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
Spring 2017
Master Degree in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme, option

Author(s): Nayani Srinivas Vijayanand
Title of the bachelor's thesis: Designing secure solutions for embedded systems
Supervisor(s): Teemu Korpela
Term and year of completion: Spring 2017 Number of pages: 69

Goal of the thesis is to explore and throw some insight on the domain of embedded security touching; general security concepts, drivers for embedded security, popular attack terminology , attack classes and vulnerabilities classification, security architecture aspects that needs to be taken into consideration and a realistic evaluation of security solutions currently in market.

In this thesis, we also touch on mechanical design issues, hardware assisted security where we discuss the role and extent of support from hardware in dealing with security.

Finally in conclusion, we discuss about the some of my views and best practices for keeping our solutions secure and safe before touching on some of the future roadmaps and technologies.

Keywords:
Embedded security, Attacks and vulnerabilities, TEE (Trust execution environment), CWE, CVE,CAPEC, PLOVER, Attack vectors, TPM, Attack vectors, MILS (Multiple Independent Levels of Security), Separation Kernel, CIA Model, Secure Element, ARM TrustZone, Root of trust

CONTENTS

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 found its way in most of the fields like telecom, aerospace, healthcare, smart, wearable devices and defence. It's presence is set to increase with the upcoming technologies like Cloud ,robotics, machine learning & AI still yet to hit the peak.

Given its importance, security should be treated as integral part of design which should be considered right from product conception stage. It should be built into the system across multiple levels often referred as layered approach. It is not feasible or often viable to design a totally fool-proof system. So designers should rather focus on systems that can be difficult to compromise and reduce the risk to an acceptable level. Amit and Barnum states in their paper [36] , The primary challenge in building secure software is that it is much easier to find vulnerabilities in soft-ware than it is to make software secure. We can apply this to designing the product end to end , not limited to just software.

2 DRIVERS FOR DEVICE SECURITY

In the current IoT(Internet of Things) age, we pretty much can assume almost every entity acts as a connected node in the network. List includes but not limited to, mobile phones, surveillance IP cameras, consumer appliances, parking spaces , power plants, cars and mass transportation vehicles [34]. By the year 2020, analysts estimate, there could be 50 billion Internet connected devices which is only bound to grow [34].

This also means that we need to protect these increasing number of points in the network: not only establishing a secure, reliable communication but also running the device without compromising the integrity and authenticity which can be stated as primary drivers for need for security across all devices.

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

3 SECURITY ATTACKS

Formal attack terminology widely used by industry are attack trees, attack windows, patterns that are used to describe any attacks.

Attacks can be classified based on certain parameters like attackers objectives, impacts, Origin ,phase of introduction, technology employed and exploitability etc. Purpose of any attacks is to achieve desired goal that can range from data stealing, inflicting serious damage and crippling the system, hobby attacks, academic or institutional attacks as part of research to expose vulnerabilities. In the attack terminology, purpose of the attack is viewed as 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 [35]. 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 [36]. Attack patterns provide a coherent way of teaching designers how their systems may be attacked and how they can effectively defend them [35]. Let us know look into attack types

3.1 Attack types

3.1.1 Focused attack

Focused attacks are highly focused on particular or specific kind of systems or environment or ecosystem. This kind of attack has no limitation on time, money and resources. Most practical examples of this kind of attacks are to target defence installations, penetrating enemy communication lines etc.. Most recent example of this kind of attack is building of “stuxnet” whose is targeted to attack only Siemens systems in Iran.

3.1.2 Cryptanalytic attacks

Cryptanalytics is a study of techniques to unravel the meaning of 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 system and knowing 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.

3.1.2.1 Known plain text attack

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

3.1.2.2 Chosen plain text attack

A chosen-plaintext attack (CPA) is an attack model for cryptanalysis which presumes that the attacker can obtain the ciphertexts for arbitrary plaintexts [5]. In this kind of attack, attacker feeds in pre-chosen text into the cipher oracle after which he analyses the result and in worst case can figure out secret key.

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

3.1.2.3 Known cipher text attacks

Under known cipher text attacks, 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 luxury of choosing cipher text or neither can produce more. This is certainly one of the weakest attacks as the attacker will have nothing to work against other than few cipher texts in hand.

3.1.2.4 Chosen cipher text attacks (CCA)

Chosen ciphertext attack is a scenario in which the attacker has the ability to choose ciphertexts and to view their corresponding decryption's plaintext [2]. In this kind of attack, the attacker selects the ciphertext, sends it to the victim, and is given in return the corresponding plaintext or some part thereof [33].

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

Chosen cipher text attacks can be adaptive or non-adaptive. Under non adaptive cipher text attacks, attacker choses certain cipher texts in advance for decrypting them. The clear texts obtained are not used for next cipher operations. A chosen-plaintext attack is called adaptive if the attacker can chose the ciphertexts depending on previous outcomes of the attack [33]. Servers using Cipher Block Chaining (CBC) mode of operation and RSA PKCS1 are under certain circumstances vulnerable to adaptive chosen-ciphertext attacks and such attacks allow an attacker to recover the encrypted data[4]. This means adaptive cipher text attacks are more context based and result of outcome of previous operations. These attacks may be quite practical in the public-key setting. Bleichenbacher in his work [1] demonstrated that plain RSA is vulnerable to chosen ciphertext attack and some implementations of RSA may also be vulnerable to adaptive chosen ciphertext attack.

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

3.1.2.5 Lunchtime Attack

Lunchtime attacks are also referred as midnight attack or CCA1 attack. This is a kind of attack targeted by attackers when the owner or user of the system is away or often when system is not logged in. This is with idea that system is vulnerable and is often less or no resistance when there is no active user which otherwise will be more challenging to penetrate. During this time, the attacker

will generate a pre-chosen cipher text quarries which are valid until period of time after which penetrating will be difficult.

3.1.2.6 Adaptive cipher text attack

Adaptive cipher text attack is also referred as CCA2 attack and is stronger in nature compared to CCA1. This attack relies on approach to select cipher dynamically at runtime when ever attacker is posed of challenge.

This is interactive based attack where attacker sends stream of ciphered texts to be decrypted and subsequent ciphered texts are choosing depending on responses from the system [4].

One increasing order from weakness to strength, above attacks can be sorted as known cipher text attack, known plain text attack, chosen plain text attack, chosen cipher text attack.

3.1.3 Network attacks

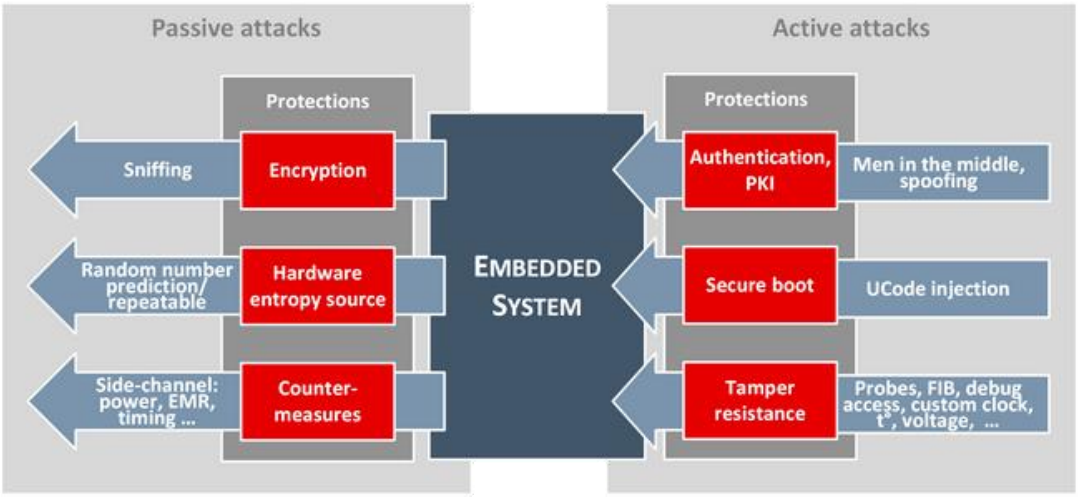
In this world of ever increasing networking, systems have become very attractive targets for external attacks through network. Networking attacks most rely on monitoring, spoofing or masquerading of network traffic.

3.1.3.1 Passive attacks

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

3.1.3.2 Active attacks

Active attackers penetrate the system by circumventing the security and breaking the protection 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 to bringing down the whole system or network especially if the attack is on servers.



Source : Rabou, S. (n.d.). Sébastien Rabou. Retrieved March 19, 2017, from https://www.chipestimate.com/tech-talks/2014/01/14/Barco-Silex-Why-do-you-need-a-hardware-solution-to-secure-your-embedded-system

3.1.3.3 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, dissatisfied employees from an organization.

3.1.3.4 Phishing attack

Phishing attacks are attacks where 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 that can result in stealing of vital information or can even result in financial frauds if the target is banking sites.

3.1.3.5 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 [6].

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 as if it is communicating with 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 certain weakness in the system who has certain knowledge of the system.

Class2: Knowledgeable insiders

Insider attackers are the one who have 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 ,no restriction on funding and are equipped with highly specialized tools to break open the systems.

Below is the table with classification of attacks calibrated against different parameters which was demonstrated in black hat conference by Joe Grand [13].

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 1. Attack difficult [13]

Level	**Name**	**Description**
1	None	Primitive ,No special tools or skill needed

2	Intent	Minimal skills needed to compromise the system
3	Common tools	Technically competent, Can be dealt with tools available in market
4	Unusual tools	Can be compromised with tools can be 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 there after exploit the vulnerabilities in the system in order to achieve his objective. Typical attack vectors are eves dropping, brute force attacks, injecting crafted packets, reverse engineering, fabrication by counterfeit assets of a product.

4 CLASSIFIYING VULNERABILITIES [7]

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 superuser or system administrator to gain full access privileges [11].

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 [11]. Classification of attacks and vulnerabilities can be help us to understand the tools, remedies , approaches that can help us 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 creep in. Some of the popular lifecycle phases that have higher chance to see system vulnerabilities are design , testing ,deployment and maintenance phases.

