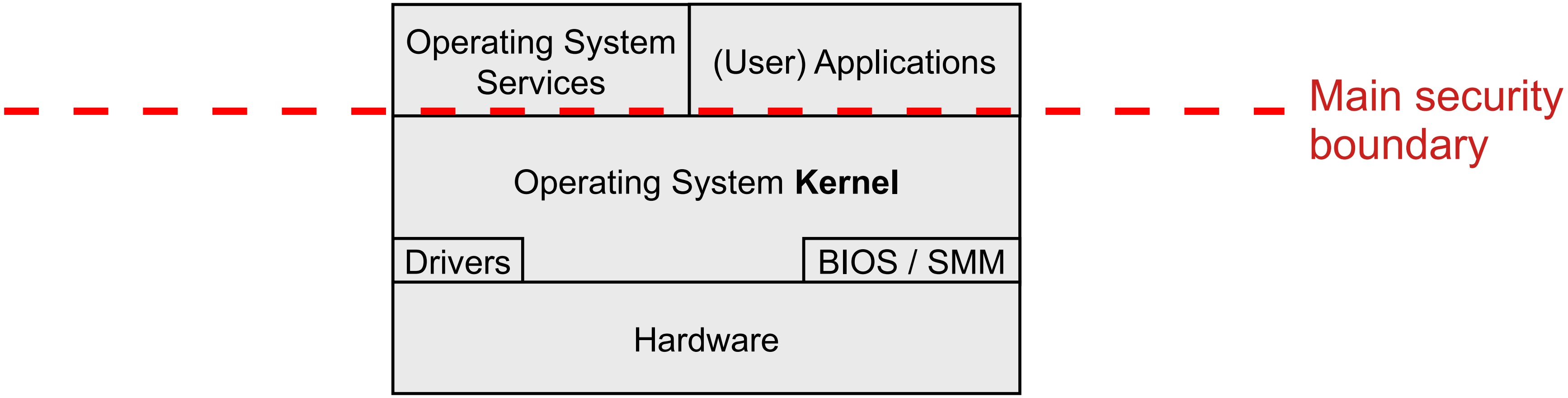
Chapter 7

Operating System Security

Operating System (OS) security

- Each layer of code needs measures in place to provide appropriate security services
- **Each layer is vulnerable to attack from below if the lower layers are not secured appropriately**



Access control to separate processes and users

- ITU-T Recommendation X.800 defines access control as follows:
  “The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.”
- RFC 2828 defines computer security as:
  “Measures that implement and assure security services in a computer system, particularly those that assure access control service”.
- Access control required for different resources such as
  - **files**
  - **memory**
  - **network**, **I/O**, **hardware**, etc.

Access control policies

- Discretionary Access Control (DAC): based on the identity of the requestor and on access rules set by the **owner** of the entity
- Mandatory Access Control (MAC): based on comparing security labels with security clearances (set by a **policy**); mandatory because owner/accessor may not be able to delegate access
- Role-Based Access Control (RBAC): based on roles that users/processes have within a system and rules based on those roles

Standard file systems implement DAC, may be extended by MAC for better security against privilege escalation

DAC access matrix

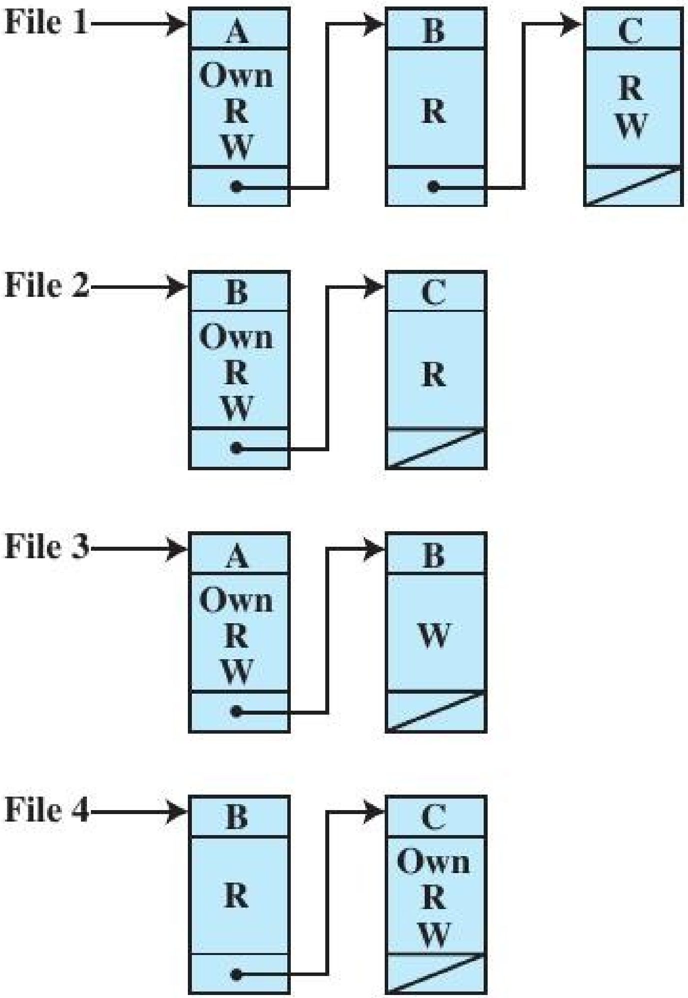
		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

