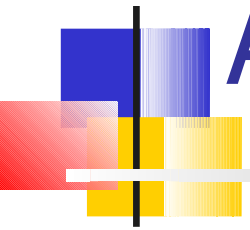


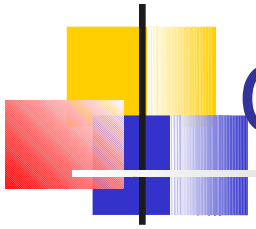
Analisi delle vulnerabilità

Vulnerabilità

- Un difetto in un componente del sistema
- Sfruttando il difetto riesco a generare un comportamento inatteso del componente
- Il comportamento inatteso permette di violare le proprietà di sicurezza

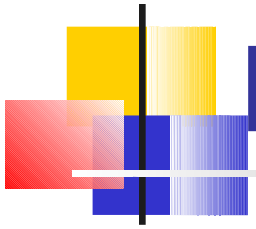
Vulnerabilità ed errore (bug)

- Un bug non ha necessariamente implicazioni sulla sicurezza
- Un bug che ha ripercussioni sulla sicurezza e' una vulnerabilità



Classificazione delle Vulnerabilità

- Esistono modi alternativi di classificare una vulnerabilità che dipendono da cosa ci interessa evidenziare
- E’ importante capire quale tipo di classificazione usare



Dove è la vulnerabilità

- Nel modo in cui si fa una cosa
  - Vulnerabilità procedurale
- Nelle persone che fanno
  - Vulnerabilità organizzativa
- Nello strumento (hardware e/o software) che si usa
  - Vulnerabilità degli strumenti informatici

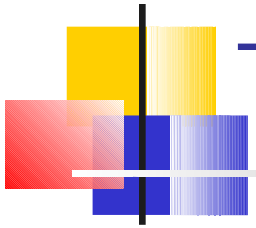
Esempio

- Procedurale
  - Una password viene stampata e comunicata in una busta aperta
- Organizzativa
  - Più persone devono contemporaneamente amministrare una macchina
  - Formazione inadeguata
- Negli strumenti
  - Password trasmessa in chiaro su una rete
  - Overflow per assenza di controlli

Vulnerabilità organizzative

- Altri casi possibili
  - Password scritta su post-it = si privilegia integrità del sistema rispetto alla confidenzialità
  - Account non eliminati
  - Dischi non cancellati quando si vende un PC
  - Dati su supporti mobili non criptati

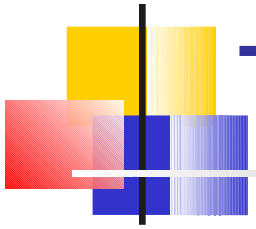
- La ricerca di vulnerabilitá deve partire dal contesto organizzativo e gestionale di un sistema



Tipi di vulnerabilità degli strumenti informatici

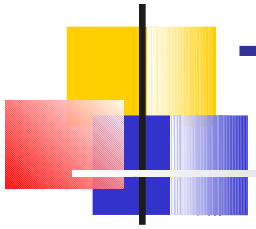
Abbiamo una ulteriore classificazione, anch'essa potenzialmente ambigua ma utile

- Specifica
  - il componente è, inutilmente, più generale del necessario
- Implementazione
  - l'errore è stato commesso nello sviluppo del componente
- Strutturale
  - l'errore nasce quando si integrano i componenti che costituiscono il sistema complessivo



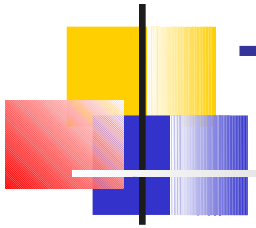
Tipi di vulnerabilità - Esempi

- Specifica = programming-in-the-large
  - Uso una libreria che contiene piú funzioni di quelle richieste
  - Chi riesce ad invocare queste funzioni ha diritti che possono violare funzioni di sicurezza
  - Riuso può introdurre vulnerabilità nel sistema che riusa un certo componente



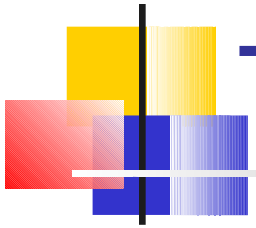
Tipi di vulnerabilità - Esempi

- Implementazione =
  - ipotesi su input (correttezza parziale)
  - mancanza di controlli su input
  - confusione tra dati e programmi = è possibile "saltare" in una struttura dati = stack o heap overflow
- Le vulnerabilità possibili sono fortemente dipendenti dal meccanismo dei tipi del linguaggio di programmazione utilizzato = un linguaggio con controlli forti sui tipi non permette overflow



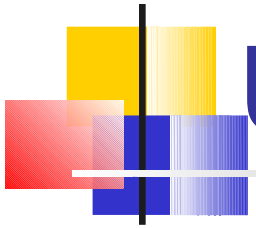
Tipi di vulnerabilità - Esempi

- Strutturale = nasce dalla composizione di componenti
  - Corretti in isolamento
  - Non compatibili
- Problemi di stack tcp/ip già visti
- Composizione di componenti che delegano ad altri i controlli con componenti che non controllano
- Funzionalità troppo generali per il sistema considerato



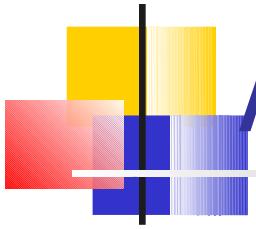
Tipi di vulnerabilità - Esempi

- Strutturale = nasce dalla composizione di componenti
  - Corretti in isolamento
  - Non compatibili
- Problemi di stack tcp/ip già visti
- Composizione di componenti che delegano ad altri i controlli con componenti che non controllano
- Funzionalità troppo generali per il sistema considerato



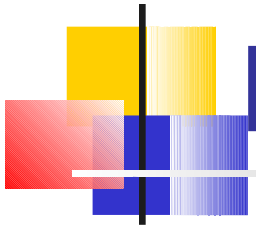
Una classificazione alternativa

- Le vulnerabilità possono essere classificate anche dal punto di vista di
  - Chi le può sfruttare
    - Chi ha un account sulla macchina
    - Chi può inviare un pacchetto alla macchina
    - …
  - Cosa permettono di fare a chi le può sfruttare



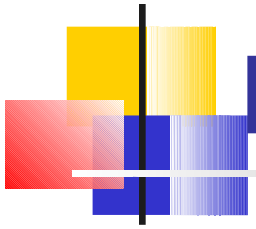
Analisi delle vulnerabilità

- Sono possibili altre classificazioni oltre a quelle già viste
- Sono pericolose le vulnerabilità dei componenti informatici ma non dobbiamo sottovalutare le vulnerabilità
  - di tipo organizzativo/procedurale
  - dovute a collusioni con attaccanti
  - social engineering



Ricerca delle vulnerabiltà

- Componenti standard soffrono di vulnerabilità ben note per cui sono già noti gli exploit
- Occorre focalizzarsi su
  - Componenti non standard
  - Vulnerabilità strutturali che nascono dalla composizione dei componenti del sistema



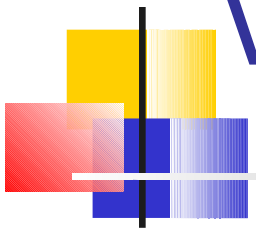
Ricerca di sistemi e vulnerabilità

- Vulnerabilità banali possono essere localizzate in maniera automatica mediante strumenti di vulnerability scanning (Nessus, Satan)
- Strumenti di VS individuano
  - i nodi in un certo range di indirizzi IP
  - Il SO e le applicazioni standard eseguite (fingerprinting)
  - le vulnerabilità di questi componenti

Utilizzano un database <applicazione, vulnerabilità>

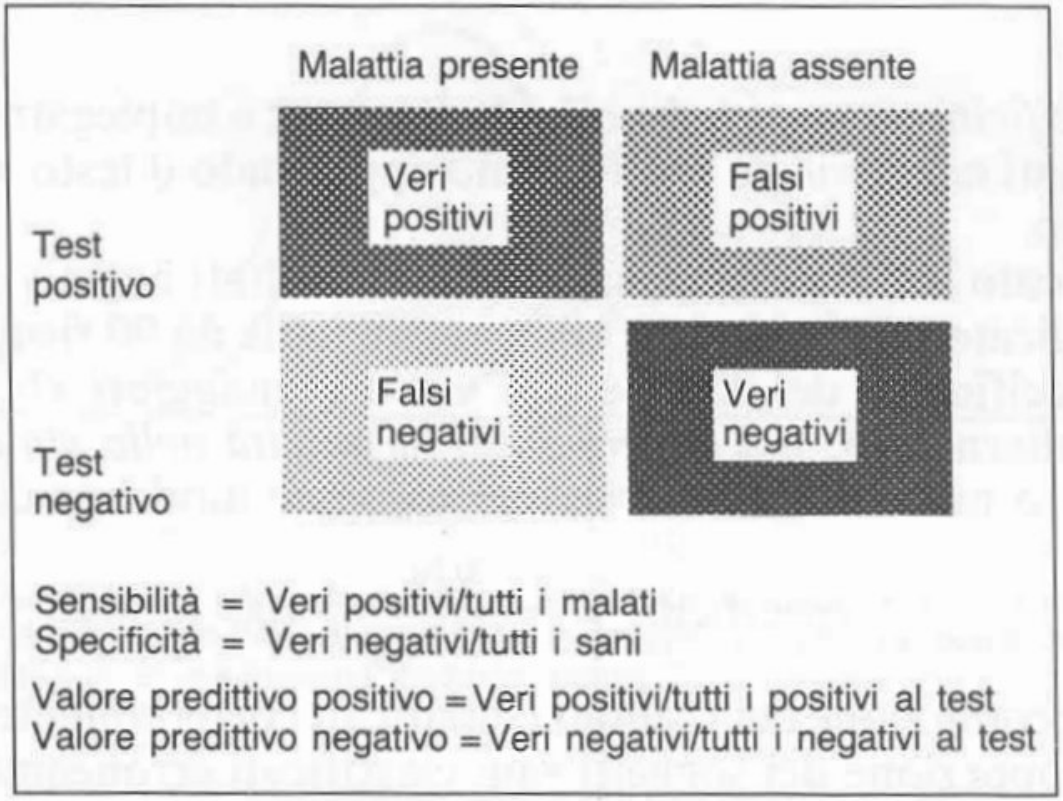
- Il vulnerability scanning
  - è una parte di vulnerability analysis
  - non coincide con la vulnerability analysis

Vulnerability Scanner



- Vedremo in alcune lezioni successive gli strumenti di vulnerability scanner per l'individuazione di vulnerabilità standard
- Non esauriscono le vulnerabilità
- Falsi positivi su vulnerabilità
- Non contestualizzano vulnerabilità nello specifico sistema ma danno valutazioni indipendenti dal sistema

Falso positivo, falso negativo ...

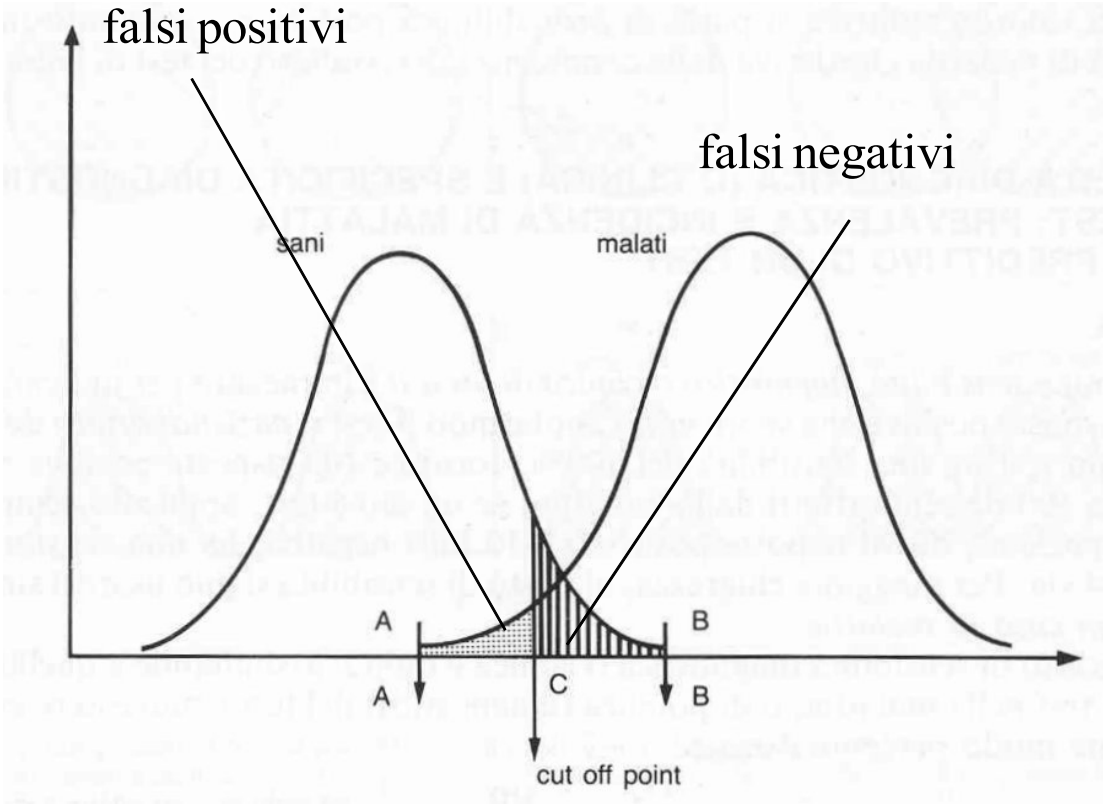


Termini di origini medica-diagnostica

Studiano correlazione tra sintomi (test) e malattia

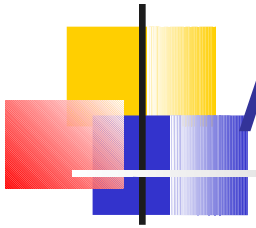
Confusion matrix

Falso positivo, falso negativo ...



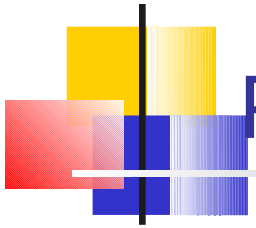
Legati a distribuzioni di probabilità non disgiunte di un certo sintomo o output di un test

Si sceglie un valore di cut off e ciò genera confusione



Analisi delle vulnerabilità

- Per determinare vulnerabilità occorre considerare che il numero di vulnerabilità in un componente determina una proprietà sistemica detta robustezza
- Sistemica = dipende dai componenti e dalle relazioni tra i componenti
- Esiste una relazione tra
  - Ricerca di Vulnerabilità
  - Robustezza = proprietà del sistema
  - Principi di progetto violati



Robustezza Informatica

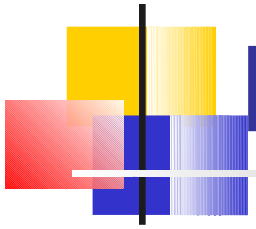
- Robustezza (robustness) di un componente =

  capacità di non danneggiare il sistema in cui è inserito quando vengono violate alcune delle specifiche del componente stesso

- violazione delle specifiche =
  - input diversi da quelli specificati
  - risorse diverse da quelle specificate
  - … (enumerating badness)

Robustezza Informatica

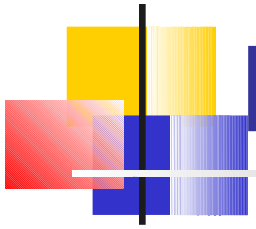
- è una proprietà diversa da correttezza, efficienza, facilità d'uso, ...
- è in contrasto con efficienza e facilità d'uso, può aumentare solo a spese di queste due proprietà



Robustezza Informatica

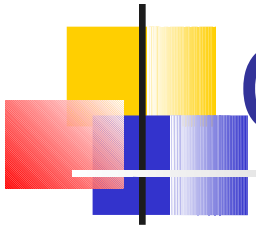
Un programma per il calcolo degli stipendi che dato un nome restituisce lo stipendio

- corretto se calcola lo stipendio esatto
- efficiente se impiega poco tempo
- facile da usare se addestramento è breve
- robusto ????



Robusto ....