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

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

In most cases , software and hardware will be tailor made for relaxing certain hard Classic examples of vulnerabilities creeping in testing phase can be debug holes either software or access points in hardware that can

4.2 Classification by attackers objective

One of the most prominent approach is to enumerate the attacks is by attackers objectives like gaining root access and higher privilages, creating denial of service, stealing confidential and sensitive data, malicious code execution, Integrity and security policy violation.

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

One approach in classifying the vulnerabilities is by deciding where exactly they appear in 7 layered OSI reference model. In practice this means to segregate the vulnerability into one of the seven baskets (Application, presentation, session, network, link and physical layer). Attacks can also be enumerated depending on their origin location in network like local system, intranet (ethernet network), internet , wireless network.

This kind of classification may not be appropriate at all times as many of the times vulnerabilities can fall between layers and hence difficult to segregate on this layered approach.For example , it is not often easy to decide on whether the vulnerability is exactly in OS or 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 C language. Likewise systems can be prone to vulnerable to attacks exploiting meta characters vulnerabilities like LDAP and SQL Injection that can happen with database languages.

Systems that run code with memory leaks, malicious code are vulnerable to resource exhaustion if the code can block the system resources and thus can result in DoS especially in the case of embedded systems where resources are

limited. Classic resource hungry operations are TCP SYN Flooding, improper handling of resources which is resource hungry.

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 creep in during implementation phase. To quote a few are, double free memory, executing content from malicious memory locations or locations program either did not allocate of 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.

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 like, memory leaks or buffer overflows etc.. So it makes sense to classify the vulnerabilities on the nature of attack scenario rather than based on effects.

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

4.7 CLASP Classification [8]

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

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 , code injection into data areas of memory.

4.9 Environmental errors

Environmental errors can be as a result of resources exhaustion (for ex: sockets , kernel objects , file descriptors, memory), execution of untrusted code and data, system variable manipulations (for ex , system paths , library paths ..etc) , spoofing of system events, failure to protect secure data and keys, 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.

4.11 Protocol Errors

Protocols errors are one that usually arises out of protocol, algorithm errors that are as a result of improper use or wrong choices. Such vulnerabilities that 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 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.

4.13 Popular Dictionaries for Attack Taxonomy

MITRE a government funded non profit organization that 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 [9].

4.14 PLOVER

PLOVER is a primary list of working examples intended for researchers that lists over 1400 real world vulnerabilities by their CVE IDs organised as a conceptual framework. This framework offers a platform for discussion for further analysis and describing them in further detailed manner. PLOVER is targeted for those who are engaged in vulnerabilities analysis in an effort to understand and communicate them in more abstract level.[9]

4.14.1 CWE

CWE stands for Common Weakness Enumeration which essentially deals with underlying software weakness in general but not specific to particular instance in system. Vunerabilites can emerge from weakness and may have potential for getting targeted. 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 not limited to any specific component or instance in any system.

4.14.2 CVE

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

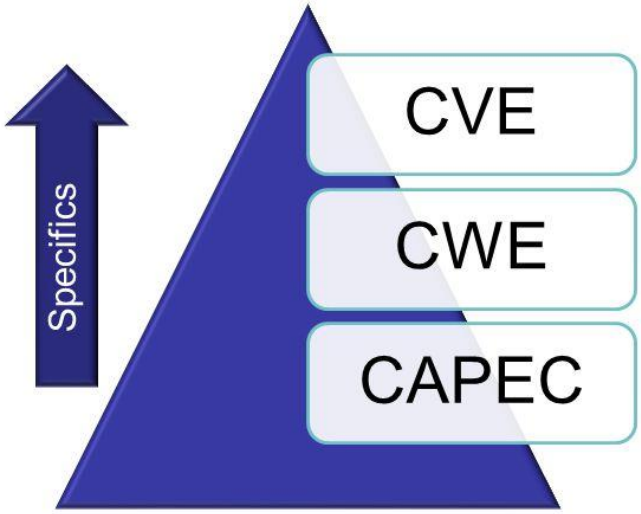
4.14.3 CAPEC

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 [10]. CAPEC provides this information to the community in order to help enhance security throughout the software development lifecycle and to support the needs of developers, testers, and educators [10]. So CAPEC is a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy created to assist in the building of secure software [10]. 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 [37] on how CWE- CVE- CAPEC stack each other.

KNOWLEDGE consulting group KCG

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

5 PRINCIPLES OF SECURITY

Primary principles of security is to preserve the confidentiality, Integrity and availability of the consumers, stakeholders, 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 [30]

Source: http://www.pearsonitcertification.com/articles/article.aspx?p=2218577&seqNum=3

5.1 Confidentiality

Confidentially refers to restriction of access to sensitive data to unauthorized sources. Loss of sensitive information like device or user passwords, credit card and banking credentials could result in identity theft, financial implications and compromise privacy.

According to Mathew and Stan [32], Designers shall strongly consider measures to ensure confidentiality and prevent sensitive information from reaching unauthorized sources, 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. [32]

5.2 Integrity

According to study published in security blog by Mathew and Stan [32], Integrity involves in maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle. Data generated at 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 shall need to plan for backups in the event of any damage to data to restore to its correct state[32].

5.3 Availability

Terry refers availability as one of the pillars in CIA traid [31], According to 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 information, but also to systems that cater services. DDos attacks on such systems will deprive and deny users to access to systems thus hurting the systems availability. If critical services (systems) are attacked and brought down, it could cripple services and can caste impact on economy and people's lives. DDos attacks on mission critical systems like aircraft and space ships can hurt the availability and can bring down the whole system threating the lives.

Terry in her report 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 is highly critical[31]. These redundant copies can be stored in different geographical areas to mitigate against natural disasters and intentional attacks by malafide people. Designers while designing embedded products should make sure the systems can repel or filter such DDos attacks, have proper intrusion detections mechanisms, can raise alarm, have damage control mechanisms in place in 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[31].

	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.

Source: Slideshare.net. 2011. Jul 21. BDPA Charlotte - Information Technology Thought Leaders, Information Security and the SDLC Retrieved from https://www.slideshare.net/bdpacharlotte/information-security-and-the-sdlc

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) [29]. Given their size, portability and cost sensitiveness, They also often come with other resource limitations computational power, absence of protective theft and shielding technologies when compared against with 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 [28]. For consumer devices like mobile phones and IoT devices, apart from size , computational power limitations, it is extremely important for these kind of devices to be designed keeping in security of the user data and software attacks in mind.

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

6.2 Reliability

Embedded software systems are generally held to a much higher reliability standard than any general purpose software [29]. They are also know to work in extreme and tough conditions, so they are prone to more failures. Some times these the failures associated to be with embedded can be quite risky, threating, can cause substantial damage in terms of monitory or personal loss. In mission-critical applications such as aircraft flight control, severe personal injury or equipment damage could result from a failure of the embedded computer [28]. Failure to safeguard the user data in mobile phone can have financial implications. Traditionally, such systems have employed multiply-redundant computers or distributed consensus protocols in order to ensure continued operation after an equipment failure [28]. So for designers it is important to device alternate

solutions for mission critical systems. But for consumer and IoT devices designers should consider damage limiting functionalities 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 the system costs controlled by keeping system complexity down which should not cast effect on integrity or reliability of the overall system.

6.4 Functionality creep

Designers tend to beef up products with functionality to compete in the market with their competitors. This is partly because consumers prefer to pick products offering the better functionality against the ones that offer more secure solutions. So in this ball game, it is often challenging for designers to balance out 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 has not kept up with the knowledge that attackers have gained [35]. So for designers it will be challenging to get better off the attackers to come up with good solution.

7 MECHANICAL DESIGN ASPECTS

Let us discuss here about mechanical design approaches that should be considered for bringing out a secure product.

7.1 Product housing

Typical attack hard point in this case would be product enclosure and attack vector where attackers will make an effort to open the casings and access to internal circuitry and components. Product designers needs 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 product with a single piece outer shell.
- Using high melting point glues where ever needed.
- Designing in such a way that opening of device needs complete destruction.
- In addition to above, leave surface points for device accessibility for service personal who can open the device with specialized tools.

7.1.1 External Interfaces

External interfaces like proprietary connectors, Ethernet, RS232, USB, Firewall, JTags, Wireless (802.11) and Bluetooth are vital lifeline for the product to outside world for proper functioning of device. 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 root of the attack tree can originate from here. Attackers indulge in probing, sniffing, flooding and push malformed packets through these interfaces. Product designers should be able to defeat all possible attacks that originate from these external interfaces. Designers should keep in mind below points while interfaces are designed.

- Device shall transmit only non-sensitive and public information in clear text format.

- Encrypt all sensitive and confidential information that has to be exchanged over interfaces.
- Adequate protection inside the device to defeat spoofing, malformed packets. For ex , Hardening of OS, strong firewall etc..
- Strict no obfuscation policy, attackers are always ahead of designers and can uncover them easily.
- All the diagnostic ports , backdoor interfaces and JTAG connecters to be removed from production software images. Closing them by employing techniques like 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 device by tampering to know confidential information and know working internals device. Some of the tamper mechanisms worth considering by product designers are

7.1.2.1 Tamper resistance

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

Products with are well protected housing would require partial if not 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.

7.1.2.2 Tamper evidence

Tamper evidence mechanisms are designed to ensure that visible evidence or trails left for an attempted break in. These mechanisms do not protect the device or the confidential data there in but to 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 mechanisms will leave a definite very well evident cracks or damages on the surface that are hard to be unnoticed.

7.1.2.3 Tamper detection [12]

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

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

7.1.2.4 Tamper response

Systems should trigger shutdown and disable themselves in response to tamper detection. They should be designed with capability to log and generate forensic data for further analysis post tamper detection that can be accessed by service personal. Designers should also consider techniques such as erasing the sensitive data in response to tamper detection. Products that house confidential data, keys should protect the data by resorting to erasing methods. Confidential data is either stored in RAM or ROM. Erasing random access memory is relatively easier and 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 system employ to ultimate mechanism of physical destruction by shorting the circuits and rendering device inoperable.

