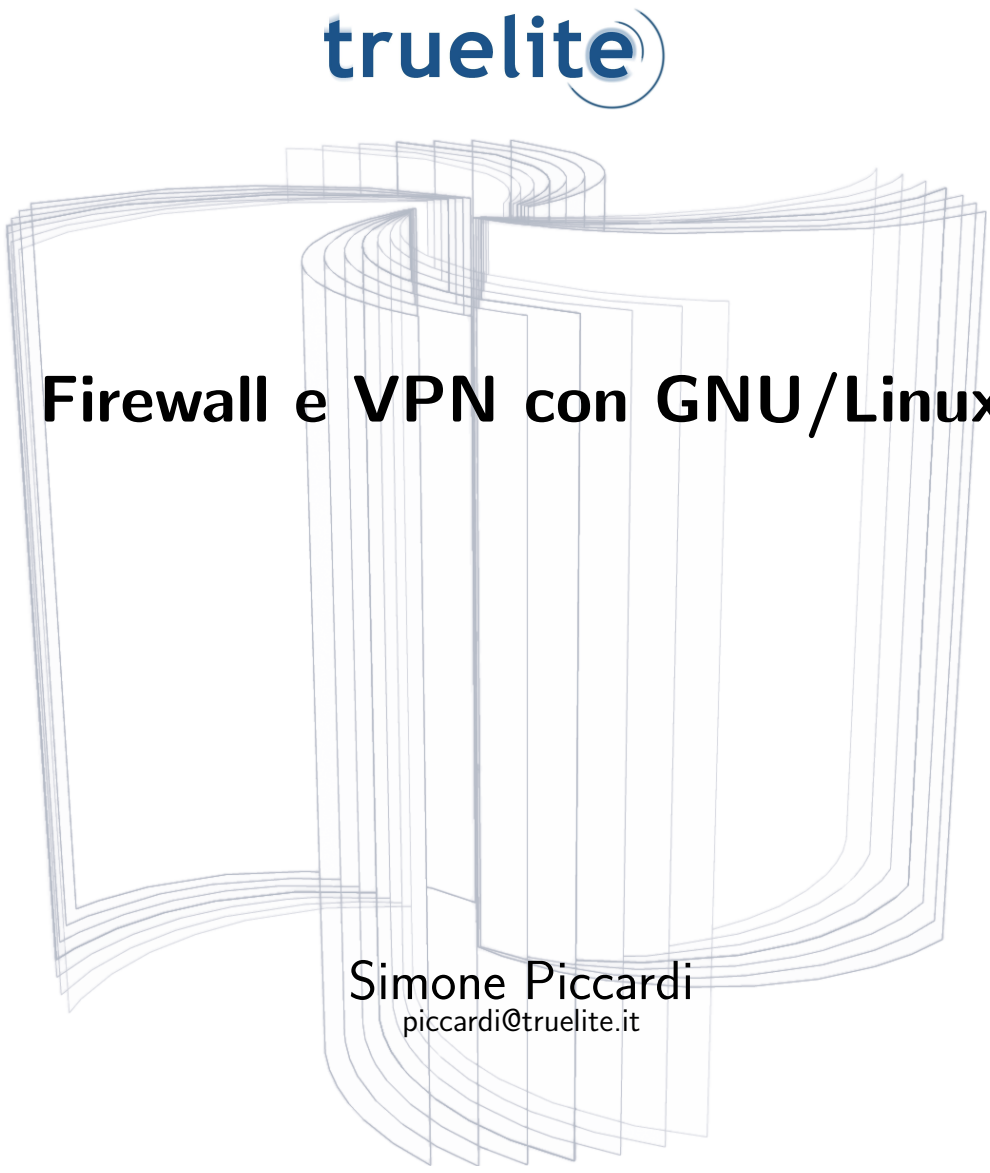


Firewall e VPN con GNU/Linux

Simone Piccardi
piccardi@truelite.it

Truelite Srl http://www.truelite.it info@truelite.it

Firewall e VPN con GNU/Linux – Prima edizione

Questa documentazione libera è stata sviluppata all'interno delle attività formative effettuate da Truelite S.r.l. Il materiale è stato finanziato nel corso della realizzazione dei corsi erogati dall'azienda, e viene messo a disposizione di tutti sotto licenza GNU FDL.

Questo testo, insieme al resto della documentazione libera realizzata da Truelite S.r.l., viene distribuito su internet all'indirizzo:

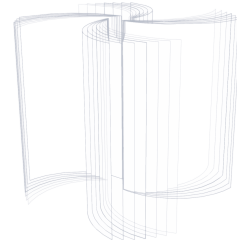
`http://svn.truelite.it/truedoc`

dove saranno pubblicate nuove versioni ed aggiornamenti.

Società italiana specializzata nella fornitura di servizi, consulenza e formazione esclusivamente su GNU/Linux e software libero.

Per informazioni:

Truelite S.r.l
Via Monferrato 6,
50142 Firenze.
Tel: 055-7879597
Fax: 055-7333336
e-mail: info@truelite.it
web: http://www.truelite.it



Indice

Capitolo 1

Firewall

1.1 Cenni di teoria dei firewall

In questa sezione prenderemo in esame le basi teoriche ed i criteri generali che si applicano alla realizzazione di un firewall; in particolare introdurremo alcuni concetti essenziali di sicurezza delle reti, definiremo il ruolo dei firewall in questo ambito e prenderemo in esame i criteri per la loro dislocazione.

1.1.1 Un'introduzione alla sicurezza delle reti

Prima di passare alla descrizione dei firewall e dei relativi criteri di impiego è opportuno fare una breve introduzione alle tematiche della sicurezza delle reti, la cui comprensione è necessaria per poter realizzare un uso efficace dei firewall.

In un mondo ideale infatti i firewall sono completamente inutili. Se tutto il software che si pone in uso non presentasse problemi di sicurezza, se tutte le configurazioni utilizzate fossero corrette e se tutti i meccanismi di controllo non fossero eludibili, sarebbe impossibile avere accessi non autorizzati e pertanto i firewall non servirebbero a nulla.

Nel mondo reale però tutti i programmi hanno errori e presentano vulnerabilità, le configurazioni vengono sbagliate, il software viene installato male, i controlli vengono elusi, il che comporta la possibilità che, nelle forme più varie, diventino possibili accessi non autorizzati.

In genere comunque i problemi di sicurezza più comuni sono relativi ad errori di programmazione. Il caso più comune è quello di un mancato (o incompleto) controllo dei dati che vengono inviati al programma nella supposizione, sbagliata, che questi abbiano una certa forma. Per cui inviando dati in una forma non prevista si può bloccare il funzionamento dello stesso, o anche fargli eseguire del codice estraneo (si ricordi quanto visto in sez. **??**).

Buona parte di questi problemi possono essere superati in maniera molto efficace anche senza ricorrere all'uso di un firewall. La riduzione all'essenziale dei programmi da installare (parafrasando Ford un servizio non installato non si può violare), scelta di programmi stabili e realizzati con attenzione ai problemi di sicurezza, la costanza nell'eseguire gli aggiornamenti (si stima che bel oltre il 90% delle violazioni di sicurezza sia dovuta all'uso versioni vulnerabili non aggiornate) l'attenzione a tutti gli aspetti delle configurazioni (ed in particolare l'uso delle

funzionalità di controllo degli accessi che tutti i principali programmi mettono a disposizione) bastano da soli a rendere ragionevolmente sicura una macchina.

Detto questo uno potrebbe allora chiedersi a che pro installare un firewall, visto che almeno in teoria se ne potrebbe fare a meno. In realtà uno dei criteri fondamentali nella sicurezza è sempre quello della ridondanza, per cui anche la semplice duplicazione con un meccanismo indipendente dei controlli di accesso sarebbe di per sé una buona ragione. A questo poi si può aggiungere il fatto che un firewall permette di costruire un punto di accesso unificato per il traffico di rete, permettendone un controllo generale e non limitato alle funzionalità (che potrebbero non esistere neanche) dei singoli programmi o delle singole macchine.

1.1.2 Cosa è un firewall

Il primo passo per addentrarsi nella pratica delle tecnologia di protezione basate su firewall è allora quello di capire cos'è un firewall, cosa fa e a cosa serve. Troppo spesso infatti il firewall è ritenuto un po' come la panacea di tutti i mali, e si pensa che basti installarne uno per risolvere tutti i problemi di sicurezza di una rete.

Un firewall, come la parola suggerisce, è una sorta di porta blindata che permette di bloccare i tentativi di accesso ad una rete, anche se in realtà la parola inglese fa più propriamente riferimento alle porte tagliafuoco usate per bloccare la propagazione degli incendi. Ma come una porta blindata serve a poco se le finestre sono senza inferriate, ancora meno se la chiave della serratura è sotto lo zerbino, e per niente se i ladri li fate entrare voi, lo stesso vale per un firewall.

Un firewall posto fra voi ed Internet cioè non sarà mai in grado di proteggervi da attacchi provenienti da altri accessi meno controllati della vostra rete, ad esempio perché qualcuno ha attaccato un modem al suo PC e apre una connessione da lì, o c'è una rete wireless vulnerabile e qualcuno ci si aggancia. E un firewall non è in grado di proteggervi dalle vulnerabilità delle applicazioni cui dovete accedere da remoto (se ad esempio un servizio aperto è vulnerabile, il firewall non vi sarà di nessun aiuto), né tantomeno dagli attacchi portati da utenti interni alla rete protetta.

Un firewall dunque non è affatto il sostituto di una buona politica di sicurezza, di cui costituisce solo uno (per quanto importante) degli strumenti. Lo scopo specifico di un firewall è in realtà solo quello di permettervi di suddividere opportunamente la vostra rete, in modo da potervi applicare politiche di sicurezza diverse (più o meno restrittive) a seconda delle differenti esigenze di ciascuna parte.

Un altro aspetto dei firewall che è bene comprendere è che benché questi possano essere usati per filtrare il traffico di rete, il livello a cui possono farlo spesso è molto superficiale. Un firewall infatti potrà bloccare il traffico proveniente da certi indirizzi o diretto a certi servizi, ma di norma un firewall non è assolutamente in grado di riconoscere il contenuto del traffico di rete.

Se pertanto si vuole fare del *content filtering* (ad esempio filtrare le pagine web in base al loro contenuto o la posta per eliminare dei virus) occorrerà utilizzare uno strumento dedicato in grado di esaminare e classificare il contenuto dei protocolli di livello superiore e questo non è di norma un firewall, e anche se ci sono programmi chiamati "firewall" che hanno di queste funzionalità, tecnicamente non sono qualificabili come tali.

Un firewall infatti agisce ai livelli più bassi dei protocolli di rete e non è in grado di analizzare il contenuto dei pacchetti che riceve,[1] e nelle forme più elementari esso è in grado di operare solo in base ad alcuni dati presenti nel singolo pacchetto, andando ad osservare i valori contenuti nelle intestazioni dei vari protocolli di rete. I firewall più elementari di questo tipo sono detti *stateless*.

Con l'evolversi della tecnologia ci è resi conto che la sola selezione in base al contenuto delle intestazioni dei pacchetti non sempre è sufficiente ed esistono situazioni in cui può essere necessario identificare del traffico che non è riconoscibile solo in base al contenuto del singolo pacchetto. I firewall più evoluti in grado di superare questo limite sono allora detti *stateful* in quanto sono in grado di classificare, osservando il traffico nel suo insieme, i vari pacchetti in un insieme di stati,[2] permettendo la selezione in base a questi ultimi.

Si tenga presente che questo non ha comunque nulla a che fare con il *content filtering*, e gli stati non vengono definiti in termini del contenuto dei dati dei pacchetti, quanto piuttosto in termini di relazioni fra gli stessi, come ad esempio tutti i pacchetti che fanno parte di una certa connessione di rete.

1.1.3 Principi di dislocamento dei firewall

Il primo passo da fare per installare un firewall è quello di prendere carta e penna (o l'equivalente strumento preferito) per fare un bello schema della rete che si vuole proteggere, evidenziando come le varie parti sono connesse fra di loro, dove viene effettuata la connessione verso l'esterno, e decidere quali sono le macchine che devono essere raggiunte dall'esterno e quali no. Inoltre dovranno essere determinate quali sono le interrelazioni fra le varie parti della rete, ad esempio se è proprio necessario che l'amministrazione abbia accesso anche alle macchine del reparto ricerca/sviluppo e viceversa.

Per quanto possa sembrare banale, la decisione sulla *dislocazione* di un firewall è in realtà il cuore del problema, e spesso anche nella più semplice delle configurazioni (quella con un solo accesso alla rete esterna) si rischiano di fare degli errori che possono compromettere la sicurezza; ad esempio ci si può dimenticare di bloccare le connessioni dalle macchine sulla DMZ (vedremo a breve di che si tratta) alla rete interna, aprendo la possibilità di un attacco a partire da quest'ultime.

La forma più elementare di dislocazione di un firewall è quella mostrata in fig. 1.1, dove il firewall serve soltanto a proteggere la rete interna rispetto all'accesso ad internet. Nel caso di Linux se si ha un modem invece che un router esterno, si può fare eseguire anche la condivisione della connessione attraverso la stessa macchina.

Il compito del firewall in questo caso sarà quello di consentire alle connessioni provenienti dalla rete interna di uscire verso internet (ricevendo indietro le relative risposte), ma di bloccare invece le connessioni provenienti dall'esterno e dirette verso l'interno.

Questa è la forma più elementare di firewall, a parte forse il cosiddetto *personal firewall*, in cui si usa un programma di firewall per proteggere una macchina singola da tutte le connessioni

[1] ci sono strumenti dedicati a questo, come gli IDS, che tratteremo nel cap. **??**.

[2] classicamente si intendeva indicare con *stateful* un firewall che riproduce la macchina a stati del protocollo TCP, in realtà il concetto è stato esteso per coprire anche *stati* di protocolli che non supportano neanche il concetto di connessione (cui di solito lo stato fa riferimento) e non è comunque detto che la macchina degli stati debba essere riprodotta interamente.

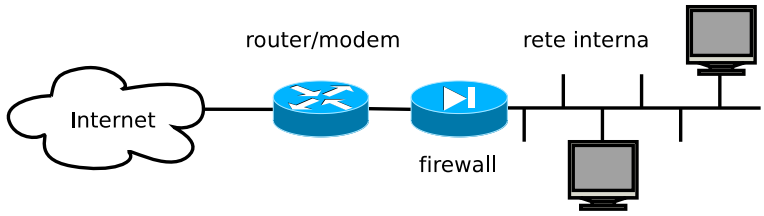


Figura 1.1: Disposizione elementare di un firewall posto a protezione di una rete con un singolo punto di accesso all'esterno che non necessita di accesso da remoto.

entranti. In questa configurazione di rete non c'è molto da dire, in quanto le direzioni dei flussi dei dati sono univoche, e la topologia della rete è elementare; la gestione della sicurezza pertanto è molto semplice e richiede solo di non consentire dall'esterno nessun accesso all'interno.

Una situazione del genere però è anche di scarsa utilità, se non nel caso di un ufficio che condivide una connessione ad internet, proprio in quanto non consente l'accesso dall'esterno a nessuna macchina interna. Il solo introdurre questo requisito rende la configurazione di fig. 1.1 inadeguata allo scopo. Infatti se anche si usasse il firewall per consentire l'accesso ad una ed una sola delle macchine all'interno della rete, questo già porterebbe ad una forte diminuzione della sicurezza della rete, dovendosi a questo punto tenere in conto la possibilità di un attacco a partire dalla macchina raggiungibile dall'esterno, attacco prima impossibile, adesso subordinato al fatto che il servizio raggiungibile dall'esterno non sia vulnerabile.

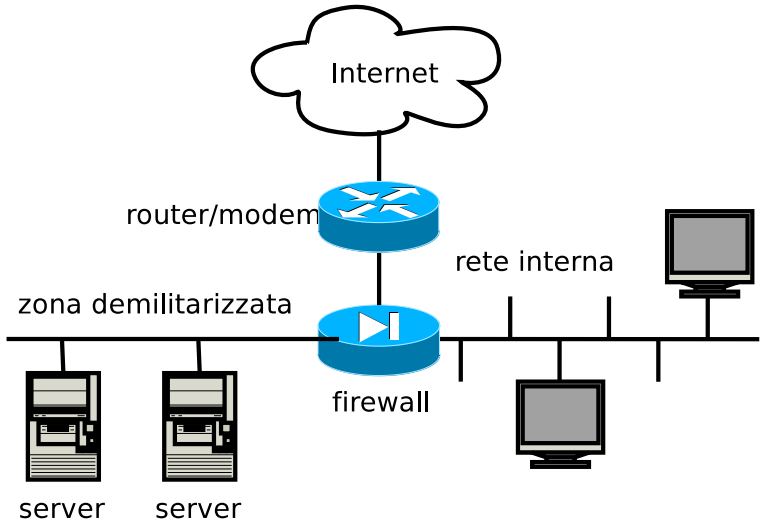


Figura 1.2: Disposizione standard di un firewall messo a protezione di una singola rete con un solo punto di accesso all'esterno e macchine accessibili da remoto.

Per questo, quando si devono fornire servizi raggiungibili dall'esterno che possono introdurre dei rischi di compromissione (i servizi web sono l'esempio più comune), si deve usare una disposizione della rete diversa, introducendo quella che viene chiamata in gergo una *zona demi-*

litarizzata, o più comunemente DMZ (sigla dell'inglese *De-Militarized Zone*), secondo lo schema di fig. 1.2.

In questo caso i servizi che devono essere raggiungibili dall'esterno devono essere messi su un tratto di rete a parte, separato dalla rete interna. Le connessioni dalla rete interna devono poter raggiungere sia internet che la zona smilitarizzata, mentre questa deve poter solo raggiungere internet, ma non la rete interna. In questo modo anche in caso di vulnerabilità di un servizio sulla DMZ la rete interna resta protetta.

Nel caso specifico si è considerato che dietro al firewall ci sia una unica rete locale, ma non è detto che debba essere così; si possono avere situazioni più complesse, con più reti separate. Di nuovo in questo caso occorre prima capire quali sono le interrelazioni fra le varie reti, e quali eventuali altri livelli di protezione si vogliono raggiungere.

1.2 Il *netfilter* di Linux

In questa sezione prenderemo in esame direttamente il funzionamento del cosiddetto *netfilter* di Linux, il sistema di manipolazione e filtraggio dei pacchetti implementato dal kernel, che consente, fra le altre cose, di costruire router e firewall estremamente potenti e flessibili.

1.2.1 La struttura del *netfilter*

Il *netfilter* è quella parte del kernel che si occupa del filtraggio e della manipolazione dei pacchetti della rete. Per poterla utilizzare deve essere stato abilitato il relativo supporto nel kernel, cosa che avviene normalmente per quelli installati da tutte le principali distribuzioni.

In fig. 1.3 è riportato lo schema di funzionamento del cuore del *netfilter* di Linux. Esso consiste in una infrastruttura che consente di inserire all'interno della parte del kernel che gestisce i protocolli di rete una serie di *punti di aggancio* (detti *hooks*, ed identificati in figura dagli ovali) in cui i vari pacchetti che arrivano sullo stack dei protocolli di rete possono essere esaminati e manipolati.

Vedremo più avanti che ciascun punto di aggancio corrisponde ad una delle cosiddette *catene predefinite* del comando `iptables`; a questo livello però il concetto di catena non esiste ancora, si tratta solo della possibilità di far eseguire, attraverso le funzioni che grazie al *netfilter* si possono agganciare a ciascun punto, una serie di elaborazioni sui pacchetti che stanno transitando, potendo decidere la sorte di questi ultimi.

L'infrastruttura del *netfilter* infatti consente alle varie funzioni poste nei punti di aggancio di decidere il destino dei pacchetti, facendoli procedere all'interno del *netfilter*, oppure scartandoli, rinviandoli sul punto di accesso (così che possano essere analizzati da altre funzioni), trattenendoli per ulteriori elaborazioni o inviandoli su una coda da cui possono essere letti da processi in user space.

Questo rende il funzionamento del *netfilter* completamente modulare, in quanto con gli opportuni moduli diventa possibile inserire nel kernel l'infrastruttura per tutte le funzionalità avanzate come il *connection tracking* e quelle relative a qualunque criterio di selezione e filtraggio dei pacchetti, fino al limite, non molto pratico, di potersi scrivere il proprio firewall direttamente in C, come un modulo del kernel.

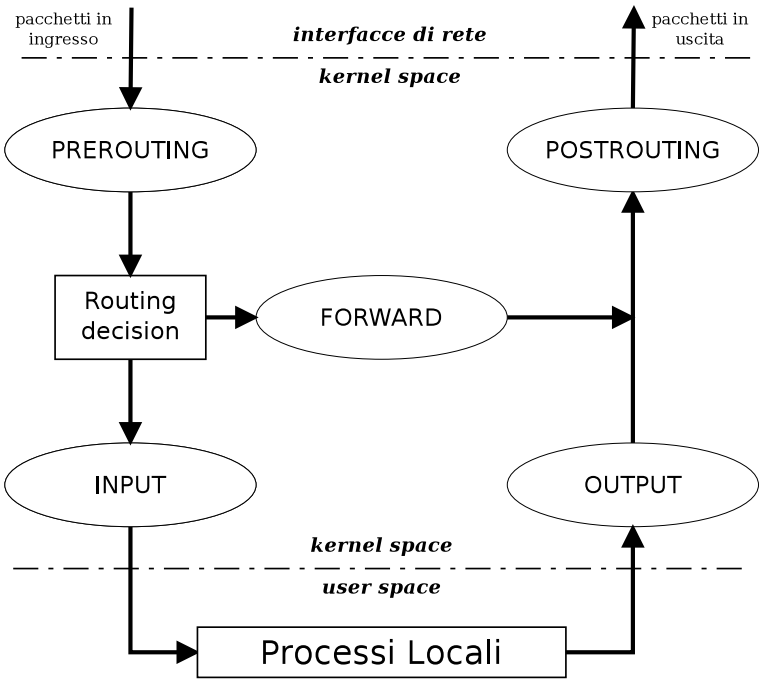


Figura 1.3: Lo schema del *netfilter* di Linux.

Vediamo allora in maggior dettaglio il funzionamento del meccanismo illustrato in fig. 1.3; i pacchetti in arrivo da una interfaccia di rete, così come emergono dal relativo dispositivo (fisico o virtuale che sia), arrivano, dopo aver passato dei controlli elementari di integrità (pacchetti troncati, corrotti, o con il valore delle *checksum* di IP non corrispondente vengono scartati a questo livello, prima di entrare nel *netfilter*) al primo punto di aggancio, indicato nello schema di fig. 1.3 con l'ovale `PREROUTING`.

Una volta attraversato il `PREROUTING` i pacchetti vengono passati al codice di gestione dell'instradamento (indicato in fig. 1.3 dal riquadro marcato *routing decision*), che deve decidere se il pacchetto è destinato ad un indirizzo locale della macchina o deve essere reinviato verso un'altra rete, passando per un'altra interfaccia. Per questo è sempre sul `PREROUTING` che vengono effettuate alcune di quelle operazioni di *traslazione degli indirizzi* (il cosiddetto NAT, acromino di *Network Address Translation*) che sono una delle caratteristiche più interessanti dei firewall contemporanei.

In particolare è qui, prima che i pacchetti entrino nel meccanismo della *routing decision* che ne stabilisce la destinazione finale, che si devono eseguire le manipolazioni sui pacchetti che permettono di cambiarla con il cosiddetto DNAT (acronimo di *Destination Network Address Translation*). Si tratta in questo caso di modificare l'indirizzo di destinazione dei pacchetti in arrivo, in modo da cambiare, in maniera trasparente per chi lo ha iniziato, la destinazione di un certo flusso di dati.[3]

[3]questo comporta, grazie all'uso del *connection tracking* che vedremo in sez. 1.2.3, anche la corrispondente

Se il pacchetto è destinato ad un indirizzo locale esso proseguirà il suo percorso nel *netfilter* attraversando un secondo punto di aggancio, indicato in fig. 1.3 con l'ovale `INPUT`, prima di poter essere recapitato al processo a cui è diretto, se esso esiste, o generare un opportuno messaggio di errore, qualora questo sia previsto.[4] È sull'`INPUT` che in genere si esegue il filtraggio dei pacchetti destinati alla nostra macchina.

Se invece il pacchetto deve essere reinviato ad un indirizzo su una rete accessibile attraverso un'altra interfaccia dovrà prima attraversare un secondo punto di aggancio, indicato in fig. 1.3 con l'ovale `FORWARD`, ed infine l'ultimo punto, indicato con l'ovale `POSTROUTING`, prima di arrivare sull'interfaccia di uscita.

La ragione di questo doppio passaggio è che il filtraggio dei pacchetti che attraversano soltanto la nostra macchina per un'altra destinazione viene effettuato nel `FORWARD`, dato che attraverso il `POSTROUTING` oltre ai pacchetti in transito dalla nostra macchina, possono passare anche i pacchetti generati direttamente su di essa.

Infatti, tutti i pacchetti generati localmente, compresi anche i pacchetti generati in risposta a quelli ricevuti in ingresso attraverso l'`INPUT`, devono passare per un altro punto di aggancio, indicato in fig. 1.3 con l'ovale `OUTPUT`, prima di raggiungere anch'essi il `POSTROUTING`. Pertanto è su `OUTPUT` che è possibile filtrare quanto viene inviato verso l'esterno a partire da processi locali o direttamente dal kernel (ad esempi messaggi di errore) in risposta a pacchetti arrivati in ingresso.

Infine è sul `POSTROUTING` che si possono compiere altre operazioni di NAT, ed in particolare le manipolazioni sugli indirizzi sorgente dei pacchetti per il cosiddetto SNAT (acronimo di *Source Network Address Translation*) in cui si modifica l'indirizzo sorgente dei pacchetti in uscita in modo da poter cambiare, in maniera trasparente per chi lo riceve, l'origine di un certo flusso di dati.[5] In questo modo ad esempio si effettua il cosiddetto *masquerading*, in cui si fa figurare un router/firewall come origine delle connessioni, anche se queste originano da computer posti in una rete interna.

1.2.2 Il sistema di *IP Tables*

Lo schema illustrato in sez. 1.2.1 riguarda la struttura di basso livello del kernel, su di esso però è stato costruita tutta una infrastruttura di più alto livello per la gestione dei pacchetti chiamata *IP Tables*, che è quella che consente di far eseguire delle operazioni sui pacchetti nei cinque punti di aggancio del *netfilter* illustrati in precedenza, senza doversi scrivere un modulo del kernel.

Questa infrastruttura è basata sulla presenza varie "*tabelle*", ciascuna delle quali è dedicata ad ospitare le operazioni relative ad un compito specifico. All'interno delle tabelle sono definite delle *catene predefinite*, che corrispondono agli effettivi punti di aggancio del *netfilter* di fig. 1.3 su cui arrivano i pacchetti, e che hanno lo stesso nome per tutte le tabelle.

Le tabelle non sono altro che dei contenitori per le "*catene*" che a loro volta sono semplicemente dei contenitori per le regole di filtraggio. Una regola è costituita da un criterio di

modifica degli indirizzi sorgente delle relative risposte, che però viene effettuata automaticamente sui pacchetti in uscita senza dover scrivere ulteriori regole.

[4] si tenga presente che un tale messaggio, qualora emesso, seguirà lo stesso percorso di tutti i pacchetti che escono dalla nostra macchina; non ha alcun senso pensare che possa tornare indietro secondo il percorso fatto dal pacchetto di andata.

[5] ed in questo caso grazie al *connection tracking*, verranno modificati automaticamente gli indirizzi di destinazione dei pacchetti in ingresso ottenuti come risposta.

selezione (detto *match*) che permette di scegliere i pacchetti in base alle loro caratteristiche, e da un criterio di destinazione (detto *target*) che permette di stabilire il destino dei pacchetti che soddisfano il criterio di selezione.

I pacchetti che attraversano una catena vengono sottoposti in sequenza a ciascuna delle regole che essa contiene, e se corrispondono al criterio di selezione viene applicato loro il criterio di destinazione scelta. Per semplificare la gestione il sistema permette di creare, oltre a quelle predefinite, su cui comunque transitano i pacchetti, delle nuove catene, da identificare con un nome fornito dall'utente, su cui inserire un qualunque insieme di regole che si vogliono raggruppare.

Una nuova catena così definita può costituire la *destinazione* (il *target*) di una regola e tutti i pacchetti che corrispondono al criterio di selezione di detta regola verranno inviati sulla nuova catena e con questo sottoposti alla serie di regole ivi definite. In questo modo una catena definita dall'utente viene a costituire una specie di "*subroutine*" che permette di trattare gruppi di pacchetti con lo stesso insieme di regole in maniera compatta.

Ciascuna delle tabelle su cui sono presenti le varie catene viene realizzata attraverso un opportuno modulo del kernel. Ognuna di esse prevede la presenza di una o più catene predefinite, a seconda dello scopo della tabella. Al momento esistono solo quattro tabelle:

filter è la tabella principale, quella utilizzata, come indica il nome, per le regole per il filtraggio dei pacchetti. Benché in teoria sia possibile filtrare i pacchetti anche sulle altre tabelle, essa costituisce il luogo naturale su cui operare per la creazione di firewall. Contiene tre catene predefinite: `INPUT`, su cui filtrare i pacchetti in ingresso sulla macchina stessa; `FORWARD`, su cui filtrare i pacchetti che vengono instradati attraverso la macchina da una interfaccia ad un'altra, e `OUTPUT` su cui filtrare i pacchetti generati localmente che escono verso l'esterno.

nat questa tabella viene utilizzata, come suggerisce il nome, per inserire le regole relativo al *Network Address Translation* (sia SNAT che DNAT) che permettono di manipolare gli indirizzi. Ha la caratteristica che solo il primo pacchetto di una connessione raggiunge questa tabella, mentre a tutti i restanti pacchetti nella stessa connessione verrà automaticamente applicata la stessa regola presa per il primo pacchetto. L'uso di questa tabella presuppone quindi l'impiego del sistema del *conntrack* che vedremo in sez. 1.2.3. La tabella contiene tre catene: `PREROUTING`, per modificare gli indirizzi dei pacchetti che arrivano sulla macchina (in genere usata per alterarne la destinazione); `OUTPUT`, per modificare i pacchetti generati localmente prima che essi arrivano all'instradamento finale; `POSTROUTING`, per modificare i pacchetti immediatamente prima che questi escano dall'interfaccia (in genere usata per alterarne l'indirizzo sorgente).

mangle questa tabella viene utilizzata, come suggerisce il nome, per manipolazioni speciali dei pacchetti; è assolutamente sconsigliato usarla per filtrare i pacchetti o alterarne gli indirizzi. Fino al kernel 2.4.17 essa conteneva solo due catene: `PREROUTING`, per modificare i pacchetti prima dell'instradamento; `OUTPUT`, per modificare i pacchetti generati localmente prima dell'instradamento. A partire dal kernel 2.4.18 sono state inserite anche le tre altre catene: `INPUT`, su cui modificare i pacchetti in ingresso sulla macchina locale; `FORWARD`, su cui modificare i pacchetti che vengono instradati

attraverso la macchina da una interfaccia ad un altra; `POSTROUTING`, per modificare i pacchetti immediatamente prima che questi escano dall'interfaccia.

