

Access Control

L'Access Control è uno dei principali meccanismi di protezione (insieme all'autenticazione) per proteggere un'organizzazione da tutti le tipologie di attacco viste fino a questo momento. In particolare, con i meccanismi di controllo degli accessi, andiamo a garantire la **confidenzialità** delle informazioni (dell'organizzazione) e dei sistemi che processano tali informazioni. Sorge a questo punto spontanea la domanda: **"Che cosa fa un sistema di controllo degli accessi?"** Un sistema di controllo degli accessi va a garantire, che le **risorse di un'organizzazione** (quali ad esempio: file, firewall, applicazioni Web e servizi Cloud) **non vengano accedute da utenti non autorizzati e allo stesso tempo, garantisce anche che utenti autorizzati non abusino dei loro privilegi per compiere delle azioni malevole ai danni dell'organizzazione** → per riuscire a fare questo, il sistema di controllo degli accessi va a specificare delle politiche di controllo degli accessi, le quali definiscono chi e che cosa (ovverosia quali entità) può compiere determinate azioni sulle risorse dell'organizzazione.

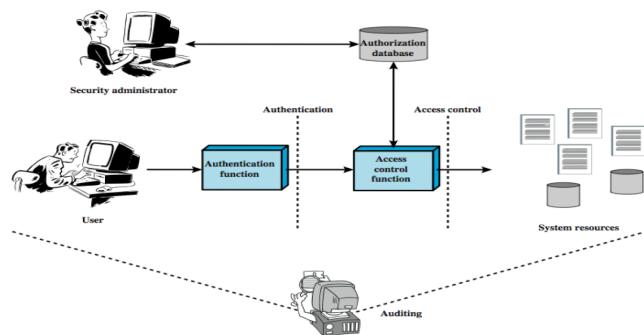
Alcuni esempi tipici di meccanismo di controllo degli accessi sono:

- quando condividiamo un file con Google Drive, in cui dobbiamo specificare la modalità di accesso del file con gli altri utenti (per esempio: solo lettura oppure modificabile);
- quando si vuole accedere al parcheggio universitario, è necessario mostrare il proprio badge universitario e questo è un classico esempio di controllo degli accessi fisico;
- quando scriviamo il codice di un'applicazione Web, specifichiamo le azioni che ogni tipologia di utente può compiere.

I sistemi di controllo degli accessi fanno parte di un sistema più ampio, il quale viene tipicamente implementato all'interno delle organizzazioni, e si tratta di un sistema di identità e gestione degli accessi chiamato **IAM System (Identity Access Management System)**. In questo sistema più ampio, il sistema di controllo degli accessi copre determinate funzionalità ed in particolare, abbiamo che l'IAM System è composto dalle seguenti componenti:

- innanzitutto, il processo di verifica dei permessi che ha un utente, inizia quando l'utente richiede l'accesso ad una risorsa aziendale;

- nel momento in cui l'utente fa la richiesta, come prima cosa viene verificata la sua identità attraverso uno dei meccanismi visti nelle lezioni precedenti (come per esempio: password oppure autenticazione a più fattori);
- una volta che l'identità dell'utente è stata verificata, entra in gioco la funzione di controllo degli accessi, la quale va a verificare se l'utente ha il permesso o meno di accedere alla risorsa aziendale di suo interesse → questa verifica viene fatta da una funzione apposita, la quale a sua volta utilizza delle politiche di controllo degli accessi, le quali sono specificate solitamente dall'amministratore di sicurezza dell'organizzazione e salvate in un apposito Database;
- un'altra componente dell'IAM System è quella di Auditing, ovverosia quella che mantiene traccia (ovverosia mantiene i logs) di tutte le attività svolte dagli utenti sulle risorse del sistema → tenere traccia dei logs ha due importanti vantaggi:
 - è possibile analizzare i logs e conseguentemente è possibile individuare dei possibili comportamenti anomali (associabili a degli attacchi al sistema) degli utenti;
 - è possibile identificare situazioni, in cui le politiche di controllo degli accessi sono state configurate in maniera errata da parte dell'amministratore di sicurezza dell'organizzazione. Con configurate in maniera errata, intendiamo il caso in cui, ad esempio, sono stati troppi privilegi ad una determinata categoria di utenti per accedere alle risorse aziendali oppure il caso in cui sono stati assegnati determinati privilegi agli utenti sbagliati.

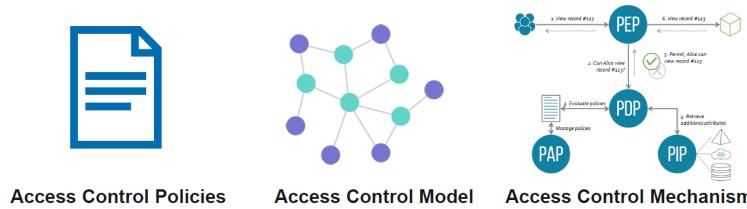


Ci chiediamo a questo punto: “**Quali sono le componenti di un sistema di controllo degli accessi?**” Le componenti sono essenzialmente tre, ovvero:

- le **politiche di controllo degli accessi** → ad alto livello definiscono quali sono le condizioni per cui un utente può accedere alle risorse dell'organizzazione. Da notare, che queste politiche devono essere formalizzate in accordo ad un

modello per essere applicate nella pratica, dato che un modello definisce quali sono i concetti in base a cui si definisce la politica;

- le **relazioni tra i concetti** che mi permettono di definire la politica di controllo degli accessi;
- l'**Access Control Mechanism** → questa componente è costituita a sua volta da due parti:
 - l'**architettura** implementata per raggiungere una decisione (di accesso garantito o accesso negato);
 - l'**algoritmo** implementato dai componenti dell'architettura per raggiungere una decisione (tenendo conto delle politiche di controllo degli accessi).



