Socarrat en mi Raspberry Pi

Aunque rasques, no podrás borrar estos ficheros

Gorka Guardiola Múzquiz
gorka.guardiola@urjc.es
Enrique Soriano Salvador
enrique.soriano@urjc.es

GSyC, URJC

5 de marzo de 2025

who

Gorka Guardiola Múzquiz: Profesor Titular en la Universidad Rey Juan Carlos de Madrid. Estudió Ingeniería de Telecomunicación en la Universidad Carlos III y tiene un doctorado en Ingeniería informática en la Universidad Rey Juan Carlos. Investiga en sistemas operativos, sistemas embebidos, programación concurrente, seguridad y matemáticas puras y programa software libre cuando se lo permiten sus obligaciones. Mas información: `https://github.com/paurea` y `http://paurea.net`

Enrique Soriano-Salvador: Doctor en Informática y actualmente Profesor Titular en la Universidad Rey Juan Carlos de Madrid. Sus líneas de investigación se centran en el software de sistemas y la ciberseguridad. Lleva usando, administrando y programando sistemas de tipo Unix (principalmente Linux) desde 1997. Más información: `https://gsyc.urjc.es/esoriano`

tl;dr ... Unix geeks!

¿Socarrat?

- ¿Alguna vez has tenido que limpiar una paella con el arroz quemado?
- Queremos que te cueste lo mismo o más borrar/modificar los datos de los logs después de asaltar un sistema.

(esto es lo que piensa chatgpt sobre el socarrat)

WORM (Write Once Read Many)

- Dispositivos `WORM` (Write Once Read Many): **una vez escrito un dato, no es posible modificarlo o borrarlo**.
- Es un concepto muy sencillo, pero **muy difícil** de implementar.

WORM: hw especializado (antiguo)

Impresoras (un clásico):

WORM: hw especializado (antiguo)

Linear Tape Open (cintas):

WORM: hw especializado (antiguo)

Discos ópticos WORM:

- P. ej. DEC RV20 laser drive con discos RV02K-01 de 2 GB (1988):

Digital's RV20 laser drive and RV02 media, a fully integrated storage subsystem for the VAX environment, is the first member of a Digital product family based on the rapidly maturing write-once-read-many (WORM) optical technology.

WORM: hw especializado (antiguo)

Discos ópticos WORM:

- P. ej. Sony WDD, 6.55 GB por cara (1992):

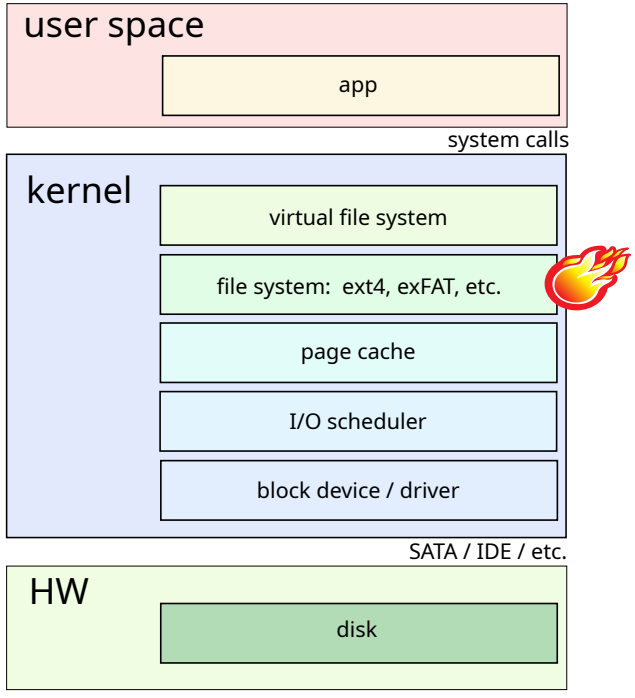
WORM: hw especializado (antiguo)

Jukeboxes de discos ópticos:

- P. ej. Sony WDA Writable Disk Auto Changer (1992):

WORM: sw convencional

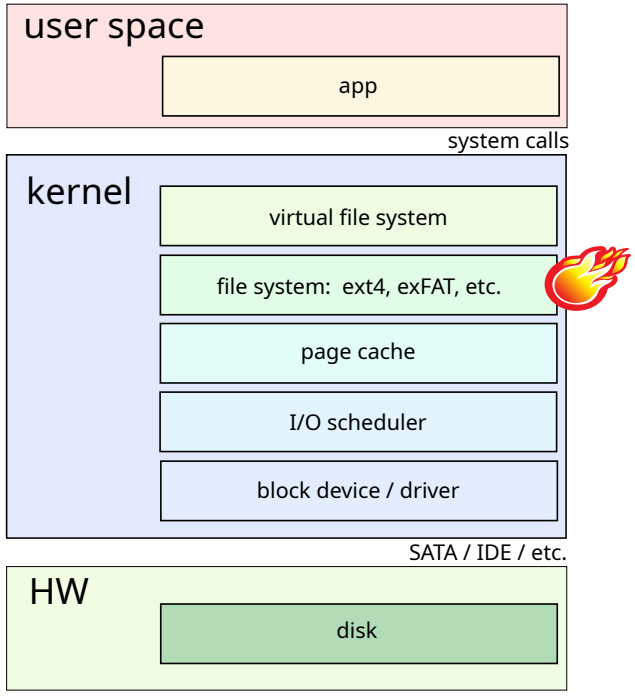
- Cualquier solución que se base en características del sistema de ficheros no nos asegura que un adversario que se hace root no pueda alterar los datos escritos, porque puede hacer lo que quiera con la interfaz del sistema de ficheros (llamadas al sistema):
  - Quitar los atributos **append-only**, **immutable**, etc.
  - Borrar o truncar los ficheros.
  - ...



WORM: sw convencional

Como root también puede:

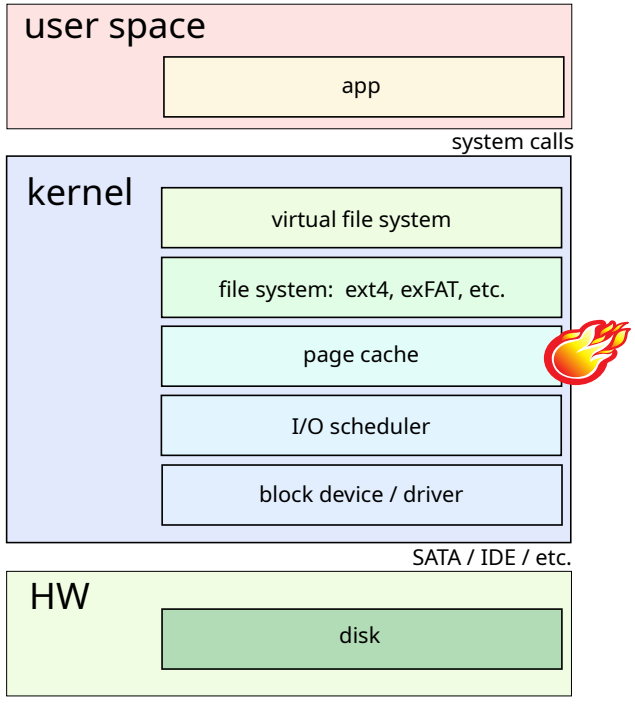
- Cambiar las direcciones de los bloques de datos del fichero en las estructuras de datos (p. ej. inodos) del sistema de ficheros saltándose la interfaz.



WORM: sw convencional

Como root también puede:

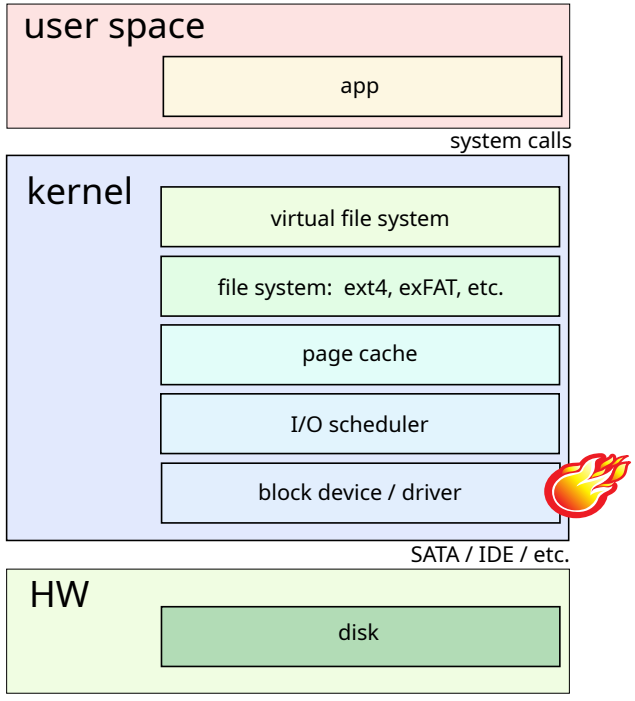
- Modificar los bloques correspondientes de la cache.