`raw` una tabella speciale usata per gestire le eccezioni alle regole del sistema del *connection tracking* (che vedremo in sez. 1.2.3), principalmente per selezionare i pacchetti per cui disabilitarlo con il target speciale `NOTRACK`. E' posta al livello più alto e viene utilizzata prima di tutte le altre, e prima che vengano eseguite le operazioni del *connection tracking* a cui tutte le altre sono invece sottoposte. Contiene solo le catene `PREROUTING` ed `OUTPUT`.

In sez. 1.2.1 abbiamo visto come i pacchetti attraversano il *netfilter* all'interno del kernel, ed in precedenza abbiamo anche accennato a come le varie catene predefinite delle tabelle di *IP Tables* corrispondano in sostanza ai cinque punti di aggancio all'interno del *netfilter*; la manipolazione dei pacchetti però viene eseguita attraverso le regole inserite nelle omonime catene presenti nelle varie tabelle.

Per questo con `iptables` il diagramma di fig. 1.3 deve essere rivisto per illustrare l'ordine in cui i pacchetti attraversano le catene predefinite all'interno delle varie tabelle, dato che da quest'ordine dipende il risultato finale dell'azione delle varie regole che si sono inserite. In fig. 1.4 si è allora riportato il nuovo schema che illustra l'ordine di attraversamento dei pacchetti delle varie catene nelle varie tabelle.

Si noti come la tabella `raw` abbia un ruolo speciale, rimanendo esterna al sistema *connection tracking* che invece è attivo per tutte le altre. Siccome l'uso della tabella attiene ai pacchetti che *entrano* nel *netfilter* essa prevede solo le catene di `PREROUTING` ed `OUTPUT` che sono i due soli punti di ingresso dello stesso. In entrambi i casi le regole della tabella saranno processate per prime e prima dell'attivazione del *connection tracking*.

Se un pacchetto è destinato ad un processo locale questo passerà prima per le catene di `PREROUTING` prima nella tabella di `mangle` e poi nella tabella di `nat`, poi passerà per il meccanismo di decisione dell'instradamento che invierà il pacchetto alle catene di `INPUT`, prima nella tabella di `mangle` e poi in quella di `filter`, per poi essere ricevuto dal processo locale cui è destinato.

Se invece il pacchetto è in transito attraverso la nostra macchina per essere reinviato su un'altra interfaccia di nuovo passerà per le catene di `PREROUTING` prima nella tabella di `mangle` e poi nella tabella di `nat`, ma dopo la decisione sull'instradamento proseguirà sulle catene di `FORWARD`, prima della tabella di `mangle` e poi di quella di `filter`, infine proseguirà verso l'interfaccia di uscita passando per le catene di `POSTROUTING`, prima nella tabella di `mangle` e poi in quella di `nat`.

Infine se un pacchetto è generato da un processo locale per essere inviato all'esterno prima passerà per le catene di `OUTPUT`, a partire dalla tabella di `mangle`, per proseguire in quella di `nat` ed infine in quella di `filter`, da qui poi passerà alle catene di `POSTROUTING` nello stesso ordine usato dai pacchetti in transito.

1.2.3 Il meccanismo del *connection tracking*

Una delle principali caratteristiche del *netfilter* di Linux è stata quella di aver introdotto una infrastruttura complessa, ma estremamente flessibile, chiamata *connection tracking* (in breve *conntrack*), che permette di analizzare il flusso dei pacchetti, ricostruire lo stato delle connessioni

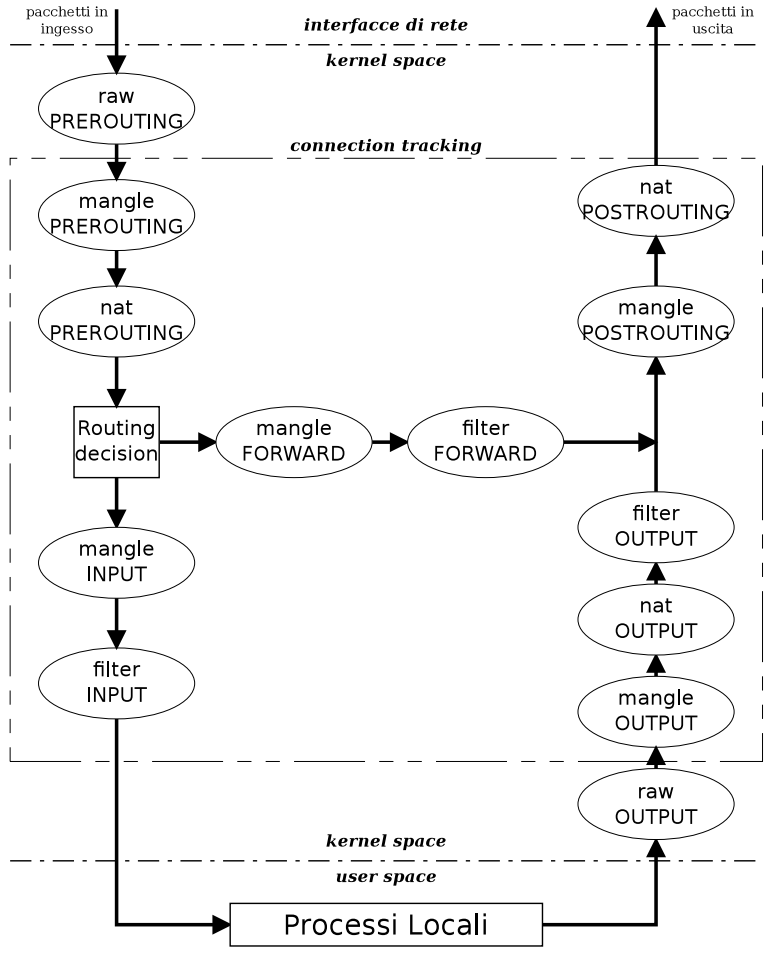


Figura 1.4: Lo schema del flussi dei pacchetti attraverso le tabelle e le catene predefinite di `iptables`.

presenti, ricondurre ad esse e classificare i vari pacchetti. É grazie a questo meccanismo che è possibile la realizzazione del filtro degli stati illustrato in sez. 1.2.4.

L'infrastruttura del *connection tracking* va ben oltre l'uso fattone dal filtro degli stati per realizzare un firewall *stateful*; essa infatti è anche la base su cui è costruito il funzionamento delle capacità di NAT che permettono al *netfilter* di modificare al volo gli indirizzi dei pacchetti, e per poterli redirigere opportunamente, in modo da gestire compiti come il *masquerading*[6] o il *transparent proxing*.[7]

Lo scopo principale del motore del *conntrack* è quello di analizzare il traffico per identificare

[6] si chiama così l'uso dello SNAT per *mascherare* una (o più) reti dietro un singolo indirizzo IP (quello usato in genere dal firewall per uscire su internet).

[7] si chiama così l'uso del DNAT per redirigere, in maniera trasparente per i client, il traffico destinato a certi servizi (il caso classico è il web) su una macchina che faccia da *proxy*.

ogni connessione presente, determinarne lo stato, e poi classificare i pacchetti associandoli a ciascuna di esse. Dato che il concetto di *stato* e di *connessione* è definito nativamente solo per alcuni protocolli (in sostanza solo per il TCP), il meccanismo è stato esteso per permettere di identificare e classificare ogni flusso di dati anche per gli altri protocolli che non supportano le connessioni, come UDP e ICMP. Ad esempio come vedremo più avanti si potrà definire una forma di connessione anche per UDP, identificando univocamente un flusso di pacchetti in base ai relativi indirizzi e porte di origine e destinazione.

L'infrastruttura del *conntrack* è stata introdotta nei kernel della serie 2.4; nei kernel precedenti non esisteva nessun meccanismo che potesse eseguire questo compito e pertanto non era possibile creare dei firewall stateful. Inizialmente disponibile solo per il protocollo IPv4 è stato esteso anche ad IPv6. L'uso del sistema ha però portato anche ad un cambiamento delle regole che gestiscono la frammentazione dei pacchetti IP.

La frammentazione è una caratteristica di IPv4 che prevede che i pacchetti che debbano essere inoltrati su una linea la cui MTU[8] eccede le dimensioni del pacchetto vengano "*frammentati*" dal router che li riceve per poter essere trasmessi sulla linea stessa. Questo significa che un singolo pacchetto IP verrà spezzato in più parti, che poi verranno riassemblate solo alla destinazione finale, con la ricezione di tutti i pezzi. Nei kernel precedenti la serie 2.4.x c'era la possibilità di attivare o meno la deframmentazione dei pacchetti prima di procedere al loro filtraggio, attraverso l'opzione `ip_always_defrag`. Dato che il *connection tracking* non può funzionare con i pacchetti frammentati, questa opzione non esiste più, il codice per la deframmentazione è stato incluso nel *conntrack* ed i pacchetti vengono sempre riassemblati prima di tracciarne lo stato.

Tutte le operazioni relative al *connection tracking* vengono eseguite nella catena di `PREROUTING` eccetto per i pacchetti che vengono generati localmente, che invece sono analizzati nella catena di `OUTPUT`. Questo significa che tutte le operazioni per il calcolo degli stati e la classificazione dei pacchetti vengono eseguite non appena questi arrivano alla catena di `PREROUTING`. La sola eccezione è per i pacchetti generati localmente che creano una nuova connessione, che vengono identificati su `OUTPUT`, ma non appena al pacchetto inviato per la creazione di una connessione arriva una risposta la successiva classificazione dei pacchetti avviene comunque sulla catena di `PREROUTING`. Come accennato l'unico modo di impostare delle regole con `iptables` prima che il *connection tracking* entri in azione è quello di usare la speciale tabella di `raw`.

Il *conntrack* mantiene nel kernel una tabella con tutte le connessioni presenti ed il relativo stato; la dimensione della tabella, e pertanto il limite massimo al numero di connessioni contemporanee che possono esistere, è un parametro configurabile che può essere impostato usando l'interfaccia del filesystem `proc` attraverso il file `/proc/sys/net/nf_conntrack_max`. Il valore di default è calcolato automaticamente sulla base della memoria disponibile, e nel caso di una macchina con 8Gb di RAM otteniamo:

```
# cat /proc/sys/net/nf_conntrack_max
65536
```

detto valore può essere comunque modificato al volo scrivendo un diverso valore nel file.

Si può inoltre accedere direttamente al contenuto della tabella delle connessioni attraverso il filesystem `proc`, usando il file `/proc/net/ip_conntrack` se interessa solo IPv4 oppure il file `/proc/net/nf_conntrack` per tutti i protocolli del livello di rete. Nel seguito esamineremo solo

[8] la *Maximum Transfer Unit* o MTU è la dimensione massima di un pacchetto di dati che può essere trasferita su un segmento fisico di una rete.

il caso di IPv4, che resta comunque l'ambito di maggiore applicazione del sistema, e faremo riferimento al primo dei due file, che manca nei dati del campo della informazione iniziale relativa al protocollo di rete cui fa riferimento la voce (in genere `ipv4` o `ipv6`).

Esaminando `/proc/net/ip_conntrack` si otterranno le informazioni mantenute dal kernel relative a ciascuna connessione; ogni riga corrisponde ad una connessione, un esempio di voce contenuta nel file è:

```
tcp      6 119 SYN_SENT src=192.168.1.1 dst=192.168.1.141 sport=35552 \
      dport=22 [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 sport=22 \
      dport=35552 use=1
```

Il primo campo è sempre quello del protocollo, espresso per nome, e ripetuto per numero (in esadecimale, anche se nell'esempio non lo si nota) nel secondo campo. Il terzo campo indica il tempo di vita restante per la voce nella tabella, nel caso 119 secondi. Il campo viene decrementato progressivamente fintanto che non si riceve traffico sulla connessione, nel qual caso viene riportato al valore predefinito (che varia a seconda del protocollo).

I primi tre campi sono sempre questi, i campi successivi dipendono dal protocollo, nel caso specifico il quarto campo, avendo a che fare con il TCP, indica lo stato della connessione, che è `SYN_SENT`: si è cioè inviato un segmento SYN in una direzione. Seguono i campi con indirizzo sorgente e destinazione, e porta sorgente e destinazione. Segue poi una parola chiave `[UNREPLIED]` che ci informa che non si è ricevuta risposta, e gli indirizzi sorgente e destinazione, e le porte sorgente e destinazione per i pacchetti appartenenti al traffico di ritorno di questa connessione; si noti infatti come rispetto ai precedenti i numeri siano gli stessi ma sorgente e destinazione risultino invertiti.

Vediamo allora il funzionamento del *conntrack* per i tre principali protocolli, cominciando con il più complesso, il TCP. In questo caso infatti il protocollo già prevede l'esistenza degli stati, e come vedremo questo comporterà un certo livello di confusione, in quanto a volte lo stato del protocollo non coincide con quello del *connection tracking*.

La caratteristica principale del TCP è che esso prevede la creazione di una connessione tramite l'invio di tre pacchetti separati, con un procedimento chiamato *three way handshake*, illustrato in fig. 1.5, dove si è riportato lo scambio di pacchetti ed i relativi cambiamenti di stato del protocollo su client e server.

Supponiamo allora di essere sul client; dal punto di vista del *conntrack* una volta che si invia un pacchetto SYN questo passerà per la catena di `OUTPUT` e verrà riconosciuto come un pacchetto che inizia una nuova connessione, creando una voce nella tabella delle connessioni simile a quella mostrata in precedenza, con lo stato `SYN_SENT`[9] ed una connessione marcata come `UNREPLIED`.

Il primo passaggio di stato si ha una volta che si riceve la risposta dal server con il segmento SYN+ACK ricevuto il quale la connessione si porta immediatamente nello stato `ESTABLISHED`, ed avremo una voce del tipo:

```
tcp      6 431996 ESTABLISHED src=192.168.1.1 dst=192.168.1.141 sport=38469 \
      dport=22 src=192.168.1.141 dst=192.168.1.1 sport=22 \
      dport=38469 [ASSURED] use=1
```

[9] si può ottenere questa situazione bloccando sul server la ricezione dei segmenti SYN, in questo modo il client non riceva da questo nessuna risposta, così che lo stato nella tabella delle connessioni resti bloccato.

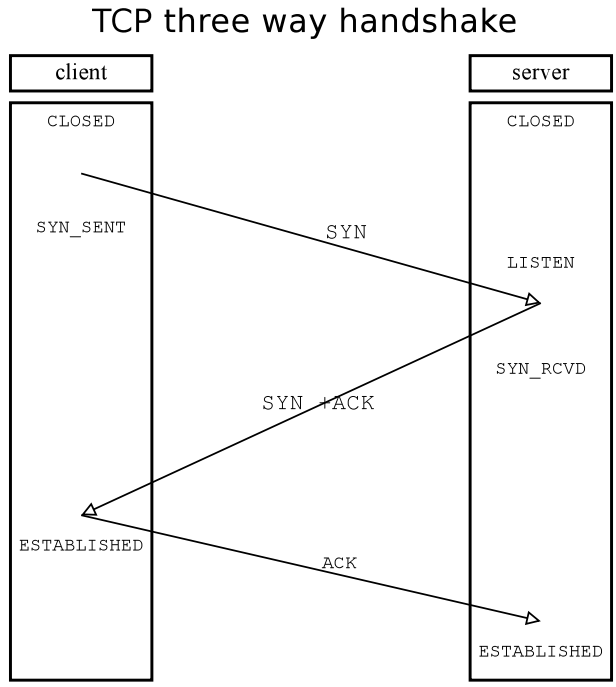


Figura 1.5: Lo schema del *three way handshake* del TCP.

dove il flag che indica la condizione di `[UNREPLIED]` non c'è più ed invece viene attivato quello che indica la condizione di `[ASSURED]`; questa ci dice che è stato osservato del traffico in entrambe le direzioni, e che la voce nella tabella delle connessione non deve essere cancellata anche se questa si riempie, cosa che invece avviene per le voci che non hanno tale flag attivo.

Si tenga presente che questo accade anche se blocchiamo a questo punto il *three way handshake*,[10] interrompendo la creazione della connessione TCP; questa è la sequenza normale per gli stati del TCP, che è anche illustrata in fig. 1.5, e la si può verificare usando `netstat` sul server, dove la connessione sarà riportata essere rimasta nello stato `SYN_RECV`.

La prima differenza fra lo stato del protocollo e quello della connessione all'interno del sistema del *connection tracking* emerge a questo punto: lo stato del protocollo infatti è modificato alla ricezione dei pacchetti (cioè sulla catena di `INPUT`), mentre lo stato della connessione è, come abbiamo già detto, modificato sulla catena di `PREROUTING`. Se si blocca con il firewall la ricezione del pacchetto SYN+ACK, allora il risultato può essere diverso a seconda di come si esegue detto blocco. Se infatti si blocca il pacchetto sulla catena di `INPUT` questo passa comunque per la catena di `PREROUTING`, ed esattamente come prima la connessione viene riconosciuta come `ESTABLISHED`.

Se invece si blocca il pacchetto SYN+ACK direttamente sulla catena di `PREROUTING`[11] la

[10] ad esempio impedendo con una apposita regola nella catena di `OUTPUT` che il segmento ACK di conclusione del procedimento venga inviato al server.

[11] per farlo occorre usare la tabella di `mangle`, bloccando il pacchetto su di essa, dato che la tabella di `nat` opera solo sul primo pacchetto di una connessione.

ricezione dell'ACK non viene riconosciuta e la conclusione della connessione non viene stabilita, viene però rilevato l'arrivo di un segmento SYN, e la connessione nella tabella degli stati viene portata nello stato `SYN_RECV`, mentre la condizione di `UNREPLIED` sparisce, avremo cioè una voce del tipo:

```
tcp      6 44 SYN_RECV src=192.168.1.1 dst=192.168.1.141 sport=35552 \
      dport=22 src=192.168.1.141 dst=192.168.1.1 sport=22 dport=35552 use=1
```

Questa condizione si incontra anche quando si riceve un pacchetto SYN di una connessione diretta verso di noi, in tal caso la voce viene creata alla ricezione di un pacchetto SYN,[12] con una voce del tipo:

```
tcp      6 55 SYN_RECV src=192.168.1.141 dst=192.168.1.1 sport=33616 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 dport=33616 use=1
```

e di nuovo, non appena si emette il segmento di risposta con un SYN+ACK, la connessione andrà in stato `ESTABLISHED`, con una voce del tipo:

```
tcp      6 431998 ESTABLISHED src=192.168.1.141 dst=192.168.1.1 sport=33617 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 \
      dport=33617 [ASSURED] use=1
```

e ci resterà anche se non si conclude il *three way handshake*,[13] questo perché nel caso del protocollo TCP lo stabilirsi di una connessione è identificato dal *conntrack* semplicemente dal fatto che c'è uno scambio di pacchetti TCP nelle due direzioni.

La chiusura di una connessione TCP avviene con lo scambio di 4 pacchetti, come illustrato in fig. 1.6, e non viene mai completamente chiusa fino a che questo non viene completato. Una volta che questa è completata la connessione entra nello stato `TIME_WAIT`, con una voce del tipo:

```
tcp      6 9 TIME_WAIT src=192.168.1.141 dst=192.168.1.1 sport=33614 \
      dport=22 src=192.168.1.1 dst=192.168.1.141 sport=22 \
      dport=33614 [ASSURED] use=1
```

che ha una durata di default di due minuti, così che tutti gli eventuali ulteriori pacchetti residui della connessione che possono arrivare saranno classificati all'interno della stessa ed essere sottoposti alle stesse regole.

Lo schema di chiusura mostrato in fig. 1.6 non è l'unico possibile, una connessione infatti può anche essere terminata da un pacchetto RST,[14] che la porta in stato di `CLOSE`, che resta attivo per 10 secondi. Si tenga presente che in risposta ad un pacchetto RST non deve essere dato nessun ricevuto e che questi pacchetti sono generati direttamente dal kernel ed utilizzati per chiudere immediatamente una connessione.

Gli stati possibili per una connessione TCP sono tutti quelli definiti per il protocollo (anche se la corrispondenza non è proprio immediata, come abbiamo appena visto). Per ciascuno

[12] si può verificare il comportamento bloccando la risposta del SYN+ACK sulla catena di `OUTPUT`.

[13] cosa che si può ottenere ad esempio bloccando con il firewall la ricezione dell'ultimo ACK di risposta.

[14] il protocollo TCP prevede infatti che in risposta a pacchetti inviati ad una porta su cui non è in ascolto nessun servizio, o in generale non ricollegabili ad una connessione debba essere risposto con l'emissione di un pacchetto RST, la ricezione del quale deve produrre la terminazione immediata di ogni eventuale connessione.

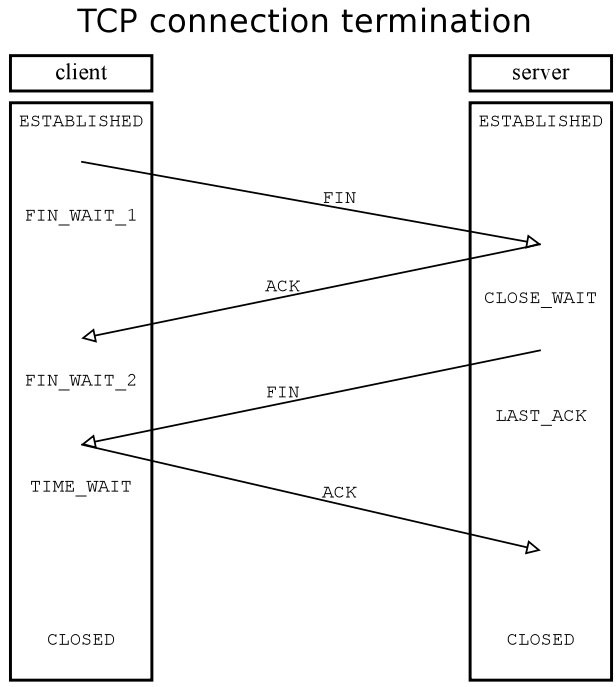


Figura 1.6: Lo schema della conclusione di una connessione TCP.

di questi il *conntrack* definisce un opportuno valore di timeout scaduto il quale la voce viene rimossa dalla tabella delle connessioni. I valori di default per il timeout (che vengono impostati quando la connessione si porta in quello stato, e reimpostati quando si ricevono nuovi pacchetti sulla connessione) sono definiti nel file `net/netfilter/nf_conntrack_proto_tcp.c` nei sorgenti del kernel e riportati in tab. 1.1.[15]

Stato	**Valore**
SYN_SENT	2 minuti.
SYN_RECV	60 secondi.
ESTABLISHED	5 giorni.
FIN_WAIT	2 minuti.
CLOSE_WAIT	60 secondi.
LAST_ACK	30 secondi.
TIME_WAIT	2 minuti.
CLOSE	10 secondi.
SYN_SENT2	2 minuti.

Tabella 1.1: Valori dei tempi di timeout delle connessioni TCP.

Al contrario di TCP, UDP è un protocollo che non supporta le connessioni, il protocollo si limita ad inviare i pacchetti, questi possono arrivare in qualunque ordine, essere perduti o

[15] la tabella è aggiornata ai valori usati dal kernel 3.1.

duplicati, e non ci sono procedimenti per definire l'inizio o la conclusione di una comunicazione. Nonostante questo sia possibile è comunque possibile cercare di classificare il traffico ed inserirlo in una logica di connessioni, anche se in questo caso (come sarà per ICMP) la classificazione non può che essere basata su una euristica dello scambio dei pacchetti.

L'algoritmo di classificazione per UDP si basa sul riconoscimento del traffico dalla stessa coppia di indirizzi IP e porte (destinazione e sorgente) sui due capi della comunicazione. Si considerano cioè parte di una connessione tutti i pacchetti UDP che partono da un nostro IP e hanno una data porta sorgente diretti verso un altro IP ed una data porta di destinazione, e quelli che provengono da detto IP con sorgente la porta usata da noi come destinazione e come destinazione la nostra porta sorgente. Si fa l'assunzione cioè che pacchetti con queste caratteristiche siano emessi e ricevuti in corrispondenza allo scambio di dati fra i due capi di uno stesso socket UDP.

Si tenga presente che questo criterio non assicura che una connessione così identificata comprenda solo i pacchetti appartenenti ad un unico flusso di dati, è infatti possibile, anche se non molto probabile, dato che in genere le porte non vengono immediatamente riusate, una situazione in cui un client si collega ad un nostro server UDP usando una certa porta, conclude le sue operazioni, e poi un client diverso ritenta la stessa connessione usando la stessa porta. In un caso come questo i pacchetti saranno classificati, se la voce nella tabella del *conntrack* non è ancora scaduta, nella stessa connessione, anche se appartengono a due flussi di dati completamente distinti.

Questo meccanismo di classificazione comporta che se si invia un pacchetto UDP che non corrisponde a nessuna voce già presente nella tabella delle connessioni, questo verrà considerato come un pacchetto iniziale e creerà una nuova voce,[16] nella forma:

```
udp      17 28 src=192.168.1.1 dst=192.168.1.141 sport=32769 \
      dport=13 [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 sport=13 \
      dport=32769 use=1
```

Si noti che la voce è analoga a quelle relative alle connessioni TCP, con il nuovo nome (e relativo numero) del protocollo nei primi due campi, ed il tempo di vita restante alla voce nel terzo. Il resto della voce è analogo a quelle usate per il TCP, però non esiste più nel quarto campo un valore dello stato e vengono riportati di seguito direttamente indirizzo sorgente e destinazione e porta sorgente e destinazione. Segue, non avendo ricevuto nessun altro pacchetto il flag della condizione `[UNREPLIED]` e indirizzi e porte (ovviamente invertite) dei pacchetti che si aspettano come risposta.

Nel caso di UDP, quando viene creata con l'invio di un nuovo pacchetto, a ciascuna voce viene dato un tempo di vita di 30 secondi, dei quali, nell'esempio precedente, ne sono restati solo 28 disponibili. Non appena si riceve un pacchetto di risposta sulla coppia di IP corrispondente e porta la condizione di `[UNREPLIED]` sparirà, e la connessione si potrà considerare stabilita, la voce nella tabella delle connessioni diventerà del tipo:

```
udp      17 25 src=192.168.1.1 dst=192.168.1.141 sport=32768 \
      dport=13 src=192.168.1.141 dst=192.168.1.1 sport=13 dport=32768 use=1
```

[16] un modo per ottenere questo è bloccare con `iptables` sulla macchina di destinazione i pacchetti di risposta al collegamento via UDP.

ed il tempo di vita verrà riportato di nuovo a 30 secondi, facendo ripartire il conto alla rovescia.

Si noti però che ancora la connessione non è considerata stabile, lo diverrà non appena sarà osservata una certa quantità di traffico, al che comparirà il flag di `[ASSURED]` e la voce diverrà:

```
udp      17 177 src=192.168.1.1 dst=192.168.1.141 sport=32768 \
      dport=13 src=192.168.1.141 dst=192.168.1.1 sport=13
      dport=32768 [ASSURED] use=1
```

ed il tempo di vita viene portato ad 180 secondi (nell'esempio ne sono passati tre).

Dato che anche il protocollo ICMP non supporta connessioni, anche questo viene classificato in base ad una euristica dei pacchetti. Inoltre questo protocollo viene usato principalmente per l'invio di messaggi di controllo relativi ad errori della rete, e pertanto non è neanche classificabile in una logica di connessioni e stati. Esistono però alcuni specifici pacchetti ICMP che richiedono una risposta, ed allora di nuovo diventa possibile classificare queste risposte in una logica di connessione.

Per questo motivo il sistema del *conntrack* classifica soltanto quattro coppie di *tipi*[17] di pacchetti ICMP, *echo request* e *echo reply*, *netmask request* e *netmask reply*, *timestamp request* e *timestamp reply*, *information request* e *information reply*. In questo caso l'euristica di classificazione delle connessioni è identica a quella di UDP, dove al posto della porta viene usato il tipo di pacchetto, che deve corrispondere alla precedente richiesta. Ad esempio pacchetti di *echo reply* provenienti da un IP a cui si era inviati un *echo request* vengono considerati facenti parte di una connessione.

Così se effettuiamo un `ping` verso una macchina all'invio del pacchetto *echo request* nella tabella del *conntrack* apparirà una voce del tipo:[18]

```
icmp     1 22 src=192.168.1.1 dst=192.168.1.141 type=8 code=0 id=29957 \
    [UNREPLIED] src=192.168.1.141 dst=192.168.1.1 type=0 code=0 id=29957 use=1
```

che ha un tempo di vita di 30 secondi. Al solito la voce riporta indirizzi IP sorgente e di destinazione del pacchetto e dell'eventuale risposta attesa. In questo caso viene anche riportato sia il tipo di pacchetto ICMP che ha creato la voce (8, essendo un *echo request*) seguita poi da quello (0, essendo un *echo reply*) che ci si attende come risposta, mentre i valori del campi `code` e `id` degli stessi.

Nel caso di ICMP non si ha invece nessuna voce relativa ad una connessione stabilita, infatti la ricezione del pacchetto di risposta esaurisce tutto il traffico previsto per questo tipo di operazioni, per cui la connessione viene immediatamente chiusa, e la voce nella tabella del *conntrack* cancellata.

1.2.4 Il funzionamento del filtro degli stati

Una delle principali innovazioni portate dal *netfilter* con la serie dei kernel 2.4.x è stata quella della possibilità di usare Linux come firewall *stateful*. Come spiegato in sez. 1.1.2 un firewall *stateful* è un firewall in grado di classificare i pacchetti in un insieme di *stati*, per poi selezionarli in base a questi ultimi.

[17] i pacchetti ICMP vengono classificati in base al valore di un apposito campo nella loro intestazione, il *tipo*, per maggiori informazioni si può fare riferimento a quanto illustrato in sez. 1.4.4 e a fig. 1.8.

[18] al solito la si può ottenere bloccando i pacchetti di risposta provenienti dalla macchina che si sta "*pingando*".

Nel caso di Linux questo viene realizzato attraverso l'infrastruttura del *connection tracking* descritto in sez. 1.2.3. Benché le modalità in cui vengono classificati i pacchetti all'interno del *connection tracking* siano abbastanza complesse, il filtro degli stati di Linux riclassifica tutti i pacchetti in solo 4 stati, che si sono riportati in tab. 1.2. Nella tabella si è menzionato anche `UNTRACKED` che in realtà non è uno stato ma identifica semplicemente i pacchetti che si sono esclusi dal *connection tracking*, e quindi anche dalla classificazione degli stati, tramite il target `NOTRAK` nella catena di `raw`.

