Proprietà di sicurezza

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Proprietà di livello 1

Segretezza o confidenzialità

Prima proprietà di tre che formano la base della piramide rappresentante le proprietà di sicurezza è la confidenzialità. Formalmente la **segretezza** di un oggetto x implica che x non sia noto a chi non è autorizzato a conoscere x.
Parlare di segretezza presuppone che vi sia una **policy**, attraverso cui viene definito chi può essere a conoscenza di un dato segreto. Se chiunque altro (al dì fuori della policy) venisse a conoscenza di quest'ultimo segreto, allora vi sarebbe una violazione della sicurezza.

Autenticazione

Definiamo l'**autenticazione** come la conferma dell'identità di un entità (e.g. un intelocutore come un umano, un processo, un ip), quindi la conferma che egli sia esattamente chi afferma di essere.

Integrità

L'ultimo elemento della tripla che forma la base della piramide è l'**integrità**. Possiamo definire l'integrità (e.g. del dato o del sistema) come la conferma che l'informazione non sia modificata da entità non autorizzate.
Esempi di **misura** per il controllo dell'integrità sono:

- Il **checksum**, una sequenza di bit che, associata al pacchetto trasmesso, viene utilizzata per verificare l'integrità di un dato. L'unico modo per verificare l'integrità di un file scaricato è il confronto con il checksum mostrato dal sito certificato. Tuttavia sarebbe possibile *inviare* (e.g. ad un collega) un file malevolo insieme ad un nuovo checksum opportunamente calcolato relativo a quest'ultimo, il che comunque differirebbe dal checksum mostrato dal sito certificato.
- La **firma elettronica**, un insieme di dati utilizzati come metodo di identificazione informatica. La firma digitale è la tecnologia con cui possono essere effettivamente soddisfatti tutti i requisiti richiesti per dare validità legale ad un documento elettronico firmato digitalmente; garantisce i servizi di integrità, autenticazione e non-ripudio. In particolare le proprietà che deve avere una firma digitale per essere ritenuta valida sono:
    - *Autenticità della firma*: la firma deve assicurare il destinatario che il mittente ha deliberatamente sottoscritto il contenuto del documento.
    - *Non falsificabilità*: la firma è la prova che solo il firmatario e nessun altro ha apposto la firma sul documento.
    - *Non riusabilità*: la firma fa parte integrante del documento e non deve essere utilizzabile su un altro documento.
    - *Non alterabilità*: una volta firmato, il documento non deve poter essere alterato.
    - *Non contestabilità*: il firmatario non può rinnegare la paternità dei documenti firmati; la firma attesta la volontà del firmatario di sottoscrivere quanto contenuto nel documento

> E' importante notare che la firma elettronica (o digitale) differisce da un checksum poichè la prima, differentemente dalla seconda, ha un robustezza di natura crittografica. La firma digitale, quindi, conferisce una robustezza crittografica al processo di confronto, il che implica che non è possibile l'esempio relativo al file malevolo con il checksum.

Privatezza (Privacy)

Passiamo adesso alla **privacy**, prima proprietà al dì sopra della base, quindi appartente al livello due. Definiamo la privacy come il *diritto* di un entità di rilasciare o meno i propri dati personali ad altre entità. Una **misura** per garantire la privacy è dato dalle policy.

> Si noti che vi notevole differenza tra la privacy e la confidenzialità (i.e. segretezza), la quale è una differenza logica ed architetturale. Quest'ultima (i.e. privacy), differentemente dalla confidenzialità, implica un diritto. Potremmo al più definire la privacy come il diritto istanziato alla confidenzialità.

