Vittorio cortosi

Sicurezza II

Index

        - User-agent
        - Location
        - Server
        - Content-encoding
- **MIME Types**
- **HTTP Authentication**
    - Autenticazione Proxy
    - Basic Authentication
- **OPENSSL**
- **Apache**
    - Funzionamento
    - Configuration Files
        - Direttive e sezioni
    - Modules
    - Basic Authentication con Apache2
    - HTTPS con Apache2
        - Creare chiavi e certificato per il server
    - Muthual Authentication con Apache2
- **Docker**
    - Immagini
        - Layers
        - Layer caching and reusing
    - Containers
    - Volumes
    - Creare un immagine
    - Creare un container
    - Avviare un container
    - Docker Compose
        - Creare un container con docker compose
        - Avviare un container con docker compose
- **PKI**
    - Infrastruttura
    - X.509
        - Tipi di certificati
        - Struttura del certificato
        - Estensioni
            - SAN (Subject Alternative Name)
        - Revoca dei certificati
        - OCSP
        - Estensioni di file per i certificati
- **SSL/TLS**
    - SSL Architecture
    - Algoritmi di scambio chiavi supportati
    - Handshake Protocol
        - Fase1: Hello Messages
        - Fase2: Server Authentication and key exchange
        - Fase3: Client Authentication and key exchange
    - Change Cipher Spec Protocol
    - Alert Protocol
    - Record Protocol
    - Header SSL
- **LDAP**

IAM

È importante considerare il concetto di sicurezza in generale come un problema legato all'identità. Questo punto di vista ci consente di vedere la sicurezza informatica non solo come un problema, ma anche come una soluzione. Inoltre, tale approccio semplifica notevolmente la presentazione delle soluzioni di sicurezza ai clienti. Per molti anni, la sicurezza ha incontrato difficoltà nel farsi comprendere dai livelli decisionali delle organizzazioni. Gli esperti di sicurezza spesso si trovavano a comunicare in un linguaggio incomprensibile per il management.

Il concetto di Identity ci permette di invertire completamente l'approccio, partendo dalla soluzione stessa. Ad esempio, esponiamo il fatto di avere degli utenti che devono svolgere il proprio lavoro. Tuttavia, per permettere loro di lavorare in modo sicuro, è necessario che possano autenticarsi in modo appropriato. L'autenticazione tramite un sistema integrato diventa indispensabile, poiché affidarsi a password diverse per ogni computer o servizio genera disorganizzazione e insicurezza. Pertanto, è fondamentale adottare un sistema di Identity Management integrato e ben strutturato.

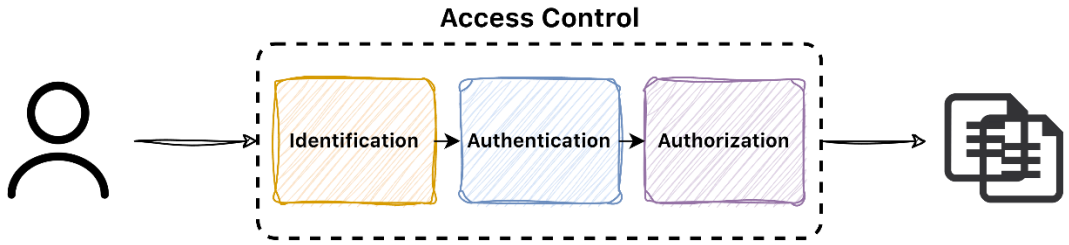
Si comprende immediatamente che questo modo di presentare il concetto è molto diverso. Si abbandoneranno le discussioni sugli hacker, i virus o le perdite di dati, per concentrarsi invece sulla gestione di un database di utenti, la gestione delle credenziali, il controllo del parco macchine, l'assicurazione dell'integrità dei server e la riservatezza delle comunicazioni.

Si passa anche dai servizi di sicurezza che operano in modo episodico e quindi inefficace, a servizi di sicurezza che sono continuativi. Questo è il motivo per cui il termine "management" è sempre più presente nel contesto della sicurezza informatica (Identity and Access Management, Security Incident and Event Management, Vulnerability Management). Il concetto di gestione continua è al centro della filosofia dell'ISO 27001 e del GDPR. È necessaria una gestione continuativa.

Ad esempio, nell'ambito dell'Identity Management, non si tratta semplicemente di abilitare l'utente una volta, fornirgli una password e lasciarlo a se stesso per sempre. Si tratta invece di gestire e mantenere l'identità nel tempo. In questo modo, si affrontano problemi che in passato non venivano considerati. Ad esempio, un errore comune nelle grandi organizzazioni era quello di mantenere l'identità di un utente, insieme alla sua password, anche dopo che quest'ultimo era andato via, licenziato o aveva cambiato lavoro. Questa situazione rappresentava una vulnerabilità classica nell'approccio precedente. Pertanto, uno degli obiettivi del sistema di Identity Management è proprio il corretto disabilitamento delle identità che non vengono più utilizzate.

Access Control

L'access control è un insieme di tecnologie e metodologie utilizzate per regolare l'accesso alle risorse o servizi. L'obiettivo principale dell'access control è quello di proteggere le risorse, come dati, informazioni sensibili o sistemi informatici, limitando l'accesso solo alle persone autorizzate.



La fase di controllo di accesso viene monitorata normalmente, quindi esiste ad esempio un log degli accessi che associa ad ogni accesso, l'utente che lo ha eseguito.

Questa fase serve anche per evitare usi illeciti di utenti autenticati: non non basta che l'utente sia autenticato. L'utente autenticato si, può fare certe cose ma non è detto che possa fare tutto. Nasce quindi il concetto di privilegio, inteso come insieme di autorizzazioni che l'utente possiede.

Enrollment

Nell'ambito dell'Identity Management, l'enrollment è il processo attraverso il quale un utente viene registrato o aggiunto al sistema. Durante questa fase, vengono raccolte le informazioni personali e le credenziali necessarie per creare un account o un profilo utente.

L'enrollment può includere diversi passaggi, a seconda del contesto e dei requisiti del sistema. Solitamente, l'utente viene richiesto di fornire informazioni come il nome, l'indirizzo, il numero di telefono, l'indirizzo email e altre informazioni identificative. Inoltre, potrebbero essere richieste credenziali di accesso, come username e password, che l'utente dovrà utilizzare per autenticarsi successivamente nel sistema.

Durante l'enrollment, potrebbero essere richiesti anche altri processi, come la verifica dell'identità dell'utente attraverso metodi di autenticazione, come la verifica via email, l'invio di un codice di conferma o l'utilizzo di tecnologie di biometria.

Identification

L'identificazione è il processo mediante il quale un utente afferma o rivendica un'identità unica all'interno del sistema. L'identificazione avviene solitamente attraverso l'uso di un nome utente, un indirizzo email o un numero di identificazione univoco. L'obiettivo dell'identificazione è stabilire chi sia l'utente, ma non fornisce alcuna garanzia sulla veridicità delle informazioni fornite. In altre parole, l'identificazione è il primo passo per identificare un utente all'interno del sistema, ma non conferma la sua identità in modo affidabile (tutti sono bravi a ricordarsi una username, ecco).

Authentication

L'autenticazione è il processo mediante il quale il sistema verifica l'identità dichiarata dall'utente. Consiste nel fornire prove che dimostrino che l'utente sia effettivamente colui che afferma di

essere. Ciò viene fatto richiedendo all'utente di fornire una prova di identità, ad esempio una password, un PIN, un'impronta digitale, una scansione di retina o un certificato digitale. L'autenticazione garantisce che l'utente sia legittimo e ha il diritto di accedere alle risorse o alle funzionalità del sistema.

I fattori di autenticazione di uno schema di autenticazione possono includere:

- **Qualcosa che l'utente ha**: qualsiasi oggetto fisico in possesso dell'utente, come un token di sicurezza (chiavetta USB), una carta bancaria, una chiave, ecc.
- **Qualcosa che l'utente sa**: alcune conoscenze note solo all'utente, come password, PIN.
- **Qualcosa che l'utente è**: alcune caratteristiche fisiche dell'utente (dati biometrici), come un'impronta digitale, l'iride, la voce, la velocità di battitura, lo schema negli intervalli di pressione dei tasti, ecc.

Se si utilizza un'autenticazione **MFA** (**Multi-Factor Authentication**), in fase di autenticazione viene concesso l'accesso solo dopo aver presentato con successo due o più fattori. In alcuni casi l'MFA è addirittura diventato obbligatorio, per esempio, nei sistemi bancari è obbligatorio per normativa europea.

L'utilizzo di più fattori di autenticazione per dimostrare la propria identità si basa sul presupposto che è improbabile che un attore non autorizzato sia in grado di fornire i fattori richiesti per l'accesso. Se, in un tentativo di autenticazione, almeno uno dei componenti manca o viene fornito in modo errato, l'identità dell'utente non viene stabilita con sufficiente certezza e l'accesso alla risorsa.

Autenticazione con Token

Un token di sicurezza è un dispositivo periferico utilizzato come una chiave elettronica per accedere a qualcosa. Esempi di token di sicurezza includono keycard wireless (smartcard) o un token bancario utilizzato come autenticatore digitale per accedere all'online banking o firmare una transazione come un bonifico bancario. Alcuni design incorporano meccanismi di anti-manomissione (Tamperproofing): è necessario che un qualsiasi tentativo di manipolazione hardware o software del dispositivo di accesso, sia impedita proteggendo i segreti memorizzati e rendendo impossibile la lettura fisica. Nel caso in cui si provasse a leggere i dati, è molto probabile che il dispositivo si autocorrompa i dati, così da rendere impossibile la lettura.

Tutti i token contengono alcune informazioni segrete che vengono utilizzate per dimostrare l'identità. Ci sono quattro diversi modi in cui queste informazioni possono essere utilizzate:

- **Token statico**: Il dispositivo contiene una password che è fisicamente nascosta (non visibile al possessore), ma che viene trasmessa ad ogni autenticazione. Questo tipo è vulnerabile agli attacchi replay.
- **Token dinamico sincrono**: Un timer viene utilizzato per ruotare attraverso varie combinazioni monouso prodotte da un algoritmo crittografico. Il token e il server di autenticazione devono avere orologi sincronizzati.
- **Token asincrono**: Una password monouso viene generata senza l'uso di un orologio, da un one-time pad o da un algoritmo crittografico.
- **Token Challenge-response**: Utilizzando la crittografia a chiave pubblica, è possibile provare il possesso di una chiave privata senza rivelare tale chiave. Il server di autenticazione crittografa una sfida (tipicamente un numero casuale, o almeno dati con alcune parti casuali) con una chiave pubblica; il dispositivo dimostra di possedere una copia della chiave privata corrispondente fornendo la sfida decrittografata.

I Token monouso sono anche chiamati **OTP** (**One-Time Password**). Quindi sia i Token dinamici sincroni che asincroni sono dei tipi di OTP.

I Token dinamici sincroni, cambiano costantemente a un intervallo di tempo prestabilito utilizzando algoritmi **TOTP** (**Time-Based One-Time password**); ad esempio, una volta al minuto. Per fare ciò, deve esistere una sorta di sincronizzazione tra il token del client e il server di autenticazione. Per i token disconnessi, questa sincronizzazione dell'ora viene eseguita prima che il token venga distribuito all'utente. Altri tipi di token eseguono la sincronizzazione quando il token viene inserito in un dispositivo di input.

Il problema principale con i token sincronizzati nel tempo è che possono, nel tempo, diventare non sincronizzati. Tuttavia, alcuni di questi sistemi, come SecurID di RSA, consentono all'utente di risincronizzare il server con il token, a volte inserendo diversi passcode consecutivi. La maggior parte inoltre non può avere batterie sostituibili e dura solo fino a 5 anni prima di dover essere sostituita, quindi c'è un costo aggiuntivo. Che cosa succede inoltre, quando l'utente ha finito le ne password? Bisogna inizializzare tutto, bisogna che il client generi altre N password ed ovviamente, utilizzando un canale sicuro, andarle a memorizzare sul server o lavorando di persona sul server. È chiaro che questo approccio è veramente scomodo, perché l'utente necessita di mantenere in memoria un sacco di password, supponendo che usi tanti servizi diversi: se per ogni servizio l'utente deve ricordare N passowrd, diventa veramente scomodo ed improponibile.

I Token asincroni invece utilizzano algoritmi **HOTP** (**HMAC-based one-time password**), implementando una catena di digest, per generare una serie di password monouso da una chiave condivisa segreta. Ogni password è unica, anche quando sono note le password precedenti. Ogni password è indipendente dalle precedenti, per cui un avversario non sarebbe in grado di indovinare quale potrebbe essere la password successiva, anche conoscendo tutte le password precedenti, evitando attacchi di Reply. Ovviamente sia sul client che sul server è necessario che ci sia la password iniziale, quindi anche qui un protocollo sicuro per lo scambio della password iniziale, e sul numero N di chiavi che andranno utilizzate.

Il Token sincrono è comodo, però è necessario avere a disposizione un token di autenticazione che abbia un minimo di potenza di calcolo necessaria almeno ad eseguire un hash function.

Poi questi sistemi sono soggetti anche ad altri problemi di natura pratica, per esempio il fatto che l'utente perda il token, in quel caso bisogna ridistribuirlo. Poi il sistema poi logistico è complesso, soprattutto quando si ha a che fare con milioni di utenti.

Vantaggi:

- È sicuramente da un punto di vista teorico molto più sicuro di qualsiasi altro metodo.

Svantaggi:

- Svantaggi di carattere pratico: il token può essere perso o rubato, naturalmente scomodo,
- Costi di distribuzione
- Costi di gestione

Autenticazione biometrica

L'autenticazione biometrica, come dice la parola, viene fatta sulla base di caratteristiche fisiche biologiche della dell'utente che vengono misurate biometricamente, la più diffusa è l'autenticazione attraverso impronta digitale, ma anche quella con il volto. Poi scanner retinico, comportamentali (camminata, scrittura), vocali.

Nel caso dell'identificazione l'utente con biometria, l'utente non dice chi e perché è il sistema che lo identifica: l'utente si presenta fornendo semplicemente la propria biometria.

La biometria permette di stabilire che ogni individuo ha caratteristiche:

- **Universali**: tutti devono averla;
- **Uniche**: due o più individui non possono avere la stessa uguale caratteristica;
- **Permanenti**: questa non varia nel tempo;
- **Collezionabili**: deve essere misurata quantitativamente.

Alcune caratteristiche fisiologiche di un individuo sono abbastanza stabili, soggette solo a piccole variazioni nel tempo, le componenti comportamentali invece possono essere influenzate dalla situazione psicologica dell'individuo, proprio per questo devono essere aggiornate spesso.

Per funzionare, il sistema deve in principio archiviare quello che viene chiamato uno **specimen** (in biologia è un campione biologico) estraendo i **parametri/features**. Durante l'identificazione, lo speciman viene confrontato con i parametri estratti dal sensore biometrico.

Comunque tutti questi sistemi, siano essi comportamentali o fisici, sono imprecisi. Ha un margine d'errore normalmente anche abbastanza alto, soprattutto quelle comportamentali hanno un forte errore. Ovviamente l'errore è dato da una somiglianza tra una persona e l'altra (quindi tra uno speciman ed un altro).

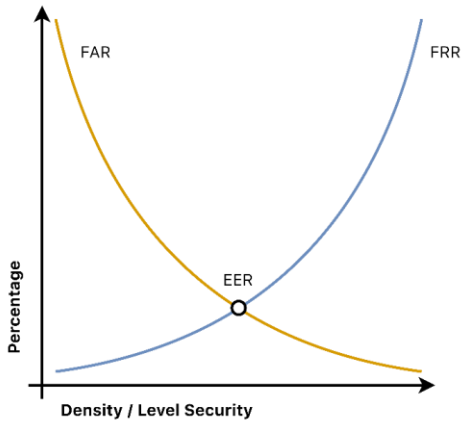
Nell'identificazione infatti possiamo trovarci di fronte a due situazioni:

1. riconoscimento positivo:
    a. la persona è vera
    b. la persona è un impostore;
2. riconoscimento negativo:
    a. il sistema o ha sbagliato dando un falso allarme
    b. la persona è realmente un impostore.

Di conseguenza abbiamo due tipologie di errore:

- FRR (False Rejection Rate) è la percentuale di falsi rifiuti, utenti autorizzati ma respinti per errore, in pratica il sistema non riesce a riconoscere le persone autorizzate.
- FAR (False Acceptance Rate) è la percentuale di false accettazioni, utenti non autorizzati ma accettati per errore, il sistema quindi accetta le persone che non sono autorizzate.

Qualunque sistema biometrico permette di aumentare e diminuire la sensibilità, regolando il rapporto tra i falsi rifiuti e le false accettazioni. Per comprendere meglio possiamo definire la variabile d (Density level security) come il grado di tolleranza del sistema. Se questo grado è basso si ha un numero elevato di false accettazioni, con un grado alto invece si ha un numero elevato di falsi rifiuti. EER(d) rappresenta il punto di equilibrio del sistema attraverso il quale è possibile regolare il rapporto FRR/FAR.



Per aumentare la sicurezza del sistema di riconoscimento si possono utilizzare più tecniche biometriche grazie ai sistemi multi modali. Questi sistemi permettono un riconoscimento più preciso e diminuiscono il failure-to-enroll rate, ovvero il tasso di errore

Autenticazione con password

L'autenticazione mediante password non riguarda solo le password in sé, ma può includere anche frasi di accesso (parole con eventuali spazi tra di esse) o un PIN (una sequenza di numeri). È il metodo di autenticazione più comune poiché è semplice: basta ricordarsi la password per accedere a qualsiasi sistema. Questo metodo è particolarmente adatto alle applicazioni basate sul web, che ormai sono predominanti. La sua convenienza deriva dalla sua universalità, diffusione e semplicità d'uso.

Tuttavia, presenta alcuni svantaggi:

- **Vulnerabilità allo sniffing e al replay**: le password possono essere intercettate e riutilizzate. Questo problema è intuitivo e può sembrare il principale, ma ci sono altre sfide da considerare.
- **Vulnerabilità ai cracking**: molte password sono troppo semplici e quindi vulnerabili ad attacchi di tipo cracking.
- **Vulnerabilità agli attacchi di guessing o dictionary**: i leak di dati di grandi aziende vengono utilizzati per creare enormi dizionari di password degli utenti.
- **Vulnerabilità al phishing**: gli attacchi di phishing mirano ad utenti inesperti o poco attenti.

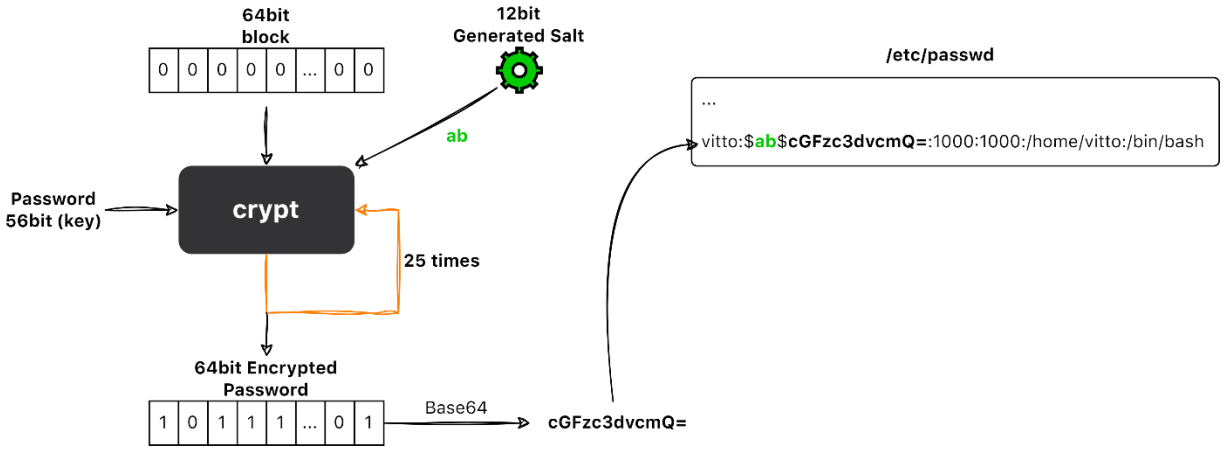
Il primo problema da affrontare è prevenire l'intercettazione delle credenziali durante la trasmissione sulla rete. Un altro problema è il cosiddetto session hijacking, che consiste nell'interferire con una sessione e dirottarla a un altro utente. È quindi necessario che la sessione dell'applicazione sia collegata all'autenticazione effettuata. Inoltre, le password non devono mai essere memorizzate in chiaro, né nel client né nel server.

UNIX password & Salt

L'implementazione originale della funzione di libreria crypt() della terza versione di Unix, imitava la macchina di cifratura M-209. Anziché crittografare la password con una chiave, che avrebbe consentito il recupero della password dal valore crittografato e dalla chiave, utilizzava la password stessa come chiave che veniva poi salvata in */etc/passwd*.

Lo schema di crittografia originale è risultato essere troppo blando e quindi soggetto a forza bruta delle password più probabili. Nella settima versione di Unix, lo schema è stato cambiato in una forma modificata dell'algoritmo DES. Inoltre, l'algoritmo incorporava un salt a 12 bit per garantire che un utente malintenzionato fosse costretto a decifrare ciascuna password in modo indipendente invece di poter prendere di mira l'intero database delle password contemporaneamente. In */etc/passwd* ci sono tante righe quanti sono gli utenti e ogni riga contiene: nome utente, il sale usato per l'utente che sono 12bit (6 bit per carattere) random generati quando si genera l'utente e la trasformazione della password dell'utente (gergo Unix si chiama encrypted password)

In dettaglio, dalla settima versione di Unix, la password dell'utente viene troncata a otto caratteri (se più piccola, paddata) e per ogni carattere, i 7 bit meno significativi vengono presi (8*7 = 56bit di chiave); questo costituisce la chiave DES a 56 bit. Quella chiave viene quindi utilizzata per crittografare un blocco di 64bit a zero, quindi il testo cifrato viene nuovamente crittografato con la stessa chiave e così via per un totale di 25 crittografie DES. Viene utilizzato un salt a 12 bit per perturbare l'algoritmo di crittografia (la S-Box viene alterata), quindi le implementazioni DES standard non possono essere utilizzate per implementare crypt().



L'ultimo blocco cifrato che si ottiene dopo tutte le iterazioni, veniva trasformato in 11 caratteri base64, quindi 64 bit in base64 sono 10 caratteri più padding (=). Il file */etc/passwd* era di sola lettura. Apparentemente una buona idea, da un punto di vista strettamente teorico la cosa non preoccupa più di tanto, in quando non ci sono le password in chiaro ma le password trasformate: la trasformazione non è invertibile, sarebbe invertibile se fosse un encription vera, perché con la password si potrebbe decifrare, ma in questo contesto non si sta cifrando la password con una chiave nota, ma si cifrano degli zeri con una chiave sconosciuta (password + salt). Quindi, senza conoscere la password in chiaro non riesco a risalire né agli zeri, tantomeno alla password che non conosco.

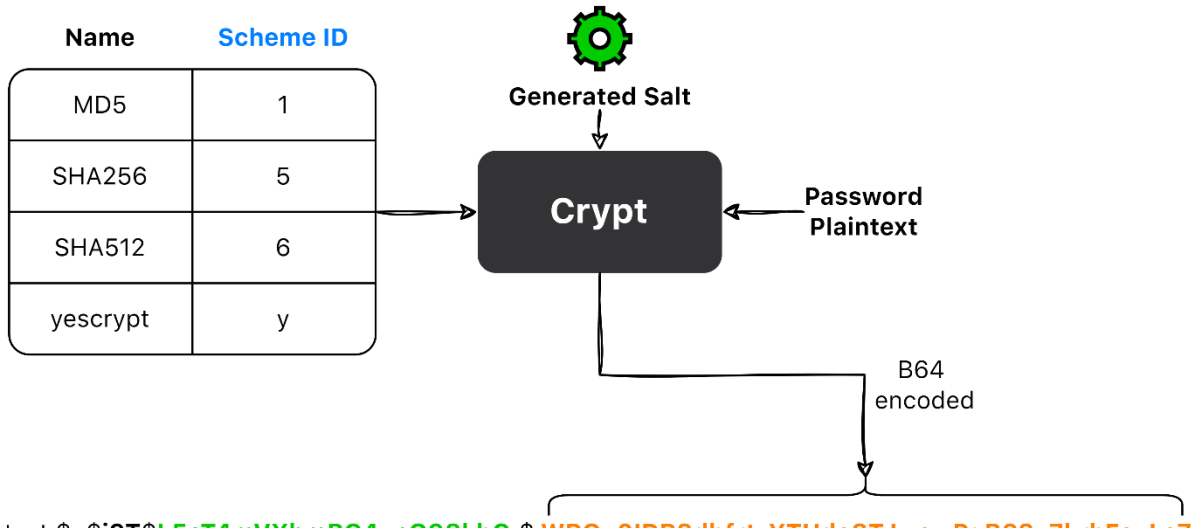
Il fatto di avere una S-box diversa rendeva più sensibile l'algoritmo. Tuttavia, l'uso del DES in questo contesto è comunque sicuro, in quanto il testo da cifrare è molto corto (un singolo blocco di 64 bit) e tecniche di analisi statistica non sono utilizzabili. Le vulnerabilità che si possono introdurre in questo modo sono tali quando il testo che offriamo è un testo molto lungo, dove è possibile fare crittoanalisi statistica, dato che qua stiamo cifrando un blocco di 64 bit, è ovvio che non è possibile fare le analisi statistiche, quindi possiamo ignorare questo problema.

Questo metodo, applicato alla versione 7, è stata una soluzione efficiente per oltre 30 anni. Nei tre decenni trascorsi da allora però, i computer sono diventati molto più potenti. La legge di Moore è generalmente valida, quindi la velocità e la capacità dei computer sono raddoppiate di oltre 20 volte da quando Unix è stato scritto per la prima volta. Ciò ha reso da tempo l'algoritmo basato su DES

vulnerabile agli attacchi di dizionario, cioè dei dizionari che sono stati costruiti negli anni che contengono le password più comuni o comunque grandi quantità di password che si sa essere utilizzate molto. Questi attacchi comunque sarebbero complessi con una password di 8 caratteri scelta in maniera casuale, il problema è che gli utenti usano password banali. Sono stati fatte delle ricerce e circa il 30% degli utenti poteva essere soggetto alla discovery della password con questa semplice tecnica.