Stato	Descrizione
`NEW`	Identifica i pacchetti usati per dare origine ad una connessione.
`ESTABLISHED`	Identifica i pacchetti riconosciuti far parte o essere associati ad una connessione già stabilita.
`RELATED`	Identifica i pacchetti che danno origine ad una nuova connessione che è però correlata ad una connessione esistente.
`INVALID`	Identifica i pacchetti che non vengono associati a nessuna connessione esistente o hanno dati o intestazioni riconosciuti come non validi.
`UNTRACKED`	Identifica i pacchetti che si sono forzatamente esclusi dal *connection tracking* e con questo anche dalla classificazione negli stati tramite il target `NOTRAK` nella tabella di `raw`.

Tabella 1.2: Gli stati delle connessioni definiti nel *netfilter*.

Come per la classificazione delle connessioni anche quella degli stati viene eseguita nella catena di `PREROUTING`, con l'eccezione dei pacchetti creati localmente che vengono analizzati su `OUTPUT`. Quando un pacchetto arriva sulla catena di `PREROUTING` prima viene eseguito un eventuale DNAT e le manipolazioni previste da eventuali regole nella tabella di `mangle`, infine viene calcolato lo stato. Lo stesso avviene per i pacchetti uscenti dalla catena di `OUTPUT`.

Quando si incontra un pacchetto che non si è mai visto prima e che può essere considerato come pacchetto di inizio di una nuova connessione, questo verrà inserito nello stato `NEW`. Il caso più comune è quello di un pacchetto TCP con il flag SYN, ma può essere anche il primo pacchetto di una sessione su UDP. Si tenga presente inoltre che possono essere classificati in questo stato anche pacchetti TCP diversi da un SYN, il che può comportare problemi in certi casi, ma può essere di aiuto quando ci sono connessioni andate in timeout ma non chiuse.

Se un pacchetto corrisponde ad una voce nella tabella delle connessioni allora verrà classificato come parte di una connessione già stabilita ed assumerà lo stato di `ESTABLISHED`. Perché un pacchetto sia classificato come tale il *conntrack* deve aver rilevato del traffico in entrambe le direzioni, si arriva allo stato `ESTABLISHED` solo dopo che una connessione è passata dallo stato `NEW`, quando arriva un pacchetto interpretabile come risposta. Questo significa che anche dei pacchetti ICMP possono entrare in questo stato, qualora generati come risposta ad un pacchetto che inizia una connessione (come l'*echo reply* in risposta ad un *echo request*).

Lo stato più complesso da realizzare, anche se è molto semplice da descrivere, è `RELATED`; in breve si può dire che se il traffico può essere collegato in qualche modo ad una connessione già stabilita (ad esempio è un pacchetto ICMP relativo ad una connessione TCP, o il traffico dati di una sessione FTP) questo verrà classificato nello stato `RELATED`. Questo significa che per potere avere un pacchetto nello stato `RELATED` deve esistere già una connessione in stato `ESTABLISHED` cui questo è collegato.

La difficoltà sta tutta nel meccanismo con cui *conntrack* è in grado di stabilire in che modo un pacchetto è collegato ad una connessione stabilita. Degli esempi per questo tipo di pacchetti sono le risposte ICMP relative ad una connessione,[19] il traffico dati FTP collegato alla presenza di una connessione di controllo, o le connessioni DCC eseguite tramite IRC. In alcuni di questi casi il meccanismo per rilevare il collegamento può essere molto complesso, e richiedere anche l'analisi (parziale) del contenuto dei dati trasmessi all'interno dei vari protocolli, nel qual caso diventa necessario utilizzare degli opportuni moduli di supporto che estendono le capacità di classificazione del *conntrack*.

Questo è ad esempio quello che serve per poter utilizzare i citati protocolli per FTP e IRC, per i quali nella distribuzione standard del kernel sono previsti due appositi moduli, che quando inclusi forniscono le estensioni necessarie; altri moduli relativi ad altri protocolli (H323, SIP) sono disponibili sia come patch esterne che nei kernel di default.

Infine tutto il traffico che non rientra nelle tre categorie precedenti, che cioè non può essere identificato o a cui non è possibile assegnare uno stato, viene classificato come `INVALID`. Si tenga presente che ciò può avvenire sia perché effettivamente si tratta di pacchetti malformati, o di risposte a connessioni inesistenti, sia perché si è esaurito lo spazio nella tabella degli stati, e non è pertanto più possibile classificare il traffico ulteriore.

Come si vede il meccanismo del filtro sugli stati è piuttosto complesso, anche se semplifica notevolmente il lavoro di gestione di un firewall. Ad esempio la classificazione dei pacchetti consente di bloccare in maniera pulita tutti i pacchetti sospetti classificati nello stato `INVALID`. Inoltre con le combinazioni degli stati `NEW` e `ESTABLISHED` consente di gestire in maniera flessibile la creazione nuove "*connessioni*" anche per protocolli come UDP e ICMP che non supportano nativamente questo concetto, e per i quali sarebbe impossibile, senza la presenza del *conntrack* che traccia il flusso dei pacchetti, identificare flussi di dati collegati fra loro con regole basate esclusivamente sul contenuto delle intestazioni dei singoli protocolli.

Inoltre l'uso dello stato `RELATED` permette di ricostruire collegamenti anche fra pacchetti, connessioni e flussi di dati che, fermandosi al punto di vista dei protocolli di livello inferiore, sarebbero completamente scorrelati fra di loro, rendendo possibile filtrare in maniera semplice su protocolli complessi a livello di applicazione, che non sarebbe possibile controllare con un firewall non *stateful*.

La semplificazione della complessità dei protocolli di rete in quattro stati diversi comporta come sempre la necessità di capire bene cosa succede sotto la superficie, onde evitare effetti indesiderati non previsti. Per questo esamineremo in dettaglio il funzionamento del filtro degli stati per ciascuno dei protocolli principali su cui esso opera.

Nel caso del protocollo TCP il punto più critico sta nella definizione di cosa si intende per un pacchetto in stato `NEW`. Infatti dal punto di vista del protocollo l'unico pacchetto che può iniziare una connessione è quello che ha attivo il flag SYN e disattivi i flag ACK e RST. La classificazione del filtro degli stati in certi casi però classifica come `NEW` anche pacchetti che non hanno questa caratteristica. Questo viene fatto per consentire al traffico di una connessione già stabilita (fatta passare da un altro firewall cui si è subentrati, o autorizzata prima di aver caricato in memoria il *conntrack*) di proseguire.

Questo però non è quello che appunto si intende normalmente, per cui se si vuole che passino strettamente soltanto i pacchetti che danno origine ad una nuova connessione TCP, occorrerà

[19] al contrario di quanto veniva affermato nell'*Iptables HOWTO*, i messaggi di errore relativi ad una connessione non sono classificati allo stato `ESTABLISHED`, ma nello stato `RELATED`.

scartare i pacchetti classificati in stato NEW ma che non hanno il flag SYN opportunamente impostato.

Un altro punto da capire è che dal punto di vista della classificazione dei pacchetti vengono considerati come facenti parte dello stato ESTABLISHED tutti i pacchetti successivi al primo (come abbiamo visto in sez. 1.2.3), anche se ancora dal punto di vista del TCP la connessione non ha concluso il *three way handshake* e non può essere considerata come stabilita. Questo avviene perché deve essere possibile consentire ai pacchetti in stato NEW e ESTABLISHED di uscire dalla nostra macchina ed accettare in risposta solo i pacchetti di stato ESTABLISHED: se si considerasse la risposta del SYN+ACK come facente parte dello stato NEW questo non sarebbe possibile.

Nel caso di UDP il meccanismo è molto più semplice, e si considera in stato NEW il primo pacchetto che non corrisponde ad una voce già presente nella tabella delle connessioni. Come sempre controllo viene fatto sulla tabella di OUTPUT per le connessioni uscenti e su quella di PREROUTING per le connessioni entranti. Ogni pacchetto successivo corrispondente viene considerato essere nello stato ESTABLISHED. Dato che UDP non prevede nessun meccanismo di conclusione della connessione questa viene effettuata tramite la scadenza della relativa voce nella tabella delle connessioni, che avviene dopo 3 minuti di assenza di traffico.

Infine nel caso di ICMP si considerano come facenti parte dello stato NEW i pacchetti di richiesta (cioè *echo request* e affini), mentre si considerano come ESTABLISHED tutti quelli di risposta (cioè *echo reply* e affini). Come già accennato dato che alla ricezione del pacchetto di risposta non può esserci alcun altro traffico legale, la relativa voce nella tabella delle connessioni viene immediatamente cancellata.

I pacchetti ICMP però vengono utilizzati anche per comunicare messaggi di controllo relativi ad errori e problemi della rete. In questo caso allora gran parte di questi pacchetti vengono classificati in stato RELATED facendo riferimento ai pacchetti della connessione in risposta ai quali sono stati creati.[20] Ad esempio due dei pacchetti ICMP più comuni sono *host unreachable* e *network unreachable*, generati dal router che riceve il nostro traffico quando questo si accorge che la macchina di destinazione o un altro router che dà accesso alla rete su cui essa si trova non è più raggiungibile.

1.3 Il comando iptables

Affronteremo in questa sezione la sintassi di uso del comando iptables, e le modalità con cui questo viene impiegato per gestire il funzionamento delle regole utilizzate dal *netfilter* per eseguire i compiti di filtraggio e manipolazione dei pacchetti.

1.3.1 La sintassi generale del comando

Come illustrato in sez. 1.2.2 il *netfilter* mette a disposizione dei punti di aggancio nei quali è possibile intercettare i pacchetti, esaminarli, e decidere cosa farne. Usando l'infrastruttura di *IP Tables* questo si può fare inserendo delle *regole* all'interno delle varie *catene* presenti nelle varie *tabelle*.

[20] questo avviene solo quando i dati riportati nel payload del pacchetto coincidono; in particolare vengono inseriti in questo stato i pacchetti ICMP di tipo: *destination-unreachable*, *source-quench*, *time-exceeded*, *paremeter-problem* e *redirect*.

Il comando che permette di selezionare ed esaminare le tabelle, controllare, creare e cancellare le catene, gestirne le proprietà, definire e creare le varie regole è appunto `iptables`. Data le molteplici funzionalità che fornisce il comando è piuttosto complesso e presenta una grande quantità di opzioni, queste si possono classificare secondo una schematizzazione generale per cui l'invocazione del comando ha la forma:

```
iptables controllo [selezione] [azione]
```

in cui le opzioni sono raggruppate in tre classi generiche, quelle per il controllo di tabelle e catene, quelle per la selezione dei pacchetti, e quelle per la decisione delle azioni da compiere sui pacchetti selezionati. Le parti indicate tra parentesi sono opzionali, nel senso che ci sono sintassi valide del comando in cui non compaiono opzioni di selezione dei pacchetti o azioni da compiere su questi ultimi.

In realtà non è necessario specificare le varie opzioni di `iptables` nell'ordine appena mostrato, né raggruppate nelle tre classi ivi elencate; esse possono essere invocate in qualunque ordine, ma usare questa schematizzazione facilita senz'altro la leggibilità del comando. Dato poi che in genere un firewall si costruisce inserendo tutte le relative regole di filtraggio all'interno del *netfilter* con una serie di chiamate successive ad `iptables` (in genere effettuate all'interno di uno script di shell), scrivere in maniera ordinata facilita la comprensione dello stesso.

Unica eccezione alla struttura appena mostrata è l'invocazione del comando senza opzioni, che si limita a stampare una riga di aiuto che invita ad usare le due opzioni `--help` o `-h`. Usando solo queste ultime è possibile ottenere la stampa a video di una breve sinossi delle opzioni principali. L'unica altra opzione utilizzabile singolarmente è, come per tutti i comandi che seguono lo standard GNU, `--version`, che stampa a video la versione del comando.

1.3.2 Le opzioni per il controllo di tabelle e catene

Cominciamo allora con il prendere in esame le varie opzioni che vengono utilizzate per scegliere su quale tabella o catena si va ad operare, per controllare i contenuti, per impostare le proprietà delle catene predefinite, per gestire la selezione, creazione, cancellazione e modifica delle catene create dall'utente stesso.

La prima opzione di `iptables` è `-t` (o `--table`) che permette di selezionare su quale tabella operare, usando come parametro il nome della stessa come indicato in sez. 1.2.2. Dato che il comando è usato principalmente per la realizzazione di firewall, se non si specifica nessuna tabella viene utilizzata di default la tabella `filter`, se invece si vogliono inserire delle regole in catene presenti sulle altre tabelle occorrerà sempre selezionarle esplicitamente specificandole come parametro per l'opzione `-t`.

Come accennato in sez. 1.2.2 oltre alle catene predefinite si possono creare (con l'opzione `-N`) delle nuove catene definite dall'utente, inoltre `iptables` consente di analizzare i contenuti di una catena, e di leggerne i contatori associati (che indicano il numero di pacchetti che l'hanno attraversata) con l'opzione `-L`. Si possono cancellare le regole di una catena con `-F` o impostarne la politica (solo per quelle predefinite) con `-P`. L'elenco completo delle opzioni per operare sulle catene è riportato in tab. 1.3.

Inoltre usando l'opzione `-v` si può aumentare la quantità di informazione fornita dal comando; l'uso di questa opzione è di particolare importanza quando usata insieme a `-L`, dato che permette di mostrare tutti i dettagli delle regole di selezione comprese quelle sulle interfacce, sul campo

Opzione		Significato
-N	--new-chain	Crea una nuova catena, il cui nome deve essere passato come parametro, e non deve corrispondere ad una catena già esistente.
-X	--delete-chain	Cancella una catena, il cui nome deve essere passato come parametro; la catena deve essere vuota e le catene predefinite non possono essere cancellate.
-E	--rename-chain	Rinomina una catena, il vecchio nome ed il nuovo devono essere passati come parametri, le catena predefinite non possono essere rinominate.
-L	--list	Elenca le regole presenti nella catena; se non si passa nessun parametro elenca le regole di tutte le catene nella tabella.
-F	--flush	Cancella tutte le regole presenti nella catena; se non si passa nessun parametro cancella le regole di tutte la catene nella tabella.
-Z	--zero	Azzera i contatori di una catena; se non si passa nessun parametro azzera i contatori di tutte le catene nella tabella.
-P	--policy	Imposta la politica di default per la catena, prende come parametro addizionale la destinazione da usare per tutti i pacchetti che arrivano alla fine della catena senza incontrare una corrispondenza in alcuna regola.

Tabella 1.3: Opzioni per la gestione delle catene.

TOS di IP, e le varie opzioni delle regole. Sempre con `-L` si possono usare anche due altre opzioni, `-n` che disabilita la risoluzione degli indirizzi e delle porte, che verranno stampati in forma numerica, e `--line-numbers` che stampa un numero progressivo per le regole presenti nella catena, in modo da poterlo utilizzare con le opzioni `-D`, `-R` e `-I`.

Dato che il funzionamento del *netfilter* prevede la scansione sequenziale delle catene, il comando `iptables` prevede diverse opzioni, riportate in tab. 1.4, per inserire o cancellare una regola all'interno di una catena in una posizione definita. Per ciascuna di questa opzioni deve essere specificata come parametro la catena da utilizzare, che può essere sia una di quelle predefinite, che una di quelle create con i comandi di tab. 1.3.

Opzione		Significato
-A	--append	Inserisce la regola in cosa alla catena. In questo modo la regola sarà sempre l'ultima della catena, fintanto che non se ne aggiunge un'altra.
-D	--delete	Cancella una regola nella catena, sia indicandola per posizione numerica nella catena, che esprimendola integralmente.
-R	--replace	Rimpiazza la regola nella catena, identificata con il numero che ne indica la posizione, con quella specificata nel comando.
-I	--insert	Inserisce una regola in una posizione specifica nella catena.

Tabella 1.4: Opzioni per l'inserimento di una regola all'interno di una catena.

Infine le opzioni `-R`, `-I` e `-D` prevedono un secondo parametro che specifica la posizione nella catena in cui rimpiazzare, inserire o cancellare una regola; questo deve essere indicato nella forma del numero progressivo della regola, che si può ottenere combinando l'opzione `--line-numbers` con quella per la stampa delle regole `-L`.

1.3.3 I criteri di selezione dei pacchetti

Come illustrato in sez. 1.3.1 oltre ad indicare tabelle e catene da usare, il comando `iptables` serve principalmente a creare le regole da inserire al loro interno. Per creare una regola allora, oltre alla catena su cui inserirla, occorre anche specificare un criterio di selezione dei pacchetti.

Per questo motivo la seconda classe di opzioni, la più numerosa, è quella che permette di definire i criteri per la selezione dei pacchetti, il cuore di ogni regola. Questi criteri possono essere di diversi tipi, i primi che affronteremo sono i criteri generici, che sono sempre disponibili perché forniti dall'infrastruttura stessa del *netfilter*, indipendentemente dal protocollo dei pacchetti o delle estensioni che si possono usare.

Si tenga presente poi che per quanto in seguito si elencheranno, per esigenza di chiarezza nell'esposizione, i vari criteri di selezione uno per uno, con le opzioni relative a ciascuno di essi, normalmente quando si definisce una regola da inserire in una catena, si possono specificare insieme più criteri di selezione. In tal caso il significato della regola è che un pacchetto, per avere una corrispondenza, deve soddisfarli tutti quanti; vale cioè una modalità di *AND* logico dei criteri di selezione, ad esempio una regola del tipo "`-s 192.168.1.24 -d 192.168.0.4`" richiede che il pacchetto abbia un certo indirizzo sorgente *e* un certo indirizzo di destinazione.

I criteri generici sono sostanzialmente i criteri che operano direttamente sui parametri più comuni contenuti nella intestazione dei pacchetti IP, e sulle interfacce da cui i pacchetti entrano o escono, e sono disponibili nativamente nel *netfilter* di Linux. Le relative opzioni sono le seguenti:

`-p, --protocol`
: Seleziona il pacchetto in base al protocollo del livello di trasporto, che deve essere specificato come parametro. I valori possibili sono `tcp`, `udp`, `icmp` o `all` per indicare tutti e tre. L'uso di questi valori speciali consente di attivare le opzioni specifiche relative a tali protocolli su cui torneremo più avanti, per cui in realtà quando si usano questi valori si va oltre i criteri generici. Oltre a questi nomi si può usare direttamente il valore numerico del protocollo, o il nome corrispondente che deve essere però indicato nel file `/etc/protocols`. Inoltre si può usare il carattere "`!`" per invertire la selezione. Possibili esempi sono:

```
-p tcp
--protocol ! udp
```

`-s, --src, --source`
: Seleziona il pacchetto in base all'indirizzo sorgente. L'indirizzo sorgente può essere specificato sia in forma numerica che con un nome a dominio; è però una pessima idea usare nomi che devono essere risolti attraverso una interrogazione al DNS, data la vulnerabilità di questo protocollo che può anche fornire risposte false. Si può anche indicare una intera rete, specificandola nella forma `indirizzo/rete`, dove la parte di rete può essere specificata sia come netmask, che con la notazione CIDR. Anche in questo caso si può invertire la selezione apponendo il carattere "`!`". Possibili esempi sono:

```
-s 192.168.1.1
-s 192.168.2.0/24
--src 10.0.0.0/255.0.0.0
--source ! 172.16.1.2
```

-d, --dst, --destination
: Seleziona il pacchetto in base all'indirizzo di destinazione. Come per l'indirizzo sorgente si può specificare (con la stessa sintassi) sia un indirizzo che una intera rete. Possibili esempi sono:

```
-d 192.168.1.1
-d 192.168.2.0/24
--dst 10.0.0.0/255.0.0.0
--destination ! 172.16.1.2
```

-i, --in-interface
: Seleziona il pacchetto sulla base dell'interfaccia di rete da cui proviene. L'opzione richiede come parametro il nome dell'interfaccia, e può essere usata solo sulle catene `INPUT`, `FORWARD` e `PREROUTING`. Oltre al nome esatto si può indicare una intera classe di interfacce usando il carattere "`+`", se ad esempio si vogliono selezionare tutte le interfacce Ethernet si potrà usare il parametro `eth+`. L'opzione supporta pure l'inversione della soluzione con l'uso del carattere "`!`". Possibili esempi sono:

```
-i ppp0
-i ! eth1
--in-interface tun+
```

-o, --out-interface
: Seleziona il pacchetto sulla base dell'interfaccia di rete in uscita a cui è destinato. L'opzione richiede come parametro il nome dell'interfaccia, e può essere usata solo sulle catene `FORWARD`, `OUTPUT` e `POSTROUTING`. Supporta le stesse estensioni per indicare classi di interfacce e la negazione della selezione viste per l'analoga precedente. Possibili esempi sono:

```
-o ppp0
-o ! eth0
--out-interface tun+
```

-f, --fragment
: Seleziona i pacchetti IP frammentati (si veda quanto detto a proposito in sez. 1.2.3 e sez. 1.4.3). Dato che solo il primo frammento contiene l'intestazione del protocollo IP i criteri di selezione basati su indirizzi sorgente o destinazione o altri dati desumibili dall'intestazione, non potranno mai corrispondere per dei frammenti successivi al primo che allora vengono selezionati tramite questa opzione. L'opzione può essere preceduta da un carattere "`!`" che inverte la selezione. Possibili esempi sono:

```
-f
! --fragment
```

Come appena accennato, qualora si richieda una corrispondenza per protocollo che coinvolge uno dei tre protocolli principali del TCP/IP (vale a dire quando si specifica uno dei criteri con `-p tcp`, `-p udp` o `-p icmp`) vengono attivate automaticamente una serie di ulteriori possibili criteri di selezione basati sulle caratteristiche particolari di questi stessi protocolli.

In particolare quando si usano sia `tcp` che `udp` si potranno effettuare selezioni ulteriori sui valori delle porte utilizzate da detti protocolli, secondo i criteri specificabili tramite le opzioni:

--sport, --source-port
Seleziona il pacchetto in base alla porta sorgente. La porta può essere specificata sia per numero che per il nome utilizzato in `/etc/services`.[21] Usando i numeri si può specificare un intervallo di porte usando il carattere "`:`" come separatore, omettendo uno dei due estremi di un intervallo si sottintende l'uso del relativo valore massimo (65535) o minimo (0). Anche in questo caso si può invertire una selezione apponendo ad essa il carattere "`!`". Possibili esempi sono:

```
--sport 25
--sport ! smtp
--source-port 0:1024
--source-port 25000:
```

--dport, --destination-port
Seleziona il pacchetto in base alla porta di destinazione. La porta può essere specificata sia per numero che utilizzando il nome utilizzato in `/etc/services`, usando la stessa sintassi vista in precedenza per `--sport`. Anche in questo caso si può invertire la selezione apponendo il carattere "`!`". Possibili esempi sono:

```
--dport 80
--dport ! ssh
--destination-port 25000:
```

Quando si effettua il filtraggio sul protocollo `udp` le precedenti sono le sole opzioni aggiuntive disponibili, nel caso invece si filtri su `tcp` invece diverranno utilizzabili anche le seguenti ulteriori opzioni:

--tcp-flags
Seleziona il pacchetto in base allo stato dei flag del protocollo TCP, questi sono una serie di bit nell'intestazione dei pacchetti utilizzati nella gestione delle connessioni. L'opzione richiede due parametri, il primo è la lista dei flag da tenere sotto controllo (separati da virgole e senza spazi) ed il secondo la lista di tutti e soli i flag che fra i precedenti si richiede siano attivi. I valori usati per specificare i flag sono quelli dei rispettivi nomi, come indicati nella definizione del protocollo,[22] cioè `SYN`, `ACK`, `FIN`, `RST`, `URG`, `PSH`, a cui si aggiungono i due valori `ALL` e `NONE`, il cui significato è ovvio. Possibili esempi sono:

```
--tcp-flags SYN,ACK,FIN,RST SYN
--tcp-flags ! SYN,ACK,RST SYN
```

--syn Seleziona i pacchetti che hanno il flag SYN attivo e i flag ACK e RST disattivi, che sono la caratteristica che identifica il pacchetto TCP usato per iniziare una connessione. Pertanto bloccare questi pacchetti significa bloccare ogni connessione TCP. È una abbreviazione della regola `--tcp-flags SYN,RST,ACK SYN`. Al solito apponendo ad essa il carattere "`!`" si inverte la selezione. Possibili esempi sono:

[21] si tenga presente però che in questo caso la scrittura delle regole viene ritardata dalla risoluzione del nome, e quando si scrive un gran numero di regole, il ritardo può essere sensibile.
[22] si faccia riferimento all'RFC 793.

```
--syn
! --syn
```

--tcp-option
Seleziona il pacchetto in base al valore della sezione dell'intestazione di un pacchetto TCP chiamata appunto *TCP option*. Questa indica delle sezioni appunto opzionali che si possono trovare nell'intestazione, queste hanno sempre una struttura divisa in tre campi in cui il primo indica il tipo di opzione, ed è un numero a 8 bit selezionabile con questa opzione. I restanti campi sono la lunghezza dell'opzione[23] ed i relativi dati. Usando il carattere "`!`" si inverte la selezione. Possibili esempi sono:

```
--tcp-option 16
! --tcp-option 16
```

--mss Seleziona i pacchetti TCP con flag SYN o SYN/ACK che abbiano il valore specificato per la MSS *Maximum Segment Size*, un parametro che controlla la dimensione massima dei pacchetti per quella connessione. Si può inoltre specificare un intervallo di valori usando il carattere "`:`". Possibili esempi sono:

```
--mss 100
--mss 150:200
```

Infine quando si specifica il protocollo ICMP con `-p icmp` diventa disponibile una ulteriore selezione tramite l'opzione `--icmp-type` che seleziona il tipo di pacchetti; si possono selezionare i vari tipi di pacchetti sia secondo il valore numerico, come specificato nella definizione del protocollo, (si veda l'RFC 792) che utilizzando uno dei nomi ottenibili con il comando `iptables -p icmp -h`, che si sono riportati in tab. 1.10.

1.3.4 Le estensioni dei criteri di selezione

I precedenti criteri di selezione sono quelli più elementari, che supportano la selezione sulla base dei principali parametri dei protocolli più importanti. La flessibilità del *netfilter* è che diventa possibile, scrivendo degli opportuni moduli per il kernel, estendere le capacità di selezione dei pacchetti.

Per questo esiste l'opzione `-m` (o `--match`) che permette di attivare queste estensioni, ciascuna delle quali ha un suo nome (di norma corrispondente al nome del modulo usato dal kernel). La sintassi comune in questi casi è richiedere l'uso dell'estensione con il comando `-m estensione` per poi utilizzarla usando le nuove opzioni ad essa relative, che diventano disponibili una volta che la si è attivata.

Grazie a questi moduli le capacità di selezione dei pacchetti possono essere ampliate in maniera incredibilmente estesa e complessa, mettendo a disposizione, oltre a criteri basati sulle proprietà specifiche di ciascun pacchetto (in genere relativi a dati presenti nelle intestazioni dei protocolli) altri criteri più astratti, relativi a proprietà più ampie e non legate al singolo pacchetto, come quelle usate dal filtro sugli stati il cui funzionamento è illustrato in sez. 1.2.3.

[23] essendo opzionale non è necessario che tutti i tipi di opzione siano riconosciuti, per questo se uno non lo è deve essere possibile scartare i dati relativi passando alla sezione successiva.

Un elenco delle principali estensioni dei criteri di selezione è quello seguente, in cui per ciascuna estensione si riportano le principali caratteristiche e le ulteriori opzioni di controllo abilitate:

`-m limit`
L'estensione permette di utilizzare uno dei criteri di selezione speciale che non dipende dal contenuto di un singolo pacchetto, ma da proprietà "*collettive*" degli stessi. Questa estensione permette di creare una selezione in base al flusso (nel senso di numero per unità di tempo) di pacchetti che corrispondono ad un determinato criterio. Si può così limitare il flusso di pacchetti accettati, in modo da rispondere ad eventuali attacchi di DoS (vedi sez. **??**) basati su tecniche di *flood*.[24]

Il funzionamento di questa estensione non è banale, dato che non è semplice calcolare un flusso medio di pacchetti, in quanto il valore di una media temporale dipende strettamente dal periodo di tempo su cui la si calcola. Se si contasse semplicemente il numero di pacchetti che arrivano durante un certo intervallo di tempo, una raffica concentrata di pacchetti potrebbe dare un flusso elevatissimo se l'intervallo su cui si media è piccolo e coincide con il picco, e nullo nei successivi intervalli, con variazioni enormi dei possibili valori della media a seconda della dimensione dell'intervallo in cui la si calcola.

Per evitare questo problema e garantire comunque un flusso medio affidabile il meccanismo di selezione è assimilabile a quello di un *serbatoio* di pacchetti. Fintanto che nel serbatoio c'è spazio tutti i pacchetti vengono accettati, e il criterio di selezione corrisponde. Quando lo spazio si esaurisce ogni ulteriore pacchetto sarà scartato. Il serbatoio però viene svuotato ad un ritmo costante, cosicché dopo un certo tempo si libera dello spazio e un nuovo pacchetto potrà essere accettato. In questo modo si può garantire su una media di lungo periodo (a serbatoio vuoto si può avere un picco dovuto ad una raffica di pacchetti) un flusso costante.

Per controllare questo meccanismo l'estensione introduce due opzioni, la prima delle quali, `--limit-burst`, è quella che ci permette di stabilire la dimensione in numero di pacchetti del nostro serbatoio. Il suo valore di default è di 5 pacchetti; questo significa che una volta inserita la regola i primi 5 pacchetti saranno sempre accettati, mentre per i successivi si dovrà aspettare che il *serbatoio* si sia svuotato a sufficienza per accettarne di altri. Questo comportamento ci fa anche capire il significato del nome dell'opzione: con questa possiamo decidere fino a che punto vogliamo rispondere a dei flussi concentrati (i *burst*) di pacchetti.

Invece l'opzione che controlla lo *svuotamento* del nostro serbatoio, e definisce pertanto il flusso medio di pacchetti che accetteremo, è `--limit` che permette di definire la frequenza con cui si libera uno spazio per un nuovo pacchetto nel serbatoio (il valore di default è 3 all'ora). L'opzione prende un parametro del tipo `N/tempo` dove `N` indica il numero di spazi da liberare, e `tempo` l'intervallo di tempo in cui farlo; quest'ultimo deve essere specificato tramite una opportuna parola chiave, l'opzione accetta i valori `second`, `minute`, `hour`, o `day`. Possibili esempi sono:

[24] si chiamano così le tecniche che mirano a creare una "*inondazione*" di pacchetti nei confronti di una macchina bersaglio, in modo da poterla bloccare nel tentativo di rispondere.

```
--limit 3/second
--limit-burst 10
```

-m connbytes

L'estensione consente di effettuare una selezione in base al numero di pacchetti/bytes passati su una connessione, sia nelle singole direzioni che in totale, e permette così di individuare quelle che consumano più banda in modo da poterle adeguatamente abbassare di priorità. Perché l'estensione possa funzionare è necessario abilitare la contabilità dei pacchetti in transito sulle connessioni scrivendo un valore diverso da zero in `/proc/sys/net/netfilter/nf_conntrack_acct`, ma si tenga presente che questa sarà presente solo per le nuove connessioni create dopo l'abilitazione. Se l'informazione non è presente il criterio non corrisponderà mai.

