

# Tema 2. Seguridad en Sistemas Operativos



# Contenidos

---

01

## Propiedades

---

Caracterización de SSOs

02

## Autenticación, autorización y control de acceso

---

Cómo accedemos al sistema y usamos los recursos

03

## Sistemas operativos confiables

---

SOs enfocados a la seguridad

04

## Fortalecimiento del SO

---

Mecanismos para fortalecer la seguridad de un SO

05

## Garantía de la seguridad

---

Medidas del nivel de seguridad de un SO

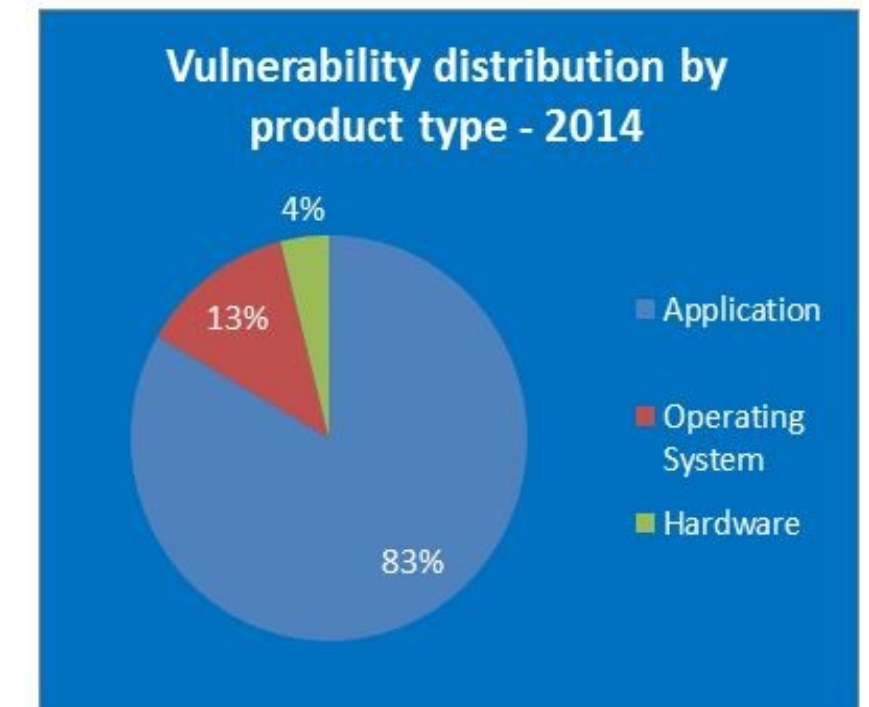
1

Propiedades



# Vulnerabilidades en SOs

	(1)	(2)	(3)	(4)
Operating system	# of vulnerabilities	# of HIGH vulnerabilities	# of MEDIUM vulnerabilities	# of LOW vulnerabilities
Apple Mac OS X	147	64	67	16
Apple iOS	127	32	72	23
Linux Kernel	119	24	74	21
Microsoft Windows Server 2008	38	26	12	0
Microsoft Windows 7	36	25	11	0
Microsoft Windows Server 2012	38	24	14	0
Microsoft Windows 8	36	24	12	0
Microsoft Windows 8.1	36	24	12	0
Microsoft Windows Vista	34	23	11	0
Microsoft Windows RT	30	22	8	0



Distribución	(1)	(2)	(3)	(4)
Ubuntu	39	7	27	5
Red Hat E.	27	6	17	4
OpenSuse	22	9	9	4
Fedora	15	3	9	4
Android <sup>1</sup>	6	4	1	1

<sup>1</sup> Sin incluir las del kernel

- C. Florian, GFI Blog, 2015 en “Most vulnerable operating systems and applications in 2014”  
<http://www.gfi.com/blog/most-vulnerable-operating-systems-and-applications-in-2014/>  
 datos de la National Vulnerability Database (NVD).




# Objetivos de seguridad

---

- **Objetivos de seguridad:** define las operaciones que el sistema puede ejecutar mientras evita el acceso no autorizado (satisfacer triada CIA). El desarrollador del SO debe definir objetivos verificables.
- Los objetivos de seguridad se pueden definir como:
  - ◆ **Requisitos** (p. ej. propiedad de seguridad-sencilla de Bell-LaPadula – un proceso no puede leer un objeto cuya clasificación de confidencialidad es mayor que de la del proceso).
  - ◆ **Funcionalmente** (p. ej. principio del *menor privilegio* - un proceso solo puede realizar las operaciones necesarias para su ejecución) - son insuficientes pues, si bien evitan algunos ataques, no prueban la ausencia de una vulnerabilidad.
- Los SOs actuales son capaces de expresar y aplicar objetivos de seguridad, pero a pesar de ello: Pueden ser muy restrictivos para un sistema de producción, o no estar muy claros.

# Modelo de confianza



-  Un **modelo de confianza** (trust) de un sistema define el conjunto mínimo de software y datos sobre los cuales el sistema depende para hacer cumplir de forma correcta los objetivos de seguridad
-  Para un SO, el modelo de confianza es sinónimo de la **Base de Computación Segura** (TCB) del sistema.
-  Forman parte del TCB:
  - ◆ Mecanismos de arranque del sistema -que habilita la carga de los objetivos de seguridad.
  - ◆ SO completo – no hay fronteras de protección entre sus funciones.
  - ◆ Algunas utilidades, p. ej. login, ssh, etc.
  - ◆ Entorno X-windows que realiza servicios para todos los procesos del sistema y mecanismos de compartición. .

# Modelo de confianza (y ii)

---

- Los desarrolladores de SO deben probar que sus sistemas tienen un modelo viable de confianza, es decir:
  - ◆ El TCB media en todas las operaciones sensibles de seguridad
  - ◆ La verificación de la correctitud del software/datos del TCB
  - ◆ La verificación de que la ejecución del software no puede ser alterada por los procesos fuera del TCB.
- Separaremos la seguridad en:
  - ◆ Interna: Proteger/eliminar vulnerabilidades del kernel (lo abordaremos en *hardening* y al estudiar *malware*)
  - ◆ Servicios de seguridad del SO:
    - Control de acceso a recursos (a continuación ...)
    - Servicios: cifrado, gestión de claves, *firewall*, IDS, ...



# Modelo de confianza: implementación



- Tradicionalmente han garantizado la confianza del TCB mediante tres mecanismos básicos:
  - ◆ **Funcionamiento en modo dual** – El SO se ejecuta en modo kernel (acceso ilimitado). Los procesos se pueden ejecutar en modo kernel pero un camino de código -acciones- esta controlado
  - ◆ **Protección de memoria** – el SO construye un espacio de direcciones aislado para cada proceso. Un proceso no puede acceder a nada fuera de su espacio de direcciones pues tan siquiera puede “nombrarlo”.
  - ◆ **Autorización** del uso de los recursos del sistema a quienes les esta permito, e impidiendo el acceso a quienes no lo están.



The background features a hand with a green semi-transparent overlay. Various white icons are floating around, including a magnifying glass with a plus sign, a speech bubble, a smartphone, a document, a mail icon, a play button, a music note, a location pin, and a person icon. The overall theme is digital security and access control.

# 2

Autenticación,  
autorización y control  
de acceso

# Autenticación, autorización y control de acceso



## Autenticación



**Autenticación** como el proceso de verificar que una entidad es quién dice ser.

Consta de dos fases:

**Identificación:** afirmación de quién somos que se realiza presentando un identificador.

**Verificación:** generar información de autenticación que corrobora la ligadura entre la entidad y el identificador.

## Autorización



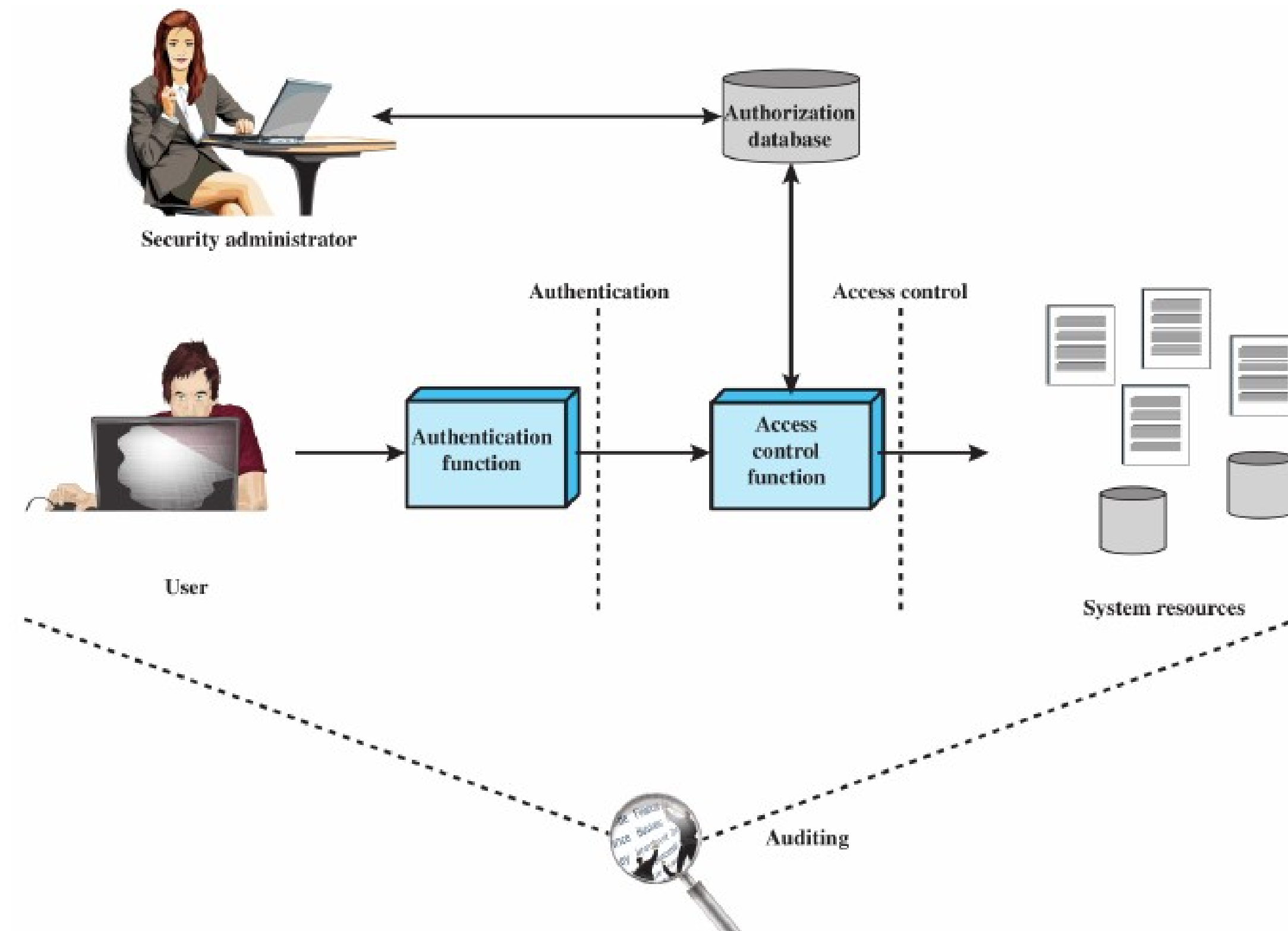
La autorización verifica que el usuario autenticado tiene los permisos correctos y derechos de acceso a los recursos solicitados (cedidos por el administrador de la seguridad)..

