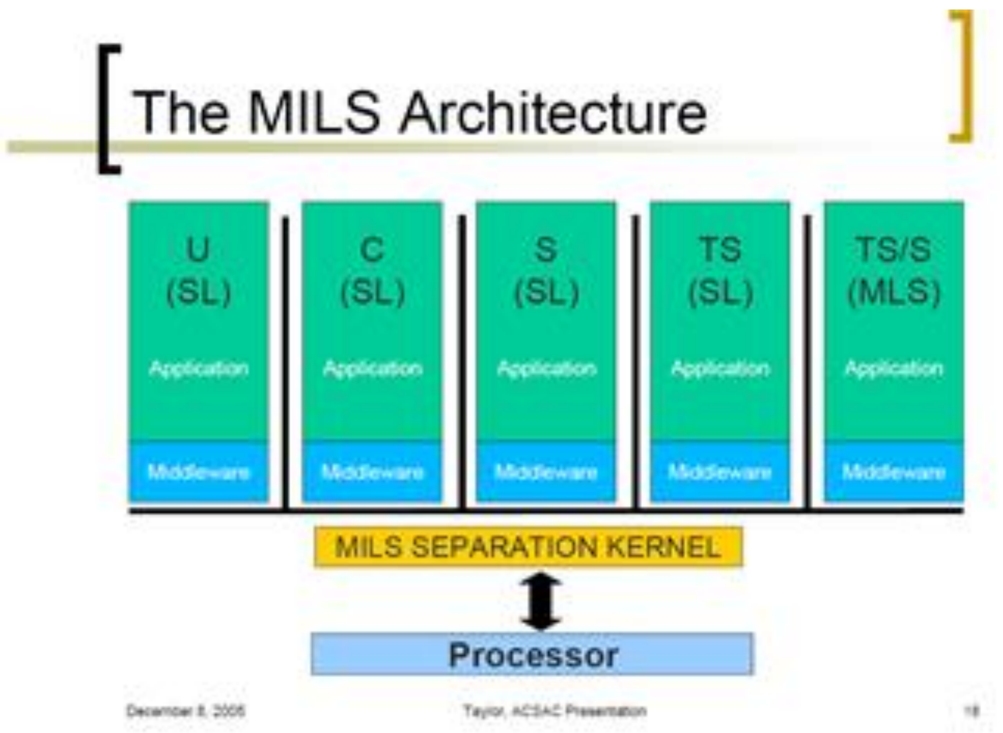


FIGURE 13. MILS Architecture *(33.)*

9.4 Trusted execution environment (TEE)

The Global Platform defines the TEE as a combination of a hardware platform, which provides isolation, and a software and operating system residing within the security domain defined by that hardware which is capable of running programs launched into that environment. This software running in the isolated environment is responsible for executing critical secure applications which access critical resources as part of their use cases. Systems that need to run secure use cases can deploy the TEE run in the secure environment to better protect their secrets, for example smart cards and DRM applications.

TEE supporting hardware platforms will come with a trusted operating system or some kind of task scheduler which runs in the hardware isolated environment as a software solution. This piece of software has access to the root of trust. It supports provisioning and has access to the Cryptographic API for accessing device secrets there by forming a core for the TEE.(See figure 14 , page 55.)

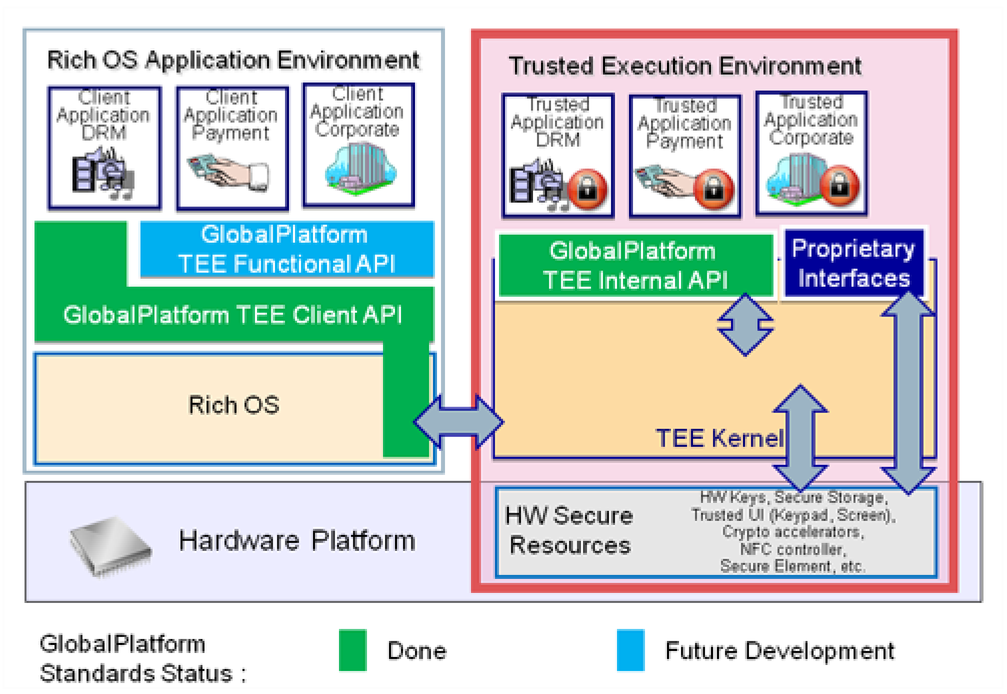

Rich OS Application Environment
Client Application DRM
Client Application Payment
Client Application Corporate
GlobalPlatform TEE Functional API
GlobalPlatform TEE Client API
Rich OS
Trusted Execution Environment
Trusted Application DRM
Trusted Application Payment
Trusted Application Corporate
GlobalPlatform TEE Internal API
Proprietary Interfaces
TEE Kernel
Hardware Platform
HW Secure Resources
HW Keys, Secure Storage, Trusted UI (Keypad, Screen), Crypto accelerators, NFC controller, Secure Element, etc.
GlobalPlatform Standards Status :
Done
Future Development

FIGURE 14. TEE Architecture *(34.)*

Secure hardware elements provide a trusted execution environment by creating a secure world (isolated environment) in a different core other than the one that runs normal world applications for software to operate in. More about the role played by hardware is discussed in the later part of the document that deals with the device security in general. The core that operates the TEE comes with immutable ROM boot code which does not need software checks that are needed to put the processor to a secure context.