L'estensione abilita tre nuove opzioni; la prima è `--connbytes` che indica l'intervallo dei valori in pacchetti o bytes su cui effettuare la selezione e deve essere specificata con valori numerici nella forma `min:max` dove la seconda parte (`:max`) può essere omessa per impostare solo un valore minimo, la selezione inoltre può essere invertita anteponendo il carattere "`!`". La seconda opzione è `--connbytes-dir` che indica quali direzioni del flusso di pacchetti prendere in esame nella selezione e richiede come argomento uno fra i valori `original` (i pacchetti in uscita che han creato la connessione) `reply` (le risposte) e `both` (entrambe le direzioni). La terza opzione è `--connbytes-mode` che invece consente di indicare se i valori specificati per `--connbytes` devono essere considerati come numero di pacchetti (`packets`), numero di byte (`bytes`) o dimensione media del pacchetto `avgpkt`. Un possibile esempio è:

```
--connbytes 1000000 --connbytes-dir both --connbytes-mode bytes
--connbytes 10000 --connbytes-dir reply --connbytes-mode packets
```

-m mac

L'estensione permette di selezionare, attraverso l'opzione `--mac-source`, il pacchetto sulla base del MAC address sorgente del pacchetto (in genere Ethernet) che arriva sull'interfaccia di rete. Questo criterio è disponibile solo sulle catene di `PREROUTING`, `FORWARD` e `INPUT`. Al solito è possibile invertire la selezione con "`!`". Possibili esempi sono:

```
--mac-source 00:0A:95:74:C3:D4
--mac-source ! 00:0A:95:74:C3:D4
```

-m mark

L'estensione permette di selezionare, attraverso l'opzione `--mark`, i pacchetti che attraversano il *netfilter* che sono stati marcati con un opportuno valore tramite l'uso dell'azione `MARK` (vedi sez. 1.3.5 per la spiegazione del meccanismo). L'opzione prende come parametro il valore numerico impostato per il relativo pacchetto. Questo può essere specificato anche nella forma `valore/maschera`, dove la maschera viene usata per eseguire un AND binario del valore del `MARK` del pacchetto prima di eseguire il confronto. Possibili esempi sono:

```
--mark 10
--mark 1/7
```

-m multiport
L'estensione permette di utilizzare le opzioni di selezione per le porte sorgente e destinazione `--source-ports` e `--destination-ports`. Queste sono molto simili (presentano solo una `s` aggiunta al nome) a quelle illustrate in sez. 1.3.4 per i protocolli TCP e UDP e come in quel caso possono essere usate solo se si è richiesta anche una selezione sui suddetti protocolli (con `-p tcp` o `-p udp`). L'estensione permette di usare una sintassi in cui diventa possibile selezionare le porte con una lista di numeri (separati da virgole e senza spazi, con un limite massimo di 15 porte diverse) invece che singolarmente o come intervalli. L'estensione definisce inoltre l'opzione `--ports` che permette di applicare la stessa selezione alla lista di numeri indicati richiedendo che porta destinazione e sorgente siano identiche, e corrispondenti ad una delle porte della lista. Possibili esempi sono:

```
--source-ports 25,80,111
--destination-ports 25,80,111
--ports 5000,5001
```

-m owner
L'estensione permette di filtrare usando alcune caratteristiche specifiche del processo che ha creato i pacchetti, ed è utilizzabile soltanto sulla catena di `OUTPUT`. Dato che alcuni pacchetti (ad esempio quelli ICMP) vengono creati dal kernel e non sono mai associabili ad un processo, essi non potranno mai essere selezionati da questa estensione. L'elenco delle opzioni possibili è il seguente:

- **--uid-owner** seleziona i pacchetti creati da un processo con user-ID effettivo uguale a quello specificato come parametro.
- **--gid-owner** seleziona i pacchetti creati da un processo con group-ID effettivo uguale a quello specificato come parametro.
- **--pid-owner** seleziona i pacchetti creati dal processo il cui *pid* è uguale a quello specificato.
- **--sid-owner** seleziona i pacchetti creati da un processo il cui session-ID è uguale a quello specificato come parametro.
- **--cmd-owner** seleziona i pacchetti creati da un processo invocato con il comando specificato.[25]

-m state
Questa è l'estensione che permette di utilizzare il filtro degli stati, la cui descrizione dettagliata è in sez. 1.2.4, usando le informazioni contenute nel sistema di *connection tracking* (anche questo descritto in sez. 1.2.3). L'estensione abilita l'opzione `--state`, cui si passa la lista (separata da virgole e senza spazi) degli stati che si vogliono selezionare, secondo i nomi riportati in tab. 1.2.

-m tos
Questa estensione permette di utilizzare l'opzione `--tos` per selezionare i pacchetti in base al valore del campo TOS (*Type of Service*) dell'intestazione dei pacchetti IP (vedi

[25] questa opzione è presente solo quando `iptables` è stato compilato con un kernel che supporta questa funzionalità.

fig. 1.7). Questo è un campo di 8 bit usato[26] per differenziare i flussi di dati, che serve a caratterizzarne la tipologia; ad esempio per indicare che i pacchetti devono essere trasmessi il più rapidamente possibile, o che invece occorre ottimizzare la quantità dei dati trasmessi, più che la rapidità o la latenza. L'opzione prende come parametro il valore numerico del TOS espresso in cifra esadecimale, o tramite uno dei valori simbolici illustrati in tab. 1.5.

Valore		Significato
`Minimize-Delay`	`0x10`	Ottimizzare la trasmissione per rendere più veloce possibile la ritrasmissione dei pacchetti (usato per traffico interattivo di controllo come SSH).
`Maximize-Throughput`	`0x8`	Ottimizzare la trasmissione per rendere il più elevato possibile il flusso netto di dati (usato su traffico dati, come quello di FTP).
`Maximize-Reliability`	`0x4`	Ottimizzare la trasmissione per ridurre al massimo le perdite di pacchetti (usato su traffico come TFTP o BOOTP).
`Minimize-Cost`	`0x2`	Ottimizzare la trasmissione per utilizzare i collegamenti con minor costo (usato per i protocolli di streaming).
`Normal-Service`	`0x0`	Nessuna richiesta specifica.

Tabella 1.5: Valori del campo TOS utilizzabili con l'opzione `--tos`.

`-m ttl`
: Questa estensione permette di utilizzare l'opzione `--ttl` per selezionare i pacchetti in base al valore del campo TTL (*Time to Live*) dell'intestazione dei pacchetti IP (vedi fig. 1.7). Il campo indica il numero di router che il pacchetto può ancora attraversare prima di essere scartato.

`-m unclean`
: L'estensione non definisce nessuna opzione ulteriore e si limita a selezionare tutti i pacchetti che sembrano essere malformati o avere caratteristiche non usuali. In genere la si usa per identificare il traffico sospetto. L'estensione è considerata ancora sperimentale e l'uso della stessa per scartare i pacchetti può risultare pericoloso e portare anche alla perdita di pacchetti facenti parti di connessioni perfettamente regolari.

1.3.5 Le azioni sui pacchetti

Il terzo ed ultimo componente di ogni regola di `iptables` è l'azione da applicare ad ogni pacchetto selezionato nella regola stessa, quello che nel vocabolario del comando è chiamato *target*. L'opzione che controlla questa azione, che decide il destino del pacchetto quando questo corrisponde alla regola di selezione, è `-j` (o `--jump`), seguita dall'identificatore dell'azione da applicare (il nome del *target*).

Come accennato in sez. 1.2.1 nell'infrastruttura del *netfilter*, a livello del kernel, sono già previste alcune azioni elementari possibili, che si riflettono in quelli che in `iptables` sono chiamati gli *special target*, cioè una serie di azioni che sono sempre definite e che determinano la conclusione della scansione delle regole all'interno di una catena predefinita (o in una eventuale ulteriore

[26] non molto, in realtà, dato che non tutti i sistemi operativi lo supportano e molti amministratori ignorano completamente questa proprietà nella configurazione dei router.

catena definita dall'utente su cui lo si poteva esser mandato). Una di queste è sempre quella di inviare il pacchetto su una catena definita dall'utente (da questo deriva il nome *jump*), le altre sono riportate in tab. 1.6.

Opzione	Significato
`ACCEPT`	Accetta il pacchetto che viene fatto immediatamente proseguire per le successive catene (secondo lo schema di attraversamento di fig. 1.4) all'interno nel *netfilter*.
`DROP`	Il pacchetto viene scartato immediatamente, ed ogni ulteriore controllo non è più necessario. Nessuna altra regola verrà presa in considerazione.
`QUEUE`	Il pacchetto viene passato in user space (dove può essere letto dagli opportuni programmi).
`RETURN`	Blocca l'attraversamento della catena corrente e torna a quella precedente, se usato su una catena predefinita applica la politica di default della stessa.

Tabella 1.6: Le azioni elementari (*special target*) del comando `iptables`.

Una delle caratteristiche di `iptables` è che oltre ai criteri di selezione è possibile ampliare anche le possibili azioni che si possono eseguire sui pacchetti, al di là di quelle elementari di tab. 1.6, attraverso quelle che vengono chiamate le "*target extension*", sempre grazie all'utilizzo di ulteriori moduli nel kernel che definiscono nuovi parametri (corrispondenti a nuovi *target*) da utilizzare con l'opzione `-j`, che a loro volta possono richiedere ulteriori opzioni.

Le principali estensioni dei *target* disponibili come azioni da compiere sui pacchetti è data dalla seguente lista (non completa, dato che molte estensioni non è detto siano incluse nei sorgenti ufficiali del kernel):

`DNAT` Questa è una azione valida solo nella tabella di `nat` e nelle catene `PREROUTING` e `OUTPUT`, e nelle eventuali catene definite dall'utente se vi si *salta* a partire da una di queste.

Questa azione permette di riscrivere l'indirizzo di destinazione di un pacchetto. Come già accennato in sez. 1.2.3 la regola di selezione viene applicata solo sul primo pacchetto di una connessione (necessita quindi dell'uso del sistema del *conntrack*) e poi viene automaticamente ripetuta per tutti i pacchetti facenti parte della stessa connessione. Una volta modificato il pacchetto questo verrà instradato secondo il nuovo indirizzo.

Si tenga presente inoltre che siccome il *Destination NAT* è operato nella catena di `PREROUTING`, nel successivo attraversamento del *netfilter* i pacchetti modificati con `DNAT` si presenteranno con l'indirizzo modificato, per cui tutte le eventuali regole di firewall che si vogliono mettere nelle catene successive dovranno usare quest'ultimo. Lo stesso vale per i pacchetti modificati sulla catena di `OUTPUT`, in quanto la catena di `nat` è attraversata prima di quella di `filter` (si riveda lo schema di fig. 1.4). In caso di DNAT le sole catene che vedono l'indirizzo originale del pacchetto sono quella di `PREROUTING` e quella di `OUTPUT` ma solo nella tabella di `mangle`.

La traslazione degli indirizzi viene specificata attraverso l'opzione `--to-destination`, questa può prendere come parametro un singolo indirizzo di destinazione, od un intervallo che va indicato con gli indirizzi degli estremi separati da un "`-`" senza spazi interposti. In questo caso la traslazione sceglierà a turno uno dei valori dell'intervallo

(dato che la scelta è fatta solo sul primo pacchetto di una connessione, ed i successivi pacchetti subiscono la stessa traslazione, questo garantisce che tutto il traffico vada verso lo stesso IP finale). In questo modo si può eseguire una forma semplificata (*round-robin*) di bilanciamento di carico.

Qualora si effettui la traslazione per pacchetti TCP o UDP, si potrà effettuare la traslazione anche della porta di destinazione, da specificare singolarmente facendola seguire all'indirizzo di destinazione separata con un ":". Si potrà anche specificare un intervallo di porte indicando i due estremi separati da un "-". Questo è possibile solo avendo abilitato con `-p` la relativa selezione sui suddetti protocolli. Possibili esempi sono:

```
--to-destination 192.168.2.2
--to-destination 192.168.2.2:8080
--to-destination 192.168.2.16-192.168.2.32
--to-destination 192.168.2.16:6000-6010
```

LOG Questa azione abilita la registrazione sul *syslog* dei dati dei pacchetti selezionati. Dato che la registrazione viene eseguita dal kernel la facility del *syslog* utilizzata è `kern`; questo significa che i messaggi vengono mostrati anche dal comando `dmesg` e inviati, a meno di non ridurne la priorità, sulla console. Una regola con criterio di selezione che usi questa azione come *target* non termina la scansione della catena, che proseguirà dalla regola successiva. Per questo se si vuole eseguire il logging di pacchetti che poi si desidera scartare occorre sempre usare una azione di `LOG` prima di quella di `DROP`. Si tenga presente inoltre che i log contengono informazioni dettagliate riguardo il vostro traffico di rete, che possono essere lette dagli utenti e costituire un possibile rischio di sicurezza.

Con l'uso di questa azione vengono attivate una serie di opzioni che permettono di definire le modalità in cui il logging viene eseguito, il loro elenco è il seguente:

`--log-level`
: imposta la priorità dei messaggi, prende come parametro il valore numerico della priorità o il corrispondente identificativo, come si può ottenere dalla pagina di manuale del *syslog*, accessibile con `man syslog.conf`. Si può usare questo valore e le impostazioni del *syslog* per evitare che i log prodotti da `iptables` siano letti dagli utenti. Possibili esempi sono:

  ```
  --log-level 3
  --log-level debug
  ```

`--log-prefix`
: Permette di aggiungere una stringa (di un massimo di 29 caratteri) da premettere ai messaggi di log, una caratteristica utile per aumentarne la leggibilità. Possibili esempi sono:

  ```
  --log-prefix "INPUT"
  ```

--log-tcp-sequence
: Permette di inserire nei log i *numeri di sequenza*[27] dei pacchetti TCP. Questa opzione non ha parametri.

--log-tcp-options
: Permette di inserire nei log il valore delle opzioni del protocollo TCP. Questa opzione non ha parametri.

--log-ip-options
: Permette di inserire nei log il valore delle opzioni del protocollo IP. Questa opzione non ha parametri.

MARK Questa azione è valida solo nella tabella di `mangle`, e serve per marcare i pacchetti con un valore numerico, questo non va a modificare niente nel contenuto del pacchetto stesso, ma è una sorta di etichetta che viene attaccata al pacchetto che resterà associata al pacchetto all'interno del kernel. Così sarà possibile filtrare i pacchetti sulla base di tale valore con l'estensione `--mark` vista in sez. 1.3.4, ma lo stesso valore sarà accessibile anche dal sistema di routing avanzato ed in questo modo si potranno impostare politiche di routing diverse (ad esempio una limitazione sul traffico) per i pacchetti "*marcati*" tramite questa azione.

Il valore da usare per marcare il pacchetto deve essere specificato attraverso l'opzione `--set-mark` che prende come parametro il valore numerico da associare al pacchetto. Si ricordi che questo valore è puramente interno al kernel e pertanto non sarà disponibile quando il pacchetto transiterà su altre macchine. Possibili esempi sono:

```
--set-mark 4
```

MASQUERADE
Questa azione è valida soltanto nella tabella di `nat` e nella catena di `POSTROUTING`. Sostanzialmente è una abbreviazione di `SNAT` e viene usata principalmente per condividere le connessioni temporanee ad Internet (in genere quelle via modem gestite con `pppd` dove viene assegnato un IP dinamico). L'azione è quella di fare un *Source Network Address Translation* usando l'IP assegnato dinamicamente (ricavato da quello assegnato all'interfaccia da cui i pacchetti devono uscire). In tutti gli altri casi è meglio usare `SNAT` in quanto la risoluzione dell'indirizzo dall'interfaccia comporta un certo degrado delle prestazioni.

L'uso di `MASQUERADE` ha inoltre l'effetto di cancellare tutte le connessioni presenti quando l'interfaccia va giù, dato che è poco probabile avere lo stesso indirizzo quando l'interfaccia sarà riutilizzata e che in ogni caso si saranno dovuti bloccare tutti i programmi che le usavano. In questo modo si evita che restino ad occupare spazio nella tabella delle connessioni voci ormai inutilizzabili, e che si possa ricevere del traffico spurio corrispondente a connessioni che in realtà non esistono più.

Se si opera su pacchetti TCP e UDP (specificando l'uso di detti protocolli con `-p`) con l'uso di questa azione viene abilitata anche l'opzione `--to-ports` che può essere

[27] i *numeri di sequenza* sono delle informazioni caratteristiche del protocollo TCP (vedi fig. 1.9), mantenute nelle intestazioni dei pacchetti, che identificano ciascun pacchetto e permettono di inserirlo al posto giusto nel flusso dei dati quando questo viene riassemblato.

usata per soprassedere l'assegnazione automatica delle porte sorgente fatta dal kernel (torneremo su questo nella spiegazione di SNAT) con un intervallo da specificare come parametro indicandone i due estremi separati con un "-". Si può anche indicare una sola porta. Possibili esempi sono:

```
--to-ports 10000
--to-ports 10000-15000
```

REDIRECT

Questa azione è valida soltanto nella tabella di nat e nelle catene di PREROUTING e OUTPUT (o nelle catene definite dall'utente che derivino da queste) e serve a modificare l'IP di destinazione dei pacchetti selezionati perché questi vengano inviati alla macchina stessa (i pacchetti generati localmente vengono mappati sul localhost). Qualora si siano selezionati (con -p) dei pacchetti TCP o UDP diventa disponibile l'opzione --to-ports che permette anche di specificare su quale porta effettuare la redirezione (normalmente viene usata quella originaria del pacchetto). L'opzione permette anche di specificare un intervallo, indicato tramite gli estremi estremi separati con un "-". Possibili esempi sono:

```
--to-ports 8080
--to-ports 8000-8010
```

REJECT Questa azione è sostanzialmente equivalente a DROP, ma oltre ad eliminare il pacchetto fermando ogni ulteriore scansione delle regole, si incarica anche di inviare indietro un opportuno messaggio di errore. L'azione è utilizzabile solo nelle catene di INPUT, FORWARD e OUTPUT. L'azione abilita l'uso dell'opzione --reject-with che permette di specificare il pacchetto inviato come messaggio di errore, e prende come parametro uno dei valori riportati in tab. 1.7.

Opzione	Significato
icmp-net-unreachable	Restituisce un pacchetto ICMP di tipo *net unreachable*.
icmp-host-unreachable	Restituisce un pacchetto ICMP di tipo *host unreachable*.
icmp-port-unreachable	Restituisce un pacchetto ICMP di tipo *port unreachable*.
icmp-proto-unreachable	Restituisce un pacchetto ICMP di tipo *proto unreachable*.
icmp-net-prohibited	Restituisce un pacchetto ICMP di tipo *net prohibited*.
icmp-host-prohibited	Restituisce un pacchetto ICMP di tipo *host prohibited*.
icmp-admin-prohibited	Restituisce un pacchetto ICMP di tipo *admin prohibited*.
tcp-reset	Restituisce un pacchetto TCP con il flag RST attivo.

Tabella 1.7: I vari tipi di messaggi di errore utilizzabili con l'opzione --reject-with.

Il valore di default è icmp-port-unreachable, degli altri icmp-admin-prohibited può essere usato solo con i kernel che lo supportano, mentre tcp-reset può essere usato solo con regole che selezionano pacchetti TCP, e serve a chiudere in maniera pulita una connessione TCP.

SNAT Questa azione è valida solo nella tabella di nat e nella catena di POSTROUTING. Come le altre regole che operano nella tabella di nat essa si applica solo al primo pacchetto di una connessione e poi viene automaticamente ripetuta per tutti i pacchetti facenti parte

della stessa connessione. L'azione modifica l'indirizzo sorgente dei pacchetti eseguendo il *Source NAT*,[28] e si usa di solito per condividere una connessione ad internet, facendo in modo che tutti i pacchetti in uscita figurino come originanti dalla macchina corrente; il meccanismo del *connection tracking* si incarica di effettuare automaticamente la traslazione inversa (sull'indirizzo di destinazione) per i pacchetti ottenuti in risposta ai precedenti.

La traslazione degli indirizzi viene specificata attraverso l'opzione `--to-source`, questa può prendere come parametro un singolo indirizzo sorgente, od un intervallo che va indicato con gli indirizzi degli estremi separati da un "`-`" (senza spazi interposti). In questo caso la traslazione sceglierà a turno uno dei valori dell'intervallo, ottenendo una forma semplificata (*round-robin*) di bilanciamento.

Qualora si effettui la traslazione per pacchetti TCP o UDP (selezionati in precedenza con `-p`) si può anche specificare (con sintassi analoga a quella di `DNAT`) una porta sorgente o un intervallo di porte da utilizzare per la traslazione. In tal caso i pacchetti uscenti avranno come porta sorgente un valore compreso nell'intervallo specificato. Possibili esempi sono:

```
--to-source 192.168.2.2
--to-source 192.168.2.16-192.168.2.20
--to-source 192.168.2.16:10000-30000
```

In realtà per poter gestire più connessioni contemporanee l'utilizzo di questa opzione prevede che possano essere modificati, oltre l'indirizzo sorgente, anche altre caratteristiche dei pacchetti. Il kernel cerca di modificare il meno possibile i pacchetti limitandosi al solo indirizzo sorgente, ma qualora le modifiche siano necessarie perché si ha un conflitto (ad esempio due connessioni allo stesso server sullo stesso servizio che usano la stessa porta sorgente) il kernel deve intervenire per consentire una identificazione univoca della connessione. Questo nel caso dei pacchetti TCP e UDP significa che deve essere modificata anche la porta sorgente, nel qual caso la regola seguita è che (sempre che non si sia specificato in un intervallo definito) se la porta originale è inferiore a 512 vengono usate altre porte sotto 512, se è fra 512 a 1024 vengono usate porte inferiori a 1024 e per tutte le altre porte sono usate quelle a partire da 1024.

Nel caso di ICMP invece, non esistendo i numeri di porta, in caso di più pacchetti inviati allo stesso IP, viene modificato il campo di codice.

`NOTRACK` Questa azione è valida solo nella tabella di `raw` e consente di disabilitare il *connection tracking* per tutti i pacchetti che corrispondono al criterio di selezione usato nella regola che la usa come `target`.

`TOS` Questa azione è valida solo nella tabella di `mangle`, ed abilita l'uso dell'opzione `--set-tos` che permette di modificare il valore del campo TOS nell'intestazione di IP,[29] usato per

[28] questo è il caso più comune di NAT, che viene implementato di default nella gran parte dei router usati per connettersi alle linee DSL, tanto che spesso si usa NAT come indicativo di questo particolare tipo di traslazione.

[29] come accennato in sez. 1.3.3 per l'omonimo criterio di selezione, il *Type of Service* è un campo usato per classificare il traffico IP in una serie di classi in modo che possa essere trattato diversamente dai router.

dare ai router indicazioni sul tipo di traffico di cui il pacchetto fa parte. L'opzione prende come parametro il nuovo valore del TOS, che può essere specificato sia con il valore numerico esadecimale che con uno degli identificativi già visti in tab. 1.5 per l'opzione relativa all'omonimo criterio di selezione. Possibili esempi sono:

```
--set-tos 0x10
--set-tos Normal-Service
```

TTL Questa azione è valida solo nella tabella di `mangle`, ed abilita l'uso di una serie di opzioni che permettono di modificare il valore del campo TTL di IP, usato per limitare il numero di volte che un pacchetto può transitare attraverso un router, in modo da evitare l'intrappolamento permanente in eventuali *routing loop*.[30] Questa azione può servire a riportare allo stesso valore di TTL tutti i pacchetti uscenti dalla macchina, in modo da mascherare all'esterno i valori diversi che indicherebbero la presenza di una operazione di `SNAT`. Le opzioni abilitate con l'uso di questa azione sono:

- **`--ttl-set`** imposta un valore per il TTL dei pacchetti selezionati, da specificare come parametro per l'opzione (il valore di default usato dal kernel è 64).
- **`--ttl-dec`** permette di decrementare il valore del TTL dei pacchetti selezionati del valore specificato come parametro. Questo può essere utilizzato per bloccare il raggiungimento di certi servizi (di norma il DNS) da macchine troppo lontane, per costringerle ad usare server più vicini.
- **`--ttl-inc`** permette di incrementare il valore del TTL dei pacchetti selezionati del valore specificato come parametro, dato che al passaggio da un router questo valore viene decrementato di uno, con questa opzione lo si può reincrementare in modo da non far rilevare la presenza del nostro firewall in programmi come `traceroute`.

Possibili esempi sono:

```
--ttl-set 64
--ttl-dec 5
--ttl-inc 1
```

1.3.6 Programmi di gestione

Abbiamo fin qui trattato le varie capacità del *netfilter* di Linux, e come queste possono essere utilizzate attraverso l'uso del comando `iptables`; esso però consente di eseguire sulle tabelle del kernel solo una operazione alla volta, per questo sono stati creati degli opportuni programmi per permettere la gestione di *gruppi* di regole.

Il comando `iptables` infatti permette di inserire le regole una alla volta, per questo in genere si suole costruire un firewall tramite uno script di shell che si incarica di invocare `iptables` più

[30] si chiama così quella situazione in cui, per un qualche malfunzionamento della rete, i pacchetti si trovano ad essere reinviati ad un router da cui sono provenuti, innescando così un circolo vizioso; per questo il valore di questo campo viene decrementato di una unità ogni volta che un pacchetto attraversa un router e quando si annulla il pacchetto viene scartato e viene inviato in messaggio di errore (*ICMP time expired*).

volte. Quando però le regole diventano molte questo meccanismo comporta una penalizzazione nelle prestazioni, infatti si deve ripetere il comando continuamente, e questo comporta che ogni volta deve essere estratto dal kernel l'intero insieme delle regole, che deve essere opportunamente modificato secondo quanto specificato dal comando, e poi reinserito nel kernel, compiendo così un gran numero volte il trasferimento dei dati da kernel space ad user space.

Per velocizzare queste operazioni insieme ad `iptables` vengono forniti altri due programmi che permettono il trasferimento di un gruppo di regole in un blocco unico. Il primo è `iptables-save`, che serve a scrivere (sullo standard output) l'insieme di regole correntemente attive, utilizzando un apposito formato di facile scansione. Il comando permette anche, con l'opzione `-c` (o `--counters`) di salvare i valori dei contatori associati a ciascuna regola,[31] mentre con `-t` (o `--table`) si può richiedere il salvataggio delle regole relative ad una singola tabella, invece che a tutte e tre. Un esempio del formato generato da questo comando è il seguente:

iptables-save

```
# Generated by iptables-save v1.2.9 on Sun Dec 14 19:52:50 2003
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -s 127.0.0.1 -d 127.0.0.1 -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
...
COMMIT
# Completed on Sun Dec 14 19:52:50 2003
```

Il secondo comando, `iptables-restore`, permette di caricare nel kernel un insieme di regole letto dallo standard input,[32] nello stesso formato usato da `iptables-save`. A meno di non specificare l'opzione `-n` (o `--noflush`) le regole presenti vengono cancellate, mentre con `-c` (o `--counters`) si può richiedere il ripristino dei relativi contatori.

Il problema con `iptables-restore` è che esso accetta, come è ovvio, solo un contenuto statico, ed è pertanto impossibile utilizzarlo con una qualche forma di scripting per permettergli ad esempio di usare un indirizzo dinamico. In casi come questi l'unica soluzione possibile è quella di scrivere uno script per modificare direttamente il file su cui si sono salvate le regole da utilizzare, prima di darlo in pasto al comando stesso.

1.4 Criteri per la costruzione di un firewall

Una volta esaminata la struttura del *netfilter* e le funzionalità dei comandi che permettono di utilizzarlo, passiamo a definire i criteri generali per il filtraggio dei pacchetti, e le regole da utilizzare per la realizzazione pratica di un firewall, entrando nei dettagli di come questo poi possa essere realizzato con Linux.

[31] e può servire quando non si vogliono perdere questi valori, che potrebbero essere utilizzati a fini statistici, tutte le volte che si modificano le regole del firewall.

[32] ovviamente, grazie alle normali capacità di redirezione presenti in qualunque shell, è sempre possibile utilizzare un file qualsiasi.

1.4.1 Le funzionalità dirette del kernel

Le regole del *netfilter* non sono il solo componente da usare nella realizzazione di un firewall; oltre al sistema del *netfilter* infatti il kernel fornisce direttamente anche una serie di funzionalità che, pur non essendo direttamente legate al filtraggio dei pacchetti, sono di grande utilità nella realizzazione di un firewall. Queste sono controllate dall'interfaccia del *sysctl*,[33] che permette di impostare i valori di tutta una serie di parametri interni al kernel.

L'accesso a questi parametri è di norma effettuabile attraverso il filesystem `/proc`;[34] infatti tutti i parametri sono visibili (e modificabili) come file all'interno della directory `/proc/sys`. Nel nostro caso quelli che ci interessano sono i parametri relativi alla rete, mantenuti nella sottodirectory `net` ed in particolare quelli usati per controllare il comportamento dello stack TCP/IP del kernel, a loro volta disponibili nell'ulteriore sottodirectory `ipv4`.

In tab. 1.8 si sono riportati, insieme ad una breve spiegazione del loro significato, ed al tipo di valore utilizzato, i principali parametri del kernel relativi allo stack TCP/IP rilevanti per la costruzione di un firewall, modificabili scrivendo direttamente negli omonimi file presenti nella directory `/proc/sys/net/ipv4`. Una descrizione più completa dei parametri presenti in questa directory (relativi alla gestione dello stack TCP/IP) si trova nella documentazione allegata al kernel, nel file `Documentation/networking/ip-sysctl.txt`.

Parametro	Tipo	Significato
`ip_forward`	`bool`	Abilita il kernel all'inoltro dei pacchetti da una interfaccia ad un'altra, di default è disabilitato (valore nullo). Un valore non nullo lo abilita.
`icmp_echo_ignore_all`	`int`	Istruisce il kernel a ignorare tutte le richieste di pacchetti ICMP di tipo *echo reply* dirette alla macchina.
`icmp_echo_ignore_broadcast`	`bool`	Istruisce il kernel a ignorare tutte le richieste di pacchetti ICMP di tipo *echo reply* dirette a indirizzi di broadcast o multicast.
`icmp_ratelimit`	`int`	Istruisce il kernel a limitare il numero di pacchetti ICMP che possono essere inviati per numero. Un valore nullo disabilita la limitazione, un valore positivo indica il numero di pacchetti al millisecondo.
`icmp_ratemask`	`int`	Imposta la maschera binaria dei pacchetti ICMP per i quali vale la limitazione di flusso impostata con il precedente parametro `icmp_ratelimit`. Viene attivata la limitazione per ciascun tipo di pacchetto per cui il bit nella posizione corrispondente al valore del tipo è attivo.
`tcp_syncookies`	`bool`	Abilita la protezione dei *syncookies* nei confronti di attacchi di *syn flood*.