WORM: sw convencional

Como root también puede:

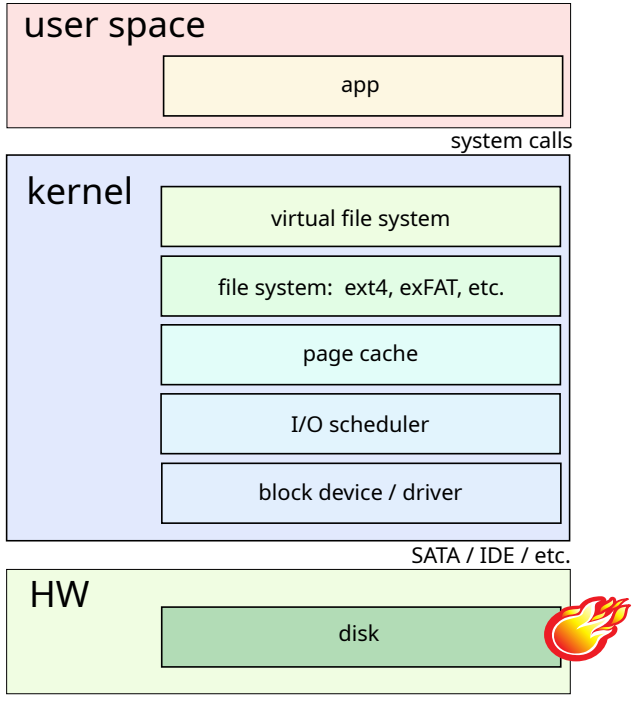
- Modificar los bloques directamente del dispositivo de bloques (p. ej. `/dev/sda`).



WORM: hw convencional

Cualquier solución que se base en hardware convencional, si el atacante tiene acceso físico puede:

- Modificar los bloques directamente en el hardware sin que podamos darnos cuenta de que han sido manipulados.



WORM: hw especializado (actual)

- Suelen ser soluciones **caras y cerradas**. Es difícil hasta saber su precio y el procedimiento de compra.
- Sony sacó una tercera generación de unidades ópticas de desde 5 TB por cartucho, con racks de 2.9 PB en 2020. Está descatalogada.
- Todavía hay cintas. P. ej. IBM LTO Ultrium 9.
- Algunos fabricantes poco conocidos ofrecen discos WORM, pero dan **poca información sobre su uso y la tecnología que usan**. P. ej. Greentec ofrece unos discos llamados WORMdisk ZT Storage para sistemas de ficheros NTFS, exFAT, FAT32, UDF, ext3, ext4... no se puede ni ver el precio ni ver qué garantías ofrecen. P. ej. Flexxon's Write-Once-Read-Many (WORM) SD Cards.
- Mucha investigación en nuevos materiales para crear memorias WORM que todavía no están en el mercado.

WORM: soluciones distribuidas (actual)

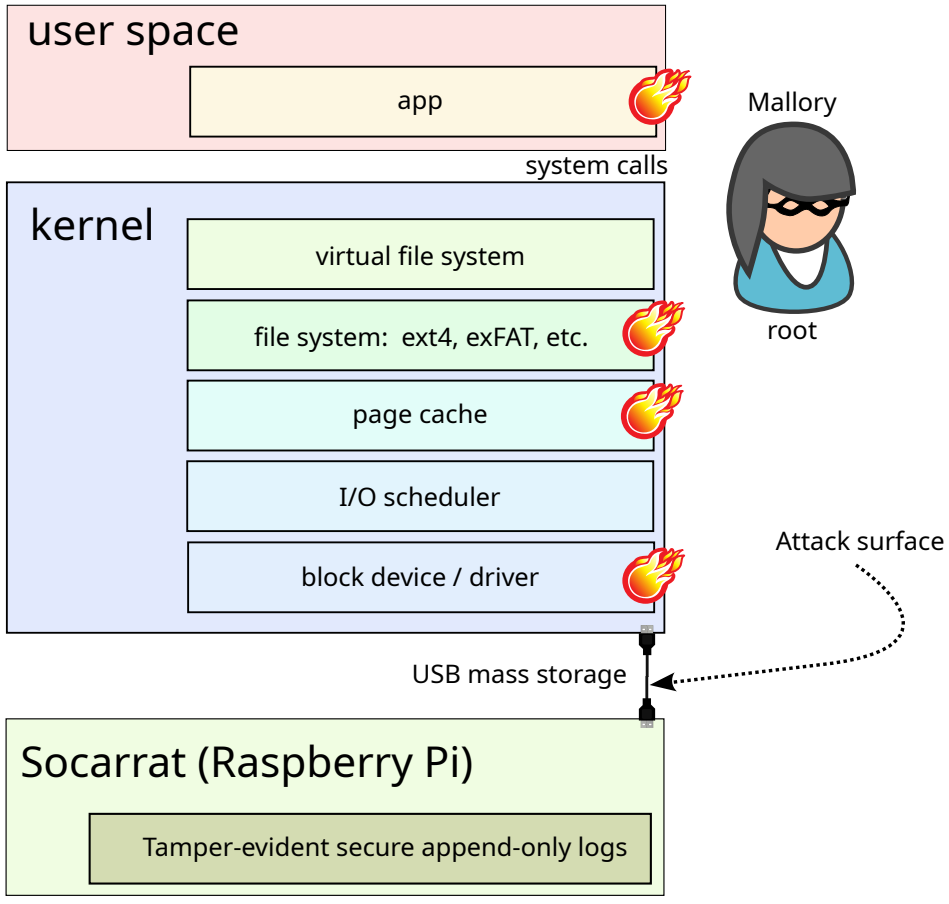
- Sistemas distribuidos para enviar los logs a otras máquinas → trasladar el problema a otra máquina. Problema: DoS.
- Sistemas de ficheros en red con capacidad WORM. P. ej. NetApp SnapLock se puede usar por NFS o SMB y soporta ficheros que crecen en bloques fijos de 256 KB.
- ¡La nube! Le damos los logs a otro, si puede ser controlado por un país hostil. P. ej. AWS S3 Object Lock proporciona WORM. ¿Soporta append? No.
- Blockchain...

Motivación

Queremos hacer un dispositivo WORM **local** que:

1. Pueda usarse **en cualquier máquina convencional** para tener ficheros de log que puedan ser usadas normalmente por cualquier aplicación.
2. Se monte como si fuera un **disco USB corriente** con un sistema de ficheros ext4 o exFAT.
3. Ofrezca ficheros de log **append-only** que soporte un uso continuo aceptando escrituras de cualquier tamaño al final de los ficheros de log.
   - Ficheros **inmutables** (`+i`) vs. ficheros **append-only** (`+a`)
4. Ofrezca **garantías de seguridad** ante intrusiones y sea **tamper-evident**.
   - Si los datos de los logs llegan a ser modificados, se detectará la manipulación en una auditoría (el usuario o un auditor)
5. Sea **barato**, **abierto**, **libre**, y que lo puedas montar tú mismo (**DIY**).
   - Raspberri Pi 4 Model B \$69 + HDD 500GB \$30 = \$99

Socarrat: idea



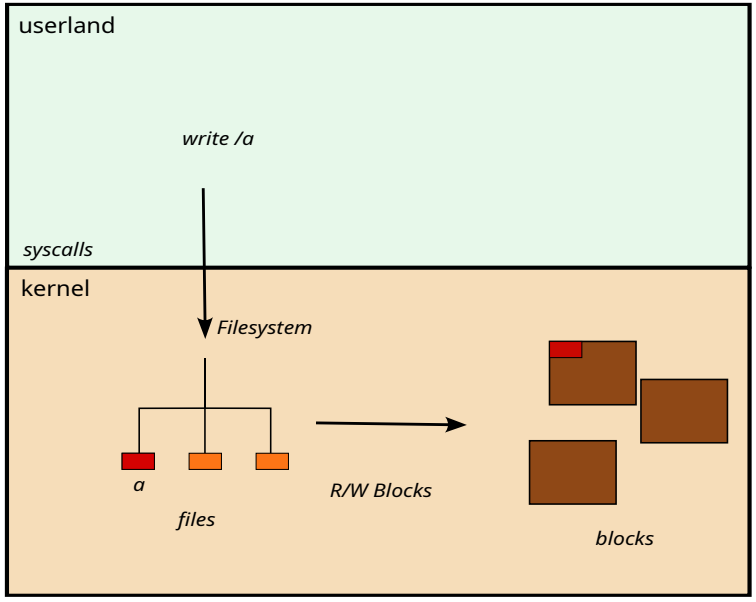
¿Cómo controlamos los ficheros de log?

- La Raspberry Pi ejecuta Linux. ¿Cómo controlamos los ficheros de log desde dentro? **No es fácil**.
- **Idea feliz:** montamos la misma partición desde dentro de la Raspberry Pi y desde fuera, así controlamos lo que se escribe en los ficheros de log.
- **No funciona**: no podemos tener montado el mismo sistema de ficheros en dos sistemas a la vez $\rightarrow$ **concurrencia**.

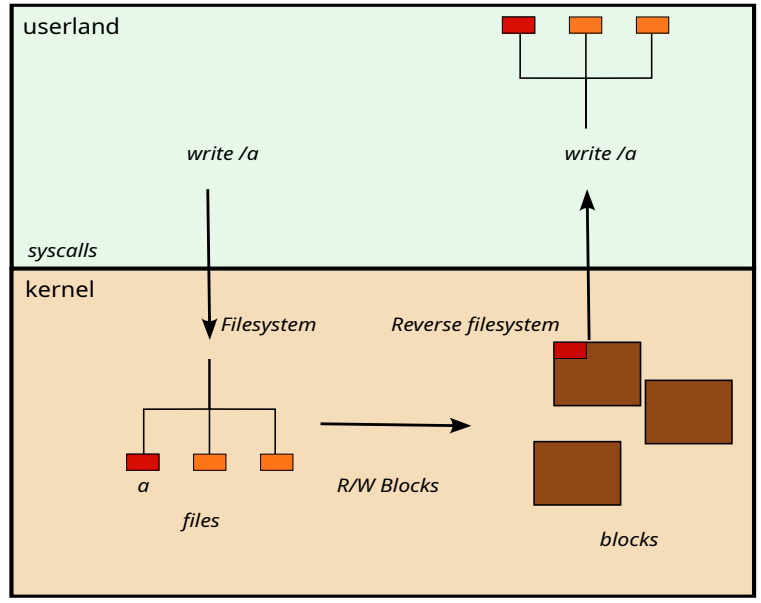
Idea principal: Reverse File System

- **Reverse File System**:
  - Proporcionamos un dispositivo de bloques
  - Analizamos los bloques que nos llegan para escribir
  - Inferimos qué operaciones se están realizando sobre el sistema de ficheros que está en niveles superiores.
  - *Neutralizamos* todas las operaciones que intentan borrar o modificar los datos que ya están escritos en el log.
- Socarrat tiene que analizar los bloques que se escriben y saber qué operaciones del sistema de ficheros se están haciendo sobre los logs $\rightarrow$ hay que saber interpretar las estructuras internas del sistema de ficheros (superbloque, inodos/clusters, etc.).

