

Disponibilità


A primo impatto la disponibilità può non sembrare un requisito di sicurezza ma un aspetto funzionale. Diventa importante quando lo scopo dell'attaccante è proprio negare la disponibilità del servizio.

DoS (Denial of Service): negazione della proprietà di disponibilità.

Il Denial of Service può riguardare sia la saturazione banda che le risorse computazionali. In generale possiamo indicare come attacco DoS qualsiasi manomissione che porti alla negazione della proprietà qui citata.

Un esempio rudimentale di attacco DoS può essere un ciclo infinito che tenta di stabilire una comunicazione o invia richieste di un qualche tipo. Una contromisura per un attacco di questo tipo potrebbe basarsi sul richiedere al client un qualche tipo di calcolo e conseguente risposta, declinando le successive richieste qualora questo non risponda in maniera corretta.

Controllo di accesso (autorizzazione)

 **Definizione** di controlli di accesso: Ciascun utente che abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato.

Pensando al concetto di autorizzazione ci viene spontaneo pensare all'esistenza di una policy a definire le modalità secondo cui e le risorse a cui un certo utente possa accedere, non c'è le generiche operazioni che può compiere. La proprietà di autorizzazione quindi è molto legata alla policy (o meglio, al concetto di coerenza con la policy).

 **Domanda d'esame:** fare degli esempi di sistemi di autorizzazione.

L'autorizzazione ha come **pre-requisiti**:

- la policy
- l'autenticazione

Uno dei punti cruciali a questo punto è la "bontà" della policy. Scrivere una policy infatti è estremamente complesso ed il problema principale nella stesura della policy sta nel fronteggiare le possibili **inconsistenze**.

Analisi di una policy di sicurezza

Presentiamo un esempio di policy:

1. Un utente ha il permesso di leggere un qualunque file pubblico
2. Un utente ha il permesso di scrivere solo sui file pubblici di sua proprietà
3. Un utente non può fare il downgrade di un file
4. Un utente ha l'obbligo di cambiare la propria password quando scade

5. Un utente segreto ha il permesso di leggere su un qualunque file non pubblico
6. Un utente segreto ha il permesso di scrivere su un qualunque file non pubblico
7. Un amministratore ha il permesso di sostituire un qualunque file con una sua versione obsoleta
8. Un utente che non cambia la sua password scaduta (negligente) ha il divieto di compiere qualunque operazione
9. Un utente che non cambia la sua password scaduta (negligente) non ha discrezione di cambiarla

Analizzando questa policy possiamo immediatamente trovare alcune **ambiguità**.

Al punto **2** ad esempio la definizione di proprietà e file pubblico e la relazione tra queste due è abbastanza generica. Al punto **7**, lì dove l'amministratore fosse un utente del sistema (e non ci fossero regole ulteriori a definire il caso specifico), si andrebbe in conflitto con il punto **3**. Al punto **8** si afferma che un utente negligente abbia il divieto di effettuare qualunque azione nel caso non abbia rinnovato la propria password scaduta ma non viene specificato se il rinnovo stesso della password è un'operazione non permessa (si entrerebbe quindi in una situazione di blocco perenne). Rimane inoltre fumosa la definizione dei ruoli e la loro eventuale capacità di sovrapporsi.

Prima ancora di scrivere una policy bisognerebbe definire al meglio gli attori in gioco.

Voglio scrivere la policy in modo tale che l'amministratore sia un utente? Un utente segreto? Questo dipende dai requisiti **che dobbiamo raccogliere prima di scrivere la policy**. È importante avere un quadro chiaro di regole e precondizioni, avere un quadro chiaro dei ruoli e se questi si sovrappongono.

A seconda di come abbiamo definito queste relazioni una policy come quella sopra riportata può risultare inconsistente o meno. Se definiamo a priori e coerentemente queste relazioni la policy può essere coerente.