Anonimato

L'**anonimato** è il *diritto* dell'iniziatore di una transazione di rilasciare o meno la propria identità ad alte entità. Una **misura** per garantire l'anonimato è lo **pseudo-anonimato**, per cui viene effettuata la scissione delle proprie *PII* (acronimo di *personal identifier information*, ossia una stringa (o tupla informativa) che identifica univocamente la persona fisica) effettuando la generalizzazione di questìultime.
Poiché l'anonimato può essere studiato a più livelli architetturali, esistono misure relative a specifici livelli:

- A livello applicativo, attraverso gli *anonymous-web-proxy* che svolgono il ruolo di *man-in-the-middle*;
- A livello di rete, con *TOR*.
  Si noti che la navigazione anonima all'interno di un browser non rende anonimi sulla rete ma semplicemente rende anonimi nei confronti della macchina su cui si opera poichè non memorizza alcun dato relativo alla sessione.

Proprietà di livello 2

Non-ripudio

La proprietà di non-ripudio indica che l'entità non possa negare la propria partecipazione ad una transazione con uno specifico ruolo. Una **misura** è fornita dalla **PEC**, un sistema che gode della proprietà di non ripudiabilità, il che rende impossibile negare l'invio di una PEC provienente dalla nostra casella.
E' importante non confondere la proprietà di non-ripudio con la proprietà di autenticazione, le quali presentano delle differenze. Per esempio, se Mario Rossi si identifica come Mario Rossi allora esiste un evidenza (i.e. conferma) della mia identità poichè altrimenti l'autenticazione non sarebbe possibile. D'altra parte, quest'ultima autenticazione (differentemente dal non-ripudio) non è *proponibile*, *verificabile* e *dimostrabile* all'esterno, come in egual modo essere il testimone di un crimine non implica incastrare il colpevole; il tutto poichè un avvocato potraebbe smontare la testimonianza di quest'ultimo (a meno della presenza di un elemento di non ripudiabilità).

Relazioni logiche fra autenticazione, anonimato e non-ripudio

1. L'autenticazione è l'opposto dell'anonimato;
2. L'anonimato implica che non vi sia autenticazione;
    1. L'autenticazione coincide con l'opposto dell'anonimato;
3. Autenticazione non implica il non-ripudio;
4. L'esistenza del non-ripudio implica l'esistenza dell'autenticazione;
5. L'anonimato implica l'inesistenza del non-ripudio;
6. Il non-ripudio implica l'inesistenza dell'anonimato.

Il problema dell'esame: il protocollo WATA

Il protocollo WATA si pone l'obiettivo di definire una modalità *sicura* per svolgere un esame. WATA consiste in un **esaminatore**, un **invigilator** ed un **candidato**. L'invigilator e l'esaminatore possono essere la stessa persona; infatti, dal punto di vista della sicurezza non cambia nulla (d'altra parte, l'esaminatore potrebbe comprarsi l'invigilator).
All'interno di WATA possiamo distinguere 4 diverse **fasi**:

1. **Preparation**.
   Esiste un database di domande dal quale viene estratto randomicamente una tupla di domande; esiste un alfabeto di lunghezza n dal quale viene estratto a random un ID, il quale sarà l'ID del test (trasformato in un bar-code). Esiste, inoltre, un database di *mark* (i.e. voti) inizializzato con gli ID dei test (lo scopo finale è inserire un voto affianco all'ID del test). Stampiamo, quindi, l'ID del test ed il modulo di autenticazione, quindi il test formato dal suo ID, le domande ed il form per le risposte. L'esaminatore firma preventivamente i test nel suo ufficio (firma grafometrica) e da tutto il materiale all' invigilator.

2. **Testing**.
   La fase di testing è la fase in cui lo studente svolge il test.
   Il test passa alle mani del candidato randomicamente. Il candidato fa l'operazione di riempire il modulo di autenticazione col test firmato costruendo un test autenticatore riempito. Per farsi autenticare fornisce all'invigilator il proprio documento di identità e quello che ha appena prodotto, nascondendo gli elementi identificativi. Così l'invigilator controlla la validità del documento, ovvero che l'anagrafica stia nell'elenco dei registrati e che l'anagrafica sul documento corrisponda con quella letta sul test preparato prima. Dopo che il test è stato corretto il candidato riempie il modulo di quel test creando un test riempito. A questo punto taglia il test riempito producendo un token e il a_test (*anonymaze test*). Il candidato da randomicamente il a_test all'invigilator.