9.5 Virtualization

Through virtualization, a secure isolated environment can be created by host operating systems by leveraging on OS concepts, such as isolating the processing contexts and memory for all the running environments. Here, the host environment is securely booted with a higher security privilege and will have enough security mechanisms in place to protect itself from other executing applications. This host operating system will typically operate smaller microkernels or hypervisor, whose trust bases are quite small, can be validated without much complexity. Using this mechanism, smaller virtualized environments are employed to run secure use cases, post its code base validation by larger parental OS environments and under its security kernel.

Employing virtualization for facilitating the TEE also has some vulnerabilities as explained below

- Virtualization technology for achieving a trusted execution environment as a security solution can be a risk as it is vulnerable to hardware attacks.
- In this approach, a larger conventional operating system or hypervisor enjoy higher privilege contexts leaving the virtualized environment OS to work under the user level. In this case, the virtualized OS that provides the TEE might not have full capabilities to protect its own kernel and applications running under its hood.

9.6 Platform security

Operating systems also play a vital role in ensuring the device security whether it can be a traditional rich OS (Android & Linux) or an OS residing in an isolated execution environment. Below various security measures, which traditional operating systems provide are discussed in detail.

9.6.1 Process Isolation

Process isolation as the term self-explain relates to an isolation of an individual process from the address space of other processes made possible by the underlying kernel to prevent one process either accidently or intentionally writing into each other’s space. The interaction between processes can still be made possible by IPC mechanisms, sockets and shared memory resources with the help of kernel. On the other side, most of the operating systems also provide a sandboxing mechanism to facilitate a highly controlled environment, which provides a controlled set of resources accessible for a set of programs. By using this mechanism, processes can be launched in their own private directory, invisible to other processes, and they work seamlessly with the existing application code to eliminate an entire class of security threats. Programs, such as access to certain IO operations, system inspection, and network access, can be restricted to these sandboxed applications. Few examples of sandboxing mechanisms are virtual machines, jails and containers in Linux.

9.6.2 Access Control permissions

To achieve platform protection, operating systems typically control access to system resources (files, directories, sockets, drivers) on who (subjects) should be able to access and what permission levels subjects have over the resources. Below different mechanisms to achieve this and their levels of protection will be discussed. Access control mechanisms can be broadly split into a discretionary access control and a mandatory access control.

Discretionary Access Control

The Discretionary access control is the most flexible access control mechanism. It allows users to control its own data. Under this Discretionary Access Control mechanism, access properties for objects are stored in Access Control Lists (ACL) which are associated with the object.

In the Unix DAC, the ACL lists form the backbone for the OS security model, Programs launched by a user run with all of the rights of that user, whether they need them or not. There is also a super user category, which is a more powerful entity that bypasses the Unix DAC policy for the purpose of managing the system. When any object is accessed in any form (read, write or execute), the operating system checks the rules contained in the ACL list for that object and makes a decision accordingly. This policy is implemented as permission bits attached to the file's inode, which may be set by the owner of the file. Permissions for accessing the file, such as read and write, may be set separately for the owner, a specific group, and other (i.e. everyone else) which is a relatively simple form of access control lists (ACLs). (35.) Running a program as the super user provides that program with all rights on the system. (35.) There are also other ALC mechanisms that deal with more fine grained schemes allowing separate permissions for different users and groups for the same resource.

Mandatory Access Control (MAC)

Unlike the DAC, the Security policy in MAC is administered centrally, and users cannot fully control the policies for their own resources. This helps in containing attacks which exploit bugs in user space software. The access control in this mechanism is wound around roles rather than users, which suits well for organizations where users and their permission to resources can be tagged against roles they have in the organization. The MAC brings in an additional layer of permissions which is associated with the user identity. Every object inside the system has its own label associated with the object. Now, based upon permissions granted to the subject (user) associated with user's role or group, they can only perform actions on tagged objects if there exists an explicit policy which grants access to the object. Conversely, a policy can be drafted based upon tags/labels to explicitly deny the access to certain users or groups.

The Role Based Access Control (RBACL) is a widely adopted mandatory access control mechanism around for quite some time. The Role based access control is more rigid in a way that access to resources cannot be granted above the role permits even though the user needs extra explicit permissions for the resource. In the Linux world, SELinux, SMACK and AppArmour are widely used implementations for the MAC. As part of the Android security model, Android uses SELinux to enforce the mandatory access control (MAC) over all processes, even processes running with root/superuser privileges (a.k.a. Linux capabilities). SELinux enhances the Android security by confining privileged processes and automating the security policy creation. (36.) (See figure 15 p 59)

