Programmazione e Sicurezza delle Reti

Alessio Gjergji

Anno accademico 2022 - 2023

Indice

Capitolo 1

Ripasso di Reti di Calcolatori

1.1 Concetti di base

Lo stack **TCP/IP** è composto da vari livelli:

- Applicazione
- Trasporto
- Rete
- Collegamento dati
- Fisico

1.2 Livello di Applicazione

1.2.1 Architetture delle applicazioni di rete

Le architetture delle applicazioni di rete possono essere:

- Client - Server.
- Peer-to-Peer.
- Architetture ibride.

Architettura client-server

Il server è un host sempre attivo, composto da indirizzo IP fisso, mentre il client comunica con il server in qualunque momento esso voglia. Il client può avere IP dinamici e non comunica direttamente con altri client.

Architettura P2P

Tale architettura è caratterizzata da un server non sempre attivo, infatti coppie arbitrarie di host, detti peer, comunicano direttamente tra loro. I peer non devono necessariamente essere sempre attivi, e cambiano indirizzo IP.

1.2.2 Servizi di trasporto di un'applicazione

- Affidabilità: alcune applicazioni possono tollerare qualche perdita di pacchetto, mentre altre non le tollerano.
- Ritardi: alcune applicazioni per essere "realistiche" richiedono bassi ritardi o bassa variazione del ritardo.
- Throughput: alcune applicazioni per essere "efficaci" richiedono almeno una certa capacità del canale.
- Sicurezza: cifratura, integrità dei dati, autenticazione.

1.3 Livello di Trasporto

1.3.1 Socket

Il socket è una tupla che identifica il flusso tra due applicazioni su host anche molto distanti tra loro. Esso è composto da **Source address**, **Destination Address**, **Source Port**, **Destination Port**.

Per Host intendiamo un sistema di elaborazione che ospita applicazioni di rete. Possono essere terminali, e tutti i dispositivi che si collegano in rete. Esse ospitano le applicazioni che utilizzano gli utenti.

L'applicazione di rete è l'interazione di diversi processi, i processi sono programmi (*sequenza di istruzioni*) in esecuzione. Avere due host su internet fa si che essi debbano avere un indirizzo IP che li identifichi e per comunicare si scambiano pacchetti per comunicare. Ai due host potrebbero corrispondere più host e perciò a livello di trasporto viene istituito anche un ulteriore valore chiamato porta.

1.3.2 User Datagram Protocol

Si tratta di un protocollo di trasporto connectionless non affidabile. Esso svolge solo la funzione di indirizzamento delle applicazioni (*porte*).

Permette di costruire applicazioni con sole due primitive **send** e **recive**. A livello IP non sono sicuro che il pacchetto arrivi effettivamente a destinazione, è infatti utilizzato per il supporto di transazioni semplici tra applicativi e per applicazioni multimediali.

1.3.3 Transport Control Protocol

I pacchetti TCP contiene un header più ricco perché si tratta di un protocollo orientato alla connessione dove la perdita di pacchetti implica il rinvio dei pacchetti persi. Il protocollo TCP fornisce affidabilità e consegna ordinata. L'header include:

- Source port - Destination port (*16 bit*): indirizzi della porta sorgente e della porta destinazione.
- Sequence number (*32 bit*): numero di sequenza del primo byte del payload.
- Acknowledge Number (*32 bit*): numero di sequenza del prossimo byte che si intende ricevere (ha la validità di un segmento ACK).
- Offset (*4 bit*): lunghezza dell'header TCP, in multipli di 32 bit.
- Reserved (*6 bit*): riservato per usi futuri
- Window (*16 bit*) ampiezza della finestra di ricezione (comunicato dalla destinazione alla sorgente).
- Checksum (*16 bit*): risultato di un calcolo che serve per sapere se il segmento corrente contiene errori nel campo dati.
- Urgent Pointer (*16 bit*): indica che il ricevente deve iniziare a leggere il campo dati a partire dal numero di byte specificato. Viene usato se si inviano comandi che danno inizio ad eventi asincroni "urgenti".
- Flag (*1 bit*):
  - URG: vale uno se vi sono dati urgenti; in questo caso il campo urgent pointer ha senso.
  - ACK: vale uno se il segmento è un ACK valido; in questo caso l'acknowledge number contiene un numero valido.
  - PSH: vale uno quando il trasmettitore intende usare il comando di PUSH.
  - RST: reset; resetta la connessione senza un tear down esplicito.
  - SYN: synchronize; usato durante il setup per comunicare i numeri di sequenza iniziale.
  - FIN: usato per la chiusura esplicita di una connessione.
- Option and Padding (*lunghezza variabile*): riempimento (fino a multipli di 32 bit) e campi opzionali come ad esempio durante il setup per comunicare il MSS.
- Data: i dati provenienti dall'applicazione.