## Control de acceso





Examinar si un sujeto posee autorización para acceder a un objeto cuando se produce una petición de acceso..

# Relación de conceptos





# Medios de autenticación



-  Los medios generales para realizarla, que se pueden utilizar solo o combinados, son:
  - ◆ Algo que el usuario conoce
  - ◆ Algo que el usuario posee
  - ◆ Algo que el usuario es (biometría estática) o hace (biometría dinámica)
  
-  **Autenticación multifactor (MFA):** cuando usamos la conjunción de dos o más de los medios básicos anteriores.



# Características



-  Las características de un buen método de autenticación son::
  - ◆ Resistente a conjeturas de un impostor.
  - ◆ El usuario debe recordarlo fácilmente..
  - ◆ Sencillo y conveniente de proveer cuando se necesita.
  
-  Los métodos más usados::
  - ◆ Password.
  - ◆ Token.
  - ◆ Biométricos..

# Claves



-  El nivel de protección es efectivo solo si se mantienen buenas prácticas de gestión. Mantener equilibrio entre seguridad y usabilidad.
-  Se compromete la seguridad si:
  - ◆ Se elige un mal *password* (*login*, corto, palabras diccionario, información personal, etc.).
  - ◆ Se comparte.
  - ◆ Se escribe en una nota.
  - ◆ Se mantiene durante mucho tiempo
  - ◆ Se utiliza el mismo, o similar, en múltiples sistemas

# Ataques a claves

- Ataques de fuerza bruta o de diccionario
- Secuestro de estación de trabajo no atendida y con una sesión abierta..
- Explotando errores del usuario – apuntar, compartir, o informar de claves (ingeniería social) o explotando el uso múltiple de claves entre dispositivos o servicios.
- Monitorización electrónica – *sniffer, keylogger, etc.*

Most Used Worst Passwords of 2015		
1. 123456	11. welcome	19. letmein
2. password	12. 1234567890	20. login
3. 12345678	13. abc123	21. princess
4. qwerty	14. 111111	22. qwertyuiop
5. 12345	15. 1qaz2wsx	23. solo
6. 123456789	16. dragon	24. passw0rd
7. football	17. master	25. starwars
8. 1234	18. monkey	
9. 1234567		
10. baseball		





# Estrategias de selección de claves



- La cuestión es eliminar claves fáciles de adivinar mientras que son fáciles de recordar (equilibrio entre seguridad y usabilidad):
  - ◆ Educación de usuarios – problemas: población grande, pueden obviar las reglas, pueden no entender la fortaleza de una clave.
  - ◆ Claves generadas automáticamente – pueden ser difíciles de recordar
  - ◆ Prueba reactiva de claves – ejecutar periódicamente un *craker* de claves y ver cuales son débiles para notificar al usuario. Problema computacional y ventana temporal de debilidad.
  - ◆ Prueba proactiva de claves – comprobar fortaleza en asignación sin producir muchos rechazos: reglas de construcción, diccionario de claves malas, filtro Bloom.

# Claves: buenas prácticas






La Norma SP800-63b cambia el enfoque hasta el momento al hacer algunas recomendaciones sobre gestión de claves basadas en la idea “not fix de user, fix de security systems”:

- ◆ Evitar reglas de complejidad de claves – usar frases de paso (pass phrases).
- ◆ Evitar la expiración de claves – salvo que este comprometida.
- ◆ Permitir los gestores de claves.




# Enfoques alternativos



-  Los enfoque **pregunta-respuesta** suelen utilizarse como autenticación de segundo nivel:
  - ◆ Enfoque cognitivo
  - ◆ Enfoque asociativo
-  **Métodos visuales** o gráficos: recordar una secuencia de imágenes, detalles de una imagen o dibujar una imagen – basados en que una imagen es más fácil de recordar.
-  La claves prevalecen: son más rápidamente comprendidas, más rápidas de usar, ampliables a un gran número de sistemas y servicios.

# Verificación en dos fases



-  Muchos servicios, especialmente en la nube utilizan una autenticación de dos fases (2FA).
-  Fases:
  - ◆ Primera: convencional
  - ◆ Segunda: se envía mensaje al móvil / e-mail con código de verificación.
-  **Problemas:**
  - ◆ Robo o pérdida del BYOD, token, ...
  - ◆ No acceso al correo
  - ◆ Fatiga del usuario
  - ◆ ...



# Autenticación basada en tokens

- El token (que solo lo tiene la persona legítima) esta presente en el proceso de autenticación.
- Características:
  - ◆ Cada token debe ser único.
  - ◆ Debe ser difícil de duplicar.
  - ◆ Debe tener un factor de forma conveniente y portable.
  - ◆ Barato y rápidamente reemplazable por si se pierde, roba o falsea.



