Threat Model, Mechanisms & Policies

Taller de Seguridad

Ing. Martin Di Paola - Fiuba

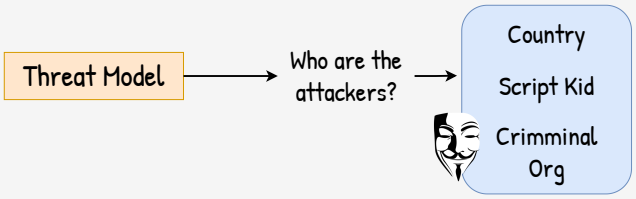
.

.

Threat Model

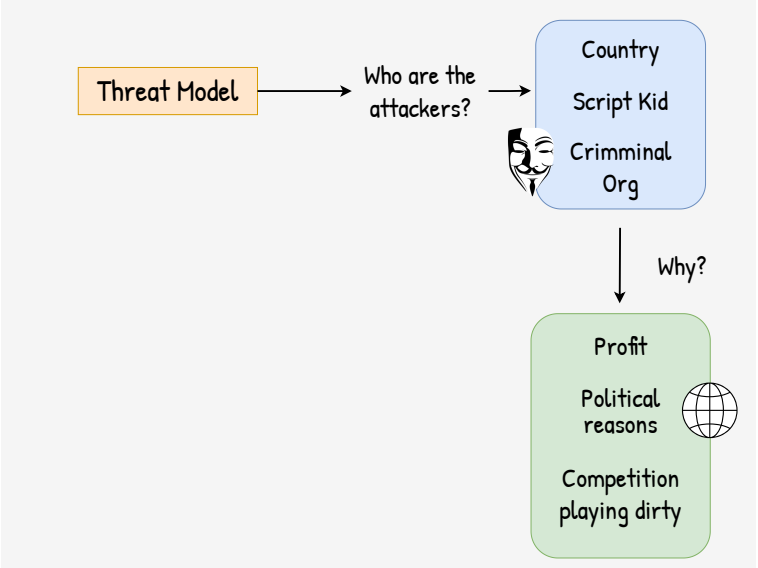
Hacer a un sistema seguro es costoso: lleva tiempo, energía, dinero y puede hacerte una persona poco popular entre tus usuarios.

Saber a *que/quien* te enfrentas te permite enfocarte en amenazas razonables y aplicar defensas acordes.



Quien puede atacarte?

Un país tiene mucha más personas especializadas con budget cuasi-infinito para atacarte; un script kid claramente no.



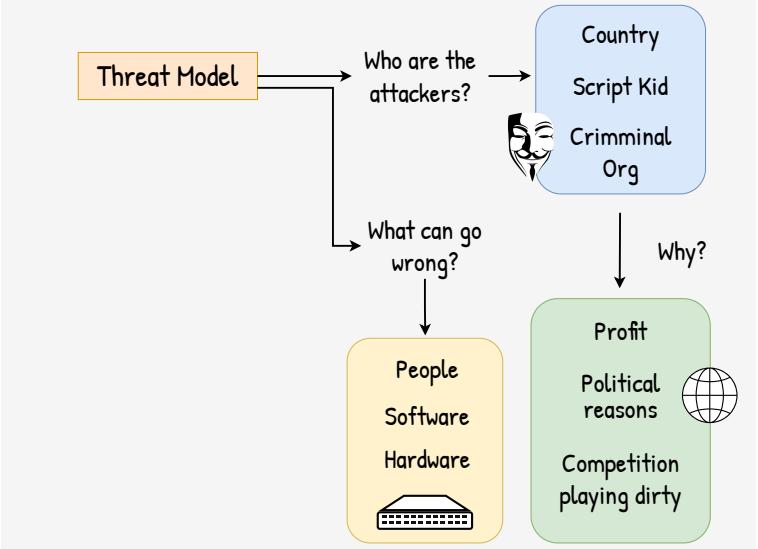
Que motivos hay para atacarte?

Sos una app para tomar notas? Posiblemente no te ataque nadie salvo algún oportunista o algún atacante que solo te usara de trampolín para su otro, verdadero objetivo.

Sos una social network? Puede q tengas que cuidarte de la competencia.

Sos una fintech? Es cuestión de tiempo que org criminales quieran robarte/estafarte, a vos o a tus usuarios.

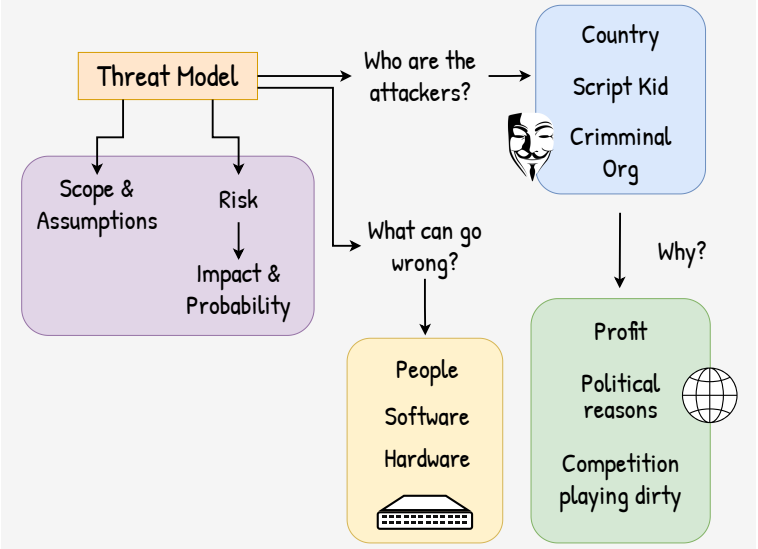
Fabricas computadoras para la infra critica de EEUU pero importas los chips de China? Puede q quieras revisar esos chips...



