Corso di Laurea
in
Ingegneria Informatica

Network Security
Elaborato d'esame

prof. Simon Pietro Romano

M63001026	Dario Daniele
M63001022	Simona De Vivo
M63001060	Salvatore Esposito

Università degli Studi di Napoli "Federico II"

Scuola Politecnica e delle Scienze di Base

Anno Accademico 2020/2021

Indice

Capitolo 1

Introduzione

Scopo del presente elaborato è installare un server **Virtual Private Network** per poi configurare i client ed utilizzarli per effettuare la connessione al suddetto server. Nello sviluppo è stato utilizzato **Docker**, un applicativo open-source che automatizza il deployment di applicazioni all'interno di contenitori software, ed **OpenVPN**, un programma VPN open source usato per stabilire tunnel crittografati punto-punto sicuri fra due computer attraverso una rete non sicura come Internet.

1.1 Virtual Private Network

Una **VPN** (Virtual Private Network) è una rete privata virtuale che garantisce privacy, anonimato e sicurezza attraverso un canale di comunicazione logicamente riservato (tunnel VPN) e creato sopra un'infrastruttura di rete pubblica. Il termine virtuale sta a significare che tutti **i dispositivi appartenenti alla rete non devono essere necessariamente collegati ad una stessa LAN locale** ma possono essere dislocati in qualsiasi punto geografico del mondo.

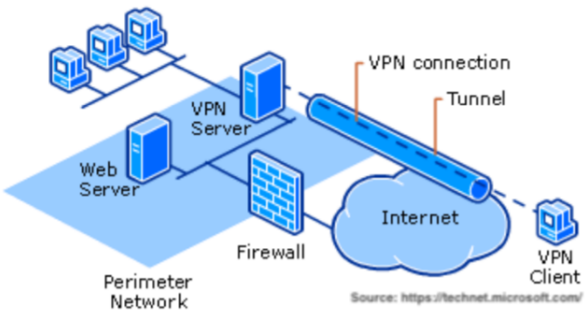


Figura 1.1: VPN Network

Può essere utilizzato per **criptare il traffico Internet e, di conseguenza, proteggere la propria identità online**. In ambito prettamente aziendale, una VPN può

essere paragonata ad una estensione geografica della rete locale privata (LAN) e che, quindi, permette di collegare tra loro, in maniera sicura, i siti della stessa azienda dislocati sul territorio. Per farlo, viene sfruttato l'instradamento dei pacchetti di dati tramite il protocollo IP per il trasporto su scala geografica: questo permette, di fatto, di realizzare una LAN "virtuale" e "privata" ma del tutto equivalente ad un'infrastruttura fisica di rete dedicata.

Le reti VPN si dividono in reti ad accesso remoto e reti site-to-site:

- **Connessione VPN ad accesso remoto**: consentono agli utenti (ad esempio in smart working) di accedere a un server su una rete privata tramite la rete Internet. Questo tipo di connessione può essere vista come un collegamento tra un PC client VPN e il server dell'azienda. Come già detto, dal punto di vista logico è come se si disponesse di un collegamento dedicato e privato;
- **Connessione VPN site-to-site**: è utilizzata per connettere in una rete privata, sempre con l'ausilio di una rete pubblica, uffici dislocati in più sedi o di altre organizzazioni, consentendo il routing ed una comunicazione sicura. In questo scenario, ogni sede avrà un router dedicato, ovvero un nodo della rete VPN che instraderà i pacchetti dati verso i destinatari omologhi secondo un modello client/server, condividendo le informazioni con le sedi remote in modo del tutto trasparente.

Viene utilizzato il **meccanismo del Tunnelling** che prevede di instaurare un **tunnel sicuro tra due entità remote** finali ed abilitate a realizzare una VPN. Non esiste nessun tunnel tecnicamente, ma piuttosto solo un collegamento logico attraverso una rete IP. Le due estremità del tunnel, anche se distanti e collegati attraverso molti nodi intermedi, durante il processo logico diventano virtualmente adiacenti. Facendo riferimento allo standard protocollare *ISO/OSI* ed all'architettura TCP/IP, possiamo affermare che con il tunneling si compie un **incapsulamento multi-protocollare dei dati**. I pacchetti di dati, anche se appartenenti a protocolli differenti, una volta giunti all'ingresso del tunnel, vengono ulteriormente incapsulate dal protocollo di tunneling e successivamente spediti sulla rete verso l'uscita del tunnel, dove, dopo avere rimosso l'incapsulamento raggiungono la destinazione.