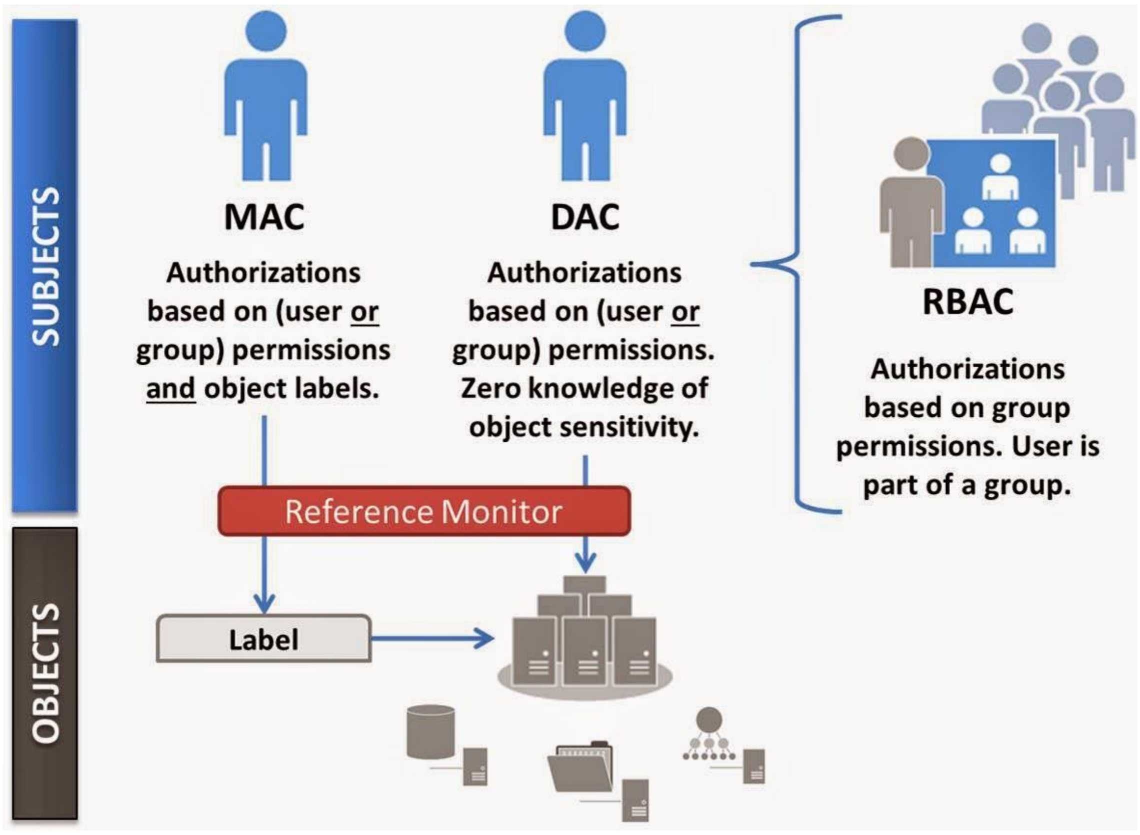

SUBJECTS
OBJECTS
MAC
Authorizations based on (user or group) permissions and object labels.
DAC
Authorizations based on (user or group) permissions. Zero knowledge of object sensitivity.
RBAC
Authorizations based on group permissions. User is part of a group.
Reference Monitor
Label

FIGURE 15. MAC vs DAV vs RBC (37.)

9.6.3 Operating systems support and extensions

Address Space Layout Randomization (ASLR)

Operating system kernels can be configured to use ASLR which enables randomizations of load address in memory areas for key sections of binaries and libraries from the user space, e.g base address, mapping of libraries, stack and heap addresses. These kinds of randomizations will help preventing a buffer overflow, a code injection and a return-to-libc attack which are difficult to exploit relying on luck to break into. External PaX projects such as Exec Shield and grsecurity projects have long maintained patches to the Linux kernel for these software-based hardening features. Android 4.0 has support for the ASLR completely discarding the support for the non-position independent execution support from 5.0.

The operating systems also supports hardware security features such as NX (Non-Execute), VT-d (Virtualization Technology), TPM, TXT (Trusted Execution), and SMAP (Supervisor Mode Access Prevention), along with the support for cryptographic operations.

Secure Computing Mode

A Secure computing mode is a mechanism which restricts access to system calls by processes. The idea is to reduce the attack surface of the kernel by preventing applications from entering system calls they do not need. The system call API is a wide gateway to the kernel, and as with all code, there have been and are likely to be bugs present somewhere. Given the privileged nature of the kernel, bugs in system calls are potential avenues of attack. If an application only needs to use a limited number of system calls, then restricting it to only being able to invoke those calls reduces the overall risk of a successful attack.

Memory protections

Memory segments which host the kernel are divided into logical areas and are marked for protective restrictions on access permissions. Code sections of the operating system kernel memory are marked as read only and execute sections where as a data section is marked as a non-execute segment. Data sections are marked as no-execute and further segmented into read-only data and read-write data sections. These features are usually enabled with kernel configuration options and specialized flags that control these features.

The operating system kernel can be further protected by restricting the kernel access to the user space memory directly. This can make a number of attacks more difficult because attackers have significantly less control over the kernel memory that is executable.

9.6.4 Integrity management

The operating systems kernel can come with an integrity management subsystem which will calculate the hash of all the non-volatile files and verify them against the cryptographic hashes stored in the TPM. Integrity measurement values can be verified by an external TEE OS Also tools, such as dm-verity, can be used to protect integrity at the block level. This module verifies the integrity of files block by block when the read from disk happens.

10 DEVICE SECURITY

This topic discusses the methods and technologies in tandem for enhancing the overall security of any embedded device. At places one can notice that the topics discussed earlier could have criss-crossed here, but here the same discussion is more looked from the prism of overall security rather than arbitrarily which was the case earlier.

10.1 Isolated Execution Environment

The Isolating execution also often referred as a trusted execution environment (TEE) gives the device and also the service and providers and OEMs the ability to run software modules in complete isolation from untrusted code. Executing code and data in complete isolation provides secrecy and integrity of that module's code and data at run-time. Conventional rich mobile systems, such as Android, IOS, Linux provide a process based isolation for protecting application address spaces from other applications and kernel resources. This leaves a possibility to circumvent the device security when the OS itself is compromised.