Darle forma al threat model es iterativo: preguntate *"que puede salir mal?"* para ir descubriéndolo. Un bug en el software? Algún backdoor en el hardware?

Confías en las personas, pero en organizaciones grandes siempre hay alguna manzana podrida.

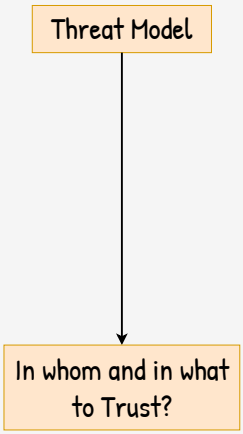
Y aun sin malicia las personas cometen errores por ignorancia or estupidez. Quien no hizo un click en un link q vino por mail prometiendo la fortuna de un rey nigeriano?



Al final el threat model define que está en alcance y q no. Que **assets** proteger y cuales no. Siempre hay tradeoffs y se asumen cosas para simplificar.

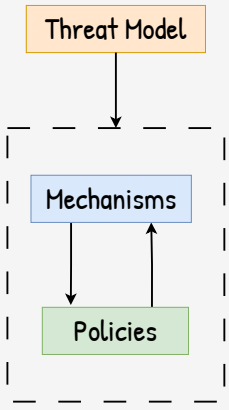
Cual es el **riesgo**? No toda vulnerabilidad es igualmente dañina. Ataques posibles pero improbables pueden quedar sin protección.

Son en estas *assumptions* y *"pero esto es poco probable"* donde el atacante va a aprovecharse.



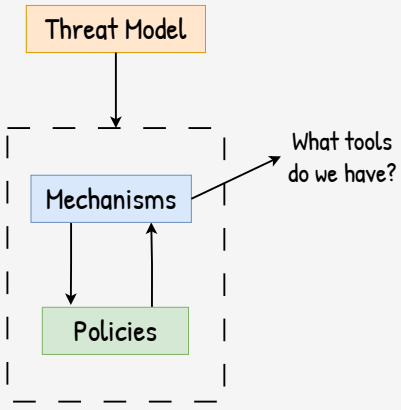
Al final se reduce a *en que/quien podes confiar?* en el hardware? en las personas? en tu proveedor de internet?

Haz tu apuesta. El hacker hara la suya.



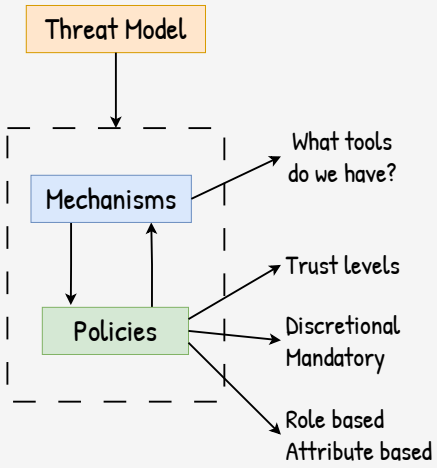
El threat model es solo papel. La materialización de las protecciones se da via los mecanismos y políticas.

Hay una sinergia entre ellos: las políticas definen que mecánicas sirven y cuales no mientras que las mecánicas ofrecen más flexibilidad o simpleza a las políticas.



Que herramientas disponemos? Pensa en una cerradura y su set de llaves: esa es la mecánica. Que los miembros de una empresa puedan entrar a su oficina pero no otras personas, eso es la policy y darle la llave a cada miembro es la puesta en acción de la policy.

Notar el threat model: *"alguien ajeno a la empresa no puede entrar sin una llave"* deja sin resolver que pasa si alguien entra a martillazos :D



.

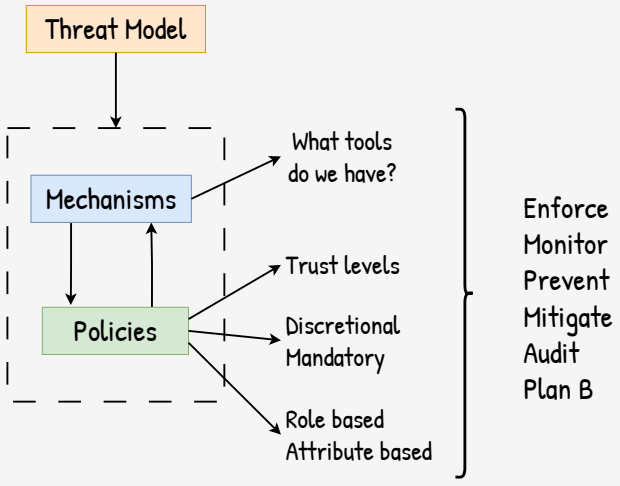
Las mecánicas y políticas están para enforzar/implementar las protecciones contra las amenazas modeladas.

