IN2120 Information Security

Lecture 8: Computer Security

UiO : Department of Informatics
The Faculty of Mathematics and Natural Sciences

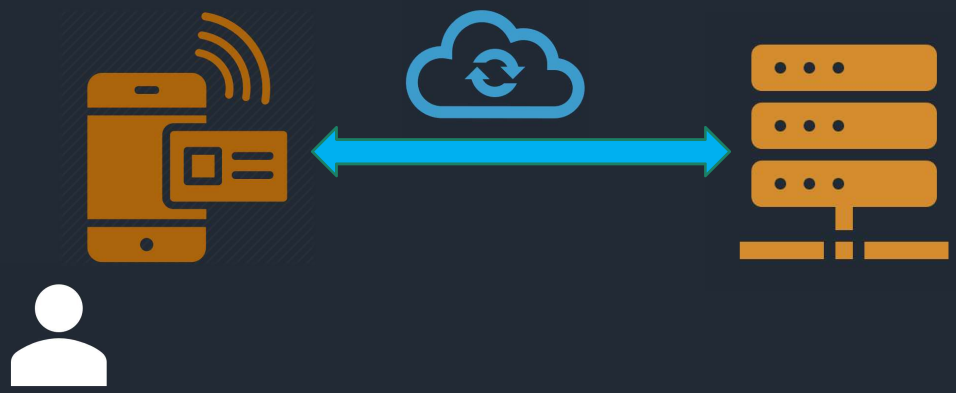
By: Ijlal Loutfi

10.10.2019

Motivation

Let's say that You are asked to develop a basic mobile payment application

What security mechanisms/measures do you implement?



Security Mechanisms-Examples

Application Security-Client side

Only accept strong passwords

Multi-factor authentication

Testing for memory safety vulnerabilities

Application Security-Server side

Hash & Salt passwords

Network Security

End-2-End encrypted traffic

Is this enough to guarantee your customers' security?

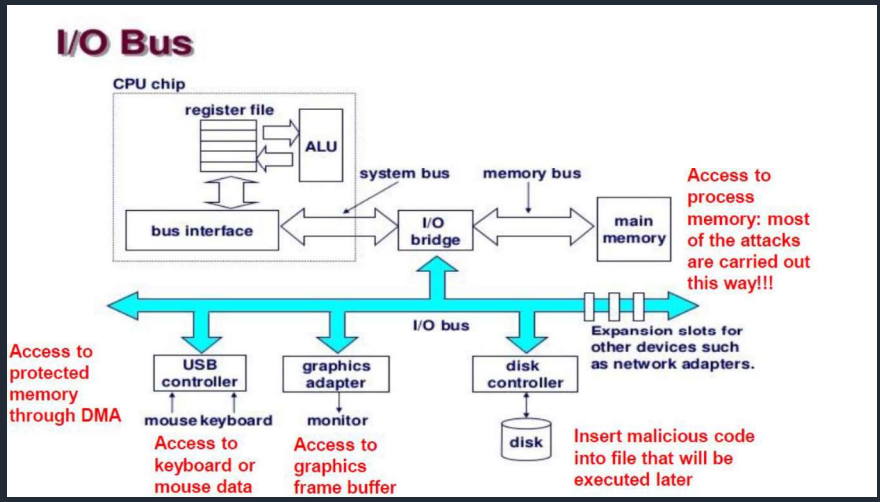
YES?

NO? PROVIDE EXAMPLE OF ATTACKS.

What happens when we run the application on commodity devices?

Examples of Attacks

- A firmware keylogger listening on the password's keystrokes and leaking them
- A browser Malware injecting scripts which modify the transaction details & what is being displayed to you: Overlay attack
- A malicious operating system dumping the system memory, including the network's encryption keys?



Platform Security

System software

- Operating system
- Virtual Machine Manager
- BIOS/UEFI
- Peripheral firmware (e.g.: GPU/NIC/SATA/HDD)
- Execution
- Access to memory
- Access to hardware

System software manages user-level application:

"Computer and Network security, in practice, starts at the hardware and firmware underneath the end points."

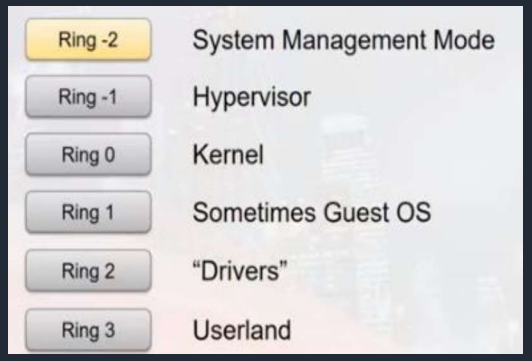
"Their security provides an upper bound for the security of anything built on top"

The Importance of Endpoint Security

"Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench."

(Gene Spafford)

Hierarchy of Security Privileges



Privileged System software gets unrestricted access to ring 3 resources/manages it:

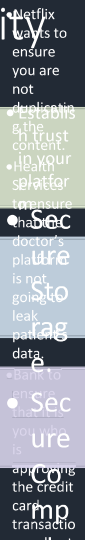
- Memory
- Execution
- Access to underlying hardware

Why we need Platorm Secrutity

Local computation

Remote computation:

You want to establish trust in service provider platforms:



OUR DEVICES ARE TRUSTED

BUT ARE THEY TRUSTWORTHY?