Con il TCP avremo quindi 4 chiamate a funzione.

- Apertura della connessione.
- Send.

- Recive.
- Chiusura della connessione.

Instaurazione della connessione

Il TCP è un protocollo **connection oriented**, ciò significa che prima di iniziare a trasferire i dati ci si deve essere una connessione tra due end-system.

L'instaurazione della connessione avviene secondo la procedura "three-way handshake"

- la stazione che richiede la connessione (A) invia un segmento SYN.
- la stazione che riceve la richiesta (B) risponde con un segmento SYN.
- la stazione A riscontra il segmento SYN della stazione B.

Terminazione della connessione

Poiché la connessione è bidirezionale, la terminazione deve avvenire in entrambe le direzioni.

- la stazione che non ha più dati da trasmettere e decide di chiudere la connessione invia un segmento FIN (segmento con il campo FIN a 1 e il campo dati vuoto).
- la stazione che riceve il segmento FIN invia un ACK e indica all'applicazione che la comunicazione è stata chiusa nella direzione entrante.
- Per chiudere completamente la connessione, la procedura di half close deve avvenire anche nell'altra direzione.

Stima del Round Trip Time

Poiché l'RTT può variare anche molto in base alle condizioni della rete, il valore di RTT (SRTT Smoothed RTT) utilizzato per il calcolo di RTO risulta una stima del valor medio di RTT sperimentato dai diversi segmenti.

$$SRTT_{attuale} = (\alpha \cdot SRTT_{precedente}) + ((1 - \alpha) \cdot RTT_{istantaneo})$$

Dove:

- $RTT_{istantaneo}$: misura di RTT sull'ultimo segmento.
- $RTT_{precedente}$: stima precedente del valore medio di RTT.
- $SRTT_{attuale}$: stima attuale del valore medio di RTT.
- α: coefficienti e peso compreso tra zero e uno.

Stima del Retrasmition Time Out

$$RTO = \beta \cdot SRTT$$

La sorgente, dunque, attende fino a due volte il RTT medio (SRTT) prima di considerare il segmento perso e ritrasmetterlo.

In caso di ritrasmissione, il RTO per quel segmento viene ricalcolato in base ad un processo di exponential backoff.

1.3.4 Interazione tra Router e Forwarding

La rete internet è tenuta insieme dai router, i router fanno due operazioni, il routing e il forwarding.

Il routing è il processo di scoperta del cammino da una sorgente ad ogni destinazione nella rete. Un protocollo di routing gestisce una tabella di routing nei router, la tabella indica, per ogni destinazione, qual è l'interfaccia di uscita su cui inviare il pacchetto. Gli algoritmi utilizzati per risolvere tale problema sono:

- Distance Vector: periodicamente ogni nodo invia ai propri vicini il vettore delle distanze verso tutte le sottoreti del mondo.
- Link State: periodicamente ogni nodo invia a tutte le sottoreti del mondo lo stato dei collegamenti verso i propri vicini.

Il forwarding consiste nel scegliere l'interfaccia di uscita adatta sulla quale incanalare il pacchetto.

1.4 Livello di Rete

1.4.1 Gerarchia degli indirizzi IP

Gli indirizzi IP sono divisi in due parti:

- Un prefisso: identifica la rete fisica a cui l'host è connesso, anche noto come **NetID**. A ciascuna rete in Internet viene assegnato un prefisso di rete univoco.
- Un suffisso: identifica un host specifico all'interno di una rete, anche noto come **HostID**. A ciascun host su una data rete viene assegnato un suffisso univoco mentre il prefisso è uguale per tutti.

Lo schema degli indirizzi IP garantisce due proprietà:

- A ciascun host viene assegnato un indirizzo unico.
- L'assegnazione dei numeri di rete (prefissi) viene coordinata globalmente, mentre i prefissi vengono assegnati localmente.

Subnet e indirizzamento classes

I prefissi sono di lunghezza fissa ed hanno indirizzamento classful (*4 classi*). Tale schema, con l'espansione di internet si è rilevato limitante, siccome tutti richiedevano l'utilizzo di indirizzi di classe A e B in modo da poter avere un numero sufficiente di indirizzi per eventuali espansioni. La conseguenza fu il sottoutilizzo di indirizzi all'interno di ogni classe.