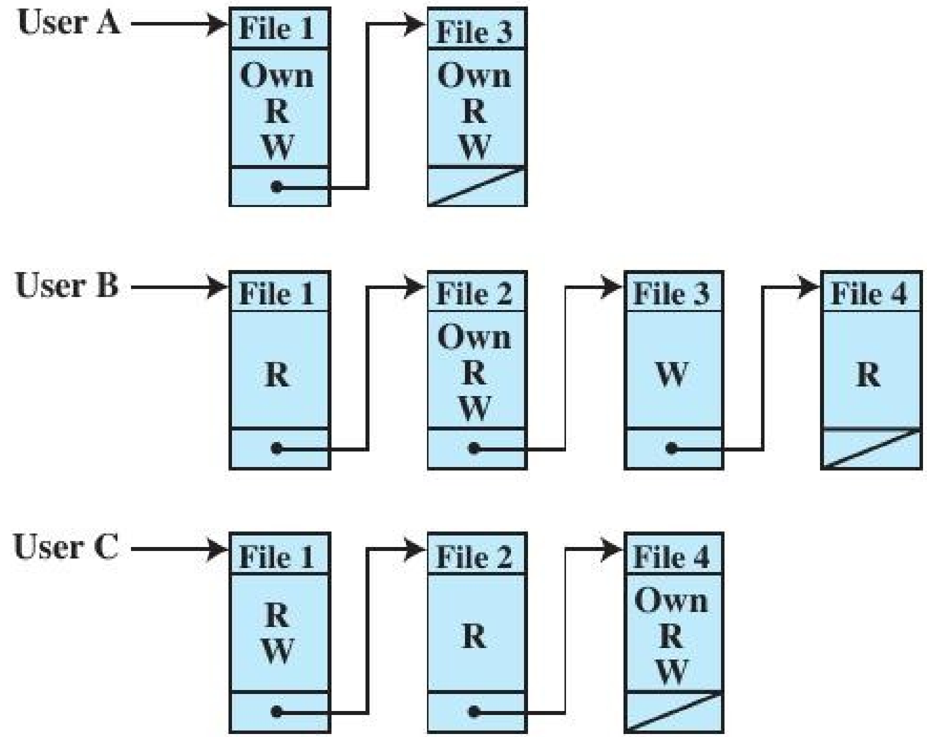
- Subjects are entities capable of accessing objects (users, their processes, etc.)
  Typical classes (from standard UNIX def.):
  - owner (creator or changed afterwards)
  - group (of subjects)
  - world (all know subjects)
- Objects are resources to which access is controlled (e.g. directories, files, network ports, virtual memory regions, etc.)
- Access rights describe the level of access to an object, standard set:
  - read
  - write
  - execute

  Or potentially more fine-grained (delete, create, search, etc.)

Access control lists (ACLs) vs. Capability lists

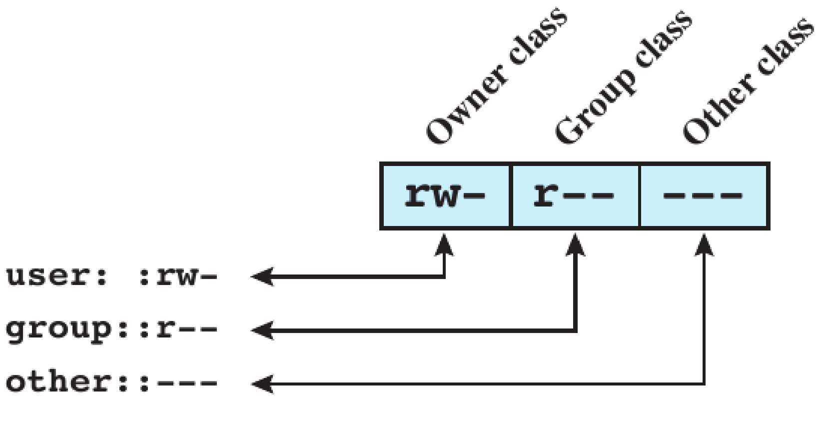


(b) Access control lists for files of part (a)

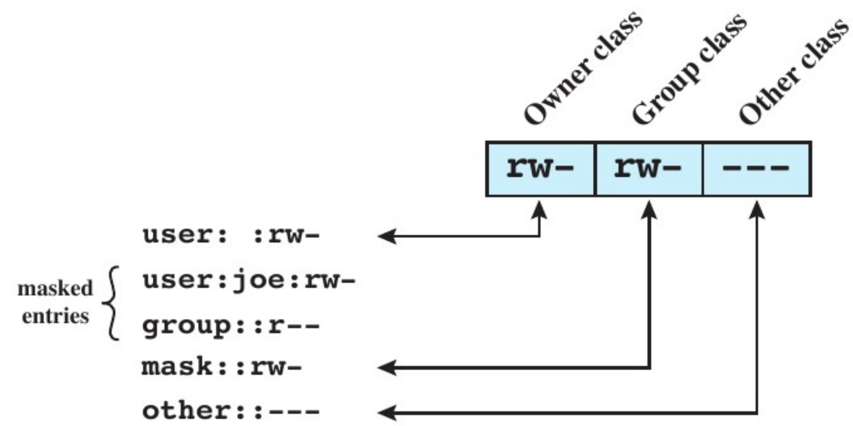


(c) Capability lists for files of part (a)