Tabella 1.8: I parametri del kernel che controllano il comportamento dello stack TCP/IP di rilevanza per la costruzione di firewall, con relativo tipo di dato e spiegazione del significato.

In realtà il primo parametro, `ip_forward`, è usato nella costruzione dei firewall solo per attivare l'inoltro dei pacchetti da una interfaccia ad un'altra, una funzionalità che in realtà non

[33] per una trattazione di questa interfaccia si può fare riferimento al capitolo 6 di [**?**], *Guida alla Programmazione in Linux*, disponibile su `http://gapil.gnulinux.it`.

[34] si può usare anche il comando `sysctl`, ma in genere l'interfaccia è più macchinosa; molte distribuzioni però consentono di impostare detti parametri all'avvio attraverso il file `/etc/sysctl.conf`, per i dettagli si possono consultare le relative pagine di manuale.

ha niente a che fare con il filtraggio dei pacchetti, ma che è necessaria quando si usano più interfacce.

Gli altri parametri permettono di bloccare o limitare direttamente (senza l'uso di specifiche regole) alcuni pacchetti ICMP, infine il parametro `tcp_syncookies` richiede un kernel che sia stato compilato con la relativa opzione, ed abilita una protezione specifica contro i cosiddetti *syn flood*, attacchi DoS in cui la propria macchina viene *inondata* di pacchetti di inizio connessione (i SYN appunto). Per bloccare questo tipo di attacchi quando la coda delle connessioni per un socket viene esaurita inizia l'emissione di speciali pacchetti (i cosiddetti *syncookies*) e riprende l'accettazione di ulteriori SYN solo in corrispondenza di una risposta. Questo però modifica il funzionamento usuale del protocollo TCP, ed è da usare solo in caso di emergenza in quanto degrada comunque notevolmente alcuni servizi.

Oltre ai parametri di tab. 1.8, che controllano il comportamento dello stack TCP/IP del kernel nel suo insieme, esistono una serie di altri parametri che permettono di configurarne il comportamento per ciascuna interfaccia. Questi sono accessibili all'interno della directory `/proc/sys/net/ipv4/conf`, che contiene a sua volta delle ulteriori directory (ad esempio `lo`, `eth0`, ecc.) per ciascuna delle interfacce di rete presenti nel sistema, e la directory `default` per indicare l'interfaccia usata per la *default route*.

Oltre a queste è inoltre presente la directory speciale `all` che contiene gli stessi parametri delle altre, in questo caso però essi assumono un significato diverso e possono essere usati o per attivare la relativa funzionalità contemporaneamente su tutte le interfacce, o per abilitare la possibilità di usarla; in quest'ultimo caso occorrerà attivare esplicitamente la funzionalità anche nella directory relative a ciascuna interfaccia (per quelle su cui la si vuole). In tab. 1.9 si sono riportati i parametri rilevanti per la costruzione di un firewall.

Parametro	Tipo	Significato
`forwarding`	`bool`	Abilita il kernel all'inoltro dei pacchetti sull'interfaccia.
`rp_filter`	`bool`	Abilita la protezione *antispoofing* sull'interfaccia.

Tabella 1.9: I parametri del kernel di rilevanza per la costruzione di firewall, che controllano il comportamento dello stack TCP/IP per le singole interfacce.

Il primo parametro può essere usato per restringere il reinstradamento dei pacchetti fra le sole interfacce selezionate attraverso di esso, evitando l'attivazione generale che si ottiene con il precedente `ip_forward`.

Il secondo invece permette di attivare una protezione *antispoofing*[35] direttamente a livello di instradamento. Una interfaccia per cui si attiva questa protezione non farà passare nessun pacchetto che arrivi su di essa che presenti un indirizzo sorgente non corrispondente alla rete su cui l'interfaccia stessa si affaccia.

[35] in questo contesto si parla di *spoofing* quando si falsificano gli indirizzi sorgenti dei pacchetti, per farli apparire come provenienti da altre macchine; in genere questa tecnica viene usata per attacchi di tipo *Distributed Denial of Service* (detti DDoS) in cui si inviano pacchetti di questo tipo, con l'indirizzo sorgente della macchina bersaglio, ad un gran numero di macchine su internet, in modo che la loro risposta *allaghi* il bersaglio, saturando la banda.

1.4.2 Regole generali e politiche di gestione

Prima di entrare nei dettagli relativi all'uso dei vari criteri di selezione e filtraggio dei pacchetti, è opportuno prendere in esame alcune regole generali da applicare nella costruzione di un firewall. Si tratta in sostanza di chiarirsi quale metodo usare per la politica di gestione dei pacchetti, che in sostanza si traduce nella decisione riguardo alla *policy* di default da applicare alle catene (quella che si imposta con l'opzione `-P` di `iptables`).

Esistono di fatto due politiche di gestione, che si riflettono anche nei due possibili valori della *policy* delle catene, quella in cui si lascia passare tutto (`ACCEPT` nel linguaggio di `iptables`) per poi selezionare quello che si vuole bloccare, e quella in cui si blocca tutto di default (`DROP` nel linguaggio di `iptables`) per poi abilitare solo il traffico consentito.

La prima politica ha il vantaggio che funziona tutto di default; ma questo è l'unico vantaggio, infatti richiede una pianificazione attenta delle regole per individuare tutti i possibili traffici anomali che si vogliono bloccare, e che proprio per la loro caratteristica di essere anomali non sono affatto facili da individuare. Il grosso problema di questa politica, che la rende estremamente debole sul piano della sicurezza, è che tutto quello che non avete previsto, ma che un attaccante può inventarsi, passa. Pertanto non la prenderemo ulteriormente in considerazione.

Se invece si usa la politica di bloccare tutto per default, e poi far passare solo il traffico consentito, avremo l'inconveniente che all'inizio non funziona nulla, ma il grande vantaggio che dovremo solo individuare il traffico che ci interessa far passare, dato che tutto il resto sarà comunque bloccato. Questo ovviamente rende impossibile costruire da zero una simile politica da remoto, ma permettere l'accesso da remoto ad un firewall, sia pure per un servizio controllato come SSH, non è comunque una buona idea. Nel caso comunque abbiate questa necessità dovrete per forza partire consentendo almeno il traffico necessario alla connessione di controllo.[36]

Da questo si potrebbe evincere l'indicazione di cambiare la politica di default di tutte le catene a `DROP`, ma in realtà questo non è necessario e comporta il rischio di chiudersi fuori (se per esempio si esegue un reset delle regole con `-F`). Basterà invece mettere come ultima regola di ciascuna catena un `-j DROP`, senza nessun criterio di selezione, così che comunque tutto quello che verrebbe sottoposto alla politica di default (che può restare ad `ACCEPT`) viene bloccato.

In questo modo un firewall ben scritto dovrà contenere soltanto (a parte quella finale) delle regole terminanti con `ACCEPT`, con la sola eccezione (che sarebbe comunque meglio riformulare adeguatamente) in cui si usa `DROP` per escludere preventivamente una selezione di pacchetti fatti passare da un successivo `ACCEPT` troppo ampio.

1.4.3 I criteri di filtraggio per IP

In sez. 1.3.3 abbiamo descritto in dettaglio le varie opzioni di `iptables` che permettono di selezionare pacchetti, il problema adesso è quello di capirne i possibili criteri di applicazione. Cominceremo iniziando a esaminare come utilizzare i criteri di selezione per i pacchetti IP, ma per fare questo occorre prima capire il significato dei vari campi che possono essere filtrati e l'uso che ne viene fatto all'interno del funzionamento del protocollo.

In fig. 1.7 si è riportato lo schema della testata di un pacchetto IP; i due dati fondamentali sono l'IP destinazione, che indica la macchina cui il pacchetto è diretto, e l'IP sorgente, che

[36] il rischio di *chiudersi fuori* in questo caso è molto alto, per cui almeno all'inizio, le prove conviene farle da una console, e non da remoto.

indica la macchina da cui proviene (e a cui pertanto saranno inviate le risposte). Il *netfilter* è in grado di selezionare i pacchetti in base a questi campi (con le opzioni `-d` e `-s` di `iptables`).

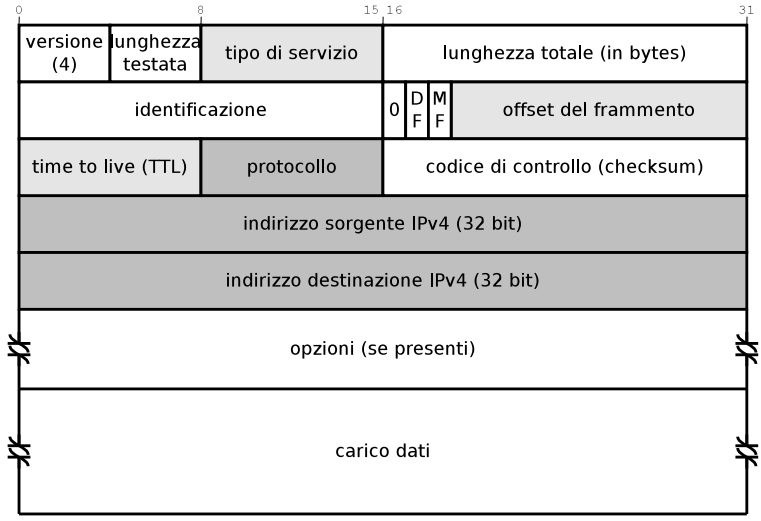


Figura 1.7: Lo schema della testata di un pacchetto IP, con evidenziati i campi controllabili con `iptables`.

Il filtraggio in base ad indirizzi destinazione e sorgente è la funzionalità più basilare di un firewall e permette di selezionare i pacchetti in base alle reti da cui provengono e sono destinati. É con questi criteri di selezione che si possono abilitare o disabilitare gli accessi a reti o singole macchine.

Si tenga presente però che per una qualunque connessione i pacchetti con i quali inviamo dati verso un server avranno come sorgente il nostro IP e destinazione quello del server, mentre quelli di risposta avranno come sorgente l'IP del server (è lui infatti che ce li spedisce) e destinazione il nostro. Pertanto per selezionare tutto il traffico fra due macchine usando solo i valori dei relativi indirizzi, occorrono due regole, una per il traffico in entrata e l'altra per quello in uscita.

Se si vogliono selezionare le connessioni in uscita verso un qualunque indirizzo di destinazione, senza però accettare quelle in ingresso, non si può pertanto usare un criterio basato soltanto sugli indirizzi in gioco,[37] in quanto per accettare il traffico di ritorno si dovrebbe garantire l'accesso da qualunque IP, consentendo con questo anche la possibilità di connessioni in ingresso. Per questa operazione è necessario l'uso del filtro sugli stati (con la sola eccezione del protocollo TCP, che vedremo in sez. 1.4.5).

Il terzo criterio di base per la selezione dei pacchetti (con l'opzione `-p` di `iptables`), è quello sul campo che contiene il numero identificativo del protocollo. Questo campo identifica il contenuto del carico di dati contenuto nel resto del pacchetto, che è sempre un pacchetto di un altro protocollo trasportato da IP. Vedremo in seguito quali altri possibilità di filtraggio si attivano qualora questo protocollo sia uno dei tre protocolli principali (TCP, UDP e ICMP). In generale

[37] il trucco delle due regole vale solo per le connessioni ad una singola macchina, e comunque una regola di questo tipo accetta pacchetti di qualunque connessione verso di noi provenienti da essa, non solo le risposte ad una nostra connessione su di essa.

il *netfilter* permette di selezionare i pacchetti in base a qualunque valore, e questo può essere usato ad esempio per redirigere il traffico relativo a certi protocolli (con il DNAT) a macchine dedicate al loro trattamento.

L'ultimo criterio di base è quello che permette di selezionare i pacchetti frammentati (con l'opzione `-f` di `iptables`). La frammentazione è una caratteristica del protocollo IPv4, che permette ai router di spezzare pacchetti troppo grandi per poter essere trasmessi su un segmento di rete. Se un pacchetto eccede la MTU di un tratto di rete non potrà essere trasmesso integralmente su quel tratto, pertanto il router lo suddividerà in varie parti. Questo significa che il campo dati sarà diviso in varie parti di dimensione opportuna, cui sarà aggiunta una nuova intestazione che contiene nel campo *offset* l'indicazione di quale parte dei dati del pacchetto originale è contenuta.

Questo comporta che tutti i pacchetti generati dallo stesso frammento (eccettuato il primo) non conterranno l'intestazione del protocollo successivo, per cui tali pacchetti non passeranno nessun criterio di selezione basato su valori relativi a questi. Attivando il sistema del *conntrack* i pacchetti vengono deframmentati all'ingresso nel *netfilter*, e quindi il criterio diventa inutile, ma se non lo si usa si possono selezionare tutti i frammenti successivi al primo (sulla base di un valore non nullo del capo *offset*).

Come accennato non si usano quasi mai dei criteri di filtraggio basati solo su IP, l'unica eccezione può essere quella del filtro relativo al *localhost*, in tal caso infatti, dato che tutto il traffico deve provenire ed arrivare dallo stesso indirizzo, non c'è il problema di dover consentire traffico verso il localhost da un indirizzo qualsiasi,[38] e sono perfettamente legittime (e necessarie, se si vuole poterlo usare) regole come:

```
iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

Sebbene i criteri di selezione basati su IP non siano mai usati da soli, sono però quelli usati più comunemente in tutte le altre regole, sia per limitare gli accessi da e per sottoreti definite (in genere in combinazione con il filtro degli stati), che per selezionare i protocolli su cui applicare le ulteriori regole di selezione.

1.4.4 I criteri di filtraggio per ICMP

Il protocollo ICMP, come indica il nome (*Internet Control Message Protocol*) è un protocollo speciale che viene utilizzato per l'invio di messaggi di controllo, e che è fondamentale per il funzionamento di IP. Con `iptables` si possono selezionare i pacchetti ICMP con `-p icmp`. I pacchetti ICMP vengono trasmessi direttamente su IP, il protocollo prevede una intestazione molto semplice il cui formato è illustrato in fig. 1.8.

Il protocollo classifica i messaggi in base ai valori dei due campi *tipo* e *codice*. Ciascun tipo di messaggio ha un suo preciso significato e viene impiegato per un certo compito ben definito. Un elenco dei vari tipi di pacchetti, insieme all'identificativo da passare come argomento all'opzione `--icmp-type` e ad una breve descrizione del significato, si trova in tab. 1.10.

In alcuni casi (la cosa vale soltanto per i pacchetti di tipo *destination unreachable*, *redirect*, *time-exceeded* e *parameter problem*) per lo stesso tipo di pacchetto possono esistere diverse

[38] un tale traffico, tra l'altro, sarebbe oltremodo sospetto.

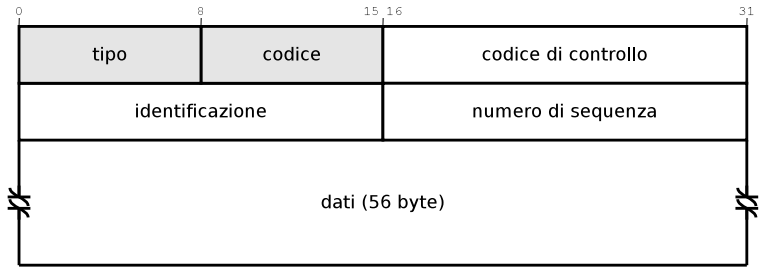


Figura 1.8: Lo schema della testata di un pacchetto ICMP, con evidenziati i campi controllabili con `iptables`.

condizioni di errore o diversi significati che il messaggio deve poter portare, questi allora vengono ulteriormente specificati attraverso un diverso valore del campo *codice*.

La selezione `iptables` consente di scegliere anche questi specifici pacchetti (i valori di tab. 1.10 selezionano solo sulla base del tipo, e il valore del codice viene ignorato), attraverso i valori riportati in tab. 1.11, nella quale si è riportato anche (dividendo la tabella in sezioni diverse per ciascuno dei quattro tipi di pacchetto precedentemente citati), il valore numerico del campo codice.

Valore	**Tipo**	**Significato**
`any`	–	Seleziona tutti i possibili valori
`echo-reply`	0	Inviato in risposta ad un ICMP *echo-request*
`destination-unreachable`	3	Segnala una destinazione irraggiungibile, viene inviato all'IP sorgente di un pacchetto quando un router realizza che questo non può essere inviato a destinazione.
`source-quench`	4	Inviato in caso di congestione della rete per indicare all'IP sorgente di diminuire il traffico inviato.
`redirect`	5	Inviato per segnalare un errore di routing, richiede che la macchina sorgente rediriga il traffico ad un altro router da esso specificato.
`echo-request`	8	Richiede l'invio in risposta di un `echo-reply`.
`time-exceeded`	11	Inviato quando il TTL di un pacchetto viene azzerato.
`parameter-problem`	12	Inviato da un router che rileva dei problemi con l'intestazione di un pacchetto.
`timestamp-request`	13	Richiede l'invio in risposta di un `timestamp-reply`.
`timestamp-reply`	14	inviato in risposta di un `timestamp-request`.
`info-request`	15	Richiede l'invio in risposta di un `info-reply`.
`info-reply`	16	Inviato in risposta di un `info-request`.
`address-mask-request`	17	Richiede l'invio in risposta di un `address-mask-reply`.
`address-mask-reply`	18	Inviato in risposta di un `address-mask-request`.

Tabella 1.10: Tipi di pacchetti ICMP selezionabili con `--icmp-type`.

In sez. 1.2.3 abbiamo visto come alcuni pacchetti ICMP, *echo request*, *netmask request*, *timestamp request* e *information request* (ed i corrispondenti pacchetti di risposta) vengano classificati dal sistema del *conntrack*. Di questi le ultime tre coppie riguardano delle funzionalità poco usate, e possono essere tranquillamente ignorati e venire bloccati da un firewall. Tra l'altro la risposta ad un *timestamp request* permette di conoscere il valore dell'orologio di sistema di una macchina, e questo può essere utile in certi attacchi.

Valore	Codice
network-unreachable	0
host-unreachable	1
protocol-unreachable	2
port-unreachable	3
fragmentation-needed	4
source-route-failed	5
network-unknown	6
host-unknown	7
host-isolated	8
network-prohibited	9
host-prohibited	10
TOS-network-unreachable	11
TOS-host-unreachable	12
communication-prohibited	13
host-precedence-violation	14
precedence-cutoff	15
network-redirect	0
host-redirect	1
TOS-network-redirect	2
TOS-host-redirect	3
ttl-zero-during-transit	0
ttl-zero-during-reassembly	1
ip-header-bad	0
required-option-missing	1

Tabella 1.11: Specifici pacchetti ICMP classificati in base ai valori del campo *codice*, suddivisi per tipo: *destination unreachable*, *redirect*, *time-exceeded* e *parameter problem*.

I pacchetti *echo request* e *echo reply* costituiscono invece la base del comando `ping` e sono molto utilizzati per scopi diagnostici, per questo non è il caso di scartarli a priori. Il problema è che molti attacchi di tipo *Distributed Denial of Service* si basano appunto sull'uso di questi pacchetti. In genere il trucco è quello di inviare (da macchine compromesse) una serie di *echo request* falsificando l'indirizzo sorgente in quello della macchina da attaccare. Questo comporta che essa riceverà tutte le risposte. Se si aggiunge che in molti casi i router lasciano passare degli *echo request* in *broadcast*, questo consente un grosso effetto moltiplicativo (un solo pacchetto scatena la risposta di tutta una rete) che conduce in genere ad una *alluvione* di *echo reply* (questo tipo di attacchi infatti viene spesso chiamato *flood*) sulla macchina vittima dell'attacco.

Per questo motivo alcuni amministratori poco competenti tendono a filtrare completamente tutto ICMP, invece di provvedere a risolvere il problema seriamente. In tal caso infatti quello che va fatto è bloccare le risposte a richieste in *broadcast* (questo può essere fatto con Linux con i parametri `icmp-ignore-broadcast` del kernel illustrati in sez. 1.4.1), e inserire (con l'estensione `limit`) un controllo sul flusso massimo di questo tipo di pacchetti.

Inoltre per evitare di essere usati come *complici* di un tale attacco occorre abilitare sui *router di frontiera*[39] la protezione *antispoofing* (sempre descritta in sez. 1.4.1, con il parametro `rp_filter`) che blocca i pacchetti uscenti da una interfaccia il cui indirizzo sorgente non corrisponda alla rete che sta su quell'interfaccia.

[39] si chiamano così (*border router*) i router che stanno ai bordi di Internet, che hanno cioè dietro di loro dei tratti di rete chiusi, dai quali non si rientra in Internet; in sostanza quelli che non stanno sulle dorsali.

Una volta fatto questo occorre poi compiere una scelta relativa ai pacchetti ICMP che si vogliono far passare, tenendo presente che alcuni di essi sono importanti per il buon funzionamento della rete, e che bloccarli tutti danneggia anzitutto noi stessi.

Un pacchetto pericoloso, e da filtrare senz'altro, è il `redirect`, in quanto esso comporta la possibilità di riscrivere da remoto la tabella di routing di una macchina modificando la rotta per la destinazione dei pacchetti, con possibilità enormi di attacchi di tipo DoS. Il pacchetto infatti veniva utilizzato quando un router identificava una rotta migliore per un pacchetto arrivato a lui, con esso si indicava alla macchina sorgente di redirigere il traffico direttamente sulla nuova rotta; il problema è che un pacchetto appositamente forgiato potrebbe ad esempio farvi redirigere il traffico sul localhost, con effetti tutt'altro che gradevoli per la vostra rete.

Lo stesso problema, anche se in maniera molto minore, può derivare dall'accettare i pacchetti `source-quench` che vengono usati dai router per comunicare uno stato di congestione della linea, richiedendo appunto uno *smorzamento* del traffico. In genere la loro ricezione fa eseguire al kernel un rallentamento del traffico in uscita verso quella destinazione, in modo da diminuire la congestione. Di nuovo un uso *malizioso* potrebbe portarvi a ridurre il traffico anche quando non ce n'è bisogno, ma il problema è senz'altro minore rispetto al `redirect`.

I pacchetti `time-exceeded` e `destination-unreachable` sono invece pacchetti utili alla rilevazione di problemi sulla rete, e sono di norma da accettare, è in particolare molto importante accettare il pacchetto `fragmentation-needed` in quanto viene usato per la determinazione della MTU (la dimensione massima) dei pacchetti da inviare su una connessione, la cosiddetta *Path MTU*.

Come accennato in sez. 1.4.3 quando un router individua un pacchetto la cui dimensione eccede la capacità del tratto di rete su cui deve inviarlo, lo frammenta. Questa operazione rallenta sia le operazioni del router, e aumenta (per la necessità di header aggiuntivi) il consumo di banda, nuocendo alla velocità di trasmissione; dato inoltre che la *Path MTU* può variare, è stato previsto il meccanismo del *Path MTU Discovery* (descritto nell'RFC 1191) che permette di riconoscere quale è il valore massimo delle dimensioni di un pacchetto da inviare su una collegamento, senza che questo debba essere frammentato. Il meccanismo sfrutta il flag *Don't Fragment* del protocollo IP (vedi fig. 1.7) che richiede al router di non eseguire la frammentazione di pacchetti di dimensione eccessiva, ma di generare in ritorno un pacchetto ICMP di tipo `destination-unreachable` e codice `fragmentation-needed`, ricevendo il quale si può riconoscere quando un pacchetto ha una dimensione eccessiva e determinare quella corretta.

Un elenco tipico di regole riguardanti ICMP è pertanto il seguente:

```
iptables -A INPUT -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -p icmp --icmp-type fragmentation-needed -j ACCEPT
```

1.4.5 I criteri di filtraggio per TCP

Il protocollo TCP è il più utilizzato dei protocolli trasportati su IP, ed è l'unico dei protocolli della suite TCP/IP che supporta nativamente connessioni e stati. Realizzare tutto questo però ha un suo costo, e come si può vedere dalla quanto mostrato in fig. 1.9, la testata dei pacchetti TCP è una delle più complesse.

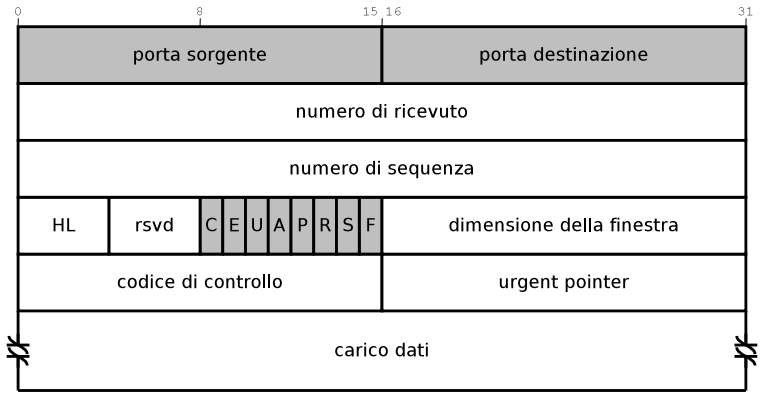


Figura 1.9: Lo schema della testata di un pacchetto TCP, con evidenziati i campi controllabili con `iptables`.

La caratteristica principale di TCP, comune con UDP come protocollo di trasporto, è la presenza di due campi di 16 bit usati per mantenere il numero di porta (sorgente e destinazione), che vengono utilizzati per distinguere, fra tutti i pacchetti che transitano fra due macchine, quelli appartenenti ad una specifica connessione. Come abbiamo visto queste possono essere selezionate attraverso l'uso delle opzioni `--sport` e `--dport`.

Usare solo i valori delle porte per identificare una connessione non è però pratico. Come abbiamo visto in sez. 1.2.3 infatti le due direzioni del flusso di dati all'interno di una connessione hanno destinazione e sorgente invertita. Allora anche se è possibile selezionare tutti i pacchetti di una connessione qualora siano note le porte (destinazione e sorgente) usate, il problema sorge dal fatto che queste non sono determinabili a priori. Infatti in una connessione solo una porta è sempre ben stabilita (quella relativa al servizio cui ci si vuole collegare), l'altra, che viene detta per questo *effimera*, viene scelta dal kernel della macchina di chi inizia la connessione con un valore dinamico, preso fra quelli disponibili, e può essere qualunque. Pertanto, se si vuole ricevere in maniera generica il traffico di ritorno di una connessione, si dovrà consentire l'accesso a tutto un insieme di porte, e benché questo sia possibile, è poco pratico (e fa passare qualunque pacchetto proveniente dalla stessa origine, anche se non collegato a nessuna connessione).

Per questo l'uso classico della selezione sulle porte TCP è quello sulla porta destinazione combinato con il filtro degli stati; questo permetterà di selezionare per quali porte di destinazione si accettano pacchetti in stato `NEW`, consentendo di effettuare connessioni verso di esse, sia in ingresso (quali server locali possono essere contattati dall'esterno) che in uscita (a quali servizi esterni si consente l'accesso).

In generale perciò i criteri di filtraggio basati su TCP si riducono sostanzialmente alla scelta dei servizi cui di vuole dare accesso. È usuale consentire l'accesso via SSH alle macchine sulla DMZ, purtroppo spesso questo viene fatto anche per il firewall nonostante il rischio che tutto ciò comporta. In genere, oltre a SSH, i servizi più comuni sono il web, e la posta elettronica, che vanno posti nella DMZ. Un elenco tipico di regole che consentono l'accesso a questi servizi su un server è il seguente:

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -m state --state NEW -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 80 -m state --state NEW -j ACCEPT
```

In genere è da evitare di fornire accesso ad eventuali database (anche se posti nella DMZ) dall'esterno, limitando questo solo alle macchine nella stessa rete. Inoltre è assolutamente da escludere l'accesso a servizi tipici di un rete interna, come i file server (Samba) o i servizi per le autenticazioni centralizzate (LDAP o NIS). Qualora alcuni dati di questi servizi dovessero risultare utili per le macchine presenti in DMZ si dovrà piuttosto predisporre un meccanismo di replicazione delle informazioni, (e solo di quelle indispensabili), a partire dalla rete interna e verso l'esterna (e mai il viceversa).

Per quanto riguarda l'accesso verso l'esterno tutto dipende da eventuali politiche di restrizione degli accessi che si intendono implementare, ad esempio si può consentire soltanto l'accesso a web e posta bloccando tutto il resto. In genere poi, per alcuni servizi, si può provvedere alla impostazione di un *proxy trasparente*, utilizzando le capacità di redirezione del *netfilter*.

1.4.6 I criteri di filtraggio per UDP

Il secondo protocollo più utilizzato su internet dopo il TCP è UDP, questo protocollo non è altro che un semplice involucro (un *wrapper*) per utilizzare le caratteristiche di IP a livello di trasporto. Pertanto l'intestazione del protocollo, illustrata in fig. 1.10, è estremamente semplice, e si limita ad aggiungere ad IP i campi relativi alle porte (per poter supportare diversi canali di comunicazione) e due campi ausiliari di controllo (la lunghezza del pacchetto e la somma di controllo).

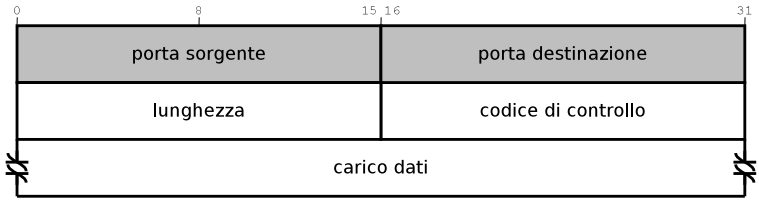


Figura 1.10: Lo schema della testata di un pacchetto UDP, con evidenziati i campi controllabili con `iptables`.

Benché nel caso di UDP non esistano connessioni (il protocollo si limita a garantire solo il massimo sforzo nella consegna dei dati) si possono comunque, come fa il filtro degli stati, identificare dei canali di comunicazione; questi hanno (in termini di porte coinvolte e flussi dei relativi pacchetti) la stessa caratteristiche di una connessione TCP, e pertanto valgono anche in questo caso le considerazioni precedenti relative al filtraggio in base al valore della porta di destinazione.