1.4.2 Notazione Classless-Domain Routing

L'utilizzo di maschere per specificare la dimensione del prefisso viene fatta per questioni di efficienza, poiché le operazioni di AND bit a bit sono molto veloci in hardware.

Tuttavia, per facilitare la gestione da parte degli utenti, si utilizza una notazione più semplice e diretta, specificando la dimensione del prefisso.

$$ddd.ddd.ddd.ddd/m$$

dove:

- *ddd* è il valore decimale nella notazione puntata.
- m è il numero di bit del prefisso.

Esempio

- NetA: 128.10.0.0/16
- NetB: 64.10.2.0/24

1.4.3 Interfacce di rete

Gli indirizzi non sono dati alle macchine, ma alle interfacce, quindi su un dispositivo possono essere molteplici.

1.5 Livello di Collegamento Dati

Le principali funzioni svolte dal livello 2 sono:

- Framing: delimitazione dei messaggi (*frame*) come gruppi di bit sul canale fisico.
- Rilevazione/gestione errori: controlla se il frame contiene errori ed eventualmente gestisce il recupero.
- Metodo di accesso al canale.
- Controllo di flusso: gestisce la velocità della trasmissione.

1.5.1 Framing

Il livello 2 riceve dal livello superiore (*rete*) dei pacchetti. Considerando che:

- La lunghezza dei pacchetti (*di livello 3*) e delle corrispondenti trame (*livello 2*) è variabile.
- I sistemi non sono sincronizzati tra loro.
- Il livello 1 tratta solo bit, e quindi non è in grado di distinguere se un bit appartiene
- ad una trama o a quella successiva.

Nasce quindi il problema della **delimitazione dei frame**. La funzionalità di framing è dunque rendere distinguibile un frame dall'altro attraverso l'utilizzo di opportuni codici all'inizio e alla fine del frame stesso.

1.5.2 Metodo di accesso al canale

La modalità di trasmissione di distingue in:

- Punto - punto: dove la trasmissione è limitata ai dispositivi.
- Broadcast: dove la limitazione non è più presente.

La metodologia Broadcast richiede l'implementazione di tecniche di allocazione:

- Statica: TDM, FDM, CDM, ...
- Dinamica.

L'allocazione dinamica richiede la suddivisione in due categorie nella tecnica di assegnazione:

- A turno: token ring, token pass, ...
- A contesa: Aloha, Slotted Aloha, CSMA, CSMA-CD, ...

Capitolo 2

Dal Web ai Webservices

2.1 HTTP/HTTPS

I protocolli `HTTP` e `HTTPS` è nato per la fruizione dei contenuti in rete (*World Wide Web*). Oggi è utilizzato anche per l'invocazione di funzionalità remote, i **webservice**.

Il client si chiama "web browser" (*o semplicemente "browser"*), il server si chiama "we server" e la comunicazione avviene sul protocollo TCP sulla rete internet.

L'apertura della comunicazione avviene attraverso gli attori *client* e *server* identificati rispettivamente da indirizzo ip e porta (*l'`HTTP` utilizza la porta* 80*, l'`HTTPS` utilizza la porta* 443). L'`HTTP` viaggia sul protocollo `TCP` e la prima cosa da eseguire è l'apertura della connessione (*attraverso il tree way handshake*). Una volta effettuata l'apertura della connessione si identifica da soli richieste seguite da risposte. Nel caso di `HTTPS` avviene l'autenticazione del server e la negoziazione di una chiave di cifratura.

Si tratta di un protocollo testuale, leggibili.

Esempio

```
GET /it/i-nostri-servizi/servizi-per-studenti HTTP/1.1
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8)
Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Esempio di risposta