Pero hay que ser proactivos: monitorear para detectar ataques lo antes posibles para prevenirlos/bloquearlos.

El hecho es q en algún momento van a entrar igual, entonces, como mitigamos el daño?

Y si entraron, tenemos registros para hacer una auditoria luego?

Y si todo falla. Tenemos un Plan B?

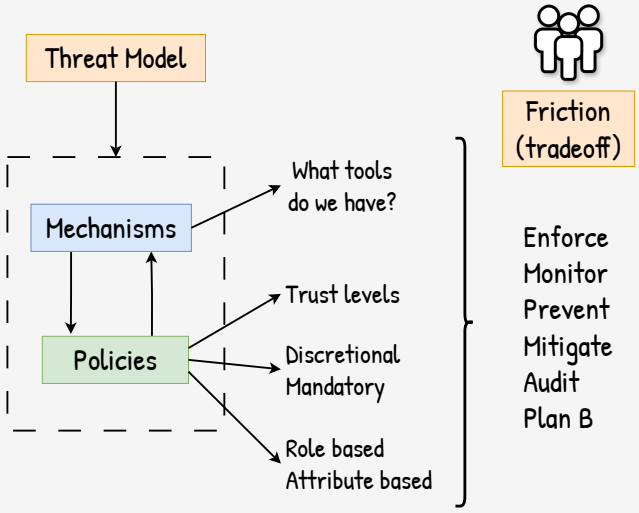


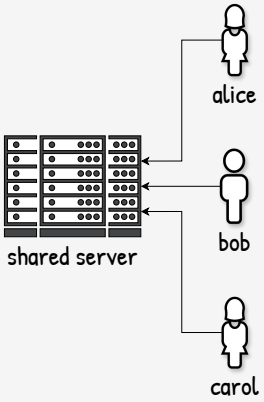
La contracara a todo esto es la **fricción** que se genera.

Hay costos en términos de tiempo y dinero, pero la fricción hara que los usuarios estén todo el tiempo incómodos y tarde o temprano sean *ellos los principales hackers*.

Pone algo muy restrictivo y los usuarios serán los primeros en bypassearlo, muchas veces, dejándolos más expuestos aun.

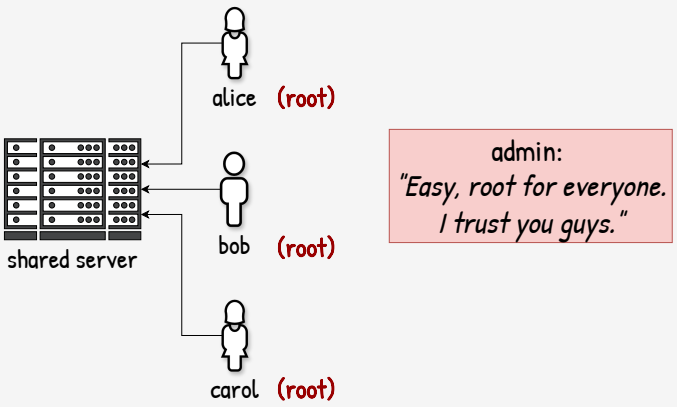
La seguridad tiene un precio. Es un **tradeoff**.





Vamos por un ejemplo.

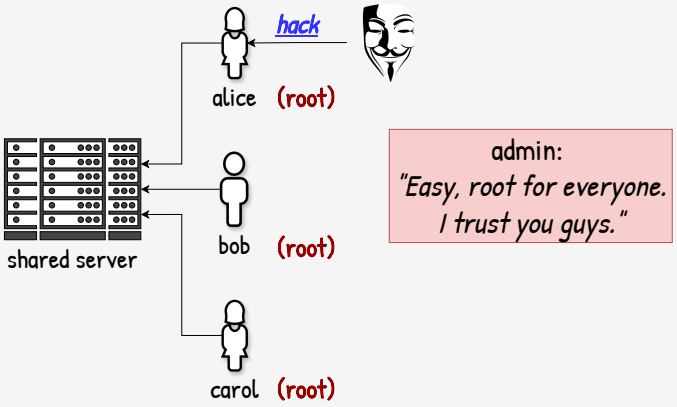
Imaginate: un server compartido y 3 usuarios que necesitan operar con él. Que permisos le darías a cada uno?



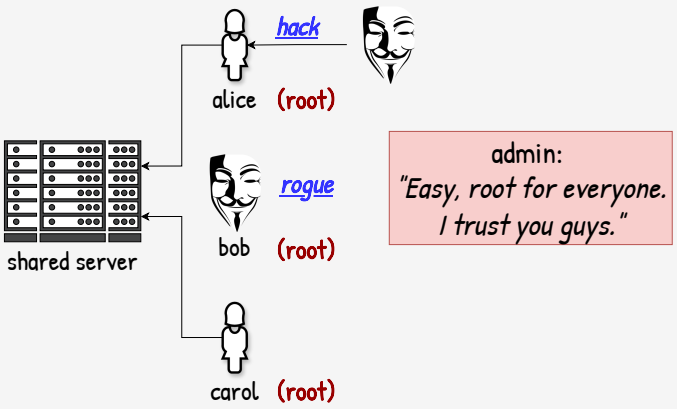
Root para todos! Totalmente inseguro pero, no es loco.