Come soluzione a questo, da un lato c'è tutto un lavoro di educazione dell'utente, da un punto di vista invece sistemistico, quello che è stato fatto su Unix, è che si è migrati ad algoritmi come SHA2 ed i digest delle password non sono più su */etc/passwd* ma una parte di quell'informazione, in particolare "**H(password + salt)"** è stato spostato in */etc/shadow* (ombra) un file ombra del file delle password. Le informazioni sull'utente, attributi dell'utente, per esempio il suo UID, la Shell, rimangono in */etc/passwd*, perché il sistema ne ha bisogno *(/etc/shadow* è in lettura e scrittura solo per root e root group).

Per aumentare la sicurezza delle password, anche le versioni successive di Unix utilizzano il "sale". Il sale è una stringa casuale che viene aggiunta alla password prima di essere convertita in hash. Il risultato finale dell'hashing è quindi la concatenazione password ed il sale. Come schema di hashing, tra i più utilizzati nei sistemi moderni sono: SHA256, SHA512, yescrypt.

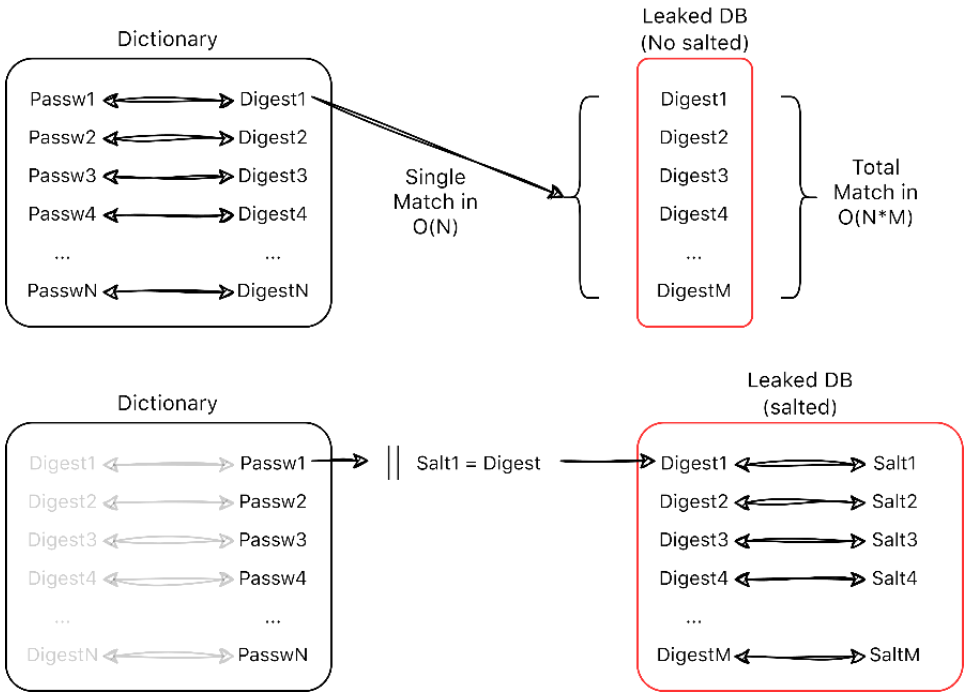


La codifica utilizzata è una variante di base64 che in Unix viene chiamata B64. La differenza è solo che nell'alfabeto B64 come caratteri speciali sono presenti il "." e "/". Nella codifica standard base64 come caratteri speciali ci sono "+" e "/".

I parametri non vengono utilizzati da tutti gli algoritmi, ad esempio, yescrypt li usa mentre SHA no.

L'utilizzo del sale aumenta la sicurezza delle password in due modi: in primo luogo, garantisce che due utenti con la stessa password avranno due hash differenti (anche se utilizzano la stessa password), poiché ogni utente avrà un sale unico. In secondo luogo, rende più difficile per un attaccante craccare le password utilizzando un attacco di dizionario.

Secondo le ultime OWASP list, l'assenza di salt è uno tra i principali problemi di sicurezza.



Authorization

La fase di autorizzazione, nel controllo di accesso è la funzione che specifica i diritti/privilegi di accesso alle risorse. L'autorizzazione quindi avviene dopo l'autenticazione (l'utente è ormai collegato, ma è necessario capire se l'utente può usare il servizio che sta richiedendo).

Da non confondete il ruolo con le autorizzazioni: è possibile avere un ruolo e quello magari non cambia mai, ma le autorizzazioni per quel ruolo possono cambiare.

La fase in cui nell'Idm vengono assegnate le autorizzazioni (o ruoli) all'utente è detta fase di Provisioning. Durante la fase di provisioning, vengono attribuiti al soggetto le autorizzazioni, i diritti di accesso e gli attributi necessari per utilizzare i servizi o le risorse offerte dal sistema. Ciò può includere l'assegnazione di ruoli, gruppi di appartenenza, autorizzazioni specifiche o altre configurazioni personalizzate. Il provisioning può essere automatizzato attraverso flussi di lavoro predefiniti o richiedere l'intervento di un amministratore del sistema per l'approvazione e la configurazione manuali.

Le autorizzazioni sono assegnate secondo un principio chiamato **principio del privilegio minimo** (**Principle of least privilege**). Questo principio richiede che, in un particolare livello di astrazione di un ambiente di calcolo, ogni modulo computazionale (che può essere ad esempio un processo, un programma o un utente a seconda del livello di astrazione considerato) abbia visibilità delle sole risorse/informazioni immediatamente necessarie al suo funzionamento. Ciò implica che gli utenti devono ottenere autorizzazioni specifiche per accedere a determinati dati o risorse, e queste autorizzazioni sono limitate alle necessità del loro lavoro. Ad esempio, un impiegato del reparto contabile potrebbe avere accesso solo ai file contabili, mentre non avrebbe accesso ai dati del personale o ad altre informazioni riservate.

Implementare il concetto di privilegio minimo richiede una corretta gestione degli accessi e delle autorizzazioni, insieme a una politica di sicurezza solida. Inoltre, l'uso di controlli tecnologici come

l'accesso basato sui ruoli (RBAC) e la separazione dei compiti può aiutare a garantire che gli utenti abbiano solo i privilegi necessari per svolgere le loro funzioni specifiche.

Approcci di Access Control

Ci sono diversi approcci per l'access control:

- DAC: Discretionary Access Control
- MAC: Mandatory Access Control
- RBAC: Role-based Access Control
- ABAC: Attribute-based Access Control

Stiamo parlando di fatto, di uno scenario di sicurezza del file system, però questi stessi concetti è possibile applicarli a qualunque tipo di risorsa, non soltanto file.

Discretionary access control

I controlli sono discrezionali nel senso che un soggetto con un certo permesso di accesso è in grado di passare tale permesso (magari indirettamente) a qualsiasi altro soggetto (a meno che non sia vincolato da un Mandatory Access Control).

Il DAC è comunemente discusso in contrasto con il Mandaroy Access Control (MAC). Occasionalmente, si dice che un sistema nel suo insieme abbia un controllo degli accessi "discrezionale" o "puramente discrezionale" quando quel sistema non dispone di MAC.

D'altra parte, i sistemi possono implementare sia MAC che DAC contemporaneamente, dove DAC si riferisce a una categoria di controlli di accesso che i soggetti possono trasferire tra loro, e MAC si riferisce a una seconda categoria di controlli di accesso che impone vincoli sulla prima.

Esistono almeno due implementazioni: con Owner (diffuso) e con capabilites.

DAC with Owner

Il termine DAC è comunemente usato in contesti in cui ogni oggetto ha un proprietario (Owner) che controlla le autorizzazioni per accedere all'oggetto.

Gli utenti Owner hanno la possibilità di prendere decisioni sulle politiche e/o assegnare attributi di sicurezza. Un semplice esempio è Unix che rappresenta scrittura, lettura ed esecuzione in ciascuno dei 3 bit per ciascuno di Utente, Gruppo e Altri. (È preceduto da un altro bit che indica caratteristiche aggiuntive). Ed è proprio l'owner a decidere a chi assegnare questi attributi.

DAC with Capabilites

In questo approccio, invece di basarsi su un proprietario predefinito, ogni oggetto dati è associato a una "capability" che rappresenta un token o un certificato che permette l'accesso a quell'oggetto specifico. Una capability può essere concessa da un soggetto autorizzato e successivamente utilizzata da altri soggetti per accedere all'oggetto. Solo i soggetti in possesso di una capability valida possono accedere all'oggetto associato ad essa.

ACL

Le ACL sono una forma comune di implementazione per il controllo degli accessi in un sistema DAC. Una ACL è una lista di controlli di accesso associati a una risorsa specifica quindi in qualche modo memorizzata insieme alla risorsa stessa, come un file o una cartella. Questa lista specifica quali

utenti o gruppi di utenti hanno il permesso di accedere alla risorsa e quali operazioni possono eseguire su di essa, come leggere, scrivere o eseguire.

In un sistema DAC con ACL, ogni risorsa ha la sua ACL associata. Quando un utente richiede l'accesso a una risorsa, il sistema verifica l'ACL per determinare se l'utente ha i privilegi di accesso necessari. Se l'utente è presente nella lista degli utenti autorizzati o nel gruppo autorizzato nell'ACL e ha i permessi appropriati, gli verrà consentito l'accesso alla risorsa.

Le ACL offrono una granularità di controllo degli accessi, consentendo ai proprietari della risorsa di specificare in dettaglio chi può accedere alla risorsa e quali operazioni possono svolgere. Tuttavia, possono diventare complesse da gestire in ambienti con un numero elevato di utenti e risorse, poiché richiedono una gestione individuale per ogni risorsa.

Role Based Access Control

Il Role-Based Access Control (RBAC) è un modello di controllo degli accessi che si basa sull'assegnazione di ruoli agli utenti all'interno di un sistema. Invece di concedere o revocare i privilegi di accesso individualmente per ogni utente, RBAC assegna privilegi ai ruoli e quindi assegna i ruoli agli utenti. Quindi si sposta il focus sull'utente invece che sulla risorsa.

Nel modello RBAC sono presenti tre concetti principali:

- **Ruoli**: I ruoli rappresentano le responsabilità o le funzioni all'interno di un sistema. Ad esempio, potrebbero essere definiti ruoli come "amministratore", "utente normale" o "moderatore". I ruoli sono creati in base alle necessità del sistema.
- **Autorizzazioni**: Le autorizzazioni definiscono le attività o le operazioni che possono essere eseguite all'interno del sistema. Ad esempio, le autorizzazioni possono includere "lettura dei file", "scrittura dei file" o "eliminazione degli utenti".
- **Assegnazione dei ruoli**: I ruoli vengono assegnati agli utenti in base alle loro responsabilità o funzioni all'interno dell'organizzazione. Un utente può avere uno o più ruoli assegnati. Ad esempio, un utente potrebbe essere assegnato al ruolo di "amministratore" e al ruolo di "utente normale".

L'idea di base di RBAC è che gli utenti acquisiscono i privilegi di accesso attraverso i ruoli loro assegnati. Ciò semplifica la gestione degli accessi, in quanto è possibile modificare i privilegi di accesso modificando i ruoli assegnati agli utenti anziché modificare gli accessi di ogni singolo utente. Inoltre, RBAC favorisce la scalabilità e la manutenibilità del sistema, in quanto i ruoli possono essere definiti in modo centralizzato e applicati a diverse risorse e applicazioni all'interno dell'organizzazione.

Possono essere applicati anche vincoli aggiuntivi e i ruoli possono essere combinati in una gerarchia in cui i ruoli di livello superiore incorporano le autorizzazioni possedute dai ruoli secondari.

Il modo più universale di implementare RBAC è con una matrice d'accesso in cui sulle righe sono presenti le risorse e sulle colonne gli utenti (o viceversa). Ogni intersezione righa-colonna descrive che cosa può fare quell'utente su quella risorsa. Le righe, quindi le risorse, possono essere dei file, potrebbero essere delle stampanti, potrebbero essere delle risorse online, una directory, potrebbe essere qualunque cosa. Se stiamo parlando di file, per esempio, i permessi possono essere scrittura, lettura ed esecuzione, attraversamento (nel caso di una directory).

I sistemi operativi, non usano spesso una struttura così perché porterebbe via molto spazio e sarebbe scomoda da gestire. Infatti RBAC è più una policy per il controllo d’accesso utilizzato in contesti aziendali.

UNIX DAC & ACL

UNIX utilizza un sistema di access control basato principalmente su DAC.

Nel sistema di access control basato su DAC di UNIX, i permessi di accesso ai file e alle risorse sono assegnati a livello di proprietario, gruppo e altri utenti. Questo significa che il proprietario di un file ha il controllo completo sui permessi di accesso ad esso, inclusi i permessi di lettura, scrittura ed esecuzione. Il gruppo a cui appartiene il file ha un secondo livello di accesso, mentre tutti gli altri utenti del sistema hanno un terzo livello di accesso.

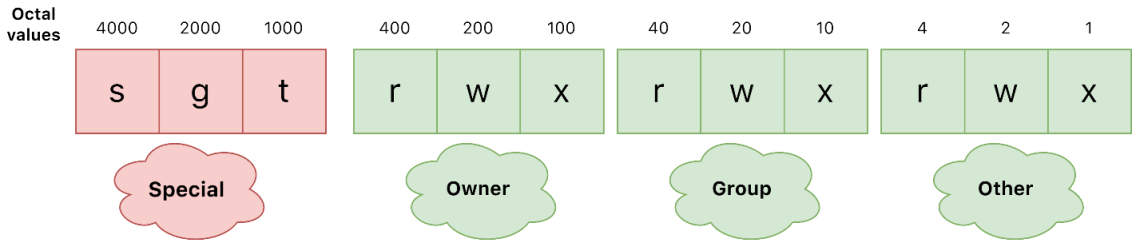
In alcuni casi, UNIX può implementare elementi di RBAC. Ad esempio, il concetto di "superuser" (solitamente l'utente root) in UNIX ha un controllo completo sul sistema e può accedere a qualsiasi file o risorsa, ignorando i permessi di accesso DAC. Questo potrebbe essere visto come un'implementazione di RBAC, in cui il superuser ha un ruolo privilegiato con accesso a tutti i livelli di risorse del sistema. Tuttavia, è importante notare che l'implementazione di RBAC in UNIX è limitata e non così sofisticata come nei moderni sistemi che si concentrano esplicitamente su RBAC come modello di access control principale.

Unix innanzitutto non elenca tutti gli accessi per tutti gli utenti, cioè non non ha una Matrice di accesso e non prevede permessi differenziati per singolo utente. Gli utenti vengono divisi in tre categorie, ogni risorsa ha:

- un proprietario (owner)
- Il gruppo del proprietario (group), perché gli utenti su unix vengono accorpati in gruppi (questo deriva un po dalla cultura di unix che è una cultura università, dove c'erano gruppi di persone che collaboravano insieme).
- Tutti gli altri utenti del sistema (other)

Quindi ogni risorsa avrà certi permessi d'accesso per ognuno di queste categorie, in particolare: lettura, scrittura ed esecuzione. Oltre ai classici permessi, in unix ci sono i cosiddetti permessi speciali:

- Setuid: Permette di dare permessi più raffinati, con una grana più fine.
- Setgid:
- Sticky bit:



Qual è il limite di questo controllo di accesso? È che, come in tutte le cose discrezionali, tutte le cose dove il programmatore può scegliere quello che gli pare, si possono fare degli errori, ma soprattutto si possono fare delle dimenticanze, per esempio si hanno delle informazioni riservate e ci si dimentica di proteggerle perché è a discrezione e non è obbligatorio.

Oltre a questa struttura base, è possibile estendere i permessi con delle ACL.

Setuid

Setuid è un meccanismo fondamentale in unix che permette di raffinare il sistema di controllo d'accesso (andando a realizzare un controllo di accesso fine grained), quindi, fa parte Della Identity and Access management di Unix. Senza il setuid non sarebbe possibile ad esempio permettere a un utente di cambiarsi la password.

Il problema è che questi programmi, hanno delle superfici di attacco molto più importanti. Essenzialmente in tre modi è possibile eseguire un exploit attraverso programmi setuid:

- Attraverso l'input dell'utente: Priviledge excalation con Attacco di buffer overflow.
- Variabili di ambiente
- Librerie statiche

Mandatory Access Control

A differenza del modello di controllo degli accessi basato su DAC, in cui i proprietari dei file hanno il controllo discrezionale sulla concessione dei permessi, il MAC è un modello di controllo degli accessi basato su regole fisse e obbligatorie. Nel modello MAC, il controllo degli accessi è determinato da politiche di sicurezza definite centralmente (di solito da un amministratore di sistema), che specificano quali soggetti (utenti, processi) possono accedere a quali oggetti (file, risorse di sistema) e con quali privilegi. Queste politiche sono applicate in modo coerente a tutti i soggetti e oggetti nel sistema.

Il MAC implementa solitamente una struttura di livelli di sicurezza (security levels) o etichette di sicurezza (security labels) che vengono associati ai soggetti e agli oggetti. Queste etichette rappresentano il livello di sicurezza o la classificazione dell'oggetto e del soggetto e vengono utilizzate per verificare l'accesso in base alle regole di sicurezza stabilite. Ad esempio, si può immaginare di avere:

1. **Un livello non classificato**: dove tutti possono accedere, quindi aperto anche ad utenti esterni.
2. **Un livello di autorizzazione minima**: per cui gli utenti esterni non identificati non possono accedere, ma tutti gli utenti del sistema che sono stati autenticati possono accedere.
3. **Un livello secret**: dove solo personale dello stesso livello o più alto può accedere
4. **Un livello top secret**: dove solo poche persone nell'organizzazione hanno permesso di attendere.

Questo riflette molto le gerarchie militari e degli enti governativi. Infatti, soprattutto nelle nelle applicazioni governative, ha avuto molta importanza soprattutto negli anni passati questo concetto di sicurezza obbligatoria.

Modello di bell-lapadula

Uno dei modelli MAC, è quello definito tra il 1973 e il 1976 da David Elliott Bell e Len LaPadula. Il modello si concentra sulla Confidentiality (Riservatezza) dei dati su un sistema Multilivello.

Questo modello sintetizza delle regole tra livelli per il mantenimento della riservatezza:

- La **Simple Security Property** (No-Read-Up): afferma che gli utenti ad un certo livello vedono tutte le risorse al loro livello di riservatezza (e più bassi), ma non possono vedere o modificare le risorse che stanno ad un livello di sicurezza superiore.
- La *** (Star) Security Property** (No-Write-Down): meno intuitiva, ma egualmente importante, con cui si vuole impedire a chi sta più in alto di scrivere verso il basso, perché se si scrive verso il basso, si va a fare Information leaking, dare informazioni che non si dovrebbe dare

verso livelli in basso che non dovrebbero vedere quelle informazioni. È possibile che succeda consciamente, perché un utente è distratto o anche inconsciamente, perché c'è un qualche malware che agisce, per esempio, ci può essere il classico Trojan Horse eseguibile.

Il brutto da un punto di vista proprio molto pratico è che i sistemi commerciali questo meccanismo non lo adottano: Unix non lo adotta, ma nemmeno Windows, e in un mondo moderno dove tutto si deve integrare con tutto, non hanno fatto moltissima strada.

Capability leaking

Una capability è definita come un riferimento protetto ad un oggetto la quale garantisce al processo la possibilità (in inglese appunto *capability*) di interagire con l'oggetto in determinati modi. Questi includono leggere il dato associato all'oggetto, modificare l'oggetto, eseguire il dato dell'oggetto come processo e altri possibili permessi di accesso. La capability consiste in un riferimento che identifica un oggetto particolare e un insieme di uno o più diritti.

Quando si ha una capability leaking, alcuni oggetti a questi riferimenti protetti sono andati persi oppure lasciati incustoditi, se un malintenzionato li trova o si accorge che ne esistono, potrebbe usarli a suo piacere per accedere a risorse protette.

Questo codice mostra un esempio di capability leaking.

```c
// cap_leak.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void main()
{
   int fd;  char *v[2];
   /* Assume that /etc/zzz is an important system file, and it is owned by root with permission 0644. Before running this
program, you should create the file /etc/zzz first. */

   fd = open("/etc/zzz", O_RDWR | O_APPEND);

   // Print out the file descriptor value
   printf("fd is %d\n", fd);
   if (fd == -1)
   {
      printf("Cannot open /etc/zzz\n");
      exit(0);
   }
   // Permanently disable the privilege by making the
   // effective uid the same as the real uid
   setuid(getuid());
   printf("Real user id is %d\n", getuid());
   printf("Effective user id is %d\n", geteuid());

   // Execute /bin/sh
   v[0] = "/bin/sh"; v[1] = 0;
   execve(v[0], v, 0);
}
```

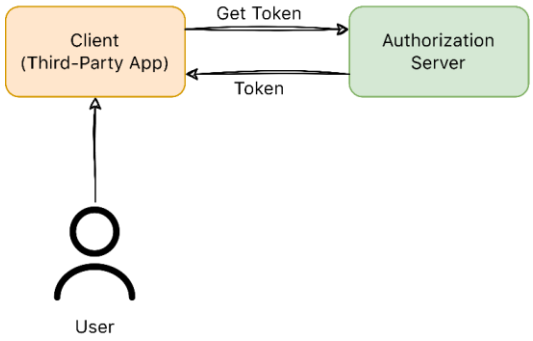
In questo esempio, il leaking è dovuto al fatto che il programmatore non ha chiuso il file descriptor del file protetto. In questo modo è possibile da una qualunque shell utilizzare il file descriptor per leggere quel file.

OAUTH

OAuth (Open Authorization, RFC 6749) è un protocollo di autorizzazione standardizzato e aperto che consente a un utente di concedere a un'applicazione terza, nota come client, l'accesso sicuro alle sue risorse o informazioni, ospitate da un'altra applicazione o servizio, conosciuto come server.

L'obiettivo principale di OAuth è fornire un meccanismo sicuro per l'autorizzazione dei client a ottenere l'accesso alle risorse protette, senza richiedere all'utente di condividere le proprie credenziali di accesso, come username e password. OAuth utilizza un sistema di token di accesso che consente al client di agire a nome dell'utente autenticato, limitando l'accesso solo alle risorse specifiche per le quali l'utente ha concesso l'autorizzazione.

Il processo di autorizzazione di OAuth coinvolge tre parti principali: l'utente, il client e il server di autorizzazione. Il server di autorizzazione gestisce l'autenticazione dell'utente e l'autorizzazione del client, generando i token di accesso necessari per consentire al client di accedere alle risorse.



Il protocollo OAuth si basa su richieste HTTP standardizzate e utilizza un flusso di autorizzazione basato su token per facilitare l'interazione tra le parti coinvolte. I token di accesso generati sono temporanei e limitati nel tempo, il che significa che hanno una durata limitata e devono essere rinnovati periodicamente. Ciò fornisce un ulteriore livello di sicurezza, in quanto i token scaduti non possono essere utilizzati per accedere alle risorse protette.

OAuth è ampiamente utilizzato nell'ambito delle applicazioni web e dei servizi API, consentendo agli utenti di condividere le loro risorse con applicazioni di terze parti in modo sicuro e controllato. Ad esempio, quando si concede ad una WebApp di salvare o leggere dei contenuto salvati sul Drive Google. Oppure ad un'app di messaggistica di accedere alla lista dei contatti sul Google Account.

Client ID & Secret Client

I client che necessitano di utilizzare il servizio di autorizzazione devono essenzialmente prima registrarsi presso il server di autorizzazione. Il server di autorizzazione, in seguito alla registrazione del client, rilascera due parametri:

- Client_id
- Client_secret

Il **client_id** e il **client_secret** sono due parametri utilizzati nel contesto di OAuth per identificare e autenticare un'applicazione client nei confronti del server di autorizzazione.

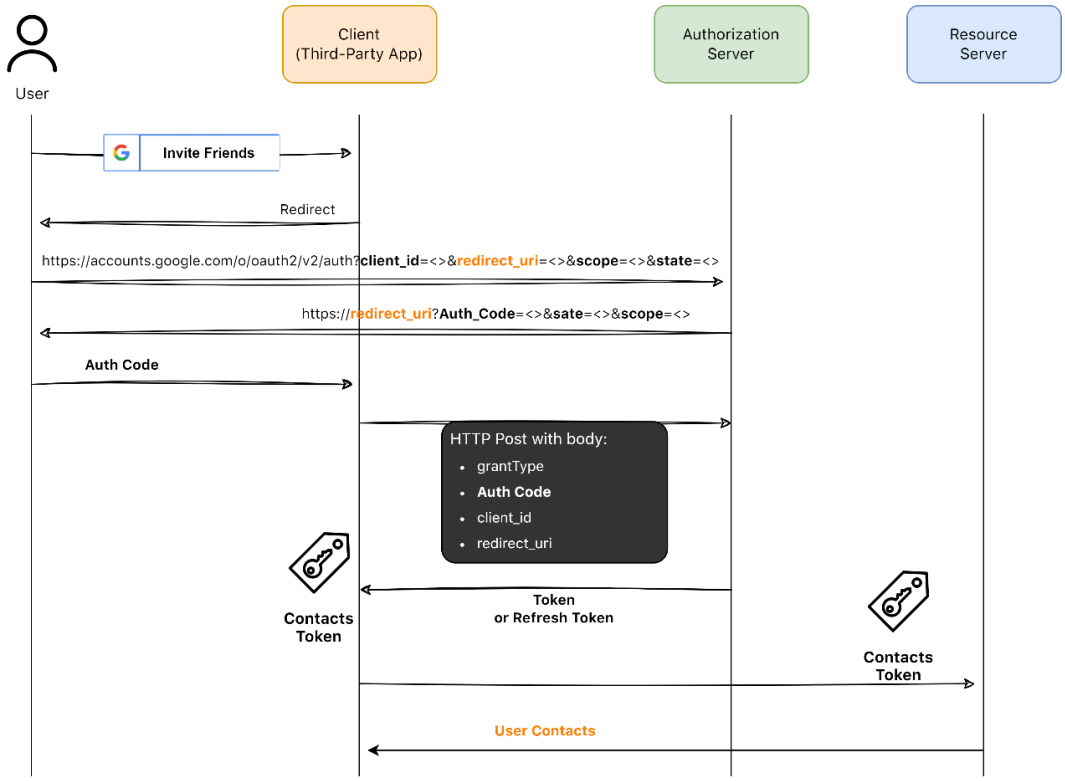
Il **client_id** serve ad identificare in modo univoco l'applicazione client durante il flusso di autorizzazione. Quando l'applicazione client invia una richiesta al server di autorizzazione, include il

client_id come parte della richiesta in modo che il server possa riconoscere quale applicazione client sta facendo la richiesta. Il **client_secret**, invece, è un segreto crittografico associato all'applicazione client. È un valore riservato e confidenziale che deve essere mantenuto segreto tra l'applicazione client e il server di autorizzazione. Il client_secret viene utilizzato per autenticare l'applicazione client durante il flusso di autorizzazione, garantendo che solo l'applicazione client autorizzata possa ottenere l'accesso alle risorse protette. Solitamente, il client_secret viene utilizzato in combinazione con il client_id per verificare l'identità dell'applicazione client durante le richieste di token di accesso.