Further care has to be taken these tamper response mechanisms does 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 like FPGA , processors and memories. 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 for ex : COB (Chip on Board) packaging, CIB (Chip In Board) packaging, Ball Grid Array packaging. Product should demark or erase all the chip markings either by black topping or using other methods like small sander or etching. 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 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 product can act as potential attack points which attackers can monitor to determine secret information. So every step needs to be taken to minimize the emissions by installing appropriate shielding and taking care of unprotected I/O buss from ESDs. A well designed power lines and grounding can reduce noise levels and emissions. All unused GPIOs should be either disabled or to be fixed to 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 shall not use the unsecure bus in the system. Memory devices like RAM and ROM are known to be unsecure. Even though systems are designed with proper tamper detection and response mechanisms like complete wipe out or erasing of data, but there is high chance that traces of data can still be being 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 the secure data can be stored in memory, should erase information once their need is done.

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 I will try to focus our discussion on the role of hardware based solutions touching associated software functionalities that exploit these specialized hardware where ever needed. Security solutions that utilize underlying platform and hardware assistance are more effective than purely software based.

8.1 Smart Card [27]

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.

Every component in the smart card is deemed secure, basic in execution with very less computational power. Smart cards also has higher level of temper resistance because of their size , complete physical isolation of the trusted area, narrow interfaces .The isolated trusted area plus higher level of temper resistance makes the trusted services leveraged by a smart card are very secure, but very limited in scope due to lower computational capabilities. [27]

8.2 TPM

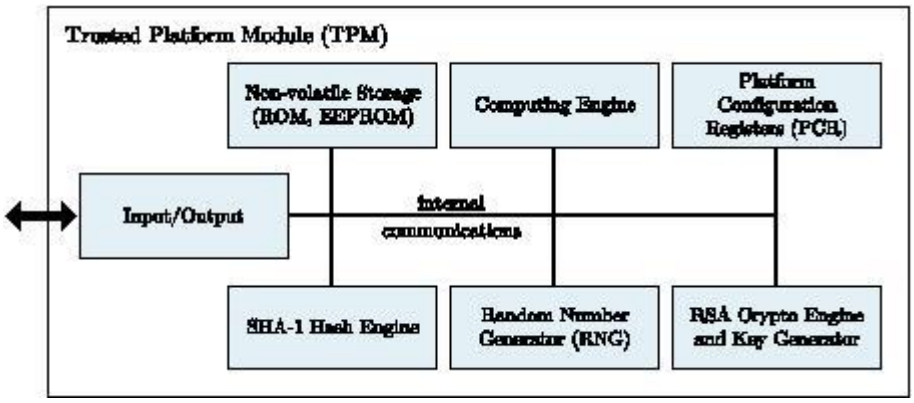
Trusted platform modules (TPM) are dedicated secure crypto-processors that are designed to secure hardware or software by integrating cryptographic keys into device. TPMs chips are passive and execute commands from CPU. They by no means will decide or control the execution flow. Generally there is additional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

Global standards for specification for TPM are controlled by Trusted computing group (TCG) alliance formed by HP, Intel, AMD, Microsoft. Objective of TCG is to develop, define and promote open standards for hardware enabled trust computing and standardizing software interfaces across platforms. These specifications aim to provide platform independent functionality that must be provided by any trusted platform by facilitating a common interface for secure computing en-

vironment thus protecting and securing the platform against hardware and software attacks. There are also criticisms on TPM specifications that it is crippling the user rights by moving rights management from software to computing platform hardware.

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

Simplified Architecture of the TPM



Source: http://www.fidis.net/resources/deliverables/hightechid/int-d37002/doc/9/

First generation chips are physically separate which made platform less secure. Latter on they were either mounted in mother board or or embedded into the SoC. Mounting the TPM chips in motherboard are prone to physical attacks and are too less secure. Mandatory objectives of TPM chips are

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

TPM chips achieve the above goals by implementing below functionality

- 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 visible outside the chip to avoid any kind of phishing attacks and prevents the key from being copied and used without TPM. They can also be configured to be used by providing right authorization values which and when only they are unsealed. This in practice means the keys are accessed only when the platform is in 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 get unsealed only TPM is provided with same values by which integrity of the system is verified. When an attempt is made to boot the system with a alternate image or presence of additional malware or suspicious code will result in different PCM values and thus fails in integrity check there by keys cannot be unsealed.
- Attestation functionality: A mechanism where TPM can certify the platform as trustworthy and haven't been breached. By using the attestation, trusted clients can prove to 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 signature using private keys known only by TPM. During attestation operation, the evaluating entity will request a TPM quote which is essentially a signed composite hash of selected PCM data and its signature. This quote is there after is matched with the values generated during provisioning.

8.2.1 Root of trust for storage management

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

8.2.2 Root of trust for reporting

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

8.2.3 Root of trust for measurement

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

8.3 Secure Element

Secure elements are temper-resistant hardware elements that holds device secrets and limited isolated execution environment. It can also be added to any device with a microSD or an Universal Integrated Circuit Card (UICC) [27] .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 the chip in for illicit use is either hard or impossible to do. The main benifits offered by Secure element are

- Data Protection : Protection of data from Unauthorized access[27]
- Hardware assisted cryptographic operations: Dedicated inbuilt hardware logic for cryptographic functionalities like encryption, decryption, hashing etc. [27]
- Isolated execution environment: Small defined tasks (applets) that perform cryptographic operations with keys and data stored inside SE can be run in isolated execution environment without the data leaving the TEE. [27]

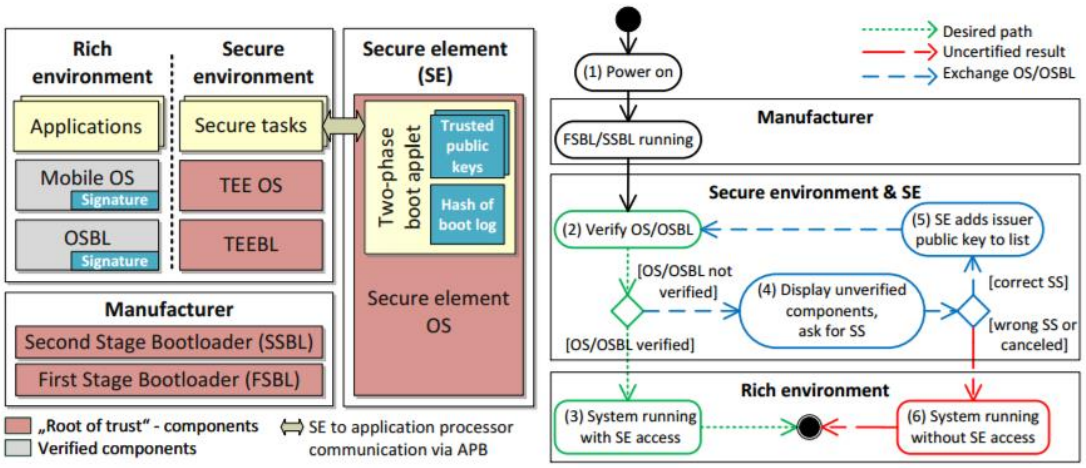
This technology is widely used in the EMV chip on payment cards. Security offered by SE is considered to be on greater degree of evaluation level (EVL5) compared to device that offer solutions over TEE.

8.4 Hardware assisted boot process

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

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

Components that are deemed to be unaltered during the lifecycle and immutable are considered as root of trust components. It is usually ROM programed in factory and are bound to never change. Contrary to the applications running in rich OS, the secure tasks running in the TEE can be trusted. Also in this case , SE is configured as a trusted peripheral and is only accessible from secure tasks that execute in 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 like device identification, device management, decryption and encryption.



Source: A Practical Hardware-Assisted Approach to Customize Trusted Boot for Mobile Devices [22]

As we can notice in the above figure, the root of trust components considered here are manufacturers boot loaders and hash of 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 that are only accessible to 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 chain of verification are pre-signed by OEMs private key corresponding to the fused public key hash in secure element or fuse memory. During the secure boot process, manufacturer bootloder verifies the signed images are signed with corresponding private key whose hash of public key is stored in Secure element. It is purely prerogative of the OEM to whether to consider TEE OS to be under root of trust. In case not, Even TEE OS and its bootloader needs to be verified. If verification at any stage fails, Device assumes to have been tampered and exits the boot phase.

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

8.5 Hardware assisted isolated/trusted execution environments

As described in below figure, there are more than one ways where isolation execution environments can be realized. Critical applications or part of use cases that need 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 results be consumed by primary processor.
- Using Embedded Secure Elements (ESE) for secure operations which will be embedded inside the same System On Chip.
- Advent of bus and memory isolation hardware extension logic blocks made it possible for single processor (SoC) to be split into untrusted and trusted execution modes.

Through secure operations can be controlled to be executed in secure mode where untrusted entities will have no access to secure peripherals.