Si Alice quiere instalar un programa lo puede hacer; si Bob necesita ver un archivo de log también;

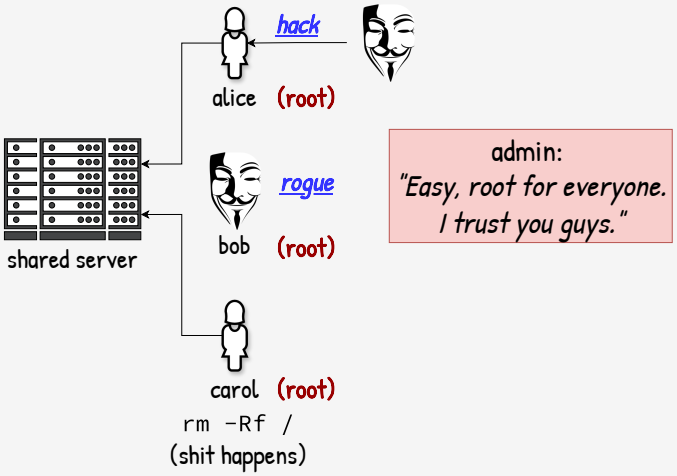
Es fácil, es friction-less.



Pero, si la pawnean a Alice? El hacker tendrá acceso al servidor como root.

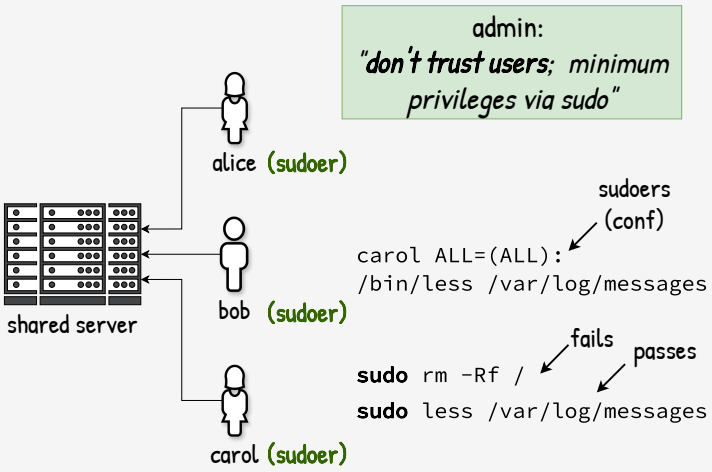


Y si Bob es un empleado desleal?



Incluso, si no pensamos en escenarios de malicia y pensamos solo en escenarios de estupidez.

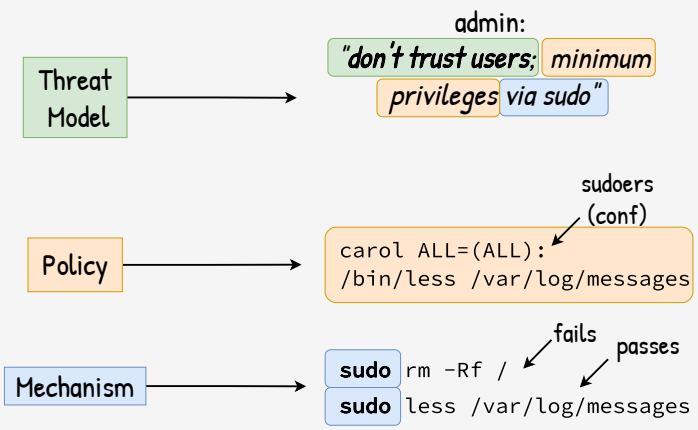
You know, shit happens...



El threat model *"confío en todos"* es un mal negocio. Es más seguro no confiar en nadie y dar solo el mínimo de permisos.

Un user ejecuta una acción privilegiada con *sudo* para acceder al recurso y se determina si continuar o fallar basado en la config *sudoers* definida por el admin.

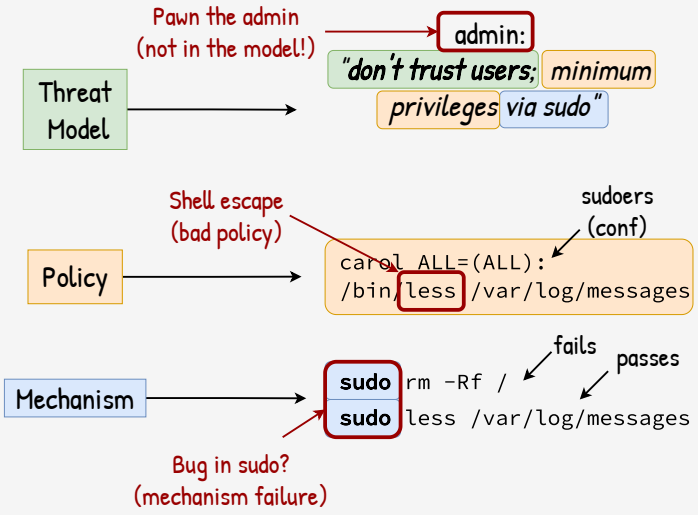
Ahora, si Carol quiere por accidente borrar el server, no puede. Pero si Alice quiere instalarse una tool, tampoco: le tendrá q pedir permiso al admin. Acá es donde empieza la fricción.



