Introduction

Trusted Computing Base (TCB)

- Base components that enforce the fundamental protection mechanisms on a computing system
  - Hardware
  - Firmware
  - Software

- TCB vulnerabilities potentially affect the security of the entire system

TCB by TCSEC (Trusted Computer System Evaluation Criteria, aka Orange Book)

The totality of protection mechanisms within a computing system – including hardware, firmware, and software – the combination of which is responsible for enforcing a computer security policy.

A TCB consists of one or more components that together enforce a unified security policy over a product or system.

The ability of a trusted computing base to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy.

TCB by MITRE

Nibaldi, G. H. *Specification of a trusted computing base (TCB)*. MITRE CORP BEDFORD MA, 1979.

A TCB is a hardware and software access control mechanism that establishes a protection environment to control the sharing of information in computer systems. A TCB is an implementation of a reference monitor, [...], that controls when and how data is accessed.

TCB fundamental components

- CPU security mechanisms
  - Protection rings
  - Virtualization
  - Other mechanisms
    - E.g. Intel SGX enclaves, etc.
- Operating system security model
  - Computational model
  - Access rights and privileges

TEE (Trusted Execution Environment)

- Isolated, secure execution environment

- CPU support
  - ARM TrustZone

- TEE implementations
  - On-board Credentials (Microsoft/Nokia)
  - <t-base (Trustonic)
  - SecuriTEE (Solacia)
  - QSEE (Qualcomm's Secure Execution Environment)
  - SierraTEE (Sierrawave, open-source)
  - OP-TEE (Linaro, open-source)

Can you trust the operating system?

- Can you trust your operating system if you do not control (or trust) the way it booted?

- Secure bootstrapping
  - TPM attestation
  - UEFI secure boot

- Remote attestation
  - TPM attestation

Can you trust the operating system?

- How can you protect your computation if you don’t trust the operating system?

- Intel SGX (Secure Guard eXtensions)
  - Allow user applications to protect code and data from others within enclaves
  - Enclaves are not observable by code running with different privileges
    - OS kernels, hypervisors, etc.

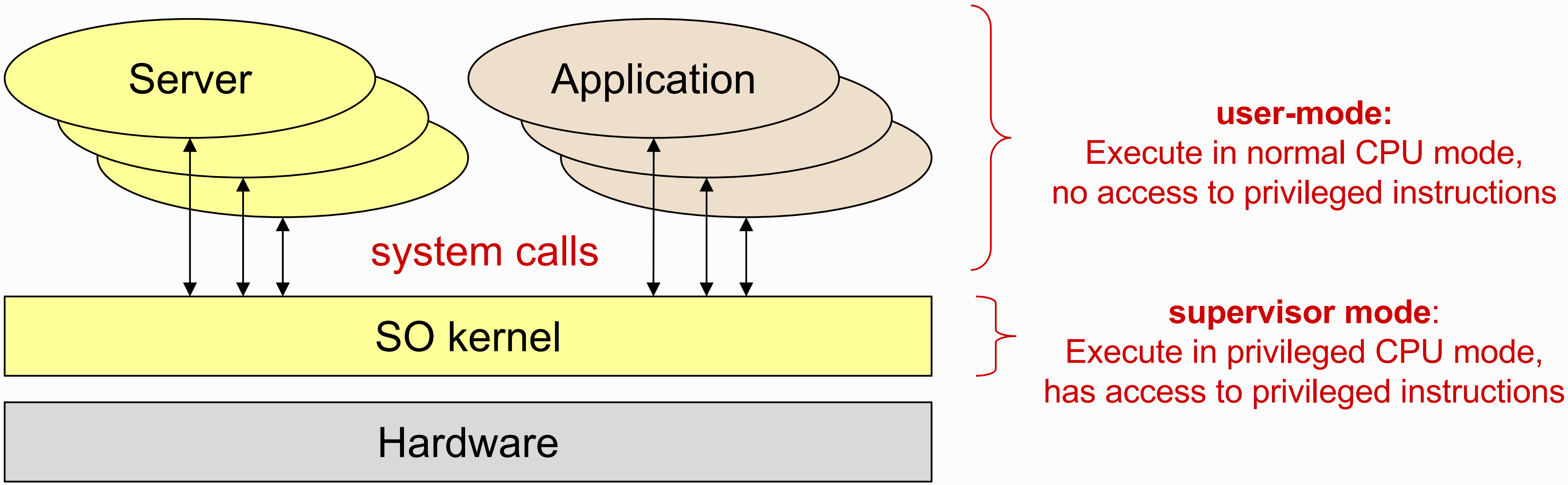
Protection from untrusted code: sandboxes

- Executing applications have a set of privileges and a view over a set of resources

- Sandboxes allow the execution of applications with less privileges or less resources
  - e.g. forbid remote communications
  - e.g. hide the majority of the file system
  - e.g. allow volatile system changes

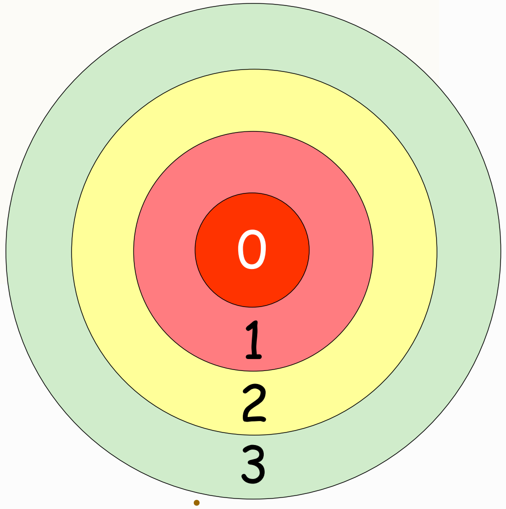
Security in Operating Systems

Operating system



- ▷ Kernel mission
  - Virtualize the hardware
    - Computational model
  - Enforce protection policies and provide protection mechanisms
    - Against involuntary mistakes
    - Against non-authorized activities