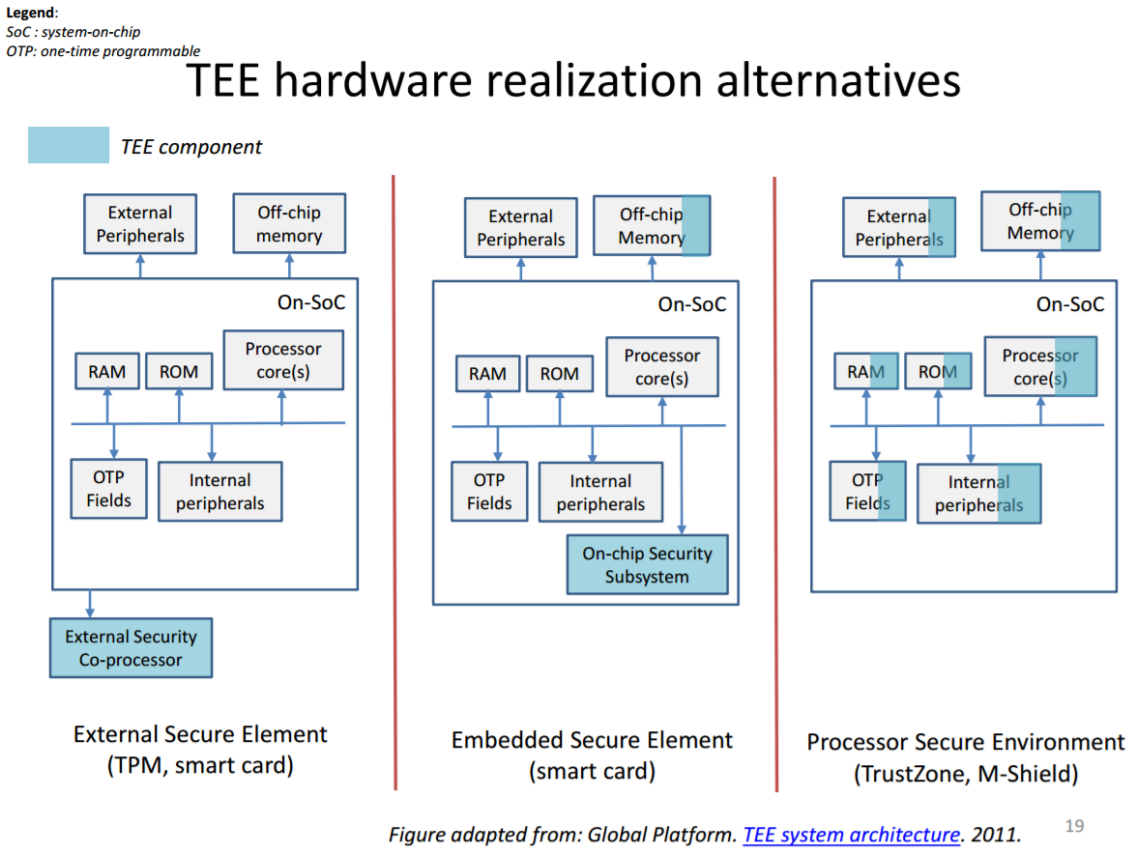


Figure adapted from: Global Platform. TEE system architecture. 2011. 19

Source: Mobile Platform Security , “Trusted Execution Environments” [20]

Although secure element offers higher degree of security, 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.

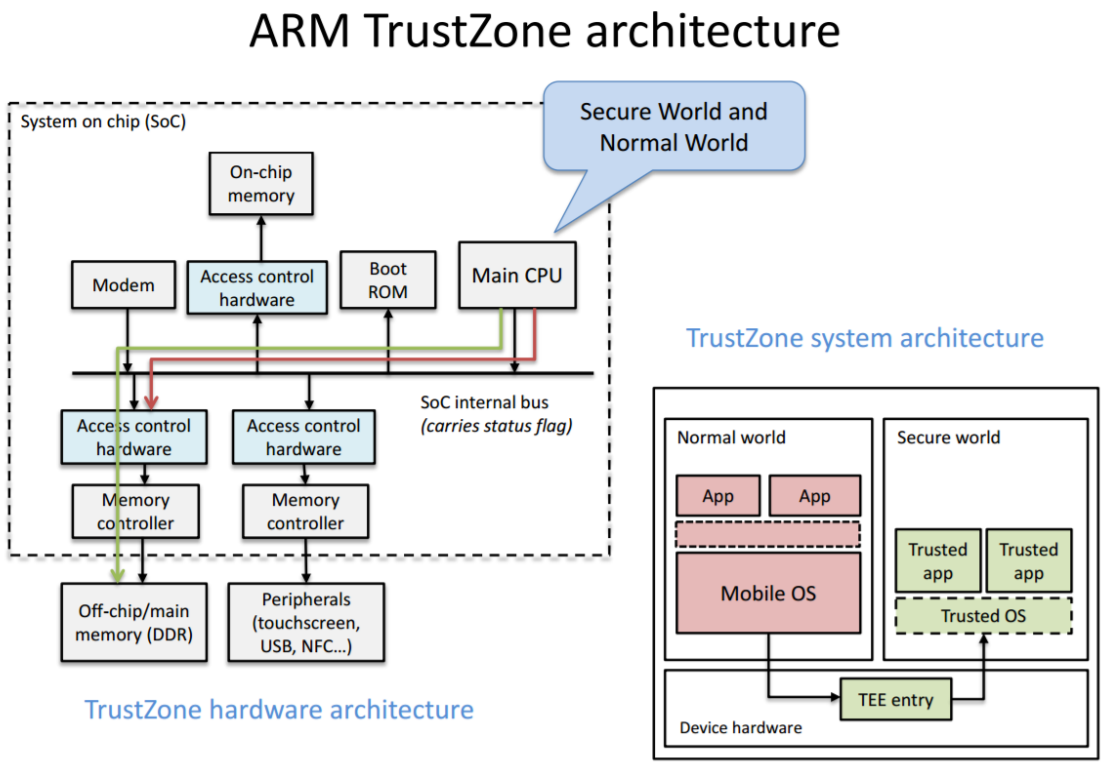
Recent trends show that TEE environments are widely used mobile payments, enterprise (bring-your-own-device), and content protection and government electronic ID solutions.

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

Trust zone technology will ensure that data and operations on the device remains secure, protecting consumer privacy and enabling a range of service like mobile banking, payment usecases and playing multimedia entertainment to be 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 minimal impact to the core area or performance while enabling developers to build any additional security, for example cryptography, onto the secure hardware foundation.



Source: Mobile Platform Security , “Trusted Execution Environments” [20]

Although ARM introduced TrustZone 10 years ago, Trustonic and Xilinx were the first ones to propose frameworks and solutions utilizing this hardware platforms and that makes it possible for the research community [21] as well as the industry to experiment and develop innovative solutions on it.

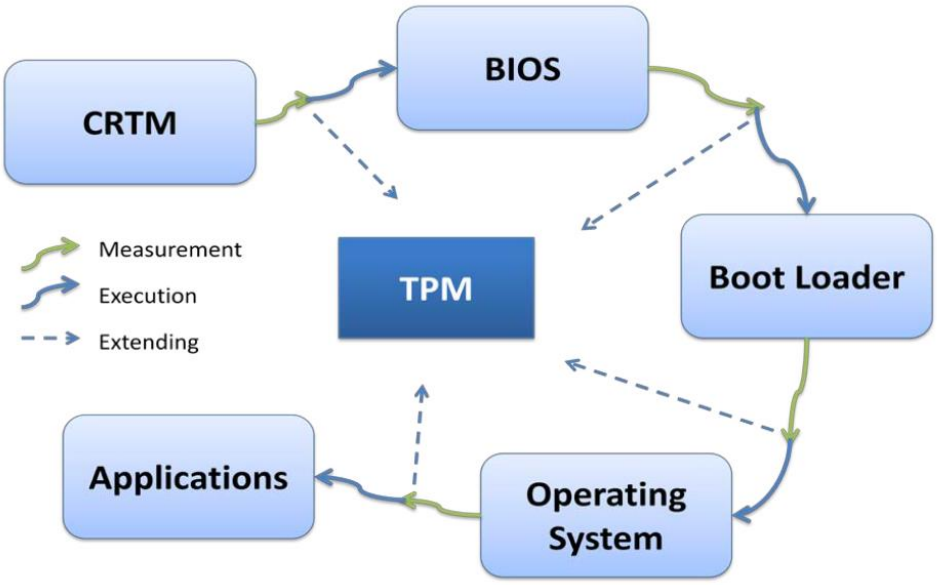
9 SECURITY ARCHITECTURES

In this section we will discuss on ways software can be designed to exploit the specialized hardware modules (discussed in earlier section) that lay foundation to secure solutions. We will also discuss about the some profound architectures to design software solutions.

9.1 Static root of trust measurement

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

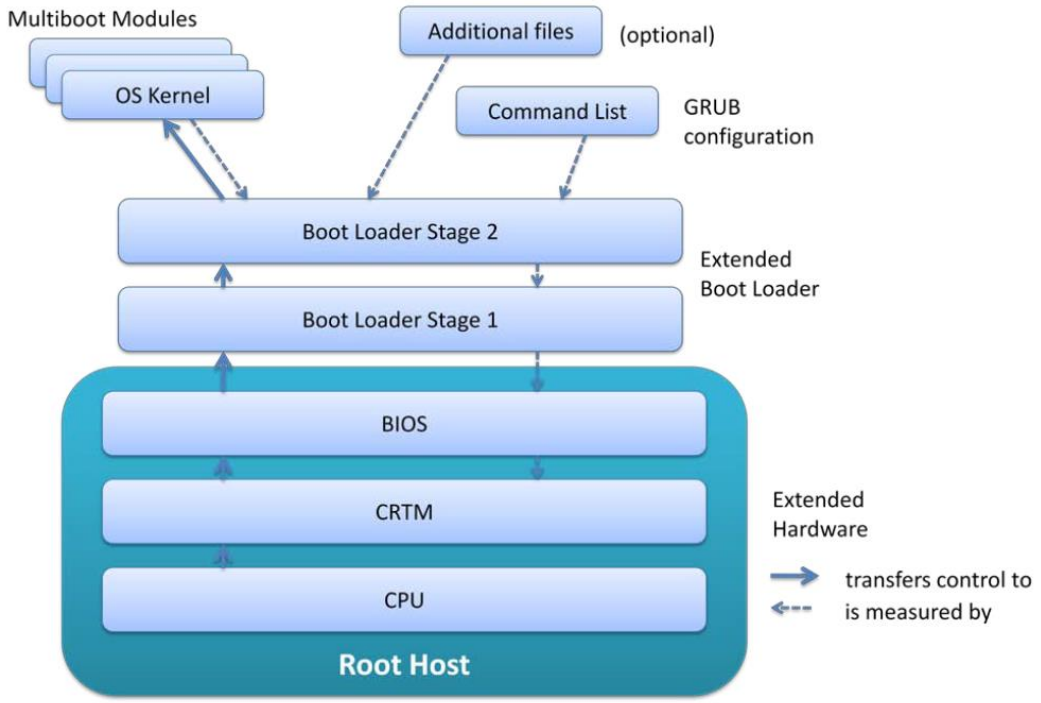
In any embedded system the first executable is usually the ROM BIOS code , but to enable static root of trust there needs to be an additional immutable component that would verify the bios before it starts to execute which is what is Core root of trust for measurement (CRTM). CRTM is either stored in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



source: http://security.hsr.ch/mse/projects/2011_Root_of_Trust_for_Measurement.pdf

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

Even though there is 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



(source: http://security.hsr.ch/mse/projects/2011_Root_of_Trust_for_Measurement.pdf)

Of course this static method to verify is good for load time ,it also comes with few drawbacks like scalability and time to check and time to use (TOCTOU) issues .