Il **processo di autenticazione**, che dipende dal tipo di protocollo adottato, è necessario al fine di **autorizzare l'accesso ed assicurare la trasmissione**. Indipendentemente dalla tipologia VPN usata (accesso remoto/site-to-site) per instaurare una connessione tra un client ed il relativo server i passi che sono richiesti possono essere così riassunti:

1. il client contatta il server;
2. il server notifica la propria presenza;
3. il client richiede al server di essere identificato;
4. il server verifica che il tentativo di connessione sia autorizzato previa autenticazione riuscita;

5. il server risponde alla richiesta di autenticazione e autorizza la comunicazione con il client;

6. inizia la comunicazione tra le due entità;

Alla base della connessione c'è la **crittografia**, tecnica che assicura la **riservatezza delle informazioni** e che trasforma il dato leggibile, mediante un algoritmo, in un dato codificato e incomprensibile per i non autorizzati. Alla ricezione, la funzione di decodifica effettua il processo inverso. Il tipo di cifratura adoperata, come per il tipo di autenticazione usata, dipende dal protocollo di comunicazione adottato dal fornitore del servizio.

Per la trasmissione VPN esistono opportuni **protocolli** la cui scelta d'utilizzo dovrebbe dipendere dalle necessità e dai requisiti desiderati. Ognuno di questi protocolli con la loro specificità, **contribuisce alla protezione dei pacchetti dati in trasmissione**. Tra i protocolli più comuni si possono citare:

- **IPSEC**: l'Internet Protocol Security è un protocollo di livello 3 che permette una comunicazione sicura sulle reti IP. La riservatezza, l'integrità e l'autenticità del traffico dati vengono assicurate attraverso meccanismi di cifratura e autenticazione;

- **SSL/TLS**: il Secure Sockets Layer (TLS – Transport Layer Security è una versione aggiornata e più sicura di SSL) è un protocollo di livello 4 la cui tecnologia può essere usata anche per garantire la sicurezza di una connessione VPN. Una delle soluzioni software per la configurazione di una VPN per mezzo di SSL è OpenVPN;

- **HTTPS**: l'Hyper Text Transfer Protocol Secure è un protocollo di livello applicazione per il trasferimento ipertestuale sicuro che si appoggia sul protocollo di trasporto SSL/TLS. Può essere utilizzato attraverso l'installazione di applicativi ad hoc e/o di estensioni browser.

1.2 OpenVPN

OpenVPN è un programma VPN open source usato per creare tunnel crittografati punto-punto sicuri fra due computer attraverso una rete non sicura, ad esempio Internet. Permette agli **host di autenticarsi l'uno con l'altro per mezzo di chiavi private condivise**, certificati digitali o credenziali utente/password. Usa le librerie di cifratura **OpenSSL** e il protocollo **SSLv3 - TLSv1**. È disponibile su una vasta gamma di piattaforme come GNU/Linux, xBSD, macOS, Solaris, Android e Windows. L'intero programma è contenuto in un singolo file eseguibile che può funzionare sia in modalità server che client, da un file di configurazione opzionale e da uno o più file contenenti le

chiavi, in funzione del metodo di autenticazione usato.

OpenSSL è una realizzazione open source dei protocolli SSL/TLS (Secure Socket Layer e Transport Layer Security) per la certificazione e la comunicazione cifrata. Si compone di alcune librerie che permettono di incorporare le funzionalità dei protocolli SSL/TLS all'interno di programmi di comunicazione, e di una serie di programmi di utilità per la gestione delle chiavi e dei certificati, arrivando eventualmente anche alla gestione di un'autorità di certificazione.