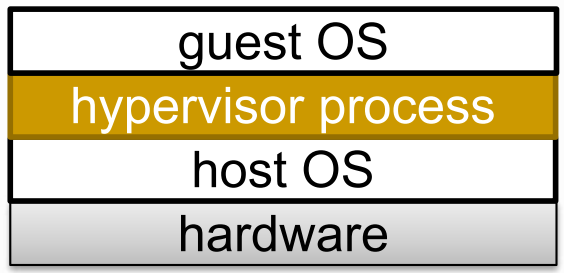
Protection rings



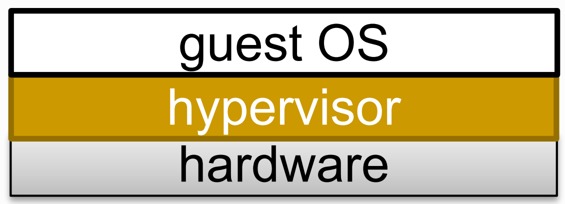
- Different levels of privilege
  - Forming a set of concentric rings
  - Used by CPU's to prevent non-privileged code from running privileged instructions
    - e.g. IN/OUT, TLB manipulation
- Nowadays processors have 4 rings
  - But OS's usually use only two of them
    - 0 (supervisor/kernel mode) and 3 (user-mode)
- Transfer of control between rings requires special gates
  - The ones that are used by system calls (syscalls)

Virtual machines and hypervisors

- Emulation of a particular (virtual) hardware with the existing one (real)



- Hosted virtualization
  - The hypervisor is a process of a given OS (host)
  - The VM runs inside the virtualizer (guest OS)

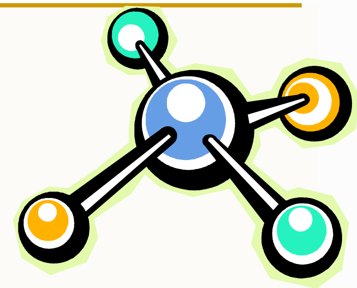


- Bare-metal virtualization
  - The hypervisor runs on top of the host hardware

Execution of virtual machines

- Common approach for hosted virtualization
  - Software-based virtualization
  - Direct execution of guest user-mode code
  - Binary, on-the-fly translation of privileged code (full virtualization)
    - Guest OS kernels remain unchanged
    - No direct access to the host hardware
- Hardware-assisted virtualization (bare-metal)
  - Full virtualization
  - There is a ring -1 below ring 0
    - Hypervisor (or Virtual Machine Monitor, VMM)
  - It can virtualize hardware for many ring 0 kernels
    - No need of binary translation
    - Guest OS's run faster

Computational model



- Set of entities (objects) managed by the OS kernel
  - High-level abstractions supported transparently by low-level mechanisms

- Processes
- User identifiers
  - Users
  - Groups
- Virtual memory
- Files and file systems
  - Directories
  - Files
  - Special files
- Communication channels
  - Pipes
  - Sockets
  - Etc.

- Physical devices
  - Storage
    - Tapes
    - Magnetic disks
    - Optical disks
    - SSD
  - Network interfaces
    - Wired, wireless
  - Human-computer interfaces
    - Keyboards
    - Graphical screens
    - Text consoles
    - Mice
  - Serial/parallel I/O interfaces
    - USB
    - Serial & parallel ports
    - Bluetooth

Computational model: User identifiers

- For the OS kernel a user is a number
  - Established during a login operation
  - User ID (UID)

- All activities are executed on a computer on behalf of a UID
  - The UID allows the kernel to assert what is allowed/denied to processes
  - Linux: UID 0 is omnipotent (root)
    - Administration activities are usually executed with UID 0
  - Windows: concept of privileges
    - For administration, system configuration, etc.
    - There is no unique, well-known identifier for an administrator
    - Administration privileges can be bound to several UIDs
      - Usually through administration groups
      - Administrators, Power Users, Backup Operators
  - Linux: concept of capabilities (similar to privileges)

Computational model: Group identifiers

- Groups also have an identifier
  - A group is a set of users
  - A group can be defined by including other groups
  - Group ID (GID)
- A user can belong to several groups
  - Actual user rights = UID rights + rights of his groups' GIDs
- In Linux all activities are executed on behalf of a set of groups
  - Primary group
    - Typically used for setting file protection
  - Secondary groups

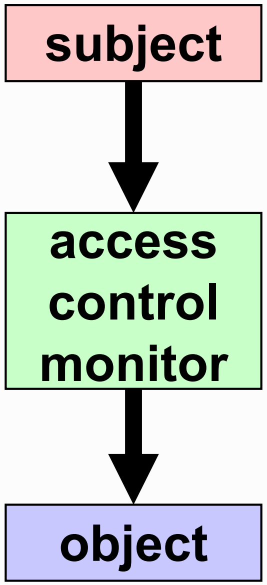
Computational model: Processes

- A process defines the context of an activity
  - For taking security-related decisions
  - For other purposes (e.g. scheduling)

- Security-related context
  - Identity (UID and GIDs)
    - Fundamental for enforcing access control
  - Resources being used
    - Open files
      - Including communication channels
    - Reserved virtual memory areas
    - CPU time used

Access control

- The OS kernel is an access control monitor
  - Controls all interactions of subjects with protected objects
- Objects
  - Hardware
  - Entities of the computational model
- Subjects
  - Usually local processes
    - Through the system call API
    - A system call (or syscall) is not an ordinary function call
  - But also messages from other hosts



Mandatory access controls

- OS kernels have plenty mandatory access control policies
  - They are part of the computational model logic
  - They cannot be overruled not even by administrators
    - Unless they change the OS kernel behavior