Scalability is the issue given the case that all the executables , libraries and scripts will need to be part of 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 varied order of execution which can potentially change the PCR measurement values.

Time of measurement plays an important role due to the fact that even though 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 can put the system to unsafe state. It is essential to be aware that the Static Root of Trust for Measurement only gives load-time guarantee not run-time guarantee; that is it gives assurance of what program is loaded, not necessarily what is running [17].

9.2 Dynamic root of trust for measurement

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

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

9.3 Multiple Independent layered security (MILS)

Legacy secure systems were designed around secure kernel and trusted computing base, with the idea that all the security decisions and the security enforcement mechanisms are an integral part of the TCB[15]. This methodology

can increase the complexity of TCB as designers tend to pump in more and more logic there thus creating issues for maintainability.

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

Multiple Independent Levels of Security (MILS) is a security architecture based on the concepts of separation and controlled information flow implemented by separation mechanisms that support 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 us with manageable and, more importantly, evaluable implementations [16].

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 [15]. Under MILS architecture, privileged mode processing is isolated from under privileged user data or applications. MILS architecture is targeted towards mission critical operations where failure cannot be even thought of. The MILS architecture is intended for use at the highest levels of security. Consequently, we are targeting an EAL 7 [16]

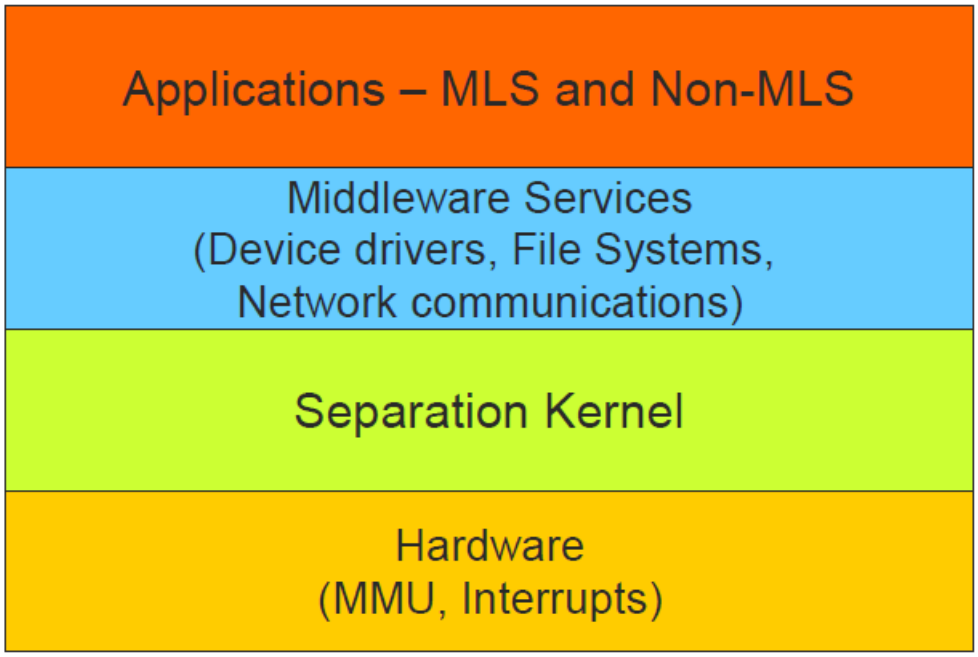
Common Criteria	***MSLS / MLS Separation Accreditation***
Basic Robustness (EAL3)	System High Closed Environment
Medium Robustness (EAL4+)	System High Open Environment
High Robustness (EAL6+)	Multi Level Separation
DCID 6/3 Protection Level 5	***Multi Nation Separation Accreditation***
DO-178B Level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

Source: Multiple Independent Levels of Safety & Security (MILS): High Assurance Architecture [14]

Advantages of MILS architecture.

- Reduce the footprint of secure critical code dramatically, the lesser is better.
- Isolate the critical components and operations in the system.
- One processor can host multiple applications at different security levels [16].
- 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.[16]

Conceptual VIEW OF MILS layers described by Taylor and Alves-Foss [15]



(https://www.acsac.org/2005/case/thu-130-taylor.pdf)

9.3.1 Separation kernel (SK)

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

Four main objectives for secure kernel are data isolation, highly controlled information flow, data sanitization, damage limitation[16].

Kernel conceals and denies access to application data and memory contents between partitions. State of executions in current partition will not effect state of executions in other partitions and vice versa. In practical scenario, it might not be desirable to achieve total isolation between partitions as minimal communication across them might be needed. For such situations, secure kernel defines some secure authorized channels for inter-partition communications through which where data sharing can happen. 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 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. Secure kernel also cater to all partitions by enforcing bounds on shared resources and guaranteeing minimum processing times , interrupt servicing times and access to resources.

9.3.2 MILLS Device Drivers

MILS architecture requires kernel to be small, having minimum footprint in order to be fully evaluated. In any typical microkernel architecture, other needed OS services are typically included in operating system running as either as separate threads or processes but not in kernel space like any other monolithic kernel [15]. But in 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 like communication across partitions but are mediated by separation kernel policy. When the device is categorized as private and unsharable, access to 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.

9.3.3 Hardware support

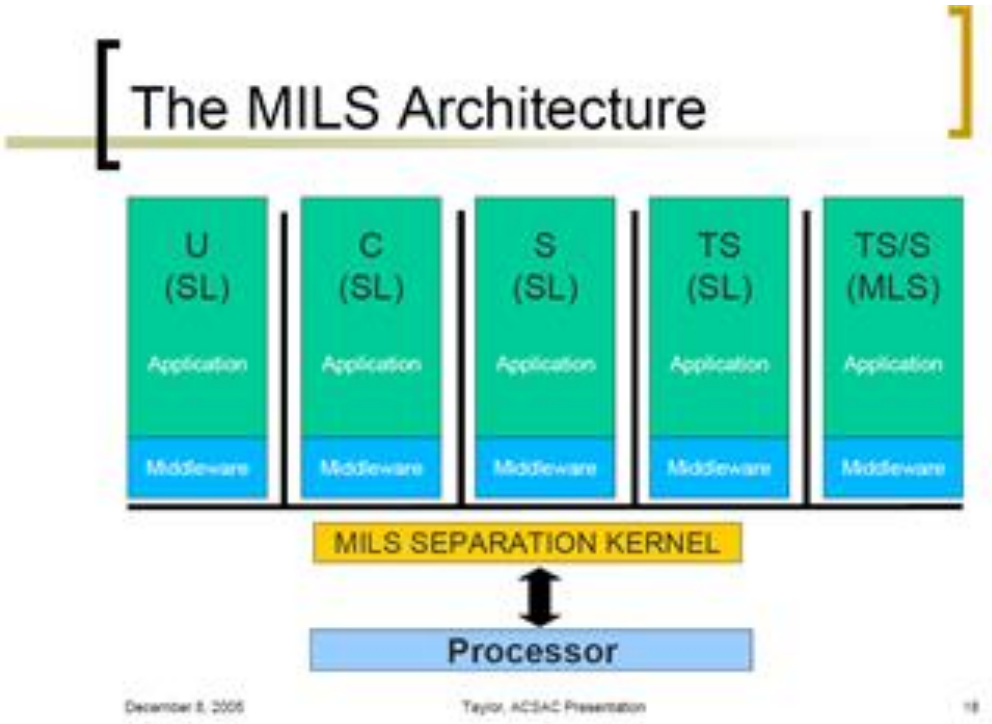
Secure kernel needs support of underlying hardware for effectively executing partitioning, control information flow and isolate system resources. SKPP (secure kernel protection profile) mandates certain functional requirements from hardware which secure kernel shall operate on.

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

9.3.4 Middleware services

Middleware layers sits on top of 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 above, middleware layer is also responsible to end to end security and communication policy by labeling, filtering and controlling 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 confinement to single process,

partition or core.