OpenVPN **concentra tutto il traffico dati e di controllo su una singola porta** che può essere una porta UDP (preferita e predefinita) oppure TCP. A differenza di altri programmi o protocolli VPN, come LP2P o PPTP, **non richiede altri protocolli per gestire autenticazione o dati**, cosa che gli permette di funzionare attraverso la maggior parte dei server proxy (HTTP incluso), **evitare limiti e blocchi imposti dall'ISP** e semplificare molto l'integrazione con i NAT. Il server può "inviare" alcune opzioni di configurazione di rete ai client. Fra queste, l'indirizzo IP, gli instradamenti, e alcune opzioni di connessione.

1.3 Docker

Docker è una piattaforma software che permette di **creare, testare e distribuire applicazioni con la massima rapidità**. Docker raccoglie il software in unità standardizzate chiamate **container** che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.

Tale strumento utilizza le **immagini**, queste sono una rappresentazione statica dell'applicazione o del servizio e della relativa configurazione e delle dipendenze. Le immagini facilitano enormemente la distribuzione delle applicazioni nei *container* e, ciascun file d'immagine, è strutturato su più livelli o strati in modo che ogni modifica applicata sul contenuto generi un nuovo livello. Una soluzione intelligente che permette di preservare la configurazione dei livelli sottostanti pur consentendo possibilità di personalizzazione illimitate, è infatti possibile tornare alla versione precedente in qualunque momento e senza difficoltà. Tutte le immagini sviluppate sono poi conservate in un **registro** disponibile ai fruitori. Nel dettaglio l'inserimento delle immagini in un registro permette di archiviare bit di applicazioni statici e immutabili, incluse tutte le dipendenze a livello di

framework. È quindi possibile creare diverse versioni delle immagini e distribuirle in più ambienti per fornire un'unità di distribuzione coerente.

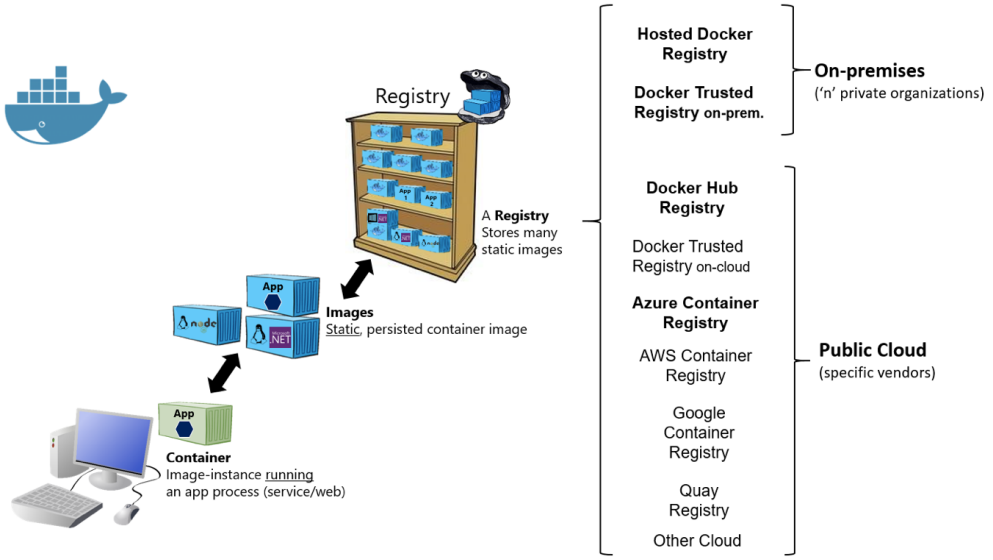
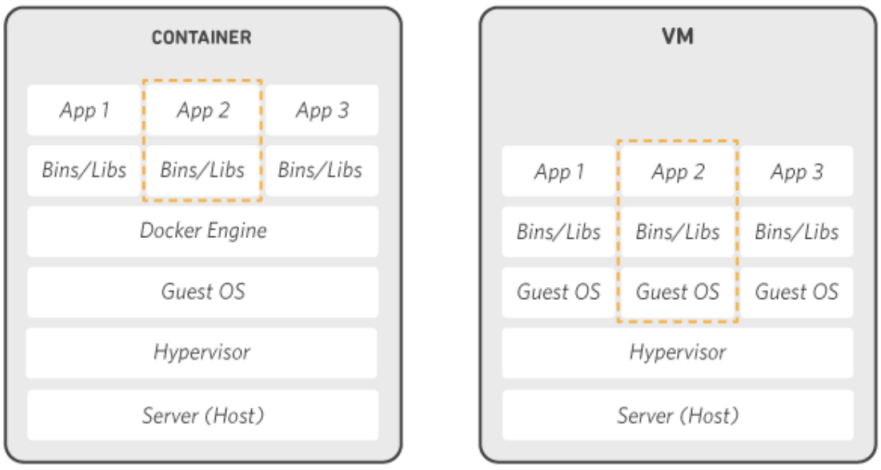