Access control lists on UNIX



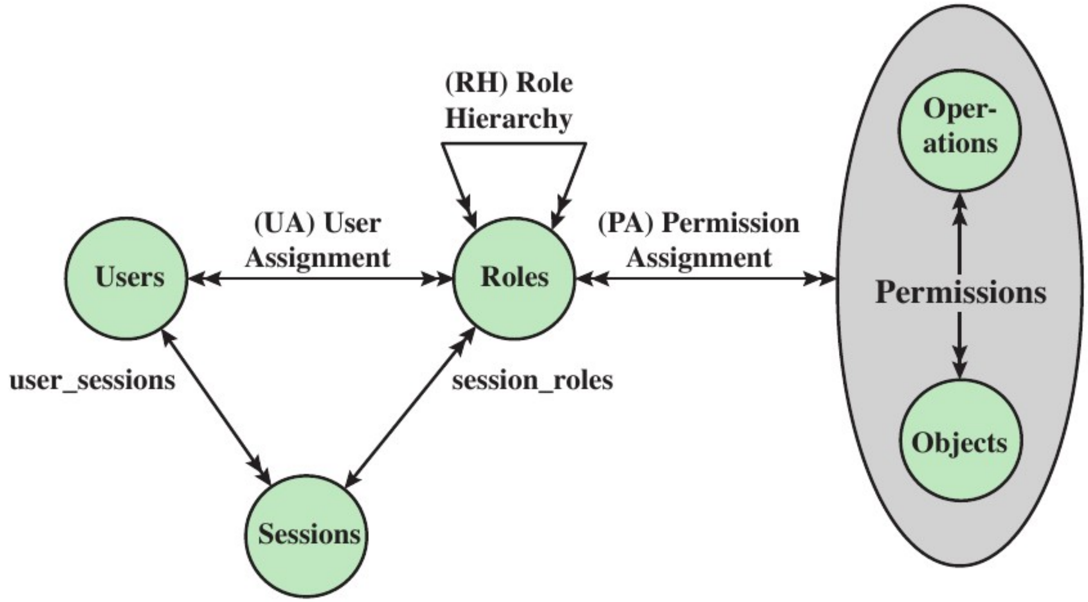
(a) Traditional UNIX approach (minimal access control list)



(b) Extended access control list

- Unique (numeric) user ID (UID)
- Member of a primary group ID (GID) and potential auxiliary groups
- Traditionally 12 bits (read/write/execute for owner/group/world plus setuid, setgid, and sticky bits)
- Modern UNIX systems support full ACL with arbitrary subject/access right combinations
- Superuser („root“) is exempt from these restrictions

Role-based access control (RBAC)



- Additional indirection between subjects and object access rights
- Can be emulated with groups in DAC model, but might lose hierarchy between roles in this case
- RBAC often coupled with MAC policy
- Many extensions, e.g. time-based, incompatible roles, one-role-at-a-time, only one role per session...

Mandatory access control (MAC)

- In contrast to DAC, MAC is managed by administrator
- In practical implementations, superuser is also subject to MAC policy
- Relates security classification of objects with security clearances of subjects to define access rights
- Security classifications and clearances are organized in levels
- With definition of multilevel security (MLS) with two main properties:
  - *no read up*: subject can only read an object of less or equal security level (called simple security property, ss-property)
  - *no write down*: subject can only write an object of greater or equal security level (star property, *-property)
  - additional property to implement DAC model, i.e. granting another subject/role access to resource under owner's discretion (ds-property)

Formal definition in terms of Bell-LaPadula (BLP) model

Case study: SELinux

- **„Security Enhanced Linux"**
- Developed by NSA and released as open source (GPL) in 2000, merged into mainline Linux kernel in 2003
- Implements MAC for Linux with policy support for MLS and RBAC
- Shipped with all modern Linux distributions (RedHat pioneered it and spends effort on policy improvements, e.g. Debian allows to easily enable SELinux support)
- Android 4.3 started shipping SELinux in permissive mode, Android 4.4 switched to enforcing/strict mode by default

Short summary: additional restrictions to user and daemon processes, very fine granularity on (pseudo-) files, network sockets, etc. → even the `root` user can be severely restricted

Case study: SELinux

Concept of “**type**”

- Files, sockets, etc. have a type
- E.g. `httpd_sys_content_t` for objects under `/var/www`
- E.g. `etc_t` for objects under `/etc`

Concept of “**domain**”

- Processes run in a domain
- Directly determines which access to types the process has
- E.g. `named_t` for the name server daemon
- E.g. `initrc_t` for init scripts

Case study: SELinux

Concept of "**role**"

- Roles define which user or process can access what domains (processes) and what type (files, sockets, etc.)
- Users and processes can transition to roles (e.g. during login)
- E.g. `user_r` for ordinary users
- E.g. `system_r` for processes starting under system role
- Rules determine which transitions are allowed
  → the "**SELinux policy**"

Files are "labeled" with types, the policy defines which domains the users and processes should run in

→ need filesystem and user space loader support for SELinux in addition to kernel support

Case study: SELinux

Concept of “**identity**”

- Every user account has an identity
- Identities do not change
- Identities determine which roles a user can transition to
- E.g. `user_u` for generic unprivileged users
- E.g. `root` for the superuser account

Concept of “**security context**”

- Every process and object has an associated security context with three fields (when printed in text, then denoted by colon)
  - *identity:role:domain* (for processes)
    or
  - *identity:role:type* (for files, directories, devices, sockets, etc.)