Let's Refocus

The security of user-level applications relies on the platform security

The platform security is dictated by the security of system level software

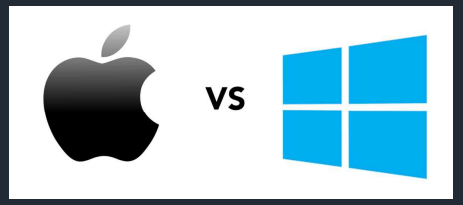
How Secure is System Software?

Assumption

The millions lines of code developed by hundreds of developpers over decades have no:

- Security vulnerabilities
- Backdoors

Reality: Monolithic Operating Systems



- Large Code Base
- Unsafe languages

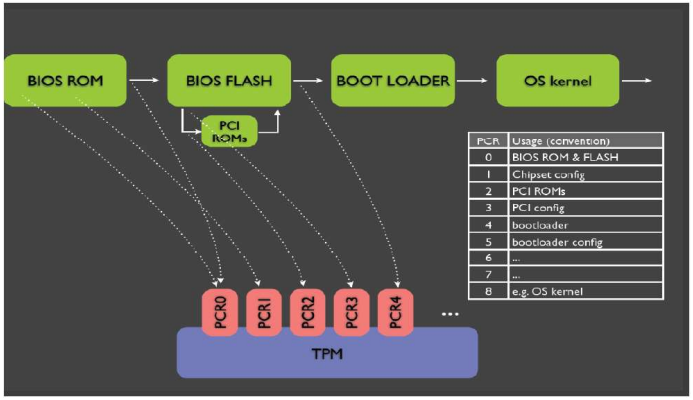
1000 OS LOC → 16 to 75 bugs

What should we do about it?

Platform Security Paradigms

- I need to know what I am running
  - I need to know that what I am runnig is what I expect to be running
- I need to improve the security of what I am running

Trusted Computing



PCR	Usage (convention)
0	BIOS ROM & FLASH
1	Chipset config
2	PCI ROMs
3	PCI config
4	bootloader
5	bootloader config
6	..
7	..
8	e.g. OS kernel

TPM Measurement

A measurement is stored by *extending* a particular PCR:

- The ***extend*** operation is:
- **PCR := SHA(PCR + measurement)**

- **A log of events is stored externally**

TPM Measurement

It is unfeasible to find 2 different measurement values such that when extended returns the same value.

It preservers order in which entities' measurement were extended (extending A then B results in a different value than extending B then A).

The operation allows unlimited number of measurement to be stored

What do we need to implement this securely?

- Secure implementation of the hashing algorithm
- A secure immutable storage for the measurements
- A way to report the final measurement
- An immutable first code
- Software Vs. Hardware

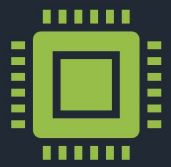
Trusted Computing

We don't trust software, thus:

- To implement such measurement of firmware, some help from hardware is needed for storing and reporting of the measurements reliably, as otherwise malicious software could have forged all the results.

→Hence: Hardware-enabled security Or Trusted Computing

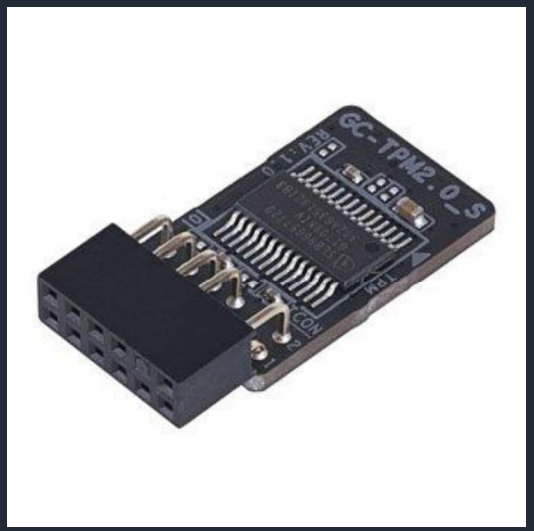
Trusted Computing



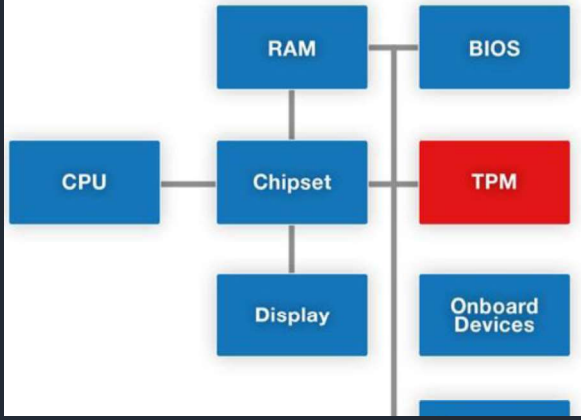
Hardware-rooted mechanism that can allow us to know the state of the platform?