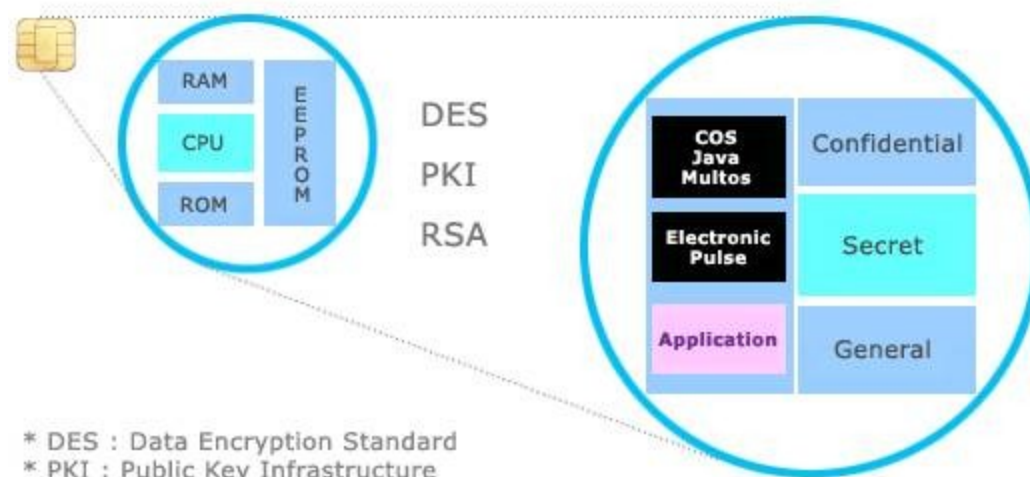
# Integridad: tecnologías



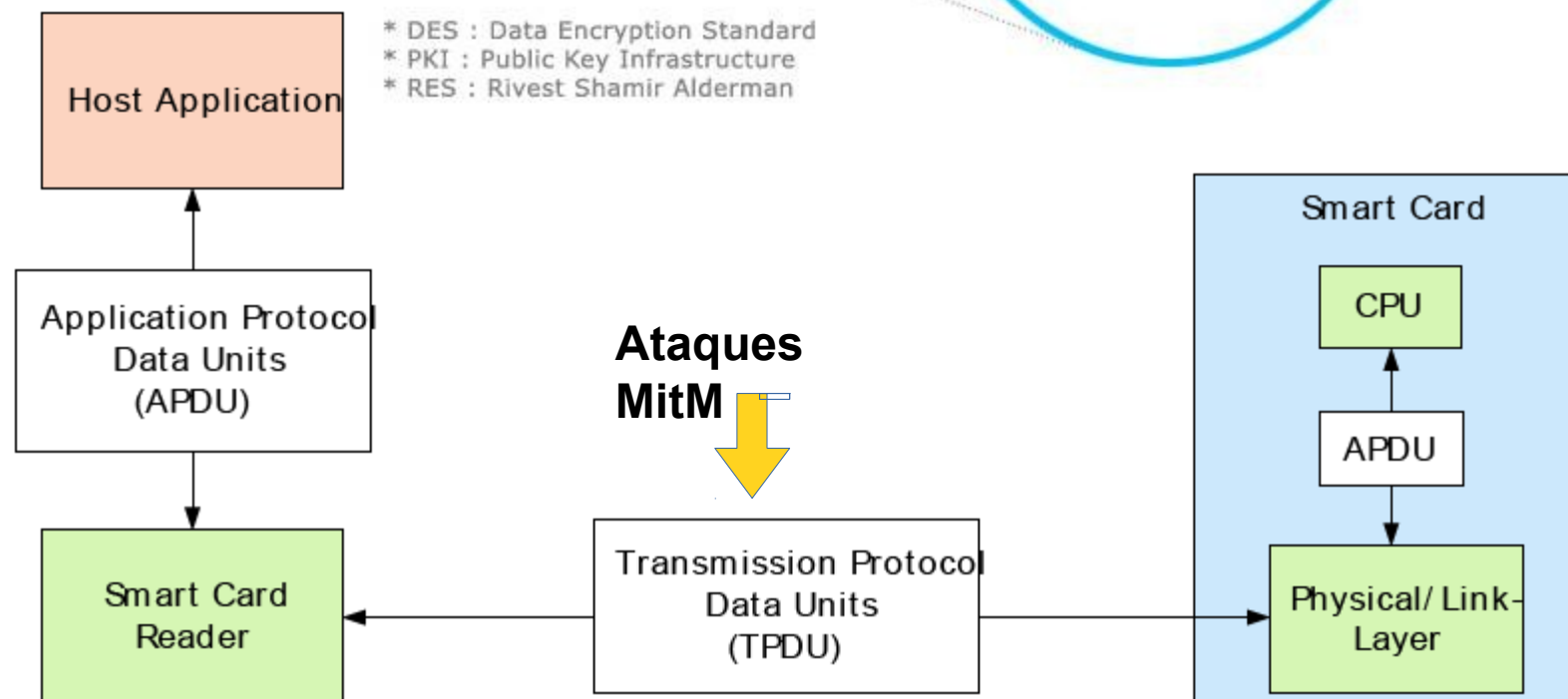
-  Pueden ser:
  - ◆ **Pasivos** - mantiene un secreto que se presenta al sistema de autenticación. Por ejemplo, banda magnética, tokens de proximidad, etc.
  - ◆ **Activos** - el secreto se procesa dentro del token para que no salga de él y genera una clave única. Por ejemplo, tarjetas inteligentes, basados en USB, etc.
  
-  Suelen usarse como mecanismo de autenticación de dos factores, junto con claves o biométricos. Ejemplo, tarjeta inteligente con PIN.

# Tarjetas inteligentes

Estándares: ISO 7816, COS (chip OS), ...



\* DES : Data Encryption Standard  
\* PKI : Public Key Infrastructure  
\* RES : Rivest Shamir Alderman



◆ **Seguridad a nivel físico** : mecanismos para evitar ataques como *Differential Power Analysis* (DPA) o *Simple Power Analysis* (SPA).




◆ **Seguridad del SO** tiene un sistema de archivos jerárquico con varios derechos de acceso a directorios y archivos:

1. Always (ALW): Acceso sin restricciones.
2. *Card holder verification 1* (CHV1): Acceso posible si esta presente un valor CHV1 válido (PIN de usuario).
3. *Card holder verification 2* (CHV2): Idem CHV2 (PIN de desbloqueo pre-grabado).
4. *Administrative* (ADM): Permitido a una autoridad administrativa.
5. *Never* (NEV): Acceso prohibido.



# Ataques contra tokens



-  La principal amenaza es el robo del token y, en los pasivos, la duplicación.
-  La autenticación de dos-factores ha desplazado el fraude a Internet. Por ejemplo, ya no es necesario conocer el PIN.
-  Los tokens activos necesitan de ataques más sofisticados, por ejemplo, secuestrar sesiones IP rastreando la red mientras se transmite la clave.

# Control de accesos: políticas

---

- El **control de accesos** se encarga de examinar si un sujeto (entidad capaz de acceder a los recursos) posee la autorización para acceder a un objeto (recurso cuyo acceso esta controlado) cuando se produce una petición de acceso al mismo.
- Podemos aplicar diferentes políticas de control de acceso:
  - ◆ **Control de Acceso Discrecional** (DAC) – permiten que el propietario del recurso determine quién tiene acceso y qué privilegios tienen.
  - ◆ **Control de Acceso Obligatorio** (MAC) – los controles se aplican en base a privilegios (o autorización) de un sujeto y la sensibilidad (o clasificación).
  - ◆ **Control de Acceso Basados en Roles** (RBAC) – dependen del rol (funciones) de los usuarios dentro del sistema y las reglas de acceso para ese rol.
  - ◆ **Control de Acceso Basado en Atributos** (ABAC) – dependen de los atributos del usuario, los recursos accedidos, y las condiciones actuales del entorno.

# Técnicas de control de accesos



- Podemos clasificarlas en:
  - ◆ Matriz de control de acceso
  - ◆ Listas de control de acceso
  - ◆ Capacidades
  - ◆ Control de acceso dependiente del contenido (el contenido del objeto determina el quien tiene acceso. Ej. Gestor accede solo al los registros de la BD de sus empleados)
  - ◆ Control de acceso dependiente del contexto (es el contexto, no contenido, Ej. Un *firewall* no permitirá paquetes SYN/ACK si no ha recibido uno SYN que correlacione la conexión)
  - ◆ Interfaces de usuario restringidas (menús y shell, vistas de BD, Interfaces restringidas físicamente).
  - ◆ Control de acceso basado en reglas (Ej. horario de acceso limitado)
  - ◆ Aislamiento temporal (basado en tiempo dado y limitado)

# Matriz de control de acceso

- Podemos materializar un esquema DAC mediante la **matriz de control de acceso**:

		Objects								
		Subjects			Files		Processes		Disk Drives	
		S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	F <sub>1</sub>	F <sub>2</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>
Subjects	S <sub>1</sub>	Control	Owner	Owner Control	Read*	Read Owner	Wakeup	Wakeup	Seek	Owner
	S <sub>2</sub>		Control		Write*	Execute			Owner	Seek*
	S <sub>3</sub>			Control		Write	Stop			

\*: Copy flag set

**Derechos:** forma en la que un sujeto puede acceder a un objeto (flujo de información entre sujeto u objeto)

# Matriz de acceso: descomposición



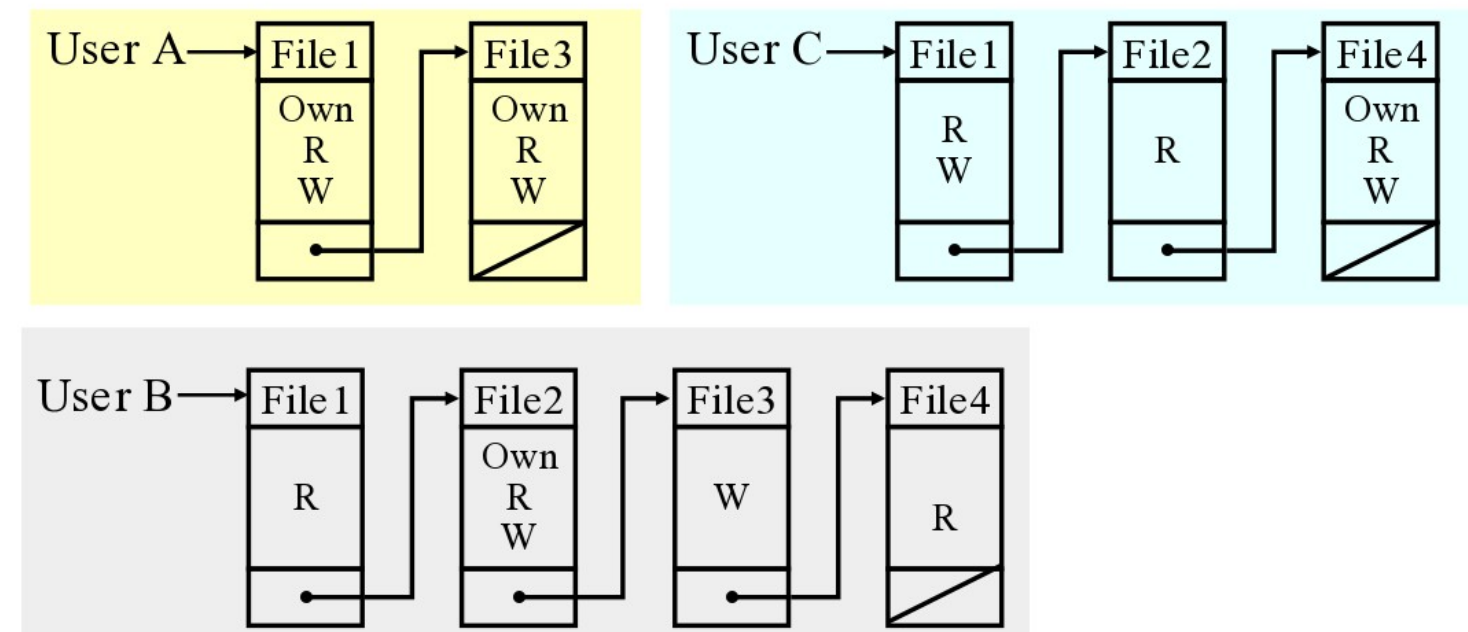
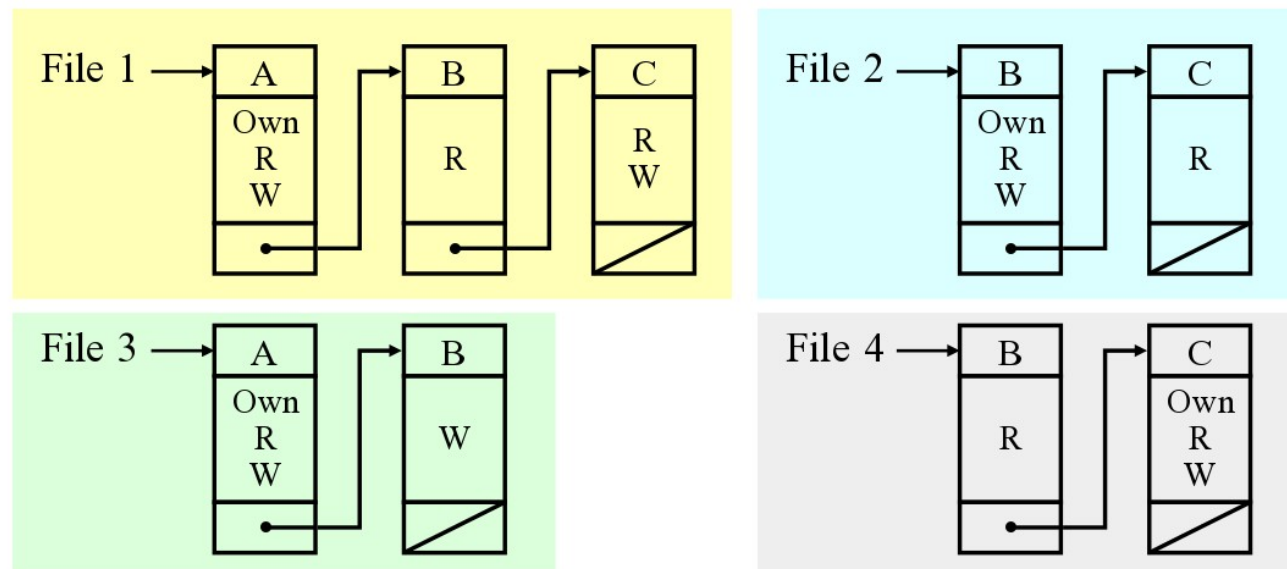
## Listas de Control de Acceso (ACL)