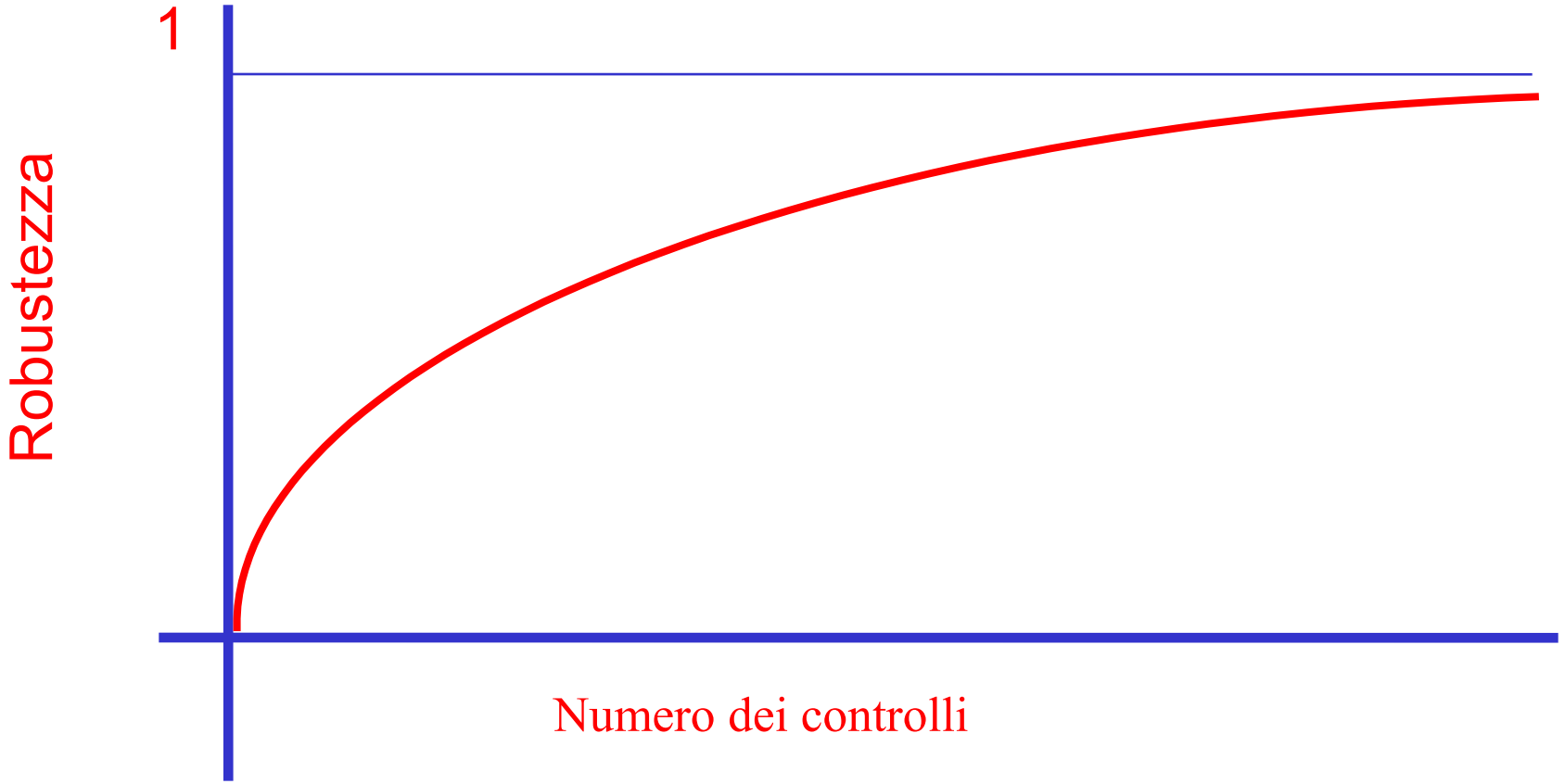
- Formato sbagliato del record
- nome del tutto non esistente
- allocata meno memoria di quella richiesta dal programma
- file con nomi non esistente
- file con qualifiche non esistente
- disco non esistente ....

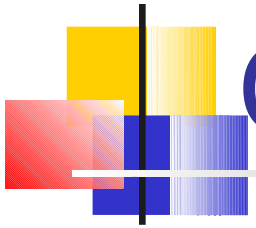


Quanto robusto ....

- è estremamente difficile prevedere le possibili violazioni delle specifiche (un caso particolare di enumerating badness)
- robustezza non è una proprietà 0/1
- misura di robustezza = associare ad ogni componente un valore da 0 a 1
- 1 è valore asintotico
- robustezza dipende dai controlli

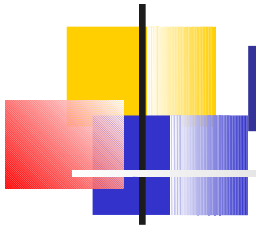
Quanto robusto





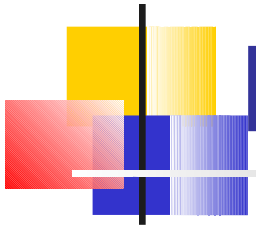
Quanto robusto

- Valore dipende dai controlli che il componente esegue sul proprio ambiente
- Robustezza $\rightarrow$1 se numero dei controlli $\rightarrow$ infinito
- Normalmente i controlli sono inutili perchè i comportamenti anomali si presentano con frequenza estremamente bassa (non c'è struttura nell'anomalia)
- Compromesso necessario tra il numero dei controlli e le prestazioni possibili = I controlli sono istruzioni eseguite dal sistema simili a quelle per implementare il sistema



Robustezza

- è stato provato come controlli anche banali siano in grado di rilevare numerose situazioni anomale
- quindi è bene eseguire controlli sofisticati solo dopo quelli semplici
- i controlli più significativi sono quelli sull'uso dei dati coerenti con i tipi dichiarati = legame tra robustezza e sistema dei tipi



Robustezza in biologia

- Ridondanza
- Feedback
  - analisi di comportamento
  - Regolazione in base al comportamento
- Modularità
- Confinamento di comportamento anormale
- Eliminazione e sostituzione di componenti difettosi
- Assenza di centralizzazione

Il complemento genera delle vulnerabilità

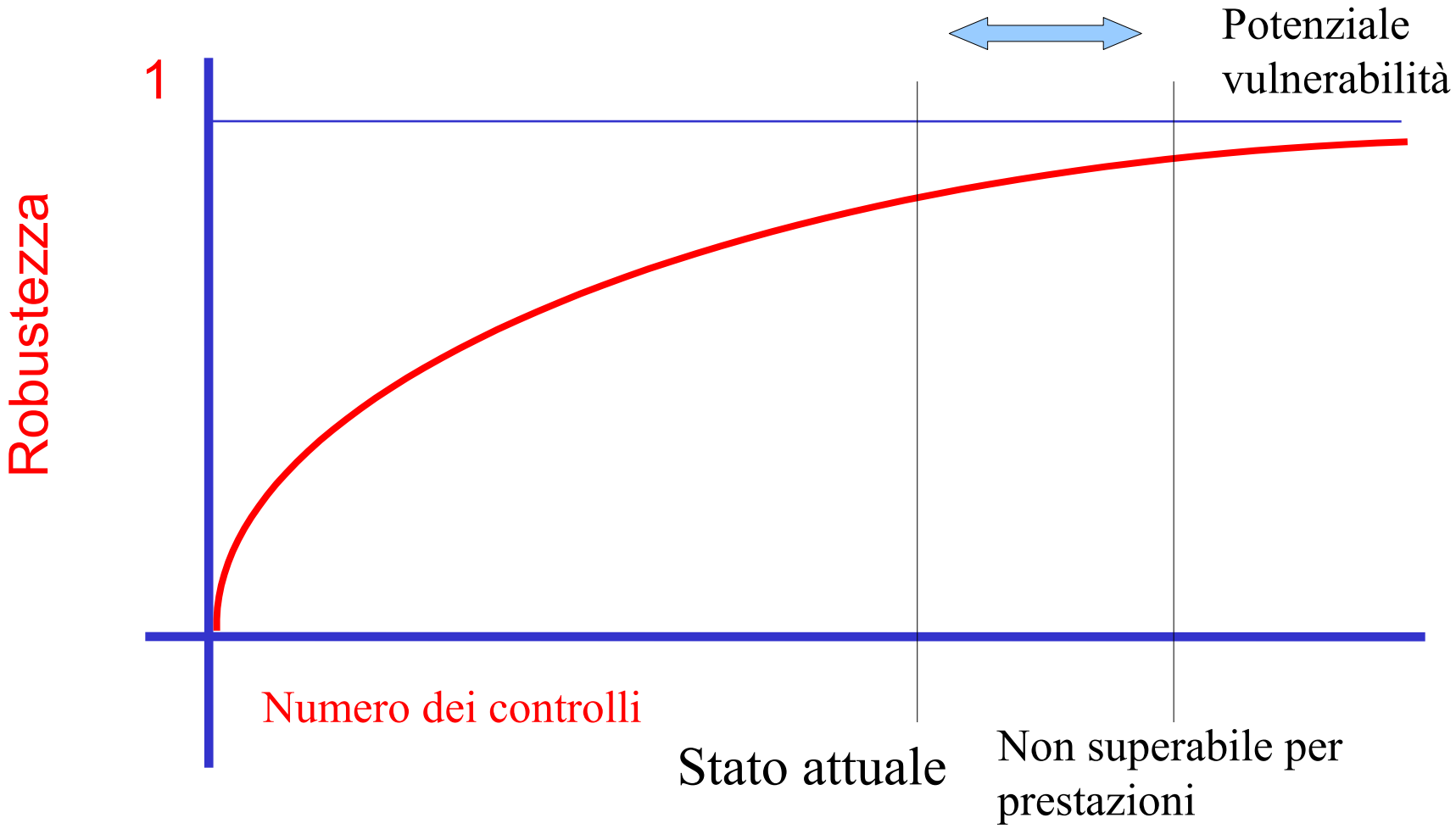
Robustezza e Vulnerabilità

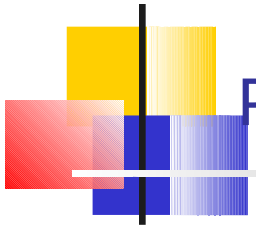
- Utilizzare la robustezza per trovare vulnerabilità equivale a confrontare il sistema da analizzare con un sistema ideale o asintotico che è perfettamente robusto perchè applica tutti i controlli possibili
- Questo equivale ad assumere che *ogni insieme di regole per progettare in modo robusto definisce anche un insieme di regole per la ricerca di vulnerabilità*
- Ogni differenza tra sistema asintotico e quello reale è una potenziale vulnerabilità
- L'effettiva esistenza della vulnerabilità dipende dal contesto del sistema da analizzare perchè il comportamento generato potrebbe essere ininfluente nel contesto specifico

Robustezza e Vulnerabilità

- Alcune differenze tra il sistema asintotico e quello reale possono essere ammissibili, poiché soddisfare i principi di progetto avrebbe portato ad un sistema che viola le specifiche di prestazioni
- Altre differenze non possono essere giustificate dalle prestazioni
- Individuare le differenze non ammissibili è la chiave per individuare le vulnerabilità

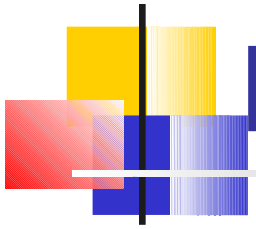
Quanto robusto





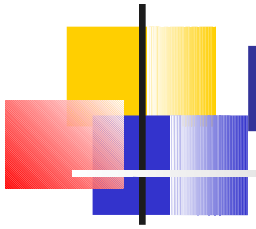
Principi di progetto per la robustezza (Saltzer&Schroder)

- Economia dei meccanismi
- Fail safe default (Default deny)
- Mediazione completa
- Open design
- Separazione dei privilegi
- Privilegio minimo
- Least common mechanism
- Accettabilità psicologica
- Proporzionalità ai rischi
- Ricordare le violazioni



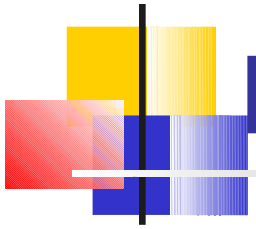
Principi di progetto

- Valgono sia per il sistema originale che per i meccanismi di sicurezza che si introducono
- Sono fondamentali non solo nel progetto di un sistema sicuro ma anche nell'analisi delle vulnerabilità perchè ogni volta che non sono rispettati potenzialmente esiste una vulnerabilità



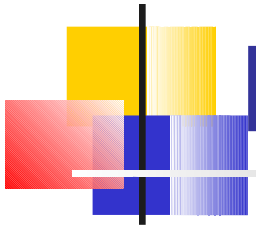
Principi di progetto

- Valgono sia per il sistema originale che per i meccanismi di sicurezza che si introducono
- Sono fondamentali non solo nel progetto di un sistema sicuro ma anche nell'analisi delle vulnerabilità perchè ogni volta che non sono rispettati potenzialmente esiste una vulnerabilità



P1-Economia dei meccanismi

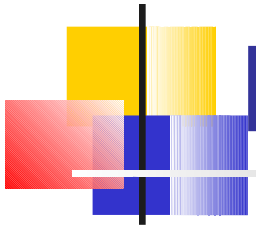
- Un meccanismo deve essere semplice poichè le vulnerabilità aumentano con la complessità del meccanismo e quindi del codice per implementarlo
- Quello che si vuole evitare è non tanto la complessità quanto quella inutile
- Sfruttare la possibilità di comporre meccanismi
- Hardening di un SO = eliminazione delle funzionalità inutili per le specifiche applicazioni supportate



P1- Semplicità dei meccanismi

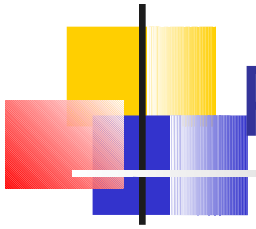
- Esokernel = Nucleo minimo

  Spostare l'implementazione delle politiche di un sistema all'esterno del kernel in modo da limitarne la complessità e quindi gli errori
- Forte integrazione tra gli strumenti (violando i principi di incapsulazione) amplifica gli effetti di una vulnerabilità



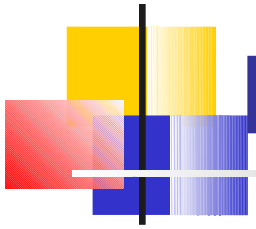
P1- Semplicità dei meccanismi

- Costruire un sistema in modo che interfaccia contenga delle operazioni semplici
- L'operazione complessa viene implementata componendo le operazioni semplici
- Con una interfaccia che contiene poche operazioni molto complesse posso essere costretto a permettere ad un utente di eseguire una operazione più potente di quella necessaria perché è l'operazione più semplice del sistema = correlato a Least common mechanism



P2-Fail safe default (Default deny)

- Il caso di default deve essere sicuro =
  di default un diritto NON esiste
- Questo principio richiede che sia possibile eseguire una operazione solo quando sono state eseguite le apposite azioni per concedere il diritto
- Quando un sistema parte deve entrare in uno stato sicuro = lo stato iniziale deve essere sicuro

