



UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea triennale in *Ingegneria Informatica e dell'Automazione*

Implementazione di OpenVPN su router 4G per site-to-site vpn in ambiente CG-NAT

*Implementation of OpenVPN on a 4G Router for site-to-site VPN in
CG-NAT environment*

Relatore:

Prof. Ennio Gambi

Correlatore:

Ing. Adelmo De Santis

Laureando:

Simone Viozzi

Prefazione

#TODO da riscrivere

Nell'ambito del mio percorso universitario ho avuto modo di approfondire le tematiche relative al mondo delle reti e del networking, a tal proposito grazie alla possibilità offerta dal Dipartimento di Ingegneria dell'Informazione, dal Prof. Ennio Gambi e dall'Ing. Adelmo De Santis ho conseguito con successo la certificazione "*HUAWEI HCIA Routing and Switching*". Successivamente, grazie alle competenze acquisite, ho collaborato con alcuni miei colleghi per progettare e realizzare una implementazione di una VPN site-to-site attraverso una connessione radiomobile per conto dell'azienda Esse-ti S.r.l.

In questo elaborato verranno esposte le principali fasi del progetto realizzato, ponendo un particolare focus sulle problematiche iniziali affrontate e all'architettura di rete nel cui ambito è stata realizzata la comunicazione tramite un canale sicuro.

NOTA:

La prefazione è una breve nota personale riguardante la tua tesi triennale o magistrale. Qui puoi fornire al lettore informazioni inerenti le origini e il contesto della tua tesi. Inoltre, la prefazione può essere usata anche per ringraziare chiunque abbia aiutato nella stesura dell'elaborato.

Indice

Prefazione	ii
Indice	iii
Elenco delle figure	v
1 Introduzione	1
1.1 TCP/IP e modello a strati	1
1.1.1 Internet Protocol Suite	1
1.1.2 Incapsulamento	3
1.2 Openvpn	3
1.3 Openwrt	4
1.4 Introduzione a Luci	4
1.5 Accesso ssh al Router	5
2 Overview dell'architettura e delle componenti utilizzate	7
2.1 Obiettivo da ottenere	7
2.2 Specifiche dei componenti	8
2.2.1 Esse-ti 4G.Router	9
2.2.2 VPS OVHCloud	10
2.2.3 Host domotico	11
2.2.4 Macchina del cliente	11
3 Configurazione del server	12
3.1 Overview della configurazione e prerequisiti	12
3.2 Creazione della Public key infrastructure Certificate Authority (PKI CA)	13
3.2.1 Configurazione della PKI di OpenVPN	15
3.2.2 Firma del certificato opnevpn dalla CA	17

3.3	Generazione della <i>tls-crypt pre-shared key</i>	18
3.4	Generazione delle chiavi per i clients	18
3.5	Creazione del file di configurazione del server OpenVPN	20
3.6	Configurazioni sulla network stack del server openvpn	20
3.7	Configurazione del firewall	20
3.7.1	Configurazione del NAT	21
3.7.2	Configurazione del packet forwarding	22
3.7.3	Conclusione della configurazione del firewall	22
3.8	Avvio del server OpenVPN	23
3.9	Script per la creazione delle configurazioni dei client	24
3.10	Test della configurazione	25
4	Configurazione Router	27
4.1	Overview della configurazione	27
4.2	Creazione della configurazione Openvpn	27
4.3	Abilitazione del Client-to-Client nel server OpenVPN	29
4.4	Assegnazione ip statico al Router	30
5	Connessione degli host domotici alla VPN	32
5.1	Overview della configurazione	32
5.2	Creazione della firewall zone per la VPN	32
5.3	Aggiunta dell'interfaccia tun0 alla zona vpn	35
5.4	Modifiche alla configurazione OpenVPN del server	35
5.5	Test della configurazione	36
	Bibliografia	39

Elenco delle figure

1.1	Rappresentazione degli strati del modello TCP/Ip con relativo incapsulamento e dispositivo di dominio	2
1.2	Interfaccia grafica LuCI	5
2.1	Schema concettuale dell'obiettivo da raggiungere. [2]	7
2.2	Schema concettuale dell'architettura che si dovrà implementare. [2]	8
2.3	Topologia virtuale. [2]	8
2.4	4G.Router	9
3.1	Configurazione di partenza e di obiettivo per questo capitolo. [2]	12
3.2	Diagramma per schematizzare la procedura di firma di un certificato client. [2] .	13
3.3	Diagramma per schematizzare la procedura di firma di un certificato client. [2] .	18
4.2	Configurazione della VPN tramite LuCI	29
5.2	Configurazione iniziale del firewall	33
5.3	Configurazione del firewall	34
5.4	Configurazione finale del firewall	34

Capitolo 1

Introduzione

1.1 TCP/IP e modello a strati

Internet e' una rete di telecomunicazione ad accesso pubblico reso possibile da una suite di protocolli.

Un sistema di comunicazione Internet consiste in un interconnessione tra reti di pacchetti, che supportano la comunicazione tra hosts usando una suite di protocolli.

L'obbiettivo di Internet e' quello di collegare hosts attraverso una rete geografica, indipendentemente dalla distanza o da quali mezzi fisici vengano usati per trasferire i dati. Deve quindi essere in grado di gestire una grande varieta' di applicazioni, tipi di rete e mezzi di trasmissione. Per garantire la flessibilita' necessaria e non aggiungere troppa complessita', l'architettura di Internet e' sta suddivisa in alcuni strati.

1.1.1 Internet Protocol Suite

Per comunicare su Internet, gli Host devono implementare un set di protocolli che costituiscono l'*Internet protocol suite* [1]. I protocolli si suddividono in strati logici che li raggruppano in 4 categorie, ogni Host deve implementare almeno un protocollo per ogni strato.



Figura 1.1: Rappresentazione degli strati del modello TCP/Ip con relativo incapsulamento e dispositivo di dominio

4. **Application Layer:** Il livello applicazione e' il layer piu' alto dell'*Internet protocol suite*, i protocolli di questo livello si suddividono in protocolli utente e di supporto. I protocolli utente espongono un servizio direttamente all'utente finale, alcuni esempi sono: [http](#), [ftp](#), [ssh](#), etc. I protocolli di supporto forniscono alcune funzionalita' di supporto per il funzionamento della rete, alcuni esempi sono: [DNS](#), [SNMP](#) etc.
3. **Transport Layer:** Il livello di trasporto fornisce una comunicazione end-to-end tra le applicazioni, infatti in generale il campo data del livello di trasporto non viene letto da nessuno se non l'applicazione di sorgente e destinazione. I protocolli principali di questo livello sono TCP e UDP: il [TCP](#) e' connection-oriented e fornisce alta affidabilita'; mentre l'[UDP](#) e' connection-less, quindi ogni inaffidabilita' della rete deve essere gestita a livello applicazione.
2. **Internet Layer:** Tutti i protocolli di trasporto usano il protocollo Internet (IP) per portare i dati dall'host sorgente alla destinazione. Al contrario dei protocolli di livello trasporto il protocollo IP non e' end-to-end, quindi e' intrinsecamente di tipo connection-less, non fornisce quindi nessuna garanzia che il pacchetto arrivi a destinazione, o arrivi danneggiato o duplicato. I layer sopra al livello IP sono responsabili di mantenere l'affidabilita' dei servizi quando essa e' richiesta. Di questo layer fanno parte i protocolli [IP](#), [ICMP](#), etc.
1. **Link Layer:** E' il layer piu' vicino al mezzo fisico su cui viaggiano i dati, ogni host deve implementare il protocollo usato per la specifica interfaccia che usa. Ad esempio un'host con un'interfaccia Ethernet deve implementare i protocolli *Ethernet II* e *IEEE 802.3*.

Possiamo vedere a cosa corrisponde in pratica con una cattura di un pacchetto eseguita con *Wireshark*:

Wireshark

```
1 1 > Frame 43408: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface
   ↳ wlp5s0, id 0
2 1 > Ethernet II, Src: IntelCor_eb:91:5f (cc:d9:ac:eb:91:5f), Dst: HuaweiDe_27:a9:24
   ↳ (0c:e4:a0:27:a9:24)
3 2 > Internet Protocol Version 4, Src: 192.168.8.119, Dst: 142.250.180.174
4 3 > Transmission Control Protocol, Src Port: 36354, Dst Port: 443, Seq: 36720, Ack:
   ↳ 13639, Len: 39
5 4 > Transport Layer Security
6     > TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
7         Opaque Type: Application Data (23)
8         Version: TLS 1.2 (0x0303)
9         Length: 34
10        Encrypted Application Data:
11        ↳ 389bf9516cce567d0d90ef62ba2a87376091fedb7f66f3b9e60e45a39376b1ae667a
        [Application Data Protocol: http-over-tls]
```

1.1.2 Incapsulamento

Ogni protocollo di ogni layer aggiunge un header e un trailer, incapsulando il prodotto del layer precedente nel suo campo data, possiamo vedere una rappresentazione grafica in fig. 1.1.

Ad esempio analizzando la cattura in Wireshark, code. 1.1.1, si può vedere lo stack dei protocolli e i loro relativi header:

- **Transport Layer Security**[6]: E' il dato effettivo che e' stato trasmesso in rete
- **Transmission Control Protocol**[5]: incapsula il layer applicazione in un layer di trasporto usando il protocollo TCP. Aggiunge un minimo di 20 byte di header
- **Internet Protocol Version 4**[4]: incapsula il layer di trasporto in un pacchetto IP. Aggiunge un minimo di 20 byte di header
- **Ethernet II**[8]: in questo caso il mezzo fisico e' una porta Ethernet quindi il pacchetto IP viene incapsulato con il protocollo Ethernet II. Questo aggiunge 14 byte di header e 4 byte di trailer.

Si vede come il dato che si voleva trasmettere sia stato incapsulato da 4 strati di protocolli che complessivamente hanno aggiunto un minimo di 58 byte di overhead.

1.2 Openvpn

OpenVPN e' un applicativo opensource che ha l'obiettivo di fornire una VPN che sia semplice da configurare e che funzioni in ogni contesto. Openvpn può incapsulare sia pacchetti IP che frame Ethernet, in un tunnel sicuro che può viaggiare sia su TCP che UDP. Nella configurazione e' possibile usare qualsiasi porta, tutto ciò lo rende estremamente flessibile.