Se non consideriamo il modello, tutte le politiche di controllo degli accessi bene o male specificano tre informazioni, ovverosia:

1. i **soggetti** che richiedono l'accesso alle risorse del sistema → da notare il fatto, che il soggetto può essere un utente oppure un'applicazione utilizzata dall'utente per accedere alle risorse;
2. gli **oggetti**, ovverosia le risorse a cui vogliamo concedere l'accesso solamente agli utenti autorizzati → anche in questo caso, notiamo il fatto che gli oggetti possono essere: file, directory, record oppure programmi;
3. i **diritti di accesso (o permessi)**, i quali indicano la modalità con cui un soggetto può accedere agli oggetti → i classici permessi sono: lettura, scrittura, esecuzione, creazione, cancellazione e ricerca.

Le politiche di controllo degli accessi si possono specificare in base a quattro modelli:

1. **Discretionary Access Control (DAC)** → secondo questo modello, i permessi vengono direttamente associati ad un'**entità**, tipicamente un username, la cui identità è stata validata dal sistema di gestione delle identità e degli accessi. Questo modello viene chiamato Discretionary, in quanto l'utente può decidere se delegare ad altri utenti, i permessi che ha sulle risorse del sistema;

2. **Mandatory Access Control (MAC)** → modello nato per proteggere informazioni (quali ad esempio file altamente confidenziali) in ambito militare. L'idea alla base di questo modello, è che alle risorse (a cui gli utenti vogliono accedere) vengono assegnate delle etichette (dette **security labels**), che indicano il livello di confidenzialità associato alla risorsa. Agli utenti, invece, vengono associate delle altre etichette, che indicano a quali risorse possono accedere gli utenti;
3. **Role based Access Control (RBAC)** → questo modello non associa i permessi direttamente ad un'entità come il modello DAC, bensì associa i permessi ad un **ruolo**, il quale tipicamente rappresenta una funzione gerarchica all'interno dell'organizzazione;
4. **Attribute-based Access Control (ABAC)** → questo modello definisce le condizioni di accesso in base:
 - a. alle risorse a cui l'utente vuole accedere;
 - b. oppure in base all'utente stesso;
 - c. oppure in base all'ambiente, ovverosia il contesto in cui l'utente chiede l'accesso ad una determinata risorsa.

Vediamo in maniera più approfondita questi quattro modelli e di conseguenza partiamo con il primo, ovverosia il **Discretionary Access Control (DAC)** → in questo modello, la struttura dati tipicamente utilizzata è quella detta di **Matrice degli Accessi**. In tale matrice abbiamo che:

- le **righe** rappresentano gli **utenti**;
- le **colonne** rappresentano gli **oggetti** a cui gli utenti possono accedere;
- i **valori** delle celle sono i **diritti di accesso** che ha ogni utente sullo specifico oggetto.

Vediamo un esempio della matrice:

The diagram illustrates a matrix for discretionary access control. On the left, a bracket labeled "subjects" groups four users: alice, bob, charlie, and dave. At the top, a bracket labeled "objects" groups three files: news.doc, photo.png, and fun.com. The matrix itself is a grid where rows represent subjects and columns represent objects. The values in the cells indicate the access rights granted to each user for each object.

	news.doc	photo.png	fun.com
alice	read	view edit	view
bob	read write	view edit	view modify
charlie			
dave		view	

In base a questa struttura dati, quindi, quando un utente (come per esempio Alice) richiede accesso ad una delle risorse (come per esempio photo.png), il sistema di controllo degli accessi va semplicemente a controllare nella cella di intersezione tra la colonna dell'oggetto photo.png e la riga dell'utente Alice, se quest'ultima ha i privilegi richiesti. Questa rappresentazione (ovverosia la matrice degli accessi) ha due principali **problemi**, ovvero:

- è **poco scalabile** nel numero degli utenti;
- è **molto sparsa**, ovvero che alcune celle della matrice sono vuote.

Capiamo, allora, che la matrice degli accessi è una rappresentazione poco efficiente dei diritti di accesso degli utenti. Per risolvere questi problemi, si è deciso di decomporre la matrice per righe oppure per colonne ed in particolare abbiamo che:

- se la matrice viene decomposta per **colonne**, allora parliamo di **Access Control List**:

	news.doc	photo.png	fun.com
bob	read write	view, edit	view
alice	read	view, edit	view
			modify

Per ognuna delle risorse, abbiamo gli utenti che vi possono accedere e i relativi permessi, che gli utenti hanno sulla specifica risorsa.



Da notare, che l'Access Control List risolve parzialmente i problemi della matrice degli accessi, questo perchè risolviamo il problema di efficienza di memorizzazione dei diritti di accesso, ma dall'altra parte andiamo a complicare la gestione delle politiche di controllo degli accessi, in quanto rende difficile determinare quali sono i diritti di un determinato utente all'interno del sistema.

- per risolvere il problema appena sopra citato, si è deciso di decomporre la matrice per **righe** e allora in questo caso parliamo di **Capability List**:

	news.doc	photo.png	fun.com
Alice's capability	read	view, edit	view
Bob's capability	read write	view, edit	view edit
Charlie's capability			
Dave's capability		photo.png view	

Per ogni utente abbiamo le risorse a cui può accedere e con i relativi permessi.