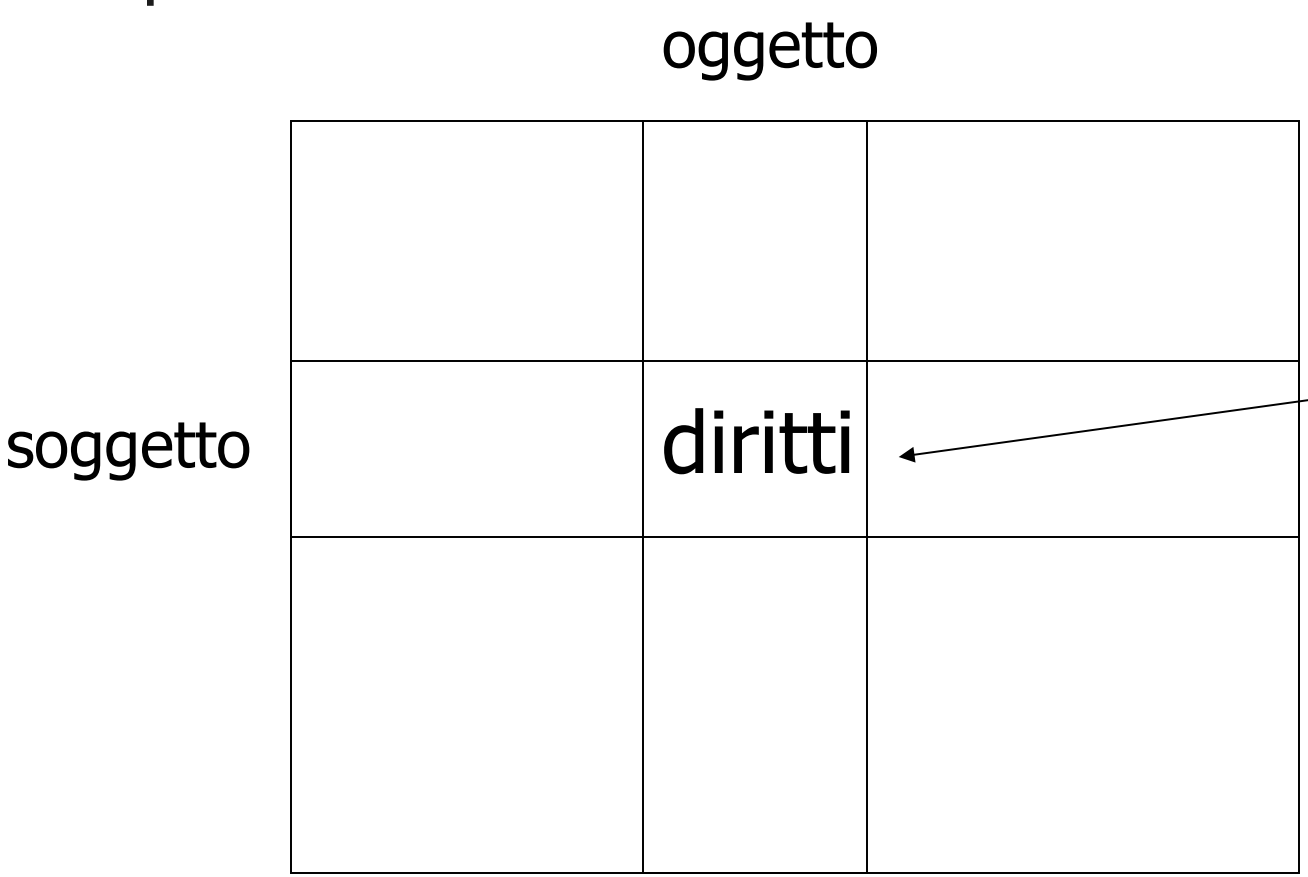


P3-Mediazione completa

- Ogni volta che un soggetto tenta di eseguire una qualunque operazione occorre verificare che possieda il diritto necessario
- Non possono esistere operazioni che sfuggono ai meccanismi

  oppure
- Ogni operazione che sfugge è una vulnerabilità perchè può essere invocata illegalmente

Matrice di controllo degli accessi



Quali operazioni Definite dall'oggetto

Il soggetto puo' invocare

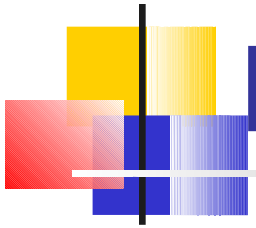
Condizione

1. necessaria
2. non sufficiente

per sicurezza

Matrice di controllo degli accessi

- La sicurezza richiede che esista questa matrice per ogni macchina virtuale
- Quindi esiste una matrice per ogni livello di astrazione
- In taluni casi (= macchine applicative) il numero di soggetti può essere cosi elevato che la matrice deve essere memorizzata in memoria secondaria



Diritti nella posizione i,j -I

- Assegnati dal proprietario dell'oggetto j nel caso di una politica DAC
- Assegnati anche in base alla classe(livello) di i ed a quella di j nel caso di MAC
- In entrambi i casi l'assegnamento dovrebbe essere una condizione necessaria che diventa anche sufficiente in base alla evoluzione della computazione e quindi al tempo in cui ci si interrroga sui diritt

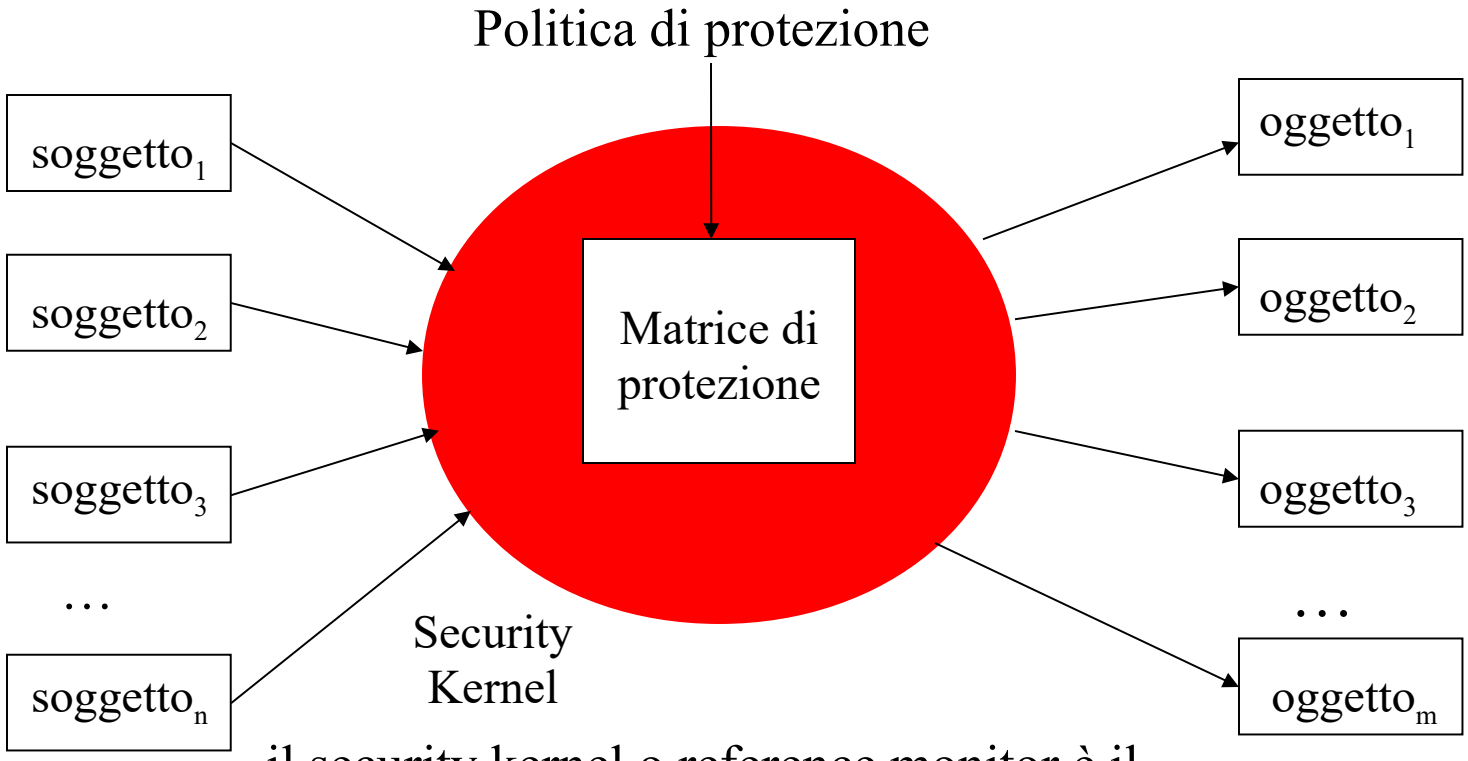
Diritti nella posizione i,j -II

- La matrice di controllo degli accessi (o di protezione) è una struttura dinamica
- Dinamicità deriva
  - dal fatto che soggetti ed oggetti possono essere creati dinamicamente
  - Dal fatto che

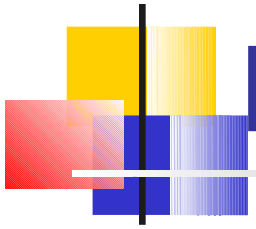
    - alcune politiche sono dinamiche

    - in generale il possesso di diritto dipende dalle operazioni eseguite in precedenza

Matrice CA: tipica implementazione

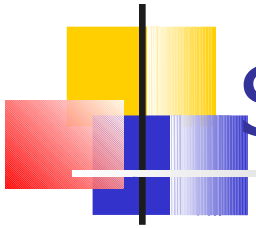


il security kernel o reference monitor è il componente che media le invocazioni dei soggetti agli oggetti in base alla matrice di controllo degli accessi



Matrice CA

- La matrice è una struttura logica, in ogni sistema può essere implementata da una diversa struttura fisica (non cercate sempre una matrice) o da una funzione
- In alcuni servizi di rete è bene ricordare che i soggetti sono potenzialmente infiniti
  - Riga della matrice = classe di soggetti
  - In questo caso è anche necessario definire delle regole che permettano di associare un soggetto ad una classe

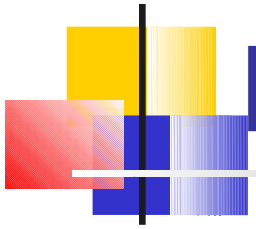


Security Kernel o Reference Monitor

- Appartiene al Trusted Computing Base (TCB) = la sua correttezza è necessaria per la corretta implementazione della politica
- Ridotto per poterne provare in modo formale o semiformale la correttezza che è fondamentale per quella di tutto il sistema
- Base per prove induttive di sicurezza: dimostrare che il security kernel è affidabile è il punto di partenza per poter provare alcune proprietà
- Spesso memorizzato in componente tamper proof

Tamper proof

- Un componente in cui ogni tentativo di accesso fisico viene
  - Impedito
  - Rilevato
- Le memorie ed i collegamenti con le memorie sono immersi nel silicono
- Le memorie sono protette da un circuito elettrico a forma di griglia che ne cancella il contenuto se si tenta di accedere fisicamente alle memorie



Matrice di protezione

- Assunzione implicita dell'uso della matrice di protezione è quello dell'autenticazione del soggetto $\Rightarrow$ come si controlla che chi afferma di essere A sia effettivamente A
  - Controllo esplicito da parte del security kernel
  - Password
  - One-time password
  - Challenge response
  - Firma elettronica

One time password

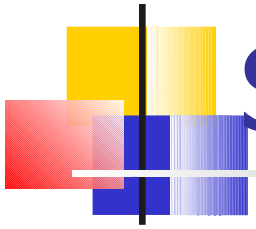
- Una funzione F che ha almeno due parametri
  - S un valore segreto
  - N il numero di richieste ricevute (implicito o esplicito)
- Chi si deve autenticare calcola F(S, N) e trasmette il risultato
- Chi riceve il risultato ricalcola e controlla
- Sincronizzazione sul valore di N

Challenge - response

- Entrambi i partner conoscono F
- F dipende da un parametro x
- Uno dei partner spedisce y (sfida)
- Chi la riceve calcola F(y) e restituisce il risultato
- Chi ha sfidato ricalcola F(y) e controlla

Soluzioni per la mediazione

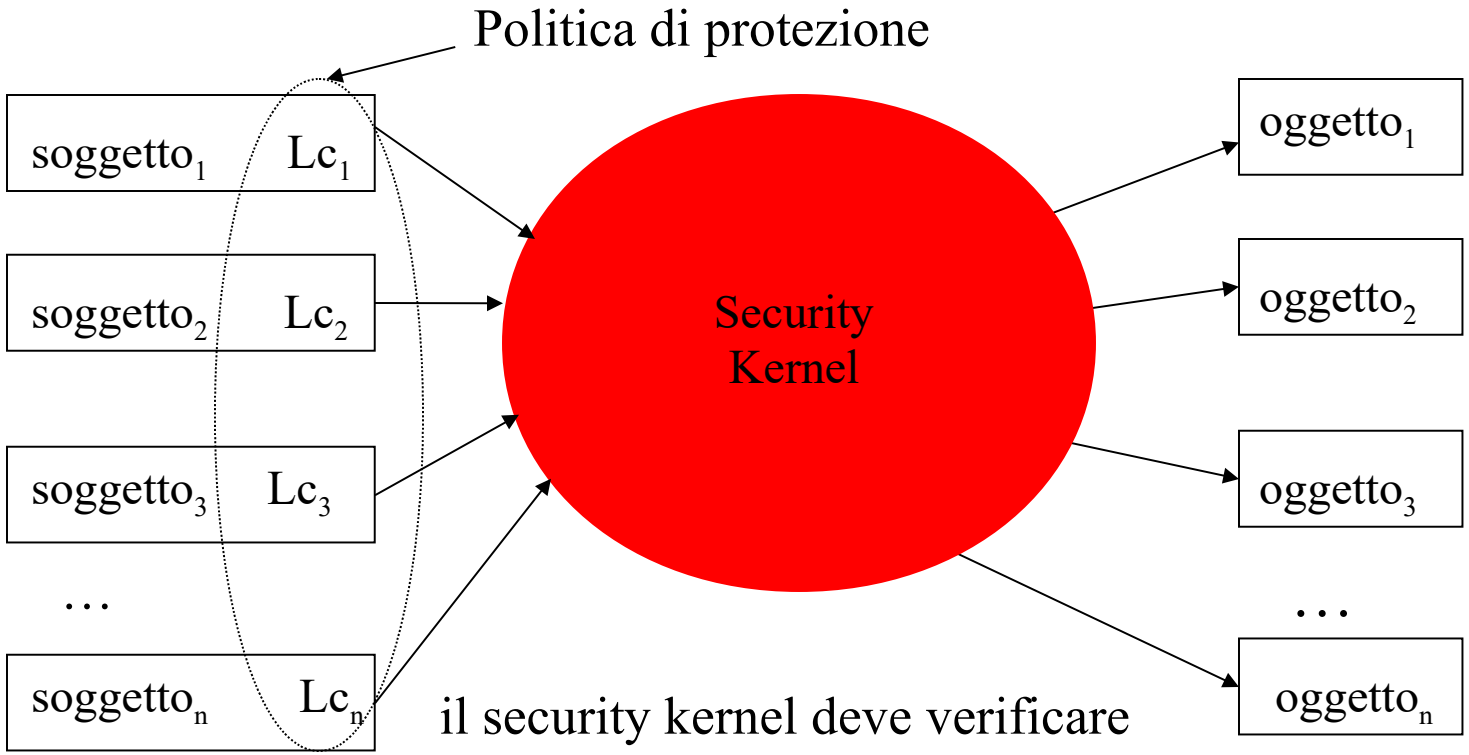
- Occorre implementare soluzioni efficienti visto l'elevato numero di controlli
- Molto difficile ottenere efficienza con una soluzione centralizzata che ricordi tutti i diritti su tutti gli oggetti
- Necessaria una soluzione distribuita e scalabile = indipendente o poco sensibile al numero di soggetti e oggetti



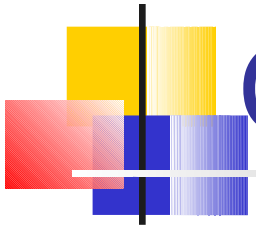
Soluzioni per la mediazione - 1

- Capability = memorizzazione della matrice per righe = per soggetti
  - Una lista di capability per ogni soggetto
  - <indirizzo, diritti> = generalizzazione del concetto di puntatore
  - Quando il soggetto invoca una operazione deve specificare la capability (cioè sia l'indirizzo che il diritto) che vuole usare

Matrice CA: Lista di capability

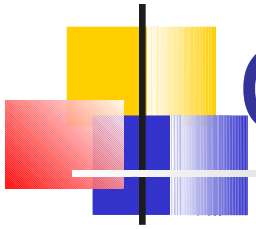


il security kernel deve verificare le capability trasmesse ma non deve accere direttamente alla matrice



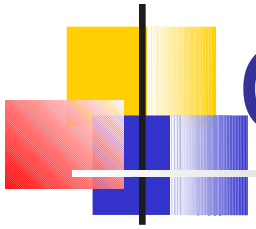
Capability -I

- Invocazione $op_i(ogg_j, par, n)$ = invoca la i-esima operazione del j-esimo oggetto usando l'n-sima capability
- Semplifica la delega dell'esecuzione delle operazioni poichè è possibile trasferire la capability ad un altro soggetto
- Capability = ticket per la risorsa
- Provoca una prolificazione dei diritti che impediscono di revocare velocemente il diritto che è stato delegato ad altri