Possiamo ad esempio vedere una cattura di wireshark di un pacchetto OpenVPN su UDP e porta 1194:

```
Wireshark
1 > Frame 90: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface
  ↳ br-08876ccdf1f5, id 0
2 > Ethernet II, Src: 02:42:0a:00:04:03 (02:42:0a:00:04:03), Dst: 02:42:0a:00:04:02
  ↳ (02:42:0a:00:04:02)
3 > Internet Protocol Version 4, Src: 10.0.4.3, Dst: 10.0.4.2
4 > User Datagram Protocol, Src Port: 47007, Dst Port: 1194
5 > OpenVPN Protocol
6     Type: 0x48 [opcode/key_id]
7     Peer ID: 0
8     Data (36 bytes)
9         Data:
          ↳ 00000019a366196eb2aca181df226faf8514ab73f524f7ef335d55fc57322d032a2095e4
```

Per la cifratura si serve della libreria OpenSSL, ne supporta quindi tutti i protocolli di cifratura. Ha inoltre ulteriori estensioni alla sicurezza come l'utilizzo dell'autenticazione di pacchetto HMAC.

Usa un'applicativo unico che puo' funzionare sia da server che da client in base alla configurazione che si usa.

1.3 Openwrt

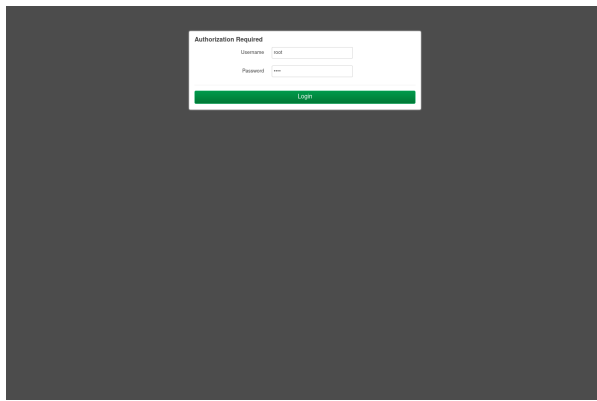
OpenWRT e' un sistema operativo per sistemi embedded, e' principalmente usato nei router. E' basato su kernel linux, con specifica attenzione dell'ottimizzazione del sistema per fare in modo che possa eseguirsi su sistemi con risorse estremamente ridotte. Al contrario di altri sistemi operativi per dispositivi embedding, OpenWRT presenta un filesystem con permessi di scrittura, cio' permette all'utente di accedervi e modificare a runtime le funzionalita' del dispositivo.

Comprende una shell linux comprensiva delle funzionalita' piu' comuni, compreso il suo gestore pacchetti opkg.

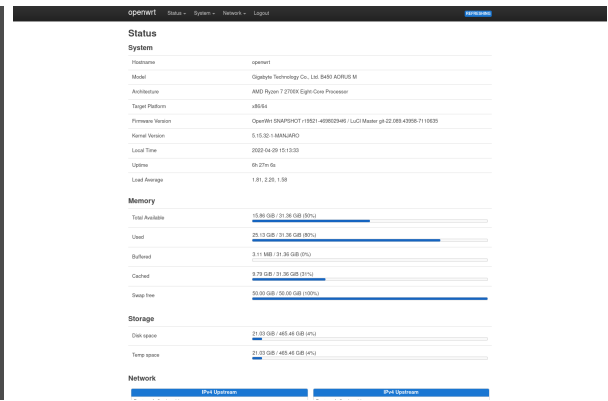
OpenWRT puo' essere configurato sia tramite shell che tramite interfaccia web (LuCI).

1.4 Introduzione a Luci

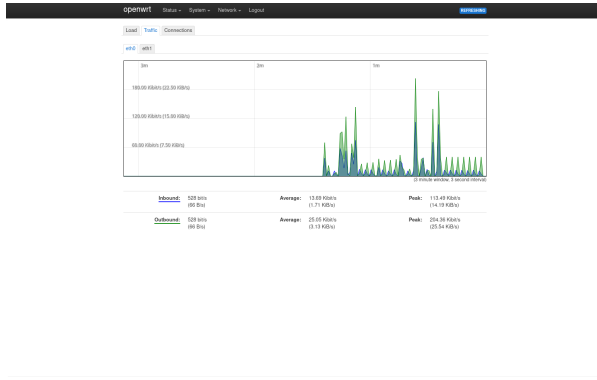
Luci è la web interface ufficiale di *openwrt*, dovrebbe essere già installata e raggiungibile, in caso contrario può essere installata e configurata seguendo la guida ufficiale di *Open Wrt* [3].



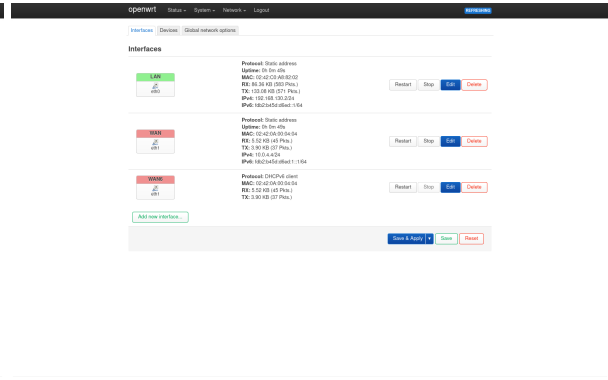
(a) Login page



(b) Status page



(c) Graphs page



(d) Interfaces page

Figura 1.2: Interfaccia grafica LuCI

Le credenziali di default sono **Username:root** **Password:root**, come mostrato in fig. 1.2a.

L'homepage, fig. 1.2b, mostra un riepilogo dello stato del router, ad esempio sono presenti: informazioni sull'hardware, informazioni sulla memoria e storage, sono presenti inoltre informazioni riassuntive sulle interfacce di rete e sul *DHCP*.

L'interfaccia è estensiva e permette di configurare quasi ogni aspetto del funzionamento del router, compreso il firewall, il *DHCP*, i processi in esecuzione, etc.

1.5 Accesso ssh al Router

Oltre all'interfaccia grafica *LuCI* si può accedere al router tramite ssh.

Router 4g

```
1 BusyBox v1.35.0 (2022-04-24 21:09:51 UTC) built-in shell (ash)
2
3  |      | |.----.----.----.| | | | |.----.| |
4  |  -  || _ | -__|      || | | | | _|| _|
5  |-----|| _|____|____|____|____|____|____|
6      |__| W I R E L E S S   F R E E D O M
7  -----
8  OpenWrt SNAPSHOT, r19521-46980294f6
9  -----
```

Non esiste un account utente quindi viene effettuato il login come root.

Capitolo 2

Overview dell'architettura e delle componenti utilizzate

2.1 Obbiettivo da ottenere

In una collaborazione tra il Dipartimento di Ingegneria dell'Informazione e l'azienda **Esse-ti S.R.L.** ci è stato esposto un progetto che consiste nel:

- fornire a dei clienti un router 4G, su cui possono essere connessi vari dispositivi, ad es. di tipo domotico.
- rendere questi dispositivi accessibili ai clienti attraverso internet

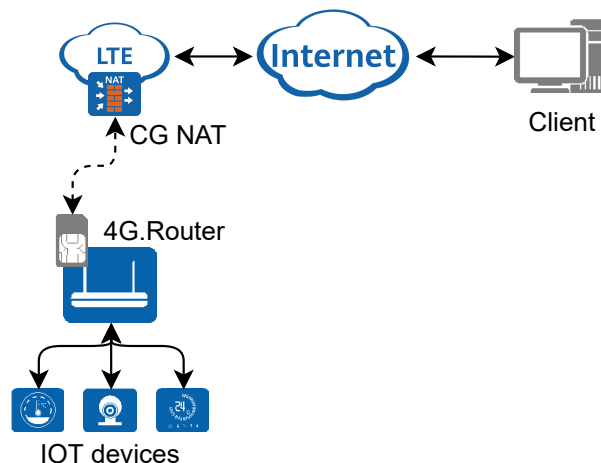


Figura 2.1: Schema concettuale dell'obiettivo da raggiungere. [2]

Data la presenza del CG-NAT si vede subito che non è realizzabile a meno che il cliente non abbia un'IP pubblico e la sua macchina venga configurata opportunamente. Questo però non è possibile nel caso generale, quindi per risolvere efficacemente questa topologia si deve necessariamente introdurre una terza macchina provvista di IP pubblico e che funga da ponte tra il 4G.Router e il cliente.



Figura 2.2: Schema concettuale dell'architettura che si dovrà implementare. [2]

In questo modo si può configurare una VPN sul server OVH e connettersi sia il 4G.Router che la macchina del cliente. In questo modo l'unica configurazione che il cliente dovrà fare è l'installazione di un cliente VPN, ciò è il minimo possibile di configurazione.

La configurazione virtuale vista dal 4G.Router e dai clienti sarà quindi:



Figura 2.3: Topologia virtuale. [2]

2.2 Specifiche dei componenti

i componenti necessari sono:

- Esse-ti 4G.Router
- Server
- Host domotico
- Macchina del cliente

vediamo le caratteristiche minime che i componenti dovranno avere:

2.2.1 Esse-ti 4G.Router

Ci è stato fornito dall'azienda Esse-ti, consiste in un gateway 4G con funzionalità di router. Le specifiche complete possono essere trovate sul sito del produttore ([link](#))



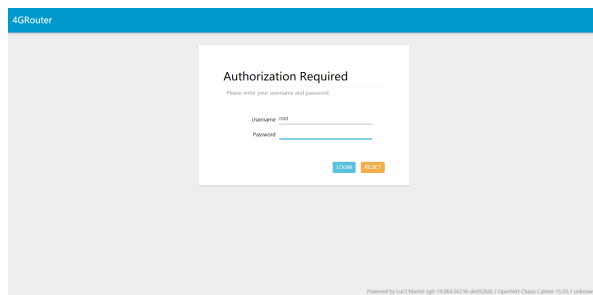
Figura 2.4: 4G.Router

Per l'implementazione di questa architettura sono necessarie solo un sub-set delle specifiche:

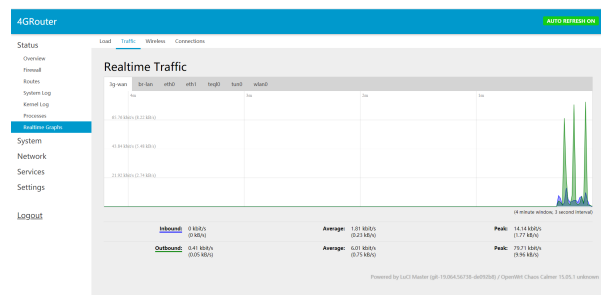
- Access Point wireless per offrire connettività Internet Wi-Fi a dispositivi wireless
- Client Dynamic DNS per consentire all'utente di raggiungere da remoto, tramite Internet, il router stesso e tutti i dispositivi connessi via Wi-Fi o porta LAN
- Gateway telefonico per consentire l'invio e la ricezione di chiamate attraverso la rete 4G LTE/UMTS/GSM a telefoni fissi, combinatori o altri dispositivi telefonici collegati all'ingresso FXS

Presenta inoltre come sistema operativo una versione personalizzata di OpenWrt.