Lista ligada al objeto (columna de la matriz) que recoge quienes y qué derechos tiene sobre él.



## Capacidades (Capabilities)

Lista ligada a un sujeto (fila de la matriz de acceso) que recoge a qué objetos tiene acceso y con qué derechos.



# Listas de control de acceso



- Podemos clasificarlas en:
  - ◆ Fáciles de implementar
  - ◆ Consume tiempo su comprobación en sistemas con muchos sujetos (Unix lo redujo colapsándolas a tres tipos de usuarios (propietario, grupo, y otros).
  - ◆ La revocación de derechos es simple (en capacidades no es fácil).

# Capacidades en Linux



- Linux separa los privilegios del administrador/superusuario en unidades denominadas capacidades (mejoran la granularidad de los privilegios de root).  
Ej. CAP\_SYS\_TIME - un usuario no privilegiado puede cambiar la fecha del sistema sin ser root.

- Las órdenes para manipularlas son:

**getcap:** Lista las capacidades de un fichero

**setcap:** Asigna/borra capacidades a un fichero

**getpcaps:** Lista las capacidades de un proceso




**capsh:** Proporciona un "interfaz" de línea de órdenes para probar y explorar el uso de capacidades.

```
$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
/* uso de socket RAW y PACKETS
permitido y efectivo*/
```



# Desventajas de DAC (i)






-  *Asegurar la politica de seguridad:* Si bien en el modelo de empresa el propietario de la información es la empresa no el creador concreto, DAC asigna la propiedad del creador (no a la empresa).
-  Deberíamos poder asegurar que la seguridad recae en la empresa no en el usuario.
-  Autorización en cascada: revocar un derecho de acceso puede ser complejo, pues un sujeto puede tener derechos otorgados por otro y nosotros no saberlo:
  - ◆ Crear enlace duro a archivos con permisos “abiertos”.
  - ◆ Permisos de escritura en “otros” (rompe confidencialidad).

# Desventajas de DAC (y ii)

- *Ataque de caballo de Troya (**troyanos**):* Un caballo de troya puede utilizarse para otorgar cierto derecho  $(o, t, p)$  del sujeto  $s_i$  al sujeto  $s_j$  ( $i \neq j$ ) sin consentimiento de  $s_j$ .
- Cualquier programa que ejecuta un sujeto actúa en su beneficio y corre con su identidad → un troyano puede otorgar derechos de acceso sin que DAC lo controle. ¿Por qué no poner el directorio “.” en la variable del shell PATH?
- Flujo de control sin información:
  - ◆ DAC aborda acceso directo a objetos a través de los sujetos, sin embargo, no controla el flujo de información entre sujetos.
  - ◆ Si un usuario accede a cierto objeto puede crear una copia del mismo, y como propietario de la misma puede otorgar derechos a otros.




# MAC (Mandatory Access Control)



-  Mientras que un DAC esta interesado en definir, modelar y asegurar el acceso a la información, MAC está además interesado en el flujo de información dentro del sistema.
  - .
-  El término MAC se basa en que el control de acceso se basa en un conjunto de reglas predefinidas, que son obligatorias en lugar de definidas por el usuario.
-  Ejemplos:
  - Modelo Need-to-Know
  - Modelo de seguridad militar:
    - Modelo Bell-LaPadula
    - Modelo Biba.

# RBAC (Rol-Based Access Control)



-  El control de acceso se realiza en función de los roles del usuario en el sistema en lugar de su identidad. Un usuario puede tener diferentes roles y estos son asignados estática o dinámicamente.
-  Descansa en el principio: el propietario de la información no es el usuario si no el rol.
  - .
-  Implementa el principio de mínimo privilegio: cada rol tiene los privilegios mínimos para realizar su tarea.




# Matriz de acceso en RBAC

	$R_1$	$R_2$	• • •	$R_n$
$U_1$	✕			
$U_2$	✕			
$U_3$		✕		✕
$U_4$				✕
$U_5$				✕
$U_6$				✕
•				
•				
$U_{n2}$	✕			

		OBJECTS								
		R <sub>1</sub>	R <sub>2</sub>	R <sub>n</sub>	F <sub>1</sub>	F <sub>2</sub>	P <sub>1</sub>	P <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>
ROLES	R <sub>1</sub>	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	R <sub>2</sub>		control		write *	execute			owner	seek *
	•									
	•									
	R <sub>n</sub>			control		write	stop			

# RBAC (Rol-Based Access Control)



-  El control de acceso se realiza en función de los roles del usuario en el sistema en lugar de su identidad. Un usuario puede tener diferentes roles y estos son asignados estática o dinámicamente.
-  Descansa en el principio: el propietario de la información no es el usuario si no el rol.
  - .
-  Implementa el principio de mínimo privilegio: cada rol tiene los privilegios mínimos para realizar su tarea.