Case study: SELinux

- Example of process security context

```
root@pub ~ # ps -o pid,ruser,args,context -C apache2.prefork
  PID RUSER    COMMAND                          CONTEXT
23214 root     /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23216 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23227 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23228 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23230 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23231 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23232 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23444 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
```

- Example of user security context

```
root@pub ~ # id -Z
unconfined_u:unconfined_r:unconfined_t:SystemLow-SystemHigh
```

- Example of file security context

```
root@pub ~ # ls -Z /etc/apache2/apache2.conf
system_u:object_r:httpd_config_t:SystemLow /etc/apache2/apache2.conf
root@pub ~ # ls -Z /var/www/html/index.html
unconfined_u:object_r:httpd_sys_content_t:SystemLow /var/www/html/index.html
```

Read-only web content

Case study: SELinux

- Additional support tools, e.g. audit daemon to log violations of SELinux policy
- Tools to create and compile policy as well as load during system bootup
- Modularized policy allows loading of policy “modules” (often rules for specific applications/daemons) at run time (if not prevented by main policy)
  - e.g. Android allows run-time loading of additional policies only when these are signed by the same private key that signed the whole system (firmware) image
  - additional support for boolean variables to en-/disable policy parts
- Two modes
  - permissive (report violations, but don't block)
  - enforcing (only allow what is permitted by policy)

Memory isolation

- One main task of OS is to isolate virtual process memory
- On standard Intel-compatible processors (x86, amd64, etc.), use separation into processor „rings“ to split privileged „kernel“ code from unprivileged „user space“ code
  - on ARM instruction set, use privilege levels (EL3-EL0)
- Communication between different processes has to use kernel interfaces → so-called context switches to copy memory regions between user space and kernel space
- Efficient memory separation is supported by processor hardware (available on all modern CPUs)

Trusted systems

- **Trust**: „The extent to which someone who relies on a system can have confidence that the system meets it specifications.“
- **Trusted system**: a system believed to enforce a given set of attributes to a stated degree of assurance
- **Trusted computing base (TCB)**: portion of a system that enforces a particular policy, must be resistant to tampering and circumvention
  - informally, those components one **has** to trust for a system to be trustworthy
  - practically, needs to be small and simple enough to allow systematic analysis or even formal validation

Trusted Platform Module (TPM)

- Concept from Trusted Computing Group
- Hardware module at heart of hardware/software approach to trusted computing (TC)
- Uses a TPM chip
  - motherboard, smart card, processor
  - working with approved hardware/software
  - generating and using crypto keys
- Slowly being used in mobile devices as well

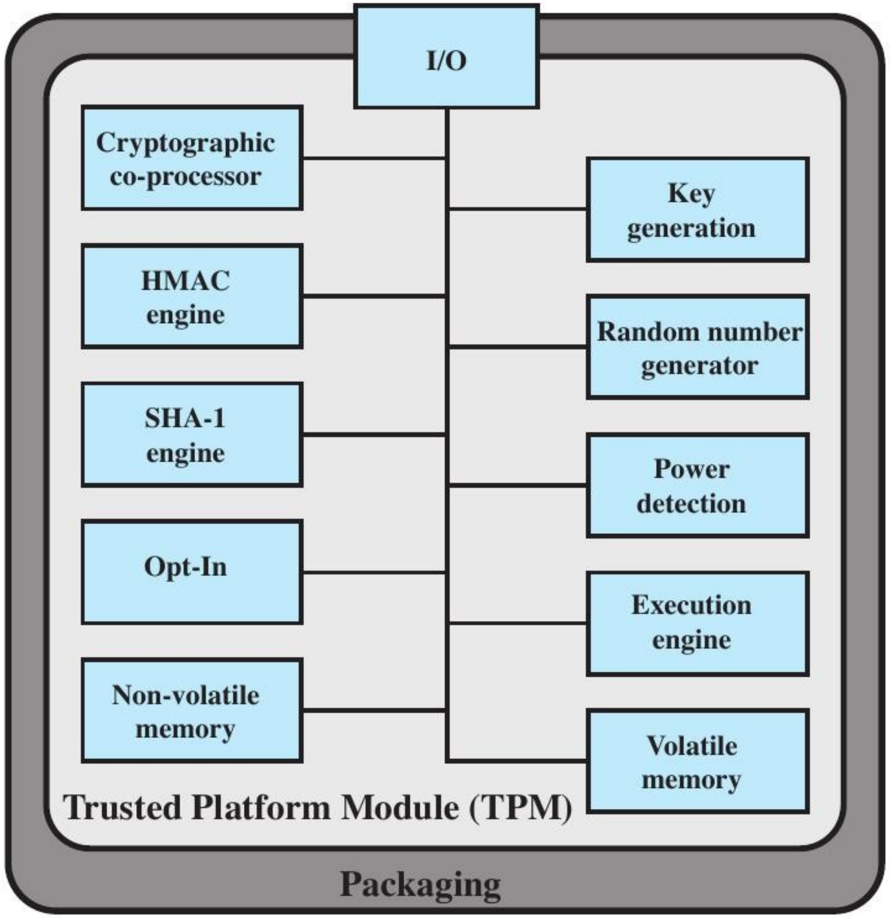


Figure 13.12 TPM Component Architecture

Secure/trusted/verified/ authenticated/... boot

- Responsible for booting entire OS in stages and ensuring each is valid and approved for use
  - at each stage digital signature associated with code is verified
  - TPM keeps a tamper-evident log of the loading process
- Log records versions of all code running
  - can then expand trust boundary to include additional hardware and application and utility software
  - confirms component is on the approved list, is digitally signed, and that serial number hasn't been revoked
- Result is a configuration that is well-defined with approved components
  - Note: "approved content" ≠ "correct content" ≠ "bug-free content"
    - bug in boot loader → load any kind of modified OS and mark it as "good"