Supponiamo di avere un client (applicazione web) che richiede di accedere ai contatti google dell'utente (magari per invitare gli amici nell'applicazione). Prima di tutto è necessario che il client (nello scenario OAUTH il client è l'App terza che necessita dei dati dell'utente) si registri presso il server di autorizzazione di Google:

Authorization Code Grant Flow

Una volta che il client ha ottenuto i propri client_id e client_secret, è abilitato a fare richiesta di Token presso l'Authorization Server secondo quello che viene chiamato l'Authorization Code Grant Flow. Durante questo flusso, il client ed il server di autorizzazione si scambiano messaggi per permettere al client di ottenere un token di autorizzazione.

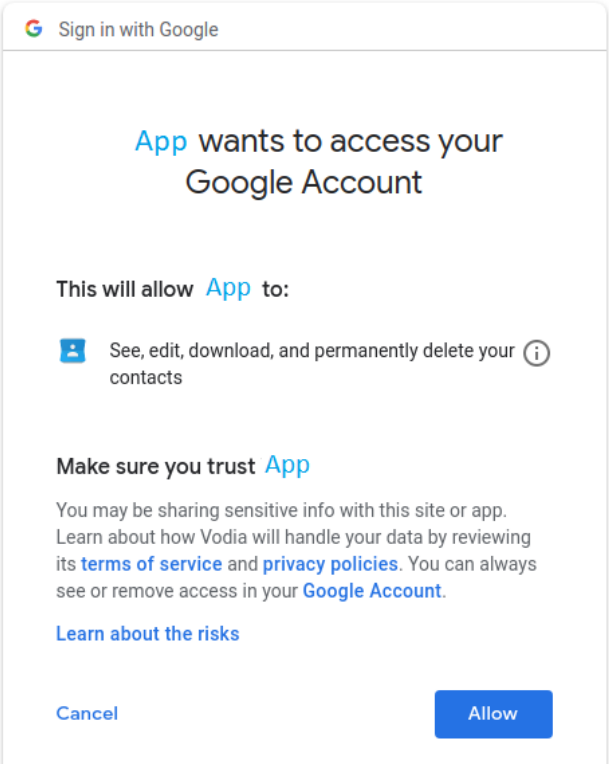


Una volta ottenuto il token, il client può usarlo per accedere ai contatti dell'utente. Tutto questo senza che l'utente abbia inserito le credenziali sul client.

Quando l'utente clicca su "Invite Friends", viene reindirizzato al Server di autorizzazione di google, insieme ad alcuni parametri nella query string, tra i più importanti:

- **redirect_uri**: questo parametro serve all'Authentication server per rieffettuare il redirect verso il client
- **client_id**: il client_id del client da cui proviene l'utente (dal sito)
- **scope**: Una o più stringhe separate da spazi che indicano le autorizzazioni richieste dal client, ad esempio (nel nostro esempio magari, scope=contacts)
- **state**: Una stringa casuale che il client genera e che include nella richiesta

Una volta effettuato il login, l'utente vedrà (nel caso di google) la finestra che indica quali dati il client utilizzerà, ad esempio:



In sostanza la lista degli scope indicati dall'Url, nel campo "scope". Se l'utente acconsente, il Server di autorizzazione di Google utilizzerà l'URL nelc ampo "redirect_uri", per inoltrare la risposta al Client contenente l'Authorization Code e altri parametri utili.

Il client a questo punto, di nuovo utilizzando il server di autorizzazione, inoltra una richiesta POST HTTP con questi parametri:

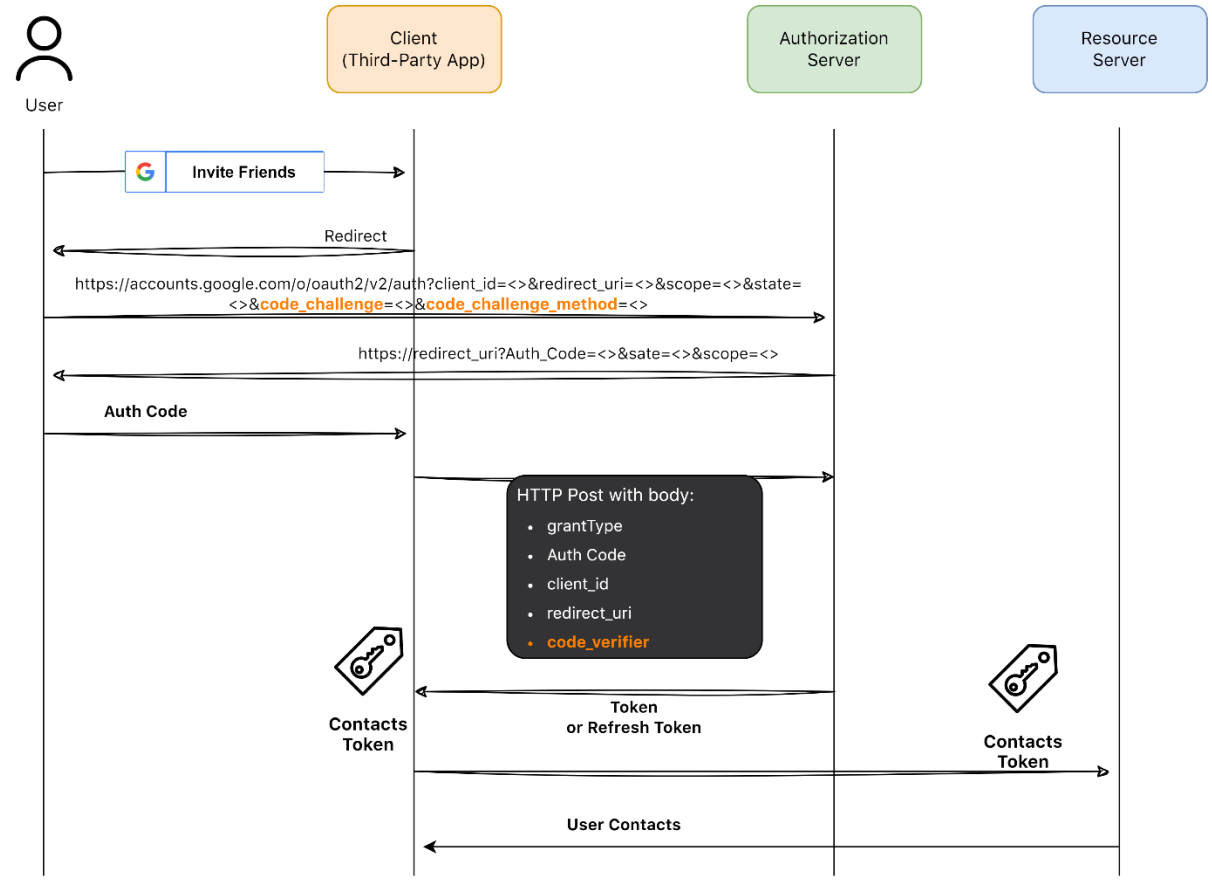
- **grant_type=authorization_code**
- **code**: il codice di autenticazione ricevuto in precedenza
- **redirect_uri**: Lo stesso URI di reindirizzamento utilizzato durante la richiesta del codice di autenticazione
- **client_id**: client_id

A quel punto, il server di autorizzazione se controllando l'Auth Code si accroge che è valido, ritornerà un token che il client potrà usare accedere alle risorse dell'utente.

Proof Key for Code Exchange (PKCE)

Ai fini di evitare che Estensioni del Browser dannose possano effettuare attacchi su OAuth2, è necessario estendere OAuth2 con PKCE (RFC 7636). Infatti, nello schema di prima, se immaginiamo che nel browser dell'utente ci sia un'estensione che ha accesso alle URL che vengono utilizzate, è possibile recuperare l'Authorization Code per poi usarlo con l'Authorization Server e ricevere un Token di autorizzazione.

Per ovviare a questo, bisogna aggiungere dei meccanismi di integrità:



Nel redirect, il Client inserisce due parametri nuovi:

- **code_challenge**: un digest generato dal client
- **code_challenge_method**: l'algoritmo utilizzato per creare il digest

Quando invece il Client richiede il Token, aggiunge un nuovo parametro al Body HTTP POST, che è il **code_verifier**.

In sostanza, il Client, all'inizio inoltra un code_challenge digest utilizzando code_challenge_method. Nella richeista POST successiva, il Client inserisce nel body anche il plaintext che produrrà quel digest (solo il Client conosce il plaintext che produce quel digest). Il server di autorizzazione (avendo salvato il code_challenge precedentemente inviatogli dal Client), quando estrae i dati dalla

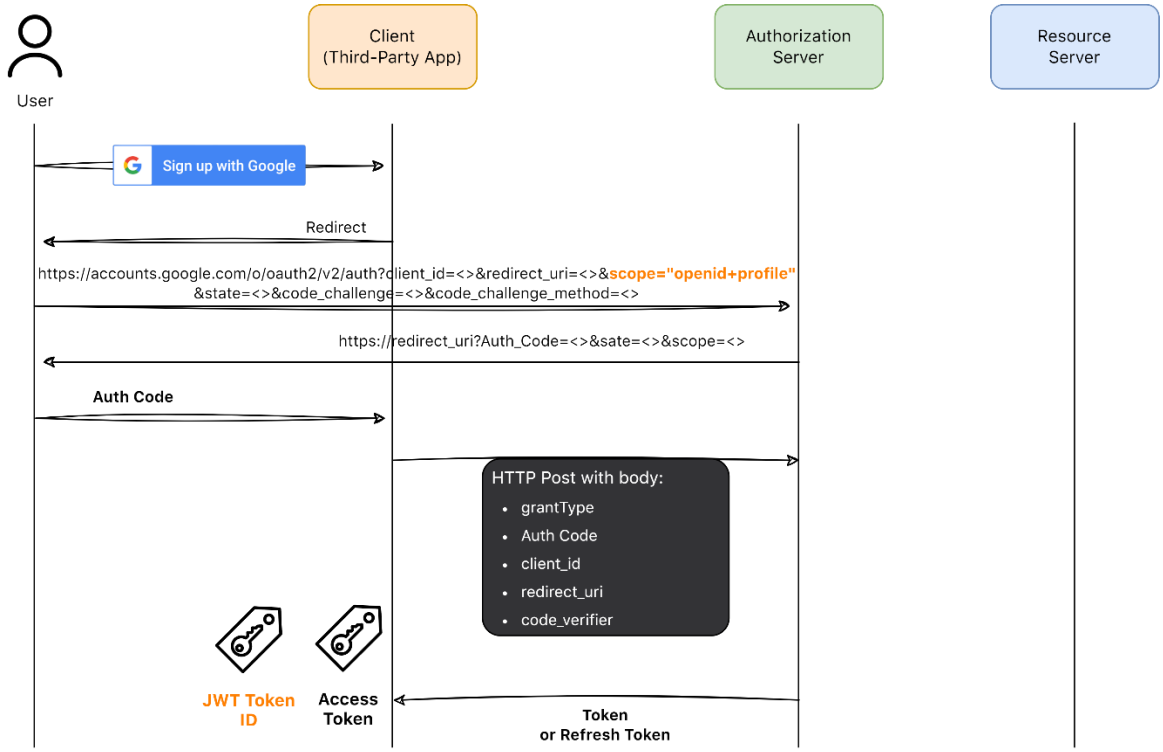
richiesta HTTP POST, utilizza il code_verifier per creare un digest con il code_challenge_method. Se i due digest combaciano, allora il Server sa che è il Client ad inoltrare la richiesta di Token.

OPENID Connect

OpenID Connect (OIDC) è un framework di autenticazione e autorizzazione basato su scambio di JSON Web Token (JWT) e costruito sul protocollo OAuth 2.0. In sostanza, OpenID Connect (OIDC) estende OAuth 2.0 introducendo la funzionalità di autenticazione, aggiungendo un flusso di autenticazione all'interno del processo. Consente ai clienti di verificare l'identità dell'utente finale in base all'autenticazione eseguita da un server di autorizzazione, nonché di ottenere informazioni di base sul profilo dell'utente finale.

OAuth 2.0, da solo, fornisce solo un meccanismo di autorizzazione per accedere alle risorse protette di un utente. OpenID Connect utilizza i flussi di autorizzazione di OAuth 2.0 per autenticare l'utente e fornire un'identità verificata tramite un token di identificazione.

Se ad esempio un client che utilizza OID Connect vuole effettuare la registrazione utente utilizzando le API di Google (il flusso è identico, cambiano gli scope):

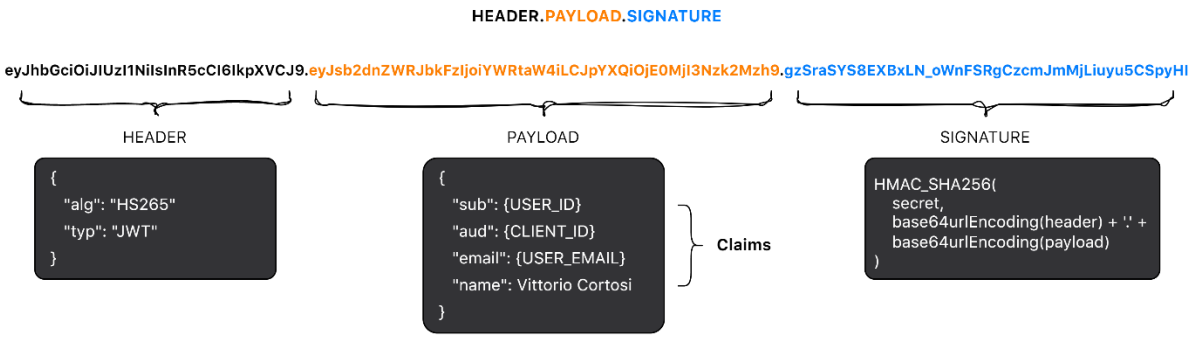


Integrando un flusso di autenticazione nel granting flow di OAuth2, è necessario indicare come scope “openid+profile”. In questo modo il server di autorizzazione di Google oltre a ritornare un Access Token, ritornerà anche un Token ID JWT contenente le informazioni riguardante l’utente che si sta autenticando.

JWT

JSON Web Token (JWT , pronunciato "jot") è uno standard aperto (RFC 7519) che definisce un modo compatto e autonomo per la trasmissione sicura di informazioni tra le parti federate sottoforma di oggetto JSON.

Il Token ID JWT ha una struttura particolare: è codificato in base64 ed ha la seguente forma:



Formato quindi da un **header** che Identifica l'algoritmo utilizzato per generare la firma (HS256 indica che il token è firmato utilizzando HMAC-SHA256). Gli algoritmi crittografici tipici utilizzati sono HMAC con SHA-256 (HS256) e la firma RSA con SHA-256 (RS256). JWA (JSON Web Algorithms) RFC 7518 ne introduce molti altri sia per l'autenticazione che per la crittografia.

Successivamente da un **payload** Contiene una serie di attributi che vengono chiamati **claims**. La specifica JWT definisce sette claims di attestazione registrati che sono i campi standard comunemente inclusi nei token. È possibile inserire anche **claims** personalizzati, a seconda dello scopo del token.

Segue la **signature** (firma) del token che convalida in modo sicuro il token. La firma viene calcolata codificando l'intestazione e il payload utilizzando Base64 e concatenando i due insieme con un separatore di punto. La firma quindi viene controllata attraverso l'algoritmo crittografico specificato nell'header del JWT.

Quando poi il client desidera accedere ad una risorsa protetta, deve inviare il JWT, in genere nell'header Authorization HTTP utilizzando lo Bearer schema. Il contenuto dell'intestazione potrebbe essere simile al seguente:

Authorization: Bearer eyJhbGci .. yu5CspyHI

Si tratta di un meccanismo di autenticazione senza stato poiché lo stato dell'utente non viene mai salvato nella memoria del server. Il server quindi verificherà che il JWT sia valido prendendolo nell'intestazione di autorizzazione e, se valido, l'utente potrà accedere alle risorse protette.

SSO

Single sign-on (SSO) è uno schema di autenticazione che consente a un utente di accedere con un singolo ID a uno qualsiasi dei numerosi sistemi software correlati, ma indipendenti. Consente quindi all'utente di accedere una volta e accedere ai servizi senza dover reinserire i fattori di autenticazione.

È possibile ottenere una versione semplice di Single Sign-On su reti IP utilizzando i cookie, ma solo se i siti condividono un dominio principale DNS comune.

Al contrario, **single sign-off** o **single log-out** (**SLO**) è la proprietà per cui una singola azione di disconnessione termina l'accesso a più sistemi software.

Con l'utilizzo dell'SSO si riduce quello che viene chiamato "affaticamento da passwrod": è la sensazione provata da molte persone a cui è richiesto di ricordare un numero eccessivo di password.

Sono stati sviluppati diversi **Protocolli di autenticazione** che permettono di costruire un sistem SSO. Alcuni dei protocolli più comuni includono:

- **SAML**
- OpenID Connect
- Kerberos

SAML

Security Assertion Markup Language (SAML, RFC 7522) è uno standard per lo scambio di dati di autenticazione e autorizzazione tra parti, in particolare tra un IdP (identity provider) ed un SP (service provider). Lo scambio di dati (tra IdP ed SP) di autenticazione avviene attraverso XML.

Ovviamente lo scambio di dati di autenticazione è alla base del SSO, infatti, un caso d'uso importante che SAML indirizza è il single sign-on (SSO) del browser web. Il Single Sign-On è relativamente facile da realizzare all'interno di un dominio (utilizzando i cookie , ad esempio), ma l'estensione dell'SSO tra i domini differenti è più difficile. SAML è stato specificato e standardizzato per promuovere l'interoperabilità tra domini.

La specifica SAML definisce tre ruoli:

- Principal (in genere un utente umano)
- Identity Provider (IdP)
- Service Provider (SP).

Nel caso d'uso principale affrontato da SAML, un utente richiede un servizio dal Service provider. Il Service provider richiede e ottiene un Authentication Assertion dall'Identity Provider. Sulla base di questa assertion, il Service Provider può prendere una decisione di controllo degli accessi, ovvero può decidere se eseguire il servizio per l'utente connesso. Prima di consegnare l'assertion dall'IdP all'SP, l'IdP può richiedere alcune informazioni al Principal (utente), come un nome utente e una password, per autenticarlo.

Al centro dell'asserzione SAML c'è un soggetto (un principale nel contesto di un particolare dominio di sicurezza) su cui qualcosa viene asserito. Il soggetto è solitamente (ma non necessariamente) un essere umano. Come nella panoramica tecnica di SAML 2.0, i termini soggetto e principale sono usati in modo intercambiabile in questo documento.

Nella specifica, SAML non indica il metodo di autenticazione presso il provider di identità. L'IdP può utilizzare un nome utente e una password o qualche altra forma di autenticazione, inclusa l'autenticazione a più fattori. Ad esempio, un servizio di directory come LDAP o Active Directory che consente agli utenti di accedere con un nome utente e una password, è una tipica fonte di token di autenticazione presso un IdP.

SAML Metadata

Affinchè l'IdP e l'SP si possano parlare reciprocamente durante lo scambio di messaggi, è necessaio che entrambi conoscano ad esempio la chiave pubblica dell'altra parte, oppure sapere chi è l'altra parte (con un ID). Questo lavoro viene risolto scambiandosi un file chiamato metadata. Ovviamente lo scambio deve avvenire in modo sicuro su un canale sicuro.

Il file metadata definisce quindi i dettagli tecnici e le configurazioni di sicurezza necessarie per l'interoperabilità tra l'IdP e l'SP. Contiene informazioni come l'URL dell'IdP o dell'SP, le chiavi utilizzate per garantire l'integrità e la riservatezza delle comunicazioni, le regole di autorizzazione, i tipi di attributi supportati e altre informazioni rilevanti per l'autenticazione federata.

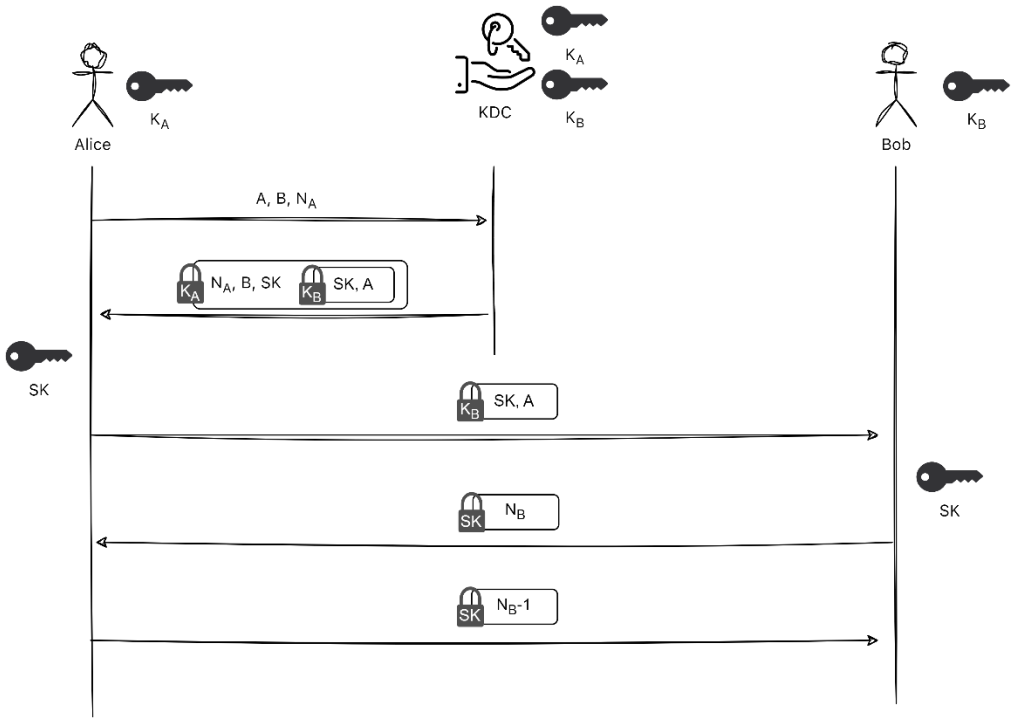
L'uso dei metadata semplifica l'integrazione tra i diversi sistemi SAML, poiché le informazioni di configurazione necessarie sono raccolte in un unico documento standard.

Protocollo di Needham-Schroeder

Il protocollo Needham-Schroeder è uno dei due principali protocolli di trasporto destinati all'uso su una rete non sicura, entrambi proposti da Roger Needham e Michael Schroeder. Questi sono:

- **Il protocollo a chiave simmetrica Needham-Schroeder**: basato su un algoritmo di crittografia simmetrico. Costituisce la base del protocollo Kerberos. Questo protocollo mira a stabilire una chiave di sessione tra due parti su una rete.
- **Il protocollo a chiave pubblica Needham-Schroeder**: basato sulla crittografia a chiave pubblica. Questo protocollo ha lo scopo di fornire l'autenticazione reciproca tra due parti che comunicano su una rete, ma nella sua forma proposta non è sicuro.

Guarderemo solo il protocollo simmetrico:



Il protocollo funziona così:

1. Alice spedisce un messaggio al server con la sua identità e quella di Bob, per dichiarare che intende comunicare con Bob. Allega anche un nonce $\mathbf{N_A}$
2. Il server genera la chiave di sessione **SK** (Session Key) e risponde ad Alice inviandole:
    a. la chiave di sessione **SK** generata e l'identificativo di B
    b. il nonce $\mathbf{N_A}$ che assicura ad Alice che il messaggio è nuovo e che il server sta rispondendo a quel particolare messaggio.
    c. la coppia **(SK, A)** criptata con la chiave $\mathbf{K_B}$, in modo che possa essere inoltrata a Bob perché venga reso partecipe
    d. Il tutto è cifrato con la chiave $\mathbf{K_A}$
3. Alice comunica a Bob la chiave di sessione **SK** e il proprio identificativo, criptati con la chiave $\mathbf{K_B}$, come comunicata dal server con il precedente messaggio. Bob può decriptare il messaggio e il fatto che sia stato cifrato da una entità fidata (il server) lo rende autentico

4. Bob risponde ad Alice con un nonce criptato con la chiave di sessione **SK**, per mostrare che è in possesso della chiave
5. Alice decifra il nonce di Bob, lo modifica con una semplice operazione, lo cifra nuovamente, sempre con **SK** e lo rispedisce indietro, provando così che è ancora attiva e in possesso della chiave

Da questo momento in poi la comunicazione è pienamente stabilita e viene condotta da entrambe le parti inviando messaggi cifrati con la chiave di sessione **SK.**

Le chiavi condivise tra gli utenti ed il KDC a volte vengono anche chiamate **Master Keys**.