Providing an isolated execution means the security no more relies on the conventional rich execution environments (REE) Oss but on an alternative operating system that hosts a trusted execution environment (TEE) , made possible either by in SoC hardware extensions or special security coprocessors. SoCs that support isolated execution environments come with special hardware extension blocks which provide hardware isolation from the non-secure world. OEMs can configure the peripherals by restricting or extending their access either to the isolated secure environment or to the untrusted execution environment or both. The memory isolation is provided by MMU extensions which restrict some predefined memory pages accessible to the processor only in a secure context. But these hardware assisted methodologies are not of much help in case the software designed to use this is not designed well for secure use cases. Figure 16 explains isolated execution environments in general.

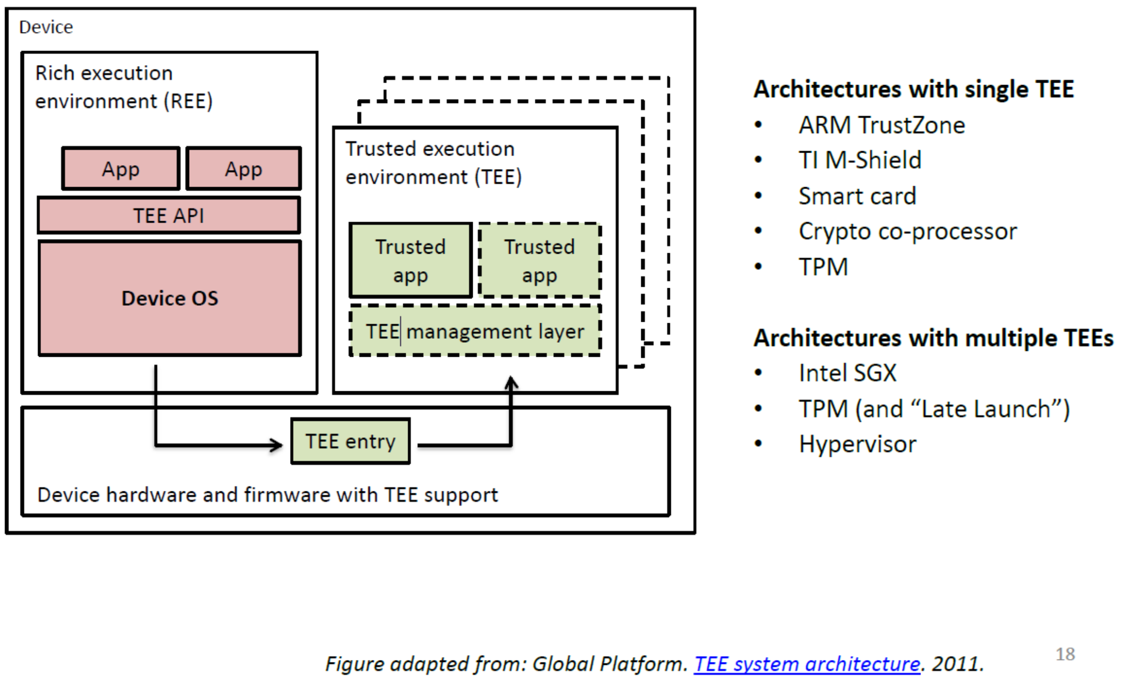

Device
Rich execution environment (REE)
App
App
TEE API
Device OS
Trusted execution environment (TEE)
Trusted app
Trusted app
TEE management layer
TEE entry
Device hardware and firmware with TEE support
Architectures with single TEE
• ARM TrustZone
• TI M-Shield
• Smart card
• Crypto co-processor
• TPM
Architectures with multiple TEEs
• Intel SGX
• TPM (and "Late Launch")
• Hypervisor
Figure adapted from: Global Platform. TEE system architecture. 2011.
18

FIGURE 16. ARM Trustzone Architecture (27.)

10.2 Protection of Confidential data

Devices should be able to protect confidential data (secrets, keys, licences, certificates) and should not reveal them to untrusted parties. Confidential data can be pertaining but not limited to a device (OEM), a user, an operator and a platform providers meta data specific or service providers.

The protection of confidential data is critical, The data cannot be accessed either by network attacks, physical tampering or access by unauthorized software entities in the device. Embedded devices can often be reprogrammed remotely as a part of fixing bugs and adding new features. In due course, devices should ensure that attackers do not attack and insert their own malicious code and hijack confidential data as it flows through the system.

10.3 Device Identification (via Remote Attestation)

Remote attestation allows a remote entity to verify that a particular message originated from a particular software entity. (.38) Expressing this in business terms, service provides widely use the remote attestation to make sure that the device they are communicating with is the real device that they are wishing to talk to and not any untrusted or unknown entity.

Trusting the device would mean attesting a certain chosen software module(s) or a complete stack of underlying software running along with it. This could in practice mean whole operating systems. Service providers who offer secure services, such as payment services, can know if the communicating device software is in rightful and known state or is operating with tampered or rooted software.

The attestation can be meaningful and easier to achieve when a computing base is relatively small, which is difficult to achieve with rich execution environments, such as Android, IOS and Linux. In practice the attestation is achieved via apps from service providers hosted in isolated execution environments which have attestation capabilities. These mechanisms are typically built using a private key which is only accessible by a small computing base and kept in a secure storage. (38.) A certificate issued by a trusted party, such as the device manufacturer, certifies that the corresponding public key belongs to the device. One or more platform configuration registers store the measurements of loaded code. The private key of the device can then be used to generate signed attestations about its state or the state of the rest of the system. (38.) (See figure 17 , p.65)