A number that can tell us whether this is a "Good" or "Bad" Platform

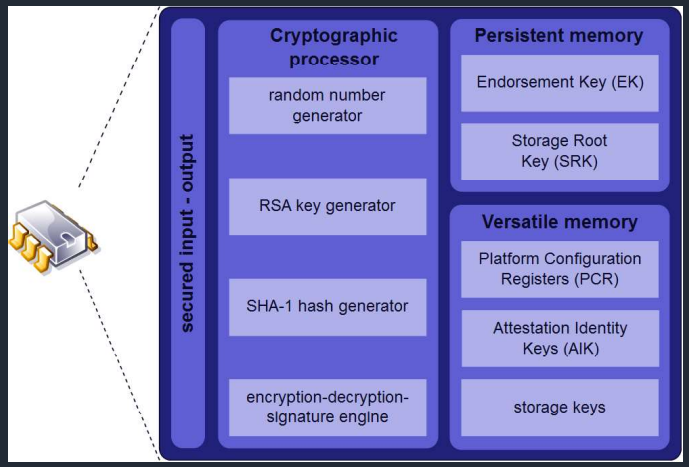
Trusted Platform Module

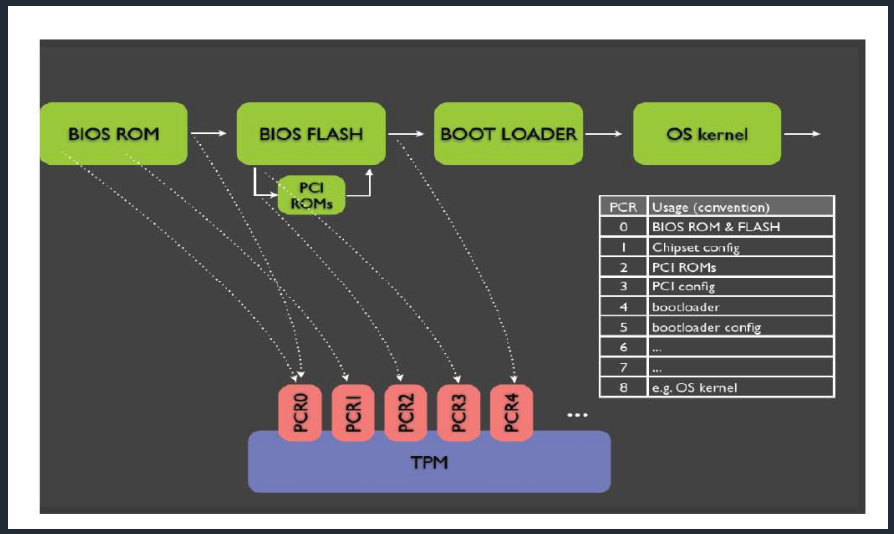


Pervasiveness of the TPM



- The TPM chip sits on the motherboard
- Installed in 2 billion devices per 2015
- Relatively obscure technology for most people





CRTM

- *"The Core Root of Trust for Measurement (CRTM) MUST be an immutable portion of the Host Platform's initialization code that executes up on a Host Platform Reset" TCG*
- Until recently the CRTM was implemented by the BIOS using .
- The normal SPI flash memory. The same flash memory the attacker can usually modify after successfully attacked the BIOS. This has only changed with the latest Intel processors which implement the so called Boot Guard technology.

Does this integrity measurement mechanism prevent an entity from misbehaving or being malicious?

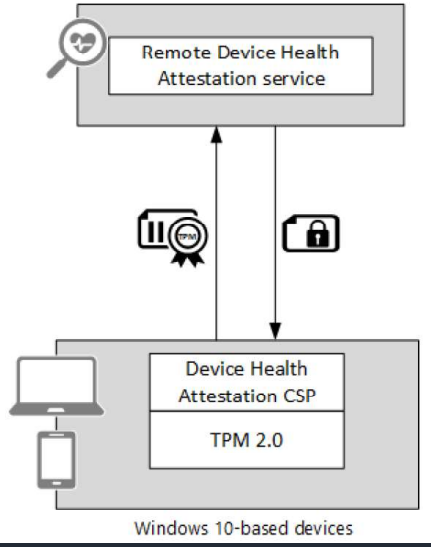
Short answer: No.

So, what does TPM Integrity provide?

An unforgeable record of all the entities that have been loaded. One can choose whether to trust the system based on this record

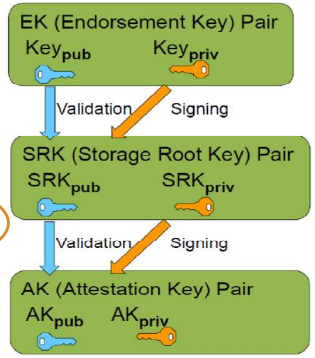
Remote Attestation

- Boot measurement with remote attestation



Key Management

- Endorsement Key (EK)
  - Created once
  - At manufacture time
  - Stored securely in non-volatile memory
- Storage Root Key (SRK)
  - Stored security in non- volatole memory
  - Validated by EK
  - At initiation time
- Attestation Key (AK)
  - Used for remote attestation
  - Validated by SRK
- Custom keys
  - Possible to create additional keys and validate that they are created under SRK.



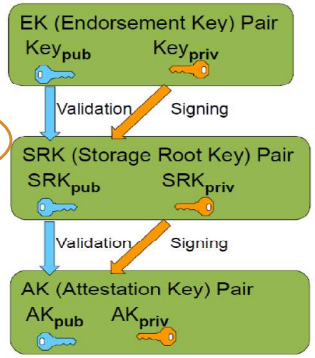
Sealed Storage/Encryption

- Encrypts data so it can be decrypted
  - by a certain machine in given configuration
- Depends on
  - Storage Root Key (SRK) unique to machine
  - Decryption only possible on unique machine
- Can also extend this scheme upward
  - create application key for desired application version running on desired system version
- Supports disk encryption

Encrypt

Key Management

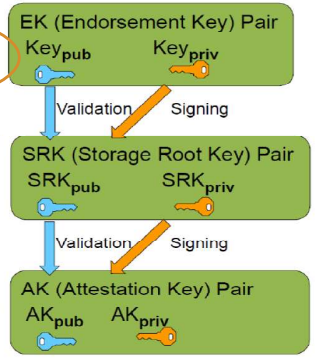
- Endorsement Key (EK)
  - Created once
  - At manufacture time
  - Stored securely in non-volatile memory
- Storage Root Key (SRK)
  - Stored security in non- volatole memory
  - Validated by EK
  - At initiation time
- Attestation Key (AK)
  - Used for remote attestation
  - Validated by SRK
- Custom keys
  - Possible to create additional keys and validate that they are created under SRK.