Il vantaggio della Capability List è che rende immediatamente comprensibile se un utente ha accesso ad una determinata risorsa e questo permette inoltre di poter aggiornare rapidamente i permessi associati a ciascun utente e di trasferire facilmente i permessi di un utente ad un altro. Lo svantaggio, invece, è che se si deve cancellare una risorsa oppure determinare tutti i permessi assegnati ad una determinata risorsa comporta andare a scansionare tutte le Capability List di tutti gli utenti e controllare se l'utente ha un permesso associato alla risorsa.

Il modello DAC viene utilizzato in Linux, in cui ricordiamo che gli utenti vengono suddivisi in tre categorie:

- owner → utente che ha creato la risorsa;
- group → gruppo a cui appartiene l'utente;
- other → tutti gli altri utenti del sistema.

Per ognuna di queste categorie viene associato un set di permessi ed in particolare, abbiamo che i permessi vengono specificati nel seguente modo:

- il primo bit indica che stiamo trattando un file oppure una directory;
- poi abbiamo 9 bit, suddivisi in 3 gruppi ognuno da 3 bit, i permessi che hanno rispettivamente: l'utente, il gruppo a cui appartiene l'utente e tutti gli altri utenti del sistema;
- poi abbiamo l'owner;
- infine abbiamo il gruppo dell'utente.

Il sistema di controllo degli accessi di Linux, quando vogliamo accedere ad una risorsa, controlla:

- innanzitutto se l'utente è l'owner della risorsa e se è l'owner, tipicamente avrà l'accesso in lettura e in scrittura sulla risorsa. Se, invece, l'utente non è l'owner della risorsa, il sistema di controllo va a verificare se l'utente appartiene al gruppo dell'owner e in caso affermativo ha almeno il permesso di lettura del file. Se, invece, l'utente non appartiene al gruppo dell'owner e quindi fa parte della categoria other, tipicamente gli viene assegnato solamente il diritto in lettura della risorsa.

Adesso che abbiamo analizzato in maniera più approfondita il modello DAC, passiamo ad analizzare il modello **Role based Access Control (RBAC)** → l'idea alla base di questo modello, sfrutta il fatto che all'interno di un'organizzazione, una figura di maggiore responsabilità debba avere maggiori privilegi rispetto ad una figura con una bassa responsabilità. In un modello RBAC, quindi, invece che assegnare i privilegi direttamente all'utente, i privilegi vengono assegnati ai ruoli, i quali (ovverosia i ruoli) rappresentano i ruoli che gli utenti hanno all'interno dell'organizzazione (per esempio, all'interno dell'Università possiamo avere i seguenti ruoli: Studente, Docente, Capo dipartimento, Rettore, Tecnico, ecc...). Un primo modello RBAC è stato introdotto nel 1996, il quale prevedeva tre famiglie di modello di controllo degli accessi, le quali si differenziano per i concetti che implementano e che vengono utilizzati per specificare le politiche di controllo degli accessi e le relazioni tra questi concetti. Vediamo meglio queste tre famiglie di modelli:

1. **RBAC zero** → in questo modello abbiamo:
 - a. gli insieme degli **utenti** che lavorano all'interno dell'organizzazione;
 - b. i **ruoli**, che rappresentano effettivamente i ruoli all'interno dell'organizzazione;
 - c. i **permessi**, ovverosia le modalità con cui un utente (in questo caso, meglio dire un ruolo) può accedere ad una determinata risorsa del sistema;
 - d. la **sessione** → per capire il concetto di sessione, dobbiamo innanzitutto capire che: un utente può essere associato a più ruoli e quindi la relazione tra utenti e ruoli è di tipo N:N. Poi abbiamo la relazione (detta **Permission Assignment**), la quale specifica quali permessi sono assegnati ai ruoli, ovverosia specifica quali diritti di accesso che un ruolo può esercitare sulle risorse di sistema. Infine, abbiamo la relazione che sfrutta il concetto di sessione, ovvero: dato che un utente può essere associato a più ruoli contemporaneamente, per esercitare un ruolo lo deve prima di tutto attivare. L'operazione di attivazione di un ruolo viene associato al concetto di

sessione e quindi una sessione viene utilizzata per specificare quali ruoli, un determinato utente, ha attivi in un certo istante temporale (per esempio, un docente può essere sia docente sia studente di un corso di aggiornamento. Quindi, in una determinata sessione viene attivato il ruolo di docente, mentre in un'altra sessione viene attivato il ruolo di studente);

2. **RBAC uno** → questo modello introduce il concetto di **gerarchia dei ruoli**, ovverosia: un utente che ha un ruolo di maggiore responsabilità all'interno dell'organizzazione solitamente gode di maggiori permessi rispetto ad un utente che svolge solamente il ruolo di impiegato. Attraverso la gerarchia dei ruoli, quindi, andiamo a definire una **relazione di ereditarietà** tra i vari ruoli. Tipicamente, se all'interno della gerarchia abbiamo un ruolo che ne domina un altro, il ruolo più in alto nella gerarchia ereditarietà i permessi del ruolo più in basso → questo ha il vantaggio, che non dobbiamo andare a specificare, per tutti i ruoli all'interno dell'organizzazione, i permessi su tutte le risorse del sistema, bensì basta andare a specificare quei permessi aggiuntivi che sono specifici di un determinato ruolo, dato che i permessi dei ruoli sottostanti vengono automaticamente ereditati;