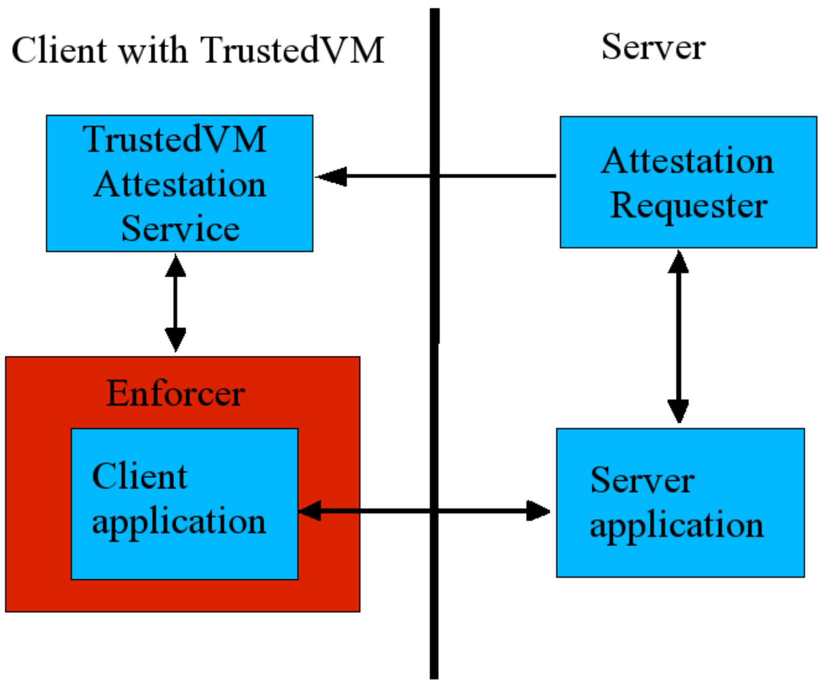


FIGURE 17. Top level architecture for dynamic checking in Trusted VM (39.)

Figure 18 below denotes a health application capable of performing device attestation (certifying the device in a rightful state) and thereby communicating with the service.

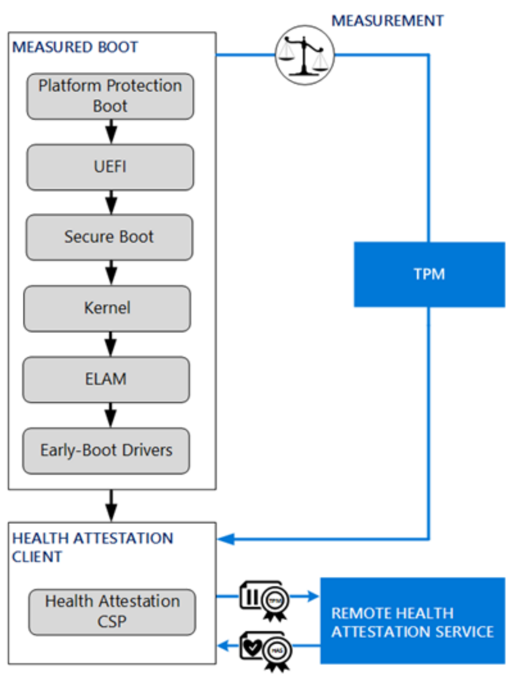


FIGURE 18. Figure showing attestation procedure for a health application (40.)

10.4 Secure provisioning

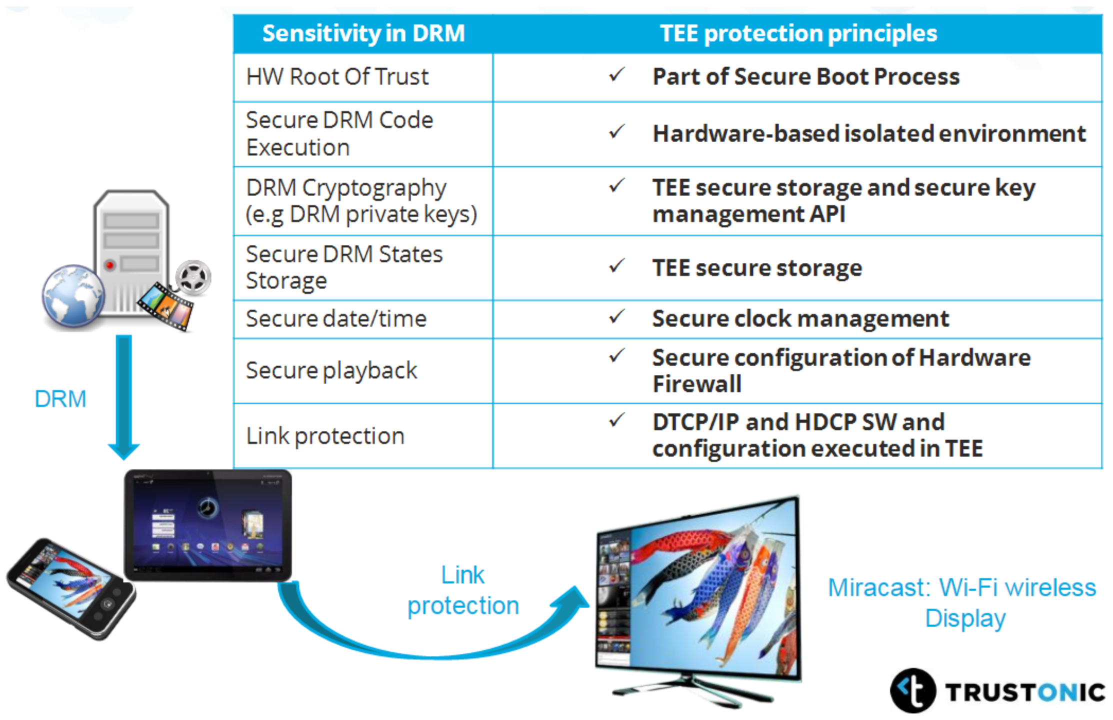
Secure provisioning is a mechanism of sending data to a specific software module, running on a specific device, while protecting the secrecy and integrity of that data. (38.) Secure provisioning comes in handy for migrating user's data between user's devices.

Remote attestation can be used for provisioning data securely by using keys belonging to a particular software entity on a specific device. The sender can then use that key to protect data to be sent to the target software module on the target device.