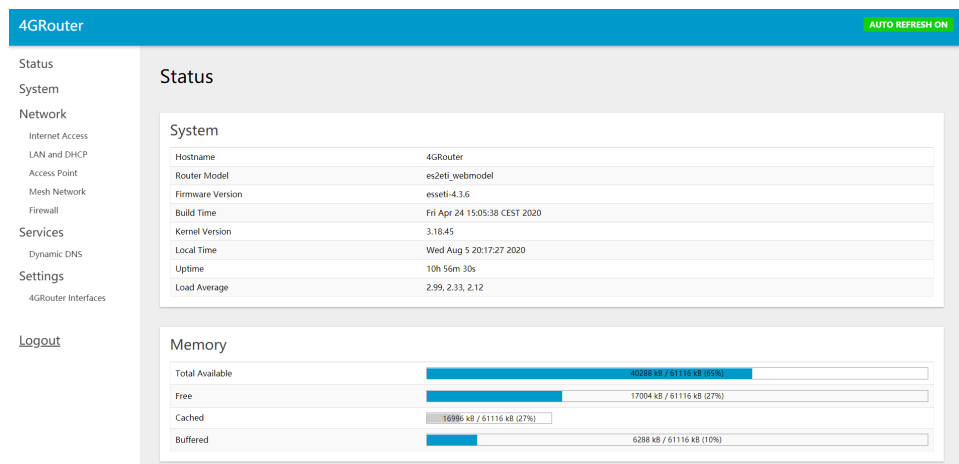
La configurazione del dispositivo può essere fatta sia da terminale, entrando in ssh, sia da interfaccia web:



(a) Schermata di autenticazione



(b) Grafico del traffico



(c) Schermata con stato riassuntivo

L'interfaccia web e' una versione personalizzata di [Luci](#).

Per semplicita' si fara' riferimento all'*Esse-ti 4G.Router* chiamandolo semplicemente *Router*.

2.2.2 VPS OVHCloud

La VPS ha il solo vincolo di dover avere un'ip pubblico e una connessione a internet abbastanza veloce. Dovra' infatti sopportare un traffico simmetrico in upload / download.

Per la realizzazione della topologia e' stata selezionata una macchina VPS del provider OVH-Cloud, con le seguenti caratteristiche:

- 2 core virtuali
- 4Gb di memoria ram
- 80Gb di storage NVMe
- 500Mbps simmetrici di banda
- ipv4 pubblico
- Ubuntu 16.04

Per semplicita' si fara' riferimento alla *VPS OVHCloud* come *Server*.

2.2.3 Host domotico

Per effettuare le varie operazioni di testing e' stato aggiunta *raspberry pi* che ha svolto il ruolo di "host domotico". Sono state fatti test con ping e iperf per testare che tutta la topologia sia stata configurata correttamente.

2.2.4 Macchina del cliente

Deve poter essere una qualunque macchina, non ha quindi vincoli di sistema operativo.

Necessita di avere il client openvpn installato:

- con sistema operativo Windows si deve scaricare l'eseguibile dal [sito ufficiale](#)
- su linux e' sufficiente cercare nei repository ufficiali della distribuzione che si sta usando.

Capitolo 3

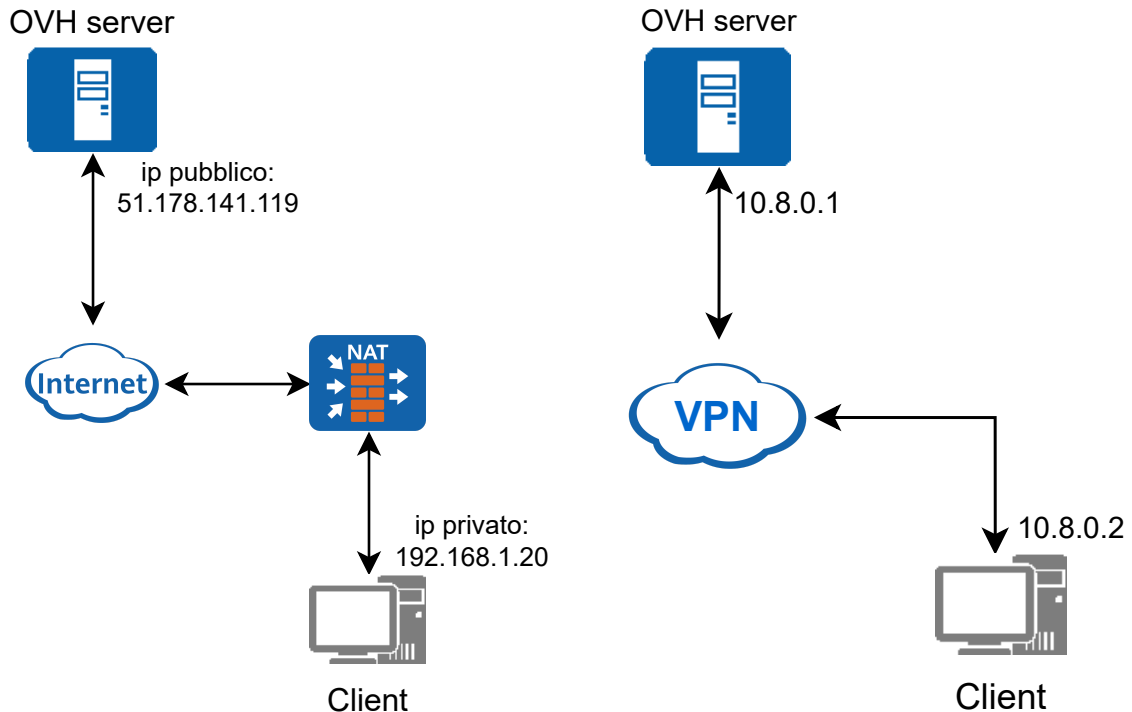
Configurazione del server

3.1 Overview della configurazione e prerequisiti

In questa sezione andremo a installare e configurare OpenVPN server sulla VPS di OVHCloud.

Supponiamo di partire da una configurazione di base, che contiene solo il server openvpn ed un generico client, che supponiamo sia sotto un NAT.

Supponiamo inoltre che l'ip pubblico del *server* sia `51.178.141.119`, si avrà quindi una configurazione come in figura [3.1a](#).



(a) Configurazione di partenza per questo capitolo. (b) Configurazione virtuale da raggiungere.

Figura 3.1: Configurazione di partenza e di obiettivo per questo capitolo. [\[2\]](#)

Per instaurare una comunicazione bidirezionale tra il server e il client, si dovrà configurare

oppportunamente una rete vpn la cui configurazione è rappresentata in figura 3.1b.

I pacchetti necessari sono `openvpn` ed `easy-rsa`, che possono essere installati con:

```
Server
1 $ sudo apt-get update
2 $ sudo apt-get install -y openvpn easy-rsa
```

È inoltre necessario avere un editor di testo, ad es. `nano` o `vim`

3.2 Creazione della Public key infrastructure Certificate Authority (PKI CA)

La CA può essere configurata sulla stessa macchina dove è stato installato `openvpn`, ma ciò è sconsigliato per motivi di sicurezza, supponiamo quindi di usare un secondo server chiamato *server CA*

La utility `easy-rsa` mette a disposizione il comando `make-cadir`, che permette di creare una cartella pronta ad ospitare la Certificate Authority.

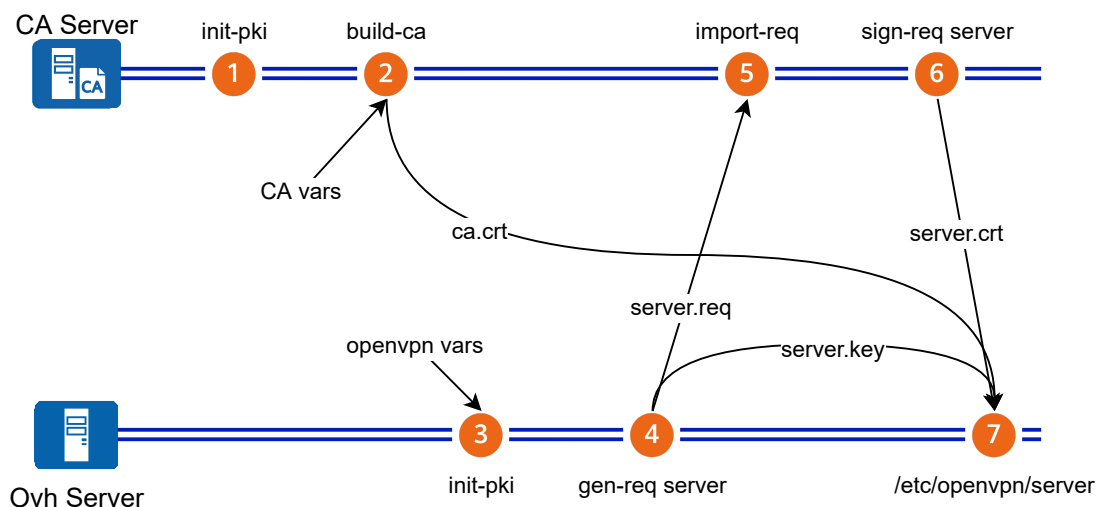


Figura 3.2: Diagramma per schematizzare la procedura di firma di un certificato client. [2]