3. **Marking**
   L'invigilator da il a_test all'esaminatore e quest'ultimo corregge tutti i test ed inserisce i voti nel database dei voti tramite il bar-code. In questo modo l'esamitore non saprà mai a chi appartiene il test.

4. **Notification**
   Lo studente da il token all'esaminatore per poter avere il proprio voto attraverso la scansione che andava a leggere l'ID del testo nel database delle domande evidenziando il voto. L'esaminatore, dopo aver visto il voto, lo registra nel database dello storico della materia con quel ID.

Disponibilità

La proprietà di **disponibilità** si basa sul concetto che il sistema sia *operante* e *funzionante* in ogni momento. Chiaramente un sistema può essere un sito, un servizio, etc. Qualcosa che può compremettere la disponibilità è il concetto di **DoS** (Denial of Service), un attacco subdolo e malevolo rispetto la negazione del servizio, fondamentale poichè nessun sistema che non funzioni è più interessante. Quest'ultimo si basa sul sovraccarico del load balancer presente all'interno di un server. Un attacco DoS distribuito su più è detto **DDoS**, ovvero *Distribuited Denial of Service*.
Una misura rispetto un attacco di questo tipo si basa sul filtrare le richieste, chiaramente con un impatto sulle prestazioni.

Proprietà di livello 3

Controllo di accesso

Arriviamo adesso al livello tre della piramide mostrata da Schneier all'interno del suo libro ed iniziamo a trattare il *controllo*, definito più nello specifico dal professore come: **controllo di accesso**. Il controllo di accesso, che è sinonimo di *autorizzazione*, viene definita nel modo seguente: *ciascun utente abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato*.
Una buona autorizzazione si basa su dei profili autorizzativi ben definiti delle classi di sistemi e servizi che vengono associate ad un utenza (e.g. loggarsi come `jax` prevede certi *profili autorizzativi*).

Alcune misure per il controllo di accesso sono (i) l'autenticazione dell'utente, attraverso cui ci assicuriamo chi e quali operazioni possa svolgere un generico utente (e.g. il professore quando entra in aula viene autenticato, motivo per cui sappiamo che egli abbia un certo set di autorizzazioni, che includono sedersi alla cattedra e spiegare la lezione), (ii) le politiche di sicurezza (in inglese, "*policy*") e (iii) l'implementazione delle policy stesse, per es. le *ACL* in Linux. Relativamente alle politiche di sicurezza, si noti che un cyber-informatico non può lavorare senza una buona compresione di una determinata policy. Un attacco, infatti, è tale se un attività che sovverte un policy. Si pensi, per es. che fino a poco tempo fa era possibile (all'interno di Ubuntu) per $utente1$ eseguire il comando `ls` su `home/utente2`, il che è permesso dalle policy di Ubuntu stesso, motivo per cui non possiamo definire quest'ultimo un attacco.

Un esempio di politica di sicurezza

Una policy giocattolo è la seguente:

1. Un utente ha il permesso di leggere un qualsiasi file pubblico;
2. Un utente ha il permesso di scrivere solo sui file pubblici di sua proprietà;
3. Un utente ha il divieto di sostituire un file con una sua versione più obsoleta;
4. Un utente ha l'obbligo di cambiare la propria password quando questa 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 più obsoleta;
8. Un utente che non cambia la sua password scaduta (i.e. *negligente*) ha il divieto di compiere qualunque operazione;
9. Un utente che non cambia la sua password scaduta (i.e. *negligente*) non ha discrezione di cambiarla;

Notiamo a partire da questa policy giocattolo i seguenti **elementi** (*fondamentali*):