Caratteristiche

Il KDC ha ovviamente una relazione di fiducia con Alice e Bob, in generale con tutti gli utenti che si appoggiano ad essa. Il numero delle chiavi che il KDC deve mantenere è lineare rispetto al numero di utenti del sistema, senza di esso ognuno dovrebbe condividere un segreto con tutti gli altri utenti del sistema, il che porterebbe ad uno spazio che cresce in maniera quadratica.

Il KDC ovviamente deve meritare la fiducia degli utenti, comportandosi come si deve, quindi cercare di evitare leaking delle chiavi.

È possibile che durante una sessione, se tirata troppo a lungo, si riesegua lo scambio di una nuova chiave di sessione. Questo perché è più sicuro nel caso in cui passasse troppo tempo, mantenere sempre la stessa.

Questo livello di protocollo viene nascosto sotto il livello applicativo, livello sessione.

Debolezze

Un attaccante può effettuare un attacco di tipo replay intercettando e conservando il messaggio 3. Successivamente, durante un'altra esecuzione del protocollo, l'attaccante può bloccare il messaggio 3 originale e inviare a Bob il messaggio salvato precedentemente, contenente la vecchia chiave di sessione. In questo modo, l'attaccante riesce a far utilizzare a Bob una chiave di sessione compromessa e può quindi compromettere la sicurezza della comunicazione tra Alice e Bob

Per mitigare la possibilità di attacchi replay, è importante utilizzare meccanismi di autenticazione più robusti e garantire la "freshness" dei messaggi scambiati nel protocollo. Ad esempio, questo difetto viene risolto nel protocollo Kerberos mediante l'inclusione di un timestamp. Inoltre, è possibile utilizzare versioni più sicure del protocollo Needham-Schroeder, come la versione con chiave pubblica, che tuttavia presenta anch'essa alcune vulnerabilità che sono state risolte con l'introduzione di modifiche proposte da Lowe.

Come si distingue una chiave da un’altra? Come dare quindi un significato alla chiave che viene scambiata? Se serve per decifrare o cifrare, etc.

Kerberos

Kerberos è un protocollo di autenticazione che funziona sulla base di ticket per consentire ai nodi che comunicano su una rete non sicura di dimostrare reciprocamente la propria identità in modo sicuro. I suoi progettisti l'hanno puntato principalmente su un modello client-server e fornisce un'autenticazione reciproca: sia l'utente che il server verificano l'identità l'uno dell'altro. I messaggi del protocollo Kerberos sono protetti da intercettazioni e attacchi di tipo reply.

Kerberos si basa sulla crittografia a chiave simmetrica e richiede una terza parte attendibile e, facoltativamente, può utilizzare la crittografia a chiave pubblica durante determinate fasi dell'autenticazione. Kerberos utilizza la porta UDP 88 per impostazione predefinita.

Sviluppo

Il Massachusetts Institute of Technology (MIT) ha sviluppato Kerberos nel 1988 per proteggere i servizi di rete forniti dal Progetto Athena, basandosi sul protocollo precedente a chiave simmetrica Needham-Schroeder. Esistono diverse versioni del protocollo; le versioni 1–4 si sono verificate solo internamente al MIT.

Neuman e John Kohl hanno pubblicato la versione 5 nel 1993 con l'intenzione di superare le limitazioni esistenti ed i problemi di sicurezza. La versione 5 è apparsa come RFC 1510, che è stata poi resa obsoleta da RFC 4120 nel 2005. All'inizio, le autorità degli Stati Uniti vietarono l'esportazione di questo protocollo perché utilizzava l' algoritmo di crittografia Data Encryption Standard (DES) (con chiavi a 56 bit).

Nel 2005, il gruppo di lavoro Kerberos dell'Internet Engineering Task Force (IETF) ha aggiornato le specifiche, aggiornando il cifrario ad AES, RFC 4120.

Protocollo

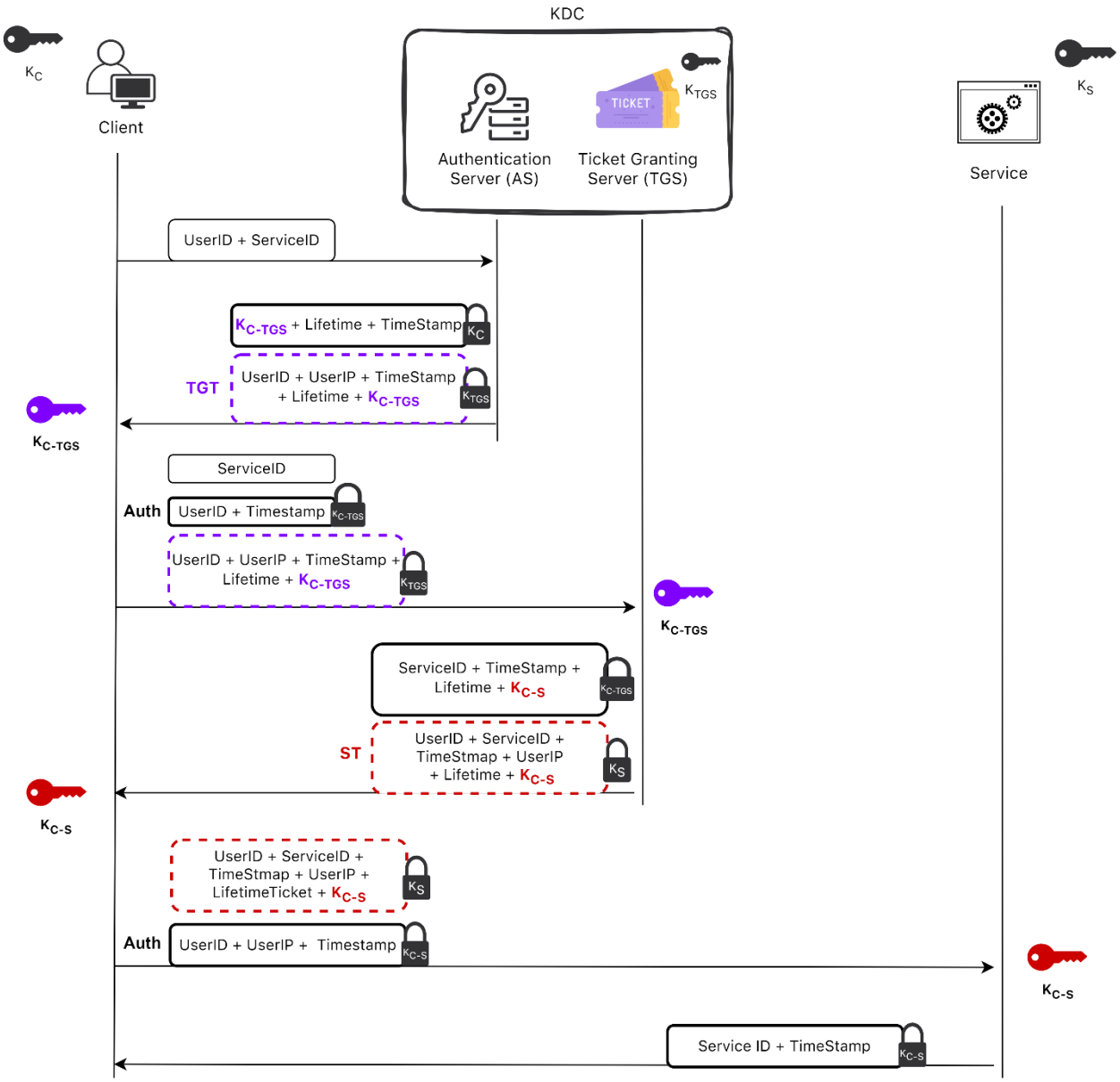
Kerberos si basa sul protocollo di Needham-Schroeder. Utilizza una terza parte affidabile per centralizzare la distribuzione delle chiavi detta Key Distribution Center (KDC), che consiste di due parti separate logicamente:

- **l'Authentication Server** (**AS**): L'AS mantiene un database delle chiavi segrete; ogni entità sulla rete, che sia un client o un server, condivide la chiave segreta solo con l'AS. La conoscenza di questa chiave serve per provare l'identità di un'entità. Inoltre l'AS emette dei ticket chiamati Ticket Granting Ticket (TGT) che viene utilizzato per autenticare l'utente presso il Ticket Granting Server (TGS)
- **Ticket Granting Server** (**TGS**): Il compito principale del TGS è quello di emettere i ticket di servizio che consentono agli utenti autenticati di accedere a risorse di rete specifiche. Il TGS mantiene le master keys di tutti i servizi attivi sulla rete, e crea Ticket chiamati Service ticket (ST) che sono utilizzati per autorizzare l'accesso a specifici servizi di rete.

Come per Needham-Schroeder, le chiavi segrete vengono anche chiamate **Master Keys**.

Kerberos funziona utilizzando dei "biglietti" (detti ticket) che servono per provare l'identità degli utenti.

Per comunicazioni tra due entità, l'AS genera una chiave di sessione, che può essere utilizzata dai due terminali per comunicare.



I messaggi scambiati sono i seguenti:

1. L'utente che vuole comunicare con un serizio, invia all'AS, il suo identificativo e l'ID del servizio che vuole usufruire.
2. L'AS, risponde con due messaggi:
    a. Il primo contenente la chiave $\mathbf{K_{C-TGS}}$ che dovrà usare il client per comunicare con il TGS, il lifetime del TGT ed un timestamp.
    b. Il secondo contiene un Ticket Granting Ticket TGT, cifrato con la chiave del TGS contenente l'User che ha bisogno di comunicare con il serizio, il suo IP, un timestamp, un tempo di validità per il TGT e la chiave $\mathbf{K_{C-TGS}}$ che dovrà usare il TGS per comunicare con l'utente.
3. L'utente decifra il primo messaggio, ed ottiene la chiave $\mathbf{K_{C-TGS}}$. Successivamente inoltra al TGS 3 messaggi:
    a. Il primo contiene semplicemente il Service ID

    b. Il secondo è cifrato con la chiave $\mathbf{K_{C\text{-}TGS}}$ e contiene l'ID dello User ed un timestamp
    c. Il terzo invece è il TGT inviato precedentemente dall'AS
4. Il TGS a questo punto decifra il TGT con la sua chiave, estrae la chiave $\mathbf{K_{C\text{-}TGS}}$ e la usa per decifrare il secondo messaggio. Successivamente compara i dati contenuti nel secondo messaggio (decifrato) ed il TGT: controlla che lo User ID sia lo stesso, ed il timestamp (con uno scarto di 1-2 minuti massimo). A questo punto il TGS invia 2 messaggi al Client:
    a. Il primo cifrato con la chiave $\mathbf{K_{C\text{-}TGS}}$ contenente il Service ID, un timestamp, il lifetime del TGT e la chiave condivisa $\mathbf{K_{C\text{-}S}}$ che l'utene dovrà usare per comunicare con il servizio.
    b. Il secondo messaggio è il Service Ticket - ST cifrato con la chiave del Servizio che contiene l'User ID che necessita del servizio, il service ID, un Timestamp, l'IP dell'Utente richiedente, un Lifetime per il ST, ed una chiave di sessione $\mathbf{K_{C\text{-}S}}$.
5. L'utente decifra il primo messaggio con la chiave $\mathbf{K_{C\text{-}TGS}}$ per estrarre la chiave di sessione. Successivamente inoltra al Service due messaggi:
    a. Il primo è cifrato con la chiave $\mathbf{K_{C\text{-}S}}$ appena estratta e contiene l'User ID ed un timestamp
    b. Il secondo è il ST, che viene semplicemente replicato
6. Il Service utilizza la sua chiave per decifrare l'ST ed estrae la chiave condivisa $\mathbf{K_{C\text{-}S}}$ per decifrare anche l'altro messaggio. Compara i dati e se tutto ok, risponde al client con un messaggio cifrato con $\mathbf{K_{C\text{-}S}}$ contenente l'ID del servizio ed un timestamp.

Da notare l'utilizzo dei Timestamp, che nel Needham–Schroeder non erano presenti.

I primi due punti in sostanza effettuano un'autentificazione presso l'AS. Questo perché, se non si è veramente l'utente che si dice di essere (punto 1), non si riuscirà nemmeno a decifrare il punto 2, in quanto non in possesso della master key.

Rinnovo dei ticket

Il rinnovo dei ticket in Kerberos offre la possibilità agli utenti autenticati di estendere la durata di validità dei loro ticket senza dover ripetere l'intero processo di autenticazione. Durante il rinnovo, l'utente presenta il suo Ticket Granting Ticket (TGT) scaduto al Ticket Granting Server (TGS) e richiede un rinnovo. Il TGS verifica l'autenticità del TGT e ne controlla la validità. Se il TGT è ancora valido e l'utente ha i diritti di rinnovo appropriati, il TGS emette un nuovo TGT con una nuova data di scadenza posticipata. Questo nuovo TGT viene quindi inviato all'utente, che lo sostituisce con il TGT scaduto. Il nuovo TGT consente all'utente di continuare ad accedere ai servizi di rete senza interruzioni, estendendo così la durata del suo accesso autorizzato. Grazie al processo di rinnovo, gli utenti possono mantenere un accesso fluido e continuativo ai servizi, senza dover affrontare la complessità dell'autenticazione ripetuta.

Non è possibile rinnovare un Service Ticket. Quando il Lifetime per un ST termina, l'utente deve richiederne un nuovo al Ticket Granting Server (TGS) per poter continuare ad accedere al servizio specifico.

È importante notare che la durata di validità degli ST è generalmente inferiore a quella del TGT. Questo è fatto per motivi di sicurezza, in modo che l'accesso ai servizi sia limitato nel tempo e periodicamente richieda una riautenticazione o il rinnovo del TGT.

Caratteristiche

Con l'aggiunta dei Timestamp si risolve il problema del Reply che ivnece era presente in Needham–Schroeder.

Kerberos ha requisiti temporali rigorosi, il che significa che gli orologi degli host coinvolti devono essere sincronizzati entro limiti configurati. I ticket hanno un periodo di disponibilità temporale e se l'orologio dell'host non è sincronizzato con l'orologio del KDC, l'autenticazione avrà esito negativo. La configurazione predefinita per MIT richiede che gli orari di clock non siano distanti più di cinque minuti. Di solito si usa NTP per questo.

Kerberos richiede che gli utenti e i servizi dispongano di una relazione di fiducia con il KDC.

L'attendibilità del client rende difficile la creazione di ambienti a fasi (ad es. domini separati per l'ambiente di test, l'ambiente di pre-produzione e l'ambiente di produzione): è necessario aggiungere client utenti aggiuntivi per ogni ambiente.

Debolezze

Poiché tutte le autenticazioni sono controllate da un centro di distribuzione delle chiavi centralizzato (KDC), la compromissione di questa infrastruttura di autenticazione consentirà a un utente malintenzionato di impersonare qualsiasi utente.

La crittografia Data Encryption Standard (DES) può essere utilizzata in combinazione con Kerberos, ma non è più uno standard Internet perché è debole. Esistono vulnerabilità di sicurezza in molti prodotti legacy che implementano Kerberos perché non sono stati aggiornati per utilizzare cifrari più recenti come AES invece di DES.

Dettagli HTTP

Le richieste HTTP possono passare per indirezione da:

- Proxies
- Common Gateway Interface
- Tunnels

Un Common Gateway Interface è una tecnologia standard usata dai web server per interfacciarsi con applicazioni esterne generando contenuti web dinamici.

Ogni volta che un client richiede al web server un URL corrispondente a un documento in puro HTML, gli viene restituito un documento statico (come un file di testo); se l'URL corrisponde invece a un programma CGI, il server lo esegue in tempo reale, generando dinamicamente informazioni per l'utente.

Il CGI è la prima forma di elaborazione lato server implementata: quando a un web server arriva la richiesta di un documento CGI (solitamente con estensione .cgi, .exe o .pl) il server esegue il programma richiesto e al termine invia al web browser l'output del programma. Il file CGI è un semplice programma già compilato (codice oggetto) e la risposta viene acquisita attraverso standard output. L'acquisizione dei parametri può avvenire attraverso variabili d'ambiente, passaggio di parametri sulla riga di comando o lo standard input a seconda della mole di dati e delle scelte del programmatore.

HTTP Request

I messaggi di richiesta vengono inviati da un client a un server di destinazione.

Formato della richiesta

Un client invia messaggi di richiesta al server, che consistono in:

- una riga di richiesta, costituita dal metodo di richiesta con distinzione tra maiuscole e minuscole, uno spazio, l'URL della risorsa, un altro spazio, la versione del protocollo, un ritorno a capo e un avanzamento riga
- zero o più campi di intestazione della richiesta (almeno 1 o più intestazioni in caso di HTTP/1.1), ciascuno costituito dal nome del campo senza distinzione tra maiuscole e minuscole, due punti, spazio bianco iniziale facoltativo, il valore del campo, uno spazio bianco finale facoltativo e che termina con un ritorno a capo e avanzamento riga
- una riga vuota, costituita da un ritorno a capo e un avanzamento riga (obbligatorio)
- un corpo del messaggio (facoltativo).

Metodi di richiesta

HTTP definisce metodi (a volte indicati come verbi) per indicare l'azione desiderata da eseguire sulla risorsa identificata. Spesso la risorsa corrisponde a un file o all'output di un eseguibile che risiede sul server nel caso di un CGI. La specifica HTTP/1.0 definisce i metodi:

- GET
    - Il metodo GET richiede che la risorsa di destinazione trasferisca una rappresentazione del suo stato. Le richieste GET dovrebbero solo recuperare dati e non dovrebbero avere altri effetti. Per recuperare risorse senza apportare modifiche, GET è preferibile a POST, poiché possono essere indirizzate tramite un URL. Ciò

consente l'aggiunta di segnalibri e la condivisione e rende le risposte GET idonee per la memorizzazione nella cache, che può far risparmiare larghezza di banda.

- HEAD
  - Il metodo HEAD richiede che la risorsa target trasferisca una rappresentazione del suo stato, come per una richiesta GET, ma senza i dati di rappresentazione racchiusi nel corpo della risposta. Ciò è utile per recuperare i metadati della rappresentazione nell'intestazione della risposta, senza dover trasferire l'intera rappresentazione. Gli usi includono il controllo della disponibilità di una pagina tramite il codice di stato e la ricerca rapida della dimensione di un file (Content-Length).
- POST
  - Il metodo POST richiede che la risorsa di destinazione elabori la rappresentazione racchiusa nella richiesta secondo la semantica della risorsa di destinazione. Ad esempio, viene utilizzato per inviare un messaggio a un forum Internet, iscriversi a una mailing list o completare una transazione di acquisto online

La specifica HTTP/1.1 ne ha aggiunti cinque nuovi:

- PUT
  - Il metodo PUT richiede che la risorsa di destinazione crei o aggiorni il proprio stato con lo stato definito dalla rappresentazione racchiusa nella richiesta. Una differenza rispetto a POST è che il client specifica la posizione di destinazione sul server.
- DELETE
  - Il metodo DELETE richiede che la risorsa di destinazione elimini il proprio stato.
- CONNECT
  - Il metodo CONNECT richiede che l'intermediario stabilisca un tunnel TCP/IP verso il server di origine identificato dalla destinazione della richiesta. Viene spesso utilizzato per proteggere le connessioni tramite uno o più proxy HTTP con TLS.
- OPTIONS
  - Il metodo OPTIONS richiede che la risorsa di destinazione trasferisca i metodi HTTP che supporta. Questo può essere utilizzato per verificare la funzionalità di un server web richiedendo '*' invece di una risorsa specifica.
- TRACE
  - Il metodo TRACE richiede che la risorsa di destinazione trasferisca la richiesta ricevuta nel corpo della risposta. In questo modo un cliente può vedere quali (eventuali) modifiche o aggiunte sono state apportate dagli intermediari.

Qualsiasi client può utilizzare qualsiasi metodo e il server può essere configurato per supportare qualsiasi combinazione di metodi. Non c'è limite al numero di metodi che possono essere definiti, il che consente di specificare metodi futuri senza interrompere l'infrastruttura esistente.

I nomi dei metodi sono case sensitive, al contrario, gli headers non lo sono.

Risposta HTTP

Un messaggio di risposta viene inviato da un server a un client come risposta al suo precedente messaggio di richiesta.

Formato della risposta

Un server invia messaggi di risposta al client, che consistono in:

- una riga di stato, composta dalla versione del protocollo, uno spazio, il codice di stato della risposta, un altro spazio, una frase motivo possibilmente vuota, un ritorno a capo e un avanzamento riga
- zero o più campi di intestazione della risposta, ciascuno costituito dal nome del campo senza distinzione tra maiuscole e minuscole, due punti, spazi bianchi iniziali facoltativi, il valore del campo, uno spazio bianco finale facoltativo e termina con un ritorno a capo e un avanzamento riga
- una riga vuota, costituita da un ritorno a capo e un avanzamento riga;
- un corpo del messaggio facoltativ.

Codici di stato della risposta

Sin da HTTP/1.0, la prima riga della risposta HTTP è chiamata riga di stato e include un codice di stato numerico seguito da una frase testuale del motivo. Il codice di stato della risposta è un codice intero a tre cifre che rappresenta il risultato del tentativo del server di comprendere e soddisfare la richiesta corrispondente del client. Il modo in cui il client gestisce la risposta dipende principalmente dal codice di stato e secondariamente dagli altri campi di intestazione della risposta. Infatti il client gestisce la risposta solo in base al codice di stato, la frase non viene presa in considerazione e possono essere sostituite con altri valori a discrezione dello sviluppatore web.

Se il codice di stato indica un problema, lo user agent potrebbe visualizzare la frase del motivo all'utente per fornire ulteriori informazioni sulla natura del problema. Lo standard consente inoltre allo user agent di tentare di interpretare la frase del motivo, sebbene ciò potrebbe non essere saggio poiché lo standard specifica esplicitamente che i codici di stato sono leggibili dalla macchina e le frasi del motivo sono leggibili dall'uomo.

I client potrebbero non comprendere tutti i codici di stato registrati, ma devono comprendere la loro classe (data dalla prima cifra del codice di stato) e trattare un codice di stato non riconosciuto come equivalente al codice di stato x00 di quella classe.

La prima cifra del codice di stato quindi, definisce la sua classe:

- 1xx (Info)
    - La richiesta è stata ricevuta, processo in corso.
- 2xx (Success)
    - La richiesta è stata ricevuta, compresa e accettata con successo.
- 3xx (Reindirizzamento)
    - Ulteriori azioni devono essere intraprese per completare la richiesta.
- 4xx (Errore del client)
    - La richiesta contiene una sintassi errata o non può essere soddisfatta.
- 5xx (Errore del server)
    - Il server non è riuscito a soddisfare una richiesta apparentemente valida.

Tra i codici più utilizzati, abbiamo:

- 2xx (Success)
    - 200 ok
    - 201 created
    - 202 accepted
    - 204 content
- 3xx (Rdirect)
    - 301 moved permanently

    - 302 moved temporarily
    - 304 not modified
- 4xx (Errore del client)
    - 400 bad request
    - 401 unauthorized
    - 403 forbidden
    - 404 not found
- 5xx (Errore del server)
    - 500 internal server error
    - 501 not implemented
    - 502 bad gateway
    - 503 service unavailable

Headers

Gli header HTTP contengono informazioni importanti per il protocollo HTTP, come ad esempio il tipo di contenuto della risposta, la lunghezza del contenuto e le informazioni sul server o sul client. Gli header HTTP sono formattati come coppie chiave-valore e sono separati dal corpo del messaggio HTTP da una riga vuota.

Ci sono quattro tipi principali di header HTTP/1.1:

- **Request headers**: contengono ulteriori informazioni sulla risorsa da recuperare o sul client che richiede la risorsa.
- **Response headers**: contengono informazioni aggiuntive sulla risposta, come la sua posizione o il server che la fornisce.
- **Representation headers**: contengono informazioni sul corpo della risorsa, come il tipo MIME o la codifica/compressione applicata.
- **Payload headers**: contengono informazioni, indipendenti dalla rappresentazione, sui dati del payload, inclusa la lunghezza del contenuto e la codifica utilizzata per il trasporto.

Alcuni Header di richiesta/risposta sono usati in entrambi i contesti. Per questo vengono ancora chiamati "Header generali" per indicare che possono venire usati sia per richiesta che per risposta, ma le versioni correnti della specifica HTTP/1.1 non categorizzano gli "header generali". Questi sono ora indicati semplicemente come header di risposta o di richiesta a seconda del contesto. Inoltre, molti chiamano i rappresentation header anche **entity headers** ma l'attuale specifica HTTP/1.1 non fa più riferimento ad entity header bensì a rappresentation.

Connection

L'header Connection è un header sia di richiesta che di risposta, che controlla se la connessione di rete rimane aperta al termine della transazione corrente. Se il valore inviato è keep-alive, la connessione è persistente e non chiusa, consentendo l'esecuzione di richieste successive allo stesso server.

Attenzione che alcuni header di http/1.1 specifici della connessione come Connection e Keep-Alive sono vietati in HTTP/2 e HTTP/3. Chrome e Firefox li ignorano nelle risposte HTTP/2, ma Safari è conforme ai requisiti della specifica HTTP/2 e non carica alcuna risposta che li contenga.

Sintassi:

- Connection: keep-alive
- Connection: close

Cache-control

“Cache-control” è un header sia di richiesta che di risposta, che controlla la memorizzazione nella cache nei browser e nelle cache condivise (ad es. proxy, CDN).

È possibile usarlo con diverse direttive, ad esempio:

- **max-age=N**: direttiva che indica che la risposta rimane aggiornata fino a N secondi dopo la generazione della risposta.
    - “Cache-Control: max-age=604800”
- **no-cache**: direttiva che indica che la risposta può essere archiviata nelle cache, ma la risposta deve essere convalidata con il server di origine prima di ogni riutilizzo, anche quando la cache è disconnessa dal server di origine. Nota che “no-cache” non significa "non memorizzare nella cache", ma richiede loro di riconvalidarla prima del riutilizzo. Il senso di "non memorizzare nella cache" è implementabile con la direttiva “no-store”.
    - Cache-Control: no-cache
- **no-store**: direttiva response indica che qualsiasi cache di qualsiasi tipo (privata o condivisa) non dovrebbe memorizzare questa richiesta/risposta.
    - Cache-Control: no-store

È possibile ancora trovare “Pragma: no-cache”, che è la versione obsoleta (HTTP1.0) di “Cache-Control: no-cache”. Tuttavia alcuni browser possono ancora supportarlo.