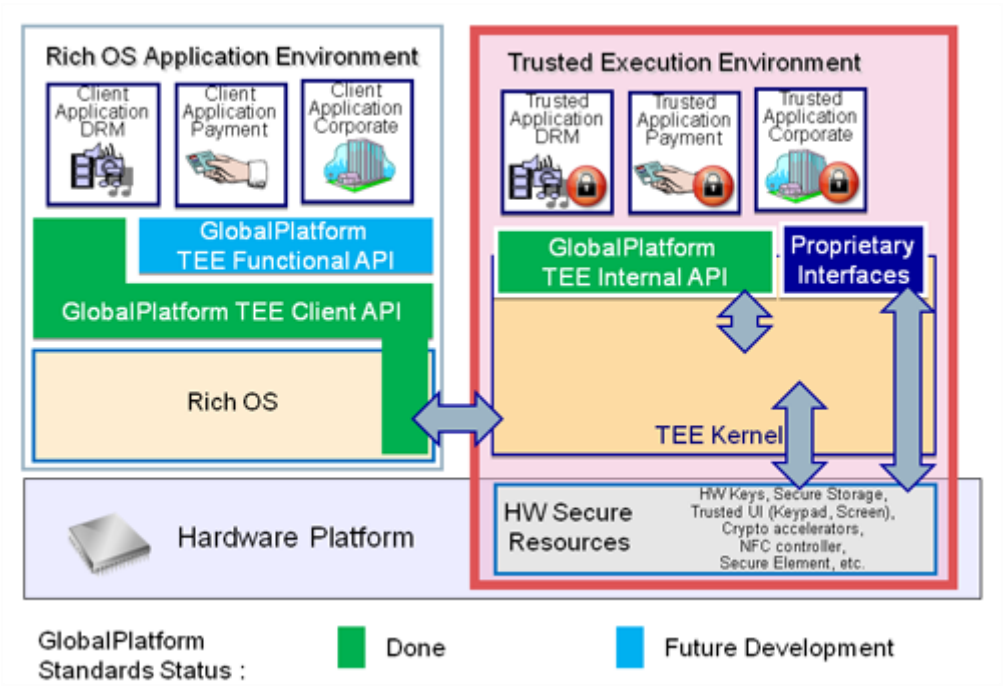


(source: https://www.acsac.org/2005/case/thu-130-taylor.pdf)

9.4 Trusted execution environment (TEE)

Global Platform defines TEE as the combination of a hardware platform that provides isolation and a software and operating system residing within the security domain defined by that hardware that is capable of running programs launched into that environment. This software running in isolated environment is responsible for executing critical secure applications which accesses critical resources as part of their use cases. Systems that needs to run secure use cases can deploy TEE run in secure environment to better protect their secrets, For example smart cards and DRM applications. Let us dwell deep into what components comprises and constitutes these TEE environments and their roles.

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



Source :https://www.arm.com/assets/images/GP_Standardization_500.png

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 later part of document that deals with device security in general. The core that operate TEE comes with immutable ROM boot code who does needed software checks that are needed to put the processor to secure context.

9.5 Virtualization

Through virtualization, secure isolated environment can be created by host operating systems by leveraging on OS concepts like 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 TEE do also have some vulnerabilities as explained below

- Virtualization technology for achieving trusted execution environment EE as security solution can be risk as it is vulnerable to hardware attacks.
- In this approach, larger conventional operating system or hypervisor enjoy higher privilege contexts leaving virtualized environment OS to work under user level. In this case, virtualized OS that provides 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 traditional rich OS (Android & Linux) or OS residing in isolated execution environment. Here I will rather confine our discussion at generalizing various security measures that traditional operating systems provide.

9.6.1 Process Isolation

Process isolation as the term self-explain relates to isolation of individual process from the address space of other processes made possible by underlying kernel to prevent one process either accidently or intentionally writing into each other's space. Interaction between processes can be still 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 sandboxing mechanism to facilitate a highly controlled environment which provides 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 work seamlessly with 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, 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 we discuss different mechanisms to achieve this and their levels of protection. Access control mechanisms can be broadly split into discretionary access control and mandatory access control.

9.6.2.1 Discretionary Access Control

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

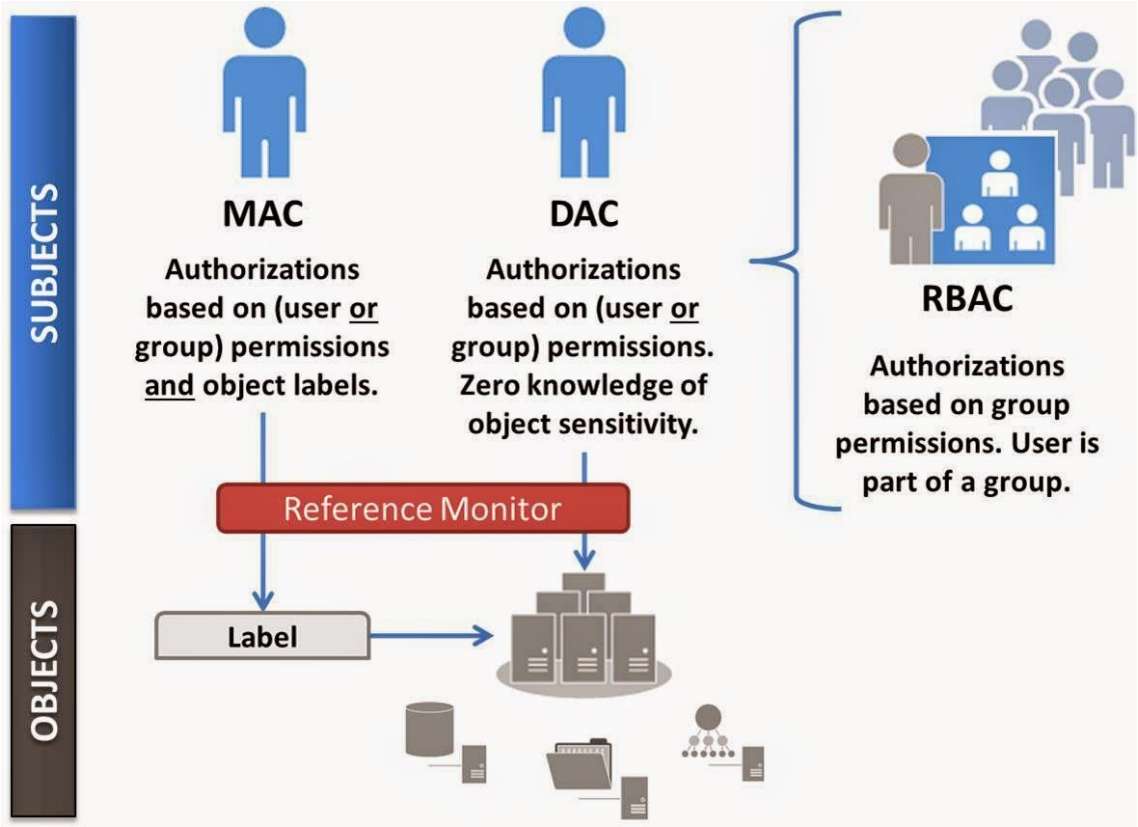
In Unix DAC, ACL lists forms the backbone for the OS security model, Programs launched by a user run with all of the rights of that user, whether they need them or not. There is also a super user category, which is more powerful entity that bypasses Unix DAC policy for the purpose of managing the system. 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 takes 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)[23]. Running a program as the super user provides that program with all rights on the system [23]. There are also other ALC mechanisms that deal with more fine grained schemes allowing separate permissions for different users and groups for same resource.

9.6.2.2 Mandatory Access Control (MAC)

Unlike DAC, 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. 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 done in the organization. MAC brings in an additional layer of permissions that are associated with user identity, every object inside the system has its own label associated with the object. Now, based upon permissions granted to subject (user) and that are associated with user's role or group, they can only perform actions on tagged objects if there exists an explicit policy that 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.

Role Based Access Control (RBACL) is a widely adopted mandatory access control mechanism around for quite some time. Role based access control is more rigid in a way that access to resources cannot be granted above the roles permits even though user needs extra explicit permissions for the resource. In the linux world , SELinux , SMACK and AppArmour are widely used implementations for MAC. As part of the Android security model, Android uses SELinux to enforce mandatory access control (MAC) over all processes, even processes running with root/superuser privileges (a.k.a. Linux capabilities). SELinux enhances Android security by confining privileged processes and automating security policy creation.[24].



Source: http://www.cloudauditcontrols.com/2014/09/mac-vs-dac-vs-rbac.html

9.6.3 Operating systems support and extensions

9.6.3.1 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 user space, for ex base address, mapping of libraries, stack and heap addresses. This kind of randomizations can help preventing buffer overflow, code injection and return-to-libc attack difficult to exploit relying on luck to break into. External PaX projects like Exec Shield, grsecurity projects have long maintained patches to Linux kernel for these software-based hardening features. Android 4.0 has support for ASLR completely discarding support for 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 support for cryptographic operations.

9.6.3.2 Secure Computing Mode (SecComp)

Secure computing mode (seccomp) 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 don't need. The system call API is a wide gateway to the kernel, and as with all code, there have 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.

9.6.3.3 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 operating system kernel memory is marked as read only and execute sections where as data section is marked as 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 usally enabled with kernel configuration options and specialized flags that control these features.

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

9.6.4 Integrity management

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

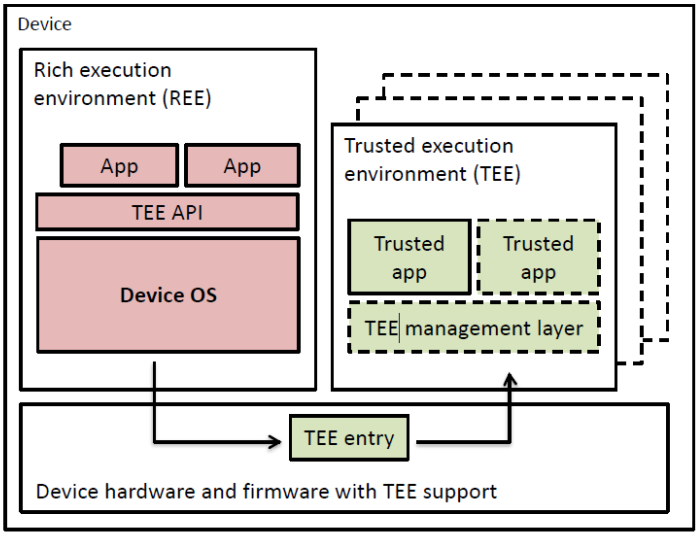
10 DEVICE SECURITY

This topic deals more about the methods and technologies in tandem for enhancing 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

Isolating execution also often referred as trusted execution environment (TEE) gives the device and so the service and providers and OEMs the ability to run a 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 like Android, IOS, linux provide 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 isolated execution means security no more relies on the conventional rich execution environments (REE) Oss but with alternate operating system that hosts 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 non secure world. OEMs can configure the peripherals by restricting or extending their access either to isolated secure environment or to untrusted execution environment or both. Memory isolation is provided by MMU extensions which restricts some predefined memory pages accessible to processor only in secure context. But these hardware assisted methodologies is not of much help in case software designed to use this is not designed well for secure use cases. Figure below explains isolated execution environments in general.