Elementi della policy:

- **Ruoli:** utente, utente segreto, sistemista, utente negligente
- **Utente:** qualunque entità che ricopra un certo ruolo
- **Operazioni:** leggere, scrivere, downgrade, cambio password
- **Modalità:** obbligo, permesso, divieto, discrezionalità

In fase di analisi potrebbero presentarsi ulteriori incongruenze, ad esempio da questa analisi si evince la non sovrapposibilità di amministratore ed utente negligente (ovvero, un amministratore non farà mai scadere una password). Questo scenario è reale? Dobbiamo modificare la policy di conseguenza al fine di coprire il caso in cui un amministratore diventi negligente?



Domanda d'esame: quali sono le inconsistenze della policy?

Esempio di risposta:

- **il dilemma**
- **la contraddizione**

$$Dilemma \equiv \text{Obbligatorio}(x) \wedge \text{Obbligatorio}(\neg x)$$

$$Contraddizione \equiv \text{Obbligatorio}(x) \wedge \neg \text{Obbligatorio}(x)$$

Disambiguare una policy

Logica come strumento di disambiguazione di policy

Oltre agli evidenti problemi già visti, la lingua parlata lascia ulteriore spazio alle ambiguità. Una soluzione potrebbe essere utilizzare la logica dei predicati, magari accoppiata a tool di validazione appositi, per disambiguare (o meglio ancora scrivere da principio in maniera non ambigua) le policy.

La prima regola della policy vista può ad esempio essere scritta come:

$$\forall u, \forall f, \text{public}(f) \rightarrow P(\text{read}(u, f))$$

È importante notare che singole regole chiare ed elementari non sono esenti da ambiguità.

La coesistenza di più regole, per quanto queste siano chiare singolarmente, può portare ad inconsistenze.

Definizione dei ruoli e la loro intersezione

Si potrebbe pensare, vista la policy presa ad esempio, che ruoli perfettamente disgiunti siano una buona strategia per evitare le ambiguità. Ad esempio, se un amministratore non è un utente non è soggetto ai vincoli tipici degli utenti e non può di conseguenza nemmeno essere negligente.

Ruoli così disgiunti non sono però soluzioni necessariamente ottimali e spesso nella pratica non sono applicabili. In questo caso definire i ruoli in maniera chiara è cruciale, un key point per ridurre l'ambiguità.

Prendiamo ad esempio la configurazione di un firewall, definire opportunamente i ruoli (e di conseguenza le capacità di accesso) è di fondamentale importanza per la sicurezza del sistema.

Metodi di verifica e risoluzione

Alcuni metodi efficaci per verificare la consistenza e gestire le sovrapposizioni nelle policy possono essere:

- Tracciare un grafo rappresentativo della policy, con azioni come archi e attori in gioco come nodi. L'assenza di cicli indica che non ci sono situazioni ambigue (ideale ma spesso non applicabile)
- **assegnare delle priorità** alle regole della policy. In questo modo se la regola X , di priorità maggiore, va in contrasto con la regola Y , di priorità minore, la prima avrà la meglio senza lasciare spazio alle ambiguità (es: le norme sociali stabiliscono che non si debba urlare in pubblico ma, in una situazione di emergenza è permesso farlo)

- utilizzo di **meta-regole** (o modelli di policy): l'utilizzo di pattern collaudati può essere un grosso aiuto per scrivere policy non ambigue. Una meta-regola può ad esempio essere: non possiamo lasciare discrezionalità, possiamo lavorare solo con gli obblighi.

Mandatory Access Control (MAC)

Le policy di questo tipo prevedono una struttura basata su obblighi (mandatorie appunto).

Questi obblighi **non sono modificabili**, e sono di provenienza ed ispirazione militare.

Role-Based Access Control (RBAC)

L'opposto di una policy mandatoria è una policy **non-mandatoria**, anche detta **Role-Based Access Control**, che prevede l'accesso per tutti e soli gli utenti autorizzati.

Elenco di lettura

- [Attacchi DoS e DDoS: modalità di difesa e contromisure](#)
- [Cos'è SELinux?](#)