- Examples:
  - Kernel runs in CPU privileged modes, user applications run in non-privileged modes
  - Separation of virtual memory areas
  - Inter-process signaling
  - Interpretation of files’ access control protections

Protection with ACLs (Access Control Lists)

- Each object has an ACL
  - It says which subjects can do what
- An ACL can be discretionary or mandatory
  - When mandatory it cannot be modified
  - When discretionary it can be tailored
- An ACL is checked when an activity, on behalf of a subject, wants to manipulate the object
  - If the manipulation request is not authorized by the ACL, the access is denied
  - The OS kernel is responsible for enforcing ACL-based protection

Protection with capabilities

- Less common in normal OS kernels
  - Though there are some good examples
- Example: open file descriptors
  - Applications' processes indirectly manipulate (open) files through file descriptors kept by the OS kernel
    - File descriptors are referenced using integer indexes (aka file descriptors for simplicity…)
    - The OS kernel has full control over the contents of open file descriptors
  - Access to open file descriptors can only be granted to other processes through the OS kernel
    - Not really a usual operation, but possible!
  - Changes in the protection of files does not impact existing open file descriptors
    - The access rights are evaluated and memorized when the file is open

Unix file protection ACLs: Fixed-structure, discretionary ACL

- Each file system object has an ACL
  - Binding 3 rights to 3 subjects
  - Only the owner can update the ACL
- Rights: R W X
  - Read (file data) / List directory
  - Write (file data) / create or remove files or subdirectories
  - Execute / use as process' current working directory
- Subjects:
  - An UID (owner)
  - A GID
  - Others

Windows NTFS file protection: Variable-size, discretionary ACLs

- Each file system object has an ACL and a owner
  - 13 types of access rights
  - Variable-size list of subjects
  - Owner can be an UID or a GID
  - Owner has no special rights over the object or its ACL
    - But usually file creators are their initial owners and have Change Permissions rights
- Subjects:
  - Users (UIDs)
  - Groups (GIDs)
    - The group "Everyone" stands for anybody
- Access rights:

File	Directory (folder)
Read (data)	List (files / folders)
Write (data)	Create (files)
Append (data)	Create (folders)
Execute	Traverse
Delete (file)	Delete (folder)
	Delete (files and subfolders)
Read attributes / extended attributes	
Write attributes / extended attributes	
Read permissions	
Change permissions	
Take ownership	

Unix file protection ACLs: Special protection bits

- Set-UID bit

```
creator:Pictures$ ls -la /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 22  2019 /usr/bin/passwd
```

  - Is used to change the UID of processes executing the file

- Set-GID bit

```
creator:Pictures$ ls -la /usr/bin/at
-rwsr-sr-x 1 daemon daemon 51464 Feb 20  2018 /usr/bin/at
```

  - Is used to change the GID of processes executing the file

- Sticky bit

```
creator:Pictures$ ls -la /tmp
total 108
drwxrwxrwt 25 root    root    4096 Dec 15 13:12 .
```

  - Hint to keep the file/directory as much as possible in memory cache

Privilege elevation: Set-UID mechanism

- It is used to change the UID of a process running a program stored on a Set-UID file
  - If a program file is owned by UID X and the set-UID bit of its ACL is set, then it will be executed in a process with UID X
    - Independently of the UID of the subject that executed the program

- Used to allow normal users to execute privileged tasks encapsulated in administration programs
  - Change the user's password (passwd)
  - Change to super-user mode (su, sudo)
  - Mount devices (mount)

Privilege elevation: Set-UID mechanism (cont.)

- Effective UID / Real UID
  - Real UID is the UID of the process creator
    - App launcher
  - Effective UID is the UID of the process
    - The one that really matters for defining the rights of the process

- UID change
  - Ordinary application
    - eUID = rUID = UID of process that executed **exec**
    - eUID cannot be changed (unless = 0)
  - Set-UID application
    - eUID = UID of **exec**’d application file, rUID = initial process UID
    - eUID can revert to rUID
  - rUID cannot change

Privilege elevation: Set-UID/Set-GID decision flowchart

- exec (path, ...)
  - File referred by path has Set-UID?
  - Yes
    - ID = path owner
    - Change the process effective UID to ID
  - No
    - Do nothing
  - File referred by path has Set-GID?
  - Yes
    - ID = path GID
    - Change the process GID to ID only
  - No
    - Do nothing

Privilege elevation: sudo mechanism

- Administration by root is not advised
  - One “identity”, many people
  - Who did what?

- Preferable approach
  - Administration role (uid = 0), many users assume it
    - Sudoers
    - Defined by a configuration file used by sudo

- sudo is a Set-UID application with UID = 0
  - Logging can take place on each command ran with sudo

Privilege reduction: chroot mechanism (or jail)

- Used to reduce the visibility of a file system
  - Each process descriptor has a root i-node number
    - From which absolute pathname resolution takes place
  - chroot changes it to an arbitrary directory
    - The process' file system view gets reduced