Some of today's mobile and IOT platforms implement mechanisms to authenticate external information from the hardware stakeholders (e.g., software updates), with the hash of the public portion of the signing key stored immutably on the device. Regarding current and previous mobile platforms such as Meego, the Linux distribution for mobile devices includes provisions for the isolated execution. Meego’s Mobile Simplified Security Framework (MSSF) implements a trusted execution environment, which is protected from the OS (41.) Other secure provisioning mechanisms are likely implemented and used by device manufacturers to implement features, such as digital rights management. So far, however, secure provisioning mechanisms are not available for direct use by arbitrary third-party developers on mobile platforms. (38.)

10.5 Trusted path

Devices should be able to setup a trusted path for all related secure use cases to ensure the data and content protection. This can be done by enabling a set of security features to enable the software to run on top of hardware isolation provided by SoC. A Trusted path protects authenticity, and optionally secrecy and availability, of communication between a software module and a I/O peripheral (e.g., a keyboard or a touchscreen). (38.) Classic examples of these requirements are playing an DRM protected media content rendering, enabling a fingerprint sensor for device locking and unlocking and providing a trusted path for password authentication for payment applications. Establishing a trusted path helps devices to protect assets against unauthorized tampering, modification of software by unauthorized software. Figure 19 below explains the protection mechanisms that are applied when rendering a DRM protected media content.



Sensitivity in DRM	TEE protection principles
HW Root Of Trust	✓ **Part of Secure Boot Process**
Secure DRM Code Execution	✓ **Hardware-based isolated environment**
DRM Cryptography (e.g DRM private keys)	✓ **TEE secure storage and secure key management API**
Secure DRM States Storage	✓ **TEE secure storage**
Secure date/time	✓ **Secure clock management**
Secure playback	✓ **Secure configuration of Hardware Firewall**
Link protection	✓ **DTCP/IP and HDCP SW and configuration executed in TEE**

FIGURE 19. TEE Protection principles to media assets *(42.)*

Building secure trusted paths is a challenging problem and in principle, many mobile platforms support a form of trusted path, but the TCB is relatively large and untrustworthy. (38.) This is because the computing bases of rich execution environments are large and it is often complex to secure trusted paths. But by removing the rich OS from the TCB of such trusted paths by preventing the OS from communicating directly with the device and running the device driver in an isolated environment, a trusted path can be established. But this requires the hardware based support for a low-level access-control policy to access to peripherals. (38.)

Figure 20 below from Jan-Erik and Kari (27.) shows an excellent relation between the functionalities and level of hardware support to achieve security.

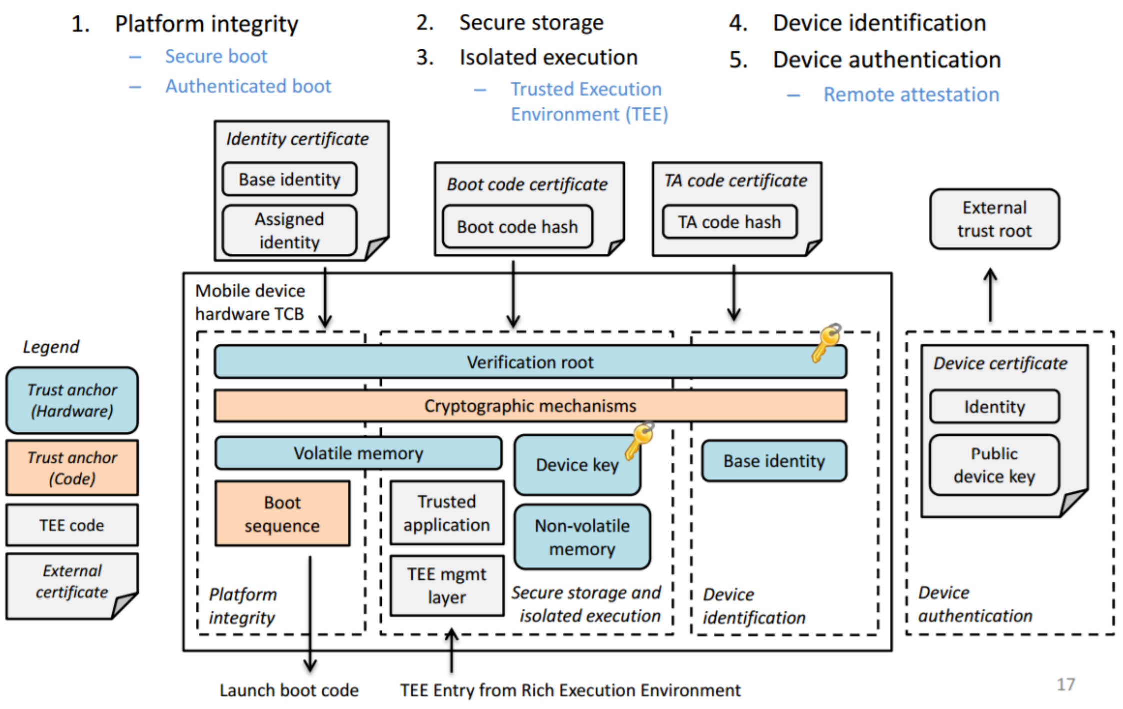


FIGURE 20. Concise picture showing different hardware security mechanisms (27.)

11 CONCLUSION

Device security should be treated with paramount importance and should be ingrained in the product design right from the conceptual phase. It should never be treated as “nice to have” only if time, features and functionality permit. Security needs to be fortified by building brick by brick, multiple layers of defences and protection mechanisms acting in tandem. No one solution could safeguard the whole chain (end to end). It is also expensive, tedious and often infeasible to inject security in the midway of product design. Therefore, care should be taken right from the initial phases of design. Thus, during the conception phase of a product/project, security requirements need to be well understood and agreed in tandem with functional and non-functional requirements.

I feel the best way to accomplish this is to base the project execution on a security driven development (SDD) where each of the requirements are tied down with a security related measure, binding or precondition.

As part of pre-design, architects need to do an end to end threat assessment, figure out asset classes, come up with attack trees and entry points and attack categories (network, physical abuse, software, cryptographic) for the product. Each of them should be thoroughly discussed, considering solutions to minimize the attack surfaces while designing.