Andiamo quindi a crearla, nella home ad esempio:

Server CA

```
1 $ mkdir ~/openvpn-ca
2 $ ln -s /usr/share/easy-rsa/* ~/openvpn-ca/
3 $ chmod 700 /home/ubuntu/openvpn-ca/
4 $ cd openvpn-ca/
5 $ ./easyrsa init-pki
6
7 init-pki complete; you may now create a CA or requests.
8 Your newly created PKI dir is: /home/ubuntu/openvpn-ca/pki
9
10 $ la
11 easyrsa openssl-easyrsa.cnf pki vars.example x509-types
```

Ora si devono personalizzare le variabili `vars`, si può sia partire da un file vuoto oppure modificare `vars.example` per poi rinominarlo `vars`. Andiamo quindi a creare un nuovo file `vars`:

Server CA

```
1 $ vim vars
2 set_var EASYRSA_REQ_COUNTRY "IT"
3 set_var EASYRSA_REQ_PROVINCE "MC"
4 set_var EASYRSA_REQ_CITY "Recanati"
5 set_var EASYRSA_REQ_ORG "Esse-ti"
6 set_var EASYRSA_REQ_EMAIL "s.gasparrini@esse-ti.it"
7 set_var EASYRSA_REQ_OU "Esse-ti"
8 set_var EASYRSA_REQ_CN "openvpn-ca"
9
10 set_var EASYRSA_ALGO "ec"
11 set_var EASYRSA_DIGEST "sha512"
```

Le variabili nel primo blocco determinano i dati che poi verranno registrati nei certificati.

Le ultime 2 sono opzioni di sicurezza, in particolare si setta l'algoritmo di cifratura

A questo punto si deve lasciare il comando `build-ca` per costruire la CA:

Server CA

```
1 $ ./easyrsa build-ca
2
3 Note: using Easy-RSA configuration from: ./vars
4
5 Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020
6
7 Enter New CA Key Passphrase:
8 Re-Enter New CA Key Passphrase:
9 read EC key
10 writing EC key
11
12 You are about to be asked to enter information that will be incorporated
13 into your certificate request.
14 What you are about to enter is what is called a Distinguished Name or a DN.
15 There are quite a few fields but you can leave some blank
16 For some fields there will be a default value,
17 If you enter '.', the field will be left blank.
18 -----
19 Common Name (eg: your user, host, or server name) [Easy-RSA CA]:
20
21 CA creation complete and you may now import and sign cert requests.
22 Your new CA certificate file for publishing is at:
23 /home/ubuntu/openvpn-ca/pki/ca.crt
24
```

Eseguendo il comando verrà chiesto di inserire una passshare, che verrà usata per criptare la chiave privata appena generata. Il secondo prompt è relativo al nome da dare alla certificazione, in questo caso è stato lasciato il valore di default `Easy-RSA CA`.

3.2.1 Configurazione della PKI di OpenVPN

Il procedimento è simile al precedente, ma questa volta va eseguito sul *server*.

Creiamo quindi una cartella per ospitare la PKI, es `/openvpn-pki`, e linkiamo `easy-rsa`. Inoltre limitiamo i permessi all'utente non root che stiamo usando, in questo caso "ubuntu".

Server

```
1 $ mkdir ~/openvpn-pki
2 $ ln -s /usr/share/easy-rsa/* ~/openvpn-pki/
3 $ sudo chown ubuntu ~/openvpn-pki/
4 $ chmod 700 ~/openvpn-pki/
5 $ cd ~/openvpn-pki/
```

Andiamo a creare un file `vars`:

Server

```
1 $ vim vars
2 set_var EASYRSA_ALGO "ec"
3 set_var EASYRSA_DIGEST "sha512"
```

Concludiamo la creazione della PKI con il comando:

Server

```
1 $ ./easyrsa init-pki
2
3 Note: using Easy-RSA configuration from: ./vars
4
5 init-pki complete; you may now create a CA or requests.
6 Your newly created PKI dir is: /home/ubuntu/openvpn-pki/pki
7
```

A questo punto il server openvpn ha tutti i prerequisiti per creare una sua chiave privata e relativa *Certificate Signing Request*.

Come nome è stato scelto "server":

Server

```
1 $ ./easyrsa gen-req server nopass
2
3 Note: using Easy-RSA configuration from: ./vars
4
5 Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020
6 Generating an EC private key
7 writing new private key to '/home/ubuntu/openvpn-pki/pki/private/server.key.438W2xM0g9'
8 -----
9 You are about to be asked to enter information that will be incorporated
10 into your certificate request.
11 What you are about to enter is what is called a Distinguished Name or a DN.
12 There are quite a few fields but you can leave some blank
13 For some fields there will be a default value,
14 If you enter '.', the field will be left blank.
15 -----
16 Common Name (eg: your user, host, or server name) [server]:
17
18 Keypair and certificate request completed. Your files are:
19 req: /home/ubuntu/openvpn-pki/pki/reqs/server.req
20 key: /home/ubuntu/openvpn-pki/pki/private/server.key
21
```

La chiave `server.key` va copiata nell'apposita cartella.

Server

```
1 $ sudo cp /home/ubuntu/openvpn-pki/pki/private/server.key /etc/openvpn/server/
```

Il secondo file creato, `server.req`, corrisponde ad una *Certificate Signing Request (CSR)* che va firmata e validata dalla CA. In questo modo ogni client che si fida della CA si fiderà di conseguenza del server OpenVPN

3.2.2 Firma del certificato openvpn dalla CA

Dobbiamo quindi copiare il file `server.req` nel *server CA*, possiamo qualunque metodo purchè sia sicuro, ad esempio con `scp`:

```
Server CA
1 $ scp ubuntu@openvpn_server:/home/ubuntu/openvpn-pki/pki/reqs/server.req /tmp
```

Dobbiamo quindi spostarci sul server CA e importare la *certificate request* e firmarlo:

```
Server CA
1 $ cd ~/openvpn-ca
2 $ ./easysrsa import-req /tmp/server.req server
3 $ ./easysrsa sign-req server server
4 Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easysrsa.cnf
5 Check that the request matches the signature
6 Signature ok
7 The Subject's Distinguished Name is as follows
8 commonName          :ASN.1 12:'server'
9 Certificate is to be certified until Mar 11 15:50:45 2025 GMT (1080 days)
10
11 Write out database with 1 new entries
12 Data Base Updated
```

Verrà creato un file in `/openvpn-ca/pki/issued` chiamato `server.crt` che conterrà la chiave pubblica che verrà usata dal server openvpn e inoltre la firma della CA.

Ora si devono copiare i file `ca.crt` e `server.crt` dal *server CA* al *server OpenVPN*:

```
Server
1 $ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/server.crt /tmp
2 $ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/ca.crt /tmp
```

Possiamo quindi tornare sul *server OpenVPN* e copiare i 2 file da `/tmp` a `/etc/openvpn/server`:

```
Server
1 $ sudo cp /tmp/server.crt /etc/openvpn/server
2 $ sudo cp /tmp/ca.crt /etc/openvpn/server
```

3.3 Generazione della *tls-crypt pre-shared key*

Per aumentare ulteriormente la sicurezza del nostro *server OpenVPN* possiamo creare un'ulteriore chiave, che consiste in una chiave *preshared* che verrà inserita in tutte le configurazioni e serve a offuscare il certificato in fase di validazione. Quindi in caso di attacco si dovrà conoscere anche questa chiave.

La creazione va fatta sul *server OpenVPN*:

```
Server
1 $ cd ~/openvpn-pki/
2 $ openvpn --genkey --secret ta.key
```

il file generato `ta.key` dovrà essere copiato nella directory del server openvpn:

```
Server
1 $ sudo cp ta.key /etc/openvpn/server
```

3.4 Generazione delle chiavi per i clients

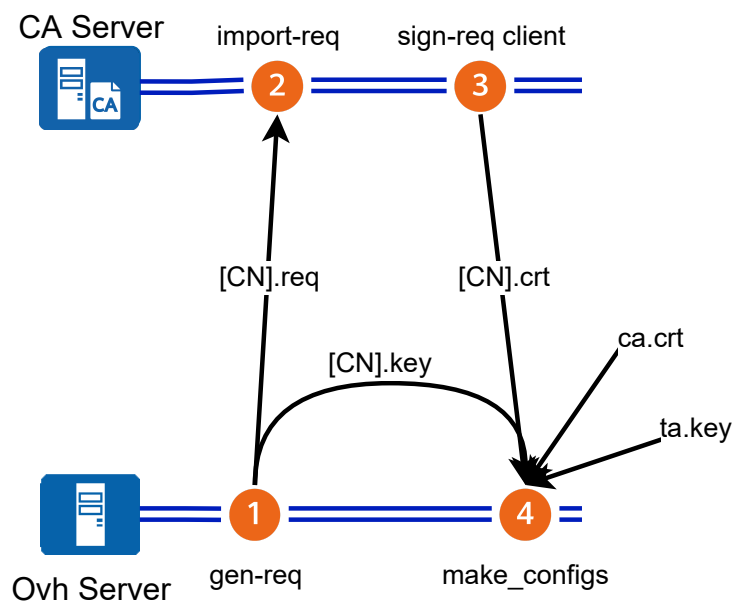


Figura 3.3: Diagramma per schematizzare la procedura di firma di un certificato client. [2]

Creiamo una cartella nella *home* che ospiterà le chiavi dei *client* e le configurazioni openvpn:

```
Server
1 $ mkdir -p ~/client-configs/keys
2 $ chmod -R 700 ~/client-configs
```


Creiamo quindi un certificato per un *client*:

Server

```
1 $ cd ~/openvpn-pki/
2 $ ./easysrsa gen-req client1 nopass
```

Ora dobbiamo copiare `client1.key` nella directory appena creata, e `client1.req` va copiato nel server CA per essere firmato:

Server

```
1 $ cp pki/private/client1.key ~/client-configs/keys/
2 $ scp /home/ubuntu/openvpn-pki/pki/reqs/client1.req ubuntu@ca_server:/tmp
```

Dobbiamo quindi spostarci sul server CA e importare la *certificate request* e firmarla:

Server CA

```
1 $ cd ~/openvpn-ca
2 $ ./easysrsa import-req /tmp/client1.req client1
3 $ ./easysrsa sign-req client client1
4 Using configuration from /home/ubuntu/openvpn-ca/pki/safessl-easysrsa.cnf
5 Check that the request matches the signature
6 Signature ok
7 The Subject's Distinguished Name is as follows
8 commonName          :ASN.1 12:'client1'
9 Certificate is to be certified until Mar 16 13:15:09 2025 GMT (1080 days)
10
11 Write out database with 1 new entries
12 Data Base Updated
```

Per poi ricopiare dal server CA al server openvpn il certificato firmato:

Server

```
1 $ scp ubuntu@ca_server:/home/ubuntu/openvpn-ca/pki/issued/client1.crt /tmp
```

Quindi ci dobbiamo spostare sul server OpenVPN e copiare le chiavi nella cartella `client-configs/keys`, in modo da prepararla per la creazione delle configurazioni OpenVPN. È necessario inoltre cambiare i permessi dei file rendendoli accessibili all'utente Ubuntu:

Server

```
1 $ cp /tmp/client1.crt ~/client-configs/keys/
2 $ cp ~/openvpn-pki/ta.key ~/client-configs/keys/
3 $ sudo cp /etc/openvpn/server/ca.crt ~/client-configs/keys/
4 $ sudo chown ubuntu:ubuntu ~/client-configs/keys/*
```

3.5 Creazione del file di configurazione del server OpenVPN

Il server openvpn viene configurato attraverso `/etc/openvpn/server/server.conf`, per non partire da una configurazione vuota si può usare la configurazione di esempio offerta da openvpn:

```
Server
1 $ cd /etc/openvpn/server/
2 $ sudo wget "https://raw.githubusercontent.com/OpenVPN/openvpn/\
3     master/sample/sample-config-files/server.conf"
```

Dobbiamo quindi modificare il file e cambiare alcune configurazioni, per facilitare la lettura sarà incluso il numero riga modificato:

```
Server
1 $ sudo vim server.conf
2 85 dh none          # non sono stati usati i parametri Diffie-Hellman
3 92 topology subnet  # topologia raccomandata
4 244 ;tls-auth ta.key 0 # This file is secret
5 245 tls-crypt ta.key  # selezione della preshared key
6 253 cipher AES-256-GCM # selezione della cifratura scelta
7 275 user nobody      # utente che eseguirà il server openvpn, in modo da
8                          # restringere i permessi
9 276 group nogroup    # stassa cosa per il gruppo
10 318 auth sha256      # selezione del metodo di autenticazione
```

3.6 Configurazioni sulla network stack del server openvpn

Per abilitare l'*ip forwarding* si dovrà modificare il file `/etc/sysctl.conf`, il comando successivo serve a ricaricare le configurazioni dai file:

```
Server
1 $ sudo vim /etc/sysctl.conf
2 69 net.ipv4.ip_forward = 1
3 $ sudo sysctl -p
4 net.ipv4.ip_forward = 1
```

3.7 Configurazione del firewall

Sulla VPS scelta è presente il firewall `firewalld`, ma per una più semplice configurazione è consigliato di disattivarlo e installare `ufw`:

Server

```
1 $ sudo systemctl mask firewalld
2 $ sudo systemctl stop firewalld
3 $ sudo apt-get install ufw
4 $ sudo ufw allow ssh
5 Rule added
6 Rule added (v6)
7 $ sudo ufw enable
```

È importantissimo ricordarsi di consentire l'SSH prima di abilitare il firewall, altrimenti si perderà l'accesso alla VPS.

3.7.1 Configurazione del NAT

Per far sì che i pacchetti provenienti dalla *VPN* entrino nella network stack del *server* si deve aggiungere una regola di *NAT* nel firewall. Per farlo si deve conoscere quale è l'interfaccia di rete del *server*, cioè quella che ha come ip il suo ip pubblico:

Server

```
1 $ ip addr
2 [...]
3 2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
   ↪ 1000
4     link/ether a6:23:5f:48:ba:de brd ff:ff:ff:ff:ff:ff
5     inet 51.178.141.119/20 brd 51.178.141.255 scope global dynamic ens3
6         valid_lft 1857sec preferred_lft 1857sec
7     inet6 fe80::23:bfff:ac24:aace/64 scope link
8         valid_lft forever preferred_lft forever
9 [...]
```

In questo caso il nome dell'interfaccia di rete è *ens3*, possiamo quindi procedere con la configurazione del firewall, si andrà a modificare il file `/etc/ufw/before.rules` e aggiungere la regola di NAT:

Server

```
1 $ sudo vim /etc/ufw/before.rules
2 # ## rules.before
3 # ## Rules that should be run before the ufw command line added rules. Custom
4 # rules should be added to one of these chains:
5 # ufw-before-input
6 # ufw-before-output
7 # ufw-before-forward
8 #
9
10 # START OPENVPN RULES
11 # NAT table rules
12 *nat
13 :POSTROUTING ACCEPT [0:0]
14 # Allow traffic from OpenVPN client to ens3
15 -A POSTROUTING -s 10.8.0.0/24 -o ens3 -j MASQUERADE
16 COMMIT
17 # END OPENVPN RULES
18
19
20 # Don't delete these required lines, otherwise there will be errors
21 *filter
22 . . .
```

Nella modifica del file si deve stare attenti a inserire la nuova regola in cima al file e sotto i commenti iniziali, è inoltre importante inserire i commenti nella regola.

3.7.2 Configurazione del packet forwarding

Precedentemente abbiamo abilitato il forwarding nella network stack del server, ora si deve abilitare la corrispondente opzione nel firewall. Si deve quindi cambiare la regola di default per i pacchetti inoltrati da `DROP` a `ACCEPT`.

Per farlo si deve modificare il file `/etc/default/ufw`:

Server

```
1 $ sudo vim /etc/default/ufw
2 DEFAULT_FORWARD_POLICY="ACCEPT"
```

3.7.3 Conclusione della configurazione del firewall

Per concludere la configurazione si deve abilitare la porta relativa alla vpn, in questo caso `1194`, e riavviare il firewall:

Server

```
1 $ sudo ufw allow 1194/udp
2 $ sudo ufw reload
3 $ sudo ufw status
4 Status: active
5 To Action From
6 --
7 22 ALLOW Anywhere
8 1194/udp ALLOW Anywhere
9 22 (v6) ALLOW Anywhere (v6)
10 1194/udp (v6) ALLOW Anywhere (v6)
```

3.8 Avvio del server OpenVPN

Ora che la configurazione del server è in una situazione stabile possiamo avviarlo:

Server

```
1 $ sudo systemctl enable openvpn-server@server.service
2 $ sudo systemctl start openvpn-server@server.service
3 $ sudo systemctl status openvpn-server@server.service
4 • openvpn-server@server.service - OpenVPN service for server
5   Loaded: loaded (/usr/lib/systemd/system/openvpn-server@.service; enabled; vendor
6   ↳ preset: disabled)
7   Active: active (running) since Mon 2022-04-18 13:08:44 CEST; 4h 22min ago
8   Docs: man:openvpn(8)
9         https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
10        https://community.openvpn.net/openvpn/wiki/HOWTO
11   Main PID: 436 (openvpn)
12   Status: "Initialization Sequence Completed"
13   Tasks: 1 (limit: 9488)
14   Memory: 4.8M
15   CPU: 199ms
16   CGroup: /system.slice/system-openvpn\x2dserver.slice/openvpn-server@server.service
17           └─436 /usr/bin/openvpn --status /run/openvpn-server/status-server.log
18           ↳ --status-version 2 --suppress-timestamps --config server.conf
19
20 Apr 18 13:08:44 server openvpn[436]: /sbin/ip addr add dev tun0 local 10.8.0.1 peer
21 ↳ 10.8.0.2
22 Apr 18 13:08:44 server openvpn[436]: /sbin/ip route add 10.8.0.0/24 via 10.8.0.2
23 Apr 18 13:08:44 server openvpn[436]: UDPv4 link local (bound): [AF_INET][undef]:1194
24 Apr 18 13:08:44 server openvpn[436]: UDPv4 link remote: [AF_UNSPEC]
25 Apr 18 13:08:44 server openvpn[436]: MULTI: multi_init called, r=256 v=256
26 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
27 Apr 18 13:08:44 server openvpn[436]: IFCONFIG POOL LIST
28 Apr 18 13:08:44 server openvpn[436]: Initialization Sequence Completed
```

Il comando `systemctl enable` abilita il servizio per essere avviato all'avvio della macchina, mentre `systemctl start` lo avvia immediatamente. Con il comando `systemctl status` si può verificare lo stato del servizio, si vede che il servizio è *active (running)*.

3.9 Script per la creazione delle configurazioni dei client

Per facilitare la creazione dei file di configurazione dei client, `clientX.ovpn`, andremo a creare un apposito script bash. Per prima cosa si deve scaricare e personalizzare la configurazione base del client:

```
Server
1 $ cd ~/client-configs/
2 $ wget "https://raw.githubusercontent.com/OpenVPN/openvpn\
3     /master/sample/sample-config-files/client.conf" \
4     -O base.conf
5 $ vim base.conf
6 42  remote 51.178.141.119 1194      # va messo l'ip e la porta del server OpenVPN
7 88  ;ca ca.crt                    # non useremo i file esterni ma ingloberemo
8 89  ;cert client.crt              # questi file in un file direttamente nella
9 90  ;key client.key                # configurazione del client
10 108 ;tls-auth ta.key 1            #
11 116 cipher AES-256-GCM           # cifratura usata
12 117 auth SHA256                  # autenticazione usata
13 118 key-direction 1              # indica che e' un client
```

Ora creiamo lo script bash `make_config.sh`:

```
Server
1 $ vim make_config.sh
2 #!/bin/bash
3
4 # usage:
5 # $ make_config.sh client1
6 # will use [ca.crt, client1.crt, client1.key, ta.key] to create client1.ovpn
7
8 KEY_DIR=~/client-configs/keys
9 OUTPUT_DIR=~/client-configs/files
10 BASE_CONFIG=~/client-configs/base.conf
11
12 cat ${BASE_CONFIG} \
13     <(echo -e '<ca>' ) \
14     ${KEY_DIR}/ca.crt \
15     <(echo -e '</ca>\n<cert>' ) \
16     ${KEY_DIR}/${1}.crt \
17     <(echo -e '</cert>\n<key>' ) \
18     ${KEY_DIR}/${1}.key \
19     <(echo -e '</key>\n<tls-crypt>' ) \
20     ${KEY_DIR}/ta.key \
21     <(echo -e '</tls-crypt>' ) \
22     > ${OUTPUT_DIR}/${1}.ovpn
23 $ chmod 700 make_config.sh
```

Lo scopo di questo script è di aggiungere al file `base.conf` il certificato della CA, `ca.crt`, il certificato e chiave relativi al client per cui si sta creando la configurazione, passato come

argomento allo script, e la *presheared key*.