"No confío en los users" es el **threat model**;

Asignar los mínimos permisos (*sudoers*) es la **policy**;

Y *sudo* es el **mecanismo** (junto con los permisos de linux).



Como hackers vamos a buscar y aprovecharnos de los puntos débiles:

- Que **no** esta cubierto por el threat model? por la policy?
- Hay algún **mismatch** entre *"lo que se quería hacer"* y lo que

  efectivamente la policy esta implementando?
- Podemos **engañar/romper** el mecanismo?

La cerradura y llaves es *"para que personas no autorizadas no puedan entrar a un edificio"*. Seguro?

Se podría clonar la llave; se podría forzar la cerradura con lockpicking; se podría engañar a alguien con llave para q nos deje entrar.

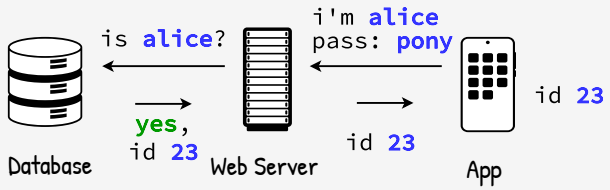
Think like an attacker.

Vamos con otro ejemplo.

Supongamos el siguiente escenario: un app y un backend q consta de un server web y una database.

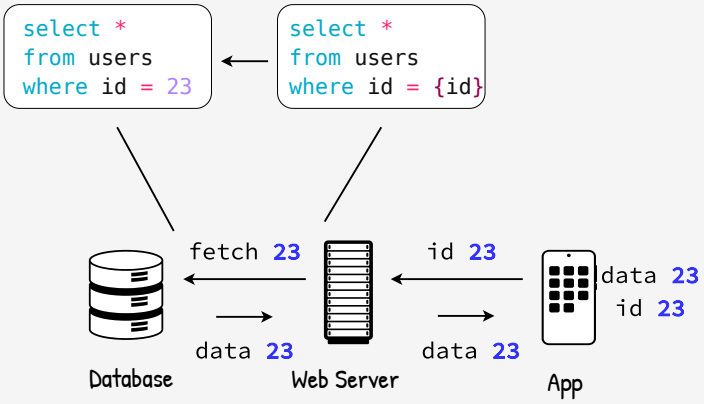
Alice son loguea a la app pasándole su user y password al server quien luego verifica si las credentials están ok contra la base de datos.

Si todo sale bien le devuelve a Alice su user id (23).



Con la app logueada, Alice ahora se trae la data de su usuario.

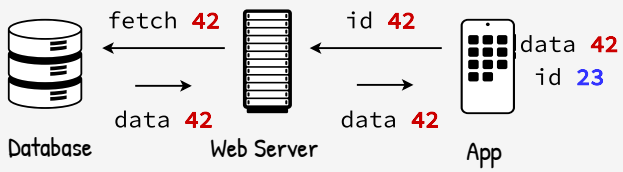
Que puede salir mal?

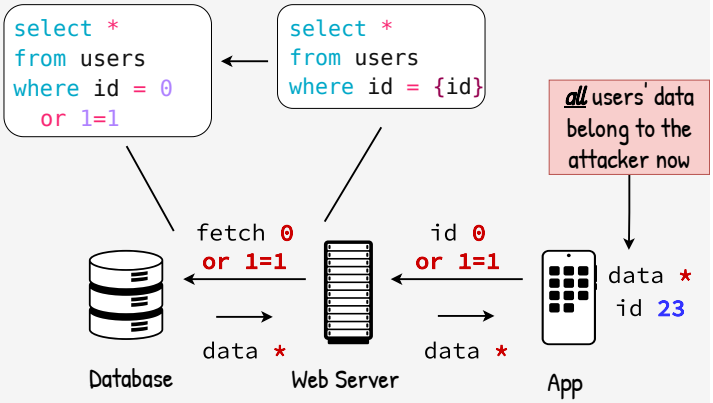


Puede que Alice pida los datos con el user id 42 y no con el suyo (el 23).

Como el servidor web *confía* en la app, no se cuestiona que hubo un cambio de id. El desarrollador del server web habrá pensado *"como la app la hacemos nosotros, siempre nos va a dar el id correcto"*.

No se imaginó que Alice podría estar hackeando la app o los mensajes al server.

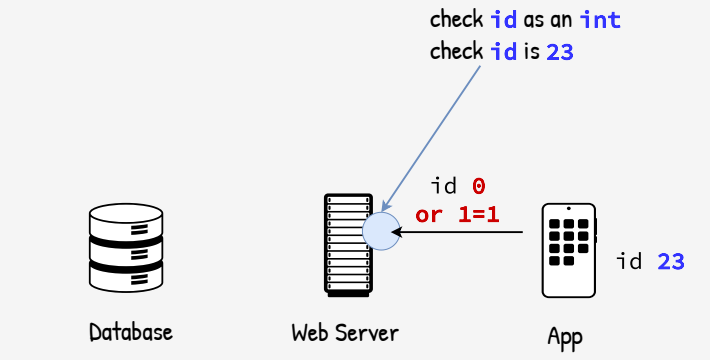




Y si ahora Alice envía *"0 or 1=1"*?