If-Modified-Since

L' If-Modified-Since è un header di richiesta che rende la richiesta condizionata: il server restituisce la risorsa richiesta, con uno 200 stato, se e solo se la risorsa è stata modificata dopo la data indicata. Se la risorsa non è stata modificata da allora, la risposta è 304 senza alcun corpo; Il server inoltre, nella risposta, inserisce l’header **Last-Modified** per indicate la data dell'ultima modifica.

A differenza di If-Unmodified-Since, If-Modified-Since può essere utilizzato solo con GET o HEAD.

Utilizzo:

- If-Modified-Since: <day-name>, <day><month><year><hour>:<minute>:<second> GMT
    - If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT

Refer

Referer è un header di richiesta HTTP che contiene l'indirizzo assoluto o parziale da cui è stata richiesta una risorsa. Consente ad un server di identificare le pagine di riferimento da cui le persone stanno visitando. Questi dati possono essere utilizzati per l'analisi, la registrazione, la memorizzazione nella cache ottimizzata e altro ancora. Ad esempio, quando si fa clic su un collegamento, il referer contiene l'indirizzo della pagina da cui proviene l’utente.

Utilizzo:

- Referer: <url>
    - Referer: https://example.com/. Indica che l’utente richiede una risorsa, e che l’indirizzo da cui proviene è quella indicata da refer.

User Agent

User-Agent è un header di richiesta ed è una stringa caratteristica che consente ai server e ai peer di rete di identificare l'applicazione, il sistema operativo, il fornitore e/o la versione dello user agent richiedente.

Utilizzo:

- User-Agent: <product> / <product-version> <comment>
    - User-Agent: Mozilla/5.0

Location

Location è un header di risposta ed indica l'URL a cui reindirizzare una pagina. Assume significato solo se servito con una risposta di stato 3xx (redirect) o (created) 201.

In caso di reindirizzamento, il metodo HTTP utilizzato per effettuare la nuova richiesta puntata da Location dipende dal metodo originale e dal tipo di reindirizzamento:

- **303 (See Other)**: si usa sempre GET
- **307 (Temporary Redirect) e 308 (Permanent Redirect)**: non modificano il metodo utilizzato nella richiesta originale.
- **301 (Moved Permanently) e 302 (Found)**: non cambiano il metodo originario (per la maggior parte delle volte) anche se gli user-agent più vecchi potrebbero (quindi in pratica non lo sai).

In caso di creazione di risorse, viene indicata l'URL della risorsa appena creata.

Utilizzo:

- Location: <url>
    - Location: /index.html

Server

Server è un header di risposta che descrive il software utilizzato dal server di origine che ha gestito la richiesta, ovvero il server che ha generato la risposta.

Utilizzo:

- Server: <product>
    - Server: Apache/2.4.1 (Unix)

Content-encoding

Content-encoding è un header di entità che elenca tutte le codifiche che sono state applicate alla rappresentazione (payload del messaggio) e in quale ordine. Ciò consente al destinatario di sapere come decodificare la rappresentazione per ottenere il formato del payload originale.

Si noti che il tipo del contenuto è specificato nell'intestazione Content-Type.

MIME TYPES

I tipi MIME, acronimo di Multipurpose Internet Mail Extensions, sono una convenzione utilizzata per identificare i tipi di dati che possono essere trasmessi su Internet. Sono nati originariamente per estendere il formato standard delle email, consentendo di allegare file non testuali come immagini, audio o video.

Con il passare del tempo, i tipi MIME sono stati adottati in molti altri contesti, tra cui la trasmissione di file attraverso il protocollo HTTP. Infatti, quando un browser richiede una risorsa da un server tramite il protocollo HTTP, il server risponde includendo un'intestazione Content-Type, che specifica il tipo MIME del file che sta inviando. Il browser utilizza questa informazione per determinare come gestire il contenuto della risorsa.

I tipi MIME sono identificati da una stringa composta da due parti, separate da una barra obliqua. La prima parte indica la categoria generale del tipo di file (ad esempio, "text" per i file di testo o "image" per le immagini), mentre la seconda parte indica il formato specifico (ad esempio, "html" per i file di pagina web o "jpeg" per le immagini in formato JPEG).

Grazie ai tipi MIME, i programmi che utilizzano questi protocolli possono facilmente interpretare e gestire i dati trasmessi, indipendentemente dal formato o dalla codifica specifica del file.

Alcuni mime type:

- Text:
    - Plain text: text/plain
    - HTML: text/html
    - CSS: text/css
    - XML: text/xml
    - JavaScript: text/javascript
    - JSON: application/json
- Image:
    - JPEG: image/jpeg
    - PNG: image/png
    - GIF: image/gif
    - SVG: image/svg+xml
- Audio:
    - MP3: audio/mpeg
    - WAV: audio/wav
    - OGG: audio/ogg
- Video:

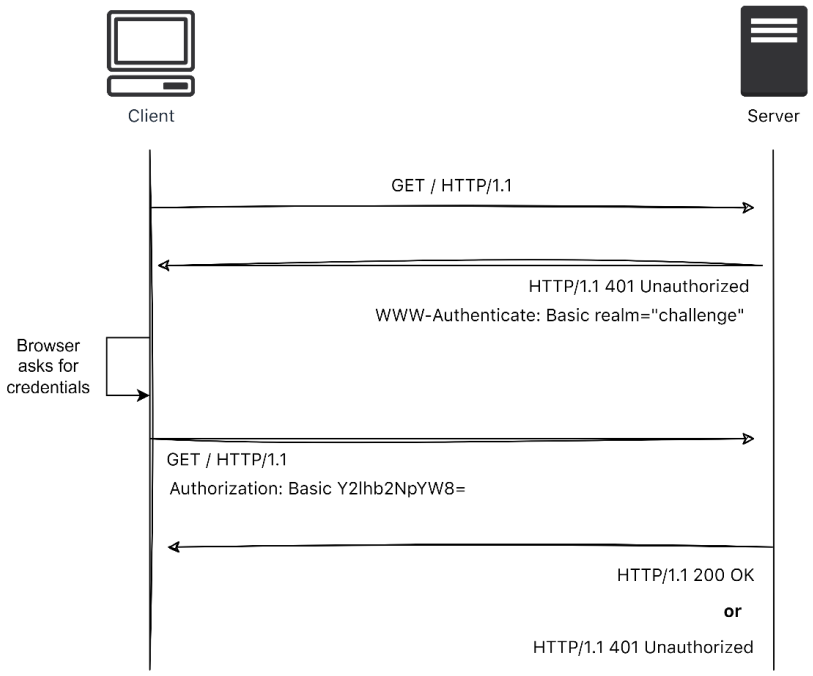
    - MPEG: video/mpeg
    - MP4: video/mp4
    - AVI: video/avi
- Application:
    - PDF: application/pdf
    - ZIP: application/zip
    - GZIP: application/gzip
    - Microsoft Word: application/msword
    - Microsoft Excel: application/vnd.ms-excel
    - Microsoft PowerPoint: application/vnd.ms-powerpoint

HTTP Authentication

HTTP fornisce una struttura generale per il controllo degli accessi e l'autenticazione.

Framework generale di autenticazione HTTP

RFC 7235 definisce il framework di autenticazione HTTP, che può essere utilizzato da un server per contestare una richiesta client e da un client per fornire informazioni di autenticazione.



Il flusso autenticazione tra client e server funziona così:

- Il client che desidera una risorsa protetta, richeide la risorsa attraverso metodi GET/POST
- Il server risponde a un client con uno status code 401 (Non autorizzato) e fornisce informazioni su come autorizzare, inserendo un header “WWW-Authenticate” di risposta contenente almeno una challenge.
- Un client che desidera autenticarsi con il server può quindi farlo includendo nella risposta successiva, l’header “Authorization” di richiesta con le credenziali.
    - Di solito nei browser questo è rappresentato da un popup di richiesta con username e password.
- Se la challenge è stata soddisfatta, il server risponde con status code 200 OK, altrimenti status code 401 Unauthorized.

Autenticazione proxy

Lo stesso meccanismo di richiesta e risposta può essere utilizzato per l'autenticazione proxy. Poiché sia l'autenticazione delle risorse che l'autenticazione del proxy possono coesistere, è necessario un set diverso di intestazioni e codici di stato. Nel caso dei proxy, il codice di stato della sfida è 407 (Proxy Authentication Required), l’header Proxy-Authenticate di risposta contiene almeno una sfida

applicabile al proxy e l'header Proxy-Authorization di richiesta viene utilizzata per fornire le credenziali al server proxy.

Intestazioni WWW-Authenticate e Proxy-Authenticate

Le intestazioni di risposta **WWW-Authenticate** e **Proxy-Authenticate** definiscono il metodo di autenticazione da utilizzare per ottenere l'accesso a una risorsa. Devono specificare quale schema di autenticazione viene utilizzato, in modo che il client che desidera autorizzare sappia come fornire le credenziali.

La sintassi per queste intestazioni è la seguente:

- WWW-Authenticate: <type> realm=<realm>
- Proxy-Authenticate: <type> realm=<realm>

Intestazioni Authorization e Proxy-Authorization

Le intestazioni **Authorization** e **Proxy-Authorization** della richiesta contengono le credenziali per autenticare uno user agent con un server (proxy). Anche in questo caso <type> è necessario seguito dalle credenziali, che possono essere codificate o crittografate a seconda dello schema di autenticazione utilizzato.

La sintassi per queste intestazioni è la seguente:

- Authorization: <type> <credentials>
- Proxy-Authorization: <type> <credentials>

Basic Authentication

Il framework di autenticazione HTTP generale è la base per una serie di schemi di autenticazione, ma quello più usato è “base”.

Lo schema di autenticazione HTTP "base" è definito in RFC 7617, che trasmette le credenziali come coppie utente:password, codificate utilizzando base64, quindi nell’header Authorization, il campo che segue Basic è codificato in base64, esempio: **Authorization: Basic Y2lhb2NpYW8=**

Sicurezza dell'autenticazione di base

Poiché l'ID utente e la password vengono passati sulla rete come testo in chiaro (è codificato in base64, ma base64 è una codifica reversibile), lo schema di autenticazione di base non è sicuro.

lo schema di autenticazione "di base" utilizzato nel diagramma precedente invia le credenziali codificate ma non crittografate. Questo sarebbe completamente insicuro a meno che lo scambio non fosse su una connessione sicura (HTTPS/TLS).

OPENSSL

Apache

Apache2 è un server web open source ampiamente utilizzato per la distribuzione di contenuti web su Internet. Il nome completo del software è Apache HTTP Server ed è sviluppato e mantenuto dalla Apache Software Foundation.

Apache2 è disponibile per una vasta gamma di sistemi operativi, tra cui Linux, Windows, macOS e altri, ed è estremamente flessibile e scalabile. Supporta numerosi protocolli di rete, tra cui HTTP, HTTPS, FTP e altri, e può essere configurato per ospitare diverse applicazioni web, come siti statici, dinamici o applicazioni basate su framework come Django o Ruby on Rails.

È altamente personalizzabile e offre una vasta gamma di funzionalità, tra cui la gestione dei log, la gestione degli utenti e la configurazione avanzata dell'autenticazione e dell'autorizzazione. Inoltre, Apache2 può essere esteso tramite moduli aggiuntivi che aggiungono funzionalità come la compressione delle risorse, la gestione dei cache e altro ancora.

installazione

$ apt-get install apache2

Funzionamento

Per avviare il processo che gestisce le richieste web, è necessario invocare il comando apachectl.

Comandi principali:

- **apachectl start**: avvia il processo demone httpd (o apache2) per iniziare a gestire le richieste.
- **apachectl stop**: interrompe il processo demone http (o apache2)
- **apachectl restart**: Riavvia il processo demone httpd (o apache2). Se il demone non è in esecuzione, viene avviato.

Il nome del processo del server Apache può variare a seconda del sistema operativo e della configurazione. Ad esempio, su alcune distribuzioni Linux, il processo potrebbe chiamarsi "httpd", mentre su altre potrebbe essere chiamato "apache2".

Workers

Quando si esegue il comando **apachectl start**, in realtà, vengono avviati non un solo gestore di richieste ma **N** (workers), dove **N** è indicato nel file */etc/apache2/mods-enabled/mpm_event.conf*.

Ad esempio, un file *npm-event.conf* potrebbe essere strutturato in questo modo:

```
<IfModule mpm_event_module>
    StartServers             3
    MinSpareThreads          25
    MaxSpareThreads          75
    ThreadLimit              64
    ThreadsPerChild          25
    MaxRequestWorkers        150
    MaxConnectionsPerChild   0
</IfModule>
```

Dove StartServers indica il numero di worker che gestiranno le richieste.

Configuration files

Apache HTTP Server viene configurato inserendo le direttive nei file di configurazione in formato testuale. Il file di configurazione principale è solitamente chiamato httpd.conf (o apache2.conf).

I file di configurazione contengono una direttiva per riga. La "\" può essere utilizzata come ultimo carattere su una riga per indicare che la direttiva continua sulla riga successiva. Non devono esserci altri caratteri o spazi bianchi tra il backslash e la fine della riga.

Direttive e sezioni

Le direttive e le sezioni sono due concetti fondamentali nella configurazione di Apache.

Le direttive sono istruzioni che vengono utilizzate per configurare il comportamento del server web. Le direttive possono essere inserite all'interno del file di configurazione di Apache e indicano al server web come gestire specifici aspetti della configurazione. Ad esempio, la direttiva "Listen" viene utilizzata per specificare la porta su cui il server web deve ascoltare le richieste in arrivo.

Le sezioni, d'altra parte, sono blocchi di codice che racchiudono un insieme di direttive correlate. Le sezioni vengono utilizzate per organizzare la configurazione di Apache in modo logico e strutturato. Ad esempio, la sezione "VirtualHost" viene utilizzata per definire le impostazioni relative ai virtual host.

È importante notare che le direttive e le sezioni possono essere annidate l'una nell'altra, creando una struttura gerarchica e complessa di configurazione. Ad esempio, è possibile creare una sezione "VirtualHost" che contiene diverse direttive "Directory" per definire le impostazioni di accesso per le diverse cartelle del virtual host.

Modules

httpd (o apache2) è un server modulare: solo la funzionalità più basilare è inclusa nel server principale, funzionalità estese sono disponibili tramite moduli che possono essere caricati in httpd (ad esempio per abilitare HTTPS è necessario caricare un modulo specifico). Per impostazione predefinita, un set base di moduli è incluso nel server in fase di compilazione. httpd deve essere ricompilato per aggiungere o rimuovere moduli.

È come scrivere un programma in C, non si inserisce tutto dentro un file e si compila, ma si importano i moduli necessari. La struttura di apache è la seguente:

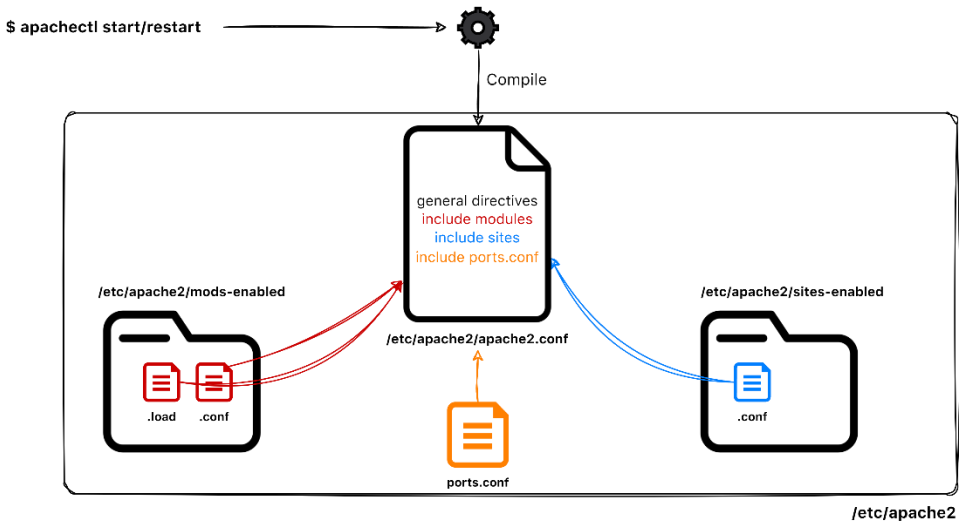
```
/etc/apache2/
#       |-- apache2.conf
#       |       `--  ports.conf
#       |-- mods-enabled
#       |       |-- *.load
#       |       `-- *.conf
#       |-- conf-enabled
#       |       `-- *.conf
#       `-- sites-enabled
#               `-- *.conf
```

In apache, il file principale è il file */etc/apache2/apache2.conf*, che contiene le direttive globali del webserver. Oltre a queste direttive, in questo file vengono importati tutti i file e moduli necessari o richiesti.

In particolare:

- Apache carica tutti i moduli che si trovano all'interno delle directory:
  - */etc/apache2/mods-enabled/*.load*
  - */etc/apache2/mods-enabled/*.conf*
- Compila i file di configurazione dei siti che si trovano in:
  - */etc/apache2/sites-enabled/*.conf*
- Compila il file */etc/apache2/ports.conf*
  - questo file contiene la configurazione delle porte utilizzate dal server. Di solito, la porta 80 è utilizzata per le richieste HTTP e la porta 443 per le richieste HTTPS.

Nella directory **mods-enabled** sono contenuti i file dei i moduli Apache2 abilitati in quel momento (i file hanno estensione **.load** o **.conf** e sono link simbolici ai file nella directory **mods-available**). Mentre la directory **sites-enabled** contiene i file di configurazione per i siti web abilitati sul server. Anche in questo caso, i file sono link simbolici ai file nella directory **sites-available**. I file di configurazione dei siti contengono informazioni su come il server gestisce le richieste per un sito web specifico, come la sua posizione nella directory del server, le impostazioni di accesso, le impostazioni di sicurezza e così via.



Per vedere quali moduli sono attualmente compilati nel server, **apachectl -M**.

Per abilitare un modulo, è necessario conoscere il nome del modulo e successivamente abilitarlo utilizzando il comando a2enmod (**a**pache**2 en**able **mod**ule):

$ sudo a2enmod <module_name>

Basic Authentication con Apache2

Il modulo responsabile di implementare la basic authentication si chiama "auth_basic". Quindi, il primo passo è abilitarlo.

$ sudo a2enmod auth_basic

Successivamente si usa il comando (che viene fornito insieme al package apache2) **htpasswd**, che viene utilizzato per creare e aggiornare i file che memorizzano nomi utente e password per la base

authentication degli utenti. Se htpasswd non è in grado di accedere al file delle password, ad esempio non essere in grado di scrivere nel file di output o non è in grado di leggere il file per aggiornarlo, restituisce uno stato di errore e non apporta modifiche.

$ sudo htpasswd -c /etc/apache2/.htpasswd username

Questo quindi creerà un file delle password nella directory "/etc/apache2" e chiederà di inserire la password per l'utente "username".

Nel file di configurazione del sito, bisogna aggiungere le seguenti direttive in una sezione Directory che indica il path del sottoalbero del file system da proteggere:

<Directory "path_to_protect ">
AuthType *Basic*
AuthName *"Restricted Content"*
AuthBasicProvider *file*
AuthUserFile */etc/apache2/.htpasswd*
Require *valid-user*
</Directory>

Dove:

- **AuthType**: Questa direttiva specifica il tipo di autenticazione utilizzato. In questo caso, stiamo utilizzando l'autenticazione di base.
- **AuthName**: Questa direttiva specifica il nome della zona protetta dal controllo di autenticazione.
- **AuthUserFile**: Questa direttiva specifica il percorso completo del file di password.
- **Require**: Questa direttiva specifica i criteri di autorizzazione per accedere alla zona protetta. In questo caso, stiamo richiedendo che l'utente abbia effettuato l'autenticazione tramite le credenziali corrette. Se ad esmepio ci fosse stato "***Require all granted***", tutti avrebbero l'accesso, senza la richiesta di credenziali. Se inve ce si vuole dare l'accesso ad un singolo utente, basterebbe usare "***Require user username***".
- **AuthBasicProvider**: Nel caso della direttiva "AuthBasicProvider file", stiamo dicendo ad Apache di utilizzare il file di password specificato nella direttiva "AuthUserFile" come provider di autenticazione di base. Questa è la configurazione predefinita per l'autenticazione di base con Apache, quindi se non specificato, verrà utilizzato il provider "file".

Se si vuole configurare la Basic Authentication per tutto il server, basta inserire le direttive nel file apache2.conf.

Dopodichè, dato che sono moduli statici, bisogna riavviare il servizio:

$ sudo apachectl restart

HTTPS con Apache2

Per mettere su un servizio HTTPS su apache2, è necessario abilitare il modulo chiamato "ssl", quindi, utilizzando a2enmod:

$ sudo a2enmod ssl

Successivamente, bisogna creare un virtual host, in ascolto sulla porta 443 aventi le seguenti direttive:

SSLEngine on
SSLCertificateFile ***<path_to_server_certificate>***
SSLCertificateKeyFile ***<path_to_server_key>***

Infine spostare il file contenente il virtual host in /etc/apache2/sites-enabled/ (oppure link simbolico), e riavviare il servizio:

$ sudo apachectl restart

Creare chiavi e certificato per il server

Per creare le chiavi ed il certificato, si utilizza openssl via CLI.

Per creare il certificato però poi sarà necessario avere un'ente di certificazione che è in grado di crearlo. Facciamo finta di essere noi stessi la CA. Per farlo è necessario creare anche le chiavi per la CA:

$ openssl genrsa -out ./CA.key 4096

Ora il certificato per la CA (necessario solo se si vuole testare il tutto su browser, in quanto non è una CA che il browser riconosce). Per fare questo esiste il comando **openssl req** che è utilizzato per generare richieste di certificato PKCS#10 ed altre utilità di generazione di certificato:

**$ openssl req -new -x509 -days 365 -key ./CA.key **

-subj="/C=IT/ST=TO/O=Unito/CN=testHTTPwebsite CA" -out ./CA.crt

I vari argomenti passati al comando **openssl req** specificano le seguenti azioni:

- **-new:** questa opzione genera una nuova richiesta di certificato.
- **-x509:** questa opzione emette un certificato autofirmato invece di una richiesta di certificato.
- **-days 365:** indica che il certificato sarà valido per 365 giorni a partire dalla sua creazione.
- **-key ./CA.key**: specifica il percorso del file della chiave privata con cui il certificato sarà firmato.
- **-subj:** specifica i dettagli del soggetto per il certificato, cioè le informazioni relative all'entità a cui viene rilasciato il certificato. In questo caso:
  - **C=IT:** indica il codice di due lettere per il paese (in questo caso, immagino si tratti di un errore, in quanto RH non è un codice di paese valido);
  - **ST=TO:** indica lo stato o la regione;
  - **O=Unito:** indica l'organizzazione;
  - **CN=testHTTPwebsite:** indica il nome comune del certificato.
- **-out ./CA.crt:** specifica il percorso e il nome del file di output.

Ora abbiamo una finta CA che firmerà il certificato del nostro server. Creiamo le chiavi del server:

**$ openssl req -newkey rsa:4096 -nodes -keyout server.key **
-subj="/C=IT/ST=TO/O=Unito/CN=*.testwebsite.it" -out server.csr

I vari argomenti passati al comando **openssl req** specificano le seguenti azioni:

- **-newkey rsa:4096:** indica che una nuova chiave privata RSA con una lunghezza di 4096 bit verrà generata insieme alla richiesta di certificato.
- **-nodes**: specifica che la chiave privata non verrà crittografata con una passphrase.
- **-keyout server.key**: specifica il percorso e il nome del file in cui la chiave privata verrà scritta.
- **-out server.csr** specifica il percorso e il nome del file in cui la CSR verrà scritta.

In questo comando si sono sia create le chiavi del server (-newkey rsa:4096) che la richiesta di certificato. Si potevano anche separare in due comandi diversi.

Ora si crea il certificato per il server, per farlo si usa il comando openssl x509:

```
$ openssl x509 -req \
-extfile <(printf "subjectAltName=DNS:testwebsite.it,DNS:*.testwebsite.it") -days 365 -in \
./server.csr -CA ./CA.crt -CAkey ./CA.key -CAcreateserial -out ./server.crt
```

Il comando **openssl x509** con questi argomenti è utilizzato per firmare una CSR con la chiave privata della CA generare un nuovo certificato firmato dalla CA. I vari argomenti passati al comando **openssl x509** specificano le seguenti azioni:

- **-req:** indica che verrà utilizzata una CSR come input per la creazione del certificato firmato (perché di default il comando riceve un certficato, non una richeista e quindi va specificato che l'input è una richiesta).
- **-extfile <(printf "subjectAltName=DNS:testwebsite.it,DNS:*.testwebsite.it")** specifica l'estensione **subjectAltName** per il certificato.
- **-days 365:** indica che il certificato sarà valido per 365 giorni a partire dalla sua creazione.
- **-in ./server.csr:** specifica il percorso e il nome del file della CSR da utilizzare per creare il certificato.
- **-CA ./CA.crt:** specifica il percorso e il nome del file del certificato radice CA utilizzato per firmare la CSR.
- **-CAkey ./CA.key:** specifica il percorso e il nome del file della chiave privata associata al certificato radice CA utilizzato per firmare la CSR.
- **-CAcreateserial:** indica che verrà creato un nuovo file seriale per il certificato.
- **-out ./server.crt:** specifica il percorso e il nome del file in cui il certificato firmato verrà scritto.

Mutual Authentication con Apache2

Per implementare la mutual authentication (e quindi sia client che server si autenticano con meccanismi a chiave pubblica) è necessario inserire delle direttive specifiche.