Come per il TCP si tratta di selezionare quali sono i servizi a cui si vuole consentire l'accesso dall'esterno, il principale servizio che opera su UDP è il DNS, che di nuovo deve essere posto in DMZ; un altro servizio importante che utilizza UDP è NFS, ma come per Samba (e qualunque altro meccanismo di condivisione dei file della rete interna) esso non deve mai reso visibile all'esterno. Un esempio di regole utilizzate per UDP è il seguente:

```
iptables -A INPUT -p udp --dport 53 -m state --state NEW -j ACCEPT
iptables -A INPUT -p udp --dport 500 -m state --state NEW -j ACCEPT
```

che forniscono accesso remoto al DNS ed al servizio IKE per IPSEC (che vedremo in sez. 2.1.2).

1.4.7 Un esempio di firewall

Vedremo adesso una serie di regole che possono essere utilizzate per costruire un firewall elementare; si è fatto la scelta di non creare e commentare uno script già pronto, perché lo scopo è quello di mettere in grado le persone di scrivere il proprio. Per questo useremo volutamente esempi particolari (senza definire variabili al posto delle varie interfacce e indirizzi) e regole spesso incomplete, in modo che ciascuno, una volta compresi i concetti che le sottendono, debba a sua volta creare un suo script.

Negli esempi usati per illustrare i criteri di filtraggio sui vari protocolli si è sempre fatto riferimento ad una singola macchina presa come origine o destinazione finale dei pacchetti, operando quindi sulle catene di `INPUT` e `OUTPUT`; nel caso si debba installare un firewall però si tratterà di operare principalmente sulla catena di `FORWARD`, dovendo filtrare i pacchetti in transito fra diverse interfacce di rete.

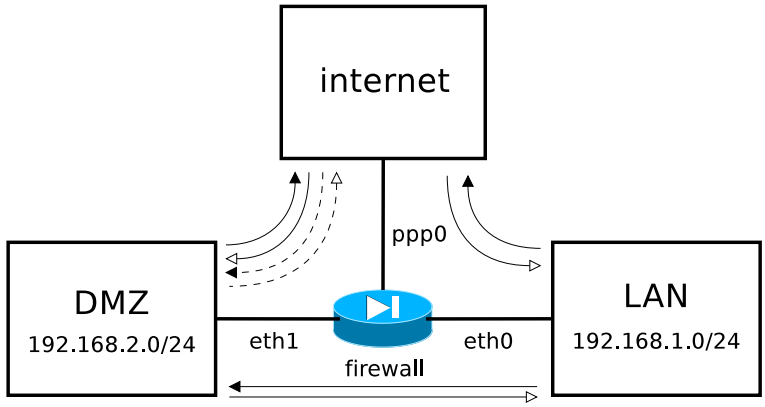


Figura 1.11: Schema delle interfacce per un firewall di esempio nella configurazione di base con DMZ, con illustrazione dei flussi di dati.

Il caso più comune, utilizzando una topologia come quella di fig. 1.2, è quello di consentire l'accesso ad internet dalla rete interna senza restrizioni, di impedire l'accesso da internet alla rete interna e di consentire l'accesso dall'esterno alla DMZ solo per i servizi pubblici che si sono posti su di essa.

Come esempio particolare consideriamo la disposizione delle interfacce e delle sottoreti illustrata in fig. 1.11, in cui si sono anche riportati i flussi dei pacchetti da consentire secondo quanto detto in sez. 1.1.3. In figura si è indicato un flusso di dati che permette la connessione con una freccia piena ed il flusso dei pacchetti di ritorno con la freccia vuota; inoltre si è usata una linea continua per indicare un flusso generico non filtrato e una discontinua per quello consentito solo per alcuni servizi.

Il primo passo è allora quello di definire quali pacchetti vogliamo comunque far passare. Useremo il filtro degli stati, per cui in generale questi saranno tutti quelli in stato `ESTABLISHED` o `RELATED`, in quanto risposte a connessioni consentite, più i pacchetti ICMP di controllo di

cui abbiamo parlato in sez. 1.4.4; metteremo il tutto su una catena `allowed` utilizzando una definizione del tipo:

firewall.sh
```
iptables -N allowed

iptables -A allowed -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A allowed -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A allowed -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A allowed -p icmp --icmp-type echo-request -j ACCEPT
#iptables -A allowed -p icmp --icmp-type fragmentation-needed -j ACCEPT
```

dove prima si crea la catena, e poi vi si inseriscono le regole per i pacchetti appena citati.

Attenendoci rigidamente al criterio per cui non deve essere possibile arrivare da internet alla rete interna, bloccheremo anche ogni accesso diretto al firewall dall'esterno. La soluzione più sicura è quella di bloccare qualunque connessione remota diretta al firewall, per comodità si può sacrificare un po' di sicurezza concedendo un accesso limitato solo dalla rete interna. Questo significa che per quanto riguarda le due catene di `INPUT` e `OUTPUT` (che si ricordi riguardano solo i pacchetti diretti o provenienti dal firewall) basteranno una serie di regole del tipo:

firewall.sh
```
#
# INPUT
#
iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A INPUT -j allowed
# iptables -A INPUT -i eth0 -s 192.168.1.0/24 --state NEW -j ACCEPT
iptables -A INPUT -j DROP
#
# OUTPUT
#
iptables -A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
iptables -A OUTPUT -m state --state NEW -j ACCEPT
iptables -A OUTPUT -j allowed
iptables -A OUTPUT -j DROP
```

Nell'esempio si abilita sia in ingresso che in uscita tutto il traffico sul `localhost`, e si accettano i pacchetti delle connessioni già stabilite e quelli ICMP di controllo secondo quanto già specificato nella catena `allowed`; poi si permettono le nuove connessioni in uscita dal server. Volendo abilitare delle connessioni in ingresso si può ad esempio scommentare la riga in cui si abilitano le connessioni dirette al firewall solo se provenienti dalla corretta interfaccia e dalla relativa sottorete (sarebbe opportuno comunque essere più restrittivi e limitarle sulla base di singoli indirizzi, ad esempio quello della macchina dell'amministratore).

Una volta definito cosa fare dei pacchetti destinati direttamente alla macchina che fa da firewall la configurazione di quest'ultimo riguarda sostanzialmente i pacchetti in transito, e cioè la catena di `FORWARD`. Per consentire i flussi illustrati in fig. 1.11 si dovranno definire delle regole del tipo:

firewall.sh
```
iptables -A FORWARD -j allowed
iptables -A FORWARD -i eth0 -o ppp0 -m state --state NEW -j ACCEPT
```

```
iptables -A FORWARD -i eth0 -o eth1 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i eth1 -o ppp0 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i ppp0 -o eth1 -m state --state NEW -j services
iptables -A FORWARD -j DROP
```

In questo caso prima si fanno passare i soliti pacchetti sempre consentiti tramite la catena `allowed`, poi si abilita l'accesso a partire dalla LAN sia per la DMZ che per internet; volendo essere pignoli, il che quando si tratta di sicurezza è sempre un pregio, occorrerebbe specificare anche le reti di origine e quelle di destinazione, lo lasciamo per esercizio. Si dà poi accesso ad internet dalla DMZ, ed infine per le connessioni provenienti da internet e dirette sulla DMZ si userà una catena `services` opportunamente creata.

Il grosso del lavoro di filtraggio è da fare su `services`, qui si dovranno definire a quali servizi e su quali server si potrà accedere da internet; di nuovo prendiamo un esempio in cui nella DMZ ci sono sulla solo due macchine, la `192.168.2.2` che fa da server di posta e DNS, e la `192.168.2.3` che fa da server web, in questo caso potremo usare delle regole del tipo:

firewall.sh

```
iptables -N services
iptables -A services -d 192.168.2.2 -p tcp --dport 25  -j ACCEPT
iptables -A services -d 192.168.2.2 -p udp --dport 53  -j ACCEPT
iptables -A services -d 192.168.2.3 -p tcp --dport 80  -j ACCEPT
```

Fin qui si sono trattate solo le regole destinate alla gestione del firewall, ma avendo usato nell'esempio un caso di reti private sia per la DMZ che per la LAN queste non sono in realtà sufficienti. L'ipotesi dell'esempio di fig. 1.11 è allora quella che il firewall faccia anche da router per una connessione con un IP pubblico solo per `ppp0`.[40] In questo caso è necessario usare le funzionalità della tabella di `nat` per compiere le opportune trasformazioni degli indirizzi.

Il primo passo è quello di eseguire un *Source NAT* per i pacchetti uscenti su internet in modo che questi assumano l'indirizzo pubblico del nostro *router/firewall*; questo si ottiene andando ad operare sulla catena di `POSTROUTING` della tabella di `nat` con una regola del tipo:

firewall.sh

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

che esegue il *Source NAT* su tutti i pacchetti uscenti su `ppp0`.

Si tenga presente che `MASQUERADE` è una azione speciale, che esegue il *Source NAT* andando a vedere quale è l'IP assegnato sulla relativa interfaccia; per questo in genere la si usa solo per le interfacce su cui l'IP è assegnato dinamicamente, dato che è in grado di riconoscere al volo un eventuale cambiamento dello stesso (ed è quindi tipica delle connessioni *dial-in*). Questo processo comporta un piccolo *overhead* per la determinazione dell'indirizzo, per cui se l'IP è statico è più opportuno usare direttamente il *Source NAT* su quest'ultimo, con qualcosa del tipo:

firewall.sh

```
iptables -t nat -A POSTROUTING -o ppp0 -j SNAT --to-source 62.94.208.119
```

[40] è il caso tipico di una connessione via modem, sia su linea telefonica analogica, che via ADSL.

Il passo successivo è quello di redirigere le connessioni provenienti da internet per i servizi cui si vuole dare accesso sui rispettivi server nella DMZ; questo significa dover eseguire un *Destination NAT* relativo a detti servizi, cosa che si otterrà operando sulla catena di `PREROUTING` della tabella di `nat` con un qualcosa del tipo:

firewall.sh
```
iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 80 -j DNAT \
                --to-destination 192.168.2.3
iptables -t nat -A PREROUTING -i ppp0 -p tcp --dport 25 -j DNAT \
                --to-destination 192.168.2.2
iptables -t nat -A PREROUTING -i ppp0 -p udp --dport 53 -j DNAT \
                --to-destination 192.168.2.2
```

Si noti come in questo caso si sia utilizzata la selezione sull'interfaccia di provenienza per identificare le connessioni entranti da internet, e selezionati i pacchetti solo sulla base della porta di destinazione. Questo perché nell'esempio in questione c'è un unico indirizzo pubblico possibile, quello associato con `ppp0`. Nel caso invece si abbiano più indirizzi esterni, con una configurazione in cui la macchina fa solo da firewall (come in fig. 1.2 e non anche da router per la connessione, come in fig. 1.11) e si vogliano usare per i server nella DMZ degli IP diversi,[41] bisognerà selezionare i pacchetti in arrivo anche in base agli indirizzi pubblici cui sono rivolti e non solo in base all'interfaccia.

Infine si noti come in tutti gli esempi precedenti le regole di firewall sono state espresse usando direttamente indirizzi presenti sulle reti locali, anche per i servizi pubblici messi nella DMZ; questo è corretto perché la traslazione degli indirizzi avviene sulle due catene di `PREROUTING` e `POSTROUTING`, cioè rispettivamente prima dell'entrata e dopo l'uscita dal firewall e pertanto a livello della tabella di `filter` si può senza problemi fare riferimento agli indirizzi come li si vedono all'interno della propria rete locale.

[41] una tecnica usata comunemente è quella di assegnare comunque tutti gli IP pubblici (con l'*IP aliasing*) all'interfaccia esterna del firewall, e poi eseguire comunque un DNAT su altrettanti diversi IP privati, messi all'interno della DMZ.

Capitolo 2

Virtual Private Network

2.1 Cenni di teoria delle VPN

Prima di parlare della implementazione pratica e delle modalità di installazione, uso e configurazione delle VPN, faremo una introduzione su cosa sono le VPN, quali caratteristiche hanno, cosa le contraddistingue, e sulle basi teoriche (algoritmi e protocolli) sulle quali queste vengono costruite.

2.1.1 Cos'è una VPN

Con la sigla VPN (*Virtual Private Network*) si intende in realtà una classe di tecnologie che permettono di mettere in comunicazione appunto delle *reti private* attraverso internet. Con reti private si intendono in genere reti non accessibili direttamente da internet, che di norma utilizzano gli indirizzi non instradabili dell'RFC 1918,[1] e condividono la connessione ad internet attraverso un router che effettua il mascheramento di detti indirizzi.[2]

Dato che questi indirizzi non vengono instradati dai router è ovviamente impossibile collegare direttamente attraverso Internet due reti di questo tipo, anche se non usano gli stessi indirizzi. Se però si trovasse il modo di far arrivare in una qualche maniera i pacchetti che le due reti vogliono scambiarsi ai due relativi router, questi non avrebbero alcun problema a consegnarli ai relativi destinatari.

Una VPN è questo canale di comunicazione alternativa che permette di trasmettere i pacchetti fra due router secondo lo schema di fig. 2.1, canale che di norma, come misura di ulteriore sicurezza, viene opportunamente cifrato in modo che nel transito da internet il traffico trasmesso non possa essere osservato; una caratteristica che rafforza l'uso del nome di *private*.

In generale non è necessario, per avere una VPN, che i due tratti di rete agli estremi della stessa usino indirizzi privati; e di per sé per collegare due reti private non è necessario che il canale di comunicazione sia cifrato, ma questa è una delle caratteristiche più tipiche di una

[1] cioè gli indirizzi della rete di classe A `10.0.0.0/8`, delle 16 reti di classe B da `172.16.0.0/16` a `172.31.0.0/16` e delle 256 reti di classe C da `192.168.0.0/24` a `192.168.255.0/24`.

[2] abbiamo visto in sez. 1.3.5 come eseguire questa operazione con il *netfilter*, attraverso il target `SNAT` nella tabella di `nat`.

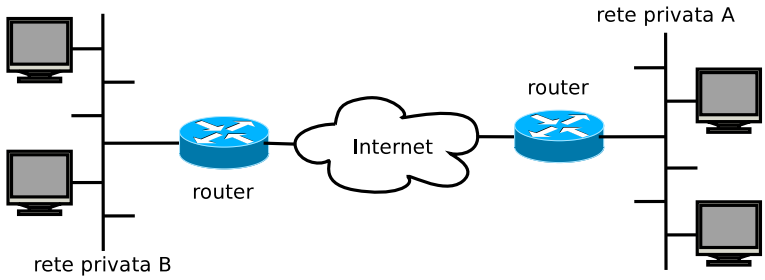


Figura 2.1: Lo schema generico di una VPN che collega due reti private.

VPN, senza la quale si avrebbe un semplice *tunnel*. Per questo normalmente il cuore di una VPN è un algoritmo crittografico che costituisce la base su cui è possibile costruire un canale di comunicazione sicuro attraverso cui passeranno i nostri dati.

Inoltre quello di fig. 2.1 è solo lo schema più comune di utilizzo di una VPN, in realtà si possono usare le VPN anche dare accesso ad una singola macchina (nella configurazione detta *road warrior*) all'interno di una rete altrimenti invisibile dall'esterno, o per mettere in contatto due macchine singole (configurazione limite della precedente, in cui la rete si riduce ad una sola macchina).

Infine è sempre possibile, usando più canali, mettere in comunicazione fra loro un numero arbitrario di reti, purché ovviamente queste mantengano indirizzi non in conflitto fra di loro. In questo caso avremo ovviamente una struttura più complessa, anche se alla base resta un incrocio di tante configurazioni del tipo di quella mostrata in fig. 2.1.

2.1.2 Il protocollo IPSEC

Benché spesso si preferisca utilizzare altre soluzioni (ed in particolare tunnel basati sui protocolli di trasporto, normalmente UDP) dal punto di vista della standardizzazione la soluzione più appropriata per realizzare delle VPN è quella di usare IPSEC, cioè una reimplementazione del protocollo IP, supportata nativamente in IPv6, ed incapsulata in IPv4, che fornisce autenticazione e crittografia direttamente al livello di rete (appunto quello di IP).

L'uso di IPSEC al posto di IP permette infatti, fra le altre cose, di cifrare completamente tutto il traffico fra due macchine su internet, risolvendo così in maniera naturale il problema di comunicare in maniera privata. Dato poi che il protocollo supporta una comunicazione in modalità tunnel fra due router che cifra pure gli indirizzi iniziale e finale dei pacchetti, è altrettanto naturale ottenere una VPN. Il protocollo IPSEC è basato su tre componenti:

AH definito nell'RFC 2405, è l'acronimo di *Authentication Header*, e fornisce un servizio di autenticazione dei pacchetti.

ESP definito nell'RFC 2406, è l'acronimo di *Encapsulating Security Payload*, e fornisce un servizio di autenticazione e cifratura dei dati.

IKE definito nell'RFC 2409 per la versione originaria (IKEv1) ed aggiornato con una seconda versione (IKEv2), definita nell'RFC 5996. è l'acronimo di *Internet Key Exchange*, e

fornisce un servizio per la negoziazione dei parametri necessari al funzionamento dei precedenti.

Di queste tre componenti quella di gran lunga più complessa è IKE, che nasce dalla combinazione di una serie di altri protocolli. In sostanza IKE combina il protocollo ISAKMP (*Internet Security Association and Key Management Protocol* definito dall'RFC 2408) che gestisce la negoziazione delle connessioni e definisce le cosiddette *Security Association*. A questo si aggiunge il DOI (*Domain Of Interpretation*, definito dall'RFC 2407) per IPSEC che completa ISAKMP nel caso specifico di IPSEC, e il protocollo *Oakley* di scelta delle chiavi (definito nell'RFC 2412), che è quello che permette l'impostazione iniziale dei parametri necessari alla comunicazione. Il protocollo è stato in seguito rivisto con la creazione di una seconda versione, la cui specificazione finale (che riprende la definizione originale dell'RFC 4306 ed alcuni chiarimenti successivi dell'RFC 7418) è fornita nell'RFC 5996.

È a questo livello che vengono negoziate le chiavi e gli algoritmi crittografici utilizzati, i tempi di vita delle connessioni, implementando in pratica quel meccanismo di crittografia ibrido descritto alla fine di sez. **??**.

Nel funzionamento di IPSEC giocano un ruolo cruciale le *Security Association* (in breve SA), queste identificano un canale di comunicazione definendone destinazione, protocollo utilizzato (AH o ESP) ed un *Security Parameter Index* (SPI) che distingue tutti i possibili diversi canali che hanno la stessa destinazione e usano lo stesso protocollo. Per quanto abbiamo appena detto è evidente che una SA è unidirezionale (è infatti sempre definita a partire dalla macchina corrente verso una certa destinazione) per cui ne occorrono sempre due (una per direzione) per ogni canale di comunicazione.

Il primo passo per poter utilizzare IPSEC è quello di eseguire IKE per la negoziazione dei parametri della connessione, questo in genere avviene in due fasi; nella prima fase viene negoziata una coppia di SA di ISAKMP per la comunicazione fra due *gateway*, in modo da poter gestire più canali indipendenti; con queste nella seconda fase vengono generate le ulteriori SA per IPSEC (ad uso di ESP o AH) per le singole comunicazioni. Il tutto avviene usando il protocollo UDP sulla porta 500. Pertanto è essenziale al funzionamento di IPSEC che un firewall lasci passare il traffico su questa porta.

IKE è progettato per la massima flessibilità e permette di negoziare separatamente per ciascuna SA un tempo di vita, l'algoritmo crittografico (simmetrico) da utilizzare per la cifratura dei pacchetti, l'algoritmo per l'autenticazione, e l'algoritmo di crittografia asimmetrica usato per lo scambio delle chiavi. Con la versione 2, oltre a vari miglioramenti relativi ai meccanismi di negoziazione, all'affidabilità delle connessione, e alla resistenza ad attacchi di DoS, è stato inserito all'interno del protocollo la gestione del cosiddetto *NAT Traversal* (su cui torneremo più avanti) con l'incapsulazione di sia di IKE che ESP in pacchetti UDP, per i quali è stata riservata la porta 4500. Se pertanto si utilizza IKEv2 è necessario che il firewall consenta il traffico anche su questa porta.

Una volta completata l'impostazione iniziale tramite IKE, tutto il resto della comunicazione avviene direttamente attraverso quello che più propriamente si indica con IPSEC, facendo riferimento solo ai due protocolli AH ed ESP, che effettivamente operano a livello di rete. Infatti IKE è un protocollo di livello superiore che serve a negoziare i parametri usati dagli altri due, l'effettiva comunicazione non avviene attraverso di esso. Da qui in avanti perciò riferendoci ad IPSEC intenderemo soltanto la parte relativa alla comunicazione al livello di rete.

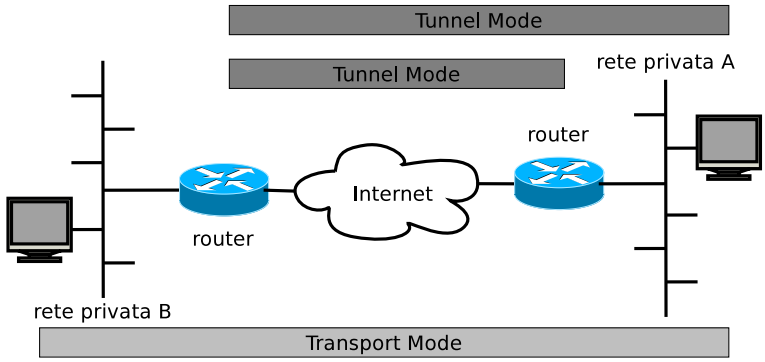


Figura 2.2: Modalità di trasporto dei dati realizzate tramite IPSEC.

Che sia incapsulato in IPv4, o implementato come estensione di IPv6, IPSEC supporta due modalità native, illustrate in fig. 2.2. La prima, detta modalità di trasporto (o *Transport Mode*) prevede la comunicazione diretta da stazione a stazione, ciascuna delle quali tratta i suoi pacchetti e li spedisce tramite IPSEC. La seconda modalità, detta tunnel (o *Tunnel Mode*) prevede invece la presenza di almeno un router (il cosiddetto *security gateway*) che faccia da tramite per la comunicazione con le macchine poste nella rete per la quale agisce appunto da *gateway*.

La modalità tunnel è appunto la modalità tipica con cui si realizza una VPN, e presenta le due possibilità mostrate nella parte alta di fig. 2.2, in cui si può realizzare la VPN per comunicare da una macchina esterna (il *road warrior*) verso il *security gateway* per accedere alla rete dietro di questi, o direttamente fra due *security gateway*, per connettere due reti private.

In generale IPSEC supporta due tipi di comunicazione, autenticata o cifrata, che possono essere usate indipendentemente l'una dall'altra. Però l'uso di una comunicazione cifrata non autenticata è insicuro e soggetto ad attacchi che possono permettere di decifrare i messaggi, per cui non viene mai usata. La comunicazione autenticata avviene attraverso l'uso di AH, mentre la comunicazione cifrata attraverso ESP, che supporta anche, ove richiesto (e come detto questo deve essere sempre fatto) l'autenticazione.

Dei due protocolli usati per la comunicazione effettiva AH serve solo ad autenticare i pacchetti; il meccanismo di funzionamento di AH, nei due casi di modalità trasporto o tunnel, è illustrato in fig. 2.3. Nel caso si usi la modalità trasporto ci si limita a frapporre fra l'intestazione di IP e quella dei protocolli di livello superiore (nell'esempio TCP) l'intestazione di AH, che contiene tutti i dati relativi all'autenticazione.

Se invece si opera in modalità tunnel si crea una nuova intestazione per il pacchetto, che conterrà solo gli indirizzi degli estremi del tunnel, e si autentica tutto il pacchetto originale, replicandolo integralmente dopo l'intestazione di AH. Alla ricezione del pacchetto il router che fa da *security gateway* rimuoverà l'intestazione usata per la trasmissione, verificherà il contenuto e ricostruirà il pacchetto originale, inviandolo a destinazione (in genere nella rete privata posta dietro il router stesso). In questo modo si possono inviare pacchetti attraverso internet anche usando gli indirizzi riservati per le reti private illustrati in sez. 2.1.1.

La procedura di autenticazione cerca di garantire l'autenticità del pacchetto nella massima

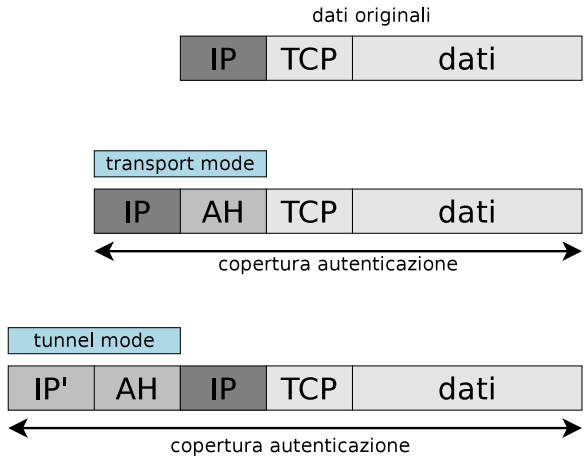


Figura 2.3: Schema di un pacchetto IPSEC autenticato l'uso di AH.

estensione possibile, ma dato che alcuni campi dell'intestazione di IP possono variare in maniera impredicibile alla sorgente, il loro valore non può essere protetto dall'autenticazione. Il calcolo dei dati di autenticazione viene allora effettuato alla sorgente su una versione speciale del pacchetto inviato in cui il TTL nell'intestazione principale è impostato a zero, così come le opzioni che possono essere modificate nella trasmissione. La copertura dell'autenticazione del pacchetto è mostrata sempre in fig. 2.3.

Qualora si debba garantire una trasmissione riservata dei dati, occorre invece usare ESP, lo schema di un pacchetto secondo questo protocollo è illustrato in fig. 2.4; di nuovo se si opera in modalità di trasporto ci si limita a cifrare il contenuto del pacchetto IP, interponendo il risultato fra l'intestazione di ESP ed una coda che chiude la sezione cifrata; a questo si aggiunge poi un'eventuale altra sezione finale che contiene le informazioni di autenticazione; le sezioni del pacchetto coperte da autenticazione e crittografia sono sempre riportate in fig.2.4.

Se invece si opera in modalità tunnel come nel caso di AH si usa una nuova intestazione per IP, cifrando completamente tutto il pacchetto originale. In questo modo non solo si potranno usare nel pacchetto originale indirizzi di reti private, ma in generale si potrà impedire ad un osservatore di conoscere gli indirizzi originali di sorgente e destinazione, dato che rimangono nella parte non cifrata solo quelli dei due *security gateway* che effettuano lo scambio.

Come accennato IPSEC è un protocollo standard ed è quello che viene utilizzato da più tempo per la creazione di VPN; nonostante questo esso non risulta essere molto diffuso e nel tempo sono stati proposti parecchi approcci alternativi. Tutto questo è dovuto al fatto che i vari protocolli che compongono IPSEC hanno una pessima interazione il NAT, per cui la versione originale di IPSEC non è utilizzabile quando i pacchetti devono attraversare un router o un firewall che eseguono un NAT.[3]

[3]il caso può sembrare irrilevante rispetto alle situazioni illustrate in fig. 2.2, dato che basterebbe utilizzare IPSEC sul firewall o sul router dopo aver eseguito il NAT; il problema è che solo una piccola percentuale di router (quelli più evoluti) supportano IPSEC, ma anche così non si risolverebbe il problema, molto comune, di tutti quei client che si vorrebbero collegare come *road warrior* partendo da una rete privata, dato che in tal caso l'indirizzo di partenza dei pacchetti IPSEC deve stare sul client.

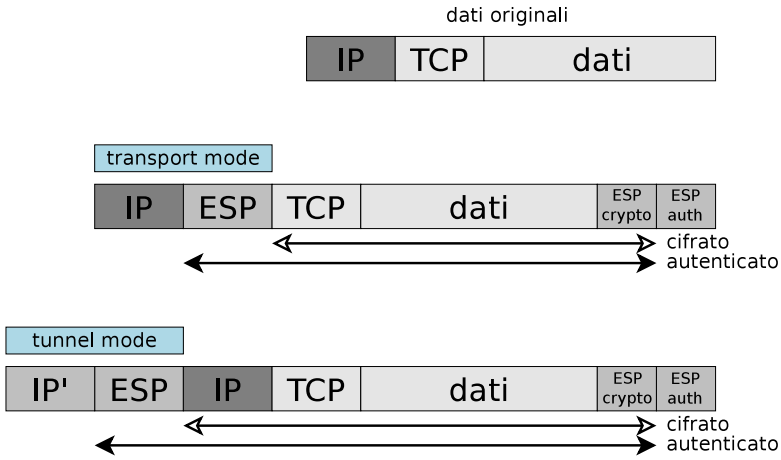


Figura 2.4: Schema di pacchetto IPSEC cifrato con l'uso di ESP.

In particolare AH semplicemente non può di funzionare in caso di NAT. Esso infatti autentica (si riveda fig. 2.3) l'intero pacchetto, compresi gli indirizzi sorgente e destinazione, che verrebbero modificati nell'attraversamento di un NAT, pertanto una volta passato il NAT il pacchetto non risulterebbe più originale. Dato che AH nasce appunto per eliminare la possibilità di ricevere pacchetti modificati, ed il NAT funziona modificando i pacchetti, i due semplicemente non possono lavorare insieme.

Con ESP, anche quando il contenuto è autenticato, il problema è più sottile in quanto in questo caso gli indirizzi esterni (vedi fig. 2.4) non sono inclusi nell'autenticazione, quindi possono essere modificati nel NAT. Restano però una serie di altri problemi; il primo è che quando il pacchetto originale contiene dei pacchetti UDP o TCP la checksum che ne verifica l'integrità è calcolata sugli IP originali, e questa, se il pacchetto è cifrato con IPSEC, non può essere aggiornata, per cui a destinazione essa non corrisponderà; inoltre si pone il problema di come reinviare all'indietro i pacchetti di risposta, dato non ci sono dati permettano di associare in maniera adeguata un certo pacchetto IPSEC ad uno specifico indirizzo interno.[4]

Oltre a questi ci sono poi un'altra serie di problemi legati all'uso di IKE che alla fine rendono impossibile l'uso della versione originale di IPSEC attraverso un NAT. Per questo motivo è stata realizzata una estensione del protocollo chiamata NAT-T (da *NAT Traversal*) che prevede l'incapsulamento dei pacchetti ESP in pacchetti UDP con l'uso della porta 4500, così che a questo punto diventa possibile attraversare normalmente un NAT come avviene per tutti i pacchetti UDP. Questa però è una estensione successiva, e compresa come parte dello standard solo con IKEv2. Inoltre alla fin fine usare questo metodo alla fine rende IPSEC equivalente alle modalità di funzionamento di altre soluzioni (come OpenVPN, che vedremo in sez. 2.3) che lavorano nativamente su UDP.

[4]nel caso di TCP e UDP questo viene fatto tenendo in conto (ed eventualmente modificando) il numero di porta, ma ESP non ha nulla di equivalente, l'unica possibilità potrebbe essere l'uso del campo SPI, che però non può essere modificato; essendo un numero 32 bit la probabilità di collisione sarebbe bassa, ma il punto è che questo è associato ad una SA, ed è quindi diverso a seconda della direzione del pacchetto e non esiste un modo semplice per associare un valore in uscita con uno in ingresso.