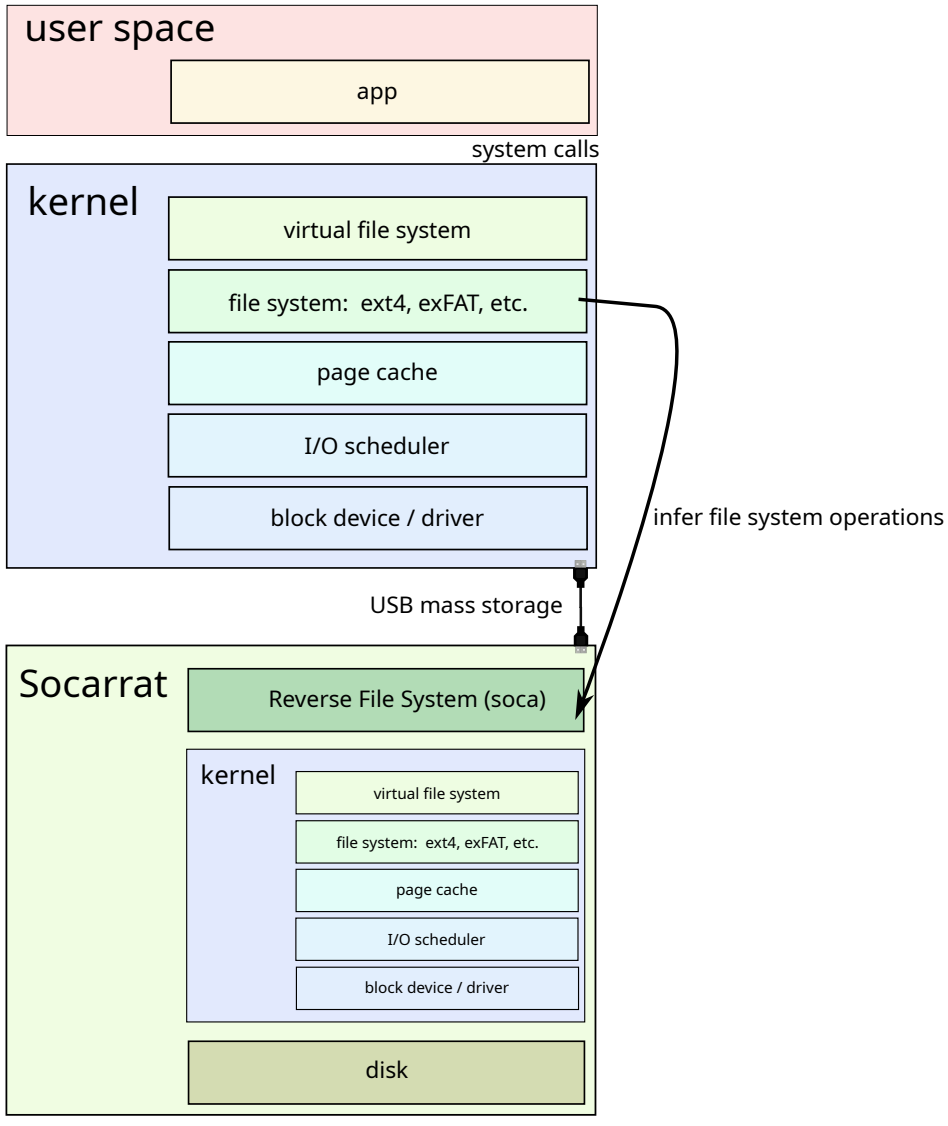
Regular FS



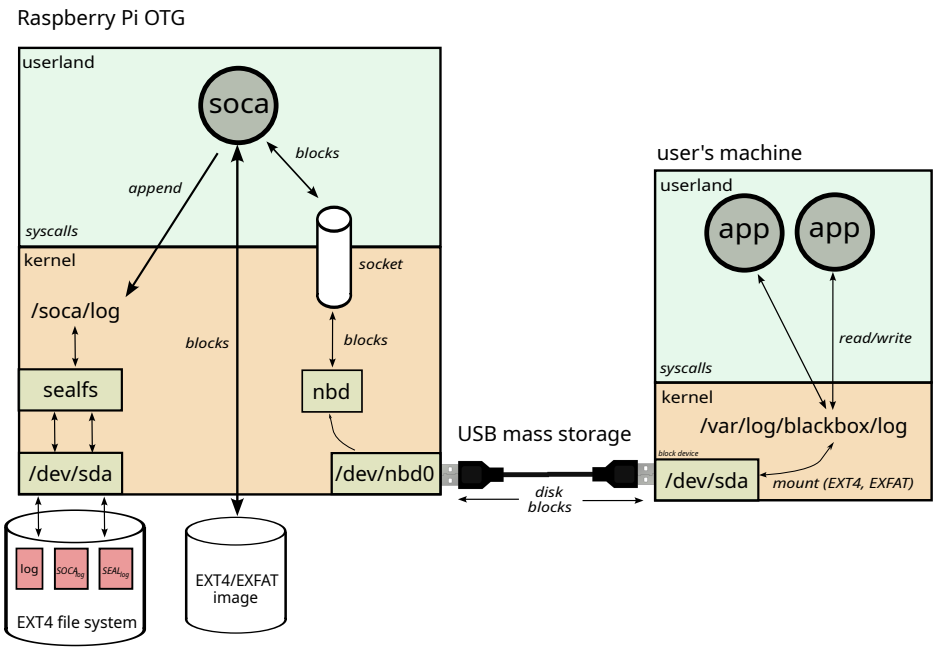
Reverse FS



Reverse File System



Socarrat: Arquitectura



- **soca** es el principal componente, un programa escrito en Go que implementa la idea del Reverse File System.
- **nbd** es el protocolo que se usa para exportar el dispositivo de bloques al cliente a través de una conexión USB. Se comunica con soca mediante un Socket de Unix.
- Los bloques servidos pertenecen a un fichero de imagen formateado (en la configuración) como ext4 o exFAT.
- Los ficheros de log *reales* **no viven dentro de la imagen**, viven en el sistema de ficheros privado de la Raspberry Pi. Soca los actualizará cuando deba (sólo en los appends).
- Los ficheros de log *reales* están además protegidos con **SealFS** [1,2], que proporciona las garantías *tamper-evident* mediante otro fichero de metalog llamado $SEAL_{log}$.
- **soca** también mantendrá su propio metalog autenticado por SealFS, $SOCA_{log}$ $\rightarrow$ en la auditoría podremos verificar las cosas que se han detectado durante el Reverse File System.

Referencias:

[1] SealFS: Storage-Based Tamper-Evident Logging. Enrique Soriano-Salvador, Gorka Guardiola Múzquiz. Computers and Security. Vol 108. Sep. 2021

[2] SealFSv2: Combining Storage-Based and Ratcheting for Tamper-evident Logging. Gorka Guardiola Múzqiz, Enrique Soriano-Salvador. International Journal of Information Security, Vol 22. 2023