Per effettuare la client authentication, è necessario creare un certificato PKCS#12, partendo dal certificato del client e la sua chiave privata:

```
openssl pkcs12 -export -in ./client.crt -inkey ./client.key -out ./client.p12
```

È necessario poi nel virtualhost di apache, inserire le seguenti direttive:

```
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile        <path_to_client_ca_crt>
```

Docker

Docker è una piattaforma di virtualizzazione leggera che permette di creare, distribuire e gestire applicazioni in container. In altre parole, Docker consente di "incapsulare" un'applicazione, con tutte le sue dipendenze e librerie, in un container isolato e portatile, che può essere eseguito in qualsiasi ambiente, indipendentemente dalla configurazione del sistema host.

I container Docker sono simili alle macchine virtuali, ma sono molto più leggeri e veloci perché condividono il kernel del sistema operativo host. Ciò significa che invece di dover installare e gestire un'intera macchina virtuale, si può creare un container Docker e farvi girare l'applicazione desiderata.

Tra le principali caratteristiche di Docker, ci sono:

- **Portabilità**: i container Docker possono essere eseguiti su qualsiasi sistema operativo o ambiente, a condizione che Docker sia installato.
- **Isolamento**: ogni container è isolato dagli altri container e dal sistema host, garantendo che le applicazioni possano essere eseguite in un ambiente sicuro e stabile.
- **Scalabilità**: è facile creare e distribuire molteplici istanze di un'applicazione in container Docker, e Docker offre anche strumenti per il bilanciamento del carico e la gestione di cluster di container.
- **Velocità**: i container Docker possono essere avviati e fermati in pochi secondi, il che li rende ideali per la distribuzione di applicazioni in ambienti dinamici o in cloud.

Immagini

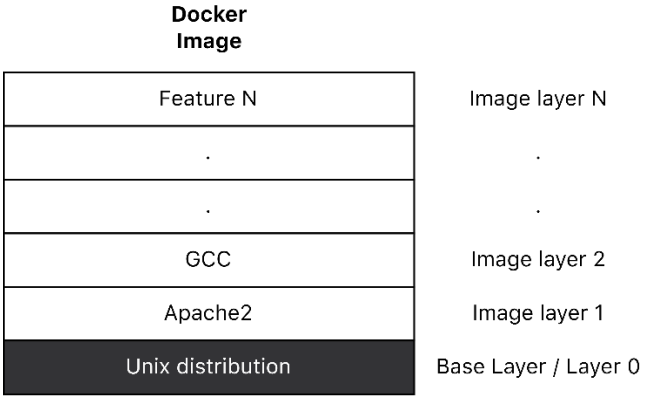
Un'immagine Docker è un pacchetto autonomo e portatile che include tutto il necessario per eseguire un'applicazione, tra cui il codice sorgente, le dipendenze e il sistema operativo. Sono in sostanza delle istantanee di un ambiente pre configurato ed isolato.

In Docker, esistono due tipi di immagini

- **Le immagini predefinite**: sono immagini già esistenti e mantenute da Docker e altre organizzazioni, disponibili nel Docker Hub o in altri registri Docker. Queste immagini contengono un sistema operativo e l'applicazione o il servizio da eseguire all'interno del container. Utilizzando un'immagine predefinita, è possibile creare rapidamente un container senza dover creare una nuova immagine personalizzata. Inoltre, le immagini predefinite sono già configurate e ottimizzate per l'esecuzione dell'applicazione o del servizio specifico.
- **Le immagini personalizzate**: sono immagini create dagli utenti, a partire da un'immagine di base, per includere l'applicazione o il servizio specifico e le configurazioni personalizzate. Le immagini personalizzate possono essere create utilizzando un Dockerfile, che specifica le istruzioni per la creazione dell'immagine personalizzata. Le immagini personalizzate possono essere utili quando è necessario includere pacchetti o librerie personalizzate o quando è necessario eseguire un'applicazione o un servizio specifico in un ambiente controllato e isolato.

In generale, l'utilizzo di immagini predefinite può essere utile per la maggior parte delle esigenze di containerizzazione, poiché riduce il tempo e lo sforzo necessario per la creazione di un container. Tuttavia, le immagini personalizzate possono essere necessarie in alcune situazioni specifiche, ad esempio per includere componenti personalizzati o configurazioni particolari.

layers



In Docker, un'immagine è composta da una serie di strati sovrapposti chiamati "layer". Ogni layer rappresenta un insieme di modifiche apportate all'immagine di base, che viene denominata "layer di base" o "layer zero". I layer vengono creati in modo incrementale a partire dal layer zero, in modo da creare una catena di modifiche applicate all'immagine.

L'immagine base è l'immagine di partenza su cui vengono create altre immagini personalizzate. Ci sono diverse immagini base disponibili, ognuna delle quali fornisce un sistema operativo e un set di strumenti di base per l'esecuzione dell'applicazione.

Di solito si fa uso di un layer zero che rappresenta una distribuzione Unix:

1. **Ubuntu**: immagine basata su Ubuntu Linux, una distribuzione popolare di Linux utilizzata in molti server.
2. **Alpine**: immagine basata su Alpine Linux, una distribuzione leggera e sicura di Linux.
3. **Debian**: immagine basata su Debian Linux, una distribuzione Linux stabile e affidabile.
4. **Centos**: immagine basata su CentOS Linux, una distribuzione Linux gratuita e open source basata su Red Hat Enterprise Linux (RHEL).

In generale, ogni layer di un'immagine Docker è indipendente dagli altri layer e rappresenta una modifica autonoma all'immagine. Ciò significa che è possibile rimuovere o sostituire un layer senza influire sugli altri layer o sull'immagine Docker in generale. Tuttavia, i layer possono avere dipendenze impliciti dagli altri layer che li precedono nella catena di sovrapposizione. Ad esempio, se un layer dipende da un file creato in un layer precedente, la rimozione di quel file potrebbe causare problemi nel layer successivo. Per questo motivo, è importante considerare l'ordine in cui si creano e si sovrappongono i layer, nonché le dipendenze tra di essi, al fine di garantire che l'immagine Docker funzioni correttamente e senza problemi.

Layer caching and reusing

Uno dei vantaggi che la stratificazione offre, è quello del caching: grazie alla capacità di riutilizzare i layer esistenti e di scaricare solo i nuovi layer dalla registry di Docker, la creazione e la distribuzione delle immagini Docker diventano molto più veloci ed efficienti.

Ad esempio, Se si crea un'immagine Y che utilizza uno o più layer presenti anche nell'immagine X, Docker utilizzerà automaticamente i layer condivisi durante la creazione dell'immagine Y. In particolare, quando si crea un'immagine Docker, Docker utilizza il meccanismo di stratificazione a layer per sovrapporre i diversi layer che compongono l'immagine. Se un layer è presente anche in un'altra immagine Docker, Docker utilizza la cache locale per scaricare rapidamente il layer

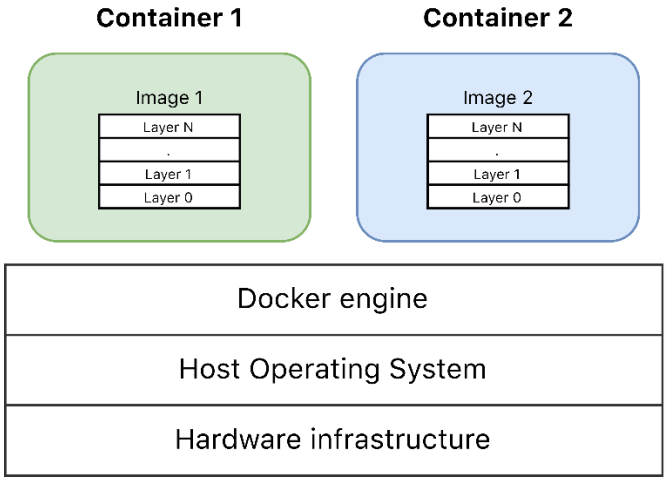
condiviso e riutilizzarlo nella nuova immagine, quindi, i layer comuni vengono scaricati una sola volta e memorizzati una sola volta: vantaggi per il download e l'archiviazione.

Containers

In Docker, un container è un'istanza in esecuzione di un'immagine Docker. Un container Docker include l'applicazione o il servizio da eseguire, insieme a tutte le librerie e le dipendenze necessarie per l'esecuzione dell'applicazione o del servizio. Inoltre, ogni container ha il proprio file system isolato dal sistema host e dagli altri container.

Inoltre, in Docker, un container condivide il kernel del sistema operativo con il sistema host. Questo significa che il sistema operativo all'interno del container è un sottoinsieme del sistema operativo del sistema host: in pratica, quando si esegue un container Docker, il sistema operativo all'interno del container utilizza le risorse del sistema operativo del sistema host per le funzionalità del kernel, come la gestione dei processi, della memoria e delle risorse di rete. Ciò significa che i container Docker hanno un'efficienza molto maggiore rispetto alle macchine virtuali tradizionali, perché non richiedono la virtualizzazione di tutto il sistema operativo.

Grazie a questa condivisione del kernel, i container Docker sono molto più leggeri e veloci rispetto alle macchine virtuali tradizionali. Inoltre, il fatto che il sistema operativo all'interno del container sia un sottoinsieme del sistema operativo del sistema host rende i container Docker altamente portabili e facili da distribuire su diversi ambienti e piattaforme.



Ovviamente è possibile avere più container che utilizzano la stessa immagine.

Volumes

I volumi Docker sono uno strumento che consente di gestire e persistere i dati tra i container Docker e l'host o tra i container stessi. I volumi forniscono un modo conveniente per archiviare i dati generati o utilizzati dai container, consentendo di separare il contenuto dei container dalla loro durata.

Ecco alcuni punti chiave sui volumi Docker:

- **Persistenza dei dati**: Quando un container Docker viene eliminato, tutti i dati all'interno di esso vengono persi. L'uso di volumi consente di separare i dati dal ciclo di vita del container, consentendo di conservare i dati anche dopo che il container è terminato o viene ricreato.

- **Condivisione dei dati tra i container**: I volumi Docker possono essere montati su più container contemporaneamente, consentendo la condivisione dei dati tra i container. Ciò può essere utile, ad esempio, quando si desidera separare il servizio web e il database in container separati ma condividere i dati tra di loro.
- **Montaggio di file o directory**: Un volume Docker può essere collegato a un file o a una directory sull'host o può essere creato come volume anonimo. Questo consente di specificare quali dati devono essere resi disponibili all'interno del container.
- **Persistenza dei dati tra ricreazioni di container**: I volumi Docker persistono anche tra le ricreazioni dei container. Ciò significa che se un container viene eliminato e poi ricreato utilizzando lo stesso volume, i dati all'interno del volume saranno ancora accessibili.
- **Gestione dei dati sensibili**: I volumi Docker possono essere utilizzati per archiviare dati sensibili come password, chiavi di crittografia o certificati. Ciò consente di separare e gestire in modo sicuro i dati sensibili all'interno dei volumi anziché memorizzarli direttamente nell'immagine del container.

Creare un immagine

Per creare un immagine docker personalizzata, è necessario creare un file che contenga tutte le informazioni necessarie per costruire l'immagine nuova. Il file in questione è denominato Dockerfile.

Un Dockerfile inizia con la direttina FROM, in quando è necessaio indicare il layer zero (base image) da cui partire. Successviamente, ogni direttiva che modifica lo stato del sistema o dell'immagine è considerato un layer dell'immagine nuova. Esempio:

```
FROM ubuntu

RUN apt-get update \

    && apt-get install -y --no-install-recommends apache2 \

    && apt-get install -y --no-install-recommends python3 \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/*

CMD ["/bin/bash"]
```

In questo caso RUN modifica lo stato dell'immagina installando nuovi pacchetti, quindi aggiunge un layer all'immagine. CMD invece è solo l'entrypoint, non modifica lo stato.

Al posto di **CMD ["/bin/bash"]**, è possibile anche usare **ENTRYPOINT ["/bin/bash"]**, i due comandi sono quasi uguali, la differenza è che **CMD** può essere sovrascritto mentre **ENTRYPOINT** no. Per sovrascritto si intende, che quando si avvia un container docker, è possibile indicare un entrypoint. Se l'immagine contiene un **CMD** questo sarà sovrascritto dall'entrypoint definito in fase di avvio del container.

Una volta scritto il Dockerfile, per creare l'immagine effettiva:

```
$ docker build -t <nome_immagine> <path_to_Dockerfile>
```

Per vedere le immagini presenti nel sistema:

```
$ docker images
```

Creare un container

Una volta creata l'immagine, è possibile creare il container:

docker create [OPTIONS] IMAGE [COMMAND] [ARG...]

Ad esempio:

$ docker create --tty <nome_immagine>

In questo modo si è creato il container, non ancora avviato. Per visualizzare la lista dei contanier è necessario eseguire:

$ docker ps

Se si vuole dare un nome al container:

$ docker create --tty --name <container_name> <nome_immagine>

In questo modo però si visualizzano solo i container avviati. Per visualizzare tutti i contaner, sia quelli avviati che non:

$ docker ps -a

Come **<nome_immagine>** è anche possibile usare immagini predefinite, tipo:

$ docker create ubuntu

Crea un container con l'immagine predefinita, presa dal docker HUB.

Avviare un container

Per avviare un contaner:

$ docker start <id_or_name>

Se si vuole avere una sessione interattiva con il container, è necessario utilizzare Docker exec:

$ docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Per avere un tty:

$ docker exec -it <id_or_name> /bin/bash

Docker run

Esiste anche il comando docker run, che è responsabile sia per la creazione che per l'esecuzione del container Docker (è come fare docker create, start, exec con un comando solo)

Quando si esegue il comando "docker run", Docker cerca l'immagine specificata nel repository locale e la scarica se non è già presente. Quindi, utilizzando l'immagine, Docker crea un nuovo container e avvia il processo specificato all'interno di esso.

In altre parole, il comando "docker run" combina tre azioni:

- Crea un nuovo container
- Avvia il container
- Esegue il comando specificato nel container

Docker Compose

Docker Compose è uno strumento che estende Docker fornendo un modo per definire e gestire applicazioni multi-container. Docker Compose consente di definire un ambiente di sviluppo composto da più container e le relative configurazioni in un file di composizione (solitamente chiamato **docker-compose.yml**). Questo file specifica i servizi, le reti, i volumi e altre configurazioni necessarie per eseguire l'applicazione complessa.

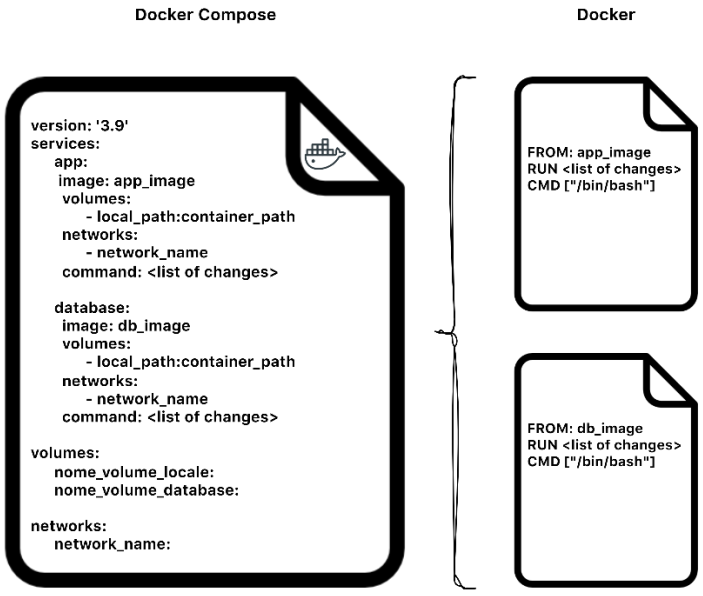
Ad esempio, si immagini di dover sviluppare un'applicazione web composta da un server backend basato su Node.js e un database MongoDB. Senza Docker Compose, bisognerebbe:

1. Avviare manualmente un container per il server Node.js
2. Avviare manualmente un container per il database MongoDB
3. Configurare manualmente la rete e le variabili d'ambiente per consentire al server Node.js di connettersi al database MongoDB.
4. Assicurarti che i container siano avviati nello stesso ordine e che le dipendenze siano soddisfatte.
5. Gestire manualmente l'aggiornamento e il ridimensionamento dei container, se necessario.

Tutti questi passaggi richiedono tempo e sforzo significativi, oltre a poter facilmente portare a errori di configurazione.

Con Docker Compose, invece, si può semplificare l'intero processo utilizzando un file di composizione (ad esempio, **docker-compose.yml**) che definisce i servizi, le reti, i volumi e le configurazioni necessarie. In questo caso, il file di composizione potrebbe contenere definizioni per il server Node.js e il database MongoDB, specificando le dipendenze e le variabili d'ambiente necessarie.

Una volta creato il file docker-compose.yml, basterà eseguire il comando **docker-compose up** e l'intero ambiente verrà configurato e avviato automaticamente. Docker Compose si occuperà di avviare i container nella giusta sequenza, di configurare le reti e le variabili d'ambiente necessarie e di gestire le dipendenze tra i servizi.



Questo scenario dimostra come Docker Compose semplifichi notevolmente la gestione dei container, riducendo la complessità e il lavoro manuale richiesto per configurare e avviare un ambiente di sviluppo composto da più container.

Essendo che docker compose permette di mettere su un ambiente applicativo più complesso, la sua funzione non comprende quella di creare immagini custom. Infatti Docker Compose non consente di creare immagini personalizzate direttamente. Per creare immagini personalizzate, è necessario utilizzare Docker direttamente utilizzando il comando docker build insieme a un Dockerfile. Docker Compose può essere utilizzato per definire la configurazione dei servizi, delle reti e dei volumi che utilizzano le immagini personalizzate create con Docker.

Creare un container con docker compose

Per creare un container (o i container se nel docker-compose.yml sono definiti più servizi), bisogna usare **docker compose create**:

docker compose create [OPTIONS] [SERVICE...]

Come opzioni sono presenti:

- --build: Build images before starting containers.
- --force-recreate: Recreate containers even if their configuration and image haven't changed.
- --no-build: Don't build an image, even if it's missing.
- --no-recreate: If containers already exist, don't recreate them. Incompatible with --force-recreate.
- --pull: Pull image before running ("always"|"missing"|"never")
- --remove-orphans: Remove containers for services not defined in the Compose file.
- --scale: Scale SERVICE to NUM instances. Overrides the scale setting in the Compose file if present.
- --dry-run: Execute command in dry run mode

[SERVICE...] si riferisce ai servizi definiti nel **docker-compose.yml** e per quali di quelli si desidera creare i relativi container. Ad esempio, se sono definiti due servizi nel **docker-compose.yml** chiamati "web" e "database", scrivendo solo docker compose web, si creerà solo il container per il servizio "web". Se ometti l'argomento **[SERVICE...]** nel comando **docker compose create**, Docker Compose creerà i container per tutti i servizi definiti nel **docker-compose.yml**.

Avviare un container con docker compose

Alla fine quando un container è stato creato, sia che sia stato creato con docker che con docker compose, è sempre un container agli occhi di Docker, quindi si può benissimo usare docker start. Il problema è quando si hanno tanti servizi contenuti in container separati. In quel caso si dovrebbe usare N volte il comando docker start. Se i servizi sono stati definiti tutti nello stesso docker-compose.yml allora basta usare il comando **docker compose start**:

docker compose start [SERVICE...]

Elencando i servizi (quindi i container) di cui si vuole avviare il container. Se si omette **[SERVICE...]** verranno avviati tutti i container contenuti nel docker-compose.yml.

Come per docker, esiste un comando che permette sia di creare i container per i servizi che avviarli in un comando solo. Basta utilizzare **docker compose up**:

docker compose up [OPTIONS] [SERVICE...]

Alcune delle [OPTIONS] importanti:

- -d: Detached mode. Run containers in the background
- --no-recreate: If containers already exist, don't recreate them. Incompatible with --force-recreate.

Per avviare invece una sessione interattiva con un container, si usa semplicemente docker exec (esiste anche **docker compose exec**, ma è esattamente identico a docker exec).

PKI

Una public key infrastructure (PKI) è un insieme di ruoli, criteri, hardware, software e procedure necessari per creare, gestire, distribuire, utilizzare, archiviare e revocare certificati digitali e gestire la crittografia a chiave pubblica.

Infrastruttura

Un'infrastruttura a chiave pubblica (PKI) è un sistema che permette la creazione, l'archiviazione e la distribuzione di certificati digitali che vengono utilizzati per verificare che una particolare chiave pubblica appartenga a una determinata entità.

Una PKI è costituita da:

- **Una certification authority (CA)**: archivia, emette e firma i certificati digitali
- **Una registration authority (RA)**: verifica l'identità delle entità che richiedono l'archiviazione dei propri certificati digitali presso la CA
- **Una central directory**: un luogo sicuro in cui le chiavi sono archiviate e indicizzate
- **Un sistema di gestione dei certificati**: gestisce cose come l'accesso ai certificati archiviati o la consegna dei certificati da emettere
- **Una certificate policy**: indica i requisiti della PKI relativi alle sue procedure. Il suo scopo è consentire agli estranei di analizzare l'affidabilità della PKI.

X.509

Le PKI utilizzano lo standard X.509 per la distribuzione delle chiavi pubbliche. X.509 è uno standard ITU (International Telecommunication Union) che definisce il formato dei certificati a chiave pubblica.

Un certificato X.509 lega un'identità a una chiave pubblica utilizzando una firma digitale. Quando un certificato è firmato da un'autorità di certificazione attendibile, qualcuno che detiene quel certificato può utilizzare la chiave pubblica in esso contenuta per stabilire comunicazioni sicure con un'altra parte o convalidare documenti firmati digitalmente dalla chiave privata corrispondente.

Tipi di certificati

Nel sistema X.509 esistono due tipi di certificati:

- certificato CA, che si divide in:
    - Root CA certificate: Il certificato CA autofirmato di primo livello
    - Intermediate CA certificate
- certificato di entità: identifica l'utente finale, come una persona, un'organizzazione o un'azienda. Un certificato di entità finale **non può** emettere altri certificati. Un certificato di entità finale viene talvolta chiamato certificato foglia poiché nessun altro certificato può essere emesso al di sotto di esso.

Struttura del certificato

La struttura prevista dagli standard è espressa in un linguaggio formale, Abstract Syntax Notation One (ASN.1). La struttura di un certificato digitale X.509 (nella versione 3) è la seguente:

- Certificato
    - Numero della versione

    - Numero di serie
    - ID dell'algoritmo di firma
    - Nome dell'emittente
    - Periodo di validità
        - Non prima
        - Non dopo
    - **Nome del soggetto**: il nome del server o dell'entità che il certificato intende autenticare.
    - Informazioni sulla chiave pubblica del soggetto
        - Algoritmo a chiave pubblica
        - Chiave pubblica del soggetto
    - Identificatore univoco dell'emittente (facoltativo)
    - Identificatore univoco del soggetto (facoltativo)
    - **Estensioni** (facoltative)
        - ...
- Algoritmo di firma del certificato
- Firma del certificato

In tutte le versioni di X.509, il numero di serie deve essere univoco per ogni certificato emesso da una specifica CA.

Estensioni

Le estensioni sono state introdotte nella versione 3.

Il campo Estensioni, se presente, è una sequenza di una o più estensioni di certificato. Ogni estensione ha il proprio ID univoco, espresso come identificatore di oggetto (OID), che è un insieme di valori, insieme a un'indicazione critica o non critica. Un sistema che utilizza il certificato deve rifiutare il certificato se incontra un'estensione critica che non riconosce o un'estensione critica che contiene informazioni che non è in grado di elaborare. Un'estensione non critica può essere ignorata se non viene riconosciuta, ma deve essere elaborata se viene riconosciuta.

Le estensioni possono essere classificate come "critiche" o "non critiche" a seconda del loro impatto sulla validità del certificato:

- Le estensioni critiche sono quelle che, se mancanti o non correttamente formattate, rendono il certificato invalido. Ad esempio, l'estensione "Basic Constraints" (limiti di base) è considerata critica, poiché se assente o mal formattata, il certificato potrebbe essere utilizzato per creare certificati intermedi o di CA (Certification Authority), compromettendo la catena di fiducia.
- D'altra parte, le estensioni non critiche sono quelle che, se mancanti o non correttamente formattate, non influiscono sulla validità del certificato. Ad esempio, l'estensione Extended Key Usage (EKU), specifica le finalità per cui il certificato può essere utilizzato (ad esempio, per autenticazione client o server, crittografia e firma digitale). Se mancante o non correttamente formattata, il certificato non è invalido, ma potrebbe non essere compatibile con tutte le applicazioni che richiedono un'indicazione precisa delle finalità del certificato.

SAN

L'estensione SAN (Subject Alternative Name), è un'estensione X.509 che permette di specificare uno o più nomi alternativi a cui il certificato può essere associato, oltre al nome comune (CN) presente nel campo del soggetto (subject) del certificato. Il campo SAN può contenere uno o più nomi alternativi, come ad esempio nomi di dominio aggiuntivi, nomi di host wildcard, indirizzi IP o nomi di dominio di base. Questi nomi alternativi permettono al certificato di essere utilizzato per autenticare diversi nomi di dominio o indirizzi IP, il che può essere particolarmente utile in situazioni in cui un singolo server web deve gestire più domini.

Ad esempio, supponiamo che il server web **myserver.example.com** gestisca anche i domini **myserver.example.org** e **myserver.example.net**. In questo caso, il certificato può includere i nomi alternativi "myserver.example.org" e "myserver.example.net" nel campo SAN, oltre al nome comune "myserver.example.com". Ciò significa che il certificato può essere utilizzato per autenticare il server web in tutti e tre i domini, senza dover creare un certificato diverso per ogni dominio, risparmiando anche soldi.

L'estensione SAN è stata introdotta per risolvere alcuni problemi che sorgono con l'uso del campo del soggetto per specificare i nomi di dominio. Ad esempio, il campo del soggetto supporta solo un singolo nome di dominio.

