principi di progetto di politiche e meccanismi di sicurezza

minimalità dei diritti

- ad un soggetto devono essere concessi solo i **diritti necessari** a eseguire i suoi **compiti**
  - diritti non necessari non devono essere concessi
  - i diritti sono dati solo per il tempo necessario

- Best practice: i diritti dovrebbero essere concessi in base al "**ruolo**" e non all'identità
  - approccio noto come **Role Based Access Control** - RBAC

default sicuri

- le configurazioni di default devono essere sicure
  - **il default deve essere di negare i diritti/accessi**
    - politica chiusa
  - un diritto negato di default lascia sicuramente il sistema in uno stato sicuro
    - tende a scontentare l'utente, ma l'amministratore sarà avvertito dallo stesso utente, se questi è stato privato di un diritto necessario
  - un diritto concesso di default può rendere il sistema insicuro in certi contesti
    - l'amministratore può non considerare di negarlo esplicitamente
    - un utente contento per avere un diritto in più non avvertirà l'amministratore per farselo togliere

semplicità (di progetto)

- meccanismi e politiche di sicurezza devono essere più semplici possibili
  - **interfacce ed interazioni semplici** tra pochi elementi
  - se il sistema ha **pochi elementi** vi sono **pochi punti in cui si possono commettere errori**
    - meno bugs
    - meno errori di configurazione
  - un sistema semplice è **più facile da capire e aggiustare** in caso di problemi

“Ciò che non c’è non si può rompere” – H. Ford

progetto aperto

- la sicurezza non deve dipendere dalla segretezza del progetto, dell'implementazione, o di politiche di sicurezza: **no "security through obscurity"**
  - la sicurezza non risiede nella segretezza dei meccanismi ma nella segretezza di password e chiavi crittografiche (detti per l'appunto "segreti")
  - non significa che il codice deve essere pubblico
    - il codice PUO' essere pubblico, se si prevede e accetta la possibile presenza di bug, a lungo andare questo aumenta al sicurezza (grazie al code review della community)
- **"security through obscurity" ammissibile solo come rimedio temporaneo**
  - nei casi in cui non si è sicuri della correttezza del progetto, del codice o della politica in questione
  - in tali casi è meglio tenerli nascosti, ma non è una best practice

progetto aperto

- principio **applicato spesso ai prodotti con larga diffusione**
  - un prodotto con larga diffusione giova del controllo dei suoi clienti/utenti
  - i clienti ritengono che un progetto aperto sia più affidabile quindi più appetibile commercialmente
- **prodotti e politiche ad-hoc spesso vengono mantenuti segreti**
  - la politica o i meccanismi di una organizzazione sono difficilmente esenti da bugs o falle di sicurezza
  - la pubblicazione faciliterebbe gli attacchi
  - i vantaggi da verifiche da parte della comunità sono minimi: comunità inesistente o troppo piccola

isolamento o confinamento

- minimizzare la condivisione di risorse tra soggetti (servizi, utenti, processi)
  - in particolare se i soggetti hanno “criticità” differenti, cioè appartengono ad una diversa **classe di sicurezza**
- la condivisione di risorse …
  - …permette attacchi da un servizio/utente ad un altro
  - …rende entrambi i servizi/utenti vulnerabili ad un fallimento della risorsa

mediazione completa

- effettuare il controllo ad ogni accesso

- spesso nei sistemi il controllo è effettuato solo una volta all'inizio della "transazione"
  - cambiamenti di permessi non hanno effetto su transazioni già iniziate (es. file già aperti)
  - es. nei sistemi UNIX il controllo di accesso ai file è effettuato solo dalla system call open e non dalle operazioni successive

defence in depth

- richiedere il consenso di più entità di controllo per ottenere l'accesso
  - più meccanismi di difesa assieme sono più difficili da forzare

- esempi
  - multi factor authentication
  - più firewall in serie
  - in crittografia distribuzione di parti di uno stesso segreto su più host

usabilità (accettabilità psicologica)

- meccanismi e politiche di sicurezza non devono aggiungere difficoltà all'accesso alle risorse da parte degli utenti
  - altrimenti gli utenti si rifiuteranno di usare il sistema…
  - …o cercheranno di aggirare le limitazioni per poter svolgere agevolmente il loro lavoro
  - …o abbandoneranno
- è un obiettivo difficile

eterogeneità

- usare sistemi eterogenei
  - servizi su sistemi diversi (es. windows e linux) hanno meno probabilità di essere attaccati dallo stesso hacker perché gli exploit sono diversi
  - vedi “defence in depth” e isolamento
- raggiungere alta eterogeneità è problematico
  - trovare sistemi diversi è difficile
    - quanti sistemi operativi conosci? quanti web server?
  - difficoltà di gestione
    - gli amministratori rifiutano di gestire sistemi molto eterogenei (vedi accettabilità psicologica)

compromesso

- questi principi sono spesso in conflitto tra loro
  - tipicamente usabilità e della semplicità sono in conflitto con tutti gli altri
- un buon progetto prevede il giusto compromesso tra
  - questi principi
  - vincoli di budget
  - altri vincoli (es. tecnologici)

principi: sinergia e antagonismo