3. **RBAC due** → questo modello aggiunge il concetto dei **vincoli** ed in particolare, vi sono due principali tipologie di vincolo:

- vincoli di **cardinalità** → impongono condizioni, per esempio, su quanti utenti possono essere assegnati ad un ruolo (per esempio, ad un solo utente può essere assegnato il ruolo di capo dipartimento). Questa categoria di vincoli, comprende anche i vincoli che vanno a determinare il **numero di permessi assegnati ad un determinato ruolo** → vengono utilizzati questi vincoli, al fine di rispettare il principio di least privilege, secondo il quale ad un ruolo assegniamo solo i permessi necessari per compiere le attività lavorative;
- vincoli di **Separation of Duty** → tali vincoli vogliono prevenire situazioni di collusione tra gli utenti del sistema, ovvero si vuole prevenire che due o più utenti mettano insieme i privilegi di cui dispongono per tentare di bypassare il meccanismo di controllo degli accessi. In particolare, esistono due tipologie di vincoli di Separation of Duty ed entrambi richiedono la definizione di un insieme di ruoli:
 - **Static Separation of Duty** → si impone che l'utente possa essere assegnato a solamente uno dei ruoli, che fanno parte dell'insieme dei ruoli mutuamente esclusivi;

- **Dynamic Separation of Duty** → è relativo al concetto di sessione, ovvero che in una data sessione posso attivare solamente uno dei ruoli dell'insieme dei ruoli mutuamente esclusivi (quindi in una sessione, non possono essere attivati più ruoli).

Da notare, che il principale **vantaggio** del modello RBAC , è quello di facilitare l'assegnamento dei permessi ad un nuovo utente. Invece, i principali due **svantaggi** di questo modello sono:

- il controllo degli accessi è molto difficile da implementare correttamente, dato che il modello RBAC viene implementato, ad esempio: nelle banche, ospedali e piattaforme Cloud (dove quindi, vi sono moltissimi dipendenti, clienti, ruoli e cambi di ruoli);
- le organizzazioni assegnano ai dipendenti eccessivi privilegi (detto anche problema di **over-entitled**), infatti si stima che dal 50% al 90% dei dipendenti è sovraccarico di permessi nelle grandi organizzazioni.

Il modello attualmente più utilizzato è **Attribute-based Access Control (ABAC)** → questo tipo di modello di controllo degli accessi ci permette di specificare politiche molto specifiche, in quanto le condizioni di accesso possono essere specificate come condizioni relative:

- al **soggetto** che richiede l'accesso;
- alla **risorsa** a cui l'utente vuole accedere;
- all'**ambiente** in cui viene effettuato l'accesso.

Vediamo un esempio: Supponiamo di essere nel Veneto e conseguentemente abbiamo un portale, che ci permette di accedere al fascicolo elettronico.

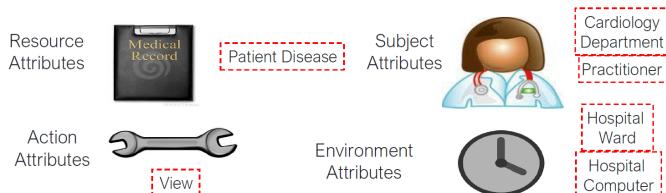
Tipicamente, il fascicolo elettronico viene acceduto da:

- i pazienti, i quali accedono al proprio fascicolo;
- i medici, che controllano il paziente, possono andare a visionare e modificare il fascicolo di quest'ultimo (ovvero del paziente).

Supponiamo, adesso di voler specificare la seguente politica: "Solamente i medici, che lavorano nel reparto di chirurgia, possono visualizzare i fascicoli dei pazienti che soffrono di malattie cardiache, utilizzando un computer dell'ospedale (il computer, quindi, deve essere connesso alla rete dell'ospedale), il quale deve essere installato nel reparto di chirurgia" → questa politica non potrebbe essere specificata con i modelli visti precedentemente, dato che abbiamo diverse condizioni, ovvero:

- il soggetto che accede al fascicolo deve essere un medico (e questo potremmo specificarlo anche con gli altri modelli), ma dobbiamo specificare che il medico lavora nel reparto di chirurgia;
- abbiamo anche delle condizioni sull'ambiente in cui viene effettuato l'accesso, in quanto il computer deve esser installato nell'ospedale ed in particolare nel reparto di chirurgia;
- i fascicoli elettronici, inoltre, che possono essere acceduti devono essere di pazienti, che soffrono di malattie cardiache;
- l'azione che si può essere svolta dai medici sui fascicoli è quella solamente di visualizzazione.

Tutte queste condizioni possono essere visualizzati meglio con la seguente immagine:



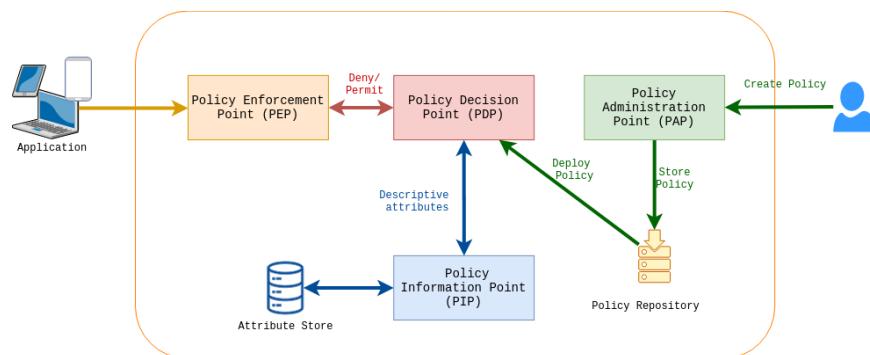
Tutte queste condizioni imposte su: **soggetto, ambiente, risorsa a cui si vuole accedere e azione da eseguire**, possono essere specificate attraverso una politica basata sugli attributi, quindi attraverso un modello ABAC.