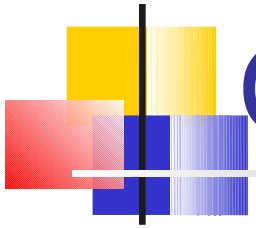
Capability - II

- I diritti di un soggetto fanno parte della rappresentazione interna del soggetto
- Le capability sono generate dal security kernel che le distribuisce ai soggetti
- Il pericolo è che il soggetto possa manipolare la lista in modo da attribuirsi diritti
- Solo il security kernel deve poter generare una capability e manipolarla
- Soggetto può solo usarla (read) o delegare (copia)



Capability -III

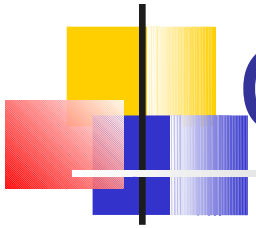
- Possibile un efficiente supporto harware e firmware mediante la MMU
- La lista delle capability viene caricata nella MMU
- Il controllo é realizzato dalla MMU in parallelo alla traduzione degli indirizzi
- La MMU controlla anche che il soggetto non modifichi i propri diritti



Capability -IV

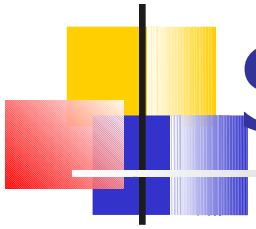
- La traduzione degli indirizzi utilizza una tabella dove sono memorizzati gli indirizzi fisici delle pagine
- In ogni posizione della tabella vengono inserite delle informazioni sulle operazioni possibili per il processo in esecuzione = forma di capability molto semplice

  (read, write, fetch)
- In taluni processori se una pagina logica è nel cache non si controllano le operazioni



Capability - V

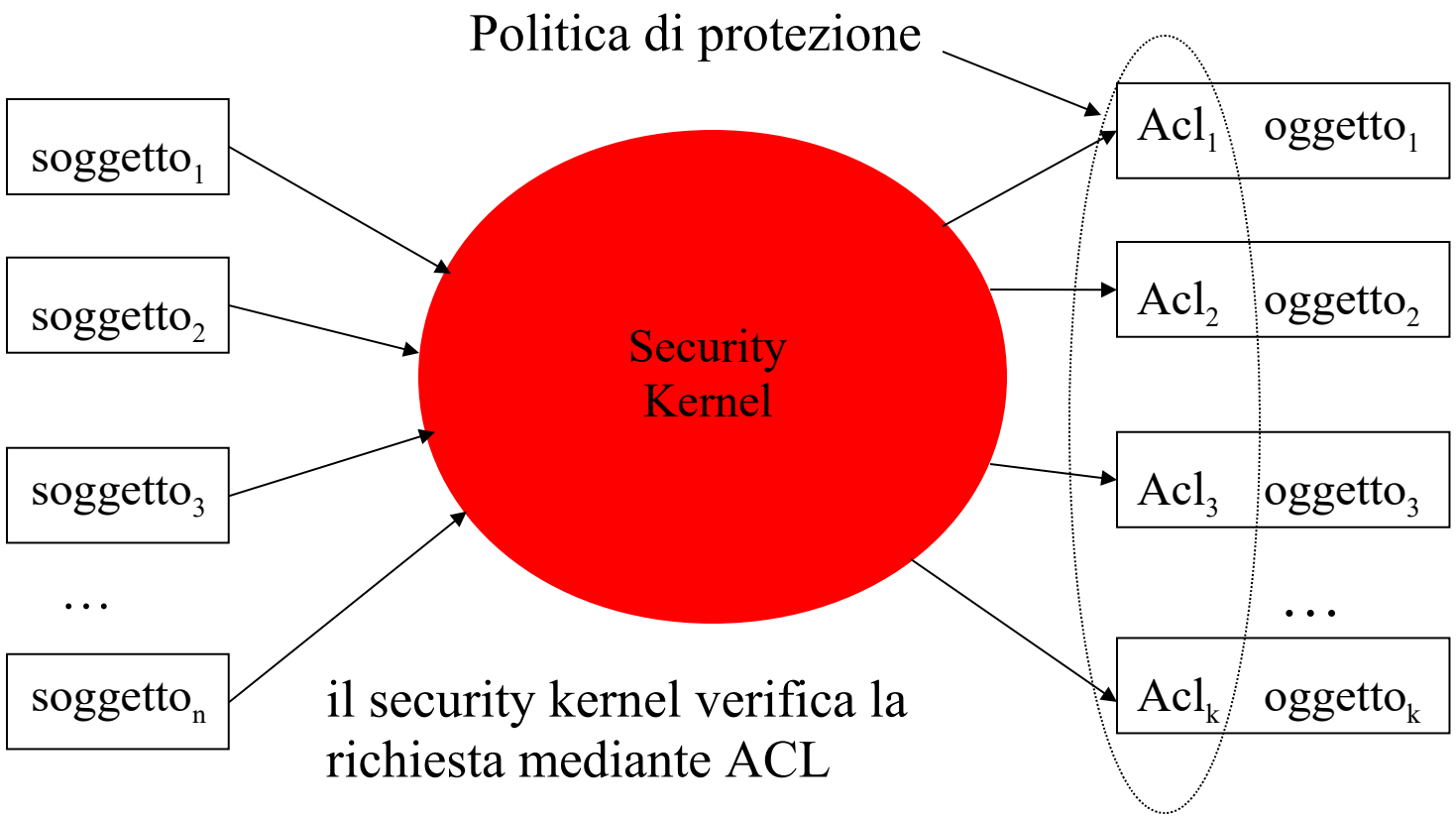
- La nozione Posix di capability è leggermente diversa da quella teorica (presente anche in Linux)
- Capability posix non è un puntatore protetto ma sono una stringa di bit che definisce operazioni possibili per un certo processo in modo assoluto
  - Cambiare owner di un file
  - Inviare segnali a chiunque
- Possibile anche specificare quali possono essere ereditate dai discendenti
- Un meccanismo per limitare i diritti più che per gestirli e trasmetterli



Soluzioni per la mediazione - 2

- Lista di controllo degli accessi = memorizzazione per colonne
- Una lista per ogni oggetto
- Ogni elemento della lista ricorda i diritti di un soggetto
- Anche in questo caso il controllo e' eseguito da un componente terzo
- Diritti centralizzati in un oggetto

Matrice CA: ACL



Lista di controllo degli accessi

- Una gestione più flessibile grazie a
  - insiemi di utenti
  - sequenzialità della scansione della lista (soggetto non conosce la posizione, è necessaria la scansione)

If utente $\in$Insieme1 *then* {op1, op2}

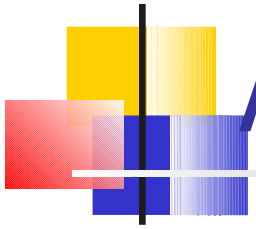
else If utente $\in$Insieme2 *then* {op3, op4}

else {op5}

Notare che mediante else finale posso concedere anche un diritto ad un utente di cui non conosco il nome, cosa impossibile con le capability (poco adatte perciò a siti web a meno di non usare transitività)

Supporto Hardware per ACL

- Memoria associativa che ha come chiave
  - Utente
  - Utente operazione
- FPGA come implementazione di una funzione binaria che comprende
  - Confronto utente con i vari casi
  - Gestione di priorita nel caso di confronti con successo



ACL e File Unix

Il gruppo di bit che viene associato ad un file e che specifica

- I diritti dell'owner
- I diritti del gruppo dell'owner
- I diritti di tutti gli altri

è una ACL associata al file

ACL e descrittore

struct stat {

mode_t st_mode; // File type & mode **access control list + set uid bit**

ino_t st_ino; // i-node number

dev_t st_dev; // device number (file system)

dev_t st_rdev; // device n. for special files

nlink_t st_nlink; // number of links

uit_t st_uid; // user ID of owner

gid_t st_gid; // group ID of owner

off_t st_size; // size in bytes, for reg. files

time_t st_atime; // time of last access

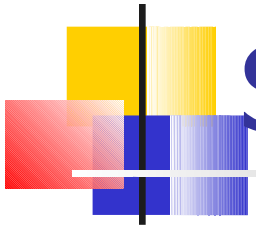
time_t st_mtime; // time of last modif.

time_t st_ctime; // time of last status change

long st_blksize; // best I/O block size

long st_blocks; // number of 512-byte blocks

}

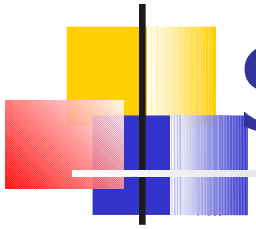


Sistemi Unix/Linux -I

- LCA utilizzano gli identificatori di un processo
  - Real user ID → owner
  - Effective user ID
  - Saved user ID

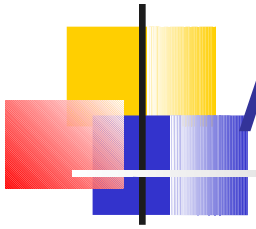
in Linux esiste anche

- File system ID



Sistemi Unix/Linux -II

- real UID: UID di chi ha creato il processo
- effective UID: UID utilizzato per la valutazione dei privilegi
- saved UID
  - binari SUID (set-user-ID):
    saved UID = UID owner del file
  - binari non SUID: saved UID = real UID



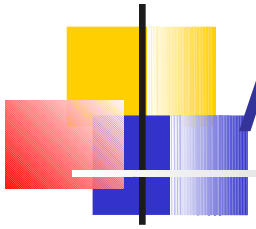
ACL & Router

- Lista costruita a partire da due casi fondamentali

  $Range_1$ IP $\rightarrow$ route

  $Range_2$ IP $\rightarrow$ drop

- Posso avere una lista per ogni interfaccia di ingresso/uscita in modo da controllare il traffico che attraversa il router anche se non conosco gli indirizzi di chi genera il traffico che attraversa il router

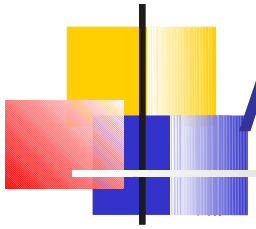


ACL & Router

- ACL di ingresso 1
  - 131.114.*.* → route
  - 131.4.5.6 → route
  - 131.4.*.* → drop

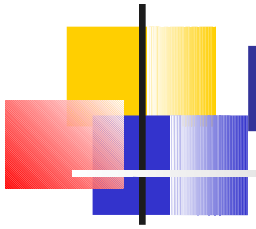
  dalla rete 131.4.*.* solo il traffico da 131.4.5.6 può passare il resto viene eliminato
- ACL di uscita 1
  - 131.114.*.* → drop
  - 131.4.*.* → drop

  nessun nodo 131.4.*.* può comunicare con la rete collegata all'interfaccia 1



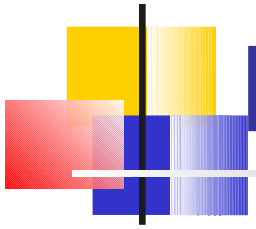
ACL & Router & Linux

- Si possono determinare le regole di filtraggio dei pacchetti in Linux mediante
  - Netfilter nel kernel
  - Iptables per configurazione a livello superiore
- Filter table: determina quali pacchetti passano mediante le catene nel seguito
- Nat table: riscrittura di alcuni campi
- Mangle table: opzioni su QOS



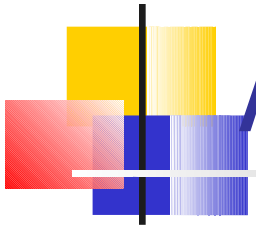
Filter table

- Input chain: regole per i pacchetti destinati a questo nodo
- Output chain: regole per i pacchetti prodotti dal nodo
- Forward chain: regole per i pacchetti che attraversano il nodo
- Default allow



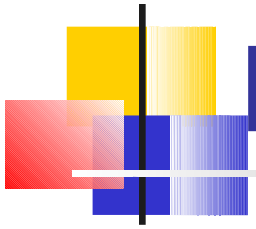
Nat table

- Prerouting chain= tutti i pacchetti in entrata
- Postrouting chain = tutti i pacchetti in uscita
- Output chain = tutti i pacchetti prodotti



Azioni possibili

- Drop
- Route
- Return – ritorna a catena chiamante
- Queue – passaggio in spazio utente
- Log
- Reject
- Dnat/Snat/Masquerade



Esempio

- iptables –A INPUT –p UDP drop

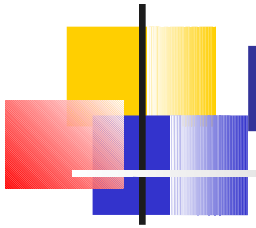
  aggiungi alla catena input una regola che scarta tutti i pacchetti UDP
- iptables –A INPUT –p TCP –dport 156 drop

  elimina i pacchetti TCP destinati alla porta 156
- iptables – N nuovicontrolli

  crea una nuova catena a cui poi si possono aggiungere controlli

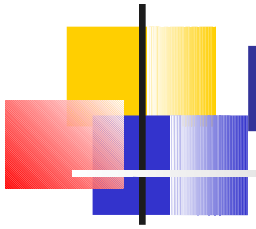
Controllo degli Accessi ed Utenti

- Può essere problematico definire le regole per il singolo oggetto e quindi il contenuto della matrice a causa del numero elevato di oggetti e soggetti
- Occorrono quindi dei meccanismi per semplificare la definizione della matrice di protezione viste le sue elevate dimensioni



Ruolo e non Soggetto

- Usato per semplificare la gestione
- Ruolo = una funzione che può essere svolta da un utente(soggetto), spesso in corrispondenza a quello professionale: medico, primario, infermiere
- Ad ogni ruolo corrisponde
  - un insieme di soggetti che lo possono assumere
  - un insieme di diritti
- I diritti sono gestiti mediante ruolo visto come classe di equivalenza tra soggetti e non considerando il singolo soggetto
- Role Based Access Control

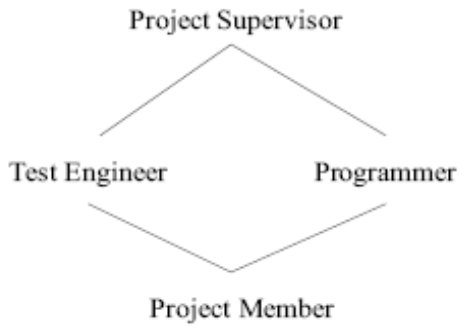


Ruolo - II

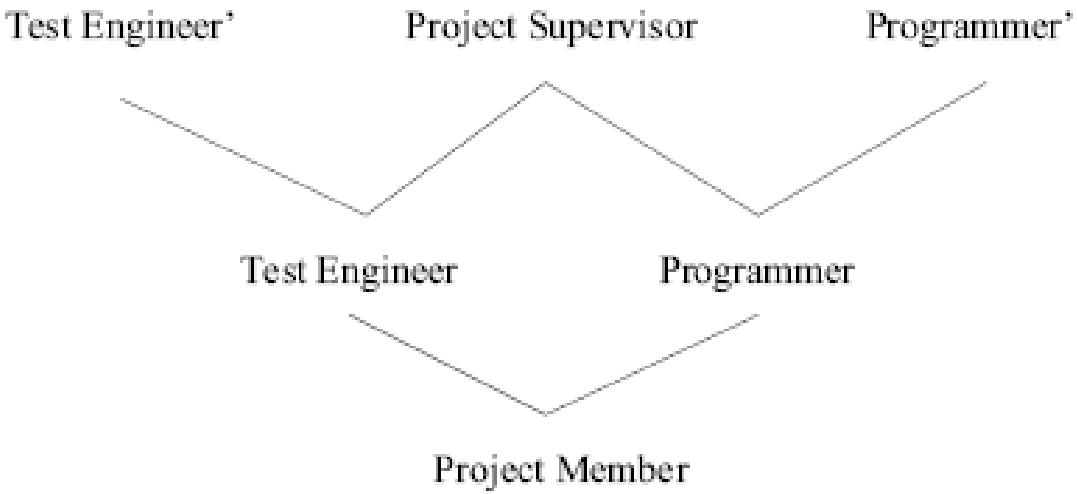
- Sono definite regole che definiscono come e quando un utente può cambiare ruolo
- Queste regole possono dipendere dalla storia delle operazioni eseguite
- Possibile anche specificare una password per il cambiamento