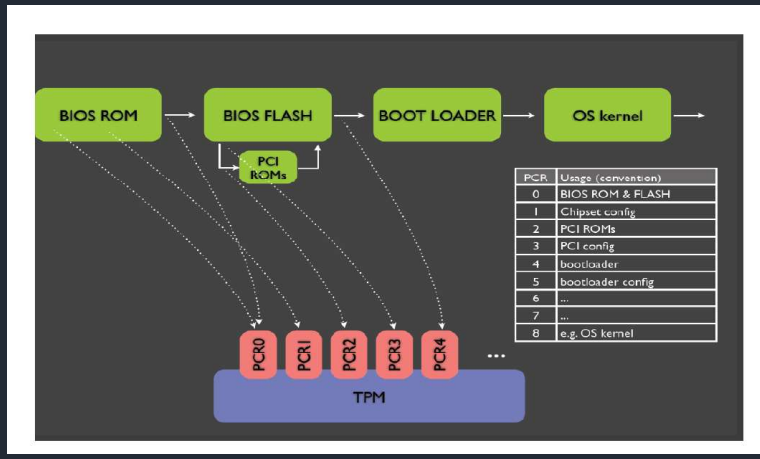


How do I prevent Replay attacks?

Key Management

- Endorsement Key (EK)
  - Created once
  - At manufacture time
  - Stored securely in non-volatile memory
- Storage Root Key (SRK)
  - Stored security in non- volatole memory
  - Validated by EK
  - At initiation time
- Attestation Key (AK)
  - Used for remote attestation
  - Validated by SRK
- Custom keys
  - Possible to create additional keys and validate that they are created under SRK.





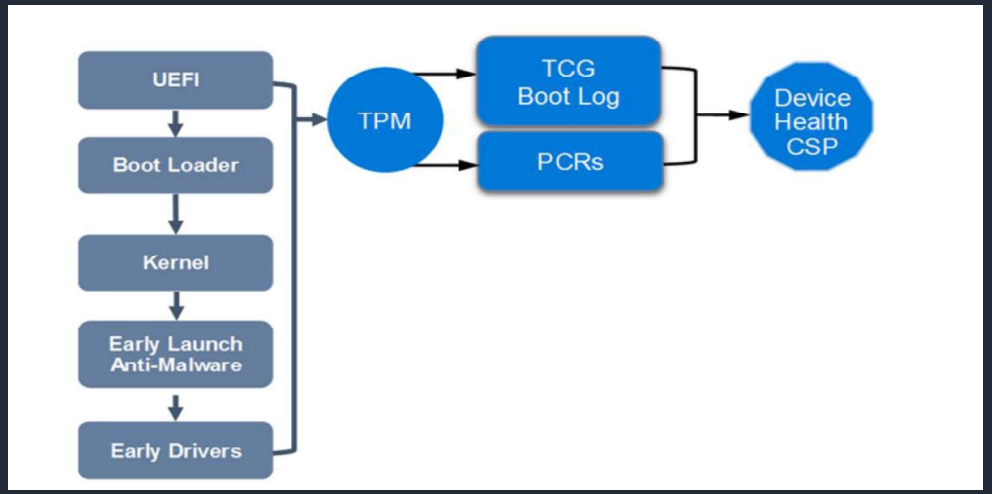
Summary of TPM Services

MEASUREMENT

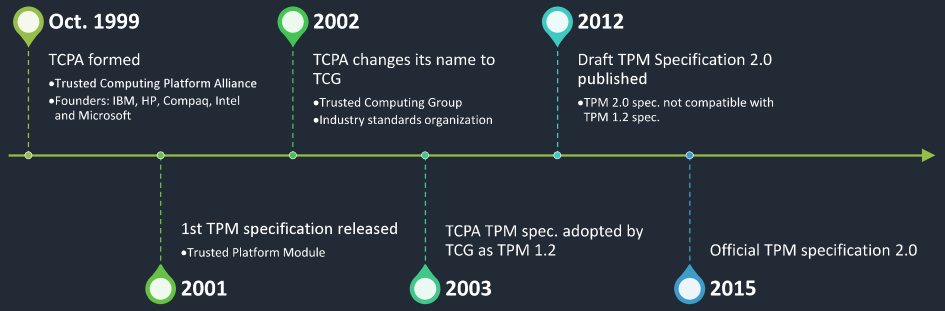
SEALING

REMOTE ATTESTATION

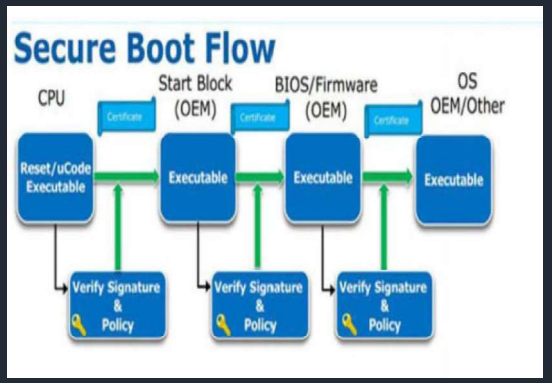
Measured Boot



Trusted Computing GroupTCG History & Evolution



Secure Boot



Secure Boot

Secure Boot

- What is referred to as “UEFI Secure Boot” is not really a new hardware technology, but rather merely a way of writing the BIOS in such a way that it hands down execution only to code, typically OS loader or kernel, which meets certain requirements.
- These requirements are checked by validating if the hash of this next block of code is signed with a select key, itself signed with a key provided by the OEM (called a Platform Key).