Ovviamente per riuscire a fare questo, abbiamo bisogno di un approccio standard per specificare e valutare le politiche di controllo degli accessi. Lo standard tipicamente utilizzato, per implementare il modello ABAC, quindi è **XACML (eXtensible Access Control Markup Language)** → protocollo basato su XML e tale protocollo fornisce:

- un'**architettura** per **valutare** le politiche di controllo degli accessi → l'architettura di XACML consiste di quattro componenti principali:
 - **PEP (Policy Enforcement Point)** → interfaccia tra l'applicazione mediante cui l'utente richiede accesso ad una risorsa del sistema e le altre componenti dell'architettura (quindi il PEP si trova tra l'applicazione e le altre tre

componenti dell'architettura). Il PEP, quindi, intercetta una richiesta di accesso dell'utente e dopo che è stata presa una decisione riguardo al fatto se l'accesso alla risorsa debba essere garantito o meno, il PEP garantisce o meno l'accesso alla risorsa da parte dell'utente (possiamo, quindi, dire che il PEP gestisce l'accesso alle risorse del sistema);

- **PAP (Policy Administration Point)** → servizio che fornisce un'interfaccia all'amministratore di sistema per definire le politiche di controllo degli accessi, le quali successivamente vengono salvate in un apposito Database;
- **PIP (Policy Information Point)** → fornisce degli attributi riguardanti:
 - il soggetto che ha richiesto l'accesso;
 - oppure la risorsa a cui si vuole accedere;
 - oppure l'ambiente in cui è richiesto l'accesso.
- **PDP (Policy Decision Point)** → si interfaccia con il Database delle politiche di controllo degli accessi e recupera le politiche applicabili alla richiesta di accesso, che è stata fatta dall'utente. In alcuni casi, inoltre, il PDP si interfaccia con il PIP per recuperare gli attributi relativi al soggetto, alla risorsa oppure all'ambiente, in modo tale da riuscire a prendere le decisioni sul fatto se l'accesso (all'utente) deve essere garantito o meno.



In realtà, a volte, vi è un'altra componente tra il PEP e il PDP, ovverosia il **Context handler** → il Context handler fa da intermediario tra il PEP e il PDP, ovverosia quando il PEP invia una richiesta al Context handler, quest'ultimo la traduce nello standard imposto dal XACML e successivamente la manda al PDP.

Sorge a questo punto spontanea una domanda: **“Come cooperano tutte queste componenti, per decidere se l'accesso ad una determinata risorsa deve essere negato o meno?”** Ovviamente il tutto parte da una richiesta fatta dal soggetto, la quale viene intercettata dal PEP. Se la richiesta è in formato

diverso dallo standard imposto dal XACML, allora la richiesta viene inviata al Context handler, il quale traduce la richiesta nel formato dello standard imposto dal XACML e poi la invia al PDP. A questo punto, il PDP si interfaccia con il PAP per recuperare le politiche applicabili alla richiesta di accesso effettuata dal soggetto. Siccome ha anche bisogno di valutare il valore degli attributi indicate nelle condizioni delle politiche di controllo degli accessi, il PDP chiede al Context handler di fornire gli attributi di cui necessita. Per fare questo, il Context handler rigira la richiesta al PIP, il quale recupera gli attributi, ad esempio del soggetto che effettua la richiesta di accesso, e tali attributi vengono rinvolti al Context handler, il quale a sua volta li manda al PDP. A questo punto, il PDP ha tutti gli elementi per valutare se l'accesso deve essere garantito o meno e di conseguente il PDP prende una decisione, la quale viene restituita al Context handler, che poi viene restituita al PEP che effettivamente la implementa.

Sottolineiamo anche il fatto, che possiamo avere due scenari di schieramento delle componenti:

1. **Centralized Scenario** → in cui tutte e quattro le componenti fanno parte della stessa organizzazione e quindi si dice, che tutte le componenti appartengono allo stesso **Trusted Domain**. Per esempio, l'ospedale avrà un Server dedicato per ospitare ogni componente;
2. **Cloud Scenario** → scenario più complesso rispetto allo Centralized Scenario ed in particolare, in questo scenario abbiamo un'organizzazione, che utilizza una serie di servizi Cloud. In questo caso: PEP, PDP, PAP e PIP vengono implementati dal Cloud provider, il quale fornisce dei servizi online all'organizzazione. L'unica cosa che l'organizzazione può fare, è di utilizzare l'interfaccia fornita dal PAP per configurare le politiche su come i clienti e i dipendenti possono accedere alle applicazioni e ai servizi Cloud forniti dall'organizzazione. Quindi, quando un utente dell'organizzazione vuole accedere ad un'applicazione, la richiesta viene intercettata dal PEP, il quale è implementato come un servizio Cloud dal Cloud provider, il PEP manda la richiesta al PDP, il quale prende una decisione in base alle politiche implementate dall'amministratore di sistema dell'organizzazione.
 - un **linguaggio per specificare** le politiche;
 - un **linguaggio per richiedere** l'accesso ad una determinata risorsa e per ottenere una risposta da parte del sistema di controllo degli accessi.