Il tutto viene scritto in un file che ha lo stesso nome del *client* per cui si sta creando la configurazione ma `.ovpn`.

Quindi per creare la configurazione di *client 1*:

```
Server
1 $ ./make_config.sh client1
```

Nella cartella `client-configs/files/` si troverà il file di configurazione per il client `client1.ovpn`.

3.10 Test della configurazione

Ora che abbiamo un file di configurazione per il client, possiamo testare che la configurazione fino a questo punto sia corretta. Per farlo ci spostiamo su una macchina client, con SO Linux ad esempio, e si può connettere il *client* alla vpn con la configurazione creata al passo precedente:

```
Client
1 $ sudo openvpn --config client1.ovpn
2 [...]
3 Thu Apr 21 12:53:04 2022 Outgoing Data Channel: Cipher 'AES-256-GCM' initialized with 256
  ↳ bit key
4 Thu Apr 21 12:53:04 2022 Incoming Data Channel: Cipher 'AES-256-GCM' initialized with 256
  ↳ bit key
5 Thu Apr 21 12:53:04 2022 ROUTE_GATEWAY 192.168.1.20/255.255.255.0 IFACE=eth0
  ↳ HWADDR=02:42:0a:00:04:03
6 Thu Apr 21 12:53:04 2022 /sbin/ip route add 10.8.0.1/32 via 10.8.0.2
7 Thu Apr 21 12:53:04 2022 WARNING: this configuration may cache passwords in memory -- use
  ↳ the auth-nocache option to prevent this
8 Thu Apr 21 12:53:04 2022 Initialization Sequence Completed
```

Se la configurazione fino a questo punto è corretta si avrà il messaggio

`Initialization Sequence Completed`.

Nel *client* si avrà una nuova interfaccia di rete chiamata `tun0`, questa è l'interfaccia virtuale creata dalla vpn.

```
Client
1 $ ip addr
2 2: tun0: <MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
  ↳ default qlen 500
3     link/none
4     inet 10.8.0.2/24 scope global tun0
5         valid_lft forever preferred_lft forever
```

Si può vedere come l'ip assegnato al *client* dalla vpn è `10.8.0.2`.

Per testare che la connessione sia instaurata correttamente si può usare la utility `ping`, ad esempio possiamo fare il ping dal *client* verso l'ip interno alla vpn del *server*:

Client

```
1 $ ping -c2 10.8.0.1
2 PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
3 64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=0.250 ms
4 64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=0.220 ms
```

Se nel frattempo si esegue la utility `tcpdump` sul server si potranno vedere i pacchetti *echo request* ed *echo reply*:

Server

```
1 $ sudo tcpdump
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 13:11:12.018615 IP 10.8.0.2 > 10.8.0.1: ICMP echo request, id 11, seq 1, length 64
4 13:11:12.018640 IP 10.8.0.1 > 10.8.0.2: ICMP echo reply, id 11, seq 1, length 64
5 13:11:13.039993 IP 10.8.0.2 > 10.8.0.1: ICMP echo request, id 11, seq 2, length 64
6 13:11:13.040018 IP 10.8.0.1 > 10.8.0.2: ICMP echo reply, id 11, seq 2, length 64
```

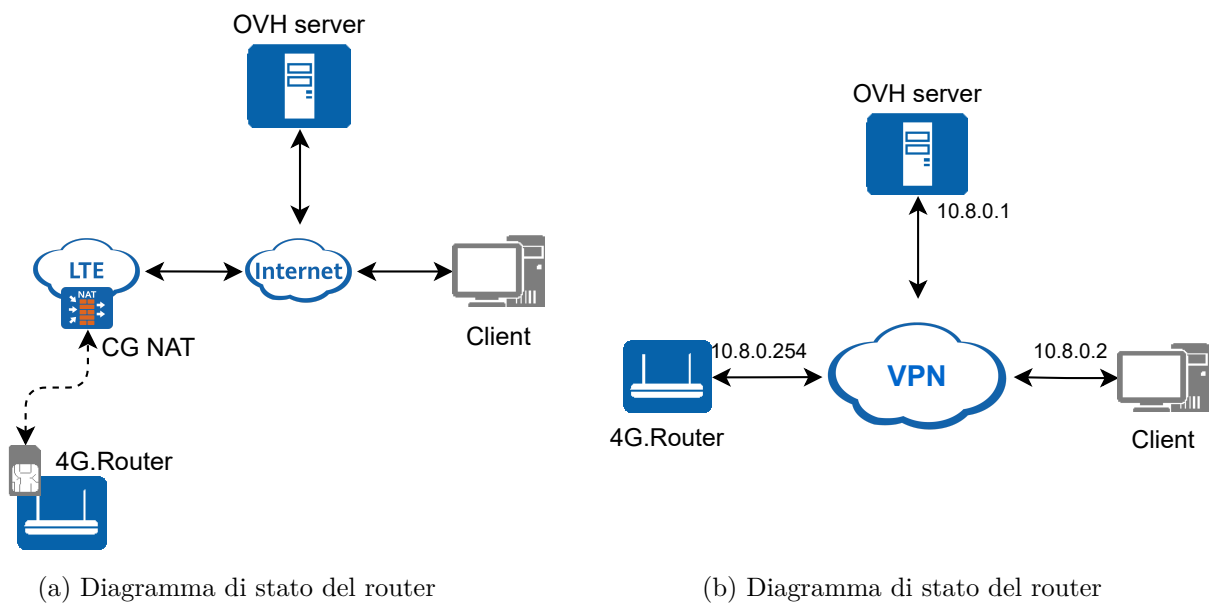
Si vede quindi che è possibile una comunicazione bidirezionale tra *client*, `10.8.0.2`, e *server*, `10.8.0.1`.

Capitolo 4

Configurazione Router

4.1 Overview della configurazione

In questo capitolo andremo a configurare il *router 4g* e connetterlo alla VPN.



4.2 Creazione della configurazione Openvpn

Si devono seguire gli step descritti in sezione 3.4, quindi creare la certificate request e firmarla nel *server CA*. Per poi usare lo script creato in sezione 3.9 per costruire il file di configurazione:

```
Server
1 $ ./make_config.sh router
```

Dopodiché si deve spostare il file `router.ovpn` dal *Server* al *Router 4g*, supponiamo di averlo copiato nella cartella `/configs`.

Di default non è presente *OpenVPN* nel *Router 4g*, lo si può installare con:

```
Router 4g
1 $ opkg update
2 $ opkg install openvpn
3 $ opkg install luci-app-openvpn
```

Ora possiamo avviare il client openvpn:

```
Router 4g
1 $ openvpn --config /configs/router.ovpn
2 2022-04-29 17:26:37 OpenVPN 2.5.6 x86_64-openwrt-linux-gnu [SSL (mbed TLS)] [LZ4] [EPOLL]
   ↪ [MH/PKTINFO] [AEAD]
3 [...]
4 2022-04-29 17:26:37 VERIFY ECU OK
5 2022-04-29 17:26:37 VERIFY OK: depth=0, CN=server
6 2022-04-29 17:26:37 Control Channel: TLSv1.2, cipher
   ↪ TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384, 2048 bit key
7 2022-04-29 17:26:37 [server] Peer Connection Initiated with [AF_INET]10.0.4.2:1194
8 2022-04-29 17:26:37 net_addr_ptp_v4_add: 10.8.0.10 peer 10.8.0.9 dev tun0
9 2022-04-29 17:26:37 Initialization Sequence Completed
```

Se il file di configurazione è stato creato correttamente si vedrà il messaggio `Initialization Sequence Completed`.

Comparirà inoltre l'interfaccia `tun0` a cui è stato assegnato l'indirizzo `10.8.0.3`.

Per abilitare l'autostart di openvpn per il router si deve, per prima cosa, modificare il file `/etc/config/openvpn` in modo che faccia riferimento alla config corretta:

```
Router 4g
1 $ vim /etc/config/openvpn
2 20 option config /configs/router.ovpn
```

Ora possiamo abilitarla usando luci:

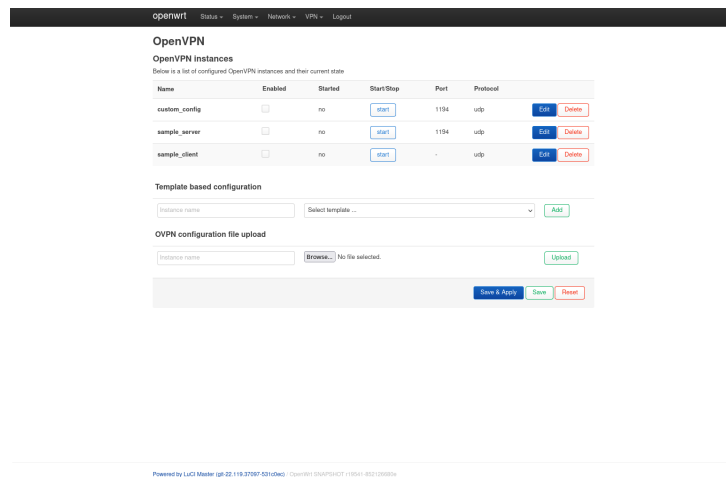


Figura 4.2: Configurazione della VPN tramite LuCI

Si deve mettere il check su *enabled* e premere start, per poi salvare le modifiche. In questo modo il router si conatterà automaticamente alla VPN anche se venisse riavviato.