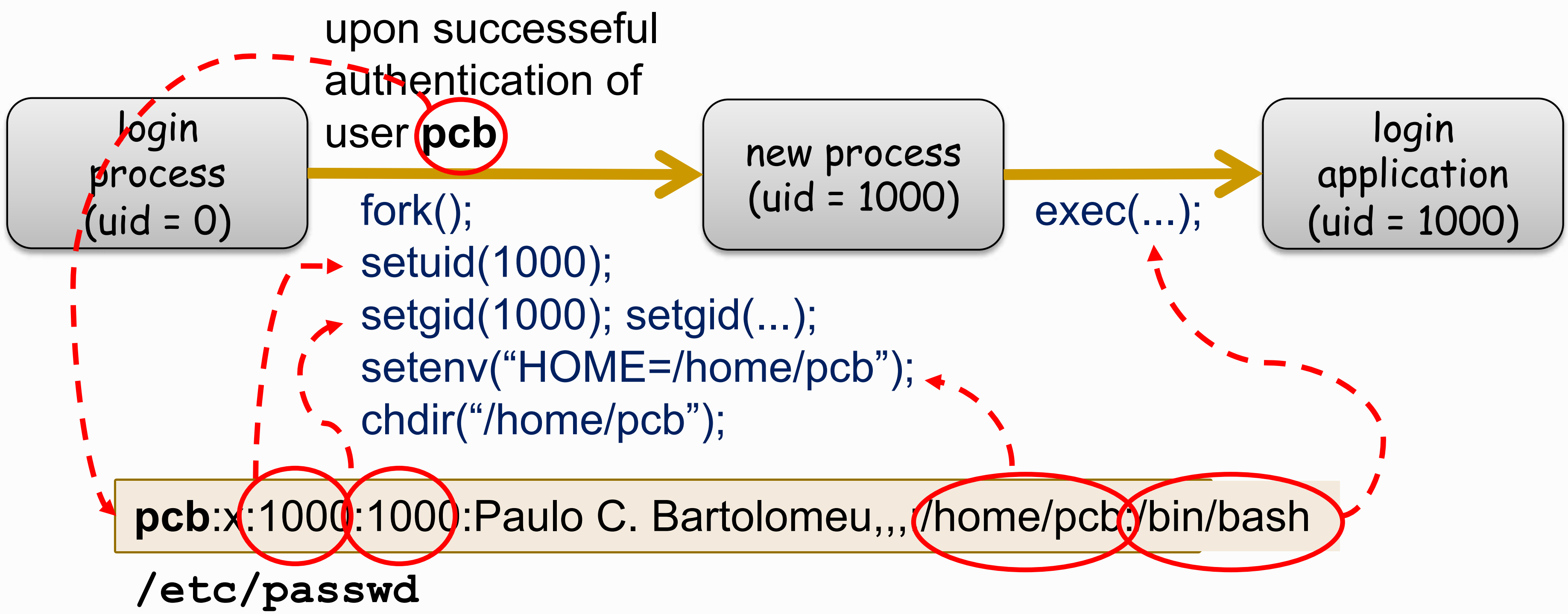
- Used to protect the file system from potentially problematic applications
  - e.g. public servers, downloaded applications
  - But it is not bullet proof!

Linux login: Not an OS kernel operation

- A privileged login application presents an interface for getting users' credentials
  - A username/password pair
  - Biometric data
  - Smartcard and activation PIN
- The login application validates the credentials and fetches the appropriate UID and GIDs for the user
  - And starts an initial user application on a process with those identifiers
    - In a Linux console this application is a shell (sh, bash, csh, tcsh, zsh, etc.)
  - When this process ends the login application reappears
- Thereafter all processes created by the user have its identifiers
  - Inherited through forks

Linux: from login to session processes

- The login process must be a privileged process
  - Has to create processes with arbitrary UID and GIDs
    - The ones of the entity logging in



Login in Linux: Password validation process

- ▷ Username is used to fetch a UID/GID pair from /etc/passwd
  - And a set of additional GIDs in the /etc/group file
- ▷ Supplied password is transformed using a digest function
  - Currently configurable, for creating a new user (/etc/login.defs)
  - Its identification is stored along with the transformed password
- ▷ The result is checked against a value stored in /etc/shadow
  - Indexed again by the username
  - If they match, the user was correctly authenticated

- ▷ File protections
  - /etc/passwd and /etc/group can be read by anyone
    - This is fundamental, for instance, for listing directories (why?)
  - /etc/shadow can only be read by root
    - Protection against dictionary attacks

Intel Software Guard Extensions

What is SGX (Software Guard eXtensions)?

- It is a TEE (Trusted Execution Environment).
- Everything outside the processor chip is not trusted.
- In particular, the BIOS (Basic Input/Output System), the SMM (System Management Mode, ring -2), the ME (Intel Management Engine, ring -3), and the OS (Operating System, ring 0) are **not** trusted.
- The SGX code and data is put inside a special container (a SGX enclave).
- The contents of the enclave are signed (they are loaded from an untrusted source…) and can be attested by an external third party.
- The contents of the enclave is **isolated** from the rest of the system.
- The enclave code runs in **ring 3** (least privileged mode)
- All SGX instructions are implemented in microcode (potential attack vector)

SGX Enclave Memory

- As mentioned in the previous slide, the trust boundary perimeter is the processor chip (core, cache, and memory controller).
- So, the memory of the SGX enclave, when it resides outside of the processor chip (DRAM) is also encrypted.
- The memory encryption key is chosen at random after every processor reset.
- Values read from memory are checked to see it they match what was written (if not the processor hangs).
- This is done on a cache-line granularity (64 bytes) using an memory integrity tree.
  - For details, see https://eprint.iacr.org/2016/204.pdf
- Very small performance penalty if the SGX enclave memory footprint fits in the processor caches.