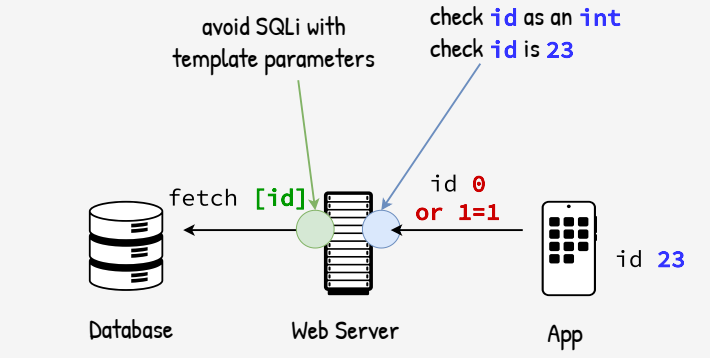
La query SQL va a retornar los users cuyo id sea exactamente al *id 0* o los users donde la condición *1=1* se cumpla (lease, todos!).

El server web *confía* en q la app le envía un integer (cosa q **no** es así); el código del server al armar la query de SQL *confía* que los argumentos son integers (cosa q **no** es así).



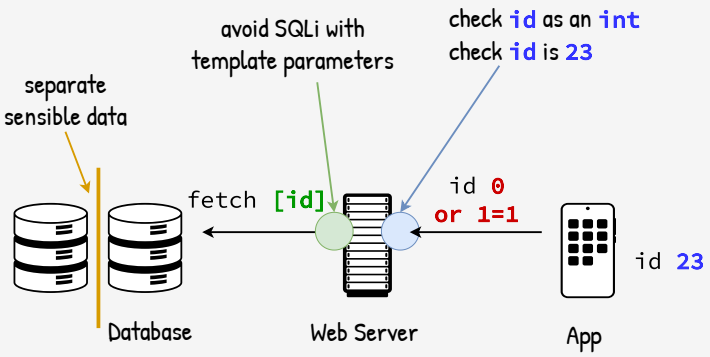
Fix: el server fuerza a q el user id sea siempre un integer y sea el correspondiente al user logueado (para Alice, el 23).

Con esto frenamos el ataque, pero...



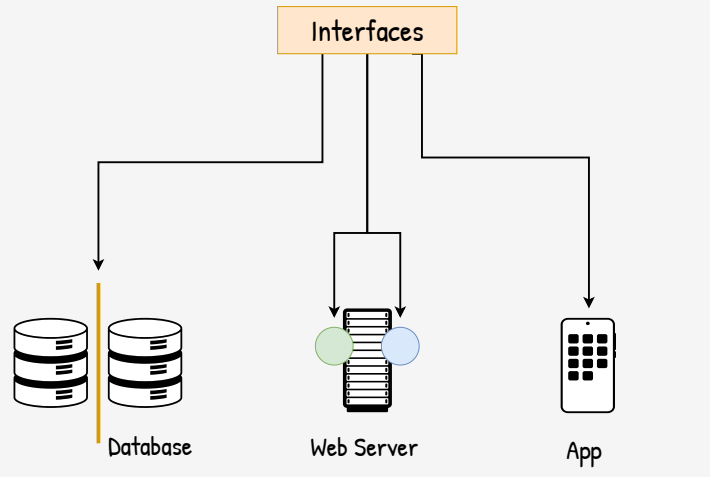
Pero no hay q quedarse en la capa superficial: el código del server cuando arma la query debe usar template parameters para evitar otro SQL Injection (no importa si están puestos los checks de integers).

Y no importa si no es posible un SQL Injection, aun así la database se separa poniendo los datos más sensibles en una DB separada.

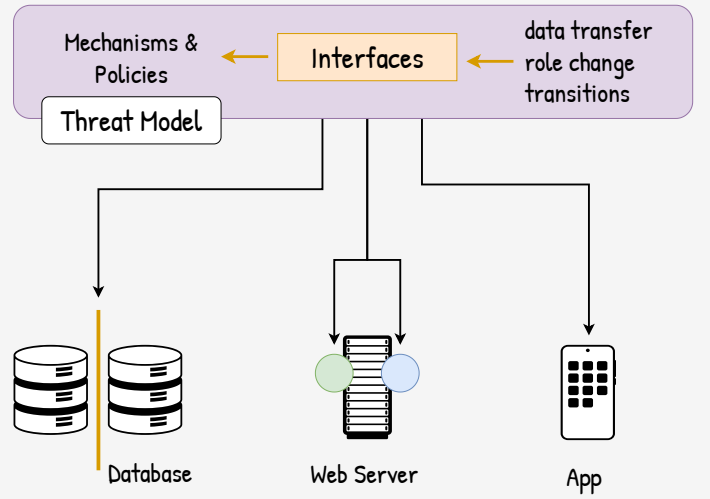


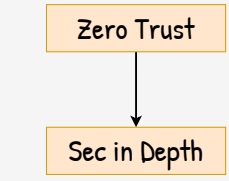
Los puntos de contacto donde hay un intercambio de datos entre componentes son las **interfaces** o **boundaries**. Un control mal implementado o inexistente o una confianza innecesaria, y un componente puede bypassear, robar, o tomar ownership de otro.