Socarrat: Garantías

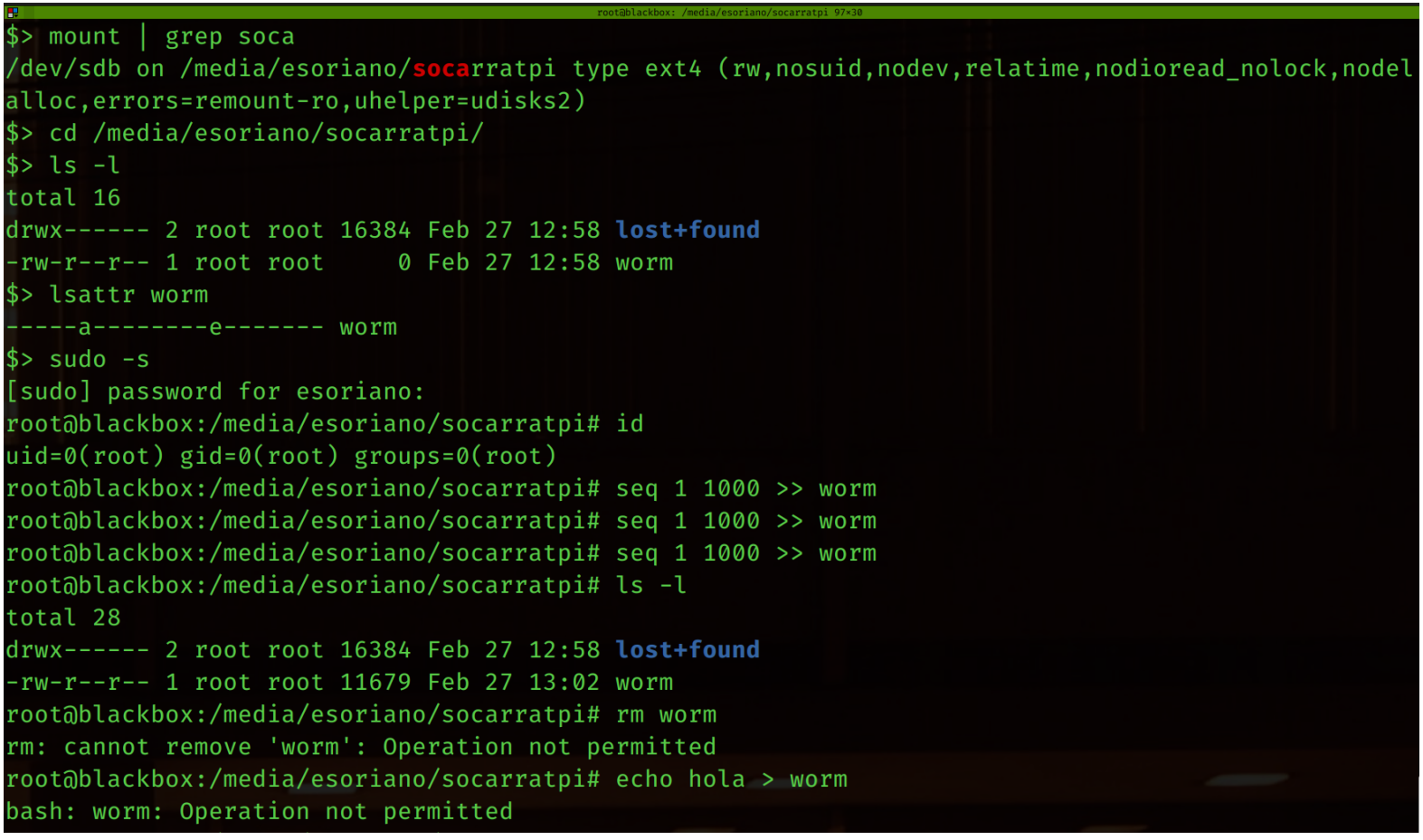
- **Continuous Printer Model (CPM)**:
  - Una vez comprometida una escritura de **append**, no puede ser borrada o modificada en el log real.
  - **Liveness**: comprometemos los datos con suficiente frecuencia como para que el sistema sea operativo, asegurando que las escrituras no esperan para siempre.
  - Después del compromiso de la máquina cliente, se podrán comprometer datos nuevos (maliciosos) en los logs, pero el primer punto se sigue garantizando:

    lo que se ha impreso no se puede "desimprimir"

    Lo mismo aplica al resto de consecuencias de un ataque (llenado de disco, paradas por fallo, etc).

Socarrat: Garantías

- No se podrán hacer cosas malas por accidente (el fichero de log se llama `worm` en este ejemplo):



Socarrat: Políticas

- Socarrat puede reaccionar a indicios de compromiso:
  - Cambiar los metadatos de los ficheros de log.
  - Llenado del dispositivo.
  - Escribir en una parte que no sea el final del fichero.
  - Modificar partes de las estructuras de datos del sistema de ficheros (p. ej. partes del superbloque).
  - Formatear la partición.
  - ...
- ¿Cómo reacciona?

Socarrat: Políticas

Vamos a definir dos políticas a elegir en la configuración para reaccionar ante indicios de compromiso:

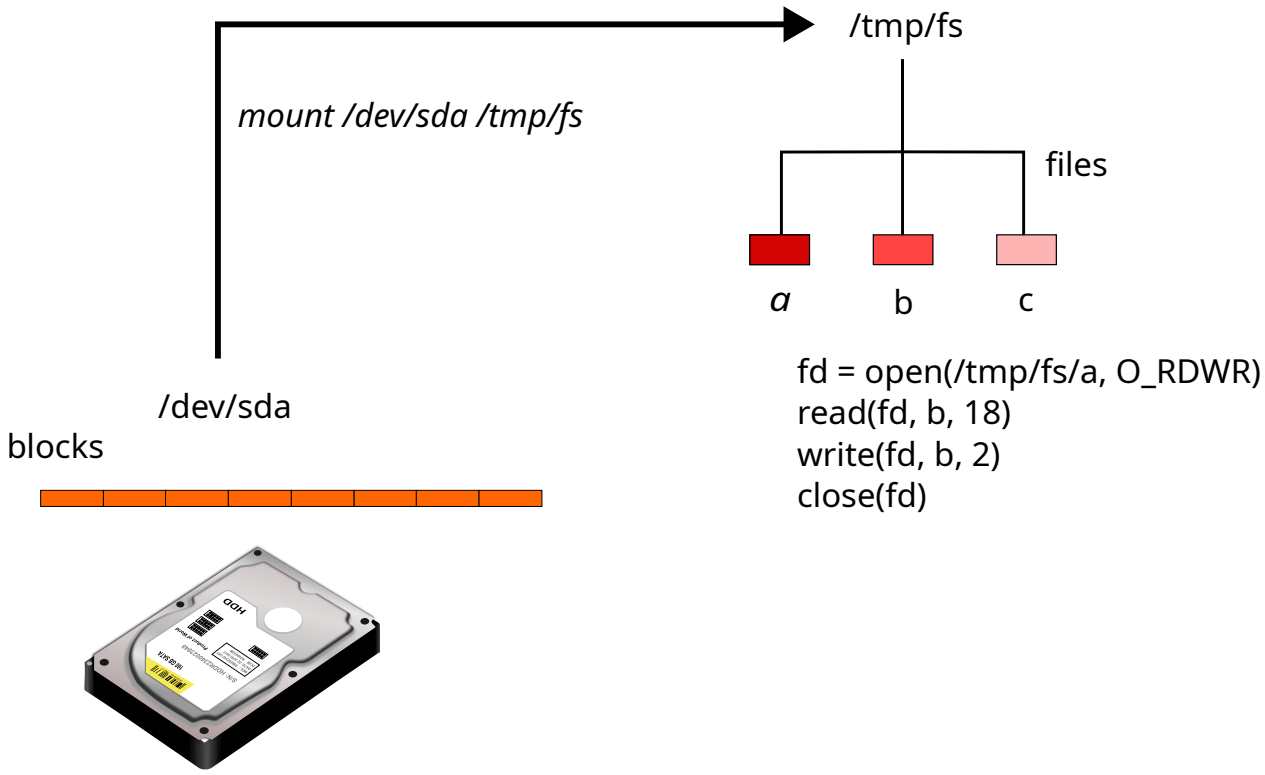
A) Parar y remontar en modo **read-only**.

B) Congelar los logs reales y seguir aceptando las lecturas/escrituras de bloques de la máquina cliente $\rightarrow$ propiedades de *honeypot*:

- El atacante sigue modificando los ficheros de la imagen a su antojo y sin sospechar nada, (incluidos los ficheros de log de dentro de la imagen).
- En la auditoría, podremos hacer un `diff` entre los logs reales y los logs de dentro la imagen, para ver qué ha querido modificar o ocultar el atacante durante la intrusión.

Interfaz de un disco

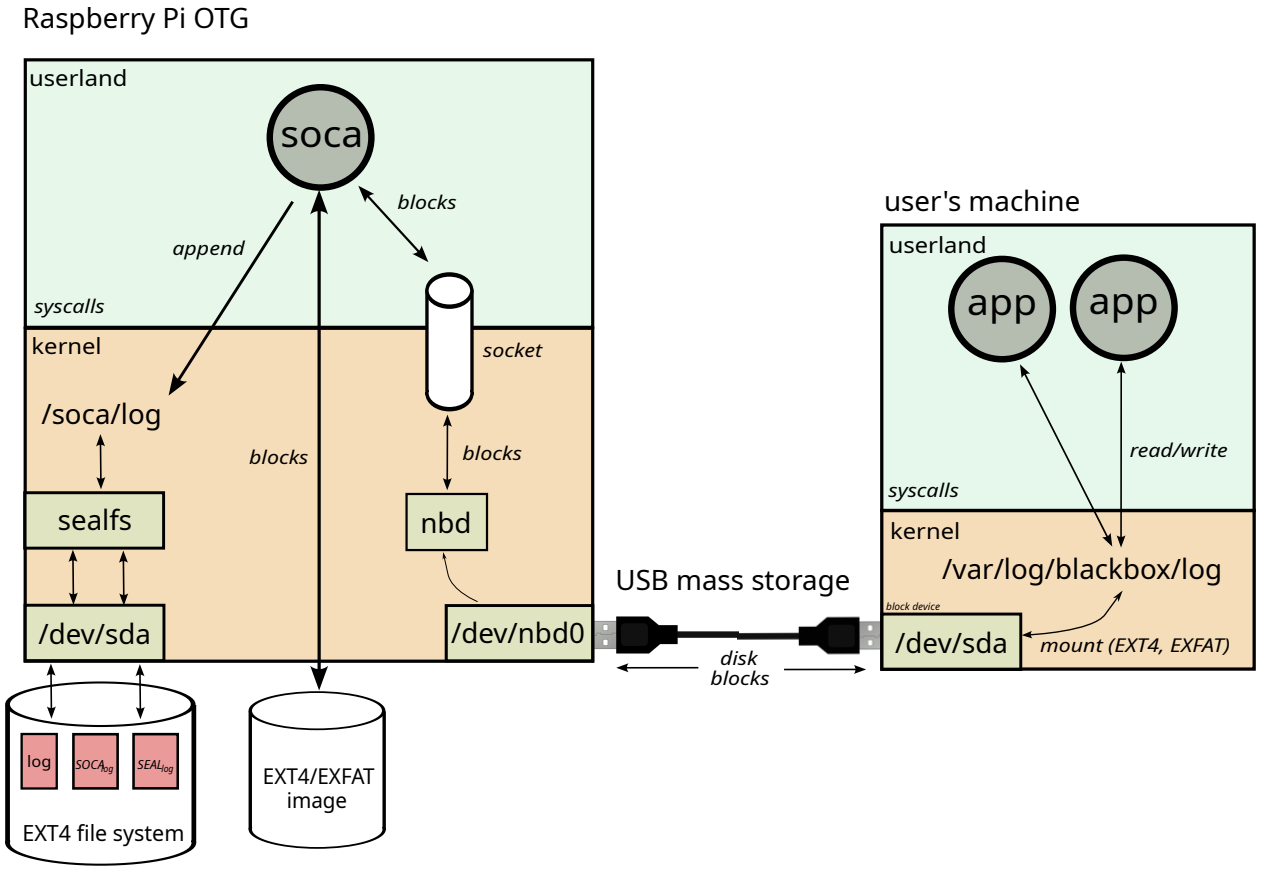
- Un disco en UNIX: dispositivo de bloques, ej: `/dev/sda`
- Cuando escribes/lees: se lee/escribe algo de tamaño de un bloque (por debajo, lee el bloque, lo modifica y lo escribe).
- Pero para el usuario: un fichero especial que puede montar.
- `mount /dev/sda /tmp/fs`