# RBAC Core

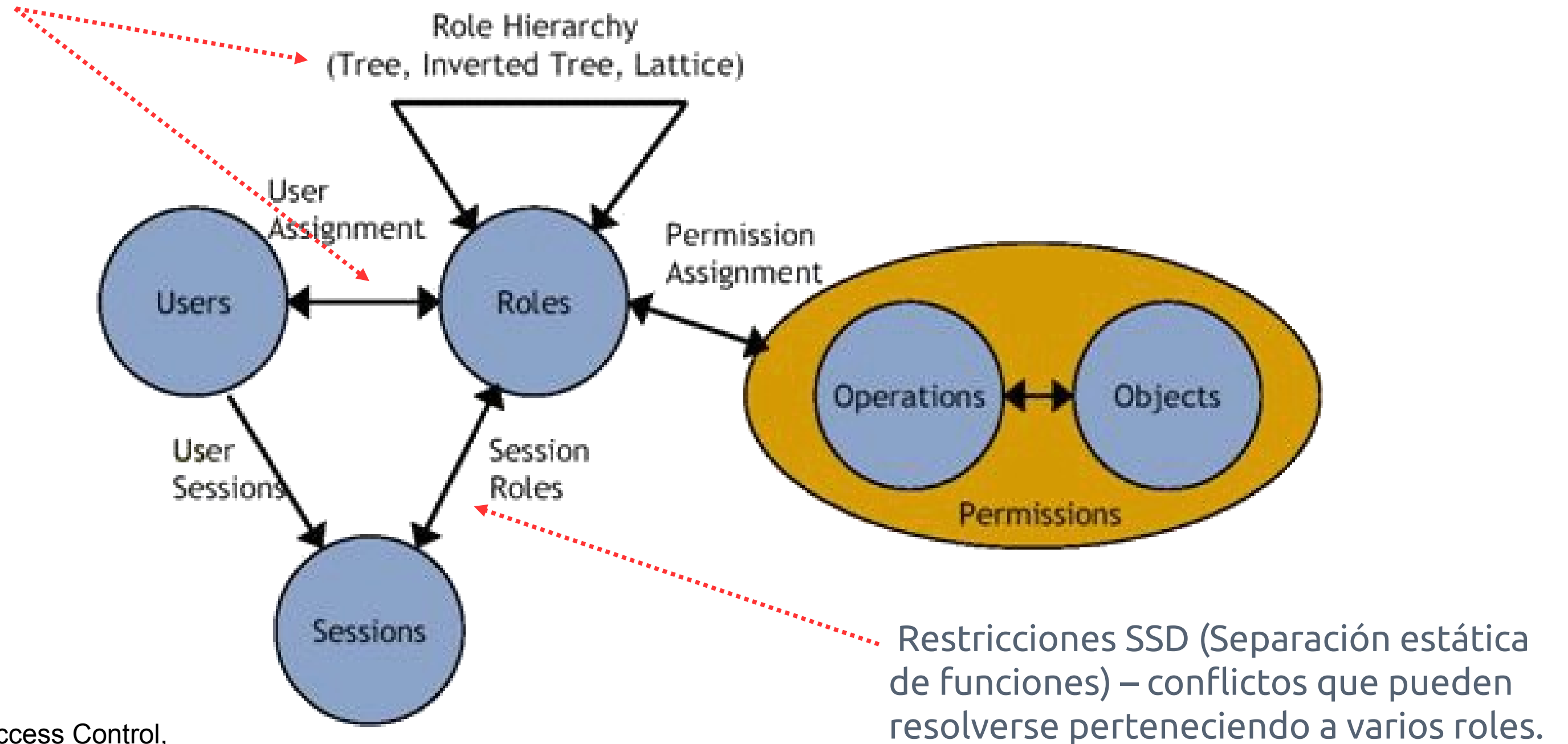


- Puede definirse como la tupla:  $RBAC=(U,O,R,P,UA,PA,user\_sesion, sesion\_rol)$ 
  - U = conjunto de usuarios
  - O = conjunto de objetos
  - R = conjunto de roles
  - P = conjunto de permisos de acceso. las actividades están ligadas a los permisos para realizarlas. Solo se asignan a un rol si es estrictamente necesario para realizar la actividad. Es una operación que se puede ejecutar sobre cierto objeto.
  - S = conjunto de sesiones, modificado dinámicamente. Sesión: representa un contexto en que se ejecutan una secuencia de actividades. Un usuario puede estar activo en más de una sesión
  - UA = asignación de usuarios (qué roles de R se asignan a un usuario)
  - PA = asignación de permisos (qué autorización de P se asigna a un rol)
  - user-sesion = par (u,s) que describe las sesiones en las que un usuario puede estar activo a la vez.
  - sesion-rol = par (s,r) que describe un usuario puede activar un subconjunto de sus posibles roles en una sesión.



# RBAC con restricciones

Restric.DSD (Separación dinámica funciones) – restricciones a la relación sesión-rol para limitar dinámicamente asignación de roles a sesiones.



Fuente: Ferraiolo, Sandhu, Gavrila:  
A Proposed Standard for Role-Based Access Control,  
2000.

# RBAC: reflexiones



- Simplifica la administración de autorizaciones.
- - En sistemas muy grandes (bibliotecas digitales, e-comercio, e-gobierno, sistemas hospitalarios):
    - La jerarquía de roles puede ser compleja.
    - Con un número alto de objetos, una asignación manual de autorizaciones es costosa y propensa a error.
    - En algunos casos el acceso depende del contexto de un objeto y del entorno del sujeto que actúa sobre él (p. ej. acceso basado en la ubicación).

•

3

SO Confiabiles

name	on	state	updb1	type	template	netvm	mem
{dom0}	*	Running	Yes	Admin	None	n/a	6776 MB
{sys-net}	*	Running		Net	fedora-21-net-interim	n/a	301 MB
{sys-firewall}	*	Running		Proxy	fedora-21-net-interim	sys-net	850 MB
{firewallvm}	*	Running		Proxy	fedora-21-net-interim	netvm	401 MB
{netvm}	*	Running		Net	fedora-21-net-interim	n/a	201 MB
debian	*	Running			debian-8	sys-firewall	850 MB
surf	*	Running			fedora-21-app	*sys-firewall	3436 MB
devel	*	Running			fedora-21-app	firewallvm	976 MB

<buongiorno signora, como estal? >

Icon size: 128x128

Failed to connect to g

=> /var/log/qubes/...torvm.log <=

libvchan\_is

=> /var/log/qubes/...torvm.log <=

```
top - 08:58 up 4:56, 7 users, load average: 0.68, 0.56, 0.43
Tasks: 3 total, 1 running, 304 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 2.8 sy, 0.0 ni, 88.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.4 st
KiB Mem: 39172 total, 4136556 used, 2802616 free, 111160 buffers
KiB S: 938044 total, 27448 used, 7910596 free, 3010636 cached
```

USER	PID	PPID	NI	U	VSZ	RES	SHR	SS	ST	TIME	COMMAND	
root	1	0	0	S	47940	5644	3848	S	0,0	0,1	0:02.30	systemd
root	660	20	0	S	76948	18720	15464	S	0,0	0,3	0:01.96	systemd-journal
root	668	20	0	S	104472	1608	1344	S	0,0	0,0	0:00.00	lvmetad
root	673	20	0	S	43240	3660	2692	S	0,0	0,1	0:00.76	systemd-udevd
root	1014	16	-4	S	51168	3224	2844	S	0,0	0,0	0:00.24	auditd
root	1026	39	19	S	16776	2476	2244	S	0,0	0,0	0:00.02	alsactl
root	1029	20	0	S	329884	27124	10416	S	0,0	0,4	0:02.32	firewalld

< whut? >

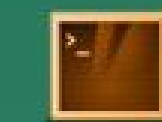
"cowsay-whut" 8L, 153C



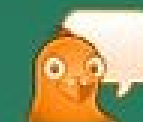
prot



shopping



conf:  
Terminal



conf:  
Pidgin



devel:  
Terminal



devel:  
Firefox



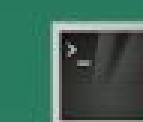
gamedeb:  
Terminal



gamez:  
Terminal



admin-far:  
Firefox



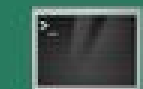
admin-far:  
Terminal



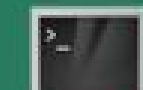
admin:  
Firefox



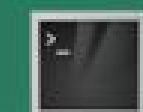
admin:  
Terminal



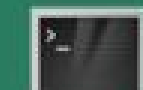
debian-7



debian-8



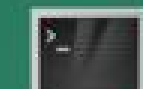
debian-8-app



fedora-21



fedora-21-app



fedora-21-net-interim



user@debian: ~





user@sys-net: ~



05:19

# MAC y sistemas operativos



-  Problema de seguridad (*safety problem*)= Problema de asegurar que un estado de protección particular y todos los estados de protección posibles futuros derivables de este estado no suministraran un acceso no autorizado.
-  Es un problema indecidible para sistemas de protección con operaciones compuestas de protección de estados. Por ejemplo: `create_file` añade tanto columnas (un nuevo archivo) como celdas (permisos) a la matriz de acceso → es imposible en general verificar que un estado de protección será seguro en el futuro.

# Sistema de protección obligatorio



Un sistema de protección obligatorio es un sistema de protección que solo puede ser modificado por un administrador confiable mediante software confiable, consistiendo en la siguiente representación de estados:

- Un estado de protección obligatorio = estado de protección con sujetos y objetos representado por etiquetas donde el estado describe las operaciones que las etiquetas de sujeto podrían asumir las etiquetas de objetos.
- Un estado etiquetado para que haga corresponder procesos y recursos del sistema en etiquetas.
- Un estado de transición que describe los medio legales mediante los cuales los procesos y los objetos recursos del sistema puede re-etiquetarse.

# Etiquetas



Las etiquetas son simplemente identificadores abstractos definidos en el sistema que representa los sujetos y objetos de la matriz de acceso.

- ▶ La asignación de permisos a etiquetas define sus semánticas de seguridad.

- ▶ Las etiquetas son inalterables (*tamperproof*) dado que:

- El conjunto de estas esta definido por administrados confiables utilizando software confiable.

- El conjunto es inmutable (administradores confiables definen las etiquetas de la matriz de acceso, estableciendo el conjunto de operaciones que los sujetos con una etiqueta particular pueden realizar sobre objetos de etiquetas particulares).

Este sistema de protección es MAC ya que es inmutable para procesos no confiables.

# Monitor de referencia



- El monitor de referencia es una forma clásica de mecanismo para la aplicación del acceso: admite como entrada una solicitud, y devuelve un binario indicando si la solicitud se autoriza por la política de control de acceso.
- Tres componentes:
  - Interfaz del monitor de referencia: define cuando se realizan las consultas al monitor de referencia.
  - Módulo de autorización: toma como entrada de la interfaz (identidad de un proceso, referencias a objetos, nombre llamada al sistema, etc.) y los convierte en consultas para el almacén de política.
  - Almacén de política: base de datos del estado de protección, estado de etiquetado, y estado de transición.