Secure Boot and Backdoors

Imagine the BIOS vendor was asked to provide a backdoor to local law enforcement

- All the vendor would have to do would be to sign an additional certificate, thus allowing “alternative” code to be booted on the platform.

Platform Security Paradigms

I need to know what I am running

I need to know that what I am runnig is what I expect to be running

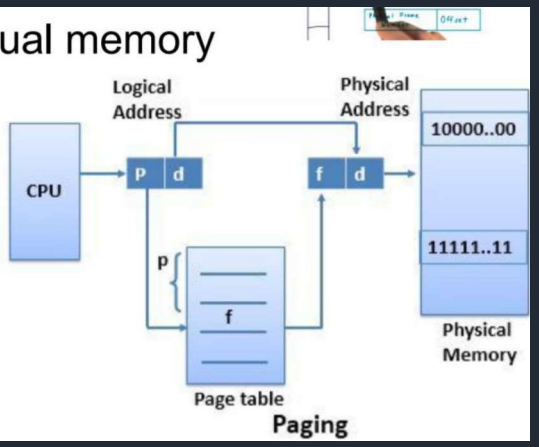
I need to improve the security of what I am running

Virtual Memory

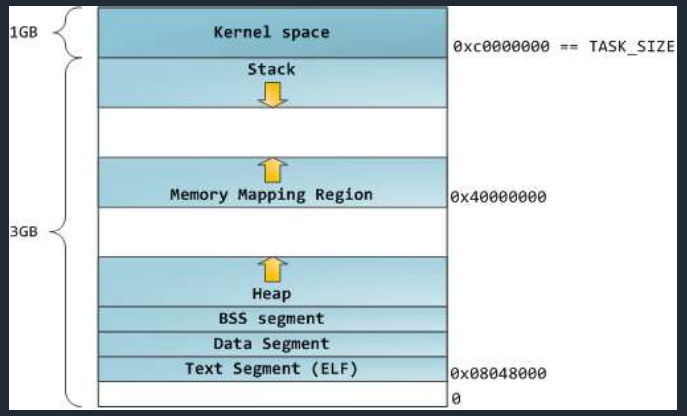
- Division of memory into “pages”
  - 1-N bytes become split at page size boundaries and become
    M = N/page-size
    pages
- We can then refer to memory by the Page Frame Number (PFN) and an offset into the page.
- Common size is 4k (Intel legacy issues)
- The MMU “creates” virtual addresses.

Paging

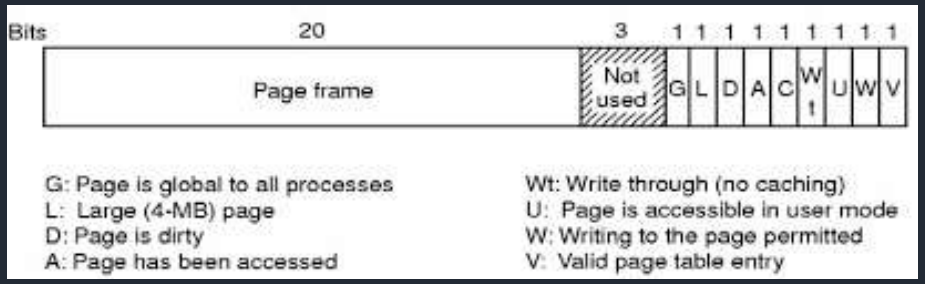
- Processes have virtual memory
- -> PFN
- Page Tables
- Faults
  - Major
  - Minor
- Virtual vs physical



Process Virtual Address Space

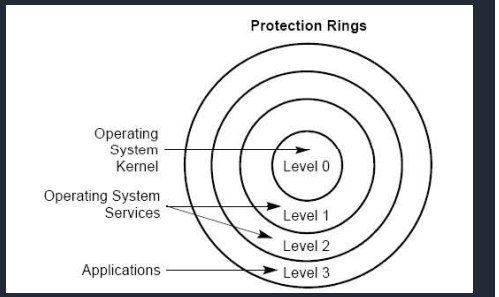


Address privilege levels



CPU Access Modes/Rings/Privilege Levels

- Hierarchic security levels were introduced in X86 CPU architecture in 1985 (Intel 80386)
- 4 ordered privilege levels
  - Ring 0: highest
  - Ring 3: lowest
  - Intended usage → see diagram:



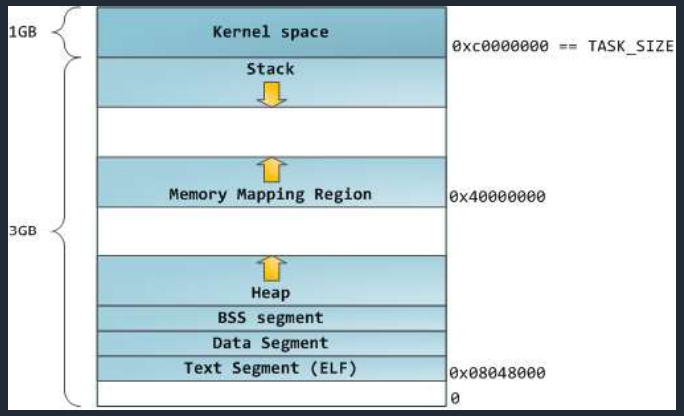
What happened to rings 1 & 2 ?