Dispositivo de bloques

NBD

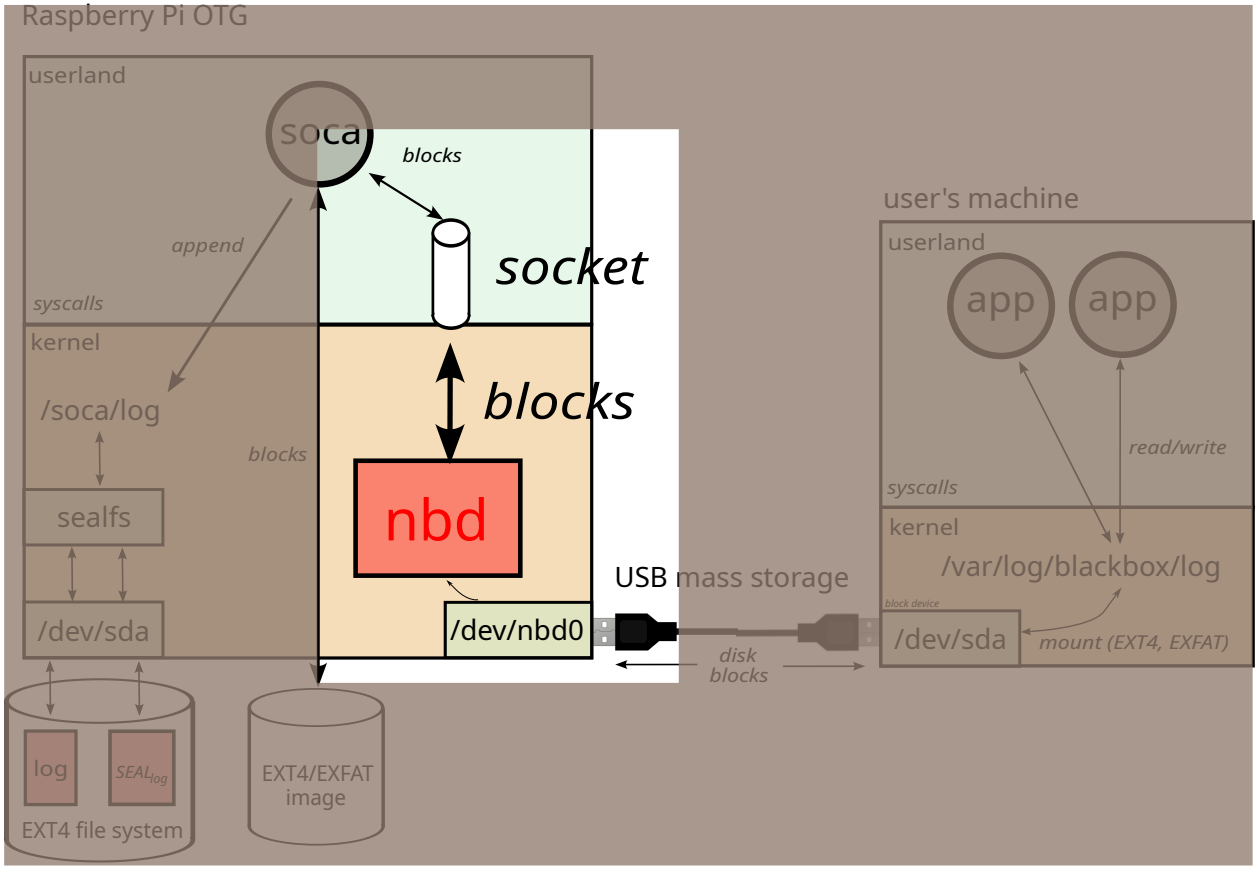
- En la arquitectura general



NBD

NBD

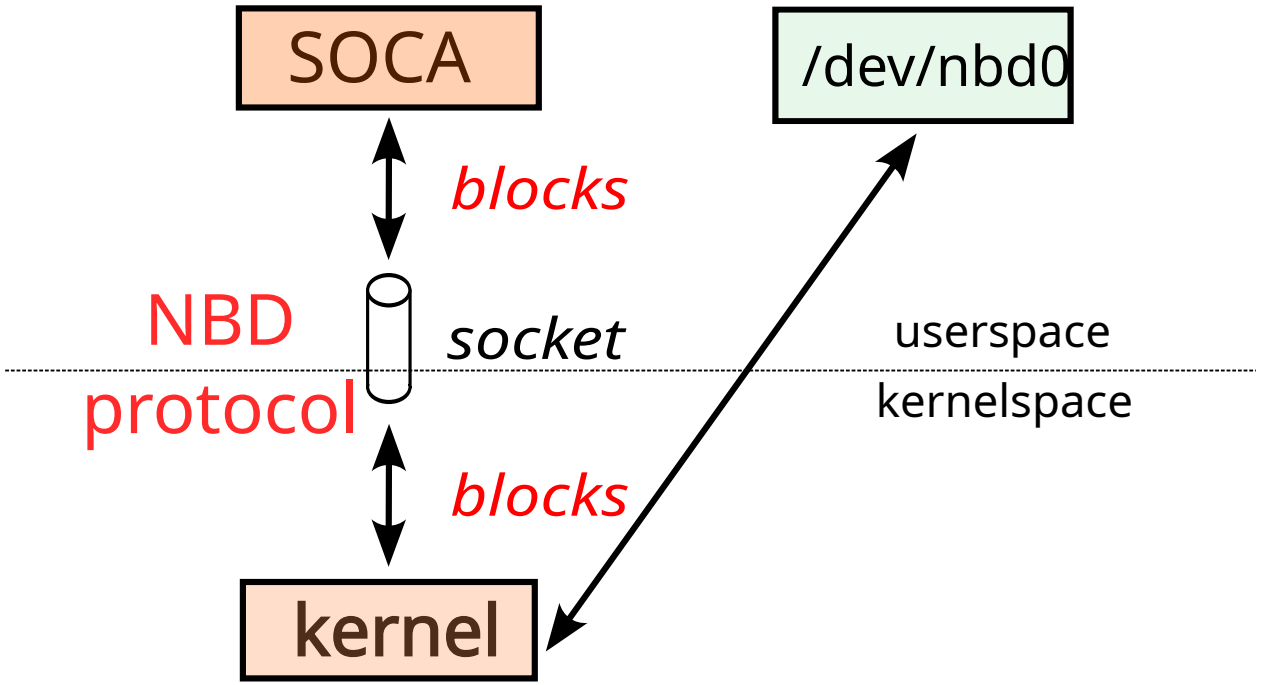
- Esto de aquí es NBD



NBD

Qué es NBD

- Protocolo para servir bloques.
- Un socket con el kernel.
- Tiene una negociación (tamaño de bloque y mil parámetros) y luego operaciones sobre bloques.
  - Read bloque
  - Write bloque
  - Sync
  - Desconectar.



Qué es NBD

- Nuestro programa cada vez que recibe un bloque.
- Lo inspecciona y decide, dependiendo del sistema de ficheros (inverso).