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.

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 device (OEM), user, operator and platform providers meta data specific, service providers.

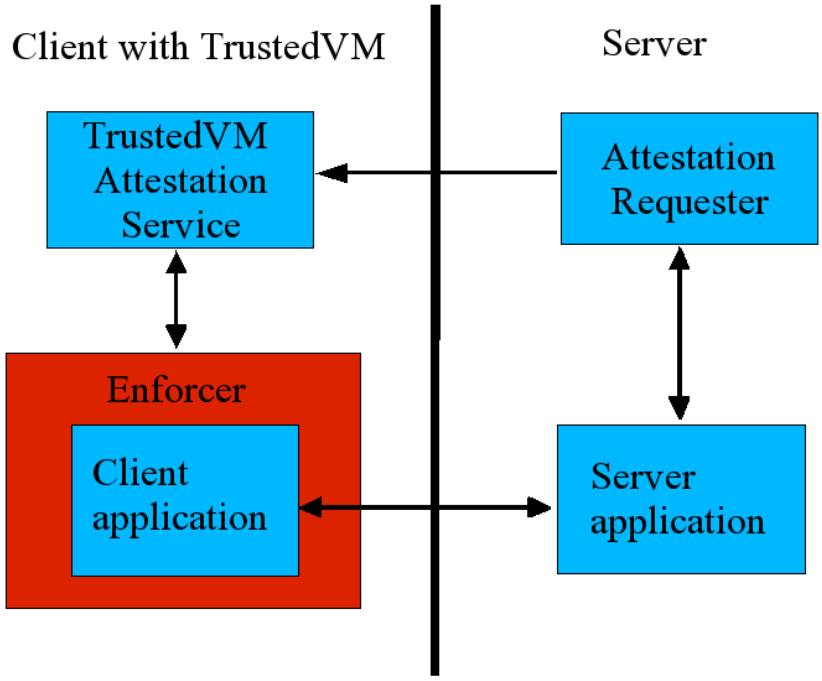
Protection for confidential data is critical so that they can't be accessed either by network attacks, physical tampering or access by unauthorized software entities in the device. Often embedded devices can be reprogrammed remotely as a part of fixing bugs and adding new features. In due course, devices should ensure that attackers don't 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 remote entity to verify that a particular message originated from a particular software entity[18]. Putting this to business terms, service provides widely use remote attestation to make sure that the device they are communicating is the real device that are wishing to talk to and not any untrusted or unknown entity.

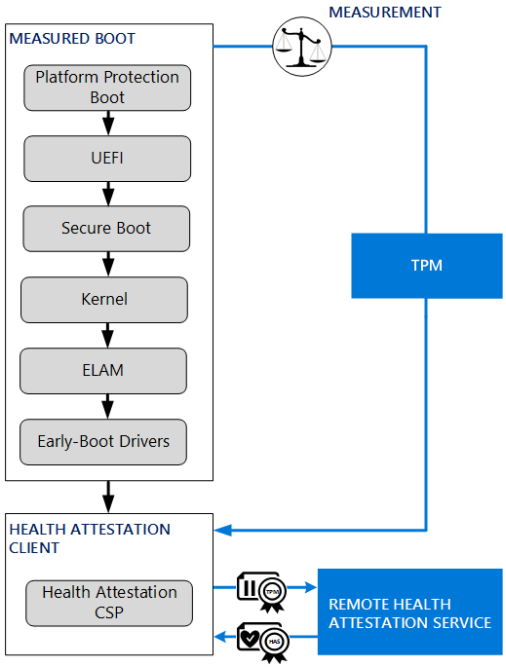
Trusting the device would mean attesting certain chosen software module(s) or complete stack of underlying software running along with it. This could in practice means whole operating systems. Service providers who offer secure services like payment services can know if the communicating device software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

Attestation can be meaningful and easier to achieve when computing base is relatively small which is difficult to achieve with rich execution environments like Android, IOS and Linux. In practice attestation is achieved via apps from service providers hosted in isolated execution environments which has attestation capabilities. These mechanisms are typically built using a private key that is only accessible by a small computing base and kept in secure storage [18]. 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 measurements of loaded code and device private key can then be used to generate signed attestations about its state or the state of the rest of the system[18].



Source:https://www.usenix.org/legacy/event/vm04/tech/haldar/haldar_html/attest.png

Figure below denotes a health application capable of performing device attestation (certifies the device in rightful state) and there by communicates to the service.



Source: https://i-technet.sec.s-msft.com/en-us/itpro/windows/keep-secure/images/hva-fig7-measurement.png

10.4 Secure provisioning

Secure provisioning is a mechanism to send data to a specific software module, running on a specific device, while protecting that data's secrecy and integrity.[18] Secure provisioning comes in handy for migrating users data between user's devices.

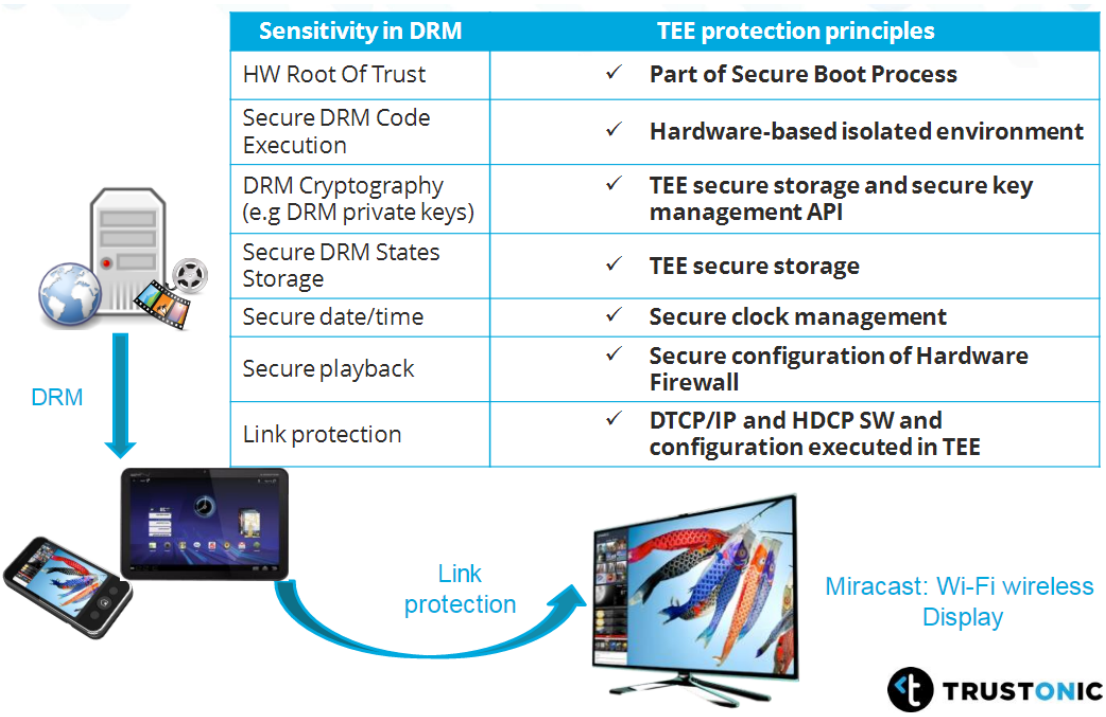
Remote attestation can be used for provisioning data securely by using keys belonging to particular software entity on specific device. The sender can then use that key to protect data to be sent to the target software module on the target device.

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 [19]. Other secure provisioning mechanisms are likely implemented and used by device manufacturers to implement features such as digital rights management. As far as we know, however, secure provisioning mechanisms are not available for direct use by arbitrary third-party developers on mobile platforms. [18].

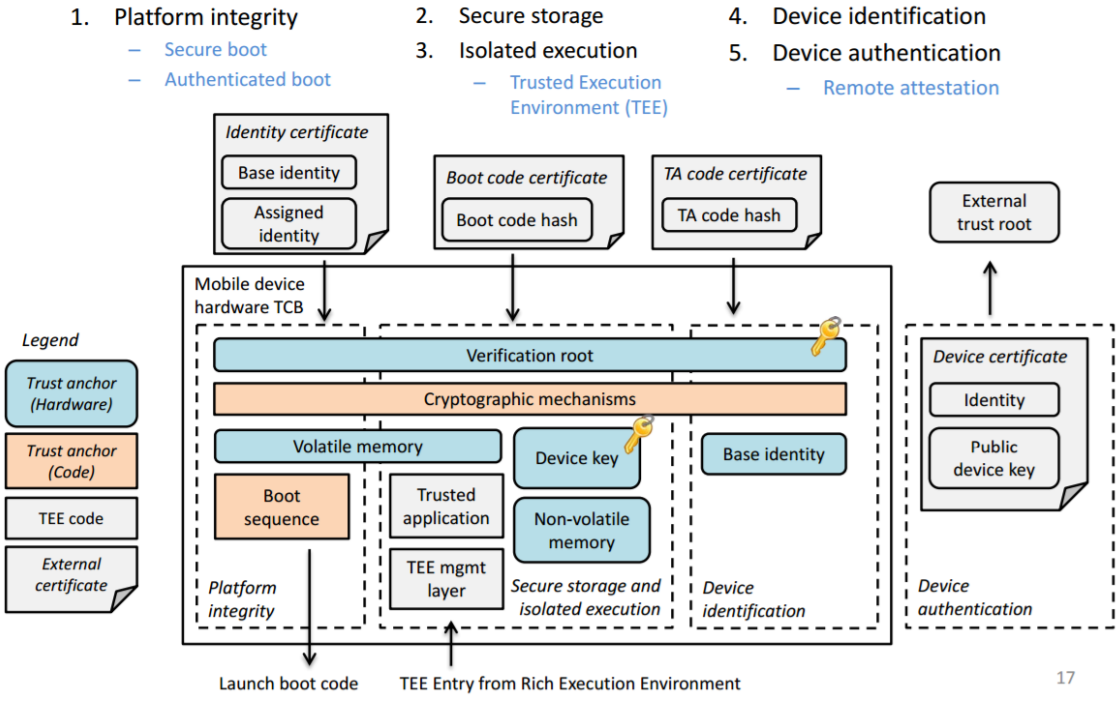
10.5 Trusted path

Devices should be able to setup a trusted path for all related secure use case to ensure the data and content protection. This can be done by enabling set of security features that enable software that runs on top of hardware isolation provided by SoC. Trusted path protects authenticity, and optionally secrecy and availability, of communication between a software module and a I/O peripheral (e.g., keyboard or touchscreen) [18]. Classic examples of these requirements is playing DRM protected media content rendering, enabling fingerprint sensor for device locking and unlocking and providing trusted path for password authentication for payment applications. Establishing trusted path helps devices in protecting assets against unauthorized tampering, modification of software by unauthorized software. Below figure explains the protection mechanisms that comes to play when rendering DRM protected media content.



