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.

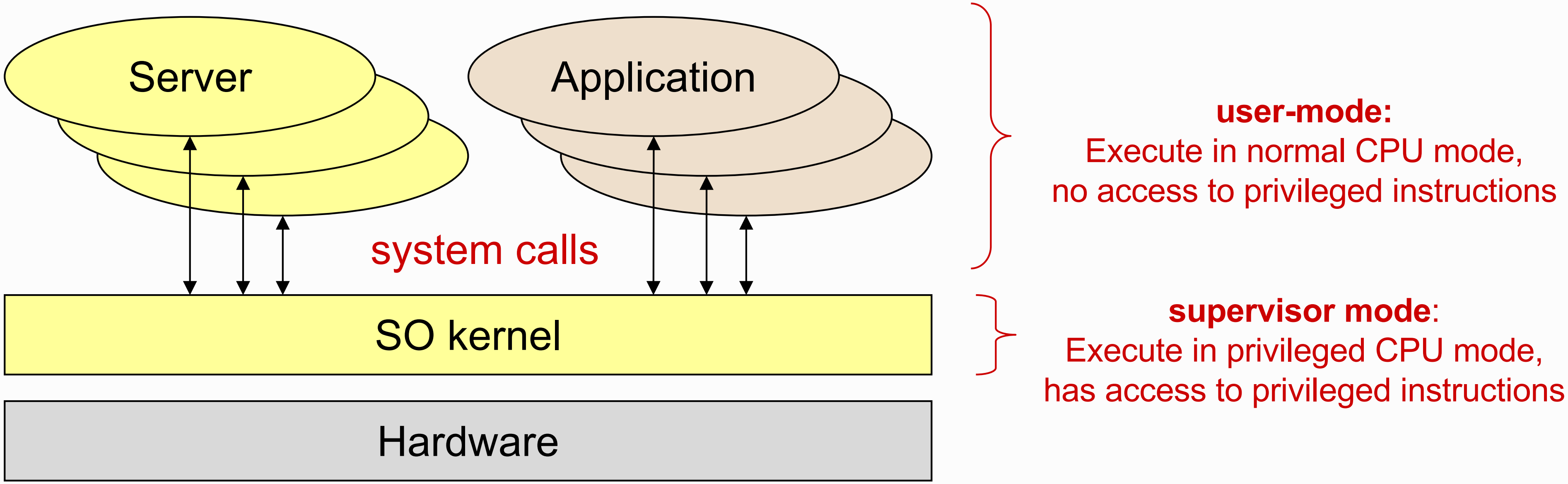
Protection from untrusted code: sandboxes

- 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

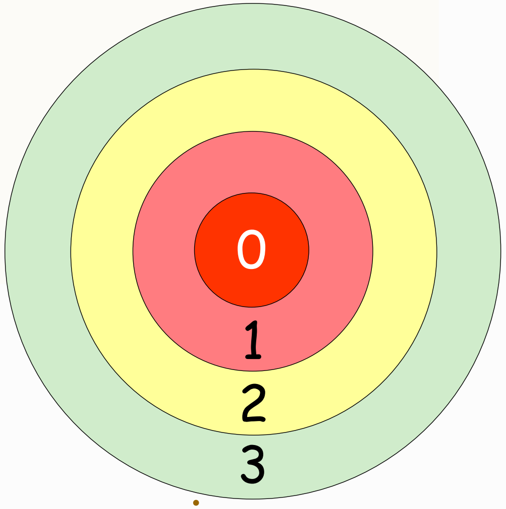
Security in Operating Systems

Operating system



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

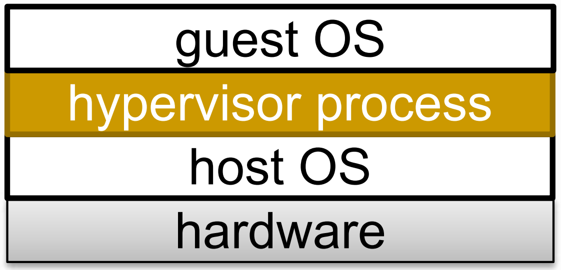
Protection rings



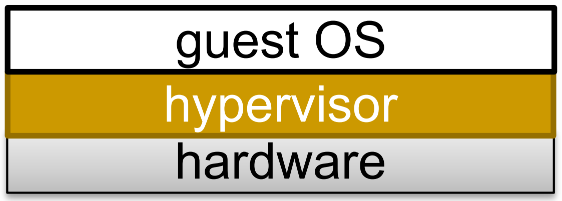
- ▷ Different levels of privilege
  - Forming a set of concentric rings
  - Used by CPU's to prevent non-privileged code from running privileged instructions
    - e.g. IN/OUT, TLB manipulation
- ▷ Nowadays processors have 4 rings
  - But OS's usually use only two of them
    - 0 (supervisor/kernel mode) 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)