# Sistemas operativos seguros

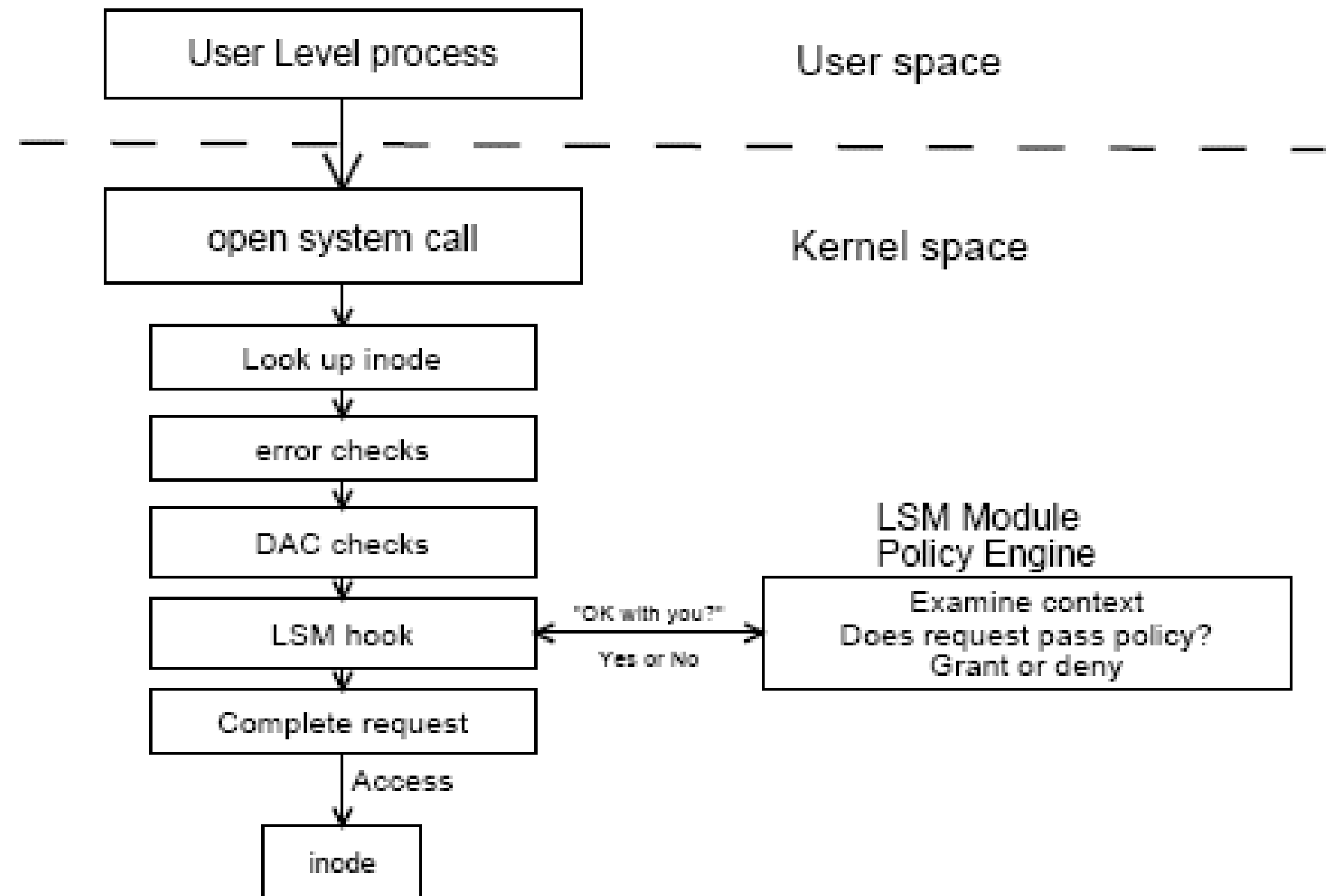


- El concepto monitor de referencia define las propiedades necesarias y suficientes de cualquier sistema que aplica seguramente un sistema de protección obligatoria, y consta de tres garantías para el mecanismo de control de accesos:
  - Mediación completa: media en todas las operaciones susceptibles de seguridad.
  - Inalterabilidad: el mecanismo, incluyendo el sistema de protección, no puede modificarse por proceso no confiables.
  - Verificabilidad: El mecanismo para garantizar accesos, incluyendo su sistema de protección “debe ser lo suficientemente pequeño para poder ser analizado y testado, de forma que podamos garantizar su completitud”.
- SO seguro = SO donde la aplicación de accesos satisface el concepto monitor de referencia.

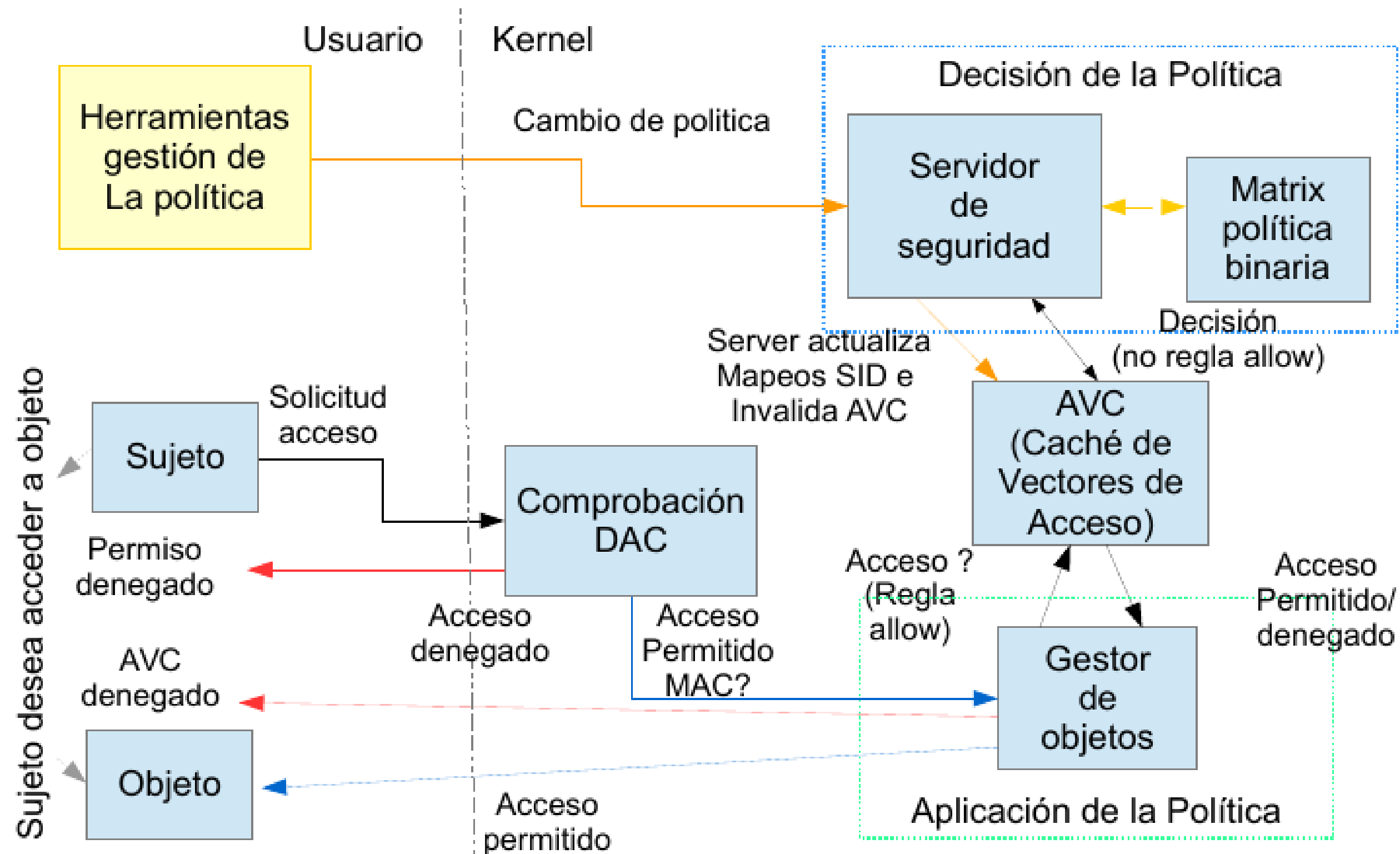


# SELinux

- SELinux (Security Enhanced Linux) se materializa mediante LSM (Linux Security Modules).
- Un gancho LSM se ejecuta tras las comprobaciones del modelo DAC y es más restrictivo.