Hay q poner controles, no solo en la interfaz más expuesta (app – web server) sino en todas las interfaces por si una falla hay otra más detrás. Esto es **seguridad en profundo** (depth).

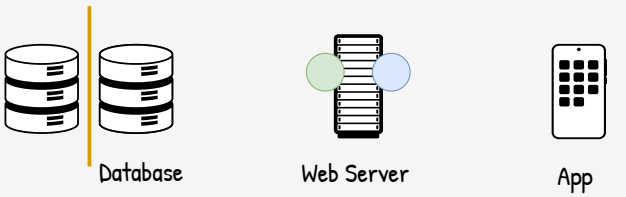


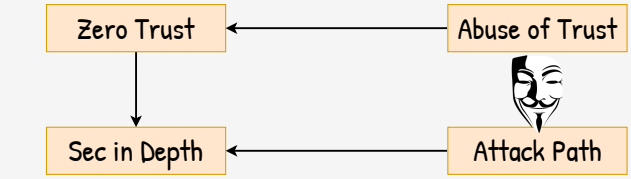
.





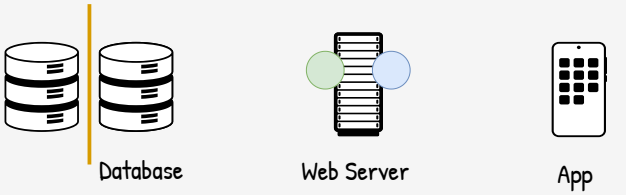
No confiar (ni en tu propio código) te lleva a trabajar a al defensiva, con controles a lo largo del sistema.

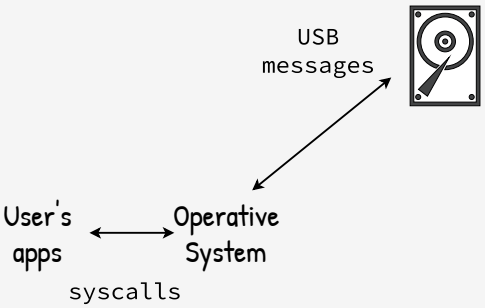




Un hacker, en su ofensiva, buscara donde poder *abusar de la confianza*.

En general un fallo de seguridad no es puntual sino la **combinación** de múltiples ataques encadenados que logran romper los checkeos a lo largo de todas las interfaces, un **attack path** hacia la cookie, el flag, o las tarjetas de crédito.

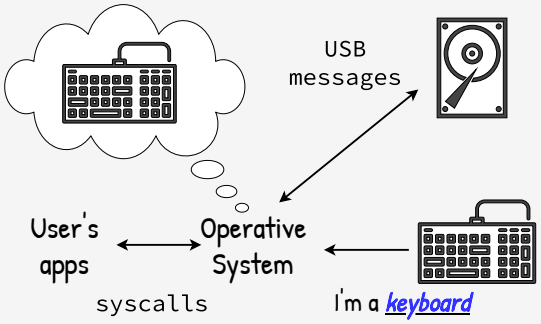




Otro ejemplo.

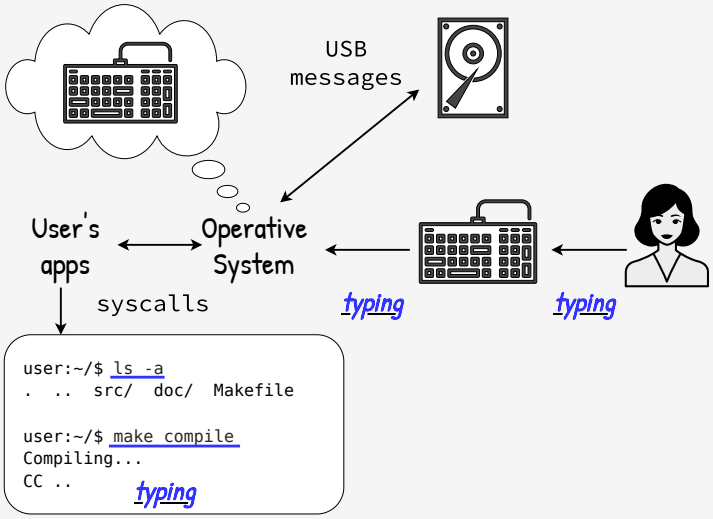
Las apps del usuario interactúan con el sistema operativo y este con el hardware.

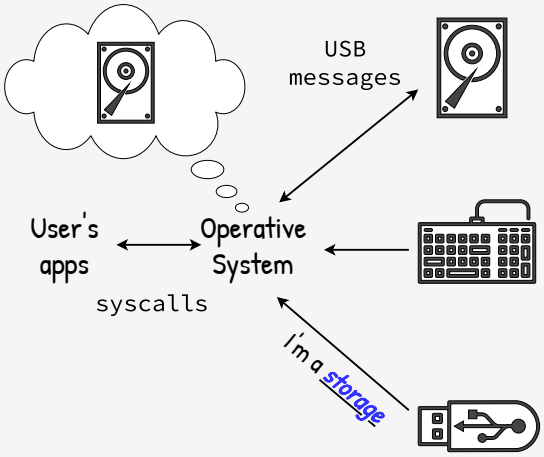
Hay claramente 2 interfaces: apps – OS y OS – HW.