2.1.3 Le VPN in *user-space*

Il problema maggiore di FreeS/WAN e di tutti i derivati successivi, come *Openswan*, così come dell'implementazione nativa del kernel 2.6 ed in generale di tutte le VPN basate su IPSEC, è che il loro funzionamento, appoggiandosi su un apposito protocollo del livello di rete, è realizzato all'interno del kernel. Questo comporta in genere una procedura di installazione molto più complessa (in caso di necessità di modifiche o aggiornamenti si tratta di reinstallare il kernel e riavviare la macchina), ed inoltre in caso di vulnerabilità fornisce ad un attaccante il massimo dei privilegi possibili, ed il massimo del rischio, dato che un problema può far crollare l'intero sistema.

Se a questo si aggiunge che né FreeS/WAN né *Openswan* brillavano per semplicità di installazione e configurazione, si comprende come si siano cercate delle alternative che permettessero di implementare una VPN in *user-space*, utilizzando dei normali programmi che implementassero il tunnel attraverso una connessione cifrata appoggiandosi ad un normale protocollo del livello di trasporto (in genere tramite socket UDP), risolvendo così anche i problemi relativi all'attraversamento di firewall e NAT che sono presenti con l'uso di IPSEC.

Una tale alternativa presenta una lunga serie di innegabili vantaggi: gli aggiornamenti sono molto più semplici (basta reinstallare un programma e riavviare il servizio), è possibile far girare il servizio per conto di utenti non privilegiati (riducendo così i rischi in caso di compromissione o di bug), si evitano completamente le problematiche relative all'attraversamento di un NAT che avviene come per qualunque altro servizio basato su socket.

In genere questo tipo di prodotti si basano sull'uso di SSL, che è un protocollo ampiamente noto e ben controllato, e che non presenta le problematiche di gestione di IPSEC, che è un protocollo molto complesso che se non è configurato correttamente rischia di perdere le sue caratteristiche di sicurezza. In questo caso si tratta in genere di fare passare i dati della VPN attraverso una connessione protetta con SSL.

Capita spesso però che i vantaggi in semplicità si traducano in una diminuzione del livello di sicurezza; questo è vero in particolare per tutte quelle VPN in *user-space* che sono realizzate con un programma che permette di aprire una connessione verso un server centrale, limitandosi semplicemente a cifrare la comunicazione con SSL. Questo può rendere la configurazione anche molto semplice, ma si perde così però uno dei cardini di IPSEC, che è quello di poter identificare con certezza l'uno con l'altro i due capi della connessione.

In altri casi invece si spaccia per VPN quello che è un banalissimo tunnel relativo ad un singolo protocollo, cioè al fatto di far passare il traffico verso certi servizi per una connessione SSL, redirigendo il traffico di una o più porte su un canale cifrato come visto in sez. **??** per `stunnel`. In questo caso manca completamente la capacità, propria di una vera VPN, di dare un accesso all'intera gamma dei servizi di rete.

È pertanto essenziale nella scelta di una VPN in *user-space*, capire quali sono le modalità in cui questa viene realizzata, e quale può essere la sua utilizzabilità su diverse piattaforme, dato che in questo caso non esiste una standardizzazione di un protocollo come quella di IPSEC. Come vedremo in sez. 2.3 OpenVPN rappresenta una valida soluzione per la realizzazione di VPN in *user-space*, con funzionalità che non hanno nulla da invidiare rispetto a nessun altro prodotto, tanto è diventata praticamente uno standard presente in ogni distribuzione GNU/Linux.

2.2 La gestione di VPN con *Openswan* e *strongSwan*

Vedremo in questa sezione come realizzare una VPN con IPSEC, sia per la parte di comunicazione di rete tramite i protocolli AH ed ESP, che per la gestione di tunnel cifrati con IKE, facendo riferimento ad una delle diverse possibili soluzioni, quella realizzata dal pacchetto *Openswan*. Dato che la realizzazione di IPSEC su Linux ha avuto una storia travagliata inizieremo con una panoramica generale sull'argomento.

2.2.1 Quale IPSEC per Linux

Linux ha acquisito una implementazione ufficiale, (presente cioè nella versione standard del kernel) di IPSEC (o meglio dei protocolli AH e ESP) a partire dai kernel della serie 2.6, di questa esiste anche un *backport* non ufficiale per i kernel della serie 2.4. Questa implementazione, denominata *NETKEY*, è stata realizzata rifacendosi alla architettura della analoga implementazione realizzata dal progetto KAME per i vari FreeBSD, OpenBSD, ecc.

In precedenza però era già presente una implementazione del protocollo IPSEC realizzata dal progetto FreeS/WAN. Questo progetto, oltre ad una implementazione dei protocolli AH ed ESP, realizzata come patch al kernel ufficiale e denominata *KLIPS*, forniva anche un demone per l'uso di IKE ed il supporto per una modalità di funzionamento, detta *Opportunistic Encription* (abbreviata in OE) che fosse in grado di far utilizzare automaticamente IPSEC per ogni comunicazione con altre macchine in grado di utilizzare detto protocollo.[5]

Gli scopi del progetto erano essenzialmente di protezione della privacy (da cui derivava lo sviluppo della parte relativa alla *Opportunistic Encription*), con una forte attenzione agli aspetti politici. All'epoca il software FreeS/WAN veniva sviluppato al di fuori dagli Stati Uniti in quanto quel paese non consentiva l'esportazione di software crittografico. Nel Gennaio del 2000 i regolamenti vennero cambiati e le esportazioni (eccettuato quelle verso alcuni paesi considerati terroristi) consentite.

Dato però che non vi era certezza legislativa (il permesso resta di natura amministrativa e può essere revocato in qualunque momento) i leader del progetto FreeS/WAN continuarono a rifiutarsi di accettare contributi da cittadini statunitensi per essere assolutamente certi di evitare problemi di natura legale. Questo, insieme ad altre considerazioni di natura tecnica, e agli immancabili attriti personali, creò un conflitto con gli sviluppatori del kernel (che ritenevano accettabili contributi da cittadini statunitensi e troppo pesanti i cambiamenti apportati allo stack TCP/IP del kernel) che finirono per reimplementare indipendentemente i protocolli AH ed ESP.

In ragione di questa situazione e della percezione di uno scarso successo dell'obiettivo principale del progetto, che rimaneva quello della diffusione capillare della *Opportunistic Encription*, nel Marzo del 2004 il team originale decise di cessare lo sviluppo di FreeS/WAN. Trattandosi di software libero il codice venne immediatamente ripreso e con la nascita di altri progetti come *strongSwan* e *Openswan* che ne hanno proseguito lo sviluppo, aggiungendo al contempo un gran numero di funzionalità, come il *NAT traversal* e l'autenticazione tramite certificati, che fino ad allora erano state sviluppate esternamente.

Così dal lato del kernel ci si è trovati così di fronte a diverse implementazioni dei protocolli di rete necessari. Però, come illustrato in sez. 2.1.2, per la creazione di una VPN con IPSEC

[5] l'idea era di inserire opportuni record con chiavi pubbliche nel DNS che se ottenuti avrebbero consentito la creazione automatica di un canale cifrato con la destinazione.

non è sufficiente disporre di una implementazione dei protocolli AH ed ESP nel kernel, ma occorre anche utilizzare una implementazione del protocollo IKE per garantire autenticazione dei capi della comunicazione ed uno scambio sicuro delle chiavi crittografiche. Benché questo tipo di supporto sia stato parzialmente introdotto al di fuori di FreeS/WAN portando su Linux i programmi di gestione ripresi dal progetto KAME (che fanno parte dei cosiddetti `ipsec-tools`) l'implementazione presente in FreeS/WAN era nettamente più evoluta ed è stata ereditata dai suoi successori.

Pertanto al momento sono disponibili una larga serie di combinazioni di elementi con cui è possibile creare una VPN basata su IPSEC; nel seguito si è preferito fare riferimento alle implementazioni derivate da FreeS/WAN (*Openswan* e *strongSwan*),[6] per le maggiori funzionalità da essi fornite (in particolare per *strongSwan* che supporta pienamente anche la versione 2 del protocollo IKE) rispetto agli strumenti di `ipsec-tools`.

Inizialmente i due progetti citati erano sostanzialmente equivalenti, differenziandosi in seguito sulle funzionalità aggiunte e sul focus dello sviluppo, più centrato sulle funzionalità di autenticazione e identificazione il primo, che ha aggiunto numerose modalità di autenticazione delle VPN e diversi algoritmi di cifratura, più sulle tecnologie di rete il secondo, che continua a sviluppare per il 2.6 anche l'implementazione originale di IPSEC (*KLIPS*) del progetto FreeS/WAN.

2.2.2 Configurazione iniziale di *strongSwan* e *Openswan*

La principale differenza nelle versioni recenti di *Openswan* e *strongSwan* è che il primo continua a supportare anche per i kernel della serie 2.6 l'implementazione originaria (KLIPS) del protocollo IPSEC realizzata dal progetto FreeS/WAN, mentre *strongSwan* usa soltanto il supporto della implementazione standard del kernel anche se è in grado di usare KLIPS su un kernel 2.4.

Dato che lo sviluppo di KLIPS è comunque destinato a seguire sempre con un certo ritardo l'evoluzione del kernel, problema che non si pone rispetto all'implementazione nativa, non prenderemo in considerazione le problematiche relative alla sua installazione facendo riferimento al supporto standard (NETKEY) che è presente su tutti i kernel delle principali distribuzioni.[7] Per questo il resto di questo paragrafo tratterà solo gli aspetti del supporto IPSEC in user-space, vale a dire dei programmi che implementano il protocollo IKE.

Fra le due alternative *strongSwan* sembra quella sviluppata in maniera più attiva: ha il supporto completo per IKEv2, fornisce una documentazione molto più aggiornata e sembra orientata verso nuove direzioni di sviluppo. Tratteremo pertanto principalmente questa, illustrando brevemente, dove possibile, le eventuali differenze con *Openswan*, anche se, a causa dell'origine comune, i nomi dei file di configurazione e la relativa sintassi, molte direttive, i comandi di gestione e le relative opzioni, sono gli stessi anche per *Openswan*.

Essendo disponibili i pacchetti per le principali distribuzioni non staremo a trattare le modalità di installazione; nel caso di Debian i pacchetti forniscono i programmi per la gestione di IKE, lo script di avvio (`/etc/init.d/ipsec`), ed il programma di controllo `ipsec`. In entrambi i

[6] che nel caso di Debian vengono distribuite direttamente negli omonimi pacchetti `openswan` e `strongswan`.

[7] nel caso di Debian comunque con *Openswan* viene distribuito il supporto di KLIPS con i due pacchetti `openswan-modules-source` e `openswan-modules-dkms` che consentono una ricompilazione dello stesso come modulo in maniera relativamente semplice, mentre il kernel 2.4 contiene già integrato il backport di NETKEY.

casi con il pacchetto viene installato uno scheletro di file di configurazione.[8] La configurazione dipende infatti da quali sono i canali IPSEC che si vogliono stabilire e dalle modalità di autenticazione che si vogliono usare per la loro creazione (che tratteremo a parte in sez. 2.2.3), e dovrà essere completata manualmente.

Nel caso di *strongSwan*, che come accennato fornisce anche una implementazione completa della versione 2 del protocollo IKE, oltre al classico demone `pluto` ereditato da FreeS/WAN, che gestisce la versione 1 del protocollo, viene installato anche un secondo demone, `charon`, sviluppato da zero per questo scopo, che prevede alcune configurazioni specifiche non contemplate da *Openswan*. Questo non è disponibile per *Openswan* che usa una sua versione modificata di `pluto` che supporta alcune funzionalità di base di IKEv2.

Il file di configurazione principale è `/etc/ipsec.conf`, la cui sintassi è descritta in dettaglio nella relativa pagina di manuale, accessibile con `man ipsec.conf`. Qualora si usi una versione di *Openswan* superiore alla 2.0 la prima riga non commentata del file deve essere `version 2.0`, in quanto con il passaggio a tale versione sono cambiate alcune opzioni di configurazione,[9] ed occorre perciò segnalare esplicitamente quale sintassi si sta utilizzando. Tutto ciò non è necessario per *strongSwan* che non supporta versioni precedenti del file di configurazione.

Il file contiene tutte le informazioni di configurazione necessarie al funzionamento di una VPN, ad eccezione delle informazioni di autenticazione reciproca che, come vedremo in sez. 2.2.3 sono mantenute in `ipsec.secrets`. Per questo di norma il file non contiene informazioni riservate, e può essere tenuto accessibile in lettura.[10] Si possono comunque raccogliere eventuali configurazioni che si vogliono mantenere private in singoli file separati (senza permesso di lettura) che possono essere inclusi in `ipsec.conf` tramite la direttiva `include`, questa richiede che si specifichi un nome di file e supporta anche inclusioni multiple con l'uso del carattere jolly "`*`".

Il formato di `ipsec.conf` segue le convezioni standard per cui righe vuote e quanto segue il carattere "`#`" viene ignorato. Il suo contenuto è suddiviso in in sezioni, che sono introdotte da una riga nella forma "`tipo nome`", dove `tipo` è una parola chiave riservata che indica il tipo della sezione, e `nome` è un identificativo scelto dall'utente per distinguere sezioni dello stesso tipo.

Si tenga presente che la dichiarazione di una sezione, così come la direttiva `include`, deve sempre iniziare all'inizio di una riga; il contenuto della sezione infatti viene identificato dal fatto che le righe che ne fanno parte iniziano almeno con uno spazio; questo consente di avere una visione immediata del contenuto delle varie sezioni usando semplicemente l'indentazione.

I tipi di sezione disponibili sono tre; il primo tipo è `config`, che viene usata per impostare la configurazione di avvio dei vari demoni ed al momento questa sezione ha un unico nome possibile che è `setup`. Il secondo tipo di sezione è `conn` che identifica una connessione fra i due capi di una VPN, ed il cui nome è un qualunque identificativo associato alla stessa. Il terzo (disponibile solo per *strongSwan*) è `ca`, che permette di specificare le proprietà di una *Certification Authority*. Si è riassunta la situazione in tab. 2.1 dove si sono riportate (nella seconda parte) anche le altre direttive che pur non definendo sezioni devono essere utilizzate in maniera simile (cioè scritte senza indentazione).

[8] nel caso di *Openswan debconf* presenta una schermata in cui si chiede se usare certificati per l'identificazione (con la possibilità successiva di crearne o importarne uno) o lasciare la cosa non configurata.

[9] questo fa differenza solo nel caso di vecchie Debian, in quanto in *woody* è presente la versione 1.96 di FreeS/WAN, che usa la vecchia sintassi, tutte le distribuzioni recenti usano la nuova sintassi e specificano questo valore.

[10] l'unico caso in cui questo non è vero era con il cosiddetto *manual keying*, vedi quanto detto all'inizio di sez. 2.2.3.

Tipo	Significato
`config`	Indica una sezione di configurazione, prevede obbligatoriamente come identificativo il valore `setup`, e viene utilizzata per le impostazioni di avvio dei demoni.
`conn`	Identifica una sezione di configurazione di una connessione VPN, richiede un identificativo arbitrario purché univoco.
`ca`	Identifica una sezione che consente di applicare valori specifici ad una *Certification Authority* (valida solo per *strongSwan*).
`include`	Consente di includere il contenuto del file passato come identificativo (pertanto non è propriamente un identificatore di sezione).
`version`	Indica la versione della sintassi del file di configurazione (valida solo, e da specificare obbligatoriamente, per *Openswan*).

Tabella 2.1: Parole chiave identificanti i tipi di sezione utilizzabili in `ipsec.conf`.

Il contenuto di ogni sezione è poi costituito da una serie di direttive di assegnazione dei valori per i valori dei parametri che sono definiti in quella sezione, da esprimere nella forma `parametro=valore`, esiste però un parametro speciale, `also`, valido per tutti i tipi di sezione, che quando viene assegnato al nome di un'altra sezione consente di aggiungere le definizioni dei parametri in essa presenti alla sezione in cui viene utilizzato, come se questi venissero scritti al suo posto. L'uso di `also` può essere nidificato, e lo si può anche utilizzare più volte purché non ci sia mai un riferimento multiplo alla stessa sezione. La sezione a cui si fa riferimento deve ovviamente esistere, ed essere specificata *dopo* quella in cui lo si utilizza.

Per le sezioni di tipo `ca` e `conn` esiste inoltre il nome riservato `%default` che serve a configurare i valori di default che saranno automaticamente assegnati (qualora non sovrascritti da altre assegnazioni esplicite) ai parametri di quel tipo di sezione.[11] Queste devono essere comunque specificate prima delle sezioni normali, inoltre non si può usare il parametro `also` in una di queste sezioni.

La prima parte della configurazione è sempre costituita dalla sezione `config`, un esempio del suo contenuto, ripreso dal file `ipsec.conf` installato con *strongSwan* su una Debian Squeeze come scheletro di configurazione è il seguente:

ipsec.conf

```
config setup
        # plutodebug=all
        # crlcheckinterval=600
        # strictcrlpolicy=yes
        # cachecrls=yes
        # nat_traversal=yes
        charonstart=yes
        plutostart=yes
```

Come accennato la sezione ha come unico nome utilizzabile `setup`, e vi si richiede semplicemente l'avvio dei demoni `charon` e `pluto` con i due parametri `charonstart` e `plutostart`, tenendo

[11] questo nome veniva usato per impostare correttamente le connessioni con le versioni di FreeS/WAN precedenti la 2.0, a partire da questa versione infatti i valori di default sono stati corretti e questo tipo di dichiarazione non viene sostanzialmente più usato.

per buoni i valori di default di tutti gli altri parametri.[12] Nello scheletro resta commentata la predisposizione di alcuni dei parametri di controllo, si sono riportati in tab. 2.2 quelli ritenuti più rilevanti, per un elenco completo si deve fare riferimento alla pagina di manuale di `ipsec.conf`.[13]

Parametro	Significato
`plutodebug`	Stabilisce quale livello di messaggi di debug di `pluto` deve essere registrato. Se vuoto o con il valore `none` indica nessun messaggio, `all` indica la massima prolissità; altri valori sono illustrati nella pagina di manuale accessibile con `man ipsec_pluto` (o `man pluto`).
`plutostart`	Richiede l'avvio del demone `pluto` per IKEv1, può assumere i valori `yes` e `no`.
`charonstart`	Richiede l'avvio del demone `charon` per IKEv2, può assumere i valori `yes` e `no`.
`nat_traversal`	Indica se accettare e offrire il supporto per il *NAT Traversal*. Prende come valori `yes` o `no` (che è il default). Questo parametro è rispecificabile anche per le singole connessioni. Nel caso di *strongSwan* si applica solo a `pluto` per IKEv1, il supporto per il *NAT Traversal* è automatico con IKEv2.
`keep_alive`	Indica in secondi l'intervallo di tempo da usare per l'invio di pacchetti che mantengano attiva la comunicazione sul canale UDP usato dal *NAT Traversal* onde evitare di perdere lo stato della stessa, il default è 20.
`cachecrls`	Indica se deve essere mantenuta in cache la lista dei certificati revocati (CRL), applicabile solo a `pluto` (per IKEv2 la cache è sempre attiva), i valori possibili sono `yes` o `no` (che è il default).
`crlcheckinterval`	Specifica il tempo, in secondi in cui viene rieseguita la acquisizione della lista dei certificati revocati (applicabile solo a `pluto`).

Tabella 2.2: I principali parametri disponibili nella sezione `config` di `ipsec.conf`.

Il grosso della configurazione si effettua comunque all'interno di sezioni di tipo `conn`; ciascuna di esse infatti indica una connessione di rete che deve essere stabilita attraverso IPSEC, ed è con questa direttiva che si imposta una VPN, specificando tutti i parametri necessari alla comunicazione fra i due capi della stessa. Questo significa anche che per ognuna di queste sezioni presente su un capo della VPN, ci deve una sezione corrispondente nel file `ipsec.conf` che sta sulla macchina all'altro capo della connessione.[14]

Un esempio di un paio di sezioni di configurazione per delle connessioni è il seguente, che si è ottenuto dalla versione di `ipsec.conf` installata come scheletro di configurazione di *strongSwan* su una Debian Squeeze, si sono rimossi i commenti che sono presenti nell'esempio per una maggiore leggibilità:

ipsec.conf

```
# Sample VPN connections
```

[12] nel caso di *Openswan* questa sezione è totalmente diversa; non sono previste le direttive di avvio dei singoli demoni in quanto esiste solo `pluto`, mentre va specificato quale implementazione dei protocolli di rete usare con il parametro `protostack`, impostato ad `auto` per indicare al programma di provare prima con NETKEY e poi con KLIPS.

[13] si tenga presente che i parametri sono diversi per *Openswan* e *strongSwan*.

[14] questo ovviamente nell'ipotesi che si usi lo stesso software da ambo le parti.

```
conn sample-self-signed
      left=%defaultroute
      leftsubnet=10.1.0.0/16
      leftcert=selfCert.der
      leftsendcert=never
      right=vpn.truelite.it
      rightsubnet=10.2.0.0/16
      rightcert=peerCert.der
      auto=start

conn sample-with-ca-cert
      left=%defaultroute
      leftsubnet=10.1.0.0/16
      leftcert=myCert.pem
      right=vpn2.truelite.it
      rightsubnet=10.2.0.0/16
      rightid="C=CH, O=Linux strongSwan CN=peer name"
      keyexchange=ikev2
      auto=start
```

Dato che la configurazione deve essere equivalente sui due capi della connessione, all'interno di una sezione `conn` questi non vengono identificati in termini di locale e remoto (altrimenti occorrerebbe invertire i ruoli passando da un capo all'altro), ma in termini di *destra* e *sinistra*. I due parametri principali (ed obbligatori) da impostare nella creazione di una VPN sono infatti `left` e `right`, che identificano gli indirizzi IP pubblici dei due *security gateway* (vedi fig. 2.1) fra cui si crea la VPN. Questi devono essere specificati in notazione dotted decimal oppure con il nome a dominio completo (FQDN).

Quale dei due capi si usi per indicare la macchina locale e quale la macchina remota in genere non ha importanza, all'avvio i programmi controllano il proprio indirizzo IP e sono in grado di identificare nella configurazione quale fra *destra* e *sinistra* è la loro parte. Solo quando questo non è possibile viene considerato locale `left` (seguendo la convenzione mnemonica delle iniziali di *locale* e *remoto*).

Si può così configurare la connessione su uno dei due capi e poi copiare la relativa sezione nel file di configurazione all'altro estremo. Questo comporta che un gran numero di parametri di configurazione sono doppi, esistono cioè nella versione il cui nome inizia con `left` ed in quella in cui inizia con `right`, il loro significato è identico, ma si applica soltanto al rispettivo capo della connessione.

Oltre alla indicazione esplicita degli indirizzi IP, si possono usare per `left` e `right` anche dei valori speciali che permettono un'assegnazione dinamica; ad esempio con il valore `%defaultroute` si assegna automaticamente come indirizzo quello associato all'interfaccia da cui si esce verso internet. Ovviamente questo può essere usato solo su uno due capi della VPN, nella configurazione dell'altro si dovrà per forza indicare l'indirizzo IP pubblico.[15]

Con il valore `%any` si indica invece che l'indirizzo IP di quel capo (sottintendendo che si tratta del capo remoto) sarà scelto automaticamente in fase di connessione. Si usa in genere questo valore per identificare l'indirizzo di un *road warrior* che si collega verso un concentratore di VPN, che in genere eseguirà la connessione da IP assegnati dinamicamente da provider e non

[15] se lo si usa cioè non si avrà più la simmetria dei due file di configurazione presenti sui due capi della configurazione.

noti a priori; in tal caso nella fase di negoziazione delle chiavi sarà anche acquisito l'IP della macchina remota che verrà automaticamente utilizzato per l'invio dei pacchetti.[16]

Un secondo parametro fondamentale è `leftsubnet` (e l'equivalente `rightsubnet`) con cui si specifica la rete privata che sta dietro il *security gateway*, da indicare in notazione CIDR. Questo parametro consente di rendere accessibile detta rete dall'altro capo della VPN. Se non lo si specifica viene presa come rete il singolo IP specificato con `left`, restringendo soltanto ad esso le connessioni via VPN (normalmente questo è quanto si fa sul lato di un *road warrior*).

Il parametro `auto` serve invece a stabilire quali operazioni effettuare all'avvio di IPSEC e corrisponde ad alcune opzioni dell'uso del comando di controllo `ipsec auto` che è quello usato per avviare e fermare manualmente le connessioni; l'uso del valore `add` aggiunge le informazioni sulla connessione al database interno di `pluto`, ma non avvia la connessione, per farlo infatti occorre indicare il valore `start`, il valore di default è invece `ignore` che non avvia automaticamente nessuna operazione.

Una lista degli altri parametri più importanti delle sezioni di tipo `conn` è riportata in tab. 2.3; un elenco più dettagliato e completo, con la descrizione dei relativi significati, si trova al solito nella pagina di manuale di `ipsec.conf`. Per brevità si sono riportati solo i parametri "`left`", dando per scontato l'esistenza e l'equivalenza dei corrispettivi "`right`".

Si tenga presente inoltre che *strongSwan* a partire dalla versione 4.2.1 ha iniziato ad utilizzare un file di configurazione ausiliario, `/etc/strongswan.conf`, con una sintassi molto diversa. La configurazione per l'uso di IPSEC resta in `ipsec.conf`, ma in questo file verranno inserite le configurazioni delle varie estensioni fornite del programma. Al momento è usato principalmente per il demone `charon`, e ad esempio nella versione installata su una Debian Squeeze l'unico parametro impostato è il numero di thread con cui viene fatto partire questo programma.

2.2.3 Autenticazione e gestione delle credenziali con IPSEC

Come illustrato in sez. 2.1.2 la cifratura e la autenticazione dei pacchetti di rete del protocollo IPSEC si basa sull'uso di appropriate chiavi di cifratura con cui questi vengono generati. Se ci si limita a questo livello (cioè a ESP ed AH) tutto quello che è necessario è fornire al kernel una di queste chiavi in maniera opportuna, che è quello che può essere fatto, senza tirare in ballo l'uso di IKE (e quindi anche senza *strongSwan* o *Openswan*) con il cosiddetto *manual keying*.

In questo caso non si fa scegliere la chiave di sessione ad IKE, ma essa viene specificata manualmente dall'amministratore impostandola direttamente attraverso uno *shared secret*. Questa modalità è poco sicura in quanto per cifrare viene sempre usata la stessa chiave, la scoperta della quale compromette tutto il traffico presente e passato. Essa deve essere utilizzata solo se costretti da compatibilità con altre implementazioni di IPSEC che supportano unicamente questa

[16] la configurazione standard su un *road warrior* vede cioè l'uso di `%defaultroute` su un capo (quello locale) per ottenere l'IP dinamicamente, e dell'indirizzo del server sull'altro, quest'ultimo invece avrà `%any` sul capo che identifica il *road warrior* e potrà avere `%defaultroute` o direttamente l'indirizzo IP sul capo che identifica se stesso.

[17] questa funzionalità è fornita per compatibilità con le versioni precedenti di FreeS/WAN che non supportavano l'uso di certificati SSL ma solo di chiavi RSA; nel caso di *Openswan* questo valore può essere ottenuto tramite il comando `ipsec showhostkey`, non presente in *strongSwan*; vista la maggiore chiarezza e semplicità di configurazione è comunque consigliato sempre indicare un file contenente la chiave pubblica (per la generazione si possono usare i comandi `ssh-keygen` o `openssl rsa`).

Parametro	Significato
`left`	Indica l'indirizzo IP pubblico di uno dei capi della connessione.
`leftsubnet`	Indica la rete privata posta dietro uno dei capi della connessione.
`leftupdown`	Indica uno script da lanciare per modificare il routing o le regole di firewall quando la connessione cambia stato, il default usa `ipsec _updown`.
`leftcert`	Indica il file che contiene il certificato da usare per la propria identificazione.
`leftrsasigkey`	Indica la chiave pubblica RSA da usare per la propria identificazione, il valore di default è `%cert` che indica che deve essere estratta da un file (se ne può anche scrivere direttamente il valore).[17]
`leftsendcert`	Indica le modalità di invio di un certificato all'altro capo a scopo di identificazione, il default è `ifasked` che lo invia su richiesta, si possono usare anche `yes` o `no` (o gli equivalenti `always` e `never`), quest'ultimo viene in genere utilizzato quando si usa una identificazione senza *Certification Authority* in cui si installano esplicitamente le chiavi pubbliche ai due capi della connessione.
`leftid`	Indica la modalità con cui si accerta l'identità dell'altro capo, può essere l'indirizzo IP, un nome a dominio completo (se preceduto dal carattere "@"), o una stringa da confrontare con le informazioni presenti un certificato.
`authby`	Indica la modalità di autenticazione usata per l'identificazione dei capi della VPN, il valore di default è `pubkey`, che indica l'uso di certificati o chiavi asimmetriche, alternativamente si può usare `secret` (o `psk`) per l'uso di un segreto condiviso, le versioni recenti supportano ulteriori meccanismi, deprecato per IKEv2.
`leftauth`	Indica la modalità di autenticazione il cui uso è richiesto sul capo locale della connessione (usare `rightauth` per il capo remoto), prende gli stessi valori di `authby`, supportato solo da IKEv2.
`auto`	Indica le operazioni da effettuare all'avvio di IPSEC, i valori possibili sono `ignore` che ignora la connessione, `add` che la predispone senza avviarla, `route` che predispone anche il routing e `start` che la avvia sempre automaticamente.

Tabella 2.3: I principali parametri generali di una sezione `conn` di `ipsec.conf`.

modalità di funzionamento. Questa modalità non è supportata da *strongSwan* che suggerisce allo scopo l'uso del comando `setkey` degli `ipsec-tools`.[18]

L'uso di IKE consente invece di realizzare il cosiddetto *automatic keying*, vale a dire impostare in maniera automatica, e cambiare periodicamente (il cosiddetto *rekeying*), le chiavi di cifratura dei pacchetti, così da rendere più sicure le comunicazioni. Inoltre nel realizzare questo procedimento IKE supporta anche le modalità per identificare opportunamente l'altro capo della connessione, onde evitare la possibilità di un attacco del tipo *man-in-the-middle* (si riveda quando spiegato in sez. **??**) ed evitare l'accesso di estranei alle proprie VPN.

[18] con *Openswan* il supporto per il *manual keying* è disponibile, ma non lo prenderemo neanche in considerazione, essendo questa una modalità di utilizzo insicura, scomoda e totalmente sconsigliata.

