Operating System Security

System Security (CM0625, CM0631) 2025-26
Università Ca’ Foscari Venezia

Riccardo Focardi
secgroup.dais.unive.it

Introduction

Programs may be **vulnerable** and have security **weaknesses**

Operating system security aims at providing **adequate security** guarantees even in presence of vulnerabilities/weaknesses

Idea: security as a **hardening** process

Hardening measures

Australian Signals Directorate (ASD)

White-list approved applications

Patch third-party applications

Patch operating system vulnerabilities, use **latest versions**

Restrict **administrative privileges**

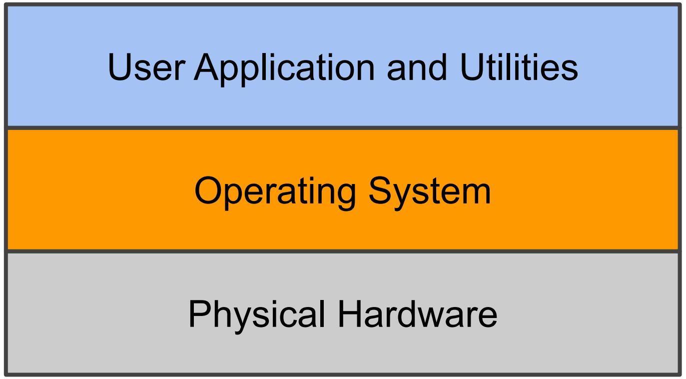
⇒ assist in creating a **defence-in-depth system**

Security layers

Physical hardware: the actual **device**

Operating system: privileged **kernel** code, **APIs**, **services**, interacting with the physical hardware

User applications and utilities: user **programs** interacting with the operating system APIs and services



⇒ Attacks from “below” if layers are not **hardened** so to provide *appropriate security services*

OS security

1. System security planning
2. Installation
3. Trusted code and patching
4. Unnecessary services
5. Access control
6. Additional security controls
7. Application security
8. Logging
9. Backup

System security planning

Aim: maximise **security** while minimizing **costs**

When: from the very **beginning** of deployment (“retrofitting” is difficult and expensive)

Planning based on:

- **purpose** of system, information **type**, security **requirements**
- categories of **users**
- how users **authenticate**
- how **access** is managed
- what access to **other hosts** (and how it is managed)
- who **administer** the system and how (remotely vs. locally)
- what additional security mechanisms are necessary. Ex. **firewalls**, **anti-virus**, **logging**, ...

Installation

Installation: ideally done in an isolated environment with **no incoming** connections

- system might be **vulnerable** in this phase
- **hardening** is done **after** installation

Outgoing connections only towards the necessary (verified) sites

Secure boot: prevent changes in BIOS and limit the boot media to the **trusted** ones

- prevent **malicious hypervisors**
- prevent trivial **bypass** of access control (e.g. boot from external drive to access filesystem)

Cryptographic file systems add a protection layer to stored data

Trusted code and patching

Device drivers: programs with kernel level privileges should be installed with **care**, especially when third party

- might be used to install **malware**

Blue Pill rootkit installed through a **rogue device driver** and run a “thin” hypervisor under Windows Vista

Stuxnet installed **rogue drivers** digitally signed using stolen keys

System should be up to date with all **security patches** installed (one of the ASD hardening measures)

👎 Updates can **introduce instability** so, in systems with **critical availability constraints**, automatic updates are turned off

⇒ For these systems patches should be timely **tested and applied**

Unnecessary services and access control

Remove unnecessary software: if fewer software packages are available, then the risk of vulnerability is reduced

Balance **security** and **usability**

Not installing is better than **removing** or **disabling**: removing does not eliminate everything, attacker might re-enable disabled software

Access control: all modern systems implement **DAC** and, in some cases, **RBAC** or **MAC**

ASD hardening measures suggest to restrict **administrative privileges**

- only **few users**
- use administrative privileges only **when necessary** and **log** any administrative action