Figura 1.2: Docker: architettura e tassonomia

Docker fornisce un **sistema operativo per container**: così come la macchina virtuale virtualizza i server hardware (ovvero elimina la necessità di gestirli direttamente), i container virtualizzano il sistema operativo di un server. Docker è installato su ogni server e fornisce semplici comandi con cui creare, avviare o interrompere i container.



Capitolo 2

Server e Client VPN

2.1 Creazione Immagine

Al fine di creare una immagine per il server e per il client, sono stati eseguiti i seguenti comandi di commit che hanno permesso la consolidazione di una immagine a partire dall'immagine di **kylemanna/openvpn** disponibile sul Docker Hub. Preventivamente l'immagine scaricata è stata inserita ed avviata in un container con id *[ID_ CONTAINER].*

```
docker commit [ID_CONTAINER] sdsteamunina/ns_image
```

2.2 Creazione Server

Il primo passo per creare il Server VPN è creare un **volume**. Nel dettaglio un volume fa riferimento a **un'unità specificatamente designata all'archiviazione dei dati** che si rivelano molto funzionali perché permettono la condivisione e il riutilizzo dei volumi tra diversi container e la persistenza dei dati indipendentemente dal ciclo di vita del container stesso. Definiamo quindi la **variabile d'ambiente** seguente per definire il nome del volume:

```
OVPN_DATA="ovpn-ns"
```

Creare il volume:

```
docker volume create --name \$OVPN_DATA
```

Per verificare in quale percorso della macchina host sia stato creato il volume, si usa il seguente comando:

```
docker volume inspect ovpn-ns
```

Figura 2.1: Docker: output comando volume inspect

Generare poi le configurazioni per il server, nel dettaglio il percorso per raggiungere i dati generati all'interno del server è */etc/openvpn*.

```
docker run -v \$OVPN_DATA:/etc/openvpn --name
   ↪ ovpn_server --rm sdsteamunina/ns_image
   ↪ ovpn_genconfig -u udp://servervpn.ns
```

Avviare la generazione delle chiavi private e dei certificati. In particolare il seguente comando richiederà l'inserimento di una *passphrase* che sarà utile per la generazione di nuovo materiale crittografico per creare la *Public Key Infrastructure* (PKI). Verranno quindi creati file come il certificato pubblico della Certificate Authority, un file contente i parametri di Diffie Helman, la chiave privata ed il certificato di autenticità del server.

```
docker run -v \$OVPN_DATA:/etc/openvpn --rm -it
   ↪ sdsteamunina/ns_image ovpn_initpki
```

In seguito creiamo ed avviamo il container docker "*openvpn_server*" in modalità *detached* (opzione -d: se si chiude il terminale, il container resta in esecuzione), **mappando il porto 1194 della macchina host sul porto 1194 del container sul protocollo UDP** (opzione -p) usando l'immagine precedentemente usata. Digitando il comando "*docker ps*" è posssibile infatti verificare che ora è presente un nuovo container il cui nome è *openvpn_server*. Lanciamo quindi:

```
docker run -v \$OVPN_DATA:/etc/openvpn --name
   ↪ openvpn_server -d -p 1194:1194/udp --cap-add=
   ↪ NET_ADMIN sdsteamunina/ns_image
```

Successivamente creare, all'interno del volume, chiave privata e certificato, usando la stessa passphrase.

```
docker run -v \$OVPN_DATA:/etc/openvpn --rm -it
   ↪ sdsteamunina/ns_image easyrsa build-client-full
   ↪ nomeclient nopass
```