In future we can expect an increased trend that software solutions will base their trust anchor on hardware which even semiconductor vendors/IP Licensees, such as ARM, INTEL, QUALCOMM and TI are going to support with hardware extensions.

In future, I also see cloud security gaining prominence, in addition to client (device) security, as cloud offers a bigger security infrastructure in terms of data protection and resource availability. In days to come, I can see device security more restricted to establish a trust and secure connection to cloud space, where all the computational and storage logic is hosted.

From software side, I see an increasing trend towards a wider adoption of software containers, Virtual environments, especially in cloud spaces, will achieve a user environment isolation and provide a better security infrastructure for applications. With an increased penetration of cloud technology, client devices could play an increased role in establishing a secure link, where real computational logic and data gets resided, between cloud and the device.

I always advocate to solution designers to view product security from attacker’s point of view rather than from that of designers. Essentially it is all about the perception “How things should not work”. Also, it is worth noting that, no technology or solution is worth its salt if designers do not religiously follow the principles listed below while designing their product.

- The weakest links should always be secured. For example, often in cryptographic operations, it is found that the weakest links have been the process of storing secret keys or restricting their access permissions rather than the cryptographic algorithm itself. It is quite usual to find keys that are not securely stored or not cleared from a memory once used.
- It is not practical to design a total fool proof system. If a failure is imminent and unavoidable, fail securely and limit the damage.
- Protection and damage control gates should be built in. Always design for a layer of protection services to kick, in the event of breach to one of your defences.
- Go by the concept of granting least privileges to resources.(E.g. avoid system wide blanket (sudo) permissions for the process)
- Develop an attitude of “reluctance” to trust information from unknown sources/parties.
- Avoid the security by obscurity, If OpenSource components are being used, be prompt to patch for security updates.
- From coding standards, use secure coding guidelines accompanied by reviews, reduce the attack surfaces by reducing code bases.

Finally, Security has to be built across all layers of software, phases of development (design, development, testing, deployment, maintenance), multiple domains (software, hardware, firmware). It should be every designer’s responsibility and only confined to few individuals, teams or ranks.

12 REFERENCES

1. Barnum, S. and Sethi,A. 2007. Attack Patterns as a Knowledge Resource for Building Secure Software. Date of retrieval 20.02.2016 https://capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf.
2. Vega, R. 2015. 3 Key Rules for Cybersecurity. Date of retrieval 02.09.2017 www.cnbc.com/2015/03/02/3-key-principles-for-cybersecurity-commentary.html.
3. Schneier, B. 1999. Attack Trees: Modeling Security Threats. Date of retrieval 14.10.2017 https://www.schneier.com/academic/archives/1999/12/attack_trees.html.
4. Anderson, R. 2001(First edition). Security Engineering: A Guide to Building Dependable Distributed Systems.
5. Biryukov, A. 2005. Chosen Ciphertext Attack. Encyclopedia of Cryptography and Security, pp. 7–7
6. Bleichenbacher, D. 1998. Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. Date of retrieval 15.10.2017 http://archiv.infsec.ethz.ch/education/fs08/secsem/bleichenbacher98.pdf.
7. WS-Attacks. 2017, Adaptive Chosen-Ciphertext Attacks. Date of retrieval 20.04.2017 www.ws-attacks.org/Adaptive_Chosen-Ciphertext_Attacks.
8. Bleichenbacher, D. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. Advances in cryptology – CRYPTO'98. Lecture notes in computer science, vol 1462. Springer, Berlin, pp 1–12
9. Liska, A. 2003. The Practice of Network Security: Deployment Strategies for Production Environments. Prentice Hall PTR.

10. Grand, J. 2004. Practical Embedded Security. Date of retrieval 15.06.2016 https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-grand/grand_embedded_security_US04.pdf.
11. Rabou, S. 2017. Why do you need a hardware solution to secure your embedded system?. Date of retrieval 20.07.2017 https://www.chipestimate.com/tech-talks/2014/01/14/Barco-Silex-Why-do-you-need-a-hardware-solution-to-secure-your-embedded-system.
12. Rouse, M. 2015. Common Vulnerabilities and Exposures (CVE). Date of retrieval 15.09.2016 searchfinancialsecurity.techtarget.com/definition/Common-Vulnerabilities-and-Exposures.
13. Secure Software, Inc.2005. The CLASP Application Security Process. Date of retrieval 15.09.2017 https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf
14. MITRE. 2017. Common Weakness Enumeration. Date of retrieval 15.09.2017 https://cwe.mitre.org/about/sources.html.
15. MITRE. Common Attack Pattern Enumeration and Classification — CAPEC™, Date of retrieval 15.09.2017 https://makingsecuritymeasurable.mitre.org/docs/capec-intro-handout.pdf.
16. Scott, E. 2011. Knowledge Consulting Group, Threat-driven Software Development Planning: Using CAPEC & CWE to Improve SDLC. Date of retrieval 15.09.2017 http://slideplayer.com/slide/4316286/.
17. Breithaupt, J & Merkow, M. 2014. Information Security: Principles and Practices, 2nd Edition
18. Haughn, M & Gibilisco, S. 2014. Confidentiality, integrity, and availability (CIA triad). Date of retrieval 10.09.2017

http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA.