Gerarchia di ruoli - I

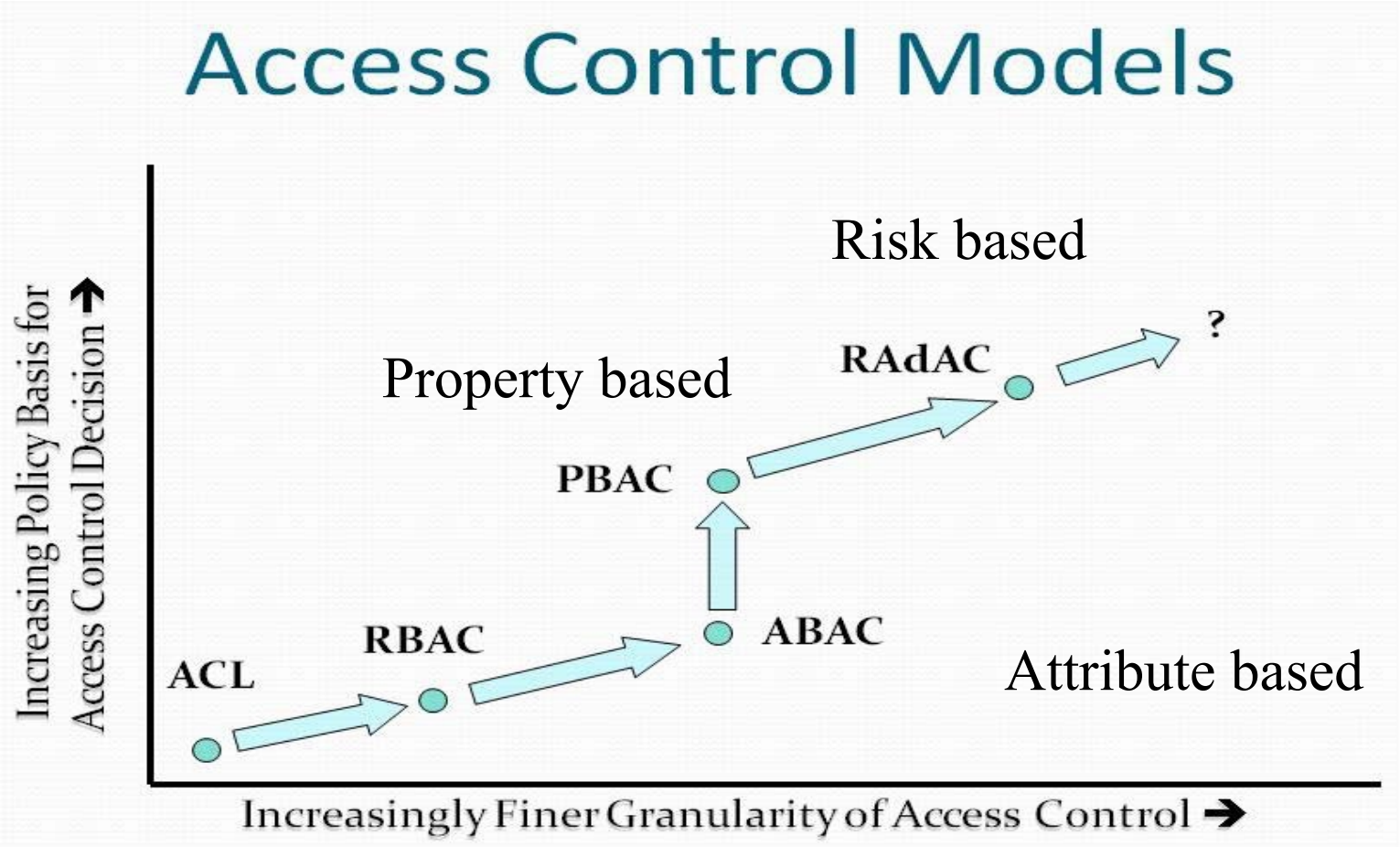
- un ordinamento parziale tra i vari ruoli
- Un ruolo è maggiore di un altro se comprende tutti i suoi diritti

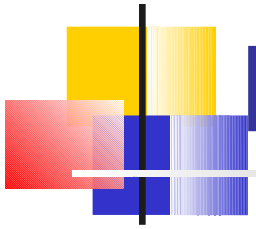


Gerarchia II



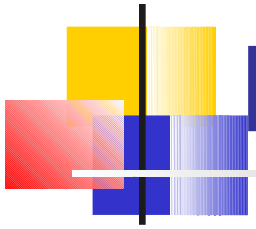
Altri modelli





P4-Open Design - I

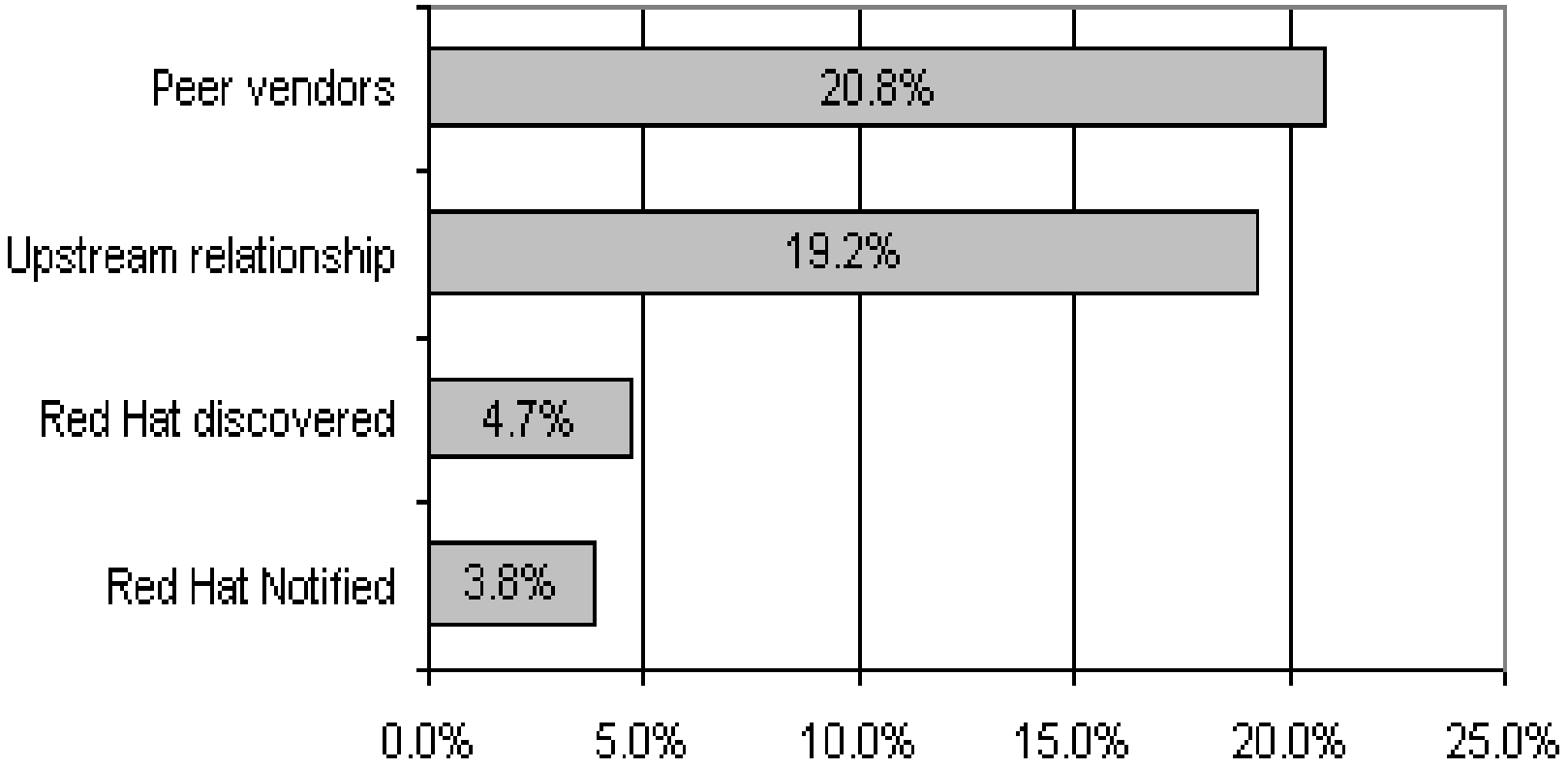
- La sicurezza offerta da un meccanismo non deve dipendere da caratteristiche non pubbliche
- Gli algoritmi utilizzati dai meccanismi di sicurezza devono essere noti
- Caratteristica molto controversa

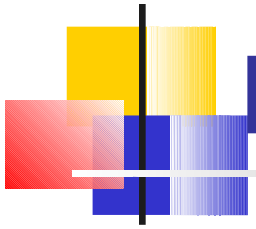


P4-Open Design - II

- La peer review di un progetto è fondamentale per scoprire le vulnerabilità del progetto stesso
- Ciò implica però che la disclosure è utile solo se c'è una peer review di cui si conoscono i risultati
- Se non c'è peer review o non si conoscono risultati open source è inutile (non aumenta la sicurezza)
- Vantaggi e svantaggi di codice open source

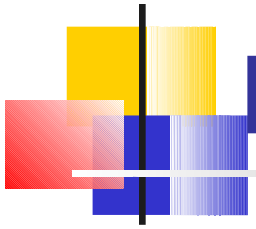
Vulnerabilità e open design





P5 - Separazione dei privilegi

- Il numero di autorizzazioni (diritti) necessario per eseguire una operazione deve essere proporzionale al rischio corrispondente all'operazione
  - Probabilità di attacco con successo
  - Impatto dell'attacco
- Per ogni diritto un controllo diverso ed indipendente
- Banche = piu' firme di autorizzazione
  in base al valore di un assegno

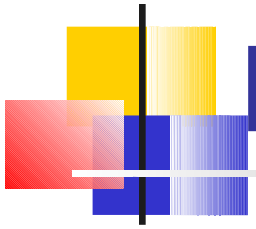


P5 - Separazione dei privilegi

- Operazione complessa viene definita componendo operazioni elementari
- Diritti diversi per operazioni elementari
- Controllare esplicitamente
  - Il diritto di invocare l'operazione complessa
  - Il diritto di invocare le singole operazioni necessarie per quella complessa

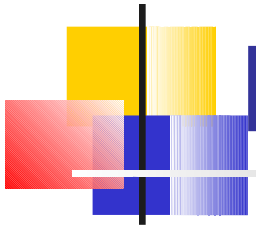
Esempio

- Op = trasferisci somma da conto1 a conto2
- 5 diritti
  - Invoca op
  - Leggi conto1
  - Scrivi conto1
  - Leggi conto2
  - Scrivi conto2
- Sarebbe sbagliato dedurre che un soggetto possiede gli ultimi 4 dal fatto che possieda il primo



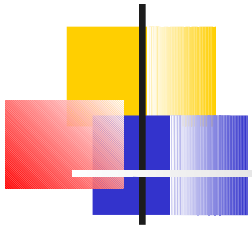
P6 - Privilegio Minimo - I

- Un soggetto deve avere *tutti e soli* i diritti necessari per le sue funzioni (o il suo ruolo) e per il tempo necessario ad eseguire funzioni
- Ogni diritto inutile assegnato ad un soggetto è una vulnerabilità
- La matrice di protezione è una struttura che si evolve nel tempo
- È necessario concedere e revocare diritti mentre la computazione si evolve per rispecchiarne l'evoluzione



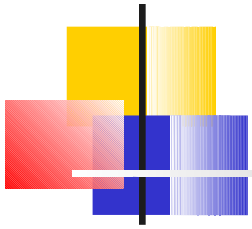
P6 - Privilegio Minimo - I

- Il principio si applica anche se la politica di sicurezza fosse statica perchè definisce come gestire un diritto che il soggetto possiede legalmente ma che non sta usando
- Se, ad un certo istante, un soggetto non usa un diritto, allora esso deve essere cancellato dalla matrice ed inserito solo quando è necessario
- Versione estrema del (can know/need to know)



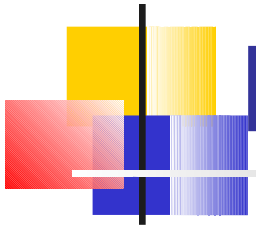
Privilegio Minimo - III

- Commutazione di dominio di protezione = non cambia il soggetto in esecuzione, come nella commutazione di contesto, ma vengono modificati i diritti nella riga della matrice di protezione (nella struttura concreta)
- Commutazione di dominio di protezione = modifica di una riga della matrice di protezione
- Il costo di questa commutazione dipende dalla specifica implementazione considerata e dal livello di soggetti ed oggetti



Privilegio Minimo - IV

- Un modo alternativo di definire questo principio è quello dei domini di protezione piccoli
- Al diminuire del dominio di protezione diminuiscono i rischi associati alle vulnerabilità che un soggetto può sfruttare

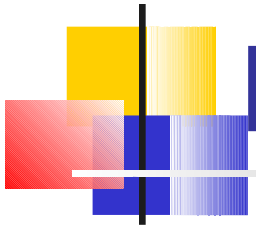


Privilegio Minimo - V

- E’ compito del progettista stabilire un compromesso perché questo principio se seguito fedelmente ha ripercussioni drammatiche sulle prestazioni

  $\Leftrightarrow$ ad ogni comando eseguito da un soggetto si deve modificare la matrice di protezione $\Leftrightarrow$ commutazione di dominio

  $\Leftrightarrow$ sistema asintotico è sovente troppo costoso



Privilegio Minimo - V

- E’ compito del progettista stabilire un compromesso perché questo principio se seguito fedelmente ha ripercussioni drammatiche sulle prestazioni

  $\Leftrightarrow$ ad ogni comando eseguito da un soggetto si deve modificare la matrice di protezione $\Leftrightarrow$ commutazione di dominio

  $\Leftrightarrow$ sistema asintotico è sovente troppo costoso

Privilegio minimo - VI

Chi innesca la transizione di dominio

1) Esistono comandi espliciti nel linguaggio di programmazione utilizzato
2) Il compilatore ed il supporto a tempo di esecuzione interagiscono per gestire i diritti

Privilegio minimo- Caso estremo

$op_i.ob_j(p_1, \ldots, p_n)$;

<calcolo locale1>

$op_h.ob_w(pa_1, \ldots, pa_m)$;

<calcolo locale2>

$op_z.ob_k(a_1, \ldots, a_w)$

Possiamo avere una commutazione di dominio

a) dopo la prima invocazione = perso diritto su $op_i.ob_j$
b) dopo il primo calcolo locale = acquisito diritto su $op_h.ob_w$
c) dopo la seconda invocazione = perso diritto su $op_h.ob_w$
d) dopo il secondo calcolo locale = …

Privilegio minimo- Grana diversa

$op_i.ob_j(p_1, \ldots, p_n)$;

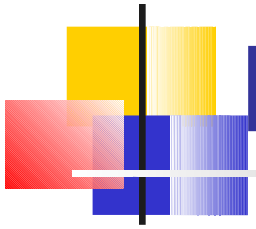
<calcolo locale1>

$op_h.ob_w(pa_1, \ldots, pa_m)$;

<calcolo locale2>

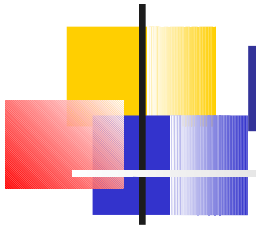
$op_z.ob_k(a_1, \ldots, a_w)$

a) ComDom dopo la prima invocazione =
   1. perdo diritto su $op_i.ob_j$
   2. acquisisco diritto su $op_h.ob_w$
b) ComDom dopo la seconda invocazione =
   1. perdo diritto su $op_h.ob_w$
   2. acquisisco diritto su $op_z.ob_k$



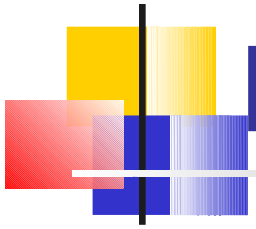
Privilegio Minimo - in pratica

- Una soluzione più realistica è quella che associa la commutazione di dominio
  - all'invocazione di una procedura (metodo)
  - al ritorno dalla procedura (metodo)
- Non si cambiano valori in una riga ma si crea/ distrugge una riga associata all'istanza di procedura
  - Creazione all'invocazione di procedura
  - Distruzione alla terminazione
  - Diritti associati all'istanza della procedura e non al codice statico della procedura stessa



Privilegio Minimo - in pratica

- I diritti inseriti nella nuova riga dipendono
  - dalle variabili locali al metodo (diritti dipendono dal tipo di dato),
  - dai parametri trasmessi (dal tipo di dato)
  - dalle modalità di trasmissione dei parametri)
- La strutturazione del programma in classi/ procedure determina la gestione dei diritti
- È possibile sviluppare programmi i cui gli insiemi dei diritti dei soggetti sono, istante per istante, piccoli e crescono solo quando è necessario
- Si semplifica lo sviluppo di strumenti automatici