- ... it eventually became clear that the hierarchical protection that rings provided did not closely match the requirements of the system programmer and gave little or no improvement on the simple system of having two modes only. Rings of protection lent themselves to efficient implementation in hardware, but there was little else to be said for them. [...]. This again proved a blind alley...
- Maurice Wilkes (1994)

Controlled Invocation

- The user process executes code in specific code segments.
- Each code segment has an associated mode which dictates the privilege level the code executes under.
- Simply setting the mode of user process code to Kernel would give kernel-privilege to user process without any control of what the process actually does. Bad idea!
- Instead, the CPU allows the user process to call kernel code segments that only execute a predefined set of instructions in kernel mode, and then returns control back to the user-process code segment in user mode.
- We refer to this mechanism as controlled invocation.

What Can Go Wrong?

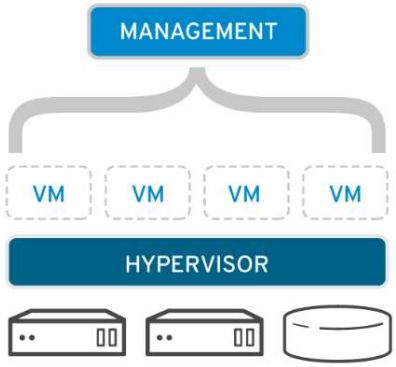


Hardware supported Data Execution Prevention

- Mark all memory locations as non-executable unless the location explicitly contains executable code.
- Relies on processor hardware to mark memory with an attribute that indicates that code should not be executed from that memory.
- Processors that support hardware-enforced DEP are capable of raising an exception when code is executed from a memory location where it should not be executed.

Other CPU Modes

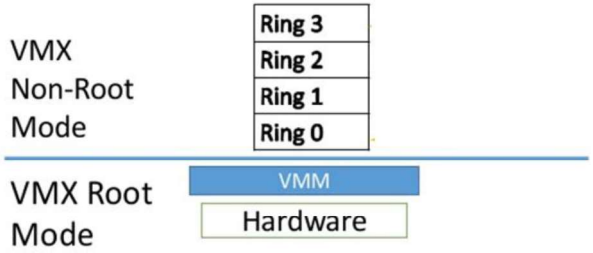
Virtualization



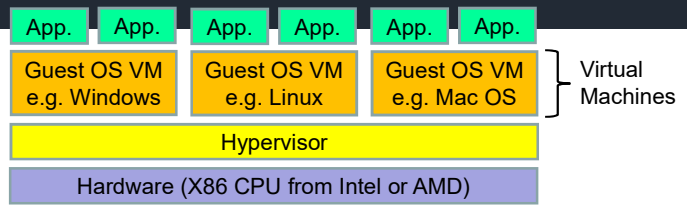
Hardware-assisted virtualization

Hardware Support – Bird's-eye View

- In 2006, Intel introduced VT-x

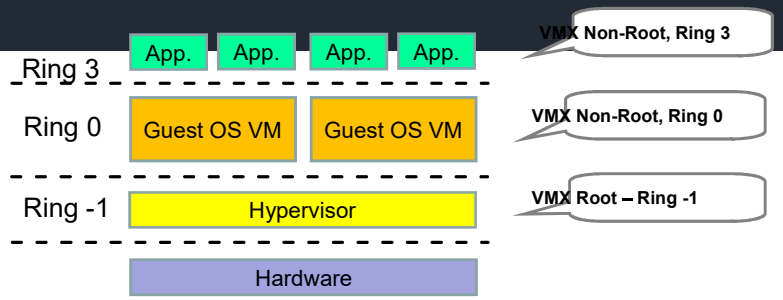


Type 1 VM Architecture (native)



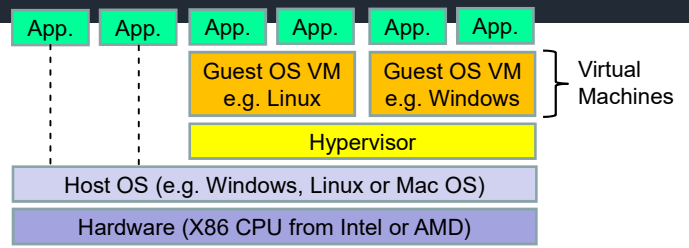
- No host OS
- Hypervisor runs directly on hardware
- High performance
- Traditionally limited GUI, but is improved in modern versions
- HW support can be an issue

Type 1 VM Architecture Ring Allocation



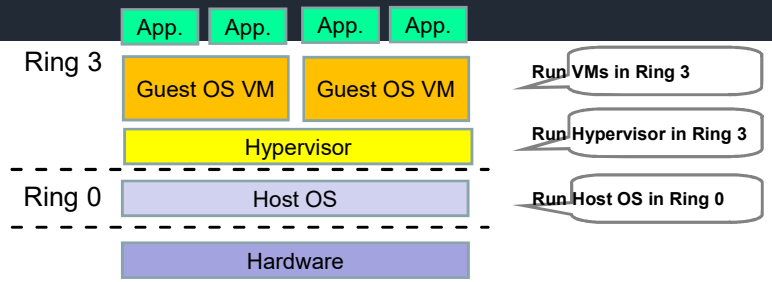
- Guest OS VMs are less privileged than the hypervisor.
- Hypervisor is well protected from the VMs.
- Good performance and good security !