Certification service

- Once a configuration is achieved and logged the TPM can certify configuration to others
  - can produce a digital certificate
- Confidence that configuration is unaltered because:
  - TPM is considered trustworthy
  - only the TPM possesses this TPM's private key
- Include challenge value in certificate to also ensure it is timely
  - replay attacks - get value from "good" boot and substitute it
- Provides a hierarchical certification approach
  - hardware/OS configuration
  - OS certifies application programs
  - user has confidence is application configuration

Encryption service

- Encrypts data so that it can only be decrypted by a machine with a certain configuration
- TPM maintains a master secret key unique to machine
  - used to generate secret encryption key for every possible configuration of that machine
- Can extend scheme upward
  - provide encryption key to application so that decryption can only be done by desired version of application running on desired version of the desired OS
  - encrypted data can be stored locally or transmitted to a peer application on a remote machine

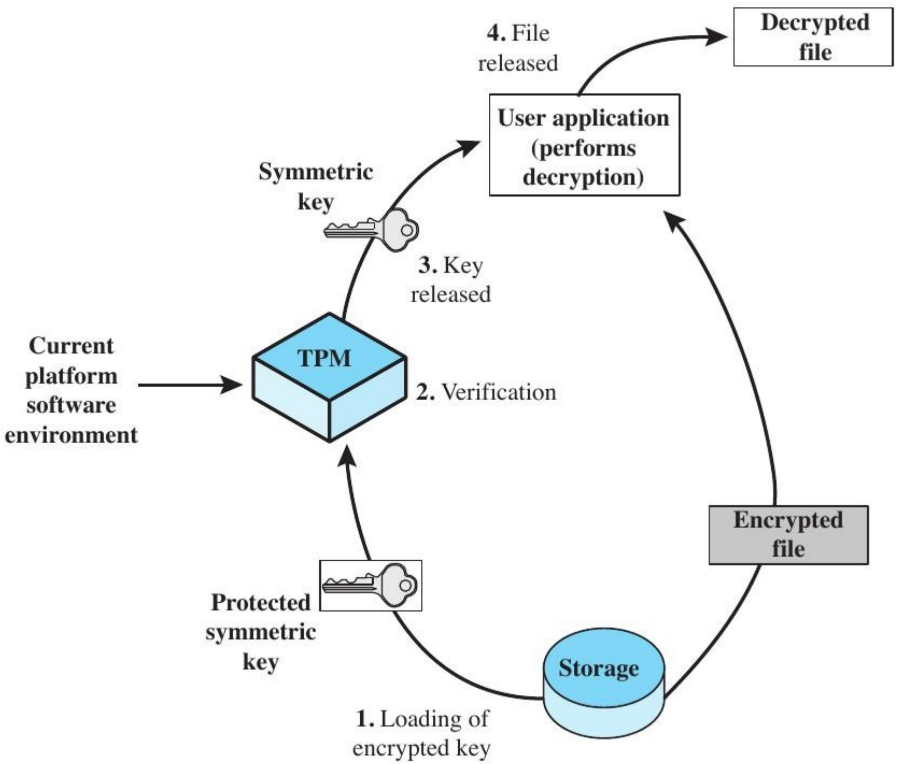


Figure 13.13 Decrypting a File Using a Protected Key

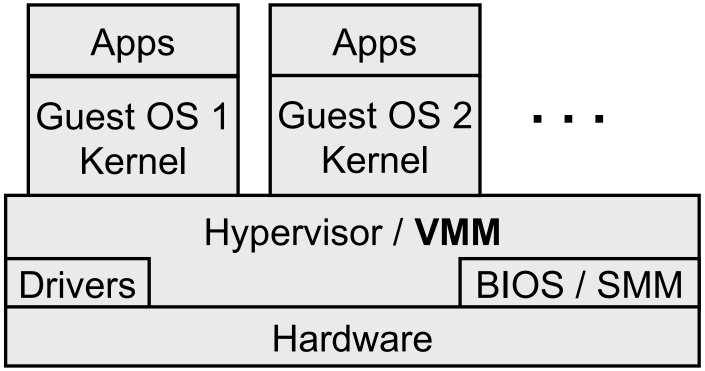
Virtual Machine Manager (VMM) as a TCB

- **Virtualization**: a technology that provides an abstraction of the resources used by some software which runs in a simulated environment called a virtual machine (VM)
  - benefits include better efficiency in the use of the physical system resources
  - provides support for multiple distinct operating systems and associated applications on one physical system
  - raises additional security concerns
- Additional software layer: Virtual Machine Manager (VMM), sometimes also called hypervisor, often related to the concept of a microkernel
- VMM is responsible for isolation/separation of guest operating systems → sometimes referred to as compartmentalization
- If VMM does this securely, guest OS cannot attack each other, the VMM, or the hardware
- Therefore, VMM becomes trusted computing base (TCB)