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 [18]. This is because the computing bases of rich execution environments are large and 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, trusted path can be established. But this requires the hardware based support a low-level access-control policy for access to peripherals. [18]

Below figure from Jan-Erik and Kari [20] shows an excellent relation between functionalities and level of hardware support to achieve security.



11 CONCLUSION

Device security should be treated with paramount importance and should be ingrained in the product design right from conceptual phase; should never be treated as “nice to have” only if time, features and functionality permits. 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 , so care should be take right from the initial phases of design. So during the conception phase of product/project, security requirements needs 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 Security driven development (SDD) ; where each of the requirements are tied down with security related measure , binding or precondition.

As part of pre-design, Architects needs to do an end to end threat assessment, figure out Asset classes, come up with attack trees and entry points, attack categories (Network , physical abuse , Software , cryptographic) for the product. Each of them should be thoroughly discussed, consider solutions to minimize the attack surfaces while designing.

In future we can expect an increased trend at software solutions basing their trust anchor in hardware which even semiconductor vendors/IP Licensees’ like ARM , INTEL , QUALCOMM , TI are gearing up with hardware extensions to support this.

In future, I also see cloud security gaining prominence in addition to client (device) security. Cloud spaces have an advantage of foot print of cloud spaces could provide bigger and better security infrastructure compared devices. In days to come, I can see devices security more restricted to establishing trust and secure connection to cloud space where all the computational and storage logic is hosted.

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

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

- Weakest links should always be secured. For example, often in cryptographic operations, it is found that weakest links have been the process of storing secret keys or restricting their access permissions rather than cryptographic algorithm itself. It is quite usual to find keys not securely stored or not cleared from 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 to 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.(For ex, avoid system wide blanket (sudo) permissions for process)
- Develop attitude of “reluctance” to trust information from unknown sources/parties.
- Avoid security by obscurity, If OpenSource components are being used, be prompt in patching 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 in across all layers of software, phases of development (design , development , testing, deployment , maintenance) , multiple domains (software , hardware , firmware) and should be every designers responsibility, just not confining to few individuals, teams or ranks.

---

12 REFERENCES

1. Bleichenbacher D (1998) Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1. In: Krawczyk H (ed) Advances in cryptology – CRYPTO'98. Lecture notes in computer science, vol 1462. Springer, Berlin, pp 1–12
2. Biryukov, Alex. "Chosen Ciphertext Attack." Encyclopedia of Cryptography and Security, pp. 205–205., doi:10.1007/978-1-4419-5906-5_556
3. "Known Ciphertext Attack." Tech-FAQ, www.tech-faq.com/known-ciphertext-attack.html.
4. "Adaptive Chosen-Ciphertext Attacks." Adaptive Chosen-Ciphertext Attacks - WS-Attacks, <www.ws-attacks.org/Adaptive_Chosen-Ciphertext_Attacks>
5. Ross Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems. The first edition (2001): http://www.cl.cam.ac.uk/~rja14/book.html
6. Liska, Allan. The Practice of Network Security: Deployment Strategies for Production Environments. Prentice Hall PTR, 2003.
7. http://homes.cerias.purdue.edu/~pmeunier/aboutme/classes_vulnerabilities.pdf
8. "The CLASP Application Security Process." cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf.< https://cwe.mitre.org/documents/sources/TheCLASPApplicationSecurityProcess.pdf>
9. "Common Weakness Enumeration." CWE - Sources, Office of Cybersecurity and Communications (CS&C), U.S. Department of Homeland Security, cwe.mitre.org/about/sources.html.https://cwe.mitre.org/about/sources.html
10. Common Attack Pattern Enumeration and Classification — CAPEC™, MItre, makingsecuritymeasurable.mitre.org/docs/cpe-intro-handout.pdf.

11. “What Is Common Vulnerabilities and Exposures (CVE)? - Definition from WhatIs.com.” SearchFinancialSecurity, TechTarget, Apr. 2015, search-financialsecurity.techtarget.com/definition/Common-Vulnerabilities-and-Exposures.
12. Aarts, Maurice. “Hardware Attacks Tamper Resistance, Tamper Response and Tamper Evidence.” <maurice.aarts.info/papers/tamper_resistance_evidence.pdf.>
13. Grand, Joe. “Practical Embedded Security.” <https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-grand/grand_embedded_security_US04.pdf>
14. UChenick, Gordon M. “Multiple Independent Levels of Safety & Security (MILS): High Assurance Architecture.” 02-2UChenick, Objective Interface, www.omg.org/news/meetings/workshops/RT_2005/02-2_Uchenick.
15. Alves-Foss, Jim & Oman, Paul & Taylor, Carol & Harrison, Scott. (2006). The MILS architecture for high-assurance embedded systems. IJES. 2. 239-247. 10.1504/IJES.2006.014859. Available from: https://www.researchgate.net/publication/220309643_The_MILS_architecture_for_high-assurance_embedded_systems [accessed Sep 20, 2017].
16. Alves-Foss, Jim & Taylor, Carol & Oman, Paul. (2004). A Multi-Layered Approach to Security in High Assurance Systems.. Proceedings of the Hawaii International Conference on System Sciences. 37. . 10.1109/HICSS.2004.1265709. Available from: https://www.researchgate.net/publication/221180631_A_Multi-Layered_Approach_to_Security_in_High_Assurance_Systems [accessed Sep 20, 2017].
17. A Root of Trust for Measurement - Security. (n.d.). Retrieved March 19, 2017, from http://security.hsr.ch/mse/projects/2011_Root_of_Trust_for_Measurement.pdf
18. 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 Lecture Notes in Computer Science, 159-178. doi:10.1007/978-3-642-30921-2_10

19. K. Koistiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, new, borrowed, blue—a perspective on the evolution of mobile platform security architectures. In Proceedings of the first ACM conference on data and application security and privacy (CODASPY), 2011
20. Aalto University, Asokan. "Trusted Execution Environments - asokan.org." www.asokan.org. N.p., n.d. Web. 20 Mar. 2017. <http://asokan.org/asokan/Padova2014/tutorial-mobileplatsec.pdf>.
21. J. González and P. Bonnet. Towards an open framework leveraging a trusted execution environment. In Cyberspace Safety and Security. Springer, 2013.
22. González, Javier, Michael Hölzl, Peter Riedl, Philippe Bonnet, and René Mayrhofer. "A Practical Hardware-Assisted Approach to Customize Trusted Boot for Mobile Devices." Lecture Notes in Computer Science Information Security (2014): 542-54. Web
23. "Overview of Linux Kernel Security Features." Linux.com | The source for Linux information. N.p., 11 July 2013. Web. 25 Mar. 2017.
24. "Security-Enhanced Linux in Android." Android Open Source Project. N.p., n.d. Web. 25 Mar. 2017. <https://source.android.com/security/selinux/>.
25. O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors.
26. M. Kuhn. Smartcard tamper resistance. In Encyclopedia of Cryptography and Security.
27. Gonzalez, Javier. “Operating System Support for Run-Time Security with a Trusted Execution Environment.”
28. Koopman, Philip. “Embedded System Design Issues.” *Embedded System Design Issues”* https://usrs.ece.cmu.edu/~koopman/iccd96/iccd96.pdf
29. Edward A.Lee “What are the Key Challenges in Embedded Software?” < https://pdfs.semanticscholar.org/e80e/f0404b027871fb09226a767b477a228f8452.pdf>
30. Book: By Mark S. Merkow, Jim Breithaupt “Information Security: Principles and Practices, 2nd Edition”
31. Terry Chia “Confidentiality, Integrity, Availability: The three components of the CIA Triad”< http://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/>
32. Matthew Haughn, Stan Gibilisco “confidentiality, integrity, and availability (CIA triad)” < http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA >

33. Bleichenbacher, Daniel. “Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1.” Advances in Cryptology — CRYPTO '98 Lecture Notes in Computer Science, 1998, pp. 1–12., doi:10.1007/bfb0055716.
34. Vega, Ralph de la, and CEO of AT&T’s Mobile and Business Solutions. “3 Key Rules for Cybersecurity: AT&T Mobile Chief.” CNBC, CNBC, 2 Mar. 2015, www.cnbc.com/2015/03/02/3-key-principles-for-cybersecurity-commentary.html
35. Schneier, Bruce. “Attack Trees: Modeling Security Threats.” Dr. Dobb’s Journal, December, 1999.
36. Barnum Sean, and Amit Sethi. “Attack Patterns as a Knowledge Resource for Building Secure Software.” Attack Patterns as a Knowledge Resource for Building Secure Software, capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf.
37. Erin Scott, Knowledge Consulting Group, Threat-driven Software Development Planning: Using CAPEC & CWE to Improve SDLC <http://slideplayer.com/slide/4316286/>

THESIS INITIATION DOCUMENT

Author _____

Customer _____

Customer's contact person and information _____

_____

Title _____

Description _____

_____

_____

_____

Objectives _____

_____

_____

Target schedule _____

_____

_____

_____

_____

Date and signatures _____

_____

_____