4.3 Abilitazione del Client-to-Client nel server OpenVPN

In questo momento i client della VPN, *client1* e *router*, possono comunicare tra loro, ma lo fanno passando per la network stack del *server*. Infatti:

```
Router 4g
1 $ ping -c2 10.8.0.2 # client1
2 PING 10.8.0.2 (10.8.0.2): 56 data bytes
3 64 bytes from 10.8.0.2: seq=0 ttl=63 time=0.519 ms
4 64 bytes from 10.8.0.2: seq=1 ttl=63 time=0.501 ms
5
6 --- 10.8.0.2 ping statistics ---
7 2 packets transmitted, 2 packets received, 0% packet loss
8 round-trip min/avg/max = 0.501/0.510/0.519 ms
```

Dal server possiamo vedere i pacchetti con *tcpdump*:

```
Server
1 $ sudo tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
3 16:20:50.791063 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
4 16:20:50.791098 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 0, length 64
5 16:20:50.791273 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
6 16:20:50.791285 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 0, length 64
7 16:20:51.791153 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
8 16:20:51.791174 IP 10.8.0.3 > 10.8.0.2: ICMP echo request, id 1759, seq 1, length 64
9 16:20:51.791365 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64
10 16:20:51.791374 IP 10.8.0.2 > 10.8.0.3: ICMP echo reply, id 1759, seq 1, length 64
```

Si vede che ogni richiesta viene duplicata, la prima è in entrata sulla network stack del *server* e la seconda in uscita. Per evitare questo traffico possiamo abilitare l'opzione `client-to-client` nel file di configurazione del server. In questo modo il layer openvpn effettuerà direttamente il forwarding tra i client della vpn [7].

```
Server
1 $ vim /etc/openvpn/server/server.conf
2 209 client-to-client
3 $ sudo systemctl restart openvpn-server@server.service
```

Possiamo quindi rieseguire gli stessi test fatti sopra:

```
Router 4g
1 $ ping -c2 10.8.0.2
2 PING 10.8.0.2 (10.8.0.2): 56 data bytes
3 64 bytes from 10.8.0.2: seq=0 ttl=64 time=0.351 ms
4 64 bytes from 10.8.0.2: seq=1 ttl=64 time=0.307 ms
5
6 --- 10.8.0.2 ping statistics ---
7 2 packets transmitted, 2 packets received, 0% packet loss
8 round-trip min/avg/max = 0.307/0.329/0.351 ms
```

Ma questa volta la network stack del *server* non vede nessun pacchetto:

```
Server
1 $ sudo tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

4.4 Assegnazione ip statico al Router

Dato che non possiamo sapere a priori quanti client si connetteranno contemporaneamente alla VPN, per sapere sempre quale ip viene assegnato al router è necessario assegnargliene uno statico.

Questo viene fatto usando la `client-config-dir`.

```
Server
1 $ sudo mkdir /etc/openvpn/server/ccd # creo la client-config-dir
2 $ sudo vim /etc/openvpn/server/server.conf # abilito l'opzione nella config del server
3 167 client-config-dir ccd
4 $ sudo vim /etc/openvpn/server/ccd/router
5 ifconfig-push 10.8.0.254 255.255.255.0 # impongo l'ip per questo common name
6 $ sudo systemctl restart openvpn-server@server.service
```

Così facendo al router gli verrà sempre assegnato l'ip `10.8.0.254`, indipendentemente dall'or-

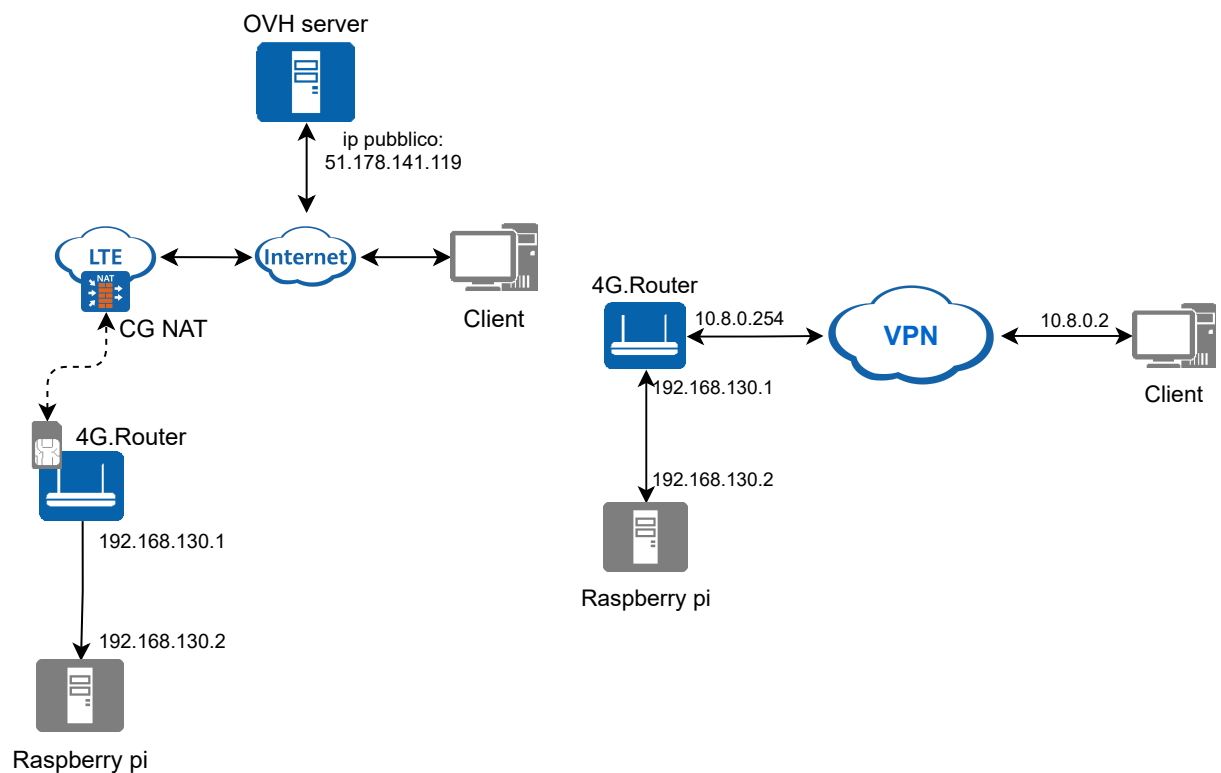
dine in cui gli host si connettono alla vpn. Ciò ci permette di sapere sempre e a priori qual è l'ip del router.

Capitolo 5

Connessione degli host domotici alla VPN

5.1 Overview della configurazione

Per simulare gli host domotici è stata usata una raspberry pi.



(a) Diagramma di stato degli host domotici

(b) Diagramma di stato degli host domotici

5.2 Creazione della firewall zone per la VPN

Per rendere possibile la comunicazione tra la rete *lan* del *Router* e la VPN è necessario configurare opportunamente il firewall.

Ciò viene fatto direttamente da luci, nella sezione firewall si avranno preconfigurate 2 zone, lan e wan, come si vede figura 5.2.

openwrt Status System Network VPN Logout

General Settings Port Forwards Traffic Rules NAT Rules

Firewall - Zone Settings

The firewall creates zones over your network interfaces to control network traffic flow.

General Settings

Enable SYN-flood protection ☒

Drop invalid packets ☐

Input: accept

Output: accept

Forward: reject

Zones

Zone →	Forwardings	Input	Output	Forward	Masquerading	
lan	→ wan	accept	accept	accept	<input type="checkbox"/>	Edit Delete
wan	→ REJECT	reject	accept	reject	<input checked="" type="checkbox"/>	Edit Delete

Add

Save & Apply Save Reset

Figura 5.2: Configurazione iniziale del firewall

Si deve quindi aggiungere una nuova zona, chiamata vpn, con le seguenti opzioni:

- Policy di forward: accept
- Forward consentito verso la zona lan
- Forward consentito dalla zona lan

Possiamo vedere la pagina di configurazione:

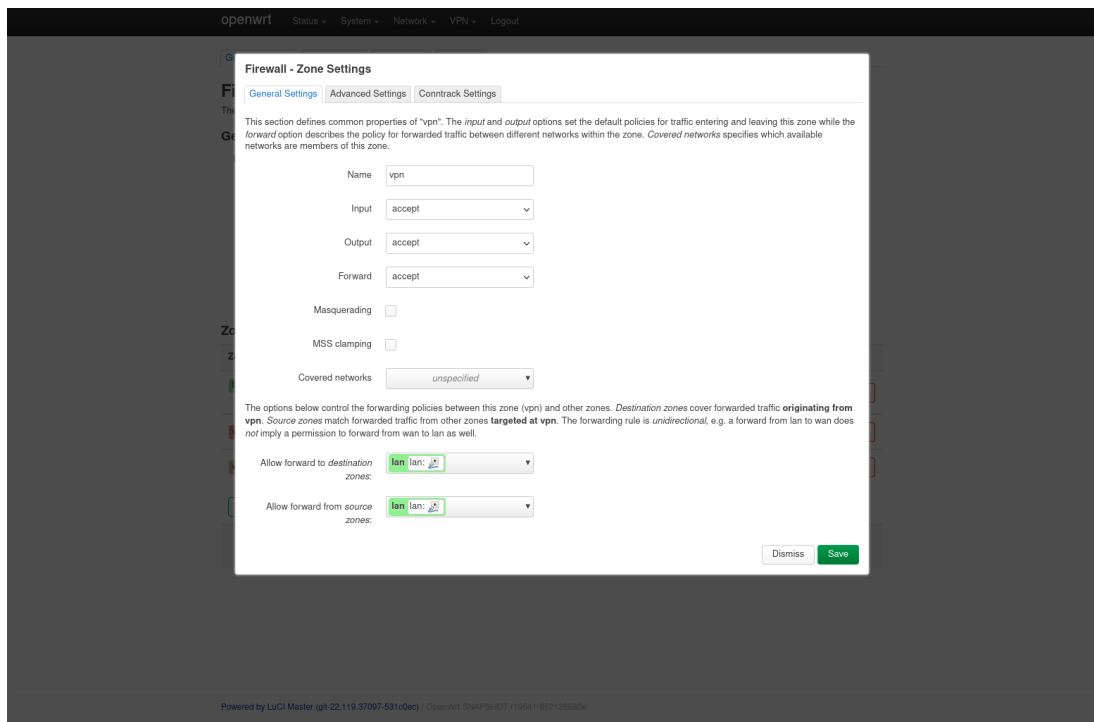


Figura 5.3: Configurazione del firewall

Dopo aver salvato, la pagina del firewall sarà:

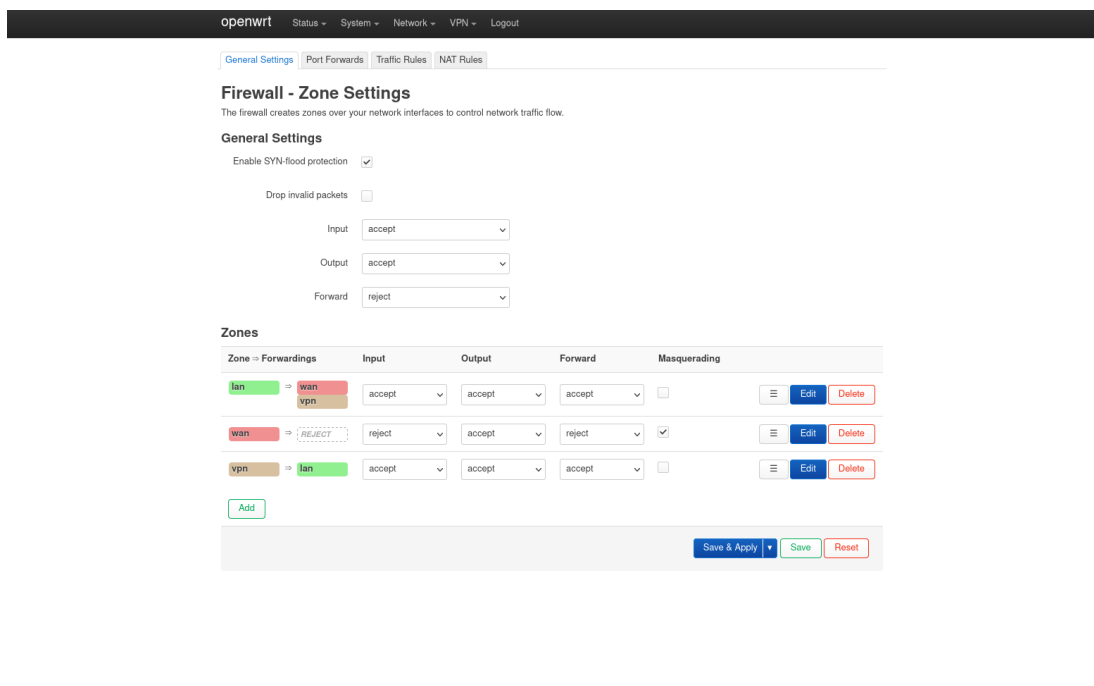


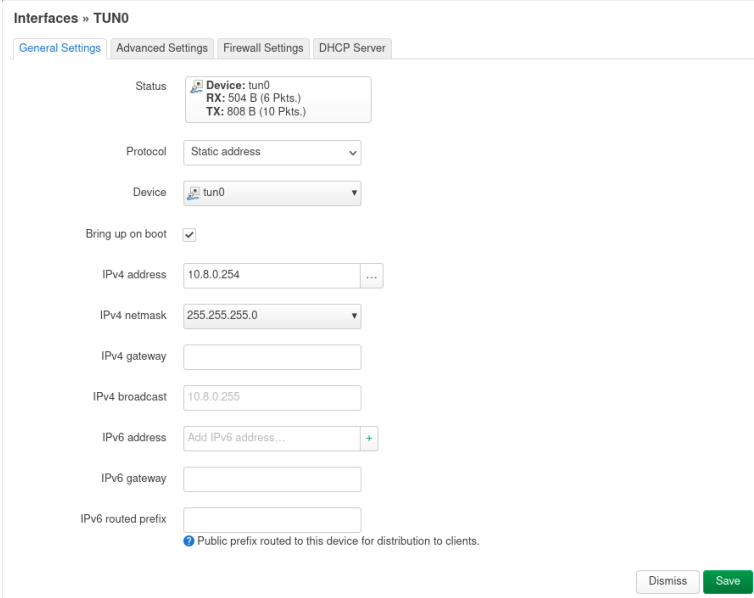
Figura 5.4: Configurazione finale del firewall

5.3 Aggiunta dell'interfaccia tun0 alla zona vpn

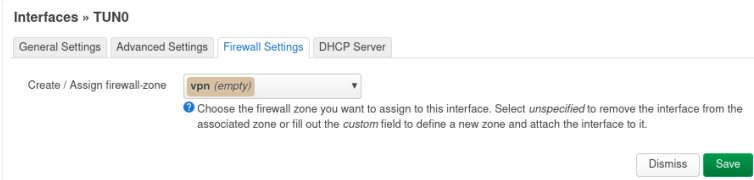
Per rendere attiva la zona firewall appena creata gli si deve assegnare almeno un'interfaccia, in questo caso gli si deve assegnare l'interfaccia tun0.

Ciò va fatto da luci, nella sezione *interfaces*. Si deve quindi aggiungere una nuova interfaccia con le seguenti opzioni:

- Name: tun0
- Proto: static
- Device: tun0
- ipv4 address: 10.8.0.254
- ipv4 netmask: 255.255.255.0
- Assign firewall zone: vpn



(a) General settings



(b) Firewall settings

5.4 Modifiche alla configurazione OpenVPN del server

Per instaurare una comunicazione bidirezionale tra *host-domotico* e *client* è necessario che:

1. il server vpn sia consapevole che il *router* vuole esporre una sua sottorete verso la vpn

2. il client abbia l'opportuna rotta per raggiungere la sottorete dove si trova l'*host-domotico*

Per il punto 1 è necessario aggiungere la seguente riga nel file

`/etc/openvpn/server/ccd/router`