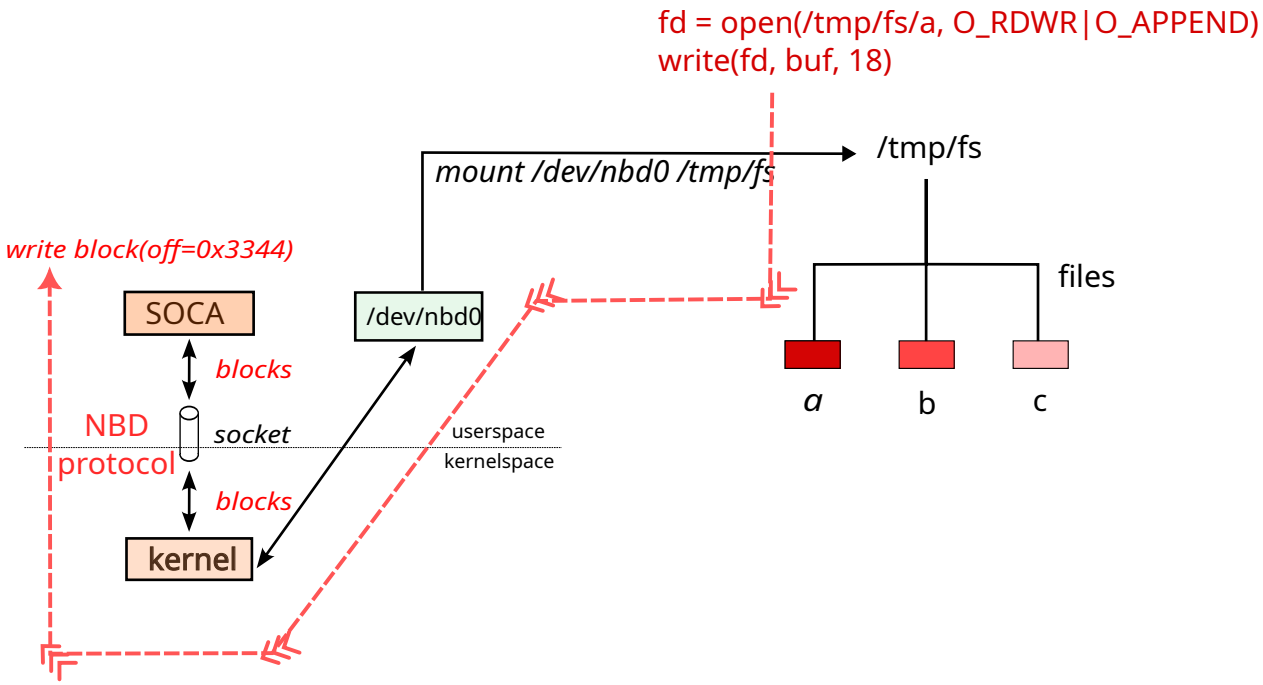
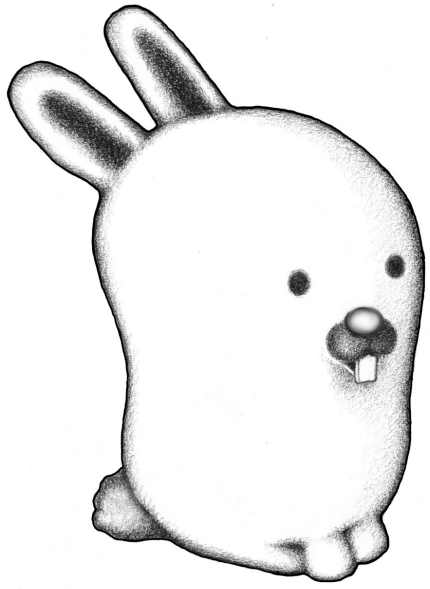


Diagrama bloques inspeccionados

Protocolos de bloques

- Soportamos NBD y 9P



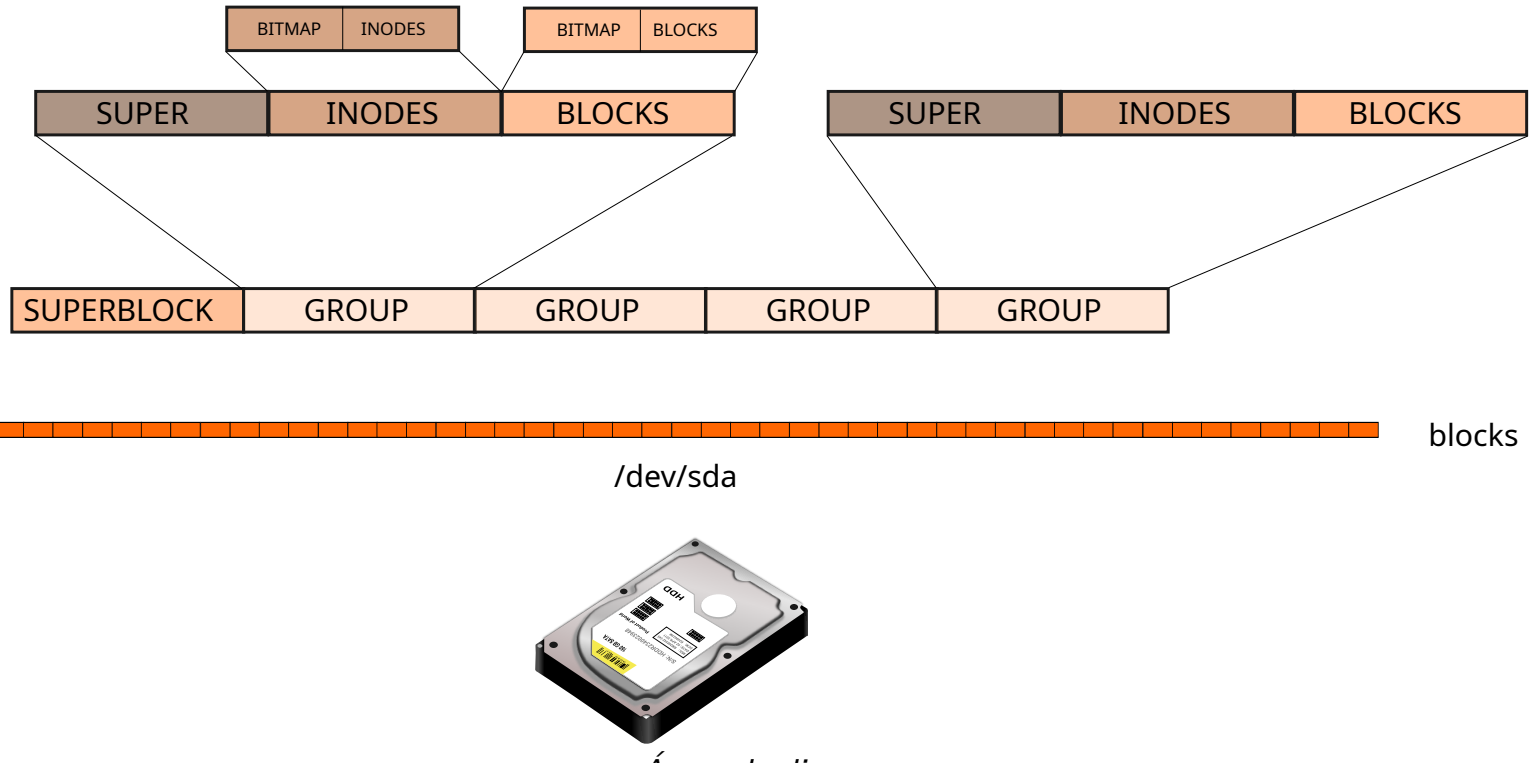
NBD

Qué es un sistema de ficheros

- Estructura de datos en los bloques de disco.
- Se escribe un árbol inicial con sólo el raíz (formateo).
- Luego el que escribe es el kernel haciendo crecer el árbol (cuando está montado).
- Soportamos dos: ext4 y exFAT, vamos a ver ext4.

Ext4: áreas de disco

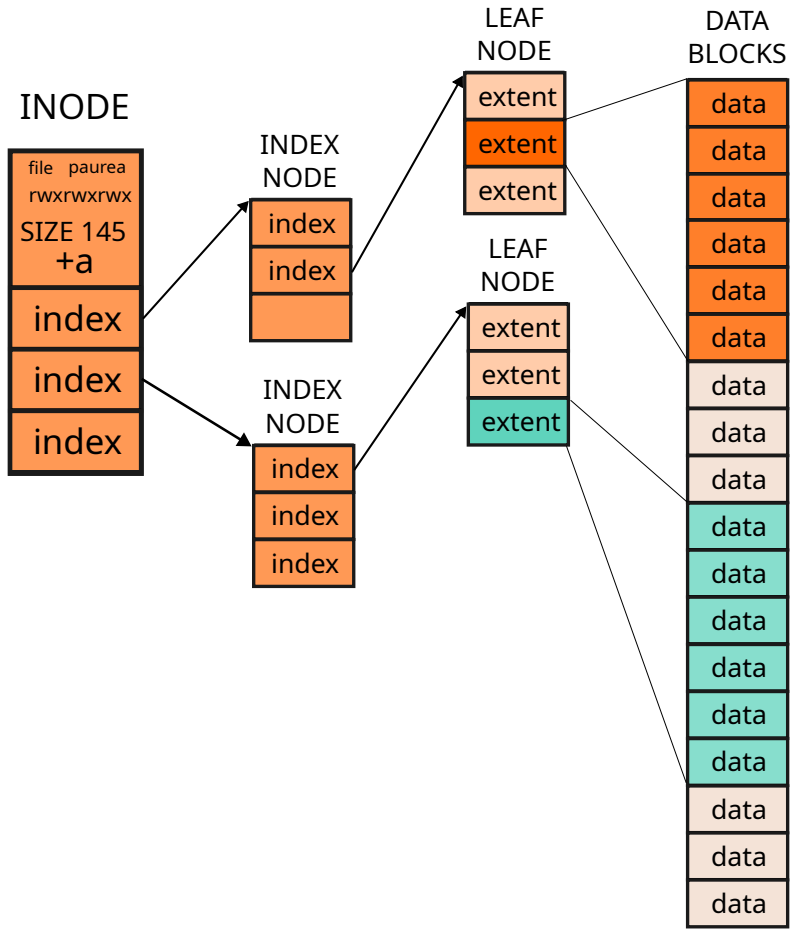
- Basado en inodos, cada inodo un fichero o directorio.
- Superbloque y grupos.
- Cada grupo, superbloque, inodos, bloques (simplificado, hay bitmaps, etc.).