Accedere poi al container ed al materiale presente al suo interno. Il parametro alfanumerico (container id) è l'identificativo del container dove è in esecuzione il server (noto come "*openvpn_server*" e individuabile nuovamente tramite il comando *docker ps*):

```
   docker exec -it <container id> bash
```

Cambiare poi la directory:

```
   cd etc/openvpn/pki
```

Creare il file di configurazione per il client openvpn per connetterlo al server OpenVPN in VPN:

```
ovpn_getclient nomeclient > nomeclient.ovpn
```

Infine con il seguente comando verificare l'indirizzo IP dell'interfaccia eth0 del server (*inet*). Questa informazione sarà necessaria per la configurazione successiva del client.

```
   ifconfig
```

2.3 Configurazione Client

Una volta creato ed avviato il server, da una nuova finestra di terminale eseguire i seguenti comandi al fine di creare ed avviare un client. Per prima cosa impostare una nuova

variabile d'ambiente per far puntare il client allo stesso volume creato dal server:

```
OVPN_DATA="ovpn-ns"
```

Avviare il container del client creando una interfaccia di rete *TUN* che è quella dedicata alla creazione delle connessioni crittografate (come quella VPN), che comunicherà con l'interfacccia *TUN* del server. Una volta eseguito il seguente comando, verrà aperto in automatico la *bash* del container.

```
docker run -v \$OVPN_DATA:/etc/openvpn --device /dev/net/
   ↪ tun:/dev/net/tun -it --cap-add=NET_ADMIN --name
   ↪ openvpn_client sdsteamunina/ns_image /bin/bash
```

Si procede poi ad eseguire il seguente comando che aprirà il file hosts che consente di risolvere i nomi di dominio. Digitando un dominio all'interno del browser, il sistema operativo verifica prima se c'è una entry corrispondente a quel dominio richiesto. Se non presente, prosegue con una query verso il DNS configurato. Perciò si modifica il file scrivendo una nuova riga ed aggiungendo indirizzo IP dell'interfaccia *eth0* del server recuperata in precedenza seguito dal nome simbolico scelto per server (in questa analisi *servervpn.ns*). Infine salvare il file.

```
vi etc/hosts
```

Cambiare poi directory:

```
cd /etc/openvpn/pki
```

Infine effettuare la connessione al server:

```
openvpn --config nomeclient.ovpn
```

2.4 Verifica della connessione

Per verificare l'effettiva connessione tra il server ed il client sono stati analizzati i ping delle connessioni tramite le interfacce *TUN* del client e del server. Di seguito sono riportate le configurazioni di rete di client e server.

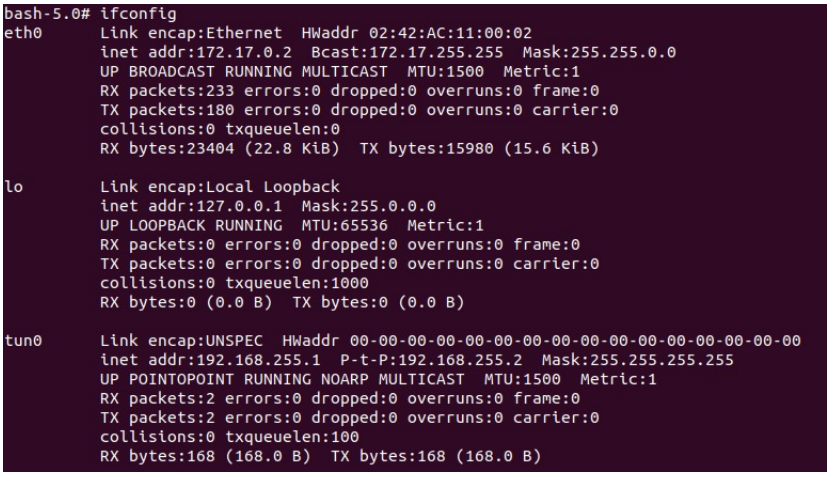


Figura 2.2: Configurazione di rete del server.