Parliamo un attimo del linguaggio adottato da XACML per specificare le politiche. Tipicamente, una politica di controllo degli accessi viene rappresentata da un

elemento chiamato “**Policy**”, il quale a sua volta è composta da uno o più elementi di “**Rule**”, i quali (ovverosia gli elementi di “Rule”) specificano le condizioni che devono essere valutate per decidere se l’accesso deve essere garantito o meno. Da notare, che le regole non possono essere valutate per prendere una decisione di controllo degli accessi, bensì devono essere valutate a livello di politica → di fatto, una politica consiste da uno o più elementi Rule, i quali vengono tutti valutati dal PDP.



Sia le politiche sia le regole devono specificare a quali richieste di accesso vengono applicate e questo viene fatto andando a specificare, sia nella politica sia nella regola, l’elemento **Target**.

L’elemento Target è stato progettato per trovare le politiche che si applicano ad una richiesta e specifica le condizioni rispetto a:

- soggetti;
- risorse;
- azioni.

L’elemento Target a sua volta è composto da altri tre elementi:

- l’elemento AnyOf → esprime la disgiunzione degli elementi AllOf (ovverosia che solamente uno degli elementi di AllOf deve essere verificato);
- l’elemento AllOf → esprime la congiunzione dell’elemento Match (ovverosia che tutti gli elementi di Match devono essere verificati);
- l’elemento Match → specifica una condizione rispetto a: soggetti, risorse e azioni.

Per quanto riguarda, invece, l’elemento Rule abbiamo che i principali elementi di cui è composto sono:

- l’elemento **Target** → condizioni per determinare se una regola è applicabile a una richiesta di accesso;
- l’elemento **Effect** → conseguenza della valutazione a vero di una regola;
- l’elemento **Condition** → espressione booleana che affina l’applicabilità di un criterio;;
- l’elemento **Obligation** (opzionale) → operazioni/azioni da eseguire con una decisione di autorizzazione.

Ricordiamo, inoltre, che le politiche sono di fatto un insieme di uno o più elementi di tipo Rule e per determinare se l'accesso deve essere garantito o negato a una determinata risorsa, vengono utilizzati degli algoritmi di **Rule Combining**, i quali specificano come le regole possono essere combinate tra loro, per far determinare al PDP se l'accesso deve essere garantito oppure negato. In particolare esistono quattro tipologie di algoritmi di Rule Combining, ovvero:

1. **Deny-Overrides** → in questa tipologia, abbiamo che la decisione di negazione ha la priorità sulla decisione di accesso. Se tutte le decisioni di accettazione, allora la decisione è di accettazione, mentre se tutte le decisioni sono indeterminate, allora il risultato è indeterminato. Altrimenti, il risultato è non applicabile;
2. **Permit-Overrides** → la decisione di autorizzazione ha la priorità sulla decisione di negazione. Se tutte le decisioni sono indeterminate, il risultato è indeterminato, mentre se tutte le decisioni sono negate, allora il risultato è negato. Altrimenti, il risultato è non applicabile;
3. **First-applicable** → il risultato è l'effetto della prima regola il cui Target è valutato come vero;
4. **Only-one-applicable** → si applica quando si ha più di una politica, che viene raggruppata all'interno di un elemento di PolicySet. Se nessun criterio è applicabile, allora il risultato è non applicabile. Se, invece, più di un criterio è applicabile, allora il risultato è indeterminato ed infine, se solo una politica è applicabile, allora il risultato è il risultato della valutazione della politica.

OAuth Authorization Flow

Il protocollo di autorizzazione OAuth serve in quegli scenari, in cui abbiamo un utente che ospita le proprie risorse (quali ad esempio: il proprio profilo) su un Server HTTP e vuole accedere a queste risorse utilizzando un'applicazione Client (la quale può essere, ad esempio: un'applicazione sullo smartphone oppure un'applicazione Web) e naturalmente, tale applicazione per accedere alle risorse deve richiamare le API fornite dal Server HTTP → l'utente, quindi, deve garantire all'applicazione il permesso per accedere alle proprie risorse ospitate sul Server HTTP. La prova, che l'utente ha autorizzato l'applicazione ad accedere alle proprie risorse, prende la forma di un **token**, il quale viene rilasciato un'entità chiamata **Authorization Server**.

Nel protocollo OAuth abbiamo quattro entità principali:

- **Client** → applicazione di terze parti, che richiede l'accesso a risorse protette per conto dell'utente proprietario della risorsa e con la sua approvazione;
- **Resource Owner** → entità in grado di concedere all'applicazione l'accesso ad una risorsa protetta;
- **Authorization Server** → il Server che rilascia il token di accesso al Client, dopo aver autenticato il proprietario della risorsa e aver ottenuto la sua autorizzazione;
- **Resource Server** → il Server che memorizza le risorse dell'utente proprietario e quando riceve l'access token decide se garantire o meno l'accesso alla risorsa.