VMM types

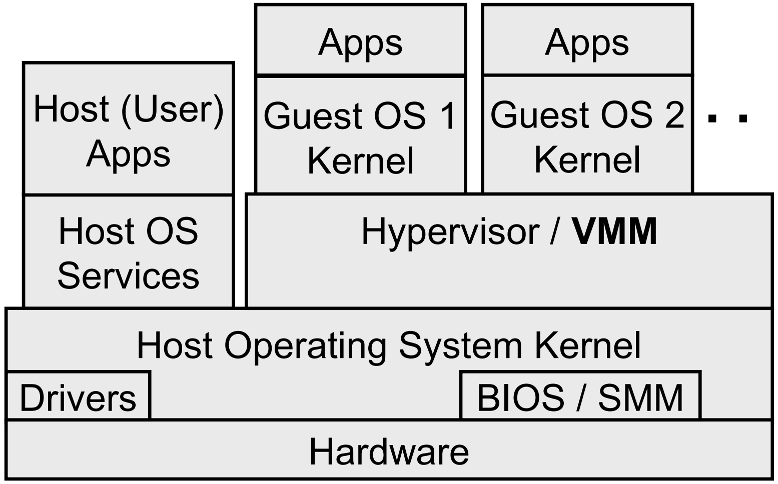
Type 1 VMM

- Also called „native“, „full“, or „bare-metal“ virtualization
- Runs natively on hardware
- Multiple OS on top, none of these guest OS is privileged



Type 2 VMM

- Also called „hosted“ virtualization
- Runs on top of „host“ OS
- Multiple guest OS on top



Comparison of VMM types

- **Type 1 VMM**
  - sometimes assumed to be the most secure
  - in practice also depends on hardware drivers and therefore adds complexity of a small OS (TCB is more than just the hypervisor!)
  - example implementations: VMware ESX(i), Xen, L4, pKVM
- **Type 2 VMM**
  - easier to set up, can be installed as a (privileged) application on top of standard OS
  - uses hardware drivers and scheduling of host OS kernel (TCB is host kernel+userspace+hypervisor)
  - example implementations: VMware Workstation, VirtualBox, KVM/Qemu
- **Application virtualization / container concepts**
  - not really virtualization, but often used as a low-overhead replacement
  - single OS kernel, compartments/containers/zones on top with different name spaces for file systems, network, processes, etc.
  - example implementations: Solaris Zones, Linux Container, Docker.io

Common Criteria (CC)

- Common Criteria for Information Technology and Security Evaluation
  - ISO standards for security requirements and defining evaluation criteria
- Aim is to provide greater confidence in IT product security
  - development using secure requirements
  - evaluation confirming meets requirements
  - operation in accordance with requirements
- Following successful evaluation a product may be listed as ”CC certified”
  - NIST/NSA publishes lists of evaluated products

Case study: Qubes OS

- **Qubes OS** is an open source desktop operating system building upon Linux and virtualization (Xen hypervisor in R1 and R2, different VMMs supported starting with R3)
- Main focus is on **security by compartmentalization**
  - task based, not application based
  - virtual machines for different security domains, e.g. work, personal, banking, private key storage and use, untrusted, etc.
  - supports different guest OS, including full virtualization (e.g. Windows)
  - innovation is nearly seamless integration of windows (with indication of security domain) and interaction between VMs
- Can be used on most recent desktop/laptop hardware (hardware driver support by Linux kernel as available in recent Fedora releases)