Type 2 VM Architecture (hosted)



- Hypervisor runs on top of host OS
- Performance penalty, because hardware access goes through 2 OSs
- Traditionally good GUI
- Traditionally good HW support, because host OS drivers available

Type 2 VM Architecture Ring Allocation



- Guest OS VMs run in Ring 3.
- Guest OS VMs call privileged instructions that are forbidden in Ring 3.
- Forbidden instructions cause exceptions that are handled by interrupt/exception handler to be executed.
- Slow performance !

Platform Virtualisation Products

Hypervisor types

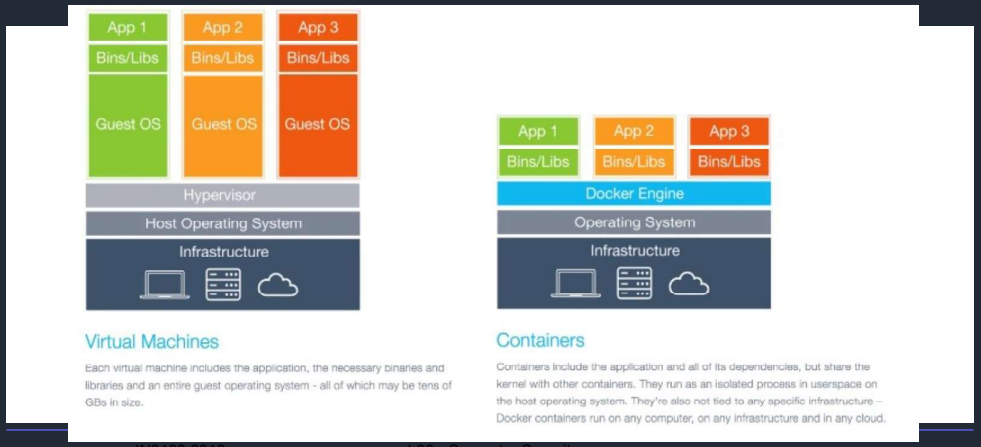
Type 1 (Native)

- Microsoft Hyper-V
- VMWare ESX
- Citrix XenServer (Xen)
- ORACLE VM Server (Xen)
- Amazon EC2 (Xen)
- IBM System Z Hypervisor

Type 2 (Hosted)

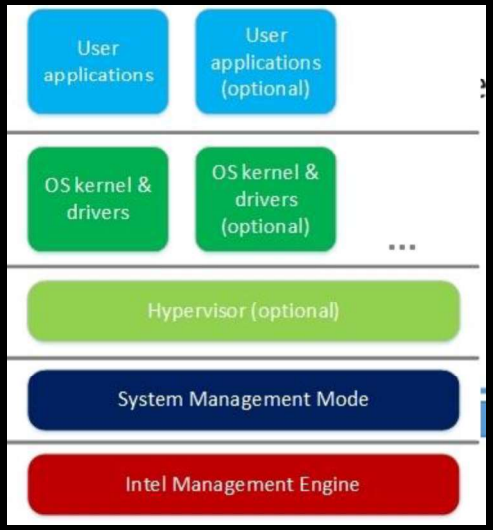
- Microsoft Virtual PC
- VMWare Workstation
- Citrix XenClient
- VirtualBox

Operating system level virtualization



More CPU Modes

All the Modes?



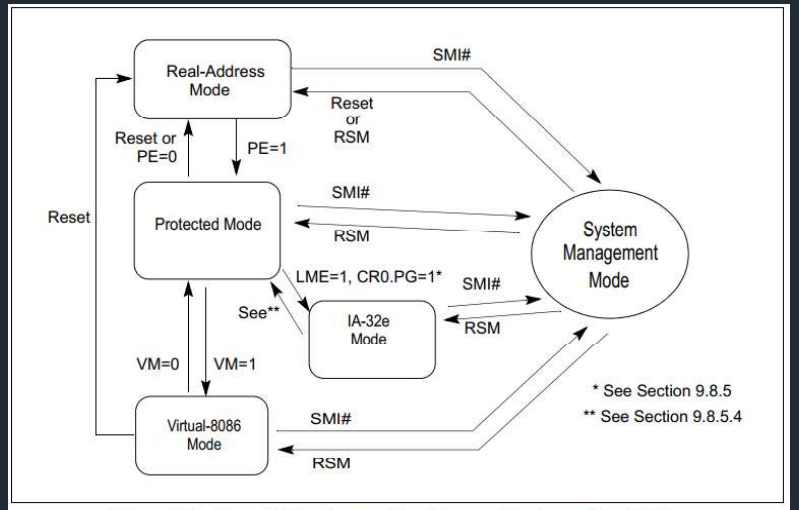


Figure 2-3. Transitions Among the Processor's Operating Modes

Platform Security Paradigms

I need to know what I am running

I need to know that what I am runnig is what I expect to be running

I need to improve the security of what I am running

Isolation

Run the security-sensitive subset of an application within a separate Execution Environments:

- Isolated from the rich vulnerable execution environment.