```
HTTP/1.1 200 OK
Date: Mon, 17 May 2022 16:10:48 GMT
```

```
Server: Apache
Last-Modified: Mon, 29 Mar 2022 13:57:17 GMT
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<html>
   ...
</html>
```

2.1.1 Metodi di richiesta `HTTP`

- `GET`: imposta i parametri nella `URL`, con il limite di 2048 caratteri.
- `POST`: imposta i dati nel corpo della richiesta, dove non c'è limite di caratteri.
- `PUT`
- `DELETE`
- `HEAD`
- `CONNECTION`
- `OPTIONS`
- `TRACE`
- `PATCH`

2.1.2 Il protocollo `HTTPS`

I messaggi che passano nella connessione `TCP` sono gli stessi dell'`HTTP` ma vengono sottoposti a cifratura dei dati in transito e autenticazione del server mediante certificato digitale.

Il server lavora sulla porta 443 invece che 80.

2.1.3 Uniform Resource Locator `URL`

Detto anche Universal Resource Locator, si tratta di una stringa che permette di identificare in maniera univoca una risorsa `HTTP` in qualsiasi parte della rete mondiale.

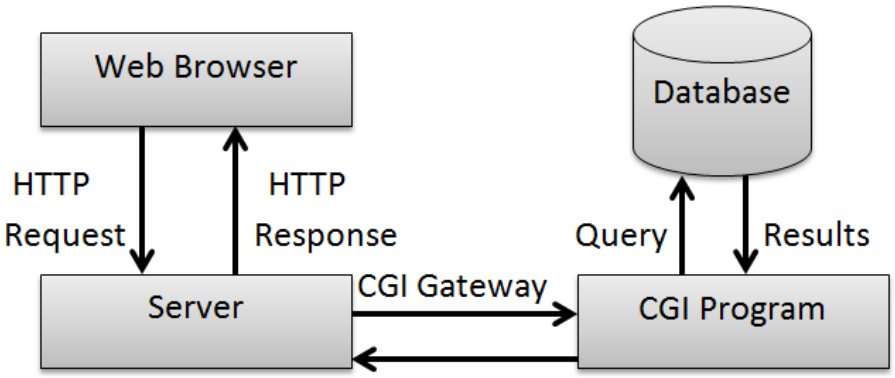
$$https://www.univr.it/servizi/studenti/carriera$$

La struttura:

- Il protocollo utilizzato a livello applicazione, ovvero il protocollo di livello di trasporto più la porta utilizzata;
- Nome dell'host (*o indirizzo* `IP`) che eroga tale risorsa.
- Nome della risorsa con il suo percorso logico completo.

La porta può essere indicata esplicitamente se non è quella standard.

2.2 Common Gateway Interface (CGI)



2.2.1 CGI e web dinamico

Il risultato della richiesta del browser non è più un documento statico, il programma CGI viene eseguito lato server e il browser web diventa il client di molte applicazioni di rete, come posta elettronica, Wiki e Content Management Systems.

2.2.2 Web e posta elettronica

Quando viene eseguita una richiesta GET da parte del client web verso il server il server effettua internamente una richiesta SMTP in direzione del server di posta. Il server di posta genererà la SMTP response e in base a tale risposta, il server web genererà una HTTP response.

2.3 Web Socket

Si tratta di un un protocollo di comunicazione alternativo a HTTP/HTTPS, ammette la comunicazione simmetrica tra i processi interagenti. Essa nasce da HTTP/HTTPS attraverso una operazione di **protocol upgrade**, da quel momento i processi possono prendere l'iniziativa per mandare dei dati dall'altra parte. È utilizzato per il refresh asincrono di una pagina web attraverso push di una nuova pagina da parte del server.

2.3.1 Protocol upgrade

I messaggi contengono due nuove stringhe che identificano l'upgrade del protocollo, come segue:

```
GET / HTTP/1.1
Host: server.example.com
Upgrade: websocket
```

```
Protocol: upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Version: 13
```

2.4 Service-oriented architecture

Le tradizionali applicazioni "monolitiche" sono costituite da l'interfaccia utente che richiama delle funzionalità fornite da una serie di **librerie** linkate in un unico programma che "gira" sulla macchina dell'utente.

Le SOA hanno l'obiettivo di realizzare applicazioni complesse attraverso la **combinazione di diversi programmi attraverso la rete**.

- L'interfaccia utente più qualche funzionalità di base "girano" sull'host dell'utente.
- Le funzionalità principali dell'applicazione sono fornite da programmi che girano su uno o più server.

2.4.1 Vantaggi

- Potenza di calcolo e memoria sono delegate ai server;
- Protezione della proprietà intellettuale su algoritmi strategici;
- Annullamento della necessità di distribuire aggiornamenti del software ; quando le modifiche riguardano solo il codice dei server;
- Nuovo modello economico: pay per use;
- Eliminazione della pirateria;
- L'infrastruttura di rete diventa un elemento essenziale;
- Se "manca la rete" l'applicazione non funziona.

2.4.2 SOA: tecnologie

Il servizio, ovvero la chiamata a funzione (*per gli ambienti orientati agli oggetti*), è il cuore delle SOA.

I generici componenti di una funzione sono:

- Nome: descrizione della funzione;
- Tipo e numero di parametri in ingresso;
- Tipo di valore restituito;
- Implementazione.

L'**interfaccia** è la descrizione di nome, parametri e valore di ritorno delle funzioni erogate da una libreria che è locale in caso di applicazioni tradizionali oppure remota nel caso delle SOA.

Le Application Program Interface, ovvero le API, sono l'insieme delle funzioni/metodi esposte da una certa libreria locale o da un server remoto. Il cuore di SOA è la **chiamata di funzione remota** perfettamente conforme al modello client/server.

2.4.3 Chiamata di funzione remota

Il componente server espone una API che descrive una serie di funzioni che il componente client (*che non è un web browser*) può invocare.

L'implementazione delle funzioni è presente nel server, il codice che chiama la funzione sul client e quello che implementa la funzione sul server possono essere scritti con linguaggi differenti e girare su architetture di calcolo molto differenti, ma è importante che **entrambe le funzioni implementano la stessa interfaccia**.

La funzione chiamata dal client apparentemente realizza la funzionalità; in realtà codifica i parametri per essere trasmessi in rete e decodifica il valore di ritorno, tale funzione si chiama **stub**.

Sul server esiste un componente chiamato **skeleton** che decodifica i parametri di input, li passa alla funzione che contiene l'implementazione vera e propria (*business logic*), prende il risultato, lo codifica e spedisce al client.

Il meccanismo utilizza il protocollo di rete per il trasporto dei dati codificati e tale codifica a volte si definisce come serializzazione.

2.4.4 Teconologie per la chiamata di funzione remota

- Remote procedure call (RPC): C su TCP;
- Java Remote Method Invocation (JAVA RMI): Java su TCP;
- Common Object Request Broker Architecture (CORBA): standard indipendente dai linguaggi di programmazione e dal protocollo di livello trasporto;
- **Webservice**: utilizza l'HTTP/HTTPS come protocollo di trasporto degli elementi della funzione, e il formato dei dati può utilizzata è la metodologia REST.

2.4.5 Webservice basati su REST

Viene eseguito un mapping del nome della funzione sulla URL. Il passaggio di parametri sulla URL utilizza i metodi GET e DELETE oppure dopo l'header attraverso POST e PUT.

La scelta del metodo HTTP in funzione della semantica della funzione, avviene seguendo tali criteri:

- POST: usato per funzioni che creano un nuovo oggetto sul server;
- PUT: funzioni che aggiornano un oggetto esistente sul server;

- `GET`: funzioni che recuperano info di un oggetto esistente sul server;
- `DELETE`: funzioni che distruggono un oggetto esistente sul server.

Il valore di ritorno nel corpo della risposta avviene sostituendo l'`HTML` con testo puro o oggetti in formato `JSON`.

2.4.6 Webservice: vantaggi

L'infrastruttura Internet è già predisposta all'uso di `HTTP/HTTPS`. L'uso di contenuti testuali nelle transazioni facilita il debugging delle applicazioni `SOA`.

2.5 Cloud computing