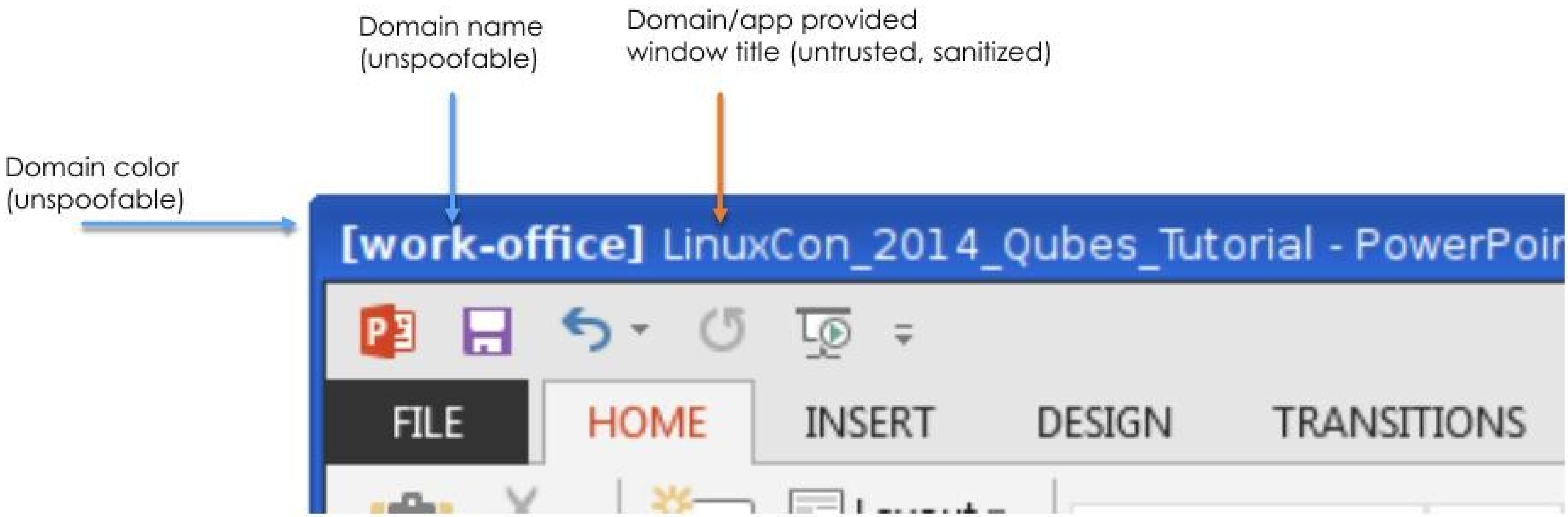
Qubes OS architecture features

- Based on a (relatively small and secure) type-1 hypervisor (Xen), support for other VMMs starting with R3
- Networking code sand-boxed in an unprivileged VM (using IOMMU/VT-d)
- USB stacks and drivers sand-boxed in an unprivileged VM (experimental in R2)
- No networking code in the privileged domain (dom0)
- All user applications run in “AppVMs,” lightweight VMs based on Linux (or Windows starting with R2)
- Centralized updates of all AppVMs based on the same template
- Qubes GUI virtualization presents applications as if they were running locally
- Qubes GUI provides isolation between apps sharing the same desktop
- Secure system boot based (optional)

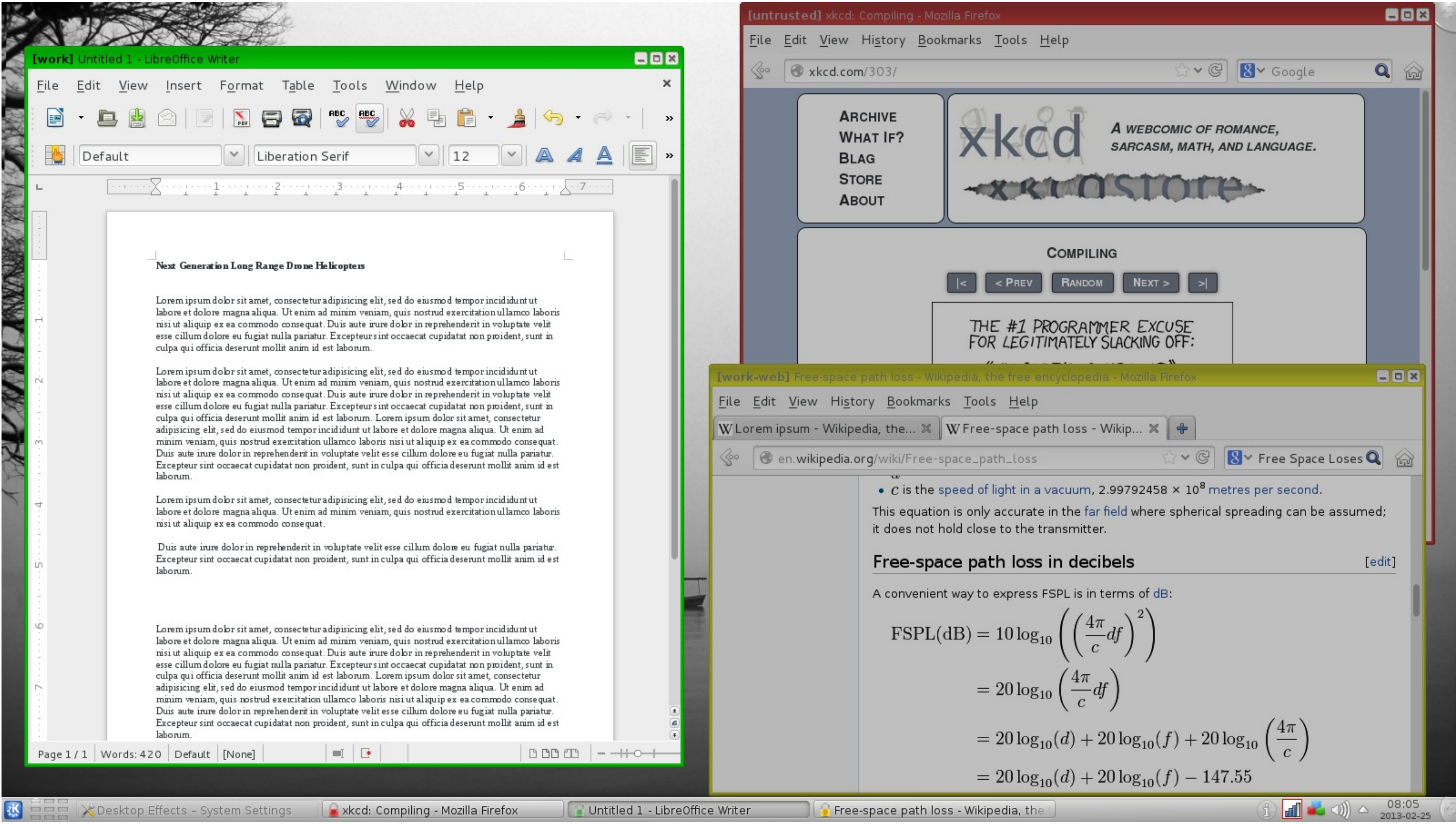
Qubes OS security domains

- Domains represent areas, e.g.
  - personal, work, banking
  - work-web, work-project-XYZ, work-accounting
  - personal-very-private, personal-health
- No 1-1 mapping between apps and VMs!
  - If anything, then user tasks-oriented sandboxing, not app-oriented
  - E.g. few benefits from sandboxing: The Web Browser, or The PDF Reader
- It’s data we want protect, not apps/system