Additional security controls

Anti-virus: traditionally on **Windows** systems (preferred target for attackers). Smartphones are more and more targeted

Host-based firewalls, IDS: improve security by filtering **connections** to ports, **blocking** usage of ports by (malicious) processes, monitoring **traffic** and file **integrity**

Whitelisting applications: limiting programs to the whitelisted ones so to prevent execution of **malware** (one of the ASD hardening measures)

NOTE: Not all organizations or all systems will be sufficiently **predictable** to suit this type of control

Security testing: tools to scan for vulnerabilities / weak configurations

Application security

Default data, scripts, or user accounts: should be reviewed, and only retained if required, and suitably **secured**

Example: Web servers often include a number of example scripts, quite a few of which are known to be **insecure**; should be removed unless needed and secured

Access rights: apply **minimum privilege**

Example: a Web server should **not** have **write access** to (most of) the web application files

⇒ In case of a vulnerability, the attacker should not be able to ***deface*** the web application by adding malicious content

Logging

Logging informs about bad things that **already happened**

Crucial for correct **remediation** and **recovery**

What is logged is part of the initial **security planning** phase, depends on

- security requirements
- information sensitivity

Log rotation: logs easily become very large. It is necessary to **compress**, **archive** or **delete** them, once they become too old or too big

Automated vs. manual analysis: manual analysis of big logs is **hard and unreliable**. Automated analysis (e.g. performed by **IDSs**) is preferred to **spot** abnormal activity that can be manually inspected

Backup

Backup: making copies of data at regular intervals, allowing the **recovery** of lost or corrupted data over relatively **short time** periods

Archive: retaining **copies** of data over extended periods of time, in order to meet **legal** and **operational requirements** to access past data

⇒ often linked and managed **together**

Online vs. offline: online backup is easier and cheaper but in case of attack backups/archives might also be **destroyed**

Example: *Distribute.IT* Australian ISP hacked in 2011, **all backups lost**

Local vs. remote: in case of **calamity** (fire, flood, …) local backups would be destroyed

Case studies

1. Linux/Unix

2. Windows

Linux/Unix (1)

System should be up to date: Various automatic tools such as `yum`, `YaST`, `apt`, `apk`, ...

Application/service configuration: Usually in `/etc` folder and in hidden "dot" files such as `.bashrc`

Permissions: `rwx` permissions, ACLs, capabilities, as discussed in the access control lab

User accounts: info in `/etc/passwd`, `/etc/shadow`, `/etc/group`. Authentication through **PAM** (pluggable authentication module)

Users: remove unnecessary users, disable login if not necessary

SUID root programs should be limited. SGID to a privileged group with appropriate permissions is preferred

Linux/Unix (2)

Remote access: tcp wrapper enforces hostname-based **access control** using `/etc/hosts.allow` and `/etc/hosts.deny`

`netfilter` and similar tools (e.g. `pf` in BSD Unix) allow for host-based **firewalling**

Logs: Typically through `syslogd`. `logrotate` can be configured to rotate any logs on the system

`chroot` jail: used to set the root directory of a service so that the rest of the filesystem is not accessible

Example: `/srv/ftp/public`, so that `/srv/ftp/public/etc/myconfigfile` appears as `/etc/myconfigfile`

Note: `root` can break out the jail

Security testing: tools such as Nessus, Tripwire, metasploit and nmap (free)

Linux/Unix (3)

Mandatory Access Control: allows for centralized policies that cannot be changed by users (even root)

Example: a vulnerability in a SUID root service would not give full access to the host. MAC would restrict access to the necessary resources

⇒ Configuration can be complex!