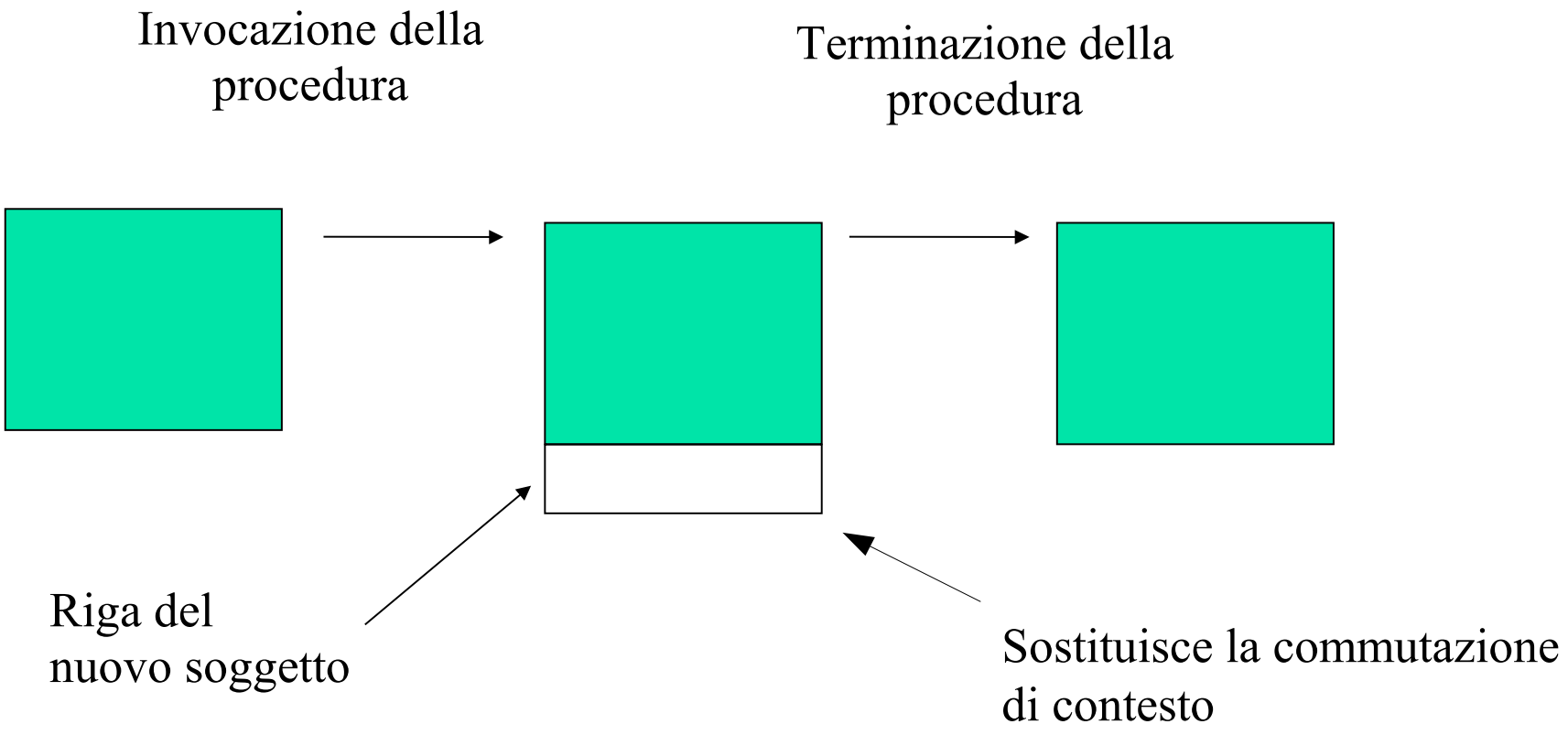


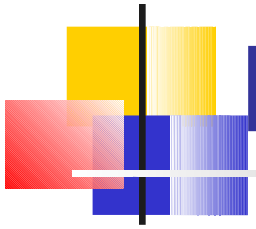
Esempio

Op(x, y)
    a : ….

- Se due programmi invocano questa operazione avremo due righe diverse
- Ognuna avrà diritti sulle variabili trasmesse come parametro e su una copia locale di a
- Solo se a è unica (dipende dalla semantica del linguaggio) avremo diritti su un oggetto condiviso
- Senza la creazione dinamica di una riga sarebbe distinguere diritti di soggetti che invocano op in momenti diversi con parametri diversi

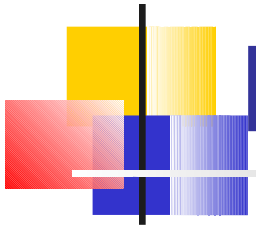
Esempio: la matrice di protezione





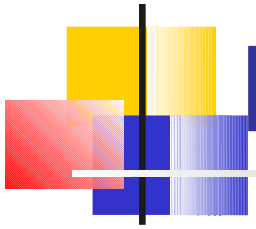
Privilegio Minimo - in pratica

- Notiamo che una metodologia ad oggetti che spinge a decomporre un sistema ad oggetti semplici ed interagenti è completamente coerente con questo principio
- Più gli oggetti sono semplici più i loro domini sono piccoli e possono essere controllati bene
- Anche se un oggetto viene violato, è semplice rilevare questo attacco
- Minimizzare comunque la condivisione



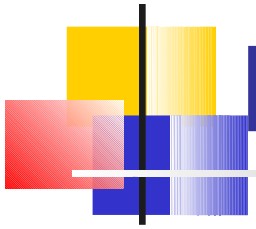
Privilegio Minimo - in pratica

- Spesso è utile che una procedura abbia diritti diversi da quelli dell'invocante
- In un linguaggio ad oggetti la procedura può conoscere l'implementazione di un oggetto, chi la invoca no
- Amplificazione dei diritti, una ulteriore operazione su un oggetto che deve essere permessa dalla matrice di protezione oppure regolata mediante la trasmissione dei parametri



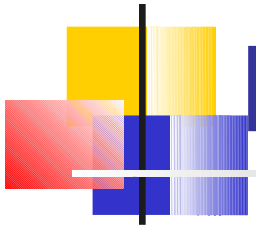
Privilegio Minimo - in pratica

- Nel caso di programmi i cui processi interagiscano mediante canali o porte è possibile avere una situazione analoga a quella delle procedure purchè
  - Interazioni diverse usano porte diverse,
  - le porte sono aperte e chiuse dinamicamente
  - se e quando una certa interazione è possibile, viene aperta una porta
  - la porta rimane aperta per il minimo tempo necessario all'interazione e poi viene chiusa



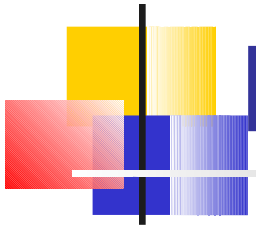
Privilegio Minimo - in pratica

- porta chiusa ≠ (porta aperta + meccanismo per scartare messaggi)
  - Se una porta è chiusa i messaggi ad essa diretti sono cancellati con un minimo sforzo
  - se la porta è aperta è richiesto più lavoro per scartare i messaggi
- Nel caso estremo l'unica cosa che il sistema fa è scartare messaggi (Denial of Service)



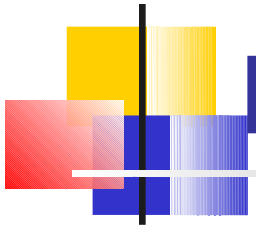
Privilegio Minimo – Unix - I

- Il principio viene violato per la presenza di un amministratore che ha tutti i diritti
- In questo caso si possono usare contromisure amministrative, ad esempio amministratori diversi che operano in tempi diversi
- Altre contromisure possono essere tecnologiche, ad esempio se non si può limitare root occorre ricordare le operazioni che ha eseguito in modo permanente



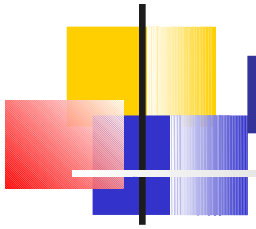
Privilegio Minimo – Unix - II

- Chroot limita l'accesso ai file definendo una nuova radice per il file system
- Jail (BSD) permette di limitare anche le connessioni di rete ed altre operazioni
- Implementazione del concetto di sandbox = un ambiente limitato per applicazioni untrusted



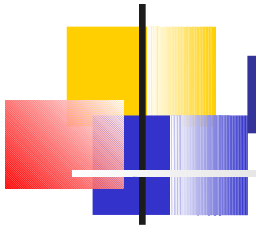
P7- Least common mechanism

- Un meccanismo estremamente potente, se necessario per il sistema, deve essere decomposto in meccanismi più semplici
- Se per eseguire operazioni diverse si usa un solo meccanismo molto potente allora
  - Molti utenti avranno diritto di usare il meccanismo
  - In realtà i diritti degli utenti sono su operazioni diverse = diversi sottoinsiemi della unica operazione di partenza
  - Sostanzialmente diritti inutili = violo privilegio minimo



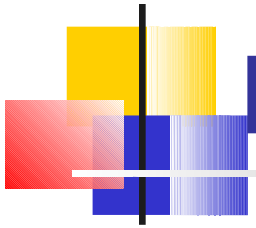
P7 – Least common mechanism

- Non ha senso applicare il privilegio minimo se le operazioni a cui si ha diritto sono eccessivamente potenti
- La decomposizione dei meccanismi in meccanismi di complessità ridotta permette di soddisfare meglio i principi di separazione e di privilegio minimo
- Si possono assegnare agli utenti solo i diritti sui meccanismi che devono invocare



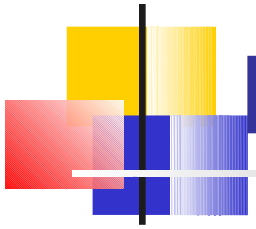
P8 - Accettabilità psicologica

- Un meccanismo troppo complesso da usare o che richiede troppo tempo innesca meccanismi per aggirarlo
- Se le password sono troppo difficili da ricordare gli utenti le scrivono sul post-it
- Se la rete prevede politiche di connessione troppo complesse gli utenti utilizzeranno un modem
- Sostanzialmente: non adottate politiche che già sapete saranno violate



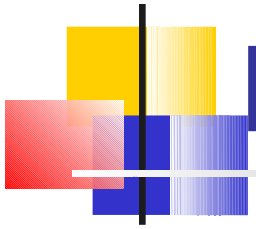
P9 - Proporzionalità ai rischi

- Anche dopo aver scelto di adottare una contromisura occorre sempre tener conto della probabilità che l'attacco avvenga ed abbia successo
- Concetto di Return On Investment, ogni investimento deve avere un guadagno proporzionale, nel nostro caso il guadagno è il rischio eliminato
- Necessaria una analisi del rischio preliminare



P10 - Ricordare le violazioni

- Occorrono meccanismi che permettano di scoprire che un attacco
  - è stato tentato
  - ha avuto successo
- Utile
  - per valutare la qualità dei meccanismi utilizzati
  - Per verificare ipotesi fatte nell'analisi del rischio su
    - Minacce
    - Attacchi
    - Obiettivi delle minacce

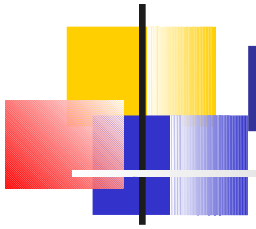


Ricordare le violazioni

- Permette di verificare le ipotesi sulle probabilità di attacco con successo
- Adottando la formula di Bayes permette di aggiornare distribuzioni di probabilità utilizzate per l'analisi
- Utile per capire se i tempi fissati a priori per rivedere l'analisi del rischio sono corretti

Ricordare le violazioni

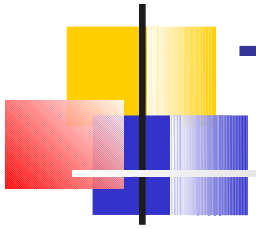
- File di log che ricordino almeno
  - Autenticazioni tentate
  - Autenticazioni fallite
  - Accesso a risorse critiche
- Protezione di file di log
  - Memoria write once
  - Inserzione di numeri di sequenza per impedire cancellazioni parziali
  - Inserire nell'iesimo dato una funzione non invertibile che dipenda da tutti i dati precedenti (hash)
- Forensics = utili per provare la violazione o il tentativo purchè sia garantità integrità



Logging policies

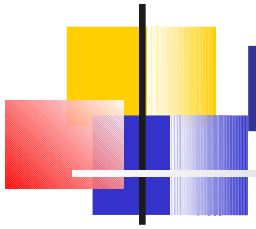
Gestione dei file di log

- Throw away – tutti i dati distrutti
- Reset – log files riinizializzati
- Rotate – si ruota tra file
- Compress and archive – memorizzati in un nastro o altri mezzi (esistono obblighi di legge, i.e. IP-persona)



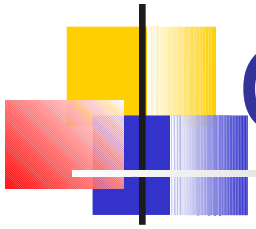
Throwing away log files

- sconsigliata
  - I log forniscono evidenza legale di intrusioni (marcatura temporale ed orologi sincronizzati)
  - Utili sia per sicurezza che per affidabilità
- anche senza obblighi di legge (IP-utente) devono essere conservati per almeno un paio di mesi
  - Tempo minimo talvolta per scoprire intrusioni



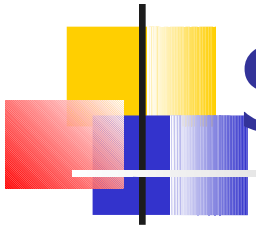
Rotating log files

- Mantenere file per n giorni
  - logfile, logfile.1 , logfile.2, … logfile.n
- Ogni giorno si utilizza logfile.n e si cambia nome agli altri



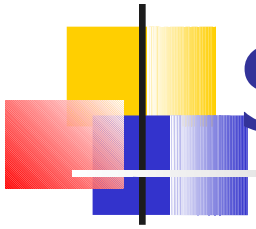
Compress and archive

- Spesso occorre ricordare informazioni di log per richieste
  - Indagini giudiziarie
  - Contestazioni di clienti
- Copia dei log su dispositivi rimovibili ed archiviabili
- Vincoli su conservazione