- **Ruoli**.
  I ruoli presenti all'interno di questa policy giocattolo sono i seguenti: utenti, utenti segreti, amministratori e negligenti.
  E' importante notare che i ruoli, in maniera *connaturata*, presentano delle intersezioni. All'interno della policy giocattolo notiamo le seguenti intersezioni: $Utenti \subseteq Utenti\ segreti \subseteq Amministratori$.
  Inoltre, sempre secondo questa policy, esistono degli utenti *negligenti* uguale ad un sottoinsieme dell'insieme degli utenti, quindi vale $Utente \subset Negligenti$.
  [*NB*] Si noti che una policy che non prevede intersezioni è formalmente possibile ma non è implementabile.
- **Utente**.
  Definiamo un utente come una qualunque entità ricopra un certo ruolo;
- **Operazioni**.
  Questa policy regolamenta le seguenti operazioni per i precedenti ruoli: *lettura*, *scrittura*, *downgrade* e *cambio password*;
- **Modalità**, obbligo, permesso, divieto e discrezionalità. In particolare, con disrezionalità intendiamo

A partire dalla definizione delle modalità sopra elencate, potrebbero nascere delle ambiguità sintattiche. Spieghiamo quest'ultime specificando cosa deve essere inteso secondo una determinata modalità svolta su una generica azione a.

Fissata una *modalità base*, fissiamo le **relazioni tra le modalità** *derivate* e la modalità base. In questo caso, fissiamo come modalità base il concetto di **obbligo** rispetto una generica azione a.

- *Modalità base*: $Obbligatorio(a)$.
- $Vietato(a) = Obbligatorio(\neg a)$.
  Il divieto di svolgere l'azione a è uguale ad affermare che è obbligatorio svolgere l'opposto dell'azione a, ovvero $not(a)$;
- $Permesso(a) = \neg Obbligatorio(\neg a)$.
  Il permesso di svolgere un azione a è uguale alla non esistenza di un obbligo rispetto l'opposto dell'azione a.
- $Discrezionale(a) = \neg Obbligatorio(a)$.
  Affermare che l'azione a è discrezionale indica che non vi è obbligo di svolgere l'azione a

Talvolta, all'interno di una politica, possono esistere delle **inconsistenze**. Distinguiamo due tipi di inconsistenza:

- **Contraddizione**, il che nasce se esiste l'obbligo di svolgere un azione a ed allo stesso tempo non esiste l'obbligo di svolgere quest'ultima attività a. Una contraddizione può essere formalmente definita attraverso la seguente notazione: $Obbligatorio(x) \wedge \neg\, Obbligatorio(x)$. Un esempio di contraddizione all'interno della policy giocattolo esiste secondo le regole 3 e 7.
- **Dilemmi**, se esiste l'obbligo di svolgere un azione a ed allo stesso tempo esiste l'obbligo di svolgere l'opposto di quest'ultima attività a, ossia $\neg\, a$. Una contraddizione può essere formalmente definita attraverso la seguente notazione: $Obbligatorio(x) \wedge Obbligatorio(\neg\, x)$. Un esempio di dilemma all'interno della policy giocattolo esiste secondo le regole 8 e 9.

E' importante notare che le singole regole all'interno della policy sono perfette. Il problema nasce a partire dalla convivenza di quest'ultime, le quali potrebbero generare (come in questo caso) delle inconsistenze. [?]

> [$Esame$] Quali sono gli elementi fondamentali di una policy?

> [$Esame$] Spiegare mediante definizione ed esempi i termini "inconsistenza" e "dilemma".

Un esempio reale di politica sicurezza: MAC

Un esempio reale di politica sicurezza è il modello **MAC** (acronimo di "*Mandatory Access Control*", ovvero "*Controllo di accesso mandatorio*"), un modo per scrivere o implementare politiche mandatorie basate sull'obbligo, le quali vengono utilizzate storicamente in ambito militare (Bell-Lapadula, Biba) o Sistemi Operativi ad alta sicurezza (per es. SELInux, AppArmor e Tomoyo).

> [$Approfondimento$] Mandatory Access Control (MAC) è un meccanismo di sicurezza che limita il livello di controllo che gli utenti (soggetti) hanno sugli oggetti che essi creano. A differenza di un DAC di attuazione, in cui gli utenti hanno il pieno controllo sui propri file, directory, ecc, MAC aggiunge etichette aggiuntive, o categorie, a tutti gli oggetti del file system. Utenti e processi devono avere l'accesso appropriato a queste categorie prima di poter interagire con questi oggetti.