19. Chia, T. 2012. Confidentiality, Integrity, Availability: The three components of the CIA Triad. Date of retrieval 10.09.2017 http://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/.
20. Clement, R. 2011. Information Security and the SDLC. Date of retrieval 1.09.2017 https://www.slideshare.net/bdpacharlotte/information-security-and-the-sdlc.
21. Lee, E. 2005. What are the Key Challenges in Embedded Software?. Date of retrieval 1.09.2017 https://pdfs.semanticscholar.org/e80e/f0404b027871fb09226a767b477a228f8452.pdf.
22. Koopman, P. Embedded System Design Issues. Date of retrieval 1.09.2017. https://users.ece.cmu.edu/~koopman/iccd96/iccd96.pdf.
23. Aarts, M. Hardware Attacks Tamper Resistance, Tamper Response and Tamper Evidence. Date of retrieval 23.03.2016 maurice.aarts.info/papers/tamper_resistance_evidence.pdf.
24. Gonzalez, J. 2015. Operating System Support for Run-Time Security with a Trusted Execution Environment. Date of retrieval 03.03.2016 https://www.itu.dk/~/media/d602e06412af44b69e3c86924fca9820.ashx.
25. FIDIS. Architecture of Trusted Computing by the TCG. Date of retrieval 03.08.2017. http://www.fidis.net/resources/fidis-deliverables/privacy-and-legal-social-content/d143-study-on-the-suitability-of-trusted-computing-to-support-privacy-in-business-processes/doc/4/single/
26. González, J., Hölzl, M., Riedl, P., Bonnet, P., & Mayrhofer, R. 2014. A Practical Hardware-Assisted Approach to Customize Trusted Boot for Mobile Devices. Date of retrieval10.09.2017 https://javigongon.files.wordpress.com/2011/12/secure-boot-vision.pdf.

27. Ekberg, J & Kostiainen, K. 2014. Mobile Platform Security and Trusted Execution Environments. Date of retrieval 10.09.2016 http://asokan.org/asokan/Padova2014/tutorial-mobileplatsec.pdf.
28. González, J & Bonnet, P. 2013. Towards an open framework leveraging a trusted execution environment. In Cyberspace Safety and Security. Springer, pp 458-467.
29. Choinyambuu, S. 2011. A Root of Trust for Measurement. Date of retrieval 19.03.2017 http://security.hsr.ch/mse/projects/2011_Root_of_Trust_for_Measurement.pdf.
30. Alves-Foss, J & Oman, P & Taylor, C & Harrison, S. (2005). The MILS architecture for high-assurance embedded systems. Date of retrieval 20/09/2017 https://www.researchgate.net/profile/Paul_Oman/publication/220309643_The_MILS_architecture_for_high-assurance_embedded_systems/links/0912f50fee695f0273000000/The-MILS-architecture-for-high-assurance-embedded-systems.pdf.
31. Alves-Foss, J & Taylor, C & Oman, P. (2004). A Multi-Layered Approach to Security in High Assurance Systems. Proceedings of the Hawaii International Conference on System Sciences. 37.
32. UChenick, G. Multiple Independent Levels of Safety & Security (MILS): High Assurance Architecture. Date of retrieval 20.10.2017 www.omg.org/news/meetings/workshops/RT_2005/02-2_Uchenick.
33. Taylor, C and Alves-Foss, J. 2005. Multiple Independent Levels of Security. Date of retrieval 20.12.2016 www.acsac.org/2005/case/thu-130-taylor.pdf.
34. Global Platform. 2015. The Trusted Execution Environment: Delivering Enhancing Security at a lower cost to the mobile market. Date of retrieval: 20/09/2017 www.globalplatform.org/documents/whitepapers/GlobalPlatform_TEE_Whitepaper_2015.pdf

35. Linux.com. 2013. Overview of Linux Kernel Security Features. Date of retrieval 25/03/2017
https://www.linux.com/learn/overview-linux-kernel-security-features.
36. Android Open Source Project. 2017. Security-Enhanced Linux in Android. Date of retrieval 25/03/2017
https://source.android.com/security/selinux/.
37. Davis, C. 2014. Cloud Audit Controls: MAC vs DAC vs RBAC. Date of retrieval 08.08.2017
www.cloudauditcontrols.com/2014/09/mac-vs-dac-vs-rbac.html.
38. Vasudevan, A., Owusu, E., Zhou, Z., Newsome, J., & Mccune, J. M. (2012). Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me?, Trust and Trustworthy Computing. Trust 2012, vol 7344, 159-178.
39. Haldar, V., Chandra, D.,Franz, M. 2004. Semantic Remote Attestation -- A Virtual Machine Directed Approach to Trusted Computing. Date of retrieval 08.08.2017
www.usenix.org/legacy/event/vm04/tech/haldar/haldar_html.
40. Jumelet, A. & Lich, B. 2017. Control the Health of Windows 10-Based Devices (Windows 10). Date of retrieval 08.08.2017.
https://docs.microsoft.com/en-us/windows/device-security/protect-high-value-assets-by-controlling-the-health-of-windows-10-based-devices.
41. Koistiainen, K., Reshetova, E., Ekberg, J.-E,. and Asokan, N. 2011. Old, new, borrowed, blue—a perspective on the evolution of mobile platform security architectures. Proceedings of the first ACM conference on Data and application security and privacy (CODASPY), 2011. PP 13-24.
42. Lu, M. TrustZone ®, TEE and Trusted Video Path Implementation Considerations. Trustonic Ltd, Date of retrieval 20.09.2017
www.arm.com/files/event/Developer_Track_6_TrustZone_TEEs_and_Trusted_Video_Path_implementation_considerations.pdf.
43. Knudsen, L. 2011. Kryszczuk, K. (2011). Block Ciphers. Encyclopedia of Cryptography and Security, pp152-157.

44. Heward, G. 2014. Cryptanalysis and Attacks. Date of retrieval 11.11.2017. https://www.experts-exchange.com/articles/12460/Cryptanalysis-and-Attacks.html
45. OSWP. 2016. CLASP Concepts. Date of retrieval 11/11/2017 https://www.owasp.org/index.php/CLASP_Concepts
46. Cisco Press. 2006. Network Security 1 and 2 Companion Guide
47. Seacord, R. & Householder, A. 2005. A Structured Approach to Classifying Security Vulnerabilities. Date of retrieval 11/11/2017. https://resources.sei.cmu.edu/asset_files/TechnicalNote/2005_004_001_14474.pdf