Syslog

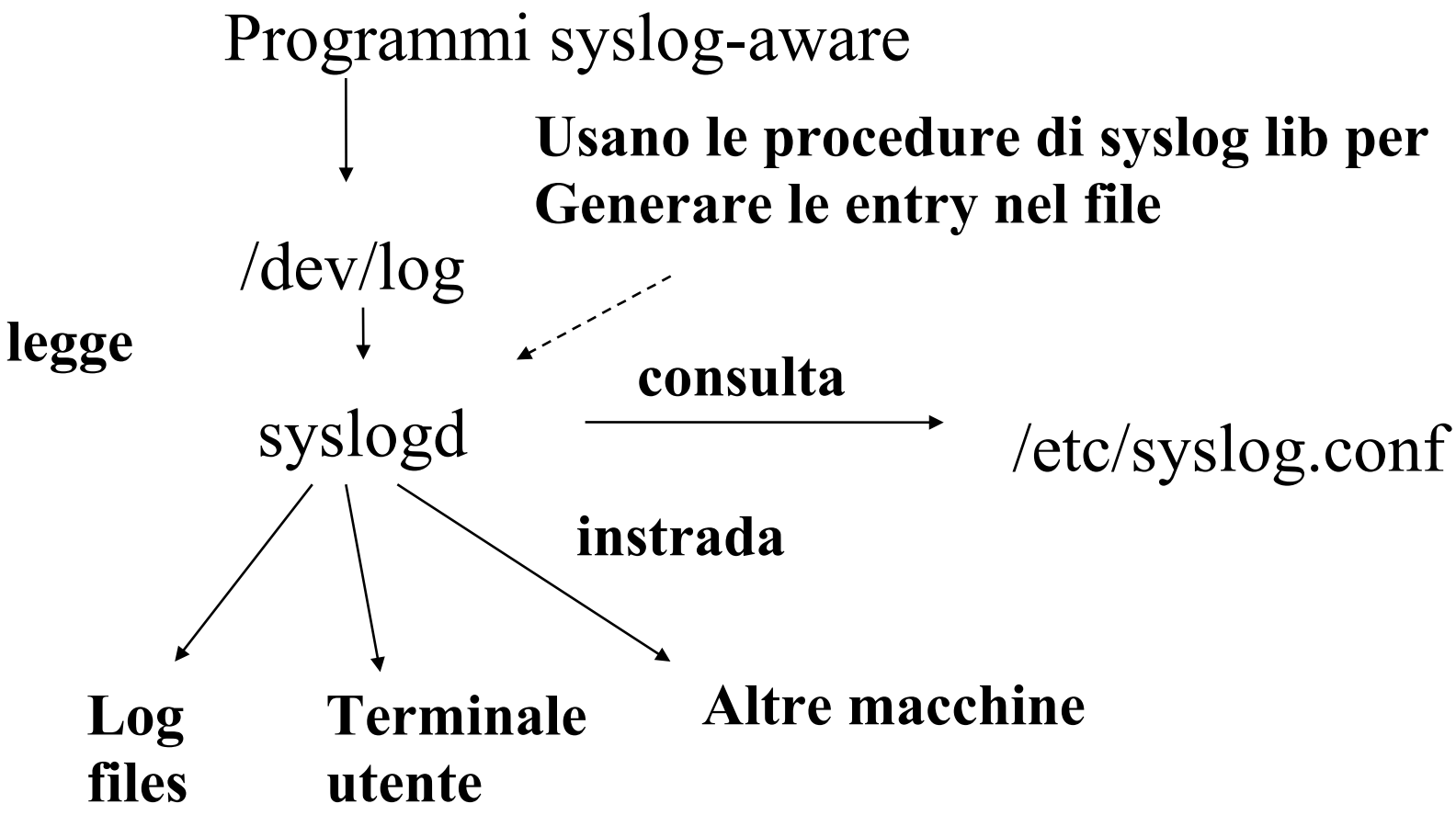
- Un sistema di logging per gestire le informazioni generate dal kernel e dalle system utilities.
- Permette di classificare messaggi in base a sorgente e criticità e di indirizzarli a più destinazioni

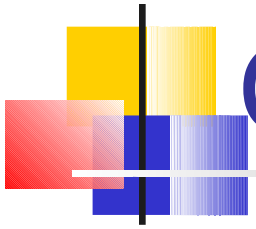


Syslog: 3 parti

- Syslogd /etc/syslog.conf
  - Il demone che implementa il logging
  - File di configurazione
- openlog, syslog, closelog
  - Procedure per generare eventi da loggare
- logger
  - Comandi utente per generare log

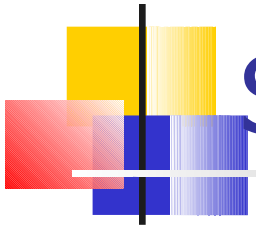
Syslog


Programmi syslog-aware
Usano le procedure di syslog lib per
Generare le entry nel file
/dev/log
legge
consulta
syslogd
/etc/syslog.conf
instrada
Log
files
Terminale
utente
Altre macchine



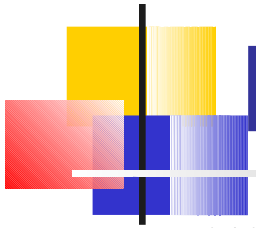
Configurazione syslogd

- Il file /etc/syslog.conf controlla il comportamento
- File di testo
  - ignorate le linee bianche e che cominciano con #
  - ***Selector*** <TAB> ***action***
  - mail.info /var/log/maillog



Selector

- Identifica
  - La sorgente – il programma (‘facility’) che sta inviando il messaggio
  - Importanza -- severity level del messaggio
- Sintassi
  - facility.level
  - facility names e severity levels devono essere scelti in una lista predefinita

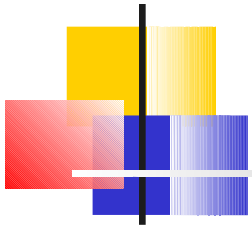


Facility names

Facility	Programmi che la usano
kern	kernel
user	user process, default
mail	mail system
daemon	System daemons
auth	Security and authorization related commands
lpr	printer spooling system
news	Usenet news system

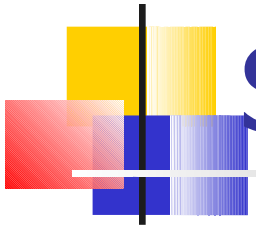
Facility names

Facility	Programmi che la usano
uucp	UUCP
cron	cron daemon
mark	Timestamps generati ad intervalli regolari (perchè)
local0-7	messaggi locali
syslog	syslog messaggi interni
authpriv	Messaggi di autorizzazioni privati o di sistema
ftp	ftp daemon, ftpd
*	altre facilities



Severity level

Livello	Significato approx.
emerg (panic)	Panic situation
alert	Urgent situation
crit	Critical condition
err	altri errori
warning	Warning
notice	merita una analisi
info	info
debug	info di debugging



Selector

- Più facilities separate da ‘,’
  - daemon,auth,mail.level action
- Più selettori combinari con ‘;’
  - daemon.level1; mail.level2 action
- I selettori possono essere messi in OR ‘|’ –un messaggio che matcha almeno uno sarà eleborato di conseguenza.
- Si possono usare ‘*’ o ‘none’, tutti o nessuno

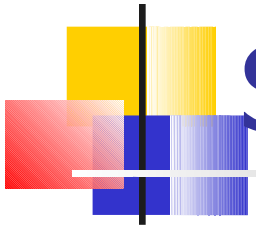
Selector

- Il livello indica la minima importanza di un messaggio perchè sia loggato
  - mail.warning, matcha tutti i messaggi dal mail system di importanza maggiore o uguale a warning
- 'none' esclude alcune facility indipendentemente dalle altre .
  - *.level1;mail.none action

Tutte le facility, livello maggiore di level1 con l'eccezione di mail

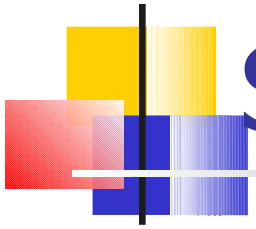
Azione: cosa fare di un messaggio

Azione	Significato
filename	Appende il messaggio ad un file locale
@hostname	inoltra il messaggio a syslogd di hostname
@ipaddress	inoltra il messaggio a host con indirizzo IP
user1, user2,…	scrivi il messaggio sugli schermi se gli utenti sono loggati
*	Scrivi il messaggio su tutti gli schermi



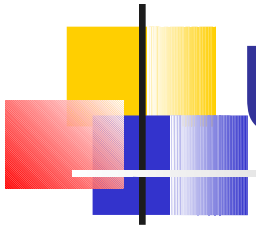
Software che usano syslog

Program	Facility	Levels	Description
amd	auth	err-info	NFS automounter
date	auth	notice	Display and set date
ftpd	daemon	err-debug	ftp daemon
gated	daemon	alert-info	Routing daemon
gopher	daemon	err	Internet info server
halt/reboot	auth	crit	Shutdown programs
login/rlogind	auth	crit-info	Login programs
lpd	lpr	err-info	BSD line printer daemon



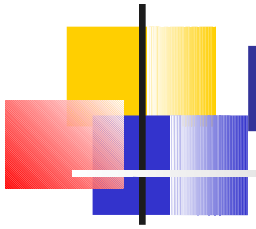
Software che usano syslog

Program	Facility	Levels	Description
named	daemon	err-info	Name server (DNS)
passwd	auth	err	Password setting programs
sendmail	mail	debug-alert	Mail transport system
rwho	daemon	err-notice	remote who daemon
su	auth	crit, notice	substitute UID prog.
sudo	local2	notice, alert	Limited su program
syslogd	syslog, mark	err-info	internet errors, timestamps



Uso di syslog

- openlog (*ident, logopt, facility*);
  - Apre connessione
  - I messaggi sono loggati con le opzioni specificati da logopt
  - Iniziano con la stringa ident
- Syslog (*priority, message, parameters*…);
  - Invio di message a syslogd, che lo logga con il priority level specificato
- close ( );



Logopt

- LOG_CONS

  Write directly to system console if there is an error while sending to system logger.
- LOG_NDELAY

  Open the connection immediately (normally, the connection is opened when the first message is logged).
- LOG_NOWAIT

  Don't wait for child processes that may have been created while logging the message. (The GNU C library does not create a child process, so this option has no effect on Linux.)
- LOG_ODELAY

  The converse of LOG_NDELAY; opening of the connection is delayed until syslog() is called. (This is the default, and need not be specified.)
- LOG_PERROR

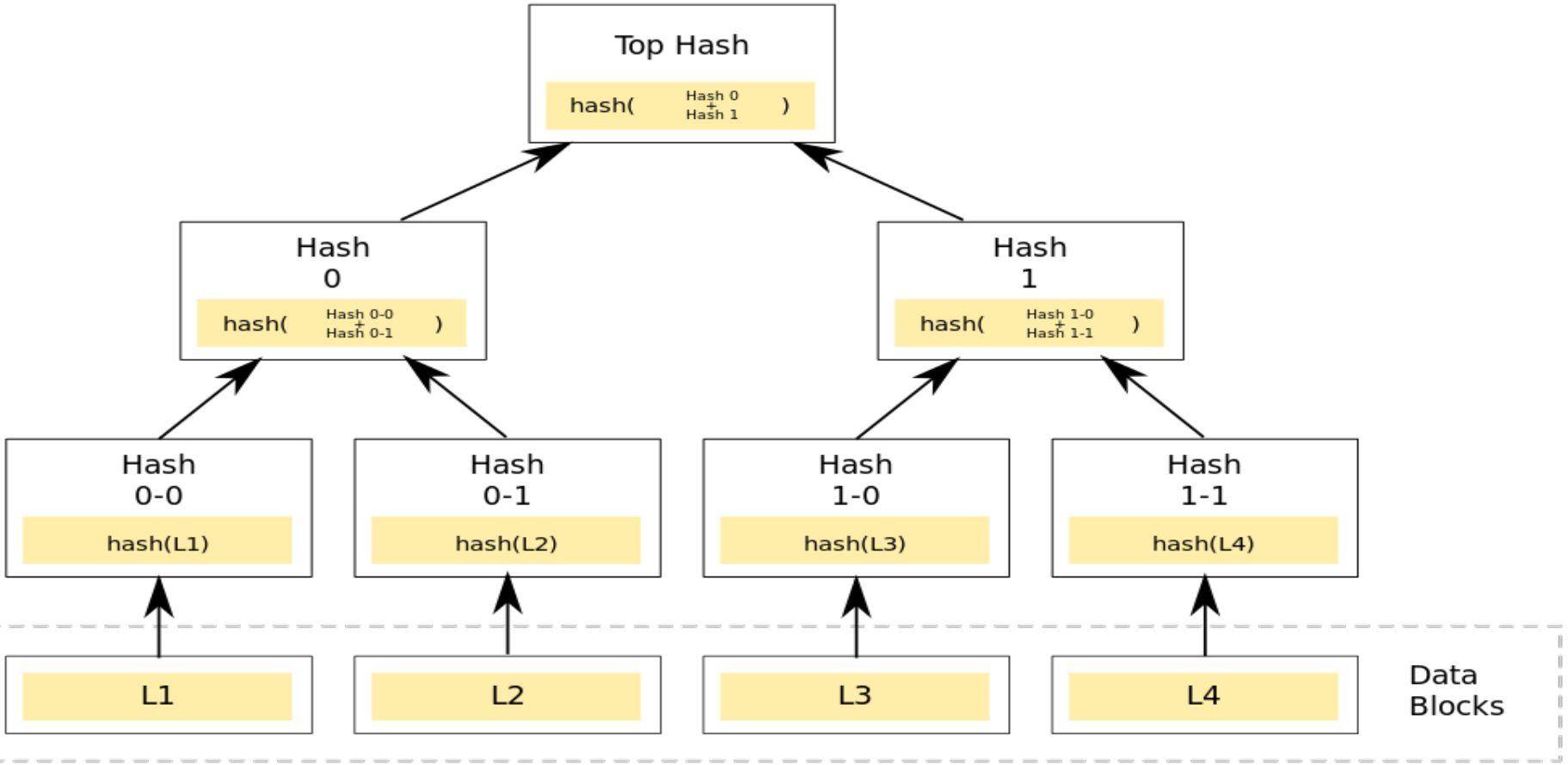
  (Not in POSIX.1-2001.) Print to stderr as well.
- LOG_PID

  Include PID with each message.

Blockchain: A probabilistic approach

- It forces information traveling over a network of computers to become more transparent and verifiable using mathematical problems that require significant computational power to solve.
- A potential attackers can corrupt a shared database with false information only if it owns a majority of the computational power of the entire network.
- Blockchain protocols ensure that transactions on a blockchain are valid and never recorded more than once, enabling people to coordinate transactions in a decentralized way without the need to rely on a trusted authority to verify and clear all transactions.

Concetto base: Merkle tree



Blockchain: A probabilistic approach -2

- A blockchain is simply a chronological database of transactions recorded by a network of computers
- Each blockchain is encrypted and organized into smaller datasets referred to as “blocks.”
- Every block contains information about some transactions, a reference to the preceding block in the blockchain, as well as an answer to a complex mathematical puzzle, to validate the data associated with that block.
- Every computer in the network stores a copy of the blockchain and it periodically synchronizes with the other ones to make sure that all of them have the same shared database.

Blockchain: A probabilistic approach -3

- To ensure a blockchain only records legitimate transactions the network confirms that new transactions are valid and do not invalidate former transactions.
- A block is appended to the blockchain only if the network computers reach consensus on validity of its transaction
- Consensus is achieved through different voting mechanisms, the most common is Proof of Work, which depends on the amount of processing power donated to the network
- A block added to the blockchain, can no longer be deleted and its transactions can be verified by everyone on the network. It becomes a permanent record that the computers on the network use to coordinate an action or verify an event.

Blockchain: A probabilistic approach -4

- The Proof of Work consensus mechanism requires that certain computers on the network (the “miners”) solve computationally-intensive mathematical puzzles, while others verify that the solution to that puzzle does not correspond to a previous transaction.
- To incentivize miners to invest computational power, the first miner to solve the mathematical problem is rewarded either through the issuance of currency or through transaction fees

Blockchain: A probabilistic approach - 5

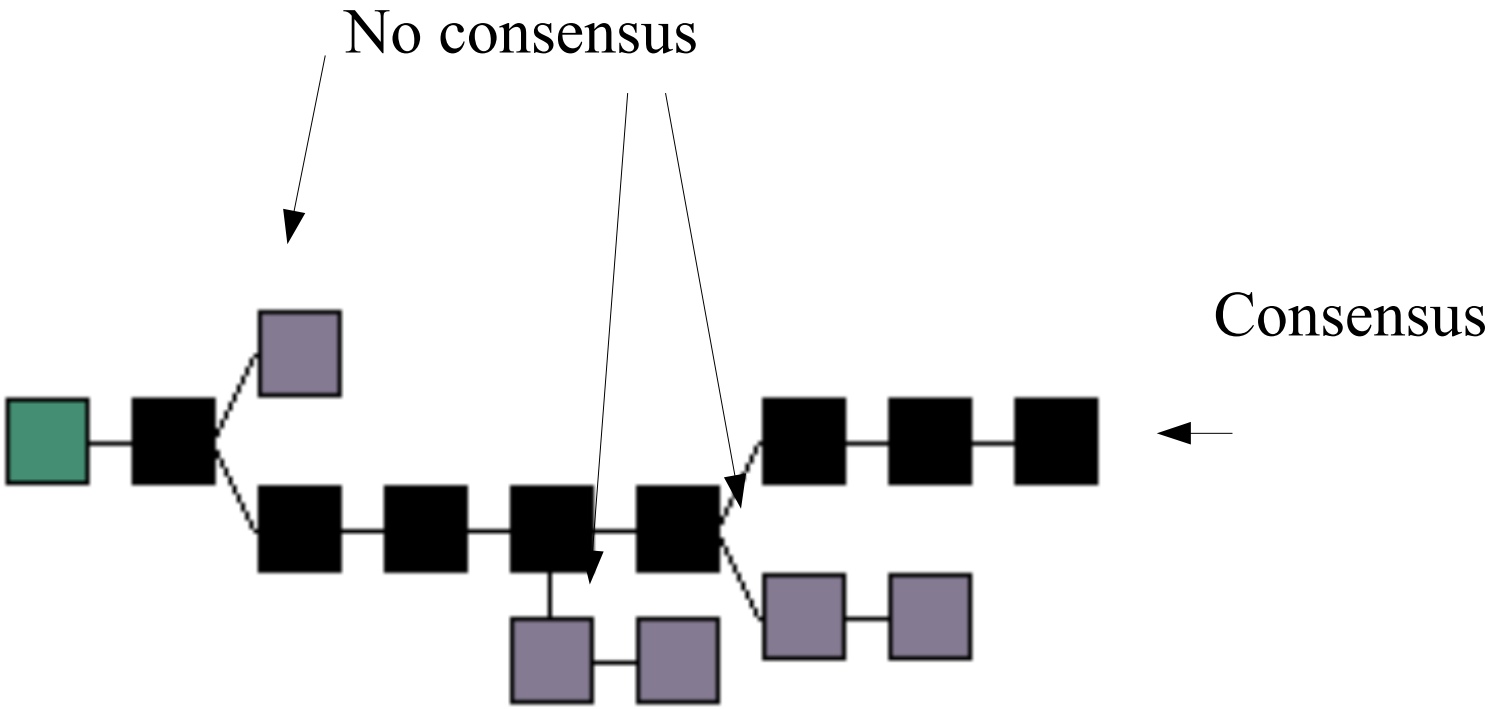
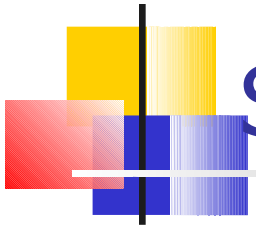
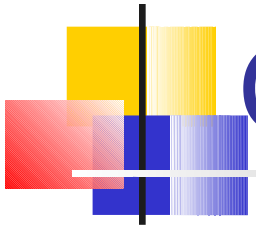