Lo standard supporta diversi scenari di autenticazione, i quali si differenziano in base al tipo dell'applicazione Client che viene utilizzata per accedere alle risorse. In particolare, possiamo avere i seguenti scenari:

- **Authorization Code Grant Flow** → viene utilizzato quando l'applicazione Client utilizzata per accedere alle risorse , è un'applicazione che ha un Back-end Server, ovverosia il codice dell'applicazione gira su un Web Server;
- **Authorization Code Grant Flow with PCKE** → viene utilizzato quando l'applicazione è un'applicazione:
 - browser based, ovverosia è un'applicazione eseguita interamente sul browser;
 - oppure è un'applicazione mobile.
- **Resource Owner Password** → viene utilizzato quando l'applicazione Client è fornita dalla stessa organizzazione che fornisce l'Authorization Server e il Resource Server e di conseguenza, l'applicazione è nota all'organizzazione che verifica il permesso di accedere alle risorse;
- **Client Credential** → viene utilizzato quando l'accesso ad una risorsa ospitata sul Server HTTP non è fatto per conto di un utente, bensì l'accesso è fatta per conto di un'applicazione e quindi quando abbiamo un'interazione macchina-macchina;
- **Device Flow** → viene utilizzato quando l'applicazione Client utilizzata per accedere alle risorse, è un'applicazione che gira su un dispositivo che ha un'interfaccia limitata o comunque limitate capacità di input.

A questo punto, analizziamo singolarmente le tipologie di protocollo. Partiamo con la prima, ovvero **Authorization Code Grant Flow** → abbiamo una fase iniziale, in cui l'applicazione deve essere registrata attraverso l'Authorization Server. Quindi, la

prima cosa che fa l'utente (ovverosia il Resource owner) è di registrare l'applicazione presso l'Authorization Server e nella registrazione specifica tre aspetti:

1. il nome dell'applicazione;
2. il tipo di Authorization Flow che vuole condurre l'Authorization Server.
3. redirect URI, ovverosia il link a cui l'Authorization Server redirigerà il browser dell'utente una volta che l'utente avrà autorizzato l'applicazione ad accedere alle proprie risorse.

Al termine della fase di registrazione l'applicazione riceve:

- un **client_id**, il quale identifica l'applicazione in maniera univoca;
- un **client_secret**, che l'applicazione deve essere in grado di memorizzare in modo sicuro.

Quando avviene il processo di autorizzazione, tipicamente abbiamo che l'Authorization Server fornisce un'interfaccia all'utente per garantire l'autorizzazione all'applicazione ad accedere alle proprie risorse. **“Come fa l'utente a raggiungere questa interfaccia?”** Tipicamente, l'applicazione Client avrà un bottone Login e quando l'utente ci cliccherà sopra, verrà rediretto alla pagina dell'Authorization Server, dove appunto verrà chiesto all'utente di autorizzare l'applicazione Client ad accedere alle proprie risorse. **Prima** che l'utente decida se autorizzare o meno l'applicazione ad accedere alle proprie risorse (ovvero quando l'utente clicca il bottone di login), le informazioni vengono inviate all'Authorization Server ed in particolare, le informazioni inviate sono:

- il tipo di risposta che si aspetta l'applicazione (nel nostro caso un **authorization code**);
- il client_id ottenuto al momento della registrazione;
- il redirect URI specificato al momento della registrazione;
- le informazioni dell'utente a cui l'applicazione vuole accedere (come per esempio il bank account), le quali vengono inserite in un campo **“scope”**;
- un valore specificato dall'attributo “state”, ovverosia un valore casuale utilizzato per prevenire replay attacks.

Una volta che l'utente si autentica e garantisce il permesso all'applicazione di accedere alle proprie risorse, l'Authorization Server restituisce un codice all'applicazione e inserisce lo stesso valore inserito dall'applicazione, nel momento

della richiesta, per consentire all'applicazione di verificare che il codice è stato inviato nella stessa sessione di autorizzazione.

Dopodiché, l'applicazione deve scambiare l'authorization code con un token di accesso e quindi, l'applicazione contatta nuovamente l'Authorization Server e gli passa l'authorization code e le stesse informazioni ottenute nella fase di registrazione (quindi: client_id, redirect URI e il client_secret). Se i dati corrispondono con quelli inviati dall'Authorization Server, allora quest'ultimo invia all'applicazione:

- l'access token;
- un refresh token, che l'applicazione può utilizzare per ottenere un nuovo access token senza la necessità di rieseguire i passi precedenti.

Infine, per accedere alle risorse, l'applicazione Client presenta l'access token (ottenuto dall'Authorization Server) al Resource Server, quest'ultimo ne verifica la validità e in caso affermativo, l'applicazione può accedere alle risorse dell'utente.

Il secondo scenario, invece, è **Authorization Code Grant Flow with PCKE**, il quale è molto simile allo scenario visto precedentemente, con l'unica differenza che viene utilizzato in aggiunta una misura di sicurezza → in particolare, viene utilizzata una misura di sicurezza in più, perché l'applicazione Client utilizzata per accedere alle risorse dell'utente, non è in grado di memorizzare in modo sicuro il client_secret restituito dall'Authorization Server all'applicazione. Quindi, anche in questo scenario abbiamo una fase di registrazione iniziale, con la differenza che non mi viene restituito un client_secret, bensì mi viene restituito solamente il client_id, il quale permette di identificare in maniera univoca l'applicazione che vuole accedere alle risorse dell'utente. A questo punto, l'applicazione genera due valori:

1. **code verifier** → stringa casuale compresa tra i 43 e 128 caratteri;
2. **code challenge** → ottenuto facendo l'hash del code verifier e poi verificando l'hash in base 64.