AppArmor and **SELinux** are popular examples of **MAC implementations** in Linux systems

They are usually shipped with a policy only restricting **crucial system processes** and using standard DAC for any other program

👎 **partial** MAC implementation

👍 more **usable**

Windows (1)

System should be up to date: Windows update

Application/service configuration: centralized in the **Registry**, a database of keys and values

Permissions: **ACLs** grant access to **SID** (Security ID) referring to a user or a group. **MAC** for **integrity** (writing): subject's integrity higher that object's

User accounts: **SAM** (Security Account Manager), centralized through **Active Directory**, based on LDAP (Lightweight Directory Access Protocol)

Deny: it is possible to deny specific accesses to users or groups

System wide privileges: for backup, change time, … should be granted with care

Windows (2)

Extra security controls: prevalence of **malware** requires anti-virus solutions (many commercial products available)

Least privilege: administrative rights only use them when required through the User Account Control (**UAC**). **Low Privilege Service Accounts** that may be used for long-lived service processes

Encrypting File System (EFS): protects against attackers with physical access to computers

Network shares: additional security and granularity.
Example: hide any objects that a user is not permitted to read

Security testing: tools such as Nessus, Tripwire, metasploit and nmap (free)

Virtualization

hypervisor: software between the hardware and the **Virtual Machines** (**VMs**), acts as a resource broker

Provides **abstractions** of all physical resources (such as processor, memory, network, and storage)

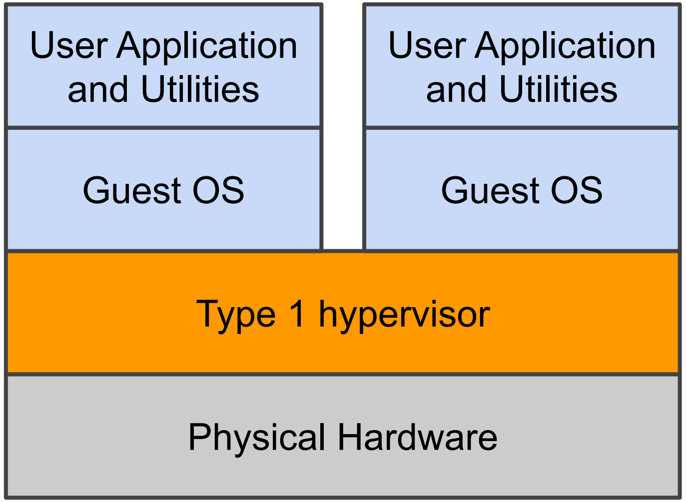
Enables **multiple VMs** to be run on a single physical host

Type 1 hypervisor: native virtualization

Type 1 hypervisor: is loaded as a software layer **directly onto a physical server**

It is called **native virtualization**: the hypervisor can **directly control** the physical resources of the host

Once installed and configured, the server is then capable of supporting virtual machines as **guest OSs**

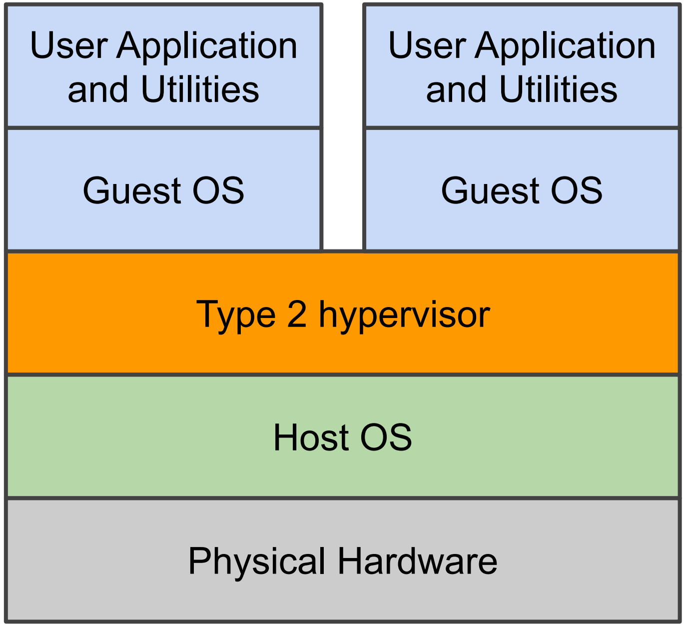


Type 2 hypervisor: hosted virtualization

Type 2 hypervisor: is loaded as a software layer **on a host OS** installed on the physical server