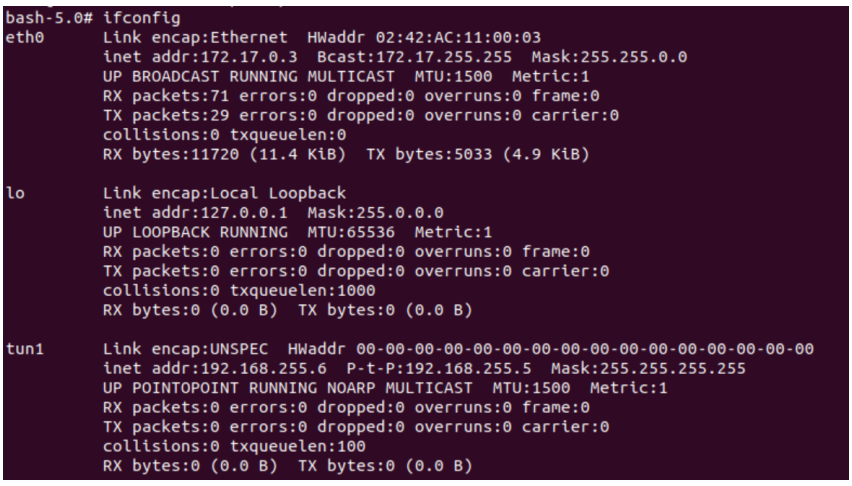


Figura 2.3: Configurazione di rete del client.

Dalle immagini sovrastanti si evince che gli indirizzi IP di client e server sono rispettivamente 192.168.255.6 e 192.168.255.1. Per verificare che client e server siano effettivamente connessi è stato effettuato il ping dal server verso il client e viceversa. Il risultato è mostrato di seguito:

```
bash-5.0# ping 192.168.255.6
PING 192.168.255.6 (192.168.255.6): 56 data bytes
64 bytes from 192.168.255.6: seq=0 ttl=64 time=0.306 ms
64 bytes from 192.168.255.6: seq=1 ttl=64 time=0.348 ms
64 bytes from 192.168.255.6: seq=2 ttl=64 time=0.270 ms
64 bytes from 192.168.255.6: seq=3 ttl=64 time=0.439 ms
64 bytes from 192.168.255.6: seq=4 ttl=64 time=0.429 ms
64 bytes from 192.168.255.6: seq=5 ttl=64 time=0.518 ms
64 bytes from 192.168.255.6: seq=6 ttl=64 time=0.224 ms
64 bytes from 192.168.255.6: seq=7 ttl=64 time=0.577 ms
64 bytes from 192.168.255.6: seq=8 ttl=64 time=0.504 ms
64 bytes from 192.168.255.6: seq=9 ttl=64 time=0.702 ms
```

Figura 2.4: Ping dal server verso il client.

```
bash-5.0# ping 192.168.255.1
PING 192.168.255.1 (192.168.255.1): 56 data bytes
64 bytes from 192.168.255.1: seq=0 ttl=64 time=0.222 ms
64 bytes from 192.168.255.1: seq=1 ttl=64 time=0.367 ms
64 bytes from 192.168.255.1: seq=2 ttl=64 time=0.499 ms
64 bytes from 192.168.255.1: seq=3 ttl=64 time=0.551 ms
64 bytes from 192.168.255.1: seq=4 ttl=64 time=0.395 ms
64 bytes from 192.168.255.1: seq=5 ttl=64 time=0.319 ms
64 bytes from 192.168.255.1: seq=6 ttl=64 time=0.292 ms
64 bytes from 192.168.255.1: seq=7 ttl=64 time=0.276 ms
64 bytes from 192.168.255.1: seq=8 ttl=64 time=0.612 ms
64 bytes from 192.168.255.1: seq=9 ttl=64 time=0.385 ms
64 bytes from 192.168.255.1: seq=10 ttl=64 time=0.306 ms
64 bytes from 192.168.255.1: seq=11 ttl=64 time=0.274 ms
64 bytes from 192.168.255.1: seq=12 ttl=64 time=0.204 ms
```

Figura 2.5: Ping dal client verso il server.

Come è possibile notare dai risultati mostrati, server e client sono connessi.

container e coker commit -id container -nome immagine