Cuando un teclado nuevo se conecta, le envía un mensaje al OS diciéndole *"hey soy un teclado"*. Y el OS entonces acepta e interpreta los key-strokes que vengan de él.

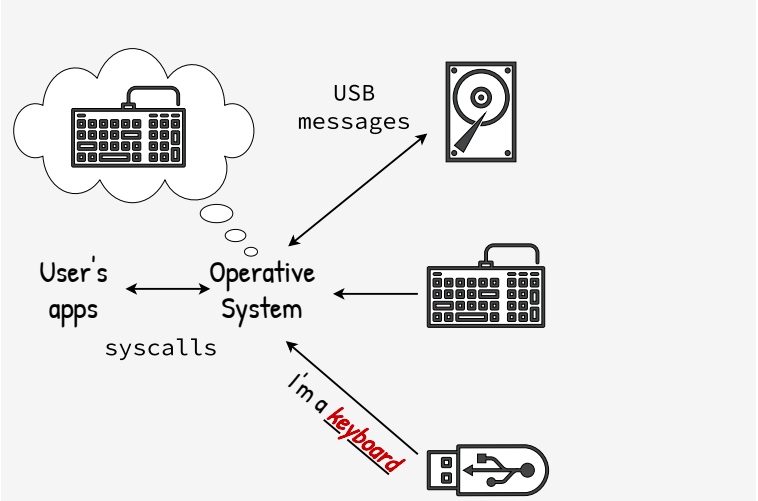
.





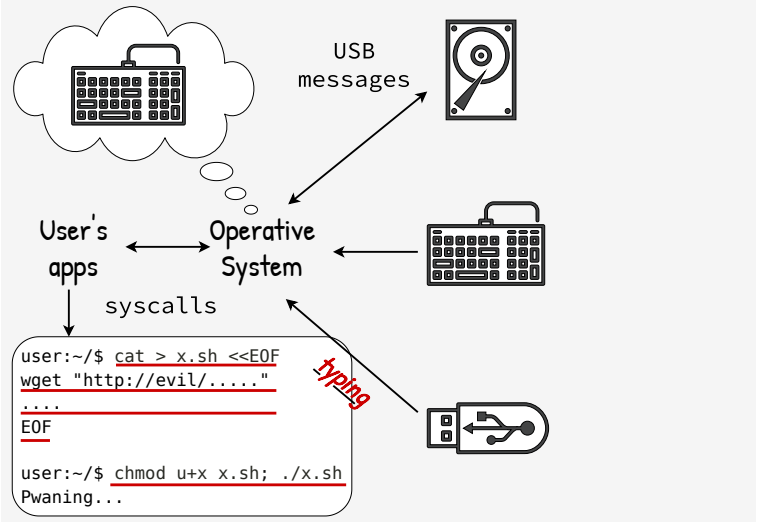
Cuando un pendrive nuevo se conecta, le envía un mensaje al OS diciéndole *"hey soy un espacio de almacenamiento"*. Y el OS entonces monta un file system para leer / escribir data desde / hacia él.

Estas viendo donde esta la *confianza*? **Think like an attacker**



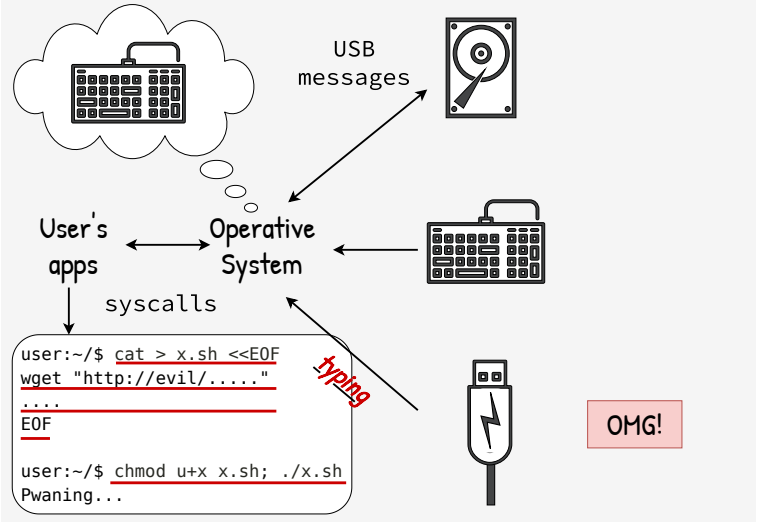
Si el pendrive le dice al OS *"hey, soy un teclado"*, el OS no sabría como detectar la mentira.

El OS esta **confiando** ciegamente en el device (y por como funciona USB no tiene tampoco mucho como protegerse!)



El firmware del pendrive puede no solo hacerse pasar por un teclado sino también emitir key-strokes como si fuese uno y *"tippear"* un shellcode para tomar posesión de la máquina.

Esto es un *bad usb* (googlealo).



Y si en vez de un pendrive fuese el cable USB q usas para cargar un celu?

Esto es un *OMG cable* (googlealo).