Fig. 1. A graphical representation of the blockchain



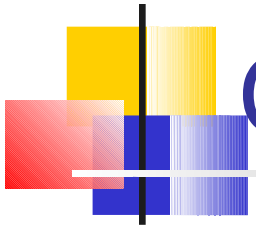
Sicurezza vs Sicurezza informatica

- I principi illustrati fino ad ora sono validi in generale e non definiscono le peculiarità di un sistema informativo
- Oltre alla possibilità di automatizzare un attacco un altro concetto fondamentale per la sicurezza di un sistema informatico è quello della gerarchia di macchine virtuali



Gerarchia di macchine virtuali

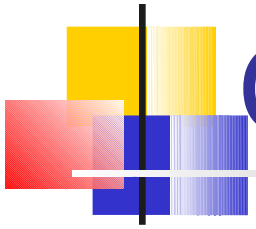
- Un qualunque sistema informativo e' una gerarchia di macchine virtuali
- Ogni macchina
  - e' caratterizzata da un linguaggio di programmazione = meccanismi che implementa a partire da quello definito dalla macchina sottostante
  - astrae ed incapsula la macchina sottostante
  - può essere standard con tutto ciò che questo implica dal punto di vista delle vulnerabilità



Cosa rende difficile la sicurezza?

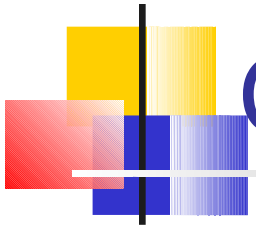
- La presenza di vulnerabilità dipende da come un linguaggio è implementato e non da come è definito
- Le vulnerabilità non si possono astrarre perchè una vulnerabilità in una macchina virtuale M permette di attaccare tutte le macchine virtuali implementate mediante (al di sopra di) M

  $\Leftrightarrow$ una vulnerabilità nella sicurezza a livello fisico puó essere devastante per tutte le macchine



Competizione verso il basso

- La tendenza degli attacchi è quella di abbassare il livello della macchina attaccata
- Controllare un livello basso della macchina equivale a controllare tutti quelli superiori
- Ultima tendenza degli attacchi: inserisco una macchina virtuale che emula l'architettura fisica
  - Difficile da scoprire
  - Devastante dal punto di vista della sicurezza

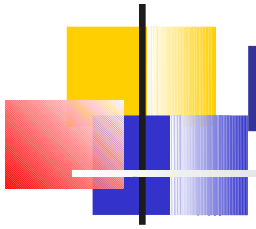


Gerarchia di MV e robustezza - I

- Robustezza a livelli
  - in ogni MV bisogna inserire meccanismi appropriati per il livello corrispondente
  - la distribuzione dei controlli a più livelli è il modo più efficace per ridurre il costo dei controlli

  Violare questo principio può voler dire che
  - I controlli per una certa macchina non esistono

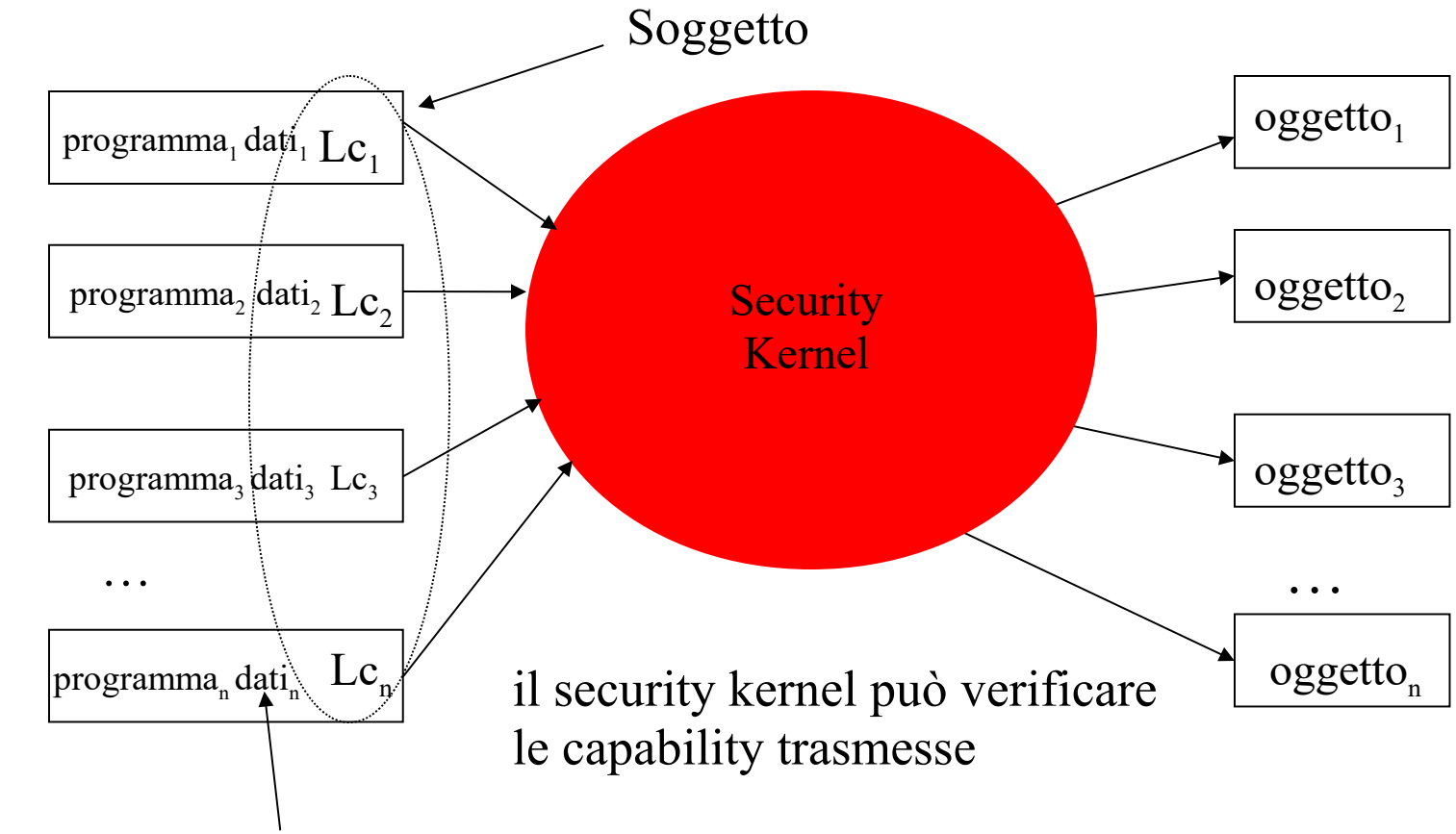
  oppure
  - I controlli per una macchina sono implementati da un'altra macchina con le complicazioni dovute alla trasformazione delle informazioni necessarie per il controllo
- Alcuni controlli possono essere ridondanti e ripetuti a più livelli

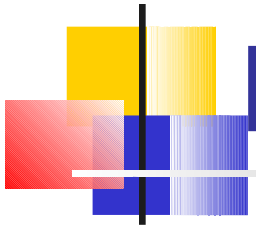


Esempio - Lista di capability

- Nella macchina di livello L abbiamo solo soggetti ed oggetti con i loro diritti
- Nella macchina L-1
  - tra i soggetti abbiamo il codice dei soggetti di livello L ed il security kernel
  - tra gli oggetti abbiamo le liste di capability
- La matrice della macchina L-1 deve garantire che i soggetti a livello L non manipolino illegalmente la lista di capability

Implementazione di capability





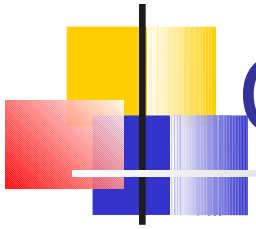
Esempio - Introspezione

- Preso un sistema operativo che offre un insieme ridotto di meccanismi di protezione posso scoprire attacchi esaminando le strutture dati del sistema stesso
- Introduco un componente che
  - analizza la memoria fisica usata da SO
  - ricostruisce lo stato delle strutture dati
  - verifica la consistenza delle strutture dati
- Complessità elevata ma unico modo di aggiungere la sicurezza a posteriori

Gerarchia di MV e robustezza - II

- Separazione di politiche e meccanismi in un approccio gerarchico:
  - ogni MV offre un insieme di meccanismi che possono essere composti dalla MV successiva in base alle proprie esigenze

    $\Leftrightarrow$
  - ogni MV può fare assunzioni su sicurezza che devono essere garantite da una o più delle MV sottostanti (come sono garantiti non importa)
  - Esempio: capability non manipolabili per
    - Cifratura
    - Protezione delle aree
    - ….
- Distribuzione dei componenti di cui è necessario fidarsi = distribuzione del TCB tra macchine virtuali

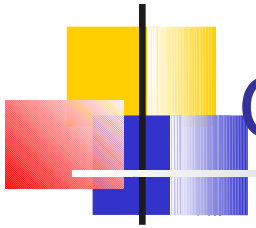


Gerarchia di MV e robustezza - III

- La robustezza di una MV dipende da quella delle MV sottostanti
- Macchine *funzionalmente* equivalenti hanno robustezza diversa in base a
  - Loro implementazione
  - Implementazione delle MV sottostanti

Altro caso in cui la robustezza viola l'astrazione $\Rightarrow$

non posso valutare la robustezza se non conosco l'implementazione



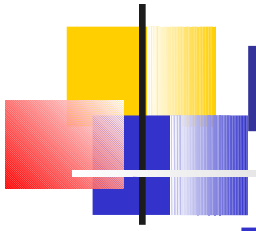
Gerarchia e robustezza - formalmente

- A è un soggetto e B un oggetto a livello L
- Al livello L-1 essi sono implementati mediante

$$(A_1, \ldots, A_k) \text{ e } (B_1, \ldots, B_m)$$

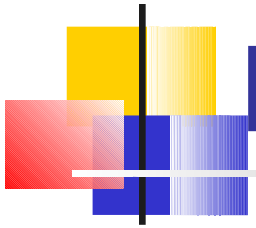
$$\Rightarrow$$

- Le interazioni tra un A_i ed un B_j devono rispettare i diritti di A su B stabiliti dalla matrice di protezione del livello L
- La matrice di protezione del livello L-1 deve essere coerente con quella di L
- Necessario garantire
  - l'integrità dei dati usati per la sicurezza
  - Non deve essere possibile violare la politica adottata al livello L lavorando al livello L-1



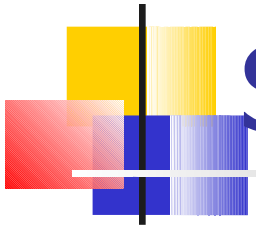
Esempio

- Se un frammento scritto in linguaggio macchina può accedere a posizioni di memoria di oggetti su cui il programma ad alto livello non ha diritti è molto semplice violare la matrice di protezione
- A livello del linguaggio macchina deve esistere una matrice di protezione almeno sulle aree di memoria per garantire la coerenza complessiva tra le varie politiche
- Questa matrice può essere implementata dalla MMU



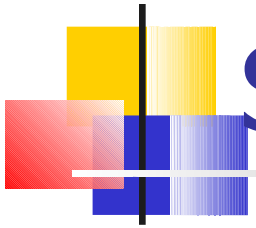
Non interferenza

- Stiamo cercando di garantire una forma più generale di non interferenza
- Due entità che non devono interferire sono due soggetti di una stessa macchina virtuale
- Gerarchia di macchine non deve facilitare l'interferenza = interazione diversa da quella permessa dal linguaggio



Situazione tipica

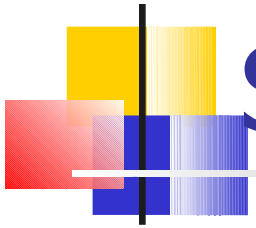
- Una zona di memoria, ad un qualche livello nella gerarchia di memoria è condivisa tra applicazioni diverse
- Le applicazioni che la condividono sono ignote a priori, dipendono da quali utenti stanno usando il sistema ad un certo istante
- Una applicazione che ottenga una zona di memoria e legga le informazioni in essa contenute può accedere a informazioni su cui non ha diritti



Soluzione

- Ogni zona di memoria
  - rilasciata da una applicazione
  - selezionata da garbage collected

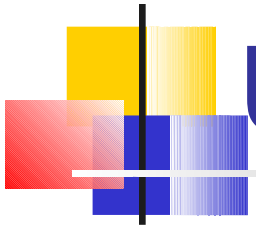
  deve essere inizializzata ad un valore predefinito in modo da cancellare tutte le informazioni in essa presenti
- Da garantire per qualsiasi zona di memoria
  - cache di ogni livello,
  - memoria principale,
  - memoria secondaria



Soluzione

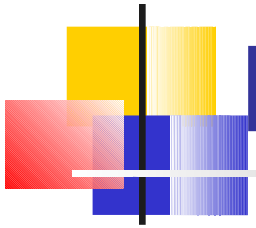
- In sistemi con requisiti di sicurezza elevati, tutte le risorse sono partizionata in livelli
- Le risorse di un livello possono essere usate solo dalla classe di applicazioni con certi requisiti di sicurezza
- Si limita fortemente la condivisione di risorse in modo da evitare

  interferenza = flusso *non controllato* di informazioni tra applicazioni con requisiti di sicurezza *diversi*



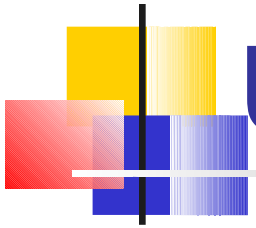
Una regola generale …

- Come dimostrato dall'approccio dei sistemi con requisiti di sicurezza elevati, la condivisione crea comunque dei problemi e disciplinare la condivisione è fondamentale per poter garantire la sicurezza
- Quindi oltre ad avere un sistema semplice dobbiamo anche avere un sistema che, se possibile, non condivide risorse
- Questo può provocare un aumento sensibile dei costi



Esempio

- Memoria partizionata in zone base alla criticità delle informazioni gestite
- Segregazione del traffico nella rete = evitare una rete piatta dove tutti i nodi possono vedere una qualunque comunicazione si possono usare
  - linee fisiche diverse e di switch invece di hub = dominio di collisione
  - linee logiche create mediante crittografia
  - Spesso si usano frequenze ma non danno sicurezza
- Criticità = un qualunque sistema comprende sempre almeno due livelli di criticità dei dati



Una distinzione importante

- In un qualuque sistema abbiamo due tipi di informazioni
  - Utente (nome utente)
  - Politica di sicurezza utilizzata per proteggere le informazioni utente (password)
- Le due informazioni appartengono a domini diversi e devono essere protette in modo diverso