Qubes OS window decorations



Qubes OS windows from different security domains



Acknowledgments: screenshot from https://qubes-os.org/wiki/QubesScreenshots

Qubes OS windows from different security domains

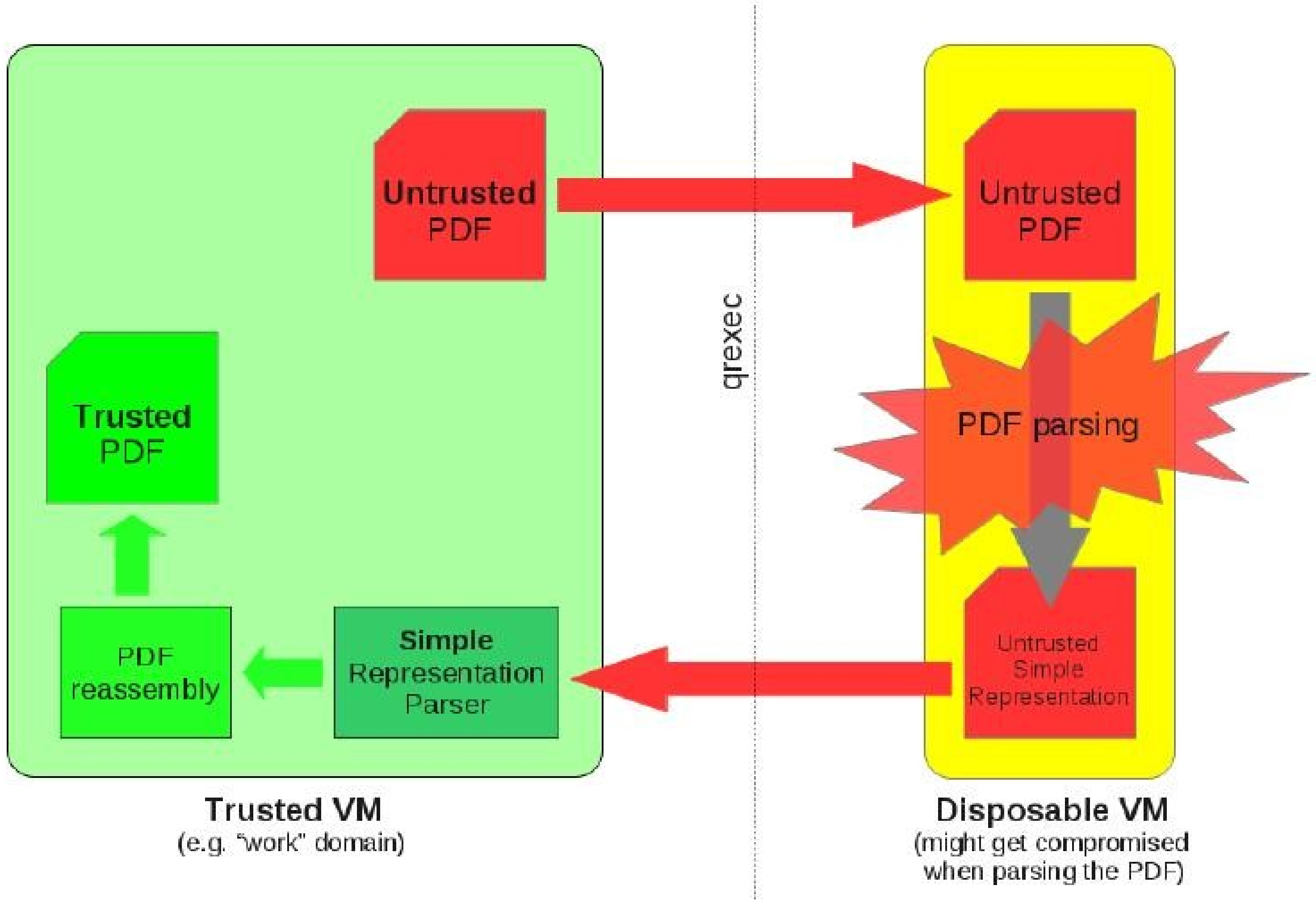
Acknowledgments: screenshot from https://qubes-os.org/wiki/QubesScreenshots

Qubes OS types of VMs from network point of view

- **NetVMs**
  - have NICs or USB modems assigned via PCI-passthrough
  - provide networking to other VMs (run Xen Net Backends)
- **AppVMs**
  - have no physical networking devices assigned
  - consume networking provided by other VMs (run Xen Net Frontends)
  - some AppVMs might not use networking (i.e. be network-disconnected)
- **ProxyVMs**
  - behave as AppVMs to other NetVMs (or ProxyVMs), i.e. consume networking
  - behave as NetVMs to other AppVMs (or ProxyVMs), i.e. provide networking
  - functions: firewalling, VPN, Tor’ing, monitoring, proxying, etc.
- **Dom0**
  - has no network interfaces!

Acknowledgments: summary by Joanna Rutkowska

Qubes OS example case: sanitizing PDFs



Acknowledgments: summary by Joanna Rutkowska