It is called **hosted virtualization**: the hypervisor relies on the host OS to access physical resources

Once installed and configured, the host OS is capable of supporting virtual machines as **guests**



Native vs. hosted virtualization

Performance: native virtualization usually performs **better** that hosted one (no extra host OS underneath!)

Security: native virtualization is usually more **secure** that hosted one

- fewer additional layers to protect
- host OS might be vulnerable
- users of host OS might access VM images

Multiple environments in the same OS: host based virtualization does not require to dedicate the full machine to VMs (typical in **clients**)

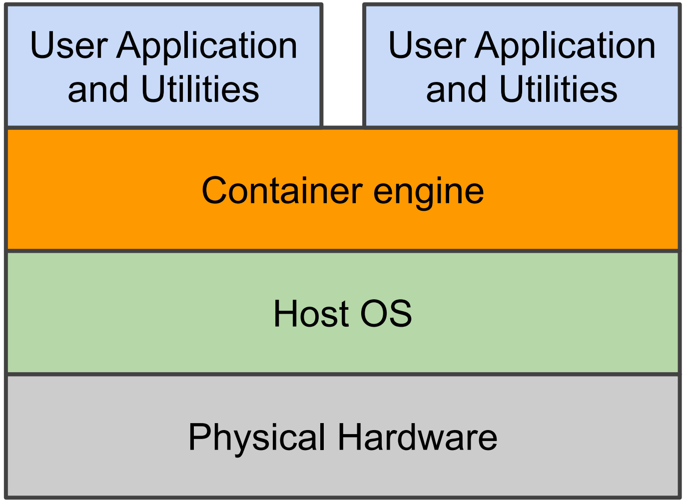
Example: developers that need multiple OSs can use host-based virtualization to run Unix / Linux / Windows on top of **any host OS**

Containers: application virtualization

Virtualization containers: is loaded as a software layer **on a host OS** installed on the physical server

Provide an **isolated environment** for applications, which share the **same OS kernel** (smaller overhead!)

Once installed and configured, the container engine is capable of supporting ***containerized apps***



Virtualization security

VM escape: a vulnerability in the hypervisor might allow VMs and virtualized applications to **access**

- the **hypervisor**
- other **VMs**
- the **host OS**

Host OS attack: vulnerability in host OS would allow to access guest OS **images**

Virtualization allows for separating services into different VMs or container applications

👍 vulnerabilities are **confined** to the VM or container

👎 **vulnerabilities** in the virtualization layers might allow for **taking full control** over the physical server and/or the host OS

Hypervisor and infrastructure security

Secured in a way similar to OS:

- installed in **isolated** environment
- clean **media**
- patched regularly (**automatic updates**)
- **unused services** not installed
- **unused hardware** disconnected

Access: only by **administrators** (locally or on a separate network)

Management traffic: for administration and configuration

Application traffic: for VMs and virtualized applications

Traffic should be ideally **separated**

- different physical **interfaces**
- **VLANs**
- Software Defined Networks (**SDNs**)

Virtual firewall

VM Bastion Host: **separate VM** running Bastion Host services: firewalls, IDS, IPS, …
The VM runs on the hypervisor and monitors (virtual) network interfaces used by VMs

VM host-based firewall: Guest OS can use **host-based protection** as if it were running on physical hardware

Hypervisor firewall: a firewall supported directly inside the hypervisor

👍 More **efficient** than VM Bastion Host (it does not compete for resources with other VMs)

👍 More **secure**, in principle, as “invisible” by other VMs

👎 Add **complexity** to hypervisor