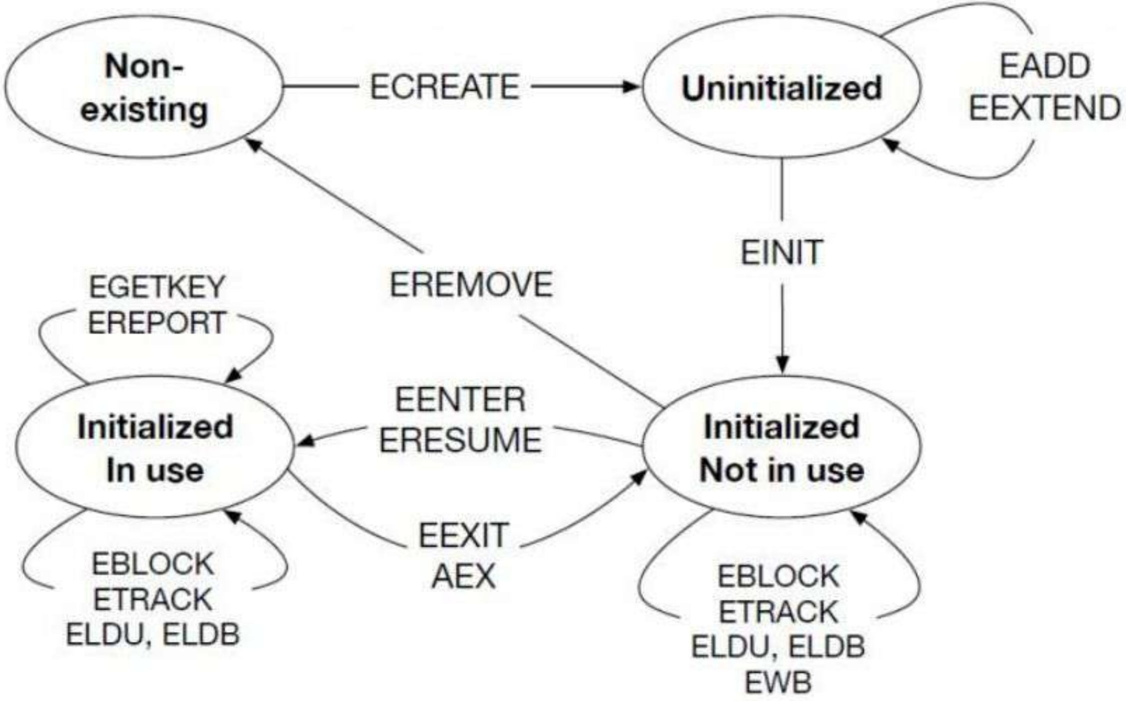
Instructions related to enclaves

- Ring 0 instructions
  - ECREATE, EADD and EINIT are used for Enclave Page Cache (EPC) management --- executed by privileged software such as an OS or a VMM
  - The EPC is an untrusted secure storage area used by the enclave; each 4KiB page has some security attributes that are stored in the Enclave Page Cache Map (EPCM), which is not accessible by software
- Ring 3 instructions
  - EENTER, EEXIT, EGETKEY, EREPORT and ERESUME are used by the user space software to execute functionality within or between enclaves.
- Illegal instructions inside an enclave
  - **`cpuid`**, **`rdtsc`**, input and output instructions, and some others are not allowed
  - **`rdrand`**/**`rdseed`** are allowed and can be virtualized (!?!)

Life cycle of an enclave

https://software.intel.com/content/dam/develop/external/us/en/documents/intelsgx enclavelifecycle.pdf

1. Creation (ECREATE)
2. Loading (EADD, EEXTEND)
3. Initialization (EINIT)
4. Enter/Exit the Enclave (EENTER/EEXIT)
5. Teardown (EREMOVE)



Intel SGX Toolkit (version 2.13) requirements
https://github.com/intel/linux-sgx

- Hardware:
  - Intel 6th Generation Core processor or newer
- 64-bit operating system:
  - Ubuntu 16.04, 18.04 or 20.04 LTS
  - Red Hat 7.6 or 8.2
  - CentOS 8.2
  - Fedora 31
- BIOS support (enabling SGX will reserve up to 128MiB of memory for the exclusive use of SGX enclaves)
- It's also possible to install it on Windows 10 (not covered in these slides)

Intel SGX Toolkit (version 2.13)

- Toolkit components:
  - Intel SGX kernel driver
  - Intel SGX PSW (Platform Software Package)
  - Intel SGX SDK

- Programming languages: **C** and **C++**

- Does my processor and OS support SGX (after BIOS configuration)?
  - `cpuid -1 | grep SGX`
  - If yes:

```
SGX: Software Guard Extensions supported = true
SGX_LC: SGX launch config supported      = true
```

Intel SGX linux driver installation

- Install needed packages:
  - `sudo apt install build-essential ocaml automake autoconf libtool wget python3 libssl-dev dkms`
- Download driver (https://01.org/intel-software-guard-extensions/downloads)
  - `wget https://download.01.org/intel-sgx/sgx-linux/2.13/distro/ubuntu20.04-server/sgx_linux_x64_sdk_2.13.100.4.bin`
- Install the Dynamic kernel Module Support (DKMS) driver:
  - `sudo bash sgx_linux_x64_driver_1.41.bin`
- If you are using secure boot, the kernel module has to be signed, and so this requires generating a new Machine-Owner Key (MOK). Just follow the instructions (a reboot will be required)
- the module location is /lib/modules/5.8.0-48-generic/updates/dkms/intel_sgx.ko and the module name is (obviously) intel_sgx.

Intel SGX PSW installation (on ubuntu)

- Install needed packages:
    - `sudo apt install libssl-dev libcurl4-openssl-dev libprotobuf-dev`
- Run the following commands
    - `echo 'deb [arch=amd64] https://download.01.org/intel-sgx/sgx_repo/ubuntu focal main' | sudo tee /etc/apt/sources.list.d/intel-sgx.list`
    - `wget -qO - https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key | sudo apt-key add -`
    - `sudo apt update`
    - `sudo apt install libsgx-launch libsgx-urts`
    - `sudo apt install libsgx-epid libsgx-urts`

Intel SGX SDK installation (on ubuntu)

- Do the following:
  - `wget https://download.01.org/intel-sgx/latest/linux-latest/distro/ubuntu20.04-server/sgx_linux_x64_sdk_2.13.100.4.bin`
  - `sudo bash sgx_linux_x64_sdk_2.13.100.4.bin`
  - answer no and choose `/opt/intel` as the installation directory
  - copy the contents of `/opt/intel/sgxsdk/environment` to your `.bashrc`
  - `wget https://download.01.org/intel-sgx/latest/linux-latest/as.ld.objdump.gold.r3.tar.gz`
  - `tar xzvf as.ld.objdump.gold.r3.tar.gz external/toolset/ubuntu20.04`
  - `sudo cp -v external/toolset/ubuntu20.04/* /usr/local/bin/`