Alcuni browser Web stanno diventando sempre più rigidi nell'applicare le regole di validazione dei certificati e richiedono l'utilizzo del campo SAN per specificare nomi alternativi. Ormai tutti i browser se un certificato non contiene l'estensione SAN restituiscono un warning. Ad esempio, chrome restituisce:

Chrome – Certificate warning – NET::ERR_CERT_COMMON_NAME_INVALID

Prima della versione 58 di chrome questo non era obbligatorio, ma dopo la versione 58 è stato reso obbligatorio includere nei certifcati il SAN: devblog.

Revoca dei certificati

X.509 definisce anche gli elenchi di revoca dei certificati, che sono un mezzo per distribuire informazioni sui certificati che sono stati ritenuti non validi da una CA.

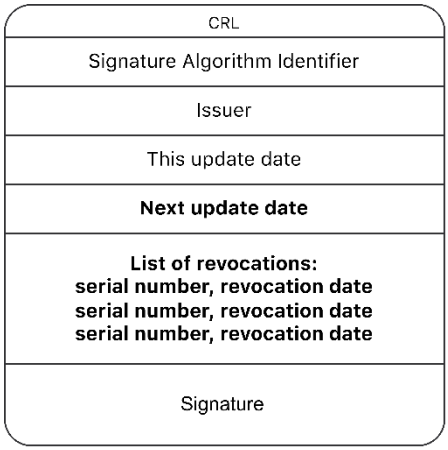
La revoca dei certificati digitali è un'operazione che consente di invalidare un certificato prima della sua data di scadenza. Ciò è necessario quando un certificato non è più sicuro o affidabile a causa di una compromissione della chiave privata, di una violazione della sicurezza o di altre ragioni. La revoca del certificato impedisce l'uso improprio del certificato compromesso e garantisce che le informazioni scambiate tramite il certificato siano protette e affidabili. Senza la revoca, il certificato compromesso potrebbe essere utilizzato per attacchi di tipo "man-in-the-middle" o per altre attività fraudolente.

Ci sono diversi scenari in cui è necessario revocare un certificato. Alcuni di questi includono:

1. **Compromissione della chiave privata**: se la chiave privata associata al certificato viene compromessa o rubata, è necessario revocare il certificato per impedirne l'uso improprio.
2. **Cambio di situazione del titolare del certificato**: se il titolare del certificato cambia la sua situazione, ad esempio, un dipendente lascia un'organizzazione, è necessario revocare il suo certificato per impedire l'accesso ai suoi dati o alle risorse dell'organizzazione.
3. **Violazione della sicurezza**: se un sistema è stato compromesso o attaccato, è possibile che i certificati associati ai sistemi o alle applicazioni coinvolte debbano essere revocati per impedire l'accesso non autorizzato ai dati e alle risorse.
4. **Mancata conformità alle politiche di sicurezza**: se un certificato non è più conforme alle politiche di sicurezza dell'organizzazione, ad esempio se è stato emesso in modo errato o se non rispetta le policy interne dell'organizzazione, potrebbe essere necessario revocarlo.
5. **Debolezza algoritmica**: si scopre che uno standard (tipo SHA1) è diventato vulnerabile e bisogna revocare tutti i certificati che utilizzano quel tipo di standard.

Un meccanismo utilizzato per invalidare un certificato digitale prima della sua data di scadenza è con l'utilizzo di liste di revoca dei certificati (CRL – Certificate Revocation List). Una CRL contiene una lista di certificati digitali revocati insieme alle informazioni sulla loro revoca.

Il processo di revoca con CRL prevede la creazione di una lista di certificati digitali revocati, che viene firmata digitalmente dall'autorità di certificazione (CA). Questa lista viene poi distribuita a tutti gli utenti che devono verificare la validità dei certificati. Quando un utente riceve un certificato digitale, lo confronta con la CRL per verificare che non sia stato revocato.



La lista di revoca dei certificati (CRL) in formato X.509 contiene le informazioni sullo stato di revoca dei certificati digitali emessi da un'autorità di certificazione (CA). Il formato della CRL in X.509 segue uno schema specifico:

- **Signature Algorithm Identifier** (alg, **params**): indica l'algoritmo di firma utilizzato per firmare la CRL, insieme ai relativi parametri.
- **Issuer**: indica il nome dell'autorità di certificazione (CA) che ha emesso la CRL.
- **This update date**: indica la data in cui la CRL è stata emessa o aggiornata per l'ultima volta.
- **Next update date**: indica la data in cui è previsto che la prossima CRL venga emessa o aggiornata.
- **Blocklist**: lista di certificati revocati
    - Serial number, revocation date: elenca il numero di serie e la data di revoca di un certificato digitale specifico.
    - ...
    - Serial number, revocation date: elenca il numero di serie e la data di revoca di altri certificati digitali.

La CRL termina con la firma della CRL, che include l'algoritmo di firma utilizzato, i relativi parametri e la firma stessa.

Il processo di verifica di un certificato digitale tramite una CRL prevede i seguenti passaggi:

1. Il client che riceve il certificato digitale richiede la CRL all'autorità di certificazione o a un distributore di CRL.
2. Il client riceve la CRL firmata digitalmente.
3. Il client verifica la firma digitale per garantire l'autenticità della CRL.
4. Il client cerca il certificato digitale nella CRL e:
    a. Se il certificato digitale è presente nella CRL, viene considerato revocato e non valido.
    b. Se il certificato digitale non è presente nella CRL, viene considerato valido.

Questo metodo è affidabile, ma ha dei problem tra cui:

- **Utilizzo di certificati non validi**: c'è il rischio che i client utilizzino una blocklist obsoleta (per via del next-update) o errata, o che un utente malintenzionato possa compromettere la blocklist e includervi certificati che in realtà sono validi. In tal caso, i client potrebbero rifiutare erroneamente certificati validi, causando problemi di accesso ai servizi protetti da tali certificati.
- **Perdita della funzionalità offline**: se un client configura il proprio software per considerare attendibili solo i certificati quando le CRL sono disponibili, questo significa che il client deve essere sempre connesso alla rete per verificare la validità dei certificati. Ciò rappresenta un problema perché, se il server delle CRL non è disponibile o la connessione Internet del client è interrotta, il client non può accedere ai certificati e non può verificare la loro validità. Questo può impedire al client di accedere a risorse protette da certificati, causando interruzioni nei servizi e nei processi aziendali.
- **Dimensioni e modelli di distribuzione contorti delle CRL**: le CRL possono diventare molto grandi, rendendo difficile per i client scaricarle e verificarle. Inoltre, la distribuzione delle CRL può essere complicata a causa della necessità di sincronizzare gli elenchi di blocchi tra i client e i server.

Per ovviare ai problemi dell'uso delle CLR, tutti i browser moderni adottano un'altra strategia. In particolare utilizzano il **protocollo OCSP**.

OCSP (**Online Certificate Status Protocol**) è un protocollo di rete che consente di verificare lo stato di un certificato digitale in tempo reale. Contrariamente alle CRL, l'OCSP non richiede il download di una lista di certificati revocati, ma consente di verificare la validità di un certificato digitale in tempo reale tramite una richiesta a un server OCSP. Il protocollo OCSP consente a un client di inviare una richiesta per verificare la validità di un certificato digitale a un server OCSP. Il server OCSP risponde con uno stato di risposta che indica se il certificato è stato revocato o è ancora valido. Questo processo di verifica in tempo reale è più efficiente rispetto all'uso di CRL, in quanto elimina la necessità di scaricare, gestire e aggiornare una lista di certificati revocati.

Ci sono diverse ragioni per cui l'OCSP è considerato un miglioramento rispetto alle CRL:

1. **Scalabilità**: l'OCSP è più scalabile delle CRL, in quanto non richiede la gestione di grandi liste di certificati revocati.
2. **Privacy**: l'OCSP non richiede la divulgazione dell'intera lista di certificati revocati, ma solo lo stato del certificato digitale specifico.
3. **Riduzione delle dimensioni dei dati**: l'OCSP trasmette solo lo stato di revoca del certificato richiesto, mentre la CRL richiede la trasmissione di un'intera lista di certificati revocati. Ciò rende l'OCSP più efficiente in termini di banda e meno oneroso in termini di risorse.
4. **Riduzione della latenza**: l'OCSP consente di verificare lo stato di un certificato in tempo reale, mentre con la CRL il tempo di latenza tra l'aggiornamento della lista e la sua distribuzione può essere significativo. Ciò rende l'OCSP più rapido e accurato nella notifica dello stato di revoca del certificato.

È importante notare che alcuni client, come i browser web, possono implementare una cache locale dei risultati di OCSP per ridurre la dipendenza dalla disponibilità del server OCSP. In questo modo, se il server OCSP non è raggiungibile, il client può utilizzare le informazioni di stato memorizzate nella cache locale. Tuttavia, questa cache ha una durata limitata e, se il server OCSP non è disponibile per un lungo periodo, la cache potrebbe scadere e la verifica tramite OCSP potrebbe non essere possibile fino a quando il server non viene ripristinato (come seconda opzione si potrebbe sempre usare le CLR).

Estensioni per i certificati

Esistono diverse estensioni utilizzate per i certificati X.509. Sfortunatamente, alcune di queste estensioni vengono utilizzate anche per altri dati come le chiavi private:

- **.cer, .crt, .der**: certificati in formato binario
- **pem** (Privacy-enhanced Electronic Mail): certificato binario (quindi **.cer**, **.der**) codificati in Base64, racchiuso tra **-----BEGIN CERTIFICATE-----** e **-----END CERTIFICATE-----**
- **.p7b, .p7c**: Struttura PKCS#7
- **.pfx, .p12**: PKCS#12, può contenere certificati (pubblici) e chiavi private (protette da password). **.pfx** è solamente il predecessore di **.p12**.

Ci sono diverse ragioni per cui vengono utilizzati formati di file diversi per l'archiviazione e lo scambio di certificati digitali, chiavi private e pubbliche e altre informazioni crittografiche.

In primo luogo, l'uso di diversi formati di file consente di supportare una vasta gamma di piattaforme e applicazioni. Ad esempio, i file PEM sono comunemente utilizzati su sistemi basati su Unix come Apache, OpenSSL e Nginx, mentre i file PKCS12 sono utilizzati su molte piattaforme, tra cui Windows, macOS e iOS. Utilizzando diversi formati di file, è possibile garantire la compatibilità con una vasta gamma di sistemi e applicazioni.

In secondo luogo, i diversi formati di file offrono diverse opzioni di sicurezza e di flessibilità. Ad esempio, i file PEM sono semplici da leggere e scrivere e possono essere facilmente modificati e manipolati, ma offrono una sicurezza limitata rispetto ai formati binari come DER e PKCS12, che possono essere crittografati e protetti da password.

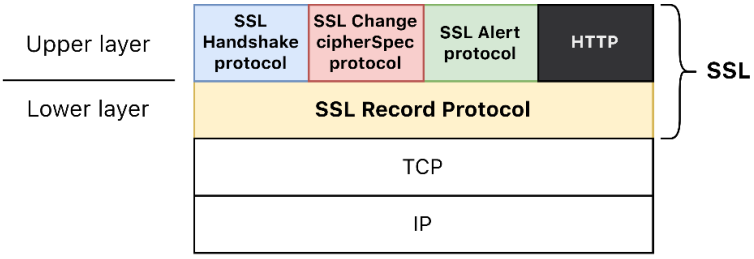
SSL/TLS

Transport Layer Security (TLS) e il suo predecessore Secure Sockets Layer (SSL) sono dei protocolli crittografici di sessione che permettono una comunicazione sicura dalla sorgente al destinatario (end-to-end) su reti TCP/IP fornendo autenticazione, integrità dei dati e confidenzialità operando al di sopra del livello di trasporto.

Nell'utilizzo tipico di un browser da parte di utente finale, l'autenticazione TLS è unilaterale: è il solo server ad autenticarsi presso il client (il client, cioè, conosce l'identità del server, ma non viceversa cioè il client rimane anonimo e non autenticato sul server).

Il protocollo TLS permette anche un'autenticazione bilaterale, tipicamente utilizzata in applicazioni aziendali, in cui entrambe le parti si autenticano in modo sicuro scambiandosi i relativi certificati. Questa autenticazione (definita mutua autenticazione) richiede che anche il client possieda un proprio certificato digitale cosa molto improbabile in un normale scenario.

SSL Architecture



La struttura protocollare è suddivisa in due layer: uno basso ed uno alto.

Il layer basso è composto dal Record Protocol. Questo si occupa della crittografia e della decrittografia dei dati scambiati tra client e server, utilizzando una combinazione di crittografia simmetrica e crittografia asimmetrica.

In particolare, durante la trasmissione, il Record Protocol riceve i dati dal layer SSL più alto, e li suddivide in blocchi di dimensioni predefinite, successivamente applica una sequenza di operazioni di sicurezza su ciascun blocco, incapsula il tutto e passa tutto a TCP. Queste operazioni includono la crittografia, la compressione dei dati (opzionale) e l'aggiunta di un valore di integrità per garantire che i dati non siano stati modificati durante la trasmissione. Lato ricevitore, i dati vengono decrittografati, decompressi (se necessario) e verificati per l'integrità. Se i dati sono stati modificati o corrotti durante la trasmissione, il valore di integrità non corrisponderà e i dati verranno scartati.

Il livello più alto è composto da più protocolli:

- Handshake Protocol:
- Change Cipher Spec protocol:
- Alert Protocol:

Quindi, durante la connessione SSL/TLS, si utilizzerà il protocollo appropriato in base al punto in cui si trova la connessione, ad esempio il protocollo di handshake per negoziare i parametri di sicurezza, il protocollo CipherSpec per specificare gli algoritmi di crittografia e gli altri parametri di sicurezza, e il protocollo di alert per segnalare eventuali errori.

Tutti e tre i protocolli del layer più alto passeranno dal record protocol, che li segmenterà, cifrerà e li autenticherà per poi inviarli a TCP.

Le spiegazioni seguranno l'RFC 6106.

Algoritmi di scambio chiavi supportati

Nel protocollo SSLv3 (SSL 3.0), sono supportati diversi metodi di scambio delle chiavi (key exchange methods):

- RSA
- DH (Fixed Diffie-Hellman)
- DHE (Ephemeral Diffie-Hellman)
- DH_anon (Anonymous Diffie-Hellman)

Il metodo di scambio avviene durante l'handshake e lo sceglie il server tra una serie proposti dal client.

Se il server utilizza **RSA**, semplicemente inoltra al client il proprio certificato firmato da una CA. Il client utilizza la chiave pubblica del server per crittografare una chiave di sessione casuale e inviarla al server. Il server utilizza la sua chiave privata corrispondente per decrittografare la chiave di sessione.

Se il server utilizza **DH**, i parametri pubblici (generatore, radice primitiva e chiave pubblica DH) saranno incorporati all'interno del certificato x509. In questo modo ovviamente i parametri non cambieranno ad ogni sessione ma rimarranno statici (altrimenti si dovrebbe modificare il certificato ad ogni sessione, cosa infattibile senza la CA). Il client che riceve il certificato, autentica i parametri pubblici (sa che è il server a mandarglieli per via del certificato, quindi no MITM) ed invia la propria chiave pubblica cifrandola con la chiave pubblica del certificato del server. Entrambi poi calcolano la chiave di sessione condivisa.

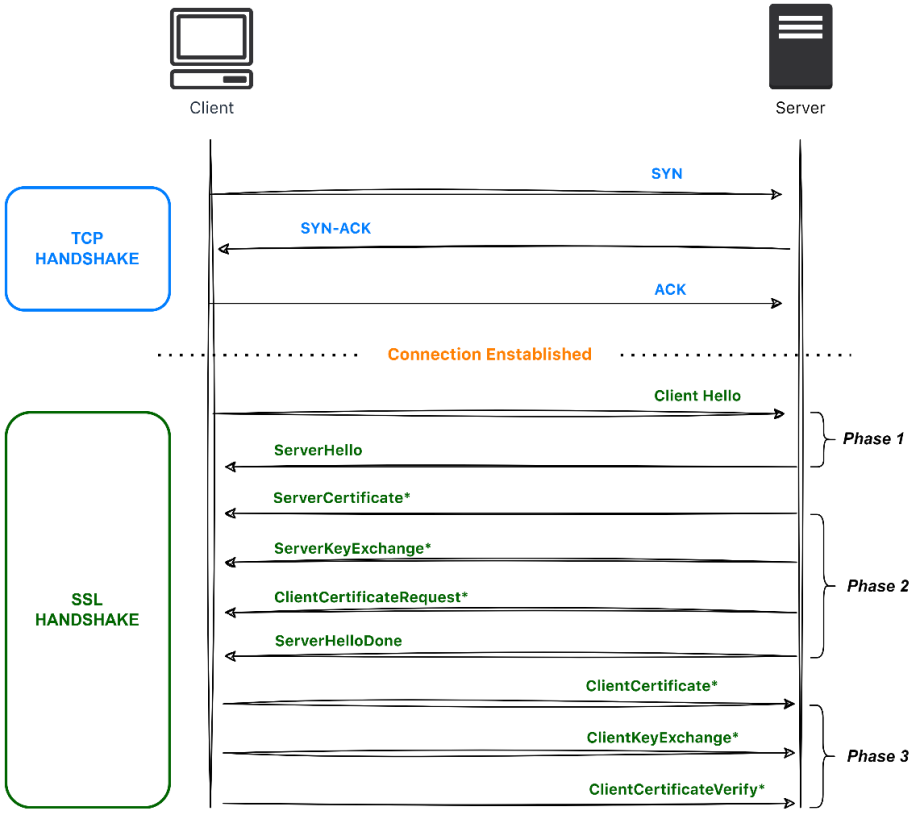
Se il server sceglie di utilizzare **DH_anon**, semplicemente si effettua lo scambio dei parametri pubblici senza autenticazione, quindi senza certificato. In questo modo si può andare in contro a MITM. Il server invia pubblicamente i parametri pubblici. Il client successivamente calcola la propria chiave pubblica e la inoltra indietro al server (sempre senza utilizzare alcuna cifratura o autenticazione). Entrambi poi calcolano la chiave di sessione condivisa.

Per quanto riguardo **DHE**, il server genera parametri Diffie-Hellman effimeri (usa e getta di una singola sessione di durata) e li manda al client firmando la chiave pubblica con la propria chiave privata del proprio certificato RSA. Il client riceve i parametri pubblici, calcola la propria chiave pubblica DH e la inoltra al server. Entrambi poi calcolano la chiave di sessione condivisa. Se è richiesta l'autenticazione del client, anche il client dovrà inviare al server la propria chiave pubblica DH firmata con la chiave privata. Poiché le chiavi pubbliche sono temporanee, una compromissione della chiave a lungo termine del server non compromette la privacy delle sessioni precedenti. Questo è noto come *Perfect Forward Secrecy (PFS)*.

Handshake Protocol

Dopo un handshake TCP, in una sessione SSL/TLS, avviene l'handshake SSL. Quando un client e un server SSL iniziano a comunicare per la prima volta, concordano una versione del protocollo, selezionano algoritmi crittografici, si autenticano facoltativamente a vicenda e utilizzano tecniche di crittografia a chiave pubblica per generare segreti condivisi. Questi processi vengono eseguiti nel protocollo di handshake.

Questo protocollo gestisce anche lo scenario in cui client e il server decidono di riprendere una sessione precedente o duplicare una sessione esistente (invece di negoziare nuovi parametri di sicurezza).



I messaggi contrassegnati con * sono ritenuti opzionali.

Fase 1: Hello messages

I messaggi della fase1 vengono utilizzati per scambiare funzionalità di miglioramento della sicurezza tra il client e il server. Quando inizia una nuova sessione, gli algoritmi di crittografia, hash e compressione di CipherSpec vengono inizializzati su null. L'attuale CipherSpec viene utilizzato per i messaggi di rinegoziazione.

Il messaggio di **HelloRequest** può essere inviato dal server in qualsiasi momento, ma verrà ignorato dal client se il protocollo di handshake è già in corso: dopo aver inviato una HelloRequest, i server non devono elaborare la richiesta fino al completamento della negoziazione attuale. Un client che riceve una HelloRequest mentre si trova in uno stato di negoziazione dell'handshake dovrebbe semplicemente ignorare il messaggio.

La struttura del clientHello è la seguente:

- **ProtocolVersion**: La versione del protocollo SSL con cui il client desidera comunicare durante questa sessione. Questa dovrebbe essere la versione più recente (con il valore più alto) supportata dal client.
- **Random**: Un numero casuale generato dal client più la data UNIX 32bit
- **SessionID**: L'ID di una sessione che il client desidera utilizzare per questa connessione. Questo campo deve essere vuoto (= 0) se non è disponibile alcun SessionID o se il client desidera generare nuovi parametri di sicurezza.

- **CipherSuite[N]**: un elenco delle opzioni crittografiche supportate dal client, ordinate in base alla preferenza del client. Ogni CipherSuite definisce sia un algoritmo di scambio di chiavi (key exchange algorithm) che un CipherSpec (algoritmo di cifratura e di autenticazione). Se il campo **SessionID** non è vuoto (implicando una richiesta di ripresa di sessione), questo campo deve includere almeno il **CipherSuite** di quella sessione
    - Ad esempio, un cipherSuite potrebbe essere: SSL_RSA_WITH_DES_CBC_SHA, in cui è contenuto l'algoritmo di scambio chiavi (RSA), l'algoritmo di cifratura (DES_CBC) e l'algoritmo per l'autentificazione (SHA).
- **CompressionMethod**: un elenco dei metodi di compressione supportati dal client, ordinati in base alle preferenze del client. Se il campo **SessionID** non è vuoto (implicando una richiesta di ripresa di sessione), questo vettore deve includere almeno il **CompressionMethod** di quella sessione. Tutte le implementazioni supportano il valore **null**.

Dopo aver inviato il ClientHello, il client attende un messaggio di ServerHello dal server. Qualsiasi altro messaggio di handshake restituito dal server, viene trattato come un fatal error.

Nota: i dati dell'applicazione non possono essere inviati prima che sia stato concluso l'handshake. È noto che i dati dell'applicazione trasmessi non sono sicuri finché non viene completato l'handshake. Questa restrizione assoluta viene ridotta se è presente una crittografia corrente non nulla su questa connessione (SessionID != 0).

Il server quindi elabora il messaggio di ClientHello del client e risponde con un **ServerHello** oppure un avviso di **HandshakeFailure**.

La struttura del ServerHello è la seguente:

- **ProtocolVersion**: conterrà il protocollo più alto supportato dal server tra quelli suggeriti dal client nel Client.ClientHello.
- **Random**: Numero random generato dal server che deve essere diverso dal Client.random**.**
- **SessionID**: ID della sessione corrispondente a questa connessione. Se ClientHello.SessionID non era vuoto, il server cercherà una corrispondenza nella sua cache di sessione. Se viene trovata una corrispondenza e il server è disposto a stabilire la nuova connessione utilizzando lo stato di sessione specificato, il server risponderà con lo stesso valore fornito dal client. Ciò indica una sessione ripresa e impone alle parti di procedere direttamente ai messaggi finish. In caso contrario, questo campo conterrà un valore diverso che identifica la nuova sessione. Il server può restituire un SessionID vuoto per indicare che la sessione non verrà memorizzata nella cache e quindi non potrà essere ripresa in futuro.
- **CipherSuite**: La singola suite di crittografia, selezionata dal server, dall'elenco in ClientHello.CipherSuite.
- **CompressionMethod:** Il singolo algoritmo di compressione, selezionato dal server, dall'elenco in ClientHello.CompressionMethod.

Fase2: Server Authentication and key exchange

Se il server deve essere autenticato (come generalmente accade), il server invia il suo certificato immediatamente dopo il messaggio di ServerHello nel **ServerCertificate*** message. Il tipo di certificato deve essere appropriato per l'algoritmo di scambio di chiavi della suite di cifratura selezionata, ed è generalmente un certificato X.509.v3. Questo non viene inviato solo nel caso di **DH_anon**

In particolare, nel messaggio **ServerCertificate*** viene inoltrata una **certificate_list[]**: questa è una sequenza (catena) di certificati X.509.v3, ordinati con il certificato del mittente per primo seguito da

eventuali certificati dell'autorità di certificazione che procedono in sequenza verso l'alto. Notare che questo messaggio non viene inviato se il server non ha un certificato. Se si utilizza DH, quello che verrà mandato sarà il certificato che incorpora i parametri pubblici e la chiave pubblica DH.

Dopo aver inviato la certificate list, Il messaggio di **ServerKeyExchange*** viene inviato dal server se e solo se il server non possiede un certificato, oppure se possiede un certificato cui la chiave pubblica al suo interno non può essere utilizzata per cifrare. Il **ServerKeyExchange*** viene inoltrato anche per DH_anon e DHE:

- Per DH_anon in questo messaggio vengono inoltrati i parametri pubblici insieme alla chaive pubblica del server
- Per DHE in questo messaggio vengono inoltrati i parametri pubblici insieme alla chiave pubblica del server cifrata con la chiave privata RSA.

Successivamente, il server può richiedere facoltativamente un **ClientCertificateRequest*** e quindi, richiedere un certificato al client, se appropriato per la suite di cifratura selezionata.

Infine, il messaggio **ServerHelloDone** viene inviato dal server per indicare la fine dei messaggi server. Dopo aver inviato questo messaggio, il server attenderà una risposta del client.

Fase 3: Client Authentication and key exchange

Alla ricezione del messaggio ServerHelloDone, il client deve verificare che il server abbia fornito un certificato valido, se richiesto, e controllare che i parametri del ServerHello siano accettabili.

Il primo messaggio inviato dal client al server è il **ClientCertificate***. Questo messaggio viene inviato se e solo se il server richiede un certificato. Se non è disponibile alcun certificato adatto, il client invia un avviso no_certificate: questo è solo un warning, tuttavia, il server potrebbe rispondere con un avviso di errore di handshake irreversibile se è richiesta l'autenticazione del client. Nota: i certificati Diffie-Hellman del client devono corrispondere ai parametri Diffie-Hellman specificati dal server.