# SELinux: arquitectura



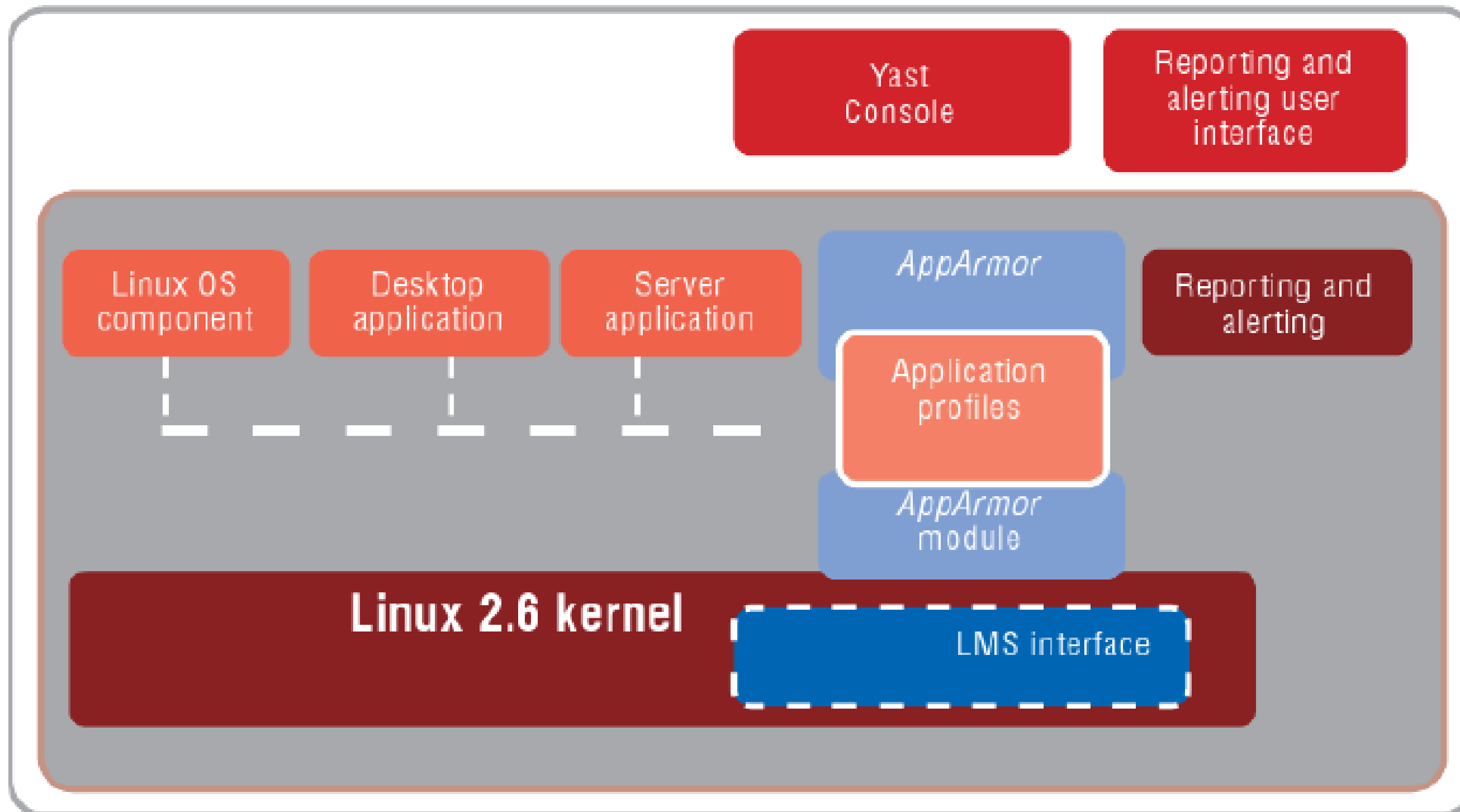
# AppArmor



-  Las principales quejas de SELinux provienen de su dificultad para instalarlo y mantenerlo.
-  AppArmor asocia con cada programa un perfil de seguridad que restrinja las capacidades del mismo, suministrando un sistema MAC. Además, incluye un modo de aprendizaje, donde las violaciones de seguridad se registran (no se previenen). Este registro sirve para establecer un perfil basado en el comportamiento del programa.
-  Se implementa usando LSM.

[http://wiki.apparmor.net/index.php/Main\\_Page](http://wiki.apparmor.net/index.php/Main_Page)

# AppArmor: arquitectura



# Sistemas Operativos centrados en seguridad

---

Algunas distribuciones de Linux centradas en la seguridad:

- **Qubes OS** – utiliza virtualización (Xen) para implementar seguridad por aislamiento o compartimentación (sandboxing). Soporta Linux, Windows y Whonix.
  - **Whonix** – soporta seguridad, privacidad y anonimato en Internet.
  - **Tomoyo** - (<http://tomoyo.osdn.jp/>) - permite que cada proceso declare los recursos y comportamiento necesarios, y vigila que se cumplan.
  - **Smack** (Simplified Mandatory Control Access Kernel) – MAC en Linux. Soporte para el SO Tizen – SO para IoT (<https://www.tizen.org/>)
- Estos sistemas se basan en el esquema LSM.

■ Más distribuciones/sistemas en [https://en.wikipedia.org/wiki/Security-focused\\_operating\\_system](https://en.wikipedia.org/wiki/Security-focused_operating_system)

# Qube OS

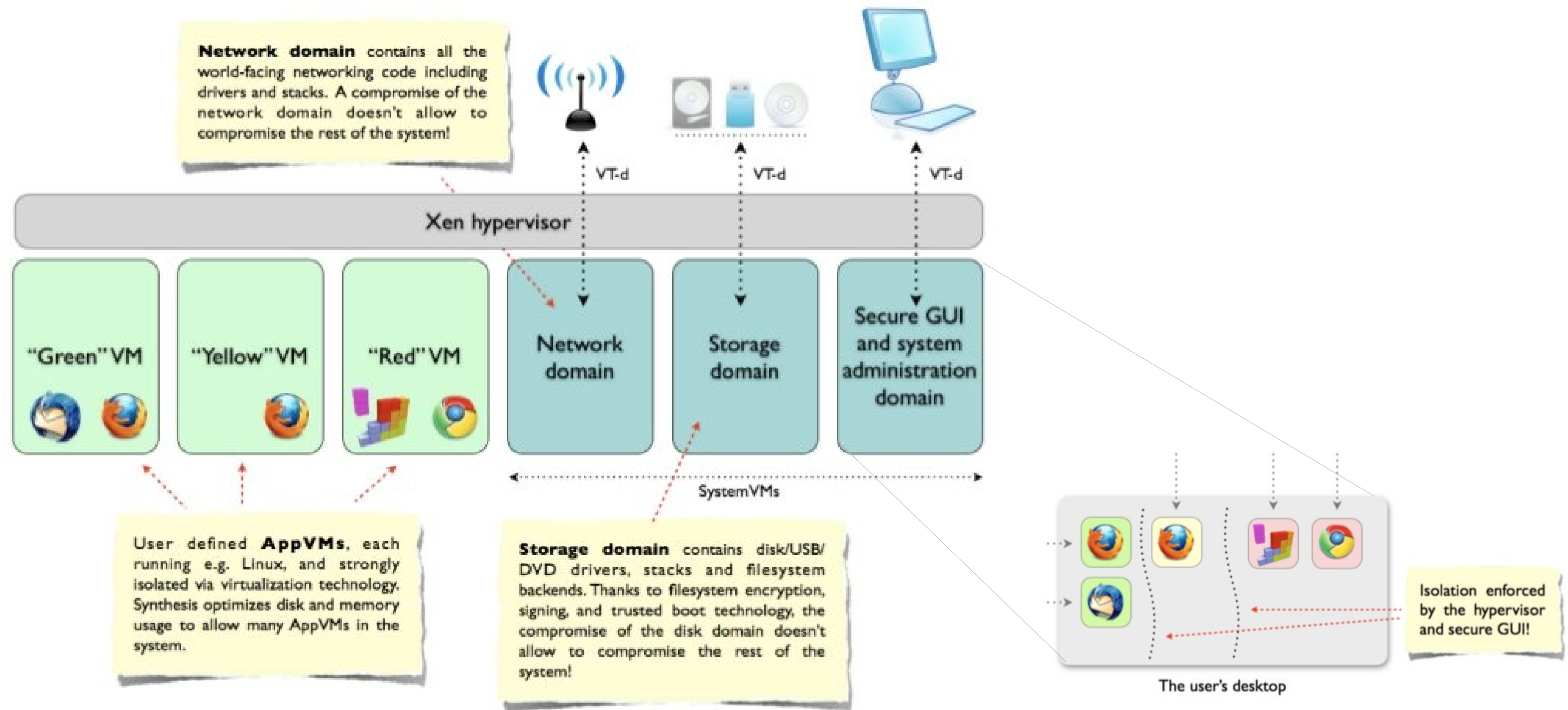


- Idea: la seguridad por aislamiento no permite acceso completo al sistema, si somos atacados confinamos el ataque a una VM (habilita el principio de mínimos privilegios).
- Basado en Xen: hipervisor nativo que gestiona la memoria y planificación de CPU de las VM's (dominios) y lanza el dominio más privilegiado, dom0.
- Integra los dominios en un desktop de usuario. Dominios separados para la red y el almacenamiento (sistemas de archivos cifrados).

<https://www.qubes-os.org/>

.

# Qube OS: arquitectura



# Whonix



- Basado en Debian, el sistema consta de dos VM: Workstation (ejecuta las aplicaciones) y Gategay (fuerza las comunicaciones vía TOR). El motor de virtualización es VirtualBox.
- La pasarela ejecuta TOR y tiene dos interfaces de red: una NAT hacia Internet, otra interna con Workstation. Esto fuerza que todas las comunicaciones de Workstation pasen por TOR. Son imposibles los *DNS leak* (uso del DNS local en lugar del anónimo), y ni el *malware* con privilegios de root puede conocer nuestra IP real.

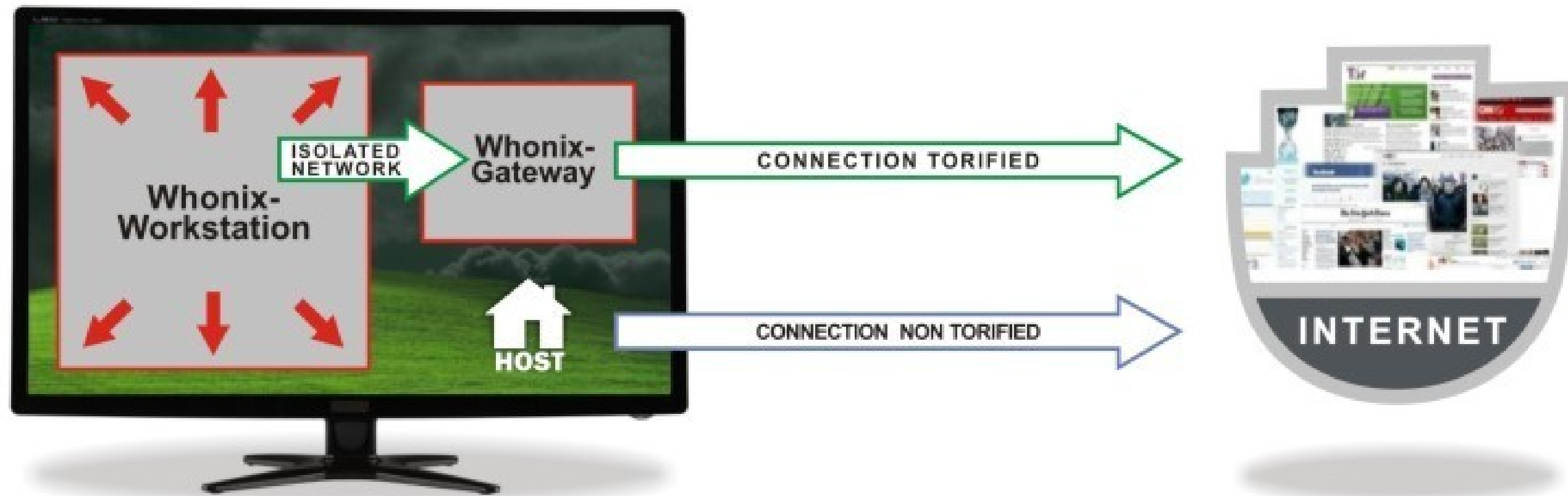
<https://www.whonix.org/>

.