Per cloud computing si intende un gruppo di host in rete che forniscono servizi di calcolo e memorizzazione senza essere indirizzati o gestiti idividualmente dagli utenti.

L'intero insieme di hardware e software è gestito dal fornitore del cloud che si fa pagare con una politica a consumo oppure forfettaria.

2.5.1 Cloud: servizi offerti

- On-site: la gestione dell'hardware viene fatta dall'utente;
- Infrastructure as a Service (`IaaS`): affitto una macchina virtuale su cui installo il mio sistema operativo e sopra tutte le mie applicazioni;
- Platform as a Service (`PaaS`): affitto l'uso di un ambiente in cui c'è già il sistema operativo e posso installare le mie applicazioni;
- Software as a Service (`SaaS`): affitto l'uso di applicazioni già installate.
- Function as a Service (`FaaS`): possibilità di creare direttamente delle funzioni.

Capitolo 3

Programmazione di rete client/server mediante interfaccia socket

3.1 Applicazioni client server

I client e i server possono essere molteplici, questo protocollo è contraddistinto dal client che esegue il primo passo. Client e server sono processi e non host, l'insieme di almeno un client e un server costituisce l'applicazione di rete.

Per testare non serve necessariamente una connessione alla rete ma si può usare una rete fittizia chiamata loopback. In tal caso l'interfaccia socket sia del client sia del server hanno indirizzo IP 127.0.0.1 chiamata **localhost**.

Le interfacce create da parte del client e del server devono essere create su porte diverse e il client deve conoscere la porta del server.

3.2 Come scrivere applicazioni di rete

3.2.1 Creazione dell'interfaccia socket

Tutti i sistemi operativi forniscono un'interfaccia software per costruire questi programmi con un qualsiasi linguaggio di programmazione. Questa interfaccia si chiama **socket**.

Il programma prima di usare la rete deve creare un oggetto di tipo socket definito da tre parametri:

- Indirizzo `IP` locale;
- Porta locale (*si aggancia al processo in esecuzione*);
- Modalità di trasmissione: `UDP` oppure `TCP`.

Il server deve decidere esplicitamente il numero di porta locale affinché i client possano saperlo. Gli indirizzi compresi tra 0 e 1023 sono riservati a protocolli applicativi noti, in questo caso il programma che crea un oggetto socket su queste porte deve avere i privilegi di root.

Il programma **client** può deciderlo esplicitamente oppure lasciarlo decidere al proprio sistema operativo.

3.2.2 Comunicazione mediante interfaccia socket

La comunicazione tra i due processi avviene conoscendo due ulteriori parametri:

- `IP` del proprio interlocutore;
- Porta del proprio interlocutore.

Quindi il client rende disponibile il proprio `IP` e la propria porta solamente durante la connessione, in modo che il server possa comunicare con tale client per soddisfare le richieste.

3.2.3 Applicazioni orientate ai datagrammi: `UDP`

Ogni pacchetto scambiato tra gli host è **logicamente indipendente** dai precedenti e successivi. In tal caso, le perdite di pacchetti non vengono compensate in automatico nè dalla rete nè dal sistema operativo. Per questo motivo il protocollo `UDP` è più leggero del `TCP` come uso di risorse di calcolo in rete, ma il tipo di applicazione deve essere compatibile con il tipo di comportamento.

La trasmissione in questo caso segue 2 punti:

- Chiamata a funzione per trasmettere;
- Chiamata a funzione per ricevere.

3.2.4 Applicazioni orientate alla trasmissione `TCP`

In questo caso l'oggetto socket si dedica alla numerazione dei pacchetti appartenenti alla stessa connessione per rilevare eventuali pacchetti persi e per poterli ritrasmettere.

La trasmissione in questo caso segue 4 punti:

- Chiamata a funzione per aprire la connessione;
- Chiamata a funzione per trasmettere;
- Chiamata a funzione per ricevere;
- Chiamata a funzione per chiudere la connessione.

Capitolo 4

Sicurezza delle reti

Protegge significa garantire: confidenzialità, integrità, disponibilità l'autenticità e la tracciabilità.

4.1 Confidenzialità

Per confidenzialità si intende che nessun utente deve poter ottenere o dedurre che non è autorizzato a conoscere.

- Riservatezza dei dati: le informazioni confidenziali non devono essere rilevate o rilevabili da utenti non autorizzati.
- Privacy: l'utente controlla o influenza quali informazioni possono essere collezionate e memorizzate

4.2 Integrità

Integrità significa impedire l'alterazione diretta o indiretta delle informazioni, sia da parte di utenti e processi non autorizzati, che a seguito di eventi accidentali. Se i dati vengono alterati è necessario fornire strumenti per poterlo verificare facilmente.