Dopo il clientCertificate, il client inoltra il **ClientKeyExchange*** che dipende dal Key Exchange Algorithm scelto dal server. Per RSA viene inoltrata una **pre_master_key** cifrata con la chiave pubblica del server (in questo caso il certificato del server può essere usato per cifrare). Se si usa DHE, il client invia la propria chiave pubblica DH. Per DH, il client utilizza questo messaggio per inoltrare la propria chiave pubblica DH firmata con la chiave pubblica del certificato del server.

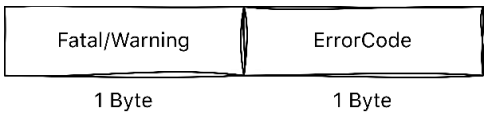
Successivamente, se richiesto dal server (dopo aver inviato il **ClientCertificate***), il client invia al server un messaggio di **ClientCertificateVerify***, che contiene una firma digitale basata sulla chiave privata del client. Il server può verificare la firma utilizzando la chiave pubblica del client contenuta nel suo certificato digitale ricevuto dal messaggio **ClientCertificate***.

Change Cipher Spec Protocol

Il protocollo di modifica delle specifiche di cifratura segnala le transizioni delle strategie di cifratura. Il protocollo consiste in un singolo messaggio, che viene crittografato e compresso sotto il CipherSpec corrente (non in attesa). Il messaggio di modifica viene inviato sia dal client che dal server per notificare all'altra parte che i record successivi saranno protetti con le nuove chiavi e le nuove specifiche di cifratura appena negoziate (durante la fase di handshake). In pratica, il protocollo consiste in un singolo messaggio di un solo byte, il cui valore è 1. La ricezione di questo messaggio fa sì che il destinatario copi lo stato di lettura in sospeso nello stato di lettura corrente.

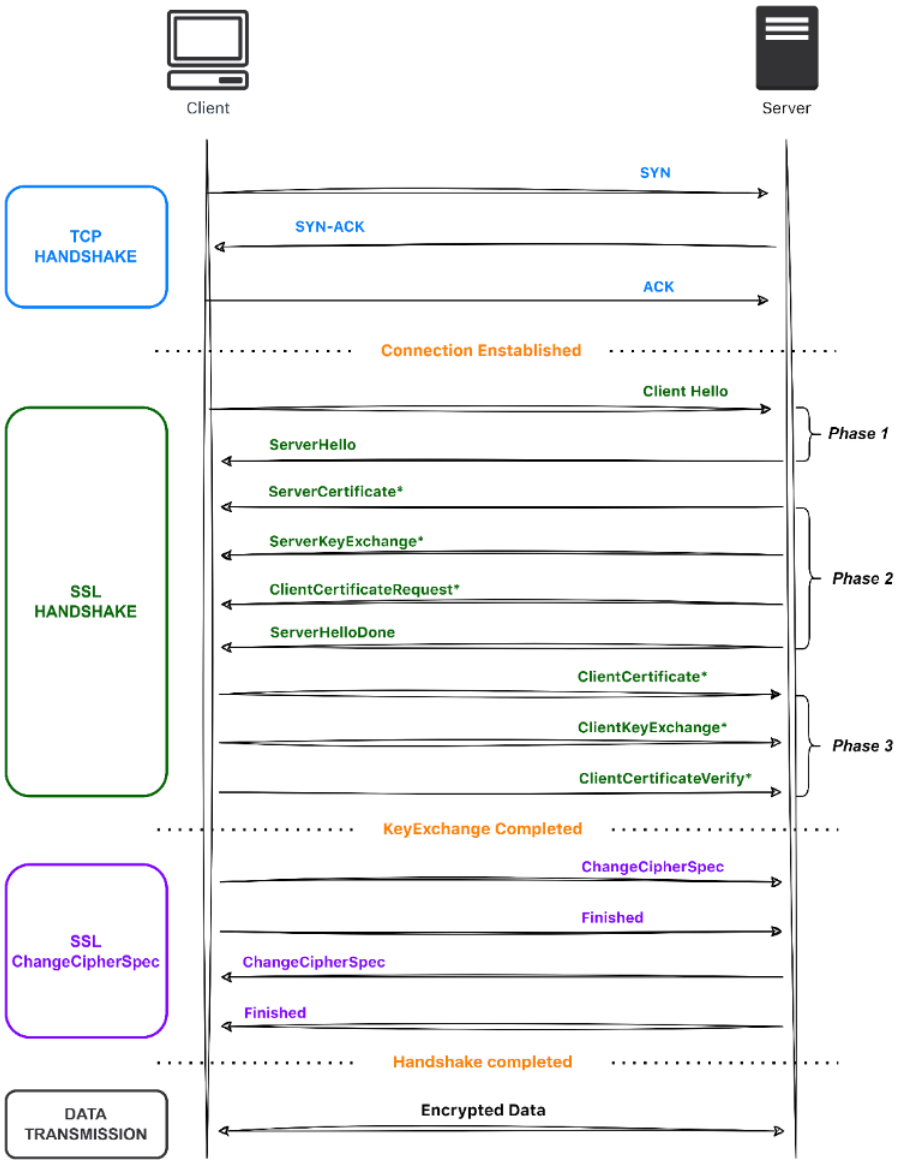
Alert Protocol

L’ultimo protocollo dell’upper layer è il protocollo di alert (avviso). I messaggi di avviso comunicano la gravità del messaggio e una descrizione dell'avviso. Alla trasmissione o alla ricezione di un messaggio di avviso fatale, entrambe le parti chiudono immediatamente la connessione. I server e i client cestinano gli identificatori di sessione, le chiavi e i segreti associati a quella connessione. Come altri messaggi, i messaggi di avviso sono crittografati e compressi, come specificato dallo stato di connessione corrente.



Il primo byte indica il livello di avviso: fatal = 2, warning = 1. Il secondo byte indica il codice dell’errore.

Lo schema di scambio di messaggi completo:



I messaggi finished (di completamento) vengono sempre inviati immediatamente dopo un messaggio di ChangeCipherSpec per confermare che i processi di scambio delle chiavi e di autenticazione abbiano avuto esito positivo. Il messaggio finished è il primo messaggio effettiavamente protetto con gli algoritmi, le chiavi e i segreti appena negoziati. Non è richiesta alcuna conferma del messaggio finished; le parti possono iniziare a inviare dati crittografati immediatamente dopo averlo inviato.

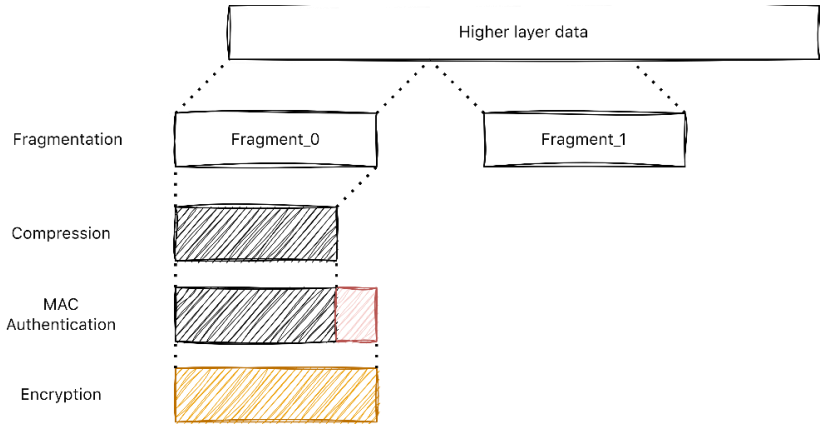
Record Protocol

Quindi ogni protocollo del layer superiore che viene consegnato al layer SSL inferiore (che è il Record Protocol) subisce una serie di operazioni di sicurezza.

Tutti i record sono protetti utilizzando la crittografia e gli algoritmi MAC definito nell'attuale CipherSpec. C'è sempre un attivo CipherSpec; tuttavia, inizialmente è SSL_NULL_WITH_NULL_NULL, che non fornisce alcuna sicurezza.

Il protocollo agisce nel dettaglio in questo modo:

1. Frammenta i dati ricevuti in frammenti da massimo 2^{14} byte (16384 bytes)
2. Ogni frammento (opzionalmente) subisce una compressione loseless. In SSLv3 (come anche TLS 1.3), non è specificato alcun algoritmo di compressione, quindi di default la compressione è null.
3. Una volta applicata la compressione, ad ogni frammento viene concatenato un MAC per l'autenticazione
4. Come ultimo step, il blocco concatenato con il MAC (tutto) viene cifrato



Nel dettaglio, il punto 3 genera il MAC in questo modo:

$$MAC = hash(sh\_key \,||\, pad_2 \,||\, hash(sh\_key \,||\, pad_1 \,||\, seq\_num \,||\, type \,||\, len \,||\, frag))$$

Dove:

- **||:** concatenazione
- **sh_key:** shared secret key
- **hash**: algoritmo di hash (MD5 o SHA)
- **pad₁**: un padding che dipende dall'hash function
  - **Se MD5**: padding sarà il byte 0x36 ripetuto 48 volte (384 bits)
  - **Se SHA**: il padding sarà 0x36 ripetuto 40 volte (320 bits)
- **pad₂**: un padding che dipende dall'hash function
  - Se MD5: il padding sarà il byte 0x5C ripetuto 48 volte

    - Se SHA: il padding sarà il byte 0x5C ripetuto 40 volte
- **seq_num**: il numero di sequenza per questo messaggio
- **type:** il tipo del protocollo del layer superiore che ha processato questo frammento
- **len**: lunghezza del frammento compresso
- **frag:** il frammento compresso (se non è stata applicata compressione, allora il plaintext)

Nel punto 2, la compressione non deve aumentare la lunghezza del frammento di più di 1024 byte.

Quindi la lunghezza massima di un frammento è di massimo 2^{14} + 2048.

Header SSL

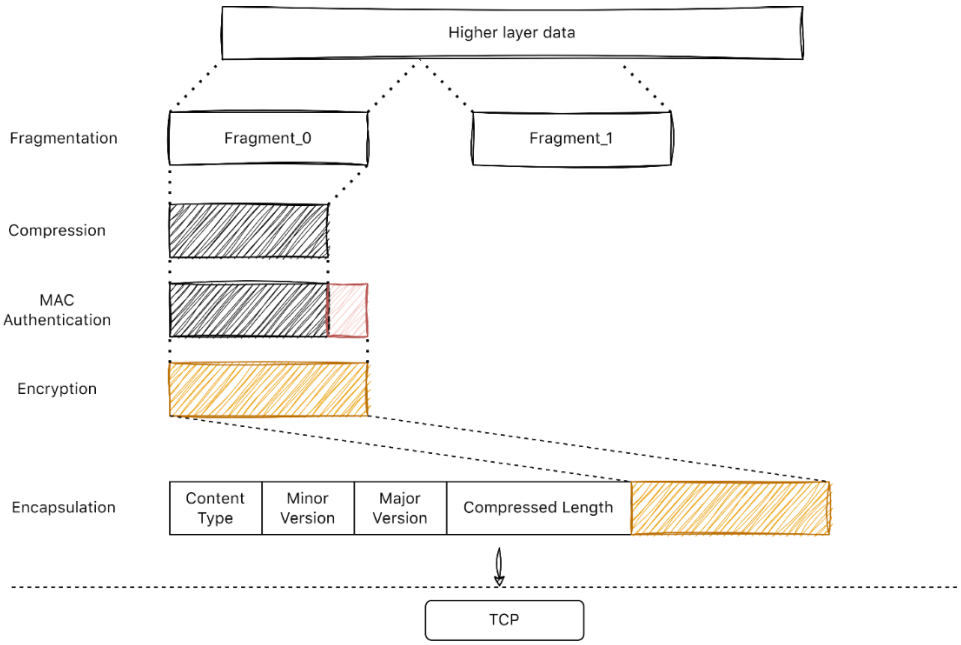
Dopo essere passati dal RecordProtocol, i frammenti cifrati vengono imbustati in un header SLL così composto:

Content Type	Minor Version	Major Version	Compressed Length
8 byte	8 byte	8 byte	16 byte

Dove:

- **ContentType**: Il protocollo di livello superiore utilizzato per elaborare il frammento:
    - **ChangeCipherSpec**: 20
    - **AlertProtocol**: 21
    - **HandshakeProtocol**: 22
    - **Application data**: 23
- **MinorVersion**: La versione minore del protocollo utilizzato (Es: SSL3.1, la minor è 1)
- **MajorVersione**: La versione major del protocollo utilizzato (Es: SSL3.1, la major è 3)
- **CompressedLength**: La lunghezza (in byte) del frammento compresso (non la lunghezza del blocco cifrato insieme al MAC)

Si nota che i dati con ContentType diversi possono essere interlacciati. I dati dell'applicazione hanno generalmente una priorità inferiore per la trasmissione rispetto ad altri tipi di ContentType.

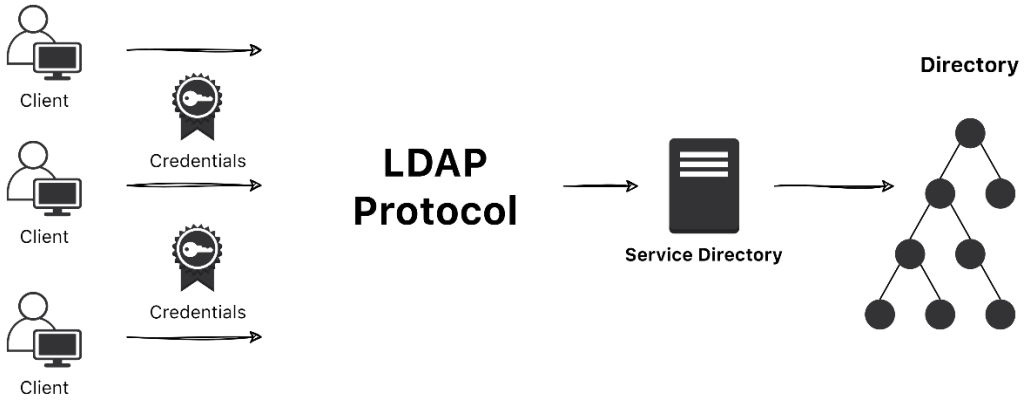


LDAP

LDAP (Lightweight Directory Access Protocol) è un protocollo standard utilizzato per interrogare un Service Directory, come elenchi aziendali di email, rubriche telefoniche o qualsiasi raggruppamento di informazioni che può essere espresso in forma gerarchica. Il protocollo LDAP ha due finalità principali: archiviare i dati nella directory e autenticare gli utenti che accedono alla directory.

Service Directory & Directory

Un Service Directory è un servizio che fornisce una struttura centralizzata per l'archiviazione, l'organizzazione e la gestione delle informazioni relative agli utenti, alle risorse di rete e ad altri oggetti.



La struttura invece che contiene queste informazioni è la directory. LDAP è il protocollo che consente l'accesso delle informazioni nelle directory.

Una directory viene considerata, per analogia, un database organizzato secondo una modalità gerarchica. Si tratta di un particolare tipo di database, un database specializzato, che ha caratteristiche differenti dai tradizionali database relazionali.

Differenza con le basi di dati relazionali

Per prima cosa, una directory, per sua stessa natura, riceve quasi esclusivamente accessi in lettura; la fase di scrittura è limitata agli amministratori di sistema o ai proprietari delle singole informazioni. Per questo motivo, le directory sono ottimizzate per la lettura, mentre i tradizionali database relazionali devono supportare sia la lettura sia la scrittura dei dati (ad esempio in un sistema di prenotazioni aeree o in applicazioni bancarie).

Ne consegue che le directory non sono adatte per immagazzinare informazioni aggiornate di frequente. Ad esempio, il numero di processi in coda in fase di stampa non va memorizzato in una directory, visto che per offrire un dato accurato il numero deve essere aggiornato continuamente. La directory può invece contenere l'indirizzo di una stampante remota che è inserita nel processo di stampa; una volta trovata la stampante con una ricerca nella directory, si può ottenere poi il numero di processi in attesa. L'informazione nella directory (indirizzo della stampante) è statica, mentre il numero richiesto è dinamico e quindi non adatto per stare in una directory.

Un'altra differenza tra directory e database relazionali è che la maggior parte delle implementazioni di directory non supporta le transazioni, ovvero operazioni atomiche.

Un'altra importante differenza sta nel metodo di accesso alle informazioni. La maggior parte dei database relazionali supporta un metodo d'accesso potente e standardizzato che è SQL. SQL permette aggiornamenti e interrogazioni elaborati a discapito delle dimensioni del programma e della complessità dell'applicazione. Le directory invece, in particolare le directory LDAP, usano un protocollo di accesso semplificato e ottimizzato che può essere usato in applicazioni snelle e relativamente semplici. Visto che le directory non hanno l'intento di fornire lo stesso numero di funzioni di un tradizionale database relazionale, possono essere ottimizzate per consentire a più applicazioni di accedere ai dati in grandi sistemi distribuiti nel modo più rapido possibile.

Struttura logica

Nelle configurazioni LDAP, si utilizza una struttura ad albero gerarchica standardizzata chiamata DIT (Directory Information Tree). Un DIT è costituito da dati rappresentati in una struttura gerarchica ad albero indicizzata da nomi (Distinguished Name - DN) delle voci del servizio di directory.

Oggi, la maggior parte delle distribuzioni LDAP seguono una struttura di directory che rispecchia quella del Domain Name System, come descritto da RFC 2247. Ad esempio, la voce per un'organizzazione con nome di dominio "example.com" avrebbe un DN di " dc=example, dc=com" e tutte le voci nell'albero delle informazioni della directory di quell'organizzazione conterrebbero quel suffisso nel Distinguished Name.

Il livello più alto di un albero di un DIT spesso rappresenta il nome di un organizzazione.

Funzionamento

Un client avvia una sessione LDAP connettendosi a un server LDAP, chiamato Directory System Agent (DSA), per impostazione predefinita sulla porta TCP e UDP 389. Il client invia quindi una richiesta di operazione al server e un server invia le risposte in cambio.

Con alcune eccezioni, il client non deve attendere una risposta prima di inviare la richiesta successiva e il server può inviare le risposte in qualsiasi ordine. Tutte le informazioni vengono trasmesse utilizzando la Basic Encoding Rules (BER).

Le operazioni che il client può richiedere sono le seguenti:

- **StartTLS**: utilizza l'estensione TLS - LDAPv3 per una connessione sicura
- **Bind**: autentica e specifica la versione del protocollo LDAP
- **Search**: cerca e/o recupera le voci della directory
- **Compare**: verifica se una voce contiene un determinato valore di attributo
- **Add a new entry**
- **Delete an entry**
- **Edit an entry**
- **Modify Distinguished Name (DN)**: sposta o rinomina una voce
- **Abandon**: annulla una richiesta precedente
- **Extended** Operation: operazione generica utilizzata per definire altre operazioni
- **Unbind**: chiude la connessione (non l'inverso di Bind)

Struttura della directory

Il protocollo fornisce un'interfaccia con le directory che seguono l'edizione 1993 del modello X.500 :

- Una voce consiste in un insieme di attributi.

- Un attributo ha un nome (un tipo di attributo o una descrizione dell'attributo) e uno o più valori. Gli attributi sono definiti in uno schema (vedi sotto).
- Ogni voce ha un identificatore univoco: il suo Distinguished Name (DN). Questo consiste nel suo Relative Distinguished Name (RDN), costruito da alcuni attributi nella voce, seguito dal DN della voce genitore. Pensa al DN come al percorso completo del file e all'RDN come al relativo nome file nella sua cartella principale (ad esempio, se /foo/bar/myfile.txtfosse il DN, allora myfile.txtsarebbe l'RDN).

Un DN può cambiare nel corso della durata della voce, ad esempio, quando le voci vengono spostate all'interno di un albero.

URL LDAP

Come da riferimento nell'RFC 4516, esiste uno schema URL LDAP, che i client supportano e che identifica un'operazione LDAP che solitamente viene utilizzato per effettuare ricerche:

ldap://**host**:**port**/**DN**?**attributes**?**scope**?**filter**?**extensions**

La maggior parte dei componenti descritti di seguito sono opzionali:

- **host:** è l'FQDN o l'indirizzo IP del server LDAP da cercare.
- **port:** è la porta di rete (porta predefinita 389) del server LDAP.
- **DN:** è il distinguished name da utilizzare come base di ricerca.
- **attributes:** è un elenco separato da virgole di attributi da recuperare.
- **scope:** specifica l'ambito della ricerca e può essere "base" (impostazione predefinita), "one" o "sub".
- **filter:** è un filtro di ricerca. Ad esempio, (objectClass=*)come definito in RFC 4515.
- **extensions**: sono estensioni del formato URL LDAP.

Ad esempio:

ldap://**ldap.example.com**/**cn=John%20Doe,dc=example,dc=com**

Questa richiesta di ricerca LDAP specifica il componente "**cn=John%20Doe,dc=example,dc=com**" come base della ricerca (**base DN**), il che significa che la ricerca inizierà da questo oggetto e includerà tutti gli oggetti nella directory sottostanti a questo livello. Tuttavia, poiché non sono stati specificati ulteriori parametri di ricerca come "scope" o "filter", la ricerca avrà lo stesso risultato della richiesta di base, ovvero recupererà solo l'oggetto corrispondente al DN specificato.

ldap://**dc=example,dc=com**??**sub**?**(givenName=John)**

In questo caso, la parte "ldap:///" indica che la richiesta è rivolta al server LDAP locale, senza specificare l'hostname o la porta. La parte "dc=example,dc=com" specifica il base DN, che indica il punto di partenza per la ricerca all'interno della directory LDAP.

La stringa "**??sub?**" dopo il base DN, indica che non sono specificati attributi specifici da recuperare e che la ricerca deve essere eseguita su tutti gli oggetti nella directory sottostante al punto di partenza (cioè, nell'intera sottoalbero). Questo è lo scope di ricerca "sub" (sottoalbero), che è l'impostazione predefinita se lo scope non è specificato.

Infine, la parte "**(givenName=John)**" rappresenta il filtro di ricerca che specifica che la ricerca deve essere limitata agli oggetti che hanno il valore "John" nell'attributo "givenName".

LDAP Scheme

In LDAP, lo schema definisce la struttura e le regole per la definizione, la modifica e la gestione dei dati all'interno di un'istanza di directory. Lo schema specifica quali oggetti possono essere creati all'interno della directory, quali attributi possono essere associati a questi oggetti e quali valori possono avere gli attributi.

Lo schema LDAP è costituito da tre elementi principali:

1. **Oggetti** (**objects**): Un oggetto definisce un elemento che può essere memorizzati nella directory.
2. **Attributi** (**attributes**): gli attributi definiscono le proprietà di un oggetto. Ad esempio, un oggetto "person" può contenere attributi come "cn" (Common Name), "sn" (Surname), "givenName" (Nome), "mail" (Indirizzo email), ecc.
3. **Classi** (**objectClass**): In LDAP, un oggetto è descritto dalla sua classe (objectClass), che definisce l'insieme di attributi che un oggetto può avere. Ogni classe può essere associata a uno o più attributi, che definiscono le proprietà dell'oggetto.

Lo schema LDAP è generalmente descritto utilizzando il formato ASN.1 (Abstract Syntax Notation One), che consente di definire in modo preciso la struttura e la semantica degli oggetti, degli attributi e delle classi nella directory LDAP.

ObjectClass

La classe objectClass è un attributo speciale, presente in tutti gli oggetti LDAP, che descrive la struttura dell'oggetto. Ogni oggetto in LDAP deve avere almeno una classe objectClass definita. Esistono diverse classi predefinite in LDAP, come "person", "group", "organizationalUnit", "domain", "inetOrgPerson", e così via.

Le classi objectClass sono organizzate in una gerarchia ad albero, in cui ogni classe può essere un sottoclasse di un'altra classe. Ad esempio, la classe "organizationalUnit" può essere una sottoclasse di "top", che è la classe di base per tutti gli oggetti in LDAP. In questo modo, è possibile definire classi personalizzate che estendono le classi predefinite per soddisfare le esigenze specifiche dell'organizzazione.

L'attributo objectClass può contenere più di un valore, il che significa che un oggetto può appartenere a più di una classe. Ciò consente di definire oggetti con proprietà multiple e complesse, che possono essere utilizzati per modellare diverse entità all'interno dell'organizzazione.

Search and compare

L'operazione di ricerca viene utilizzata sia per cercare che per leggere le voci. I suoi parametri sono:

- **baseObject**
  - Il nome della voce dell'oggetto di base (o possibilmente la radice) relativa alla quale deve essere eseguita la ricerca.
- **scope**
  - Quali elementi sotto baseObject cercare. Questo può essere:
    - baseObject: cercare solo la voce denominata, in genere utilizzata per leggere una voce
    - singleLevel: voci immediatamente sotto il base DN
    - wholeSubtree: l'intero sottoalbero a partire dal base DN
- **filter**
  - Criteri da utilizzare nella selezione degli elementi all'interno dell'ambito. Ad esempio, il filtro (&(objectClass=person)(|(givenName=John)(mail=john*))) selezionerà

"persone" (elementi di objectClass person) in cui le regole di corrispondenza per givenName e mail determinerà se i valori per tali attributi corrispondono all'asserzione del filtro.

- **derefAlias**
  - Se e come seguire gli alias (voci che fanno riferimento ad altre voci)
- **attributes**
  - Quali attributi restituire nelle voci dei risultati.
- **sizeLimit, timeLimit**
  - Numero massimo di voci da restituire e tempo massimo consentito all'esecuzione per la ricerca. Questi valori, tuttavia, non possono ignorare alcuna restrizione imposta dal server sul limite di dimensione e sul limite di tempo.
- **typesOnly**
  - Restituisce solo i tipi di attributi, non i valori degli attributi.

Il server restituisce le voci corrispondenti e potenzialmente i riferimenti di continuazione. Questi possono essere restituiti in qualsiasi ordine. Il risultato finale includerà il codice del risultato.

L'operazione Compare prende un DN, un nome di attributo e un valore di attributo e controlla se la voce denominata contiene quell'attributo con quel valore.

Fonti: Internet e Wikipedia. lmao.