# Whonix: arquitectura

## Whonix Anonymous Operating System

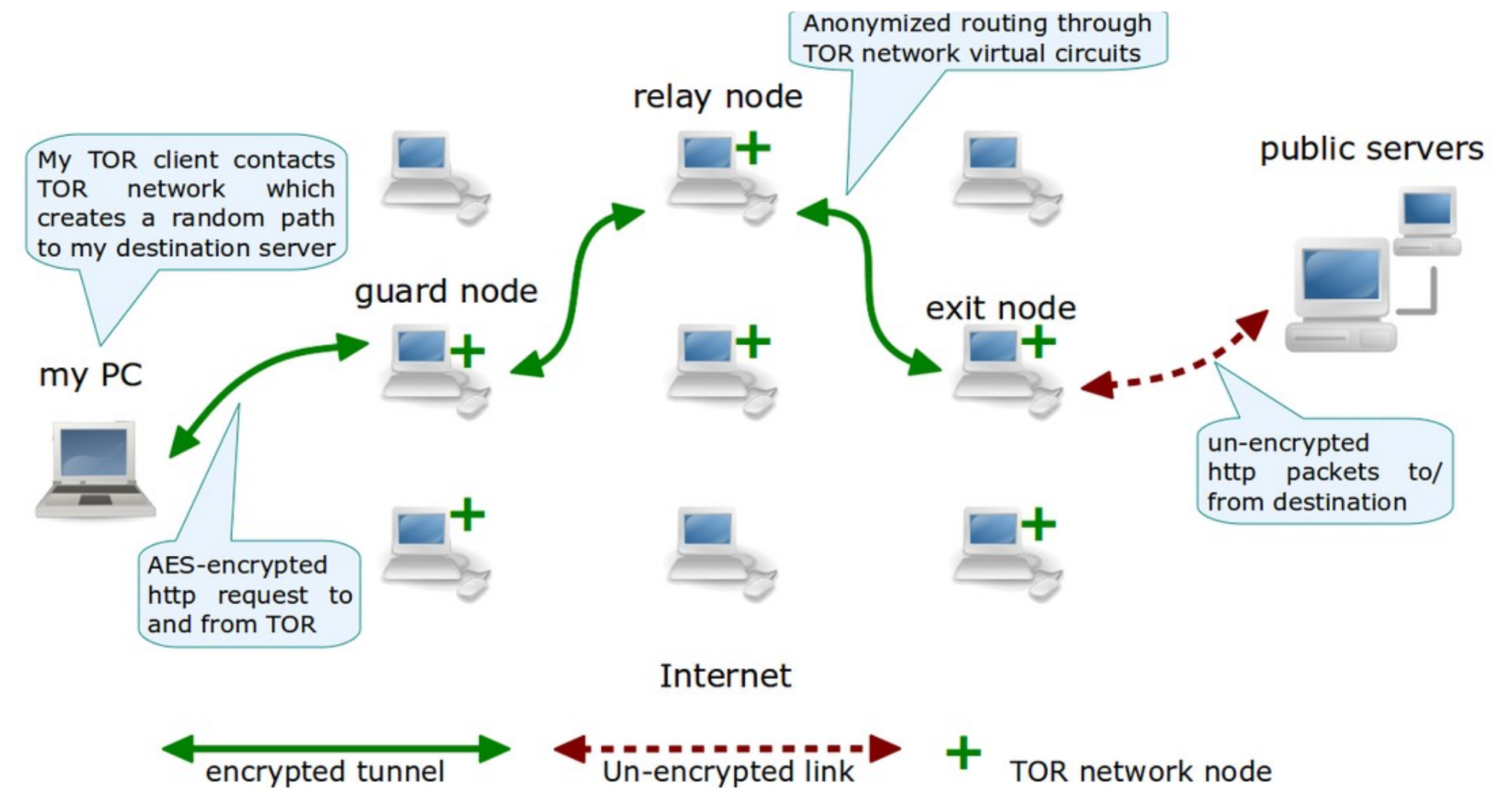


The red arrow ➤ indicate that misbehaving / leaky applications can't break out of the **Whonix Workstation**.

All network connections ➤ are forced to go through **Whonix Gateway** where they are torified and routed to the Internet.

# TOR (The Onion Router)

- Red donde el encaminamiento de los mensajes no revela su identidad (IP). Suministra anonimato a nivel de red.
- Tiene servicios ocultos (.oni) y mensajería instantánea.








# 4

Fortalecimiento  
del SO



# OS Hardening



-  El fortalecimiento del SO es un proceso para asegurar el sistema reduciendo la superficie de amenaza que es mayor cuanto más funciones ejecute.
-  Es un proceso que requiere planificación sobre los servicios a instalar en el sistema, y a reducir tras un proceso de auditoría.
-  Es un proceso que no tiene fin: el sistema no es estático y aparecen nuevos ataques, lo que provoca aumentar el fortalecimiento y su posterior auditoría.

# Planificación de la seguridad



-  El objetivo es planificar una instalación específica para maximizar la seguridad y minimizar los costes.
-  Debemos considerar:
  - ◆ Propósito del sistema, tipo de información almacenada, aplicaciones y servicios a suministrar y sus requisitos de seguridad.
  - ◆ Categorizar a los usuarios: privilegios e información a la que acceden.
  - ◆ Mecanismo de autenticación.
  - ◆ Cómo se gestiona el acceso a la información almacenada en el sistema.
  - ◆ Acceso a información almacenada en otros *host*: servidores de archivos, Bds, etc.
  - ◆ Quién administra el sistema, y cómo lo gestiona (local o remotamente).
  - ◆ Cualesquiera medidas adicionales: firewall, anti-malware, etc.




# Etapas



- Podemos abordar el endurecimiento en todas las etapas de la vida del sistema:
  - ◆ Distribución / kernel
  - ◆ Arranque del Sistema
  - ◆ Funcionamiento regular del sistema
  - ◆ Aplicaciones



# Seguridad de la distribución / kernel



-  La instalación de una nueva distribución/kernel para por obtener las fuentes/binarios de la distribución/kernel e instalarlos.
-  Las posibles amenazas a:
  - ◆ Los fuentes/distribución
  - ◆ Control de descargas – comprobar hash o firma
  - ◆ Compilar el kernel: herramientas y opciones de compilación
-  Algunas características de las distribuciones pre-compiladas están pobremente configuradas respecto a la seguridad.

# Instalación





-  En la instalación de una distribución debemos considerar:
  - ◆ Habilitar claves shadow con hash
  - ◆ Elegir una clave fuerte para el root
  - ◆ Crear un usuario adicional con una clave fuerte
  - ◆ Habilitar un firewall
  - ◆ No instalar paquetes que no sean necesarios: juegos, servidores de red, herramientas de desarrollo, de impresión, ...
  
-  La instalación debe hacerse con el sistema desconectado de red.



# Seguridad en el arranque



-  Un atacante con acceso físico al sistema puede evitar los mecanismos de seguridad del sistema y manipular el proceso de arranque para permitir el mismo desde un cd/dvd, pendrive o HDD.
-  Esto puede crear dos problemas:
  - ◆ Algunos sistemas permiten ciertos accesos a quienes pueden arrancar la máquina.
  - ◆ No poder arrancar la máquina es un perfecto ataque DoS.

# Seguridad en el arranque (ii)





Para evitar los problemas anteriores:

- ◆ Proteger la BIOS-UEFI.
- ◆ Proteger el cargador de arranque
- ◆ Deshabilitar los arranques desde otros medios.
- ◆ Proteger los servicios que acceden a la red en el arranque. Por ejemplo, arrancar antes de conectar a la red iptables y syslog (apagar al revés).

# Seguridad en operación



-  Los elementos a configurar son:
  - ◆ Eliminar todo el software innecesario, en especial herramientas de desarrollo
  - ◆ Eliminar/desactivar los servicios no necesarios.
  - ◆ Alterar las cuentas por defecto y los mecanismos de acceso.
  - ◆ Establecer el principio de menor privilegio asignado roles administrador/usuario y permisos sobre archivos.
  - ◆ Aplicar las actualizaciones de software
  - ◆ Implementar registro de eventos y auditoría.
  
-  Precaución: dado que algunos de estos cambios pueden tener efectos no previstos, estos cambios deben experimentarse en una máquina de pruebas (no de producción).





# Actualizaciones



- Uno de los elementos más críticos de la seguridad son las actualizaciones del SO y las aplicaciones ya que nuevos ataques requieren nuevos parches.
- Son necesarias las mismas precauciones que en la instalación:
  - ◆ Comprobar la integridad de los mismos
  - ◆ Instalar el máquina de pruebas desconectada de la red.




# Estándares y guías



-  CIS (*Center for Internet Security*) – establece un modelo de consenso para definir recomendaciones de formalecimiento.
-  FDCC (Federal Desktop Core Configuration) – establece la referencia para SO de desktop del gobierno de EEUU.
-  NSA Security Configuration Guides (<http://csrc.nist.gov/groups/SNS/checklists/>).
-  CIS Benchmarks (<https://benchmarks.cisecurity.org/downloads/browse/index.cfm?category=benchmarks.os.linux>)

# Herramientas



-  Fáciles de usar, no hay gran variedad y pueden estar desactualizadas o limitadas en soporte.
-  En Linux: SCAP, Lynis, Bastille, tripwire, ...
-  En Windows: Secunia Online Software Inspector, Microsoft Baseline Security Analyzer (MBSA).