Intel SGX SDK test (on ubuntu)

- Do the following:
  - `mkdir tmp`
  - `cd tmp`
  - `cp -av /opt/intel/sgxsdk/SampleCode/SampleEnclave .`
  - `cd SampleEnclave`
  - `make SGX_DEBUG=0 SGX_PRERELEASE=1`
  - `./app`
  - `make clean`

- The output should be

```
Checksum(0x0x7ffeac1ee4f0, 100) = 0xfffd4143
Info: executing thread synchronization, please wait...
Info: SampleEnclave successfully returned.
Enter a character before exit ...
```

Guidelines for designing applications using SGX

- Partition the software into trusted and untrusted components
- Use the SGX SDK tools to create the enclave module (a shared object) --- it implements the trusted component of the software
- The enclave code and data **is not secret**
- Secrets has to be loaded in a secure manner (using an ECDH key exchange for example) from a trusted outside source.
- Enclave data has to be **sealed** (encrypted and signed) if it is stored outside of the enclave
- Enclave data has to be **unsealed** if it is loaded into the enclave
- A large memory footprint (in the enclave) will give rise to a significant performance degradation in the memory accesses

Performance Overhead

- Creating an enclave is slow (4KiB pages have to be added one at a time)
- Calling an enclave function from outside of the enclave (an **ecall**) takes about 10k clock cycles
- Calling a non-enclave function from inside the enclave (an **ocall**) takes also about 10k clock cycles

SDK documentation

- Developer reference for Linux OS (PDF)
- Intel developer zone --- Software Guard Extensions (online)

Intel SGX SDK compilation modes

- SGX applications can be compiled in several modes:
  - hardware debug mode (signed with Intel's key, code not optimized)
    - **`SGX_MODE=HW SGX_DEBUG=1 SGC_PRERELEASE=0`**
  - hardware prerelease mode (signed with your key, code is optimized)
    - **`SGX_MODE=HW SGX_DEBUG=1 SGC_PRERELEASE=0`**
  - hardware release mode (signed with your key, code is optimized, cannot be debugged)
    - **`SGX_MODE=HW SGX_DEBUG=0 SGC_PRERELEASE=0`**
    - This mode may require a [commercial licence]
  - simulation mode (in debug mode)
    - **`SGX_MODE=SIM SGX_DEBUG=1`**

SGX SDK Tools

- Edger8r (**`sgx_edger8r`**)
  - Generates "edge" routines (interface between the untrusted application and the enclave) described in a Enclave Description Language (EDL) file
  - Using it on file XYZ.edl produces files XYZ_[tu]t.[hc] where t=trusted, u=untrusted, h=prototypes, and c=functions
- Enclave signing tool (**`sgx_sign`**)
  - supports key management
- Enclave Memory Measurement Tool (**`sgx_emmt`**)
  - Use it to measure how much memory the enclave uses (needed by the Enclave Configuration File)

Writing Enclave Functions

- Describe each function that may be called from outside of the enclave in the .edl file
- The functions can use special versions of the **C/C++** runtime libraries (available in the SDK)
- System calls are not allowed (use **ocalls** instead; **C** linkage only!)
- Not all **C/C++** language features are available
- The sgx_edger8r tool will take care of the details of making the execution flow enter or leave an enclave
  - Pointer arguments in **ecall** functions must point to untrusted memory
  - Pointer arguments in **ocall** functions must point to trusted memory
  - You may need to copy buffer from untrusted memory to trusted memory
- Keep in mind that the enclave will be statically linked

Some available trusted libraries

- **`libsgx_tstdc.a`** (standard **C** library, math, strings, etc.)
- **`libsgx_tcxx.a`** (standard **C++** libraries, STL)
- **`libsgx_tservice.a`** (seal/unseal, EC DH library, etc.)
- **`libsgx_tcrypto.a`**
- **`libsgx_tkey_exchange.a`**
- **`libsgx_tpcl.a`** (Protected Code Loader, for enclave code confidentiality)

Hello world in an enclave (https://github.com/sangfansh/SGX101_sample_code)

- One enclave function, **`printf_helloworld`**, prints the text "Hello World"
- It cannot do this directly, so it calls an untrusted function, **`ocall_printf_string`**, to do the actual printing
- In this example, for the enclave code, the **`printf`** function is re-implemented so that its output goes to a string
- In the untrusted part, the enclave is loaded and the **`printf_helloworld`** function is called

Hello world in an enclave

- List of files
  - **`Makefile`**
  - **`App/App.cpp`**
  - **`App/App.h`**
  - **`Enclave/Enclave.config.xml`**
  - **`Enclave/Enclave.cpp`**
  - **`Enclave/Enclave.edl`**
  - **`Enclave/Enclave.h`**
  - **`Enclave/Enclave.lds`**
  - **`Enclave/Enclave_private.pem`**

- Relevant parts of the **`Makefile`**:
  - **`SGX_SDK ?= /opt/intel/sgxsdk`**
  - **`SGX_MODE ?= HW`**
  - **`SGX_ARCH ?= x64`**
  - **`SGX_DEBUG ?= 1`**
- You can also add
  - **`SGX_PRERELEASE ?= 0`**
- List your untrusted source code files (the application) in the

  `######## App Settings ########`

  section
- List your trusted source code files (the SGX enclave) in the

  `######## Enclave Settings ########`

  section

- Relevant parts of **`App/App.h`**:

```
#include "sgx_error.h"   /* sgx_status_t     */
#include "sgx_eid.h"     /* sgx_enclave_id_t */

# define TOKEN_FILENAME   "enclave.token"
# define ENCLAVE_FILENAME "enclave.signed.so"

extern sgx_enclave_id_t global_eid;  /* global enclave id */
```