- Integrità dei dati: Le informazioni e i programmi possono essere modificati solo se autorizzati.
- Integrità del sistema: Il sistema funziona e non è compromesso.

4.3 Disponibilità

Per disponibilità si intende Rendere disponibili a ciascun utente abilitato le informazioni alle quali ha diritto di accedere, nei tempi e nei modi previsti in determinate condizioni, in un preciso istante, in un intervallo di tempo. Nei sistemi informatici, i requisiti di disponibilità includono prestazioni e robustezza.

4.4 Autenticità

Ciascun utente deve poter verificare l'autenticità delle informazioni: messaggi, mittenti, destinatari.

Si richiede di poter verificare se una informazione è stata manipolata, anche informazioni non riservate.

4.5 Tracciabilità

Le azioni di un'entità devono essere tracciate in modo univoco in modo tale da supportare la non-ripudiabilità e l'isolamento delle responsabilità. Ad esempio nessun utente deve poter ripudiare o negare in tempi successivi messaggi da lui spediti o firmati.

4.6 In che modo le risorse sono minacciate

Le minacce compromettono le proprietà di confidenzialità, integrità e disponibilità.

Una minaccia è una possibile violazione, mentre l'attacco è l'effettiva violazione.

Gli attacchi possono essere:

- Attivi: tentativi di alterare le risorse o modificare il funzionamento dei sistemi.
- Passivi: tentativi di capire informazioni e utilizzarle senza intaccare le risorse.
- Interni: iniziati da un'entità al sistema (*dipendente che porta la chiavetta*).
- Esterni: esterni da un'entità esterna, tipicamente attraverso la rete.

4.6.1 Classi di minacce

- Disclosure: accesso non autorizzato alle informazioni.
- Deception: accettazione di dati falsi.
- Disruption: interruzione o prevenzioni di operazioni corrette.
- Usurpation: Controllo non autorizzato di alcune parti del sistema.

4.7 Cosa bisogna fare per contrastare le minacce?

La complessità della sicurezza è in questa domanda, non esiste un risposta unica, le risposte cambiano nel tempo.

Le risorse da proteggere sono sistemi composti da sotto-sistemi. La sicurezza dipende non solo da algoritmi o protocolli, ma anche dagli utenti.

Capitolo 5

Crittografia

5.1 Introduzione

Si tratta di una scienza che si occupa di proteggere l'informazione rendendola sicura, in modo che un utente non autorizzato che ne entri in possesso non sia in grado di comprenderla.

La crittoanalisi è invece la scienza che cerca di aggirare o superare le protezioni crittografiche, accedendo alle informazioni protette.

L'insieme di crittografia e crittoanalisi è detto crittologia.

5.2 Algoritmo crittografico

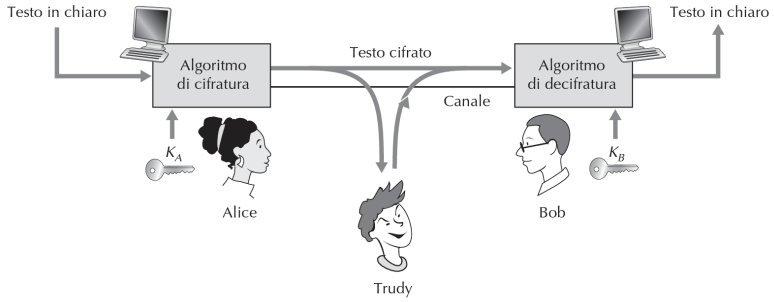
Si tratta di una funzione che prende un messaggio e un parametro detto **chiave**, e produce in uscita un messaggio trasformato.

Cifratura

Testo in chiaro (*plaintext o cleartext*) trasformato in testo cifrato (*ciphertext*).

Decifratura

Testo cifrato che viene trasformato in testo in chiaro.



5.2.1 Elementi del processo crittografico

Se le chiavi di cifratura e decifratura sono uguali: ha algoritmo simmetrico e la chiave deve essere segreta. Se le chiavi sono diverse: l'algoritmo è asimmetrico e una chiave è pubblica, l'altra privata (*e quindi segreta*).

Robustezza crittografica

Un algoritmo di cifratura è robusto se non è facilmente possibile:

- Dato un testo cifrato ottenere il corrispondente testo in chiaro senza conoscere la chiave di decifratura.
- Dato un testo cifrato e il corrispondente testo in chiaro ottenere la chiave di decifratura.

In generale, nessun algoritmo crittografico è assolutamente sicuro, quindi si dice che è computazionalmente sicuro se:

- Il costo necessario a violarlo è superiore al valore dell'informazione cifrata.
- Il tempo necessario a violarlo è superiore al tempo di vita utile dell'informazione cifrata.

5.3 Crittoanalisi

La crittoanalisi tenta di ricostruire il testo in chiaro senza conoscere la chiave di decifratura. L'attacco più banale è quelli **a forza bruta**:

- Tentare di decifrare il messaggio provando tutte le chiavi possibili.
- Applicabile a qualunque algoritmo, ma la sua praticabilità dipende dal numero di chiavi possibili.
- È comunque necessario avere informazioni sul formato del testo in chiaro, per riconoscerlo quando si trova la chiave giusta.

Principio di **Kerckhoffs**:

- Nel valutare la sicurezza di un algoritmo crittografico si assume che il crittoanalista conosca tutti i dettagli dell'algoritmo.

- La segretezza deve risiedere nella chiave, non nell'algoritmo!

5.4 Crittografia a chiave simmetrica

La crittografia simmetrica, altrimenti detta crittografia a chiave segreta, utilizza una chiave comune e il medesimo algoritmo crittografico per la codifica e la decodifica dei messaggi !

Due utenti che desiderano comunicare devono accordarsi su di un algoritmo e su di una chiave comuni, La chiave deve essere scambiata su un canale sicuro.

Cifratura di cesare

Ogni carattere viene sostituito da un altro (*permutazione*), secondo un certo alfabeto che costituisce la chiave.

In chiaro :	$ABCDEFGHILMNOPQRSTUVZ$
Cifrate :	$DEFGHILMNOPQRSTUVZABC$

Le chiavi possibili sono solamente 20.

Cifratura monoalfabetica

Ogni carattere viene sostituito da un altro (permutazione), secondo un certo alfabeto che costituisce la chiave

In chiaro :	$ABCDEFGHILMNOPQRSTUVZ$
Cifrate :	$MZNCBVLAHSGDFQPEORITU$

Le chiavi possibili sono pari al numero di permutazioni possibili 21!, ovvero $5.1 \cdot 10^{19}$.

Anlisi delle frequenze

Spazio delle chiavi di un algoritmo monoalfabetico molto grande, ma la crittoanalisi è semplice tramite l'analisi delle frequenze.

In un testo scritto in una determinata lingua (italiano, inglese...) ogni lettera dell'alfabeto si presenta secondo una certa frequenza:

Ad esempio in italiano E ed A sono molto comuni, Q e Z sono poco comuni. E poi ci sono gruppi di 2 o 3 lettere ("ch", "che", "qu", ...).

Contando il numero di occorrenze di ogni lettera nel testo cifrato è possibile ipotizzare con buona probabilità quale sia la lettera corrispondente.

Cifrari a blocchi

Fa parte delle tecniche di cifratura simmetrica:

- Usati in molti protocolli sicuri di Internet, compreso `PGP` (posta elettronica), `SSL` (connessione `TCP`) e `IPSec` (trasmissione a livello di rete).

- Chiamati anche cifrari a flusso.

Dati k bit, i possibili 2^k ingressi vengono permutati, esempio $k = 3$.

Le permutazioni possono essere combinate tra loro per creare schemi più complessi.

5.5 Crittografia a chiave asimmetrica

Negli algoritmi simmetrici la chiave è la stessa in cifratura e decifratura, dunque deve essere segreta. Esiste quindi il problema della distribuzione delle chiavi. Serve quindi un canale sicuro di comunicazione per trasmettere la chiave.

Nel 1976 Diffie e Hellman propongono uno schema che supera questa limitazione, ovvero la crittografia a chiave pubblica o asimmetrica.

5.5.1 Chiave pubblica e chiave privata

Nella crittografia a simmetrica ogni utente ha una coppia di chiavi, costruita da una chiave pubblica e una chiave privata.

La chiave pubblica viene resa nota, quella privata deve rimanere segreta.

Il dato viene cifrato con la chiave pubblica del destinatario, che potrà decifrarlo con la propria chiave privata.

Vantaggi

In questo caso non è più necessario incontrarsi per scambiare chiavi. Inoltre ,la stessa chiave pubblica può essere usata da più utenti.

5.6 Algoritmo `RSA`

`RSA`, così chiamato dalle iniziali dei suoi inventori (*Rivest, Shamir, Aldeman*), è sicuramente il più noto algoritmo crittografico asimmetrico, si basa sulla difficoltà di scomporre un numero in fattori primi.

La chiave `RSA` ha di solito dimensioni di almeno 2^{10} bit.

Capitolo 6

Wireshark

TODO

Capitolo 7

Docker

7.1 Motivazioni

Gli hosts sono sempre connessi attraverso la rete che semplifica la distribuzione del software. Com'è possibile adattare tutti questi sistemi eterogenei con architetture, librerie e CPU diverse?

Le applicazioni di rete non sono ospitate in computer isolati, ma in:

- Clusters: gruppi di decine di computers connessi con una LAN ad alte prestazioni.
- Data centers: gruppi di centinaia di computers connessi tramite una gerarchia attraverso LAN ad alte prestazioni.

7.2 Soluzioni

Le soluzioni a questi problemi può passare attraverso:

- La virtualizzazione: è il sistema più pulito per rendere il sistema il più isolato possibile.
- Containers: cerca di semplificare la gestione attraverso l'eliminazione del sistema operativo ospite e mantenendo solamente un sistema operativo per ogni applicazione e inserendo il container tra il sistema operativo e le librerie utente.

Il container crea una membrana attorno ai processi per isolarli. Il programma è quindi scalabile, in questo modo posso gestire le risorse, è portabile (*per ogni hardware viene distribuito l'immagine*).

7.2.1 Ciclo di vita