Attraverso la code challenge, abbiamo che l'utente clicca sul bottone di login presente nell'interfaccia fornita dall'applicazione e viene rediretto alla solita pagina dell'Authorization Server, dove appunto verrà chiesto all'utente di autorizzare l'applicazione Client ad accedere alle proprie risorse. Quando l'utente clicca il bottone di login, viene specificato:

- il tipo di risposta che si aspetta l'applicazione;
- il client_id;

- il redirect URI;
- il tipo di informazioni a cui vuole accedere;
- **il valore della challenge che è stato appena generato;**
- **l'algoritmo utilizzato per generare e verificare la challenge.**

Quando poi l'utente autorizza l'applicazione Client ad accedere alle proprie risorse, l'Authorization Server invia un authorization code e a questo punto, l'applicazione deve scambiare l'authorization code con un access code. Quando l'applicazione invia questa richiesta, **l'applicazione non invia il client_secret, bensì invierà il code verifier**. L'Authorization Server, a questo punto, va a verificare che la code challenge sia la stessa di quella che l'applicazione gli aveva inviato al passo precedente. Se le due code challenge sono le stesse, allora l'Authorization Server restituisce l'access token e il refresh token all'applicazione e a questo punto l'applicazione può accedere alle risorse dell'utente.

Il terzo scenario, invece, è il **Resource Owner Password** → in questo caso, supponiamo che l'utente voglia accedere al proprio profilo Google utilizzando l'applicazione Gmail fornita da Google, la quale (ovvero Gmail) gioca anche il ruolo di Resource Server e Authorization Server. La prima cosa che fa l'utente è collegarsi con il proprio account Google all'applicazione Client, ovverosia Gmail. Le credenziali vengono passate all'Authorization Server (ovvero Google) specificando:

- il flusso di autorizzazione, che in questo caso è “password” (nei due scenari precedenti era “code”);
- il client_id;
- le credenziali dell'utente, in particolare vengono inviati l'username e la password dell'utente;
- la risorsa a cui si vuole accedere (nel nostro caso Gmail).

L'Authorization Server (ovvero Google) verifica che le credenziali siano corrette e gli restituisce immediatamente l'access token e un refresh token all'applicazione Gmail. A questo punto, l'applicazione Gmail inoltra la richiesta per accedere alla casella di posta dell'utente allegando l'access token e ottiene l'accesso alla casella di posta.

Il quarto scenario, ovverosia il **Client Credential**, è lo scenario in cui l'accesso non viene richiesto per conto di un utente, bensì viene richiesto per conto di un'applicazione. Supponiamo di avere un'applicazione, che ospita i suoi file su un servizio di Cloud Storage (come quello fornito da Google), quindi in questo caso, l'applicazione deve accedere alle proprie risorse, le quali sono memorizzate su

Google Cloud → in questo scenario, abbiamo sempre che l'applicazione deve registrarsi e ottenere un client_id e un client_secret, il quale viene memorizzato all'interno del codice dell'applicazione. Dopodiché, l'applicazione genera una richiesta per Google, il quale gioca il ruolo di Authorization Server e di Resource Server, allegando il client_id e il client_secret. A questo punto, Google verifica che il client_id e il client_secret siano quelli ottenuti dall'applicazione in fase di registrazione e nel caso siano uguali, Google restituisce l'access token all'applicazione, che potrà poi utilizzare per accedere alle proprie informazioni salvate su Google Cloud.

Infine, abbiamo lo scenario di **Device Flow** → supponiamo di avere l'applicazione YouTube che gira su una Apple TV, mediante la quale (ovvero l'Apple TV) l'utente vuole accedere alle proprie risorse memorizzate su YouTube. Anche in questo caso, quindi, Google gioca il ruolo sia di Authorization Server sia di Resource Server. In questo scenario, abbiamo che l'applicazione YouTube ha un bottone di login e quando l'utente clicca sul bottone di login, viene inviata una richiesta all'Authorization Server Google contenente:

- il client_id dell'applicazione YouTube;
- la tipologia di flusso di autorizzazione che vuole avere l'Authorization Server.

Dopodiché, l'Authorization Server Google invia una risposta all'applicazione YouTube indicando due informazioni importanti:

1. un **URL**, che deve essere visitato dall'utente con un altro dispositivo (laptop o smartphone);
2. un **codice**, che deve essere inserito dall'utente per iniziare il processo di autorizzazione.

L'applicazione mostrerà una pagina contenendo queste informazioni all'utente e a questo punto l'utente deve visitare l'URL con un altro dispositivo e inserire il codice, il quale viene inviato all'Authorization Server Google.

L'Authorization Server Google mostra la classica pagina all'utente, in cui gli viene chiesto di autorizzare o negare l'accesso all'applicazione YouTube. Nel mentre, l'applicazione YouTube manda al Server Google delle post_request contenenti:

- il client_id;
- un codice identificativo inviato dall'Authorization Server.

Queste post_request servono per far ottenere all'applicazione YouTube l'access token. Se l'utente ha garantito l'accesso all'applicazione YouTube, quest'ultima

ottiene l'access token dall'Authorization Server Google e a questo punto, l'applicazione YouTube può accedere al profilo dell'utente.



Riassumendo, quindi, possiamo dire che OAuth è un protocollo di autorizzazione che consente a un utente finale (il Resource owner) di concedere l'accesso a un'applicazione Client a risorse protette con il consenso dell'utente finale.