- Relevant parts of **`App/App.cpp`**:

```
int initialize_enclave(void) { /*...*/ }

void ocall_print_string(const char *str){ printf("%s",str); }

int SGX_CDECL main(int argc, char *argv[])
{
  if(initialize_enclave() < 0) return -1;
  printf_helloworld(global_eid);
  sgx_destroy_enclave(global_eid);
}
```

- **`Enclave/Enclave_config.xml`**:

```xml
<EnclaveConfiguration>
  <ProdID>0</ProdID>
  <ISVSVN>0</ISVSVN>
  <StackMaxSize>0x40000</StackMaxSize>
  <HeapMaxSize>0x100000</HeapMaxSize>
  <TCSNum>10</TCSNum>
  <TCSPolicy>1</TCSPolicy>
  <DisableDebug>0</DisableDebug>
  <MiscSelect>0</MiscSelect>
  <MiscMask>0xFFFFFFFF</MiscMask>
</EnclaveConfiguration>
```

- Relevant parts of **`Enclave/Enclave.edl`**:

```
enclave {
  trusted {
    public void printf_helloworld();
  };
  untrusted {
    void ocall_print_string([in,string] const char *str);
  };
};
```

- Relevant parts of **`Enclave/Enclave.h`**:

```
#include <stdlib.h>
#include <assert.h>

#if defined(__cplusplus)
extern "C" {
#endif
void printf(const char *fmt, ...);
void printf_helloworld();
#if defined(__cplusplus)
}
#endif
```

- Relevant parts of **`Enclave/Enclave.cpp`**:

```
##include <stdarg.h>
#include <stdio.h>      /* vsnprintf    */
#include "Enclave.h"
#include "Enclave_t.h"  /* print_string */

void printf(const char *fmt, ...) { char buf[BUFSIZ];
  va_list ap; va_start(ap, fmt);
  vsnprintf(buf, BUFSIZ, fmt, ap);
  va_end(ap); ocall_print_string(buf); }

void printf_helloworld() { printf("Hello World\n"); }
```

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

ARM TrustZone

SoC and IP

- SoC (System-on-Chip)
  - Tackles the provisioning of complex and application-specific, multifunctional processors
  - The major functional components of a complete end-product are integrated into a single chip
- Intellectual property (IP) modules
  - Pre-designed, reusable electronic components for hardware chips

SoC structure

- An SoC usually contains
  - Processors
  - IPs
    - Namely security IPs
  - Memory elements (RAM, ROM, etc.)
  - Buses

ARM TrustZone

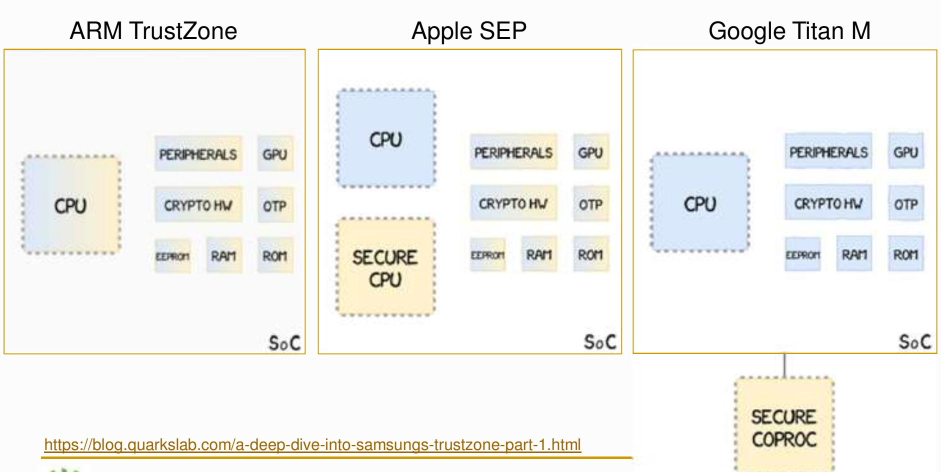
- Set of technologies for packing special security features into a SoC
  - Extra security-related features on processor cores
    - Instructions
    - Bus lines
    - Execution levels
    - Extra logic for dealing with interruptions
  - Security-related IPs

ARM TrustZone: goal

▷ TEE for ARM-powered embedded systems
- Providing hardware-based isolation

▷ It allows to run a trusted system in parallel with the main operation system
- Rich OS
  - Where most applications will run
- Secure (or Trusted) OS
  - Where secure (or trusted) applications will run
  - It can be a simple library, and not a full-fledged OS

ARM TrustZone: Comparison with other similar TEEs



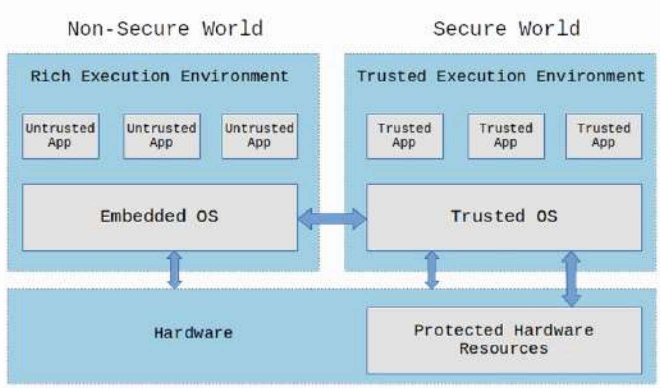
https://blog.quarkslab.com/a-deep-dive-into-samsungs-trustzone-part-1.html

Worlds

- ▷ Isolation is achieved by exploring the same CPU in two different worlds (or states)
  - Normal world → for running the Rich OS
  - Secure world → for running the Secure OS
- ▷ A CPU flag bit defines the current world
  - NS bit of the SCR (Secure Configuration Register)
  - 0 – Secure state
  - 1 – Non-secure state