Áreas de disco

Ext4

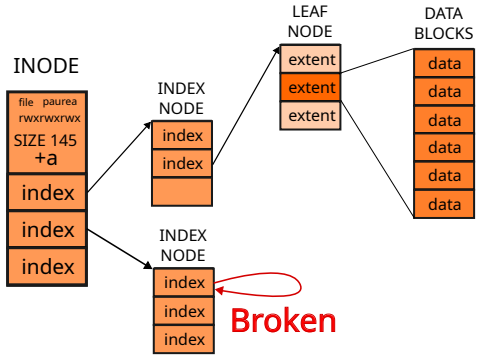
- Basado en inodos, cada inodo: fichero o directorio.
- El inodo tiene los metadatos (el tamaño, el atributo +a, permisos, etc.).
- El inodo apunta a un árbol de regiones de disco: extents.



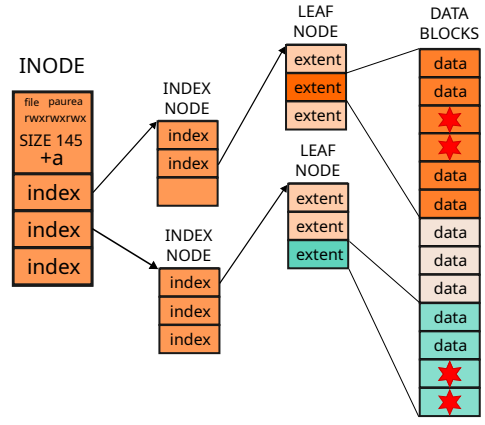
Inodos y extents.

Ext4: journaling

- Ext4 opcionalmente tiene journaling, un buffer circular de bloques.
- Antes de hacer una operación, lo apunta.
- Puede hacerlo con los metadatos o con los datos.
- Si se va la luz, dos posibles garantías:
  - El árbol no acaba roto.
  - No hay datos inventados en los ficheros.
- Siempre se pueden perder datos (caché).



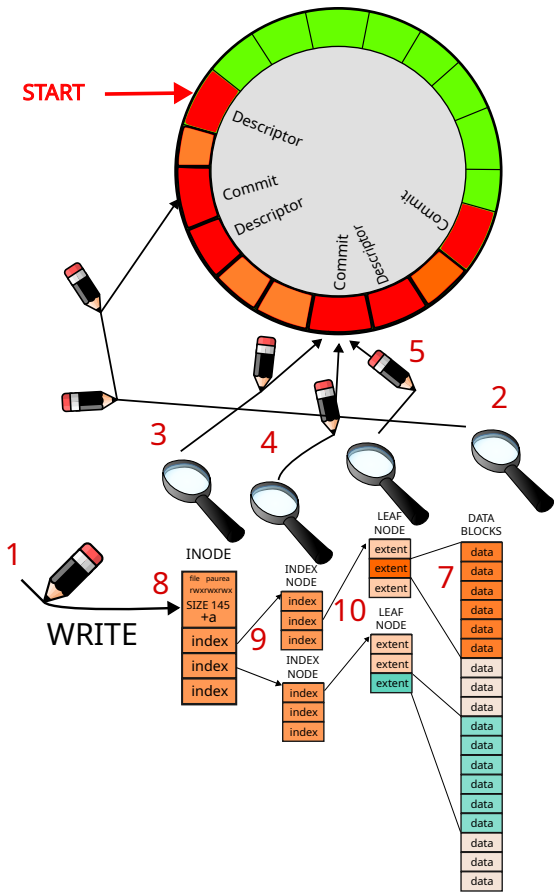
Metadatos rotos



Datos rotos

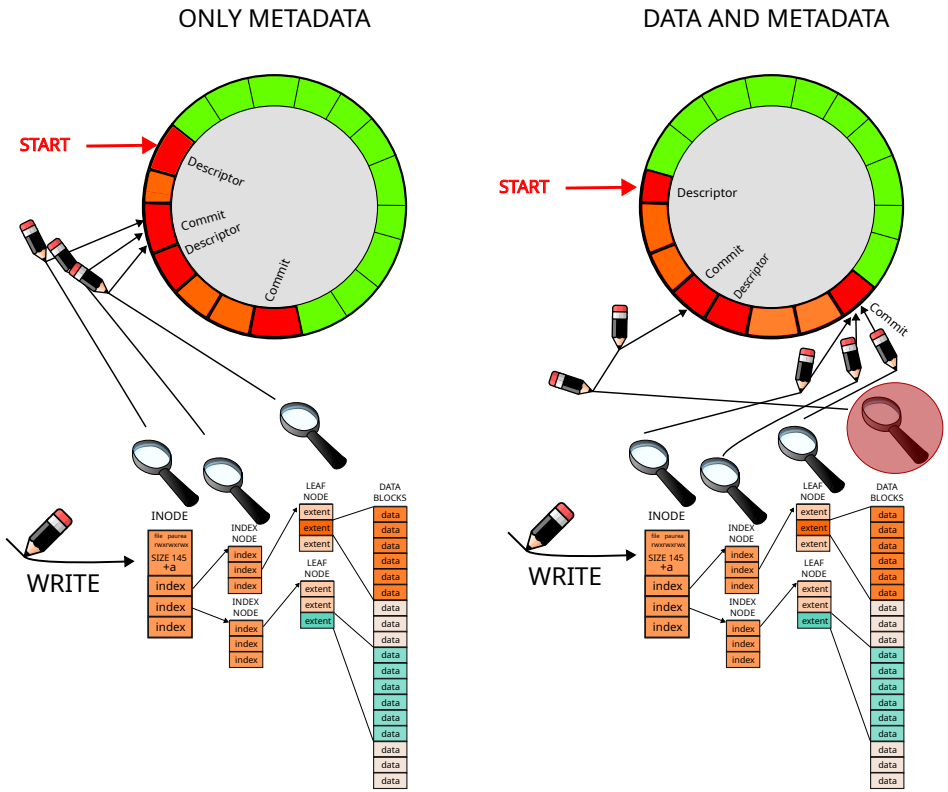
Ext4: journaling

- El formato del journal de Ext4 es un estándar llamado jbd2.
- En el journal van transacciones identificadas por número de sequencia `Seq`.
- Primero vienen `Descriptor`, con los bloques, luego `Commit` que cierran la transacción.
- El orden en el árbol, puede cambiar, en el journal, no.



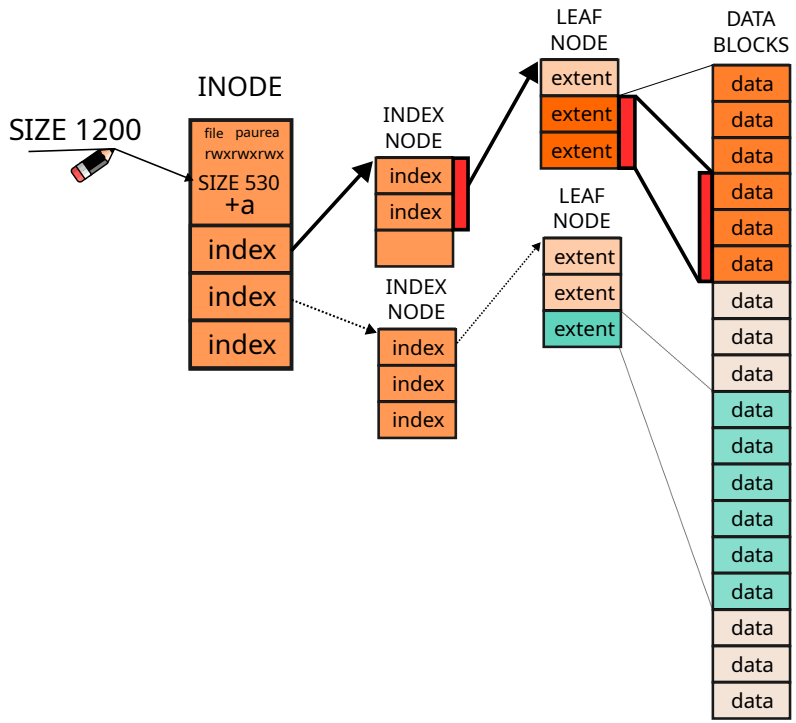
Ext4: modos de journaling

- Sin journaling (es opcional).
- data: journaling de todo (los datos van antes en el journal).
- data_ordered: journaling de metadatos, se hace flush antes del journaling.
- data_writeback: journaling de metadatos, los datos se escribe cuando le viene bien.



Ext4, sin journaling

- Cuando vemos que se modifica el tamaño de un inodo.
- Y ha pasado un tiempo prudencial (ojo, esto es sólo latencia para escribir en los logs).
- Suponemos que el prefijo del árbol se ha estabilizado.
- Todos los bloques empiezan a cero y hay CRCs. Garantías extra.
- **Si el sistema de ficheros es razonable, poca ventana entre datos y metadatos**.



Ext4, con journaling: ordered

- Hace **journaling de los metadatos**, lo miramos y utilizamos los datos del journal.
- Se supone que hace **flush de los datos antes**, pero no nos vale.
- ¡¡¡Los datos esperan indefinidamente el algunos casos (testing)!!!
- Como antes, esperamos un tiempo prudencial, pero mirando el journal.

Ext4, con journaling: data

- Hace **journaling de todo**.
- Sólo fijándonos en el journal podemos actualizar los logs.
- Garantía total (el sistema de ficheros es algo más lento).

