

principi di progetto di politiche e meccanismi di sicurezza

minimalità dei diritti

- ad un soggetto devono essere concessi solo i **diritti necessari** per eseguire i suoi **compiti**
 - diritti non necessari non devono essere concessi
 - meglio fare whitelist anziché blacklist, meno possibilità di errori e dimenticanze
 - i diritti sono dati solo per il tempo necessario
- capire quale sono i diritti minimi per ciascun soggetto può essere difficile o costoso
 - best practice: spesso conviene decidere i diritti in base al “**ruolo**” del soggetto e non direttamente al soggetto
 - approccio noto come **Role Based Access Control** – RBAC
- dati i diritti *minimi ideali* non è detto che la tecnologia permetta di realizzare una tale configurazione

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

default sicuri

- le configurazioni di default devono essere sicure
 - **il default deve essere di negare i diritti/accessi**
 - politica massimamente restrittiva (obbliga alla riconfigurazione)
 - minimi accessi per caso d'uso tipico
 - 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

default password

- un caso notevole sono le **password di default**
 - c'è necessità di avere un primo default
 - critico per gli apparati di rete e IoT
 - accessibili solo via rete e quindi soggetti ad attacchi
 - caso famoso è la botnet Mirai
- password uguale per tutte le istanze
 - nota a tutti, anche agli hacker (repository on-line)
 - si devono prevedere contromisure
 - es amministrazione possibile solo da interfacce “fidate”, possibile per firewall ma non per altri dispositivi IoT
- password diverse per ciascun istanza
 - scomode da inserire (es. password dei router forniti dagli operatori)

isolamento

(o confinamento)

- molte configurazioni di sicurezza isolano certi soggetti e oggetti da altri
 - secondo una suddivisione in **zone di sicurezza**
 - spesso in base alla criticità (es. amministrazione, ingegneria, studenti, ecc.)
- l'effetto è di limitare la condivisione delle risorse o la comunicazione
 - es. servizi, utenti, processi
 - limita possibilità di attacchi da un servizio/utente ad un altro

isolamento: vantaggi

eventuali incidenti in una zona difficilmente si propagano nelle altre zone

- es. contagio di malware
- semplifica la pianificazione e la gestione della sicurezza
 - es. permette di separare responsabilità di gestione e pianificazione
- spesso implica una limitazione di risorse condivise e quindi di single point of failure
 - attenzione, oggi la condivisione di certe risorse è spesso mascherata dalla virtualizzazione per cui questo può non essere vero

isolamento: punti di vista

- isolare per proteggere
 - es. ciascun soggetto la cui funzionalità è critica può essere isolato per proteggerlo dagli altri
- isolare per contenere
 - es. un soggetto può essere isolato perché vulnerabile e quindi l'isolamento protegge gli altri
- isolamento asimmetrico
 - es. l'amministratore vs. utenze normali

defence in depth

- dato un budget, ripartirlo tra più contromisure spesso protegge meglio che usarlo per una sola contromisura
 - le contromisure devono essere tutte contemporaneamente attive nella difesa
 - ma avere modi di operare diversi (eterogeneità di fini non di vendor)
 - spesso è più difficile forzare varie contromisure invece di una sola, anche se costosa
- a.k.a. layered security

defence in depth

- es. richiedere il consenso di più entità di controllo per ottenere l'accesso
 - multi factor authentication
 - più firewall in serie
 - in crittografia distribuzione di parti di uno stesso segreto su più host
 - multisignature in ambito firma elettronica
 - combinazioni antivirus/firewall/IDS

semplicità (di progetto)

- meccanismi e politiche di sicurezza devono essere più semplici possibili
 - diminuire il numero di elementi del sistema
 - diminuire le interazioni tra gli elementi del sistema
 - semplificare le interfacce, i protocolli e l'interazione tra gli elementi
 - diminuire la quantità di linee di codice
 - rendere semplice la configurazione e la gestione.
- vantaggi
 - 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

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 la 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

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
- un progetto aperto e semplice aiuta a raccogliere feedback e ciò normalmente migliora l'usabilità
- è un obiettivo difficile

eterogeneità di vendor

- 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