Protected hardware resources

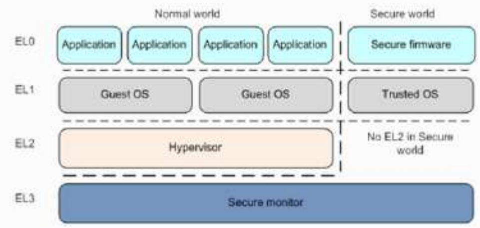
- ▷ Resources accessible only to the Secure OS



https://embeddedbits.org/introduction-to-trusted-execution-environment-tee-arm-trustzone

ARM (v8) exception levels

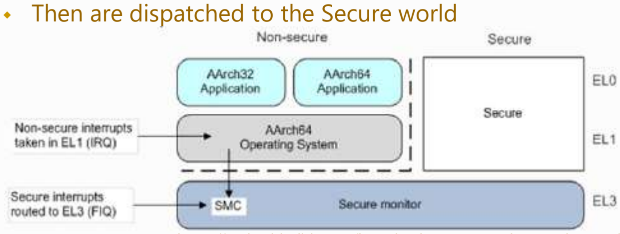
- Similar to run levels
- TrustZone introduces one EL more
  - Secure monitor (EL3)
- Combination of exception levels and states



https://embeddedbits.org/introduction-to-trusted-execution-environment-tee-arm-trustzone

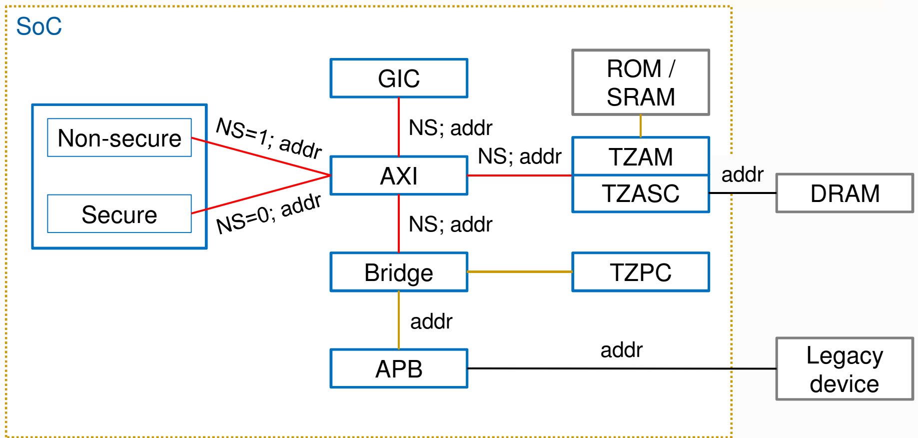
Access to the Secure world

- Calls from the Rich OS
  - SMC (Secure Monitor Call)
  - Typically implemented by Rich OS drivers
- Interrupts from the Secure hardware
  - Must be handled by the Secure OS
- Both enter first in EL3
  - Then are dispatched to the Secure world

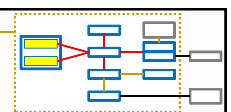


https://embeddedbits.org/introduction-to-trusted-execution-environment-tee-arm-trustzone

Architecture overview



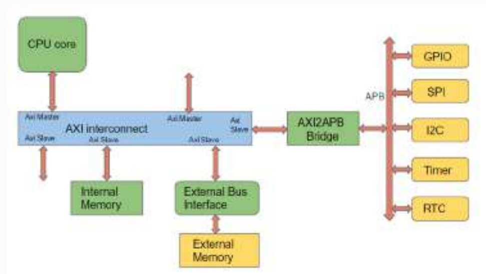
Architectural details: MMU / TLB / Cache Controllers



- 2 separate, virtual MMUs
  - Indexed by NS
- Single TLB
  - But entries keep the value of NS that created them
  - No need to invalidate them when switching between worlds
- The Secure world can still access non-secure memory
  - Extra bit on each entry in the secure translation table
- Single cache
  - Cache lines keep the NS address bit

Architectural details: AXI (Advanced eXtensible Interface)

▷ SoC internal bus



▷ Extra NS line for secure read/write operations

- Non-secure master cannot access a resource marked as secure

https://anysilicon.com/understanding-amba-bus-architecture-protocols

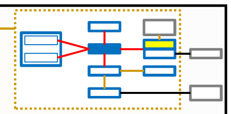
Architectural details: TZASC (TZ Address Space Controller)

▷ Allows a dynamic classification of AXI slave memory-mapped devices as secure or non-secure

- Partitioning of single memory units

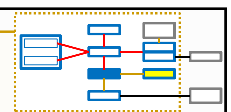
▷ Controlled by the Secure world

Architectural details: TZMA (TZ memory Adapter)



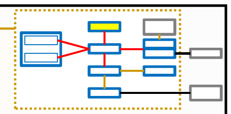
- Keeps a classification of in-SoC memory areas as secure and non-secure
  - ROM or SRAM
- Non-secure accesses cannot access secured memory areas
- Controlled by the Secure world

Architectural details: TZPC (TZ Protection Controller)



- Allows to dynamically set the security of a peripheral connected to the APB (Advanced Peripheral Bus)
  - Protects non-secure access requests to reach peripherals marked as secure
- Controlled by the Secure world

Architectural details: GIC (Generic Interrupt Controller)



- ▷ Classifies interrupts as secure or non-secure
  - Once set, cannot be changed
- ▷ Interrupts can be normal or fast (high-priority)
  - Secure interrupts usually have higher priority
- ▷ Interrupts with a security classification different from the current world force the switching to Monitor (EL3)
- ▷ Controlled by the Secure world

TrustZone bootstrap

- ▷ A TZ-enable ARM SoC boots on the secure world
  - It allows a the Secure world to configure the TZ-related components to enforce a given security policy
- ▷ The configuration data can be
  - Embedded in the SoC ROM
  - Provided by external peripherals and validated with information in SoC ROM
    - e.g. must contain a signature validated with a in-SoC public key