Testing

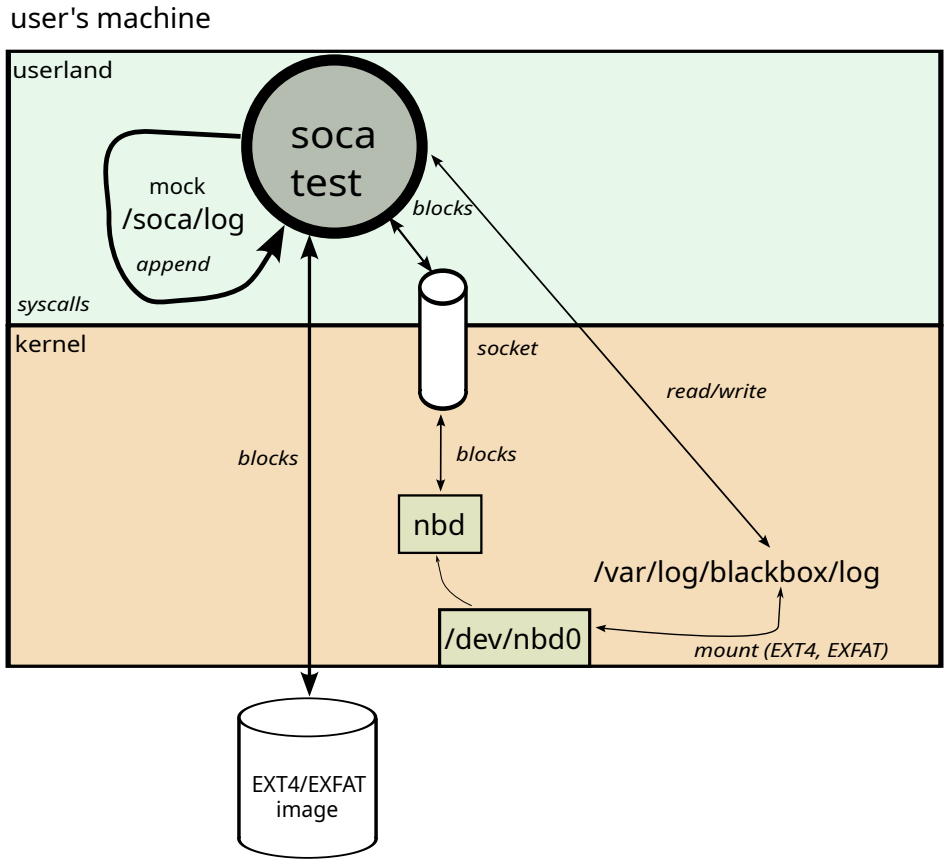
- Intentar romperlo...
- Escrituras pequeñas en paralelo, grandes, en diferentes ficheros...

Testing

- De forma controlada.

Testing

- Complicado.
- Tests de integración, dos tipos, Go, shell.
- Go: Se monta todo el chiringo, pero soca escribe en un tipo de datos sincronizado.
- Shell: Se prueba a saco con scripts.

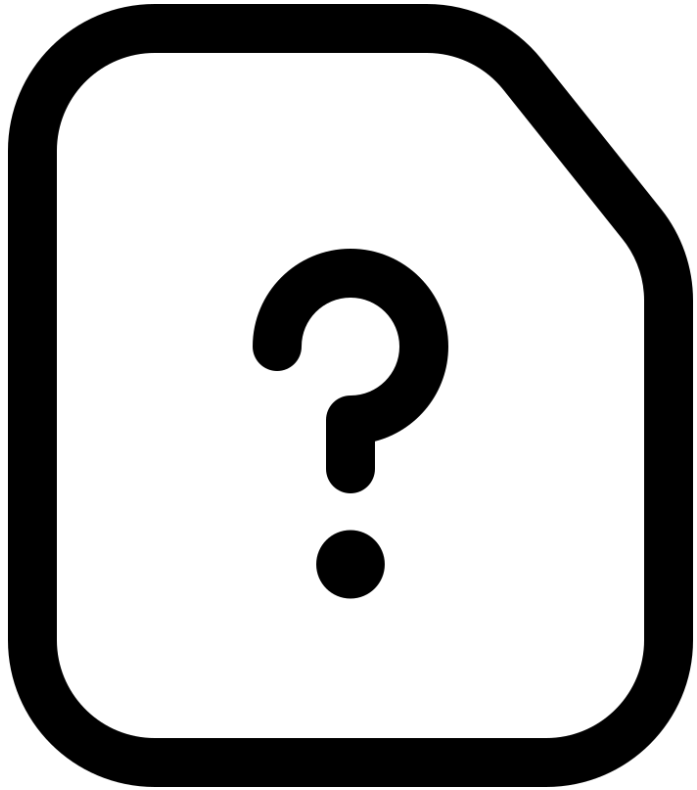


Chiringo de tests de Go

Hasta ahora

- Soca totalmente probado.
- Soca+SealFS probado con Raspberry Pi y todo montado (no testing duro).
- Próximamente haremos el Git público.

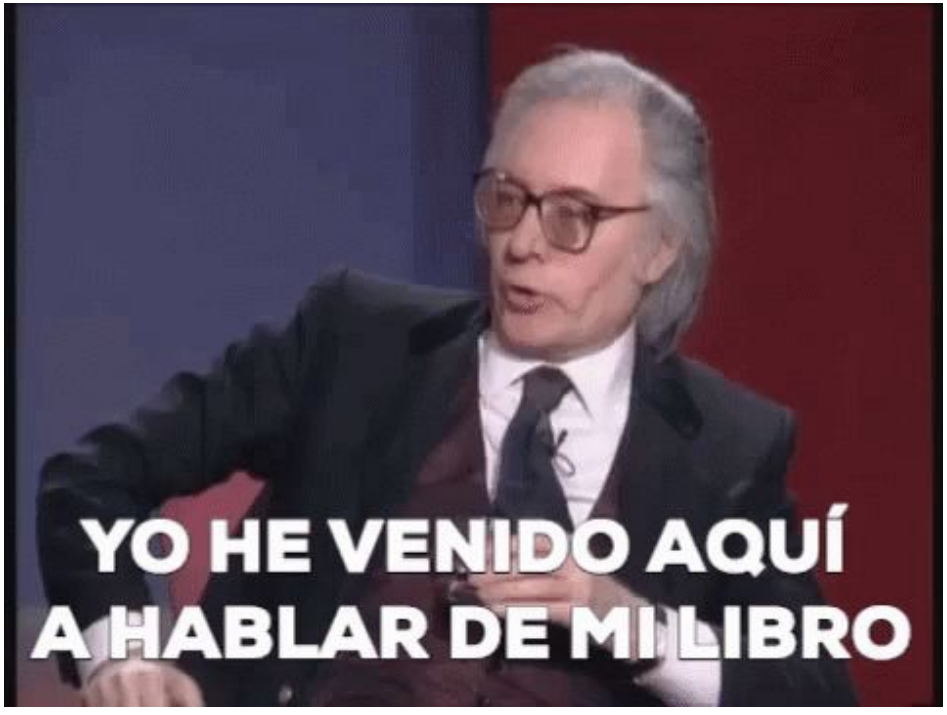
¿Preguntas?



Si surgen luego:

Gorka Guardiola Múzquiz
gorka.guardiola@urjc.es

Enrique Soriano-Salvador
enrique.soriano@urjc.es



- Para aprender más de estas cosas.
- Gratuito, libre, online.
- `https://honecomp.github.io/`

Escenario de uso (I): configuración

1. Antes de empezar a usarlo, la usuaria Alice o un auditor Bob configura la caja negra USB (i.e. Socarrat) con una herramienta.
2. En la configuración, se establecen qué ficheros de log servirá el dispositivo.
3. En un pendrive conectado a la caja negra, se almacenan las claves y la configuración que permitirá la auditoria del dispositivo en el futuro.
4. Al terminar la configuración, se retira el pen drive y se pone a buen recaudo.

Escenario de uso (II): operación

1. Alice conecta la caja negra a su servidor/portátil/robot por USB.
2. La caja negra se identifica como un volumen ext4 y el sistema lo monta.
3. El sistema de ficheros montado tendrá los ficheros indicados en la configuración, que son ficheros convencionales (regular) que tienen el atributo +a (*append-only*).
4. Cualquier aplicación que ejecute la máquina de Alice puede abrir los ficheros y escribir en ellos para añadir datos.
5. Cualquier operación que no sea una escritura al final del fichero se "neutralizará": no será posible modificar o borrar los datos ya escritos de log.

Escenario de uso (III): ataque

1. Si la máquina de Alice es atacada y el adversario se hace `root`, podrá tomar el control de todo el sistema y las aplicaciones.
2. Todo lo que se escriba en el log a partir de ese momento, estará controlado por el adversario. Todo lo escrito antes ya estará en los ficheros de log asegurados (i.e. toda la actividad pre-elevación, incluidos los indicios del ataque).
3. El adversario no podrá borrar o alterar los ficheros de log, ya que son WORM.
4. Si es un ataque lógico (i.e. remoto), la superficie de ataque al dispositivo WORM se limita al enlace USB $\rightarrow$ será prácticamente imposible comprometerlo.
5. Si es un ataque físico o se logra comprometer la caja negra a través del USB, cualquier modificación de los datos de logs hasta este punto será detectada.

Escenario de uso (IV): verificación

1. Se desmonta el sistema de ficheros de la máquina de Alice.
2. El auditor extrae los ficheros de log junto con otro fichero de metalog.
3. Con el pen drive que solo tiene él y esos ficheros, se comprueba si los logs son confiables o no: cualquier modificación desde su escritura hará que la verificación falle.