Trusted Execution Environment

TEE Realization

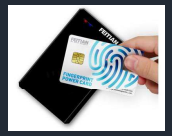
Full Physical Hardware Isolation

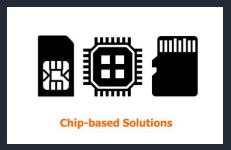
Logical Multiplexed Isolation

Full hardware isolation

- The code runs within a physically isolated processor
- Does not share any context with the untrusted execution environment:

1. Co-processors
2. Smart card
3. Secure elements:





   - Isolated: Universal Integrated Circuit Card (UICC), SD Card
   - Embedded

1. Trusted Platform Modules, TPM

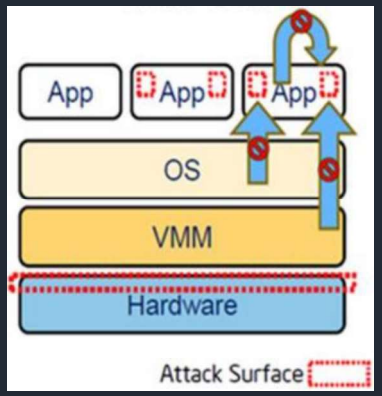
Multiplexed Logical Isolation

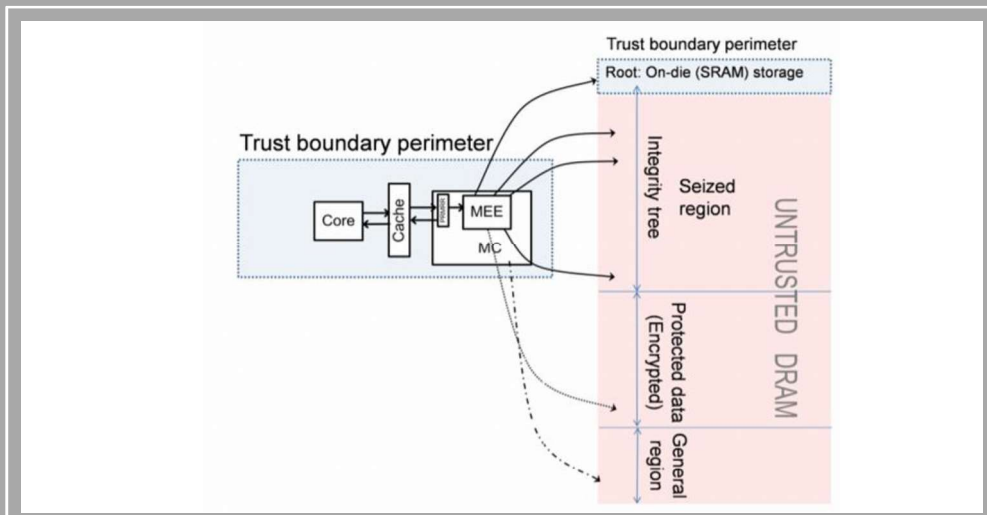
- The security sensitive functions run within the same host commodity processor
- They share its physical execution context.

- Additional CPU-based hardware access control mechanisms are introduced in order to enforce strict logical isolation of resources.

Intel Software Guard Extensions

Security extensions for the X86 Intel processor family.





TEEs are an Old Primitive

Commodity laptop:

1. TPM
2. Embedded secure element,
3. Removable secure elements
4. A system management mode,
5. A CPU based-TEE such as Intel SGX.
6. Intel management engine

Security-sensitive user-level applications **should be leveraging TEE** functionalities to protect the integrity and confidentiality of their code and data.

Unfortunately, this is not the case

User applications still rely on the **default software-based protection** mechanisms provided by **the untrusted commodity system software**

Who Uses TEEs?

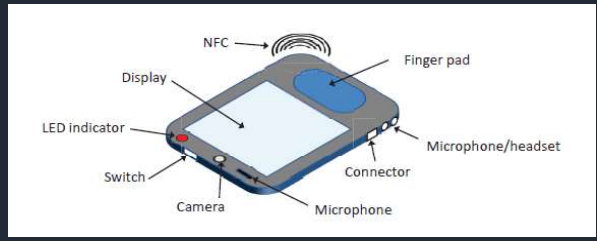
Commodity TEEs are used **by premium stakeholders:**

- Original Equipment Manufacturers (OEMs),
- Mobile Network Operators (MNOs),
- Operating system vendors.

Protect their assets from being compromised by the platform's system software and end-users:

- (MNOs, SIM cards): carrier locking,
- (OEMs, embedded secure elements, TPMs): Boot-loader locking, Digital Rights Management

OffPAD



Commodity hardware TEEs have been designed to **primarily** satisfy the security requirements for **select few premium stakeholders**

They have thus **underprioritized** the requirements **for end-users and other application developers.**

Monolithic closed provisioning ecosystem→ No security multiplexing.

Confidential Computing

- There are ways to secure data at rest and in transit, but you need to protect your data from threats as it's being processed.
- TEE hardware provides a protected container by securing a portion of the processor and memory. Only authorized code is permitted to run and to access data, so code and data are protected against viewing and modification from outside of TEE.

Confidential Computing Consortium

- CCC is a project community at the Linux Foundation dedicated to defining and accelerating the adoption of confidential computing. It will embody open governance and open collaboration that has aided the success of similarly ambitious efforts.
- The effort includes commitments from Alibaba Cloud, Arm, Baidu, Google Cloud, IBM, Intel, Microsoft, Red Hat, Swisscom and Tencent.
- It will establish open source software and standards and provide tools for developers working on securing data in use.