```
Server
1 $ vim /etc/openvpn/server/ccd/router
2 ifconfig-push 10.8.0.254 255.255.255.0
3 iroute 192.168.130.0 255.255.255.0
4 $ vim /etc/openvpn/server/server.conf
5 169 route 192.168.130.0 255.255.255.0 # aggiungo la rotta nel server
```

Per il punto 2 è necessario aggiungere la seguente riga alla configurazione OpenVPN nel *Server*:

```
Server
1 $ vim /etc/openvpn/server/server.conf
2 168 push "route 192.168.130.0 255.255.255.0"
```

Dopo aver modificato la configurazione del server va riavviarlo con `systemctl restart`

In questo modo nel *client1* verra' aggiunta una nuova rotta nel momento in cui si connette alla vpn:

```
Client
1 $ ip route
2 [...]
3 192.168.130.0/24 via 10.8.0.1 dev tun0
```

5.5 Test della configurazione

Per testare la connessione tra *client* e *host-domotico* usiamo sia `ping` che `tracert`, possiamo inoltre usare `tcpdump` sul *Server* e *Router* per vedere il percorso dei pacchetti.

Vediamo quindi un ping dal client verso l'*host-domotico*, con server e router in ascolto sull'interfaccia tun0:

```
Client
1 $ ping -c 1 192.168.130.3
2 PING 192.168.130.3 (192.168.130.3) 56(84) bytes of data.
3 64 bytes from 192.168.130.3: icmp_seq=1 ttl=63 time=0.643 ms
4
5 --- 192.168.130.3 ping statistics ---
6 1 packets transmitted, 1 received, 0% packet loss, time 0ms
7 rtt min/avg/max/mdev = 0.643/0.643/0.643/0.000 ms
```

Server

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

Il server non ha visto traffico poichè ha l'opzione `client-to-client` abilitata, ciò comporta che il layer vpn effettua il routing senza passare per l'interfaccia di rete del router.

Router

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 10:44:07.228387 IP 10.8.0.2 > host-domotico: ICMP echo request, id 12, seq 1, length 64
4 10:44:07.228550 IP host-domotico > 10.8.0.2: ICMP echo reply, id 12, seq 1, length 64
```

Il router vede il pacchetto e lo instrada verso l'host corretto.

Provando nella direzione inversa si ottiene lo stesso risultato

host-domotico

```
1 $ ping -c 1 10.8.0.2
2 PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
3 64 bytes from 10.8.0.2: icmp_seq=1 ttl=63 time=0.428 ms
4
5 --- 10.8.0.2 ping statistics ---
6 1 packets transmitted, 1 received, 0% packet loss, time 0ms
7 rtt min/avg/max/mdev = 0.428/0.428/0.428/0.000 ms
```

e il router vede:

Router

```
1 $ tcpdump -i tun0
2 listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
3 11:56:10.469901 IP host-domotico > 10.8.0.2: ICMP echo request, id 14, seq 1, length 64
4 11:56:10.470230 IP 10.8.0.2 > host-domotico: ICMP echo reply, id 14, seq 1, length 64
```

Possiamo inoltre usare `tracpath` per vedere gli hop:

Client

```
1 $ tracpath 192.168.130.3
2 1?: [LOCALHOST] pmtu 1500
3 1: 10.8.0.254 0.471ms
4 1: 10.8.0.254 0.473ms
5 2: 192.168.130.3 0.567ms reached
6 Resume: pmtu 1500 hops 2 back 2
```

host-domotico

```
1 $ tracepath 10.8.0.2
2 1?: [LOCALHOST] pmtu 1500
3 1: router-1 0.062ms
4 1: router-1 0.068ms
5 2: 10.8.0.2 0.592ms reached
6 Resume: pmtu 1500 hops 2 back 2
```

Bibliografia

- [1] R. Braden. Requirements for internet hosts - communication layers. STD 3, RFC Editor, October 1989. <http://www.rfc-editor.org/rfc/rfc1122.txt>.
- [2] Huawei. Product image gallery. https://info.support.huawei.com/network/imagelib/getImagePartList?product_family=Router&product_type=Access%20Router%7CIOT%20Gateway&domain=&lang=en.
- [3] OpenWrt. Luci essentials. <https://openwrt.org/docs/guide-user/luci/luci.essentials>.
- [4] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc791.txt>.
- [5] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc793.txt>.
- [6] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018.
- [7] ServerFault. Descrizione openvpn client-to-client. <https://serverfault.com/a/738558/558773>.
- [8] Wikipedia. Ethernet ii. https://en.wikipedia.org/wiki/Ethernet_frame#Ethernet_II.