Un esempio reale di politica sicurezza: RBAC

L'opposto rispetto il MAC è il modello **RBAC** (acronimo di *Role-Based Access Control*), un modo per scrivere politiche *non mandatorio* bensì modificabili e basate sui permessi associati a ciascun ruolo. Le policy role-based spesso prevedono una semplificazione delle modalità (per es. le ACL prevedono una semplificazione della modalità basata sul permesso).
Per completezza, definiamo il *permesso* di svolgere un attività a come l'assenza del divieto di svolgere quest'ultima attività, mentre definiamo il *divieto* di svolgere un attività a come il non permesso di svolgere a.
Il modello RBAC è spesso utilizzato all'interno dei comuni SO.

Un esempio di meccanismo implementativo: ACM

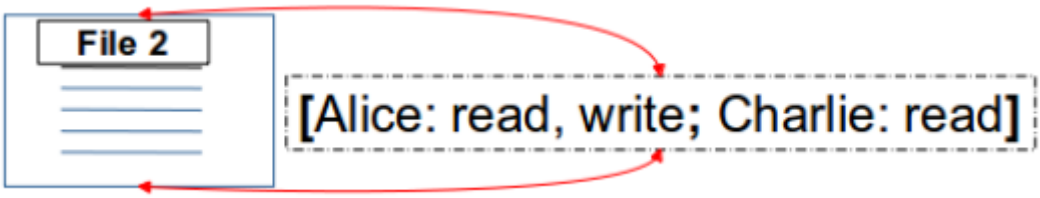
Un secondo esempio reale di politica di sicurezza è **ACM** (acronimo di "*Access Control Matrix*"), il quale si basa su una tabella che specifica quale utenza possa / abbia il *permesso* di fare svolgere una specifica attività.
Un esempio di ACM è rappresentato dalla seguente tabella.

User	File1	File2	Program1
Jax	Read, write	Read	Exec
Bob		Read	
Charlie	Read	Read, write	Read, exec

ACL e CaL

Due meccanismi implementativi per il meccanismo ACM sono:

- **ACL**, (acronimo di *Access Control List*), il quale sceglie di implementare l'ACM attraverso la memorizzazione di ogni colonna della matrice. Un esempio di ACL è la seguente: `Jax: Read, write, Bob: , Charlie: Read`.



- **CaL**, (acronimo di *Capability List*), il quale sceglie di implementare l'ACM attraverso la memorizzazione di ogni riga della matrice. Un esempio di CaL è la seguente: ``.

Modello di attaccante

Attraverso lo Schneier è possibile notare la tassonomia degli attaccanti, i quali possono essere classificati secondo moventi, risorse ed altri criteri ed essere raccontati secondo varie definizioni (e.g. hacker, servizi segreti ed altro ancora). Purtroppo limitarci a studiare quest'ultima sarebbe riduttivo. Per nostra fortuna il professore espleta al meglio l'argomento.

I **modelli di attaccante** ci consentono di astrarre dal contesto e dalle definizione (per es. *spionaggio industrale*, etc...) nella stessa misura in cui la complessità asintotica ci permette di astrarre dalle capacità computazionali della macchina.

Possiamo definire un modello di attaccante come la specifica (i.e *le capacità offensive*) di un preciso attaccante.

Un modello di attaccante è il modello **Dolev-Yao**, nato nel 1983 e così chiamato dai suoi autori "*Danny Dolev*" ed "*Andrew Yao*", il quale definisce l'attaccante come *unico* e *super-potente*, ovvero in grado di controllare l'intera rete, ma non di violare la **crittografia**.

Nel 2003, il prof. Bella definisce un nuovo modello di attaccante, il modello **General Attacker**, più aderente ai tempi recenti ed in cui l'attaccante non è uno solo bensì può essere chiunque. Quest'ultimi non necessariamente hanno interesse nel colludere con altri ma nel caso peggiore hanno totale controllo della rete e non sono in grado di violare la crittografia.