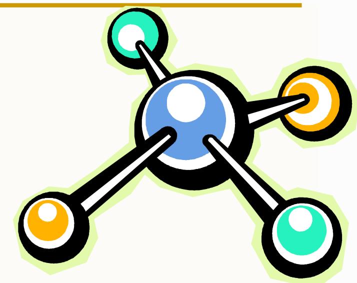
- Bare-metal virtualization
  - The hypervisor runs on top of the host hardware

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

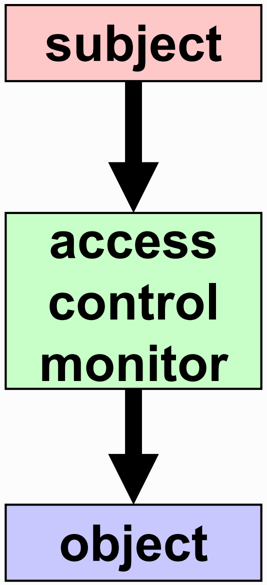
Computational model: Processes

- 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

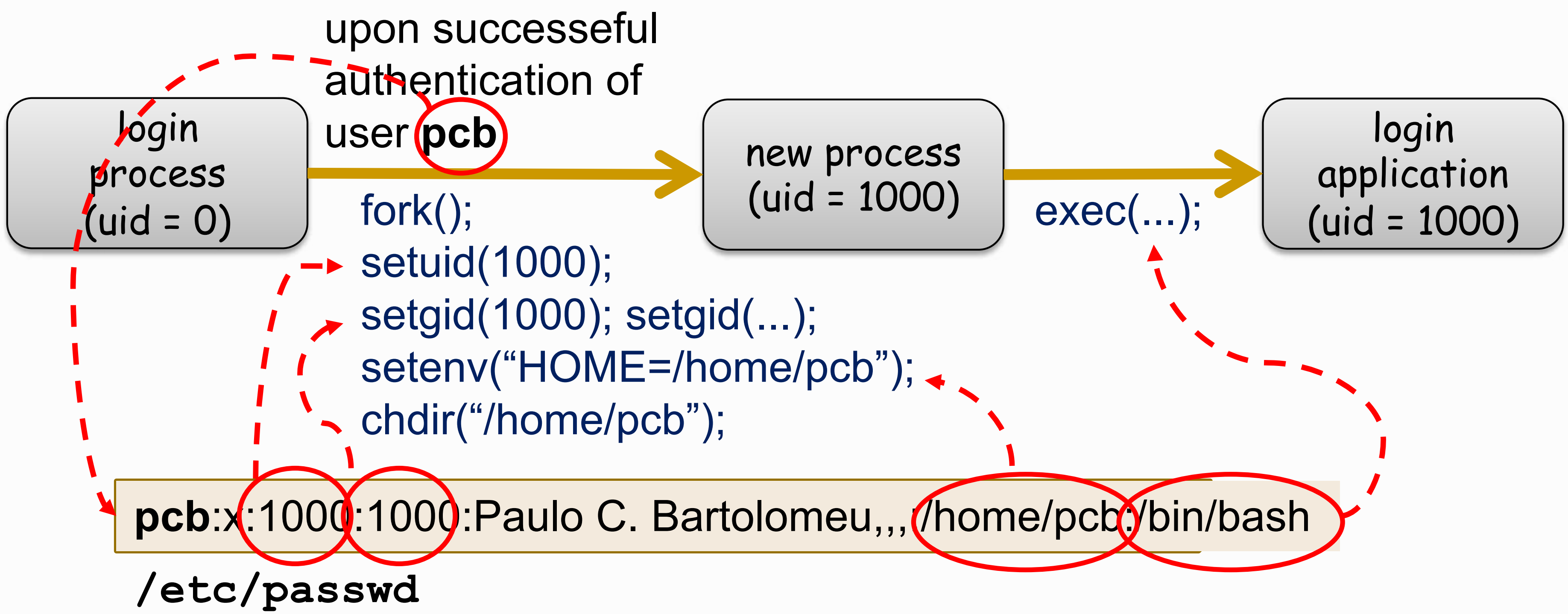
- Used to protect the file system from potentially problematic applications
  - e.g. public servers, downloaded applications
  - But it is not bullet proof!

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