Inizialmente il supporto di IKE fornito da FreeS/WAN prevedeva soltanto due modi per effettuare questo riconoscimento, l'usi di coppie di chiavi RSA (analoghe a quelle usate da `ssh`) in cui è necessario scambiare le chiavi pubbliche fra i due capi della connessione o l'uso di una chiave condivisa (denominata *Pre-Shared Key* o PSK) che deve essere la stessa su tutti e due.[19] In entrambe le modalità tutti i dati privati relativi alla gestione delle VPN venivano mantenuti nel file `/etc/ipsec.secrets` che per questo deve essere proprietà dell'amministratore ed accessibile solo da questi. In particolare andavano inserite in questo file le chiavi private RSA,[20] o le *Pre-Shared Key*.

Quando con le implementazioni successive è stata effettuata l'integrazione delle estensioni che consentono di utilizzare certificati SSL, sia nella forma si certificati autofirmati (sostanzialmente equivalente all'uso di chiavi RSA, formato dei file a parte) che di certificati rilasciati da una *Certification Authority*, il ruolo di `/etc/ipsec.secrets` come unica fonte dei dati privati è finito. Visto infatti che chiavi e certificati vengono in genere mantenuti in file di opportuno formato (ad esempio *strongSwan* supporta sia il formato testuale PEM che il binario DER) ormai non è più necessario inserire direttamente i dati delle chiavi private all'interno di `/etc/ipsec.secrets`,[21] e si può semplicemente usare un riferimento ai file che le contengono.

Questo ha fatto sì che una parte dei dati di autenticazione sia stata spostata nella directory `/etc/ipsec.d/`, in particolare per quanto riguarda la possibilità di mantenere in opportune sottodirectory della stessa chiavi e certificati; quelli dei propri corrispondenti, in caso di certificati autofirmati, o quelli delle *Certification Authority*, se le si usano.

Oltre a queste modifiche nel caso di *strongSwan* sono stati introdotti anche ulteriori metodi di autenticazione, cosa che ha comportato altri cambiamenti. Il formato generale di `/etc/ipsec.secrets` resta comunque lo stesso, si tratta in sostanza di una tabella di voci ciascuna delle quali è composta da una lista di indicatori di selezione, seguiti dalla specificazione del segreto da usare per gestire le connessioni che li riguardano. Lista e segreto devono essere separati dal carattere "`:`" seguito o da uno spazio o da un ritorno a capo, poi dall'identificatore del tipo di segreto (si sono riportati i valori possibili in tab. 2.4) a sua volta seguito da uno spazio e dall'opportuna indicazione del segreto stesso.

Ogni voce deve iniziare al margine sinistro del file, ma può proseguire su più righe, nel qual caso però le righe successive la prima devono essere rientranti ed indentate. Si possono inserire nel file delle righe di commento facendole iniziare con il carattere "`#`", ed inoltre si può specificare una direttiva `include` che permette di includere il contenuto del file specificato, per il nome del quale si può usare anche il carattere jolly "`*`" che sarà espanso con le regole del *filename globbing* dalla shell.

Gli indicatori della prima parte di ciascuna voce sono usati per selezionare quale segreto deve essere usato per identificare i capi di una VPN. Il significato dei selettori dipende dal tipo di segreto,[22] ma per chiavi e certificati indicano gli indirizzi dei capi corrispondenti possono

[19] questa chiave non va confusa con lo *shared secret* di cui si è parlato a proposito del *manual keying*, in questo caso la *Pre-Shared Key* serve solo per determinare l'autenticità dei due capi della VPN con IKE e non viene usata per cifrare il traffico, per il quale viene comunque usato l'*automatic keying* e delle chiavi di sessione gestite dal protocollo che vengono continuamente rinnovate.

[20] in un opportuno formato testuale; per *Openswan* questo è ancora possibile.

[21] con *strongSwan* ad esempio questo non è più possibile, e pertanto non lo prenderemo neppure in considerazione, essendo molto più semplice ricorrere all'indicazione dei file.

[22] la sintassi purtroppo è ambigua e confusa e nasce del sovrapporsi di diversi metodi, ed è inoltre pure scarsamente documentata, se non per esempi; si può considerare questo file e la sua documentazione come un ottimo

Identificatore	Significato
`PSK`	Un segreto condiviso (*Pre-Shared Key*) da utilizzare come chiave simmetrica su entrambi i capi della connessione, da specificare come stringa di caratteri delimitate da virgolette (sono validi tutti i caratteri tranne il ritorno a capo e le virgolette stesse).
`RSA`	Una chiave RSA o la chiave di un certificato X.509, da specificare tramite pathname assoluto o come pathname relativo a `/etc/ipsec.d/certs/`, se la chiave è protetta da password la si può specificare come secondo argomento o richiederne l'immissione da terminale all'avvio con `%prompt`.
`ECDSA`	Specifica una chiave privata ECDSA, usa la stessa sintassi usata per `RSA`.
`EAP`	Definisce delle credenziali per l'autenticazione con il protocollo EAP; richiede di specificare una password di autenticazione come stringa di caratteri delimitate da virgolette, deve essere usato come indicatore il nome utente (utilizzabile solo con IKEv2).
`XAUTH`	Definisce delle credenziali per XAUTH; richiede di specificare una password di autenticazione come stringa di caratteri delimitate da virgolette, deve essere usato come indicatore il nome utente (utilizzabile solo con IKEv2).
`PIN`	Definisce un PIN da usare per una smartcard, non è previsto l'uso di un indicatore.

Tabella 2.4: Identificatori del tipo di segreto utilizzabili all'interno di `ipsec.secrets`.

essere espressi come numeri IP (in notazione *dotted decimal*) come nome di dominio qualificati (preceduti dal carattere "`@`"). Se si usano le *Pre-Shared Key* non è consigliabile fare riferimento a nomi a dominio, dato che non è detto che ci si possa sempre fidare delle risoluzioni del DNS, questo non vale quando si usano segreti a chiave asimmetrica in quanto in tal caso l'identificazione è basata direttamente sulla chiave o sul certificato. Per indicare un IP qualunque si può utilizzare la notazione `%any`, anche se per compatibilità viene supportato con lo stesso significato anche l'indirizzo `0.0.0.0`.

Per determinare quale segreto (e relativa modalità di autenticazione) verrà usato fra quelli indicati in `/etc/ipsec.secrets` viene cercata una corrispondenza fra gli indirizzi locale e remoto dei due capi della connessione con gli indicatori della prima parte di ogni voce, e viene scelta la corrispondenza più specifica. L'assenza di indicatori implica che la voce corrisponderà ad una combinazione qualsiasi per gli indirizzi dei capi della VPN. Un solo indicatore verrà confrontato con l'indirizzo del capo locale (il ricevente) senza nessun controllo sul capo remoto (il richiedente), qualora ve ne siano indicati più di uno devono trovare corrispondenza entrambi i capi della connessione, a meno che non si stia usando una autenticazione a chiave asimmetrica (RSA) nel qual caso la lista viene interpretata come elenco delle possibili identità del capo locale e basta che l'indirizzo di questo corrisponda con uno degli indicatori dell'elenco.

Per capire meglio il funzionamento delle varie modalità di identificazione dei capi di una VPN tratteremo più esplicitamente i metodi più usati per la gestione delle autenticazioni: *Pre-Shared Key*, chiavi RSA e certificati. Si ricordi che indipendentemente dalla configurazione di VPN scelta (*road-warrior*, fra reti o fra singole macchine), è comunque necessario che ciascun *security*

esempio di come *non* fare le cose.

gateway possa identificare l'altro e questo comporta sia la presenza delle proprie credenziali (che nel caso di una *Pre-Shared Key* coincidono con quelle dell'altro capo) in `ipsec.secrets` per farsi autenticare, che l'indicazione di come identificare l'altro in `ipsec.conf`.

Benché meno sicuro rispetto agli altri metodi, che non prevedono la necessità di trasmissione su un canale sicuro della chiave condivisa e la compromissione di entrambi i lati della comunicazione in caso di intercettazione, l'uso delle *Pre-Shared Key* è ancora abbastanza comune per la sua semplicità di configurazione, specie quando ci si deve collegare con altri apparati per i quali l'uso degli altri metodi potrebbe risultare o più complesso o non supportato. Per autenticare un corrispondente con questo metodo occorre indicarne la scelta nella sezione che definisce della connessione in `ipsec.conf` con il parametro `authby=secret`.

In questo caso occorre anche la chiave sia indicata in `ipsec.secrets` in una voce di tipo `PSK` come stringa delimitata da delle virgolette che può contenere qualunque carattere eccetto le virgolette stesse o un a capo.[23] Un esempio di questo tipo di voci è il seguente:

ipsec.secrets

```
# sample /etc/ipsec.secrets file for 10.1.0.1
10.1.0.1 10.2.0.1: PSK "secret shared by two hosts"

# an entry may be split across lines,
# but indentation matters
www.xs4all.nl @www.kremvax.ru
    10.6.0.1 10.7.0.1 1.8.0.1: PSK "secret shared by 5"

# shared key for road warrior
10.0.0.1 0.0.0.0 : PSK "jxTR1lnmSjuj33n4W51uW3kTR55luUmSmnlRUuW..."
```

Si tenga presente che, come nell'esempio, si possono avere diverse *Pre-Shared Key*, specificate in altrettante voci del file, da usare per VPN fra macchine diverse. In questo caso come accennato la chiave da usare per una certa connessione viene determinata in base agli IP delle due macchine ai capi della VPN, cercando una corrispondenza nella lista degli indici associati a ciascuna chiave.

Questo ha una conseguenza precisa quando si usano le *Pre-Shared Key* per i *road-warrior*. In tal caso infatti, dato che l'indirizzo che essi assumeranno non può essere noto a priori, si dovrà usare nell'indice l'indirizzo generico, e a questo punto non si potranno usare due chiavi diverse per due macchine diverse, in quanto non sarebbe più possibile la risoluzione univoca della chiave da usare. Ciò comporta che per tutti i *road warrior* è necessario usare la stessa chiave simmetrica, cosa che rende questa modalità di autenticazione piuttosto debole.

La seconda modalità di autenticazione è quella basata su chiavi RSA, identificate dall'omonimo identificatore illustrato in tab. 2.4. In questo caso il possesso della chiave privata serve soltanto ad identificare noi stessi presso gli altri capi di tutte le eventuali VPN che si vogliono creare, cui bisognerà distribuire la nostra chiave pubblica. L'autenticazione degli altri capi della connessione, utilizzando un meccanismo a chiave asimmetrica, non sarà più fatta in base al contenuto di `ipsec.secrets` (non serve più infatti un segreto condiviso), ma sarà effettuata sulla base della *loro* chiave pubblica, cui dovrà essere associata, dalla loro parte, la relativa chiave privata mantenuta nel *loro* file `ipsec.secrets`. Questo consente di utilizzare chiavi diverse per ciascun *road warrior*.

[23] in questo caso la chiave serve sia ad identificare l'altro capo che a farsi identificare.

Le chiavi possono essere specificate in maniera diversa, a seconda della modalità utilizzata per definirle. Il parametro principale da usare in `ipsec.conf` in questo caso è `leftrsasigkey` che indica la chiave pubblica dell'altro capo, inoltre in questo caso non è necessario assegnare `authby=pubkey` perché questo è il valore di default. Con FreeS/WAN occorreva scrivere direttamente la chiave nel file in un opportuno formato testuale,[24] con le versioni più recenti si usa il valore speciale `%cert` e si indica il file da cui leggerla con `leftcert`, per cui è sufficiente disporre della chiave in un file di formato standard.

Allo stesso tempo perché l'altro capo della connessione possa identificarci sarà necessario specificare la nostra chiave privata in `ipsec.secrets`; questo viene fatto con l'indicativo RSA seguito dal nome del file in cui questa è contenuta.[25] Questo può essere un pathname assoluto o relativo rispetto a `/etc/ipsec.d/private`. Opzionalmente si può indicare come secondo argomento una password da usare per sbloccare la chiave (se questa è protetta) o l'uso della parola riservata `%prompt` per far effettuare la richiesta della stessa sulla console all'avvio della VPN. Un esempio di questa configurazione potrebbe essere il seguente:

ipsec.secrets

```
: RSA /etc/ipsec.d/private/hollandKey.pem
```

Si tenga presente che la configurazione andrà eseguita per entrambi i capi della VPN, usando le chiavi pubbliche di ciascuno di essi. Di nuovo non importa quale è la macchina identificata come `left` e qual'è quella identificata come `right`, l'importante è che si assegnino le chiavi pubbliche in maniera coerente; un errore comune che si aveva quando le chiavi pubbliche erano scritte direttamente dentro `ipsec.conf` era quello di assegnare per sbaglio la chiave pubblica dell'altro capo al proprio capo e viceversa, con la conseguente impossibilità di stabilire la connessione per la non corrispondenza alle rispettive chiavi private.

Con l'uso delle chiavi RSA si ottiene una maggiore robustezza nella gestione delle singole VPN, in quanto basterà generare una sola di queste chiavi per ciascun capo di una VPN, e potremo usare detta chiave per tutte le connessioni con tutti gli altri, distribuendo la relativa chiave pubblica. In questo modo non sarà necessario distribuire segreti condivisi. L'altro vantaggio è che in questo caso le configurazioni *road warrior* possono mantenere chiavi RSA diverse in quanto la scelta di quale chiave usare nella comunicazione sarà fatta in base alla chiave pubblica che sta sull'altro capo.

Infine se si usa un certificato X.509 per identificare l'altro capo si danno in sostanza due possibilità. La prima è quella di usare certificati autofirmati,[26] per i quali occorrerà semplicemente specificare il nome del file nel parametro `leftcert`, in forma assoluta o relativa se il certificato è posto nella directory `/etc/ipsec.d/certs`. Come per le chiavi RSA si dovrà aver cura di copiare il proprio certificato anche sull'altro capo e viceversa, e per entrambi assegnare correttamente sia `leftcert` che `rightcert`.

La comodità dell'uso dei certificati X.509 sta però nella seconda possibilità che prevede di eseguire l'autenticazione usando una *Certification Authority* che garantisce l'autenticità dei

[24] con *Openswan* questo è ancora possibile e si può ottenere il valore da inserire all'altro capo usando il comando `ipsec showhostkey` dalla propria parte.

[25] con *Openswan* è ancora supportata l'indicazione diretta della chiave privata dentro questo file con un opportuno formato testuale, non la tratteremo essendo una configurazione inutilmente complessa e facilmente soggetta ad errori.

[26] che ad esempio su Debian nel caso di *Openswan* vengono generati direttamente dal sistema di *debconf*.

certificati senza doverli trasferire a mano. Se si vuole usare questa funzionalità però occorre definire quale è il certificato della *Certification Authority* che firma i certificati usati dalle singole macchine. Questo può essere fatto con il parametro `leftca` in `ipsec.conf`, il cui valore deve essere assegnato al nome del rispettivo file. In generale però questo non è necessario, perché se detto parametro non viene definito, verranno usati automaticamente tutti i certificati che si trovano nella directory `/etc/ipsec.d/cacerts`, per cui tutto quel che serve è copiare in tale directory il certificato della nostra CA.

Questa seconda modalità ha il vantaggio che, come per SSL, basterà distribuire un solo certificato, quello della CA che ha firmato i certificati dei vari capi delle connessioni, invece di dover distribuire i singoli certificati pubblici di ciascuno di essi. In ogni caso si dovrà comunque impostare (con `leftcert` o `rightcert`) quale è il proprio certificato ed indicare all'interno di `ipsec.secrets` la corrispondente chiave privata; cosa che, trattandosi in genere anche in questo caso di chiavi RSA, si fa esattamente come illustrato in precedenza.

2.3 La gestione di VPN con OpenVPN

Tratteremo in questa sezione una modalità alternativa di realizzare delle VPN che non si basa su un protocollo che opera direttamente a livello di rete come IPSEC, e che per questo deve essere implementato nel kernel, ma viene realizzata a livello di applicazione da un apposito programma in user space. Pur esistendo diversi programmi in grado di fare questo[27] abbiamo scelto di concentrarci su *OpenVPN* per l'utilizzo di una tecnologia standard ed ampiamente consolidata come SSL/TLS, da cui deriva una notevole robustezza ed affidabilità dell'implementazione.

2.3.1 Introduzione

Abbiamo accennato in sez. 2.2 come l'uso di FreeS/Wan presentasse delle notevoli complessità sia sul piano dell'installazione che su quello amministrativo. FreeS/Wan inoltre implementava direttamente la gestione del routing dei pacchetti uscenti dal tunnel IPSEC, appesantendo la configurazione del servizio, ed introducendo una distinzione innaturale fra connessioni dirette fra i due *security gateway* (che devono essere impostate a parte) e connessioni fra le reti che stanno dietro di essi. Anche se con le implementazioni più recenti tutto questo si è semplificato notevolmente resta il fatto che si deve gestire esplicitamente la eventuale presenza di NAT ed una complessità di configurazione non trascurabile.

Per questi motivi l'utilizzo di un programma in user space può risultare notevolmente semplificato, sia per quanto riguarda l'attraversamento di un firewall, sia per i dettagli implementativi, che non necessitano più di una implementazione all'interno del kernel, ma possono appoggiarsi direttamente a tutte le librerie e le funzionalità che sono disponibili in user space.

Il principio di *OpenVPN* è quello di utilizzare l'infrastruttura delle interfacce `tun/tap` del kernel, una funzionalità che consente di creare delle interfacce *virtuali* sulle quali i pacchetti ricevuti, invece che da un dispositivo fisico, arrivano da un programma, al quale poi vengono mandati tutti i pacchetti inviati sulla stessa interfaccia.

[27] esempi sono `vtun` o `cipe`, che presentano però presentano mancanze notevoli sul lato crittografico, come rilevato da Peter Gutmann in `http://www.cs.auckland.ac.nz/~pgut001/pubs/linux_vpn.txt`.

In questo modo si può realizzare una VPN utilizzando due applicazioni che dialogano fra loro con la normale interfaccia dei socket, ed il cui solo compito è cifrare opportunamente tutto il traffico che vedono arrivare da questa interfaccia per inviarlo sul socket, e rimandare indietro alla stessa interfaccia, dopo averlo decifrato, il traffico che invece ricevono dal socket. Si otterrà così che i pacchetti inviati in ingresso su una interfaccia su un capo della VPN compariranno in uscita sulla corrispondente interfaccia sull'altro capo, secondo lo schema di fig. 2.5.

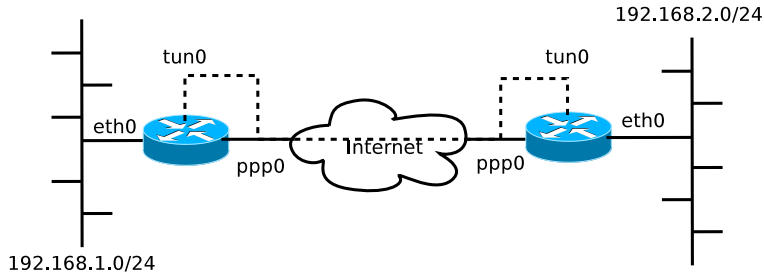


Figura 2.5: Schema di funzionamento di una VPN basata su OpenVPN.

Come si vede il concetto è tutto sommato molto semplice, l'unico problema di questo approccio è quello di una opportuna gestione del canale cifrato. OpenVPN ha risolto questo problema utilizzando dei socket UDP per l'invio dei dati attraverso internet e SSL/TLS come protocollo per la autenticazione degli estremi del tunnel e per lo scambio delle chiavi per la cifratura degli stessi.

La scelta di UDP è naturale in quanto i pacchetti inviati su una VPN implementano in sostanza una comunicazione a livello di rete facendo passare sul canale pacchetti IP generici, e UDP è il protocollo del livello di trasporto più vicino ad IP. L'uso di TCP per un tale lavoro infatti può portare a dei problemi di prestazione notevoli, in quanto le funzionalità che ne garantiscono le caratteristiche di affidabilità sono pensate per essere usate direttamente sopra IP, se queste vengono ripetute all'interno di un altro stream TCP si possono avere degli effetti di interferenza che possono portare ad una forte degradazione del traffico e anche alla caduta delle connessioni.[28]

La scelta di SSL/TLS è quello che rende superiore OpenVPN rispetto a molte altre implementazioni finora effettuate con lo stesso criterio. Programmi come `vtun` o `cipe` infatti hanno reimplementato un proprio modello di sicurezza che in analisi successive si è dimostrato essere molto debole; SSL/TLS invece è un protocollo di crittografia ed autenticazione la cui validità è stata verificata sul campo in anni di studio e di utilizzo, che pertanto rende OpenVPN pienamente affidabile sul piano della sicurezza.

Un'altra caratteristica notevole di OpenVPN è la sua disponibilità per una grande quantità di piattaforme diverse; in sostanza è stato portato su tutti gli Unix conosciuti e pure su Windows (a partire da Windows 2000), è pertanto molto semplice creare VPN anche con macchine che utilizzano sistemi operativi diversi; questo rende la soluzione estremamente flessibile e facilmente dispiegabile anche in ambienti eterogenei.

[28] una spiegazione molto chiara del perché è una cattiva idea fare una trasmissione di pacchetti TCP incapsulati su TCP si trova su `http://sites.inka.de/sites/bigred/devel/tcp-tcp.html`.

Data la maggiore semplicità dell'implementazione ed anche un sostanziale vantaggio sul piano della sicurezza non dovendo operare a livello del kernel, ci si può chiedere che senso abbia utilizzare ancora i derivati di FreeS/Wan ed un protocollo complesso come IPSEC. Un tempo uno degli svantaggi maggiori era la necessità di utilizzare una istanza diversa del programma per ciascun canale cifrato, ciascuna con un suo file di configurazione separato, cosa assai poco appetibile dal punto di vista dell'amministrazione (è in genere più gradito avere un singolo file di configurazione per ciascun servizio). Questo problema però non sussiste più con la versione 2.0 che consente di gestire con una sola istanza quanti canali si vogliono.

In realtà oggi la sola ragione a favore dei derivati di FreeS/Wan resta quella dell'interoperabilità. Benché OpenVPN sia ampiamente portabile e sia stato portato anche su molti router (ad esempio dal progetto OpenWRT), il vantaggio dell'uso di un protocollo standard come IPSEC consente di interagire con qualunque apparato che supporti lo standard, comprese implementazioni diverse dello stesso, come quelle presenti su router e firewall hardware su cui non si è in grado di installare OpenVPN. Qualora non si abbia questa necessità si può tranquillamente utilizzare OpenVPN risparmiandosi la complessità di gestione di IPSEC.

2.3.2 Installazione e configurazione di base

L'installazione di OpenVPN è estremamente semplice, anche perché viene fornito già pacchettizzato dalle principali distribuzioni;[29] inoltre il progetto fornisce pacchetti già pronti e l'installazione dai sorgenti è immediata segue la procedura standard dei classici `./configure; make; make install`.

Per quanto riguarda l'utilizzo OpenVPN può essere invocato direttamente a linea di comando, specificandone le relative opzioni, e, con l'eccezione delle opzioni utilizzate per la gestione delle chiavi, il programma verrà automaticamente eseguito in background, funzionando come un qualunque demone. Per ciascun canale cifrato si dovrà lanciare una istanza del programma, assegnandone opportunamente le porte da utilizzare.

Con la versione 2.x è diventato possibile utilizzare il programma in modalità server (vedi sez. 2.3.3), in cui più client si collegano ad una sola istanza dello stesso. Questo comporta una procedura di configurazione leggermente diversa rispetto alla configurazione classica ereditata della versione 1.x, che comunque resta utilizzabile. Inoltre nel Novembre 2004 OpenVPN 2.0 ha ottenuto dalla IANA l'assegnazione ufficiale della porta 1194 per il funzionamento in modalità server.

Tutte le opzioni di configurazione si possono specificare sia direttamente a linea di comando, come opzioni estese (cioè precedute da un `--`), che all'interno di un file di configurazione, in tal caso si potrà usare come direttiva il nome dell'opzione, omettendo il `--` iniziale. L'opzione che permette di specificare un file di configurazione è `--config` che richiede si fornisca come parametro il nome dello stesso, questa può essere ripetuta anche all'interno del file di configurazione stesso, consentendo una nidificazione (che comunque è limitata ad un numero ragionevole di livelli).

Dato che alla fine l'uso di un file di configurazione rende la gestione molto più semplice rispetto ad dover specificare un gran numero di opzioni a riga di comando, da qui in avanti faremo riferimento alle opzioni di configurazione nella loro forma di direttive.

[29] ad esempio Debian fornisce direttamente il pacchetto `openvpn`.

In genere con il pacchetto `openvpn` viene installato anche un opportuno script di avvio che permette di avviare automaticamente più canali, creati con altrettanti file di configurazione. Lo script legge il contenuto della directory `/etc/openvpn` e lancia una istanza del programma per ogni file con estensione `.conf` che trova in tale directory, passandogli lo stesso come file di configurazione.

L'installazione di default non configura nessun canale e lascia la directory `/etc/openvpn` vuota, questo significa che di default il servizio non verrà attivato. Per poter avviare il servizio occorrerà creare almeno un file di configurazione. Con il progetto vengono distribuiti una serie di file di esempio,[30] relativi a diverse configurazioni della rete, che possono essere utilizzati come scheletri, adattandoli alle proprie esigenze.

Le direttive del comando si possono dividere sommariamente in varie classi, la prima delle quali è quella delle direttive relative alla gestione del tunnel, che permettono di controllare le proprietà di quest'ultimo e le modalità della sua creazione. Alcune di queste sono fondamentali e devono essere sempre specificate per poter utilizzare il programma.

La direttiva `dev` indica quale tipo di interfaccia virtuale usare per creare il canale, ed i valori possibili sono `tap`, per l'incapsulamento su Ethernet e `tun` per l'incapsulamento su IPv4. Il tipo deve essere lo stesso su entrambi i capi della VPN, e se lo si indica senza specificare un numero di interfaccia viene utilizzata la prima disponibile, altrimenti l'assegnazione può essere fatta in maniera statica; ad esempio con "`dev tun0`" si richiede esplicitamente l'uso dell'interfaccia `tun0`.

La direttiva `ifconfig` permette di configurare le interfacce ai due capi del tunnel; i parametri da specificare cambiano a seconda che si sia usato una interfaccia di tipo `tun` o di tipo `tap`. Nel primo caso si devono specificare gli indirizzi IP ai due capi del tunnel, come si farebbe con le interfacce `ppp0` per i due capi di un collegamento punto-punto via modem.

Nel secondo caso invece si va ad operare direttamente a livello di protocollo Ethernet, e si deve quindi specificare un indirizzo IP e la relativa netmask. Questi verranno associati all'interfaccia `tap` esattamente come se si trattasse una interfaccia Ethernet ordinaria che si affaccia su una rete locale connessa tramite la VPN. Si tenga presente che questo deve essere fatto solo per un client che si collega dall'esterno in VPN, che a questo punto potrà inviare pacchetti ethernet sulla interfaccia `tap` come se questa fosse una scheda connessa, tramite OpenVPN, allo switch della LAN.

Perché questo funzioni dal lato del server che si affaccia su una LAN si dovrà in questo caso adottare una configurazione diversa, in quanto per poter reinviare i pacchetti che arrivano dall'interfaccia `tap` sulla rete locale questa dovrà essere unita alla interfaccia ethernet che su di essa si affaccia con un *bridge*, scruttando il relativo supporto e creando quest'ultimo con il comando `brctl`. Questo permette ad esempio di mantenere entrambi i lati della VPN sulla stessa rete, e di far passare sul canale cifrato quest'ultimo anche i protocolli di livello più basso (ad esempio si potrà usare un DHCP remoto attraverso la VPN).

Una terza direttiva fondamentale, che deve essere specificata per almeno uno dei due capi del tunnel, è `remote`, che serve a indicare l'indirizzo IP a cui è possibile contattare (passando attraverso internet) l'altro capo. Non specificarla significa che `openvpn` accetterà connessioni da qualunque indirizzo, previa autenticazione dell'altro capo con uno dei vari metodi supportati, permettendo così configurazioni *road warrior* in cui un capo della VPN (quello che non la usa) è quello fisso, e l'altro può porsi in contatto con lui attraverso appunto questa direttiva.

[30] su Debian sono installati in `/usr/share/doc/openvpn/example/sample-config-files/`.

Direttiva	Significato
`local`	indica l'indirizzo IP locale su cui far ascoltare il tunnel, se non specificato ascolta su tutti gli indirizzi disponibili su tutte le interfacce di rete.
`remote`	indica l'indirizzo IP dell'altro capo del tunnel.
`proto`	specifica il protocollo da utilizzare per la creazione del canale; il default è `udp`, ma se si vuole usare TCP allora occorrerà usare `tcp-client` sul capo del tunnel che inizia la connessione e `tcp-server` sull'altro capo che resterà in ascolto (è comunque sconsigliabile usare TCP).
`shaper`	impone una limitazione, specificata in byte al secondo, sul traffico che può passare attraverso il canale.
`ifconfig`	specifica due IP assegnati ai due capi del tunnel, il primo è l'IP locale, il secondo quello remoto.
`port`	specifica la porta da utilizzare per la comunicazione su internet, qualora si vogliano specificare due porte diverse per il capo locale e quello remoto si possono usare le opzioni `lport` e `rport`.
`nobind`	non effettua il binding ad una specifica porta usando la porta effimera data dal kernel.
`dev`	specifica l'interfaccia di rete da usare per creare il canale (assume i valori `tun` o `tap`).
`user`	imposta l'utente per conto del quale deve essere eseguito il demone una volta completata l'inizializzazione, in modo da rilasciare i privilegi non necessari.
`group`	analoga ad `user` per impostare il gruppo.
`log`	imposta un file sul quale eseguire il log dei messaggi del demone.
`verb`	imposta la prolissità dei messaggi di log.

Tabella 2.5: Le principali direttive di `openvpn` relative alla gestione del tunnel.

Altre direttive utili sono `log` che specifica un file su cui salvare i messaggi del demone e `verb` che indica un livello di *prolissità* degli stessi; in tab. 2.5 si sono comunque riportate le direttive per la gestione del tunnel maggiormente utilizzate; l'elenco completo, insieme ai dettagli dei parametri possibili e del loro significato si trova al solito nella pagina di manuale del comando, accessibile con `man openvpn`.

Una seconda classe di direttive è quella che riguarda le modalità per gestire l'autenticazione dei due capi della VPN e la trasmissione dei dati cifrati sul canale. OpenVPN supporta due modalità di funzionamento, la prima è quella più elementare e prevede dell'uso di una chiave statica condivisa fra i due capi della connessione, che viene usata sia per autenticare gli stessi che per cifrare i dati.

In questo caso si ha il vantaggio di una configurazione immediata, basta infatti indicare nella configurazione di ciascun estremo quale è il file contenente la chiave condivisa con l'uso della direttiva `secret`. Per gli ovvi motivi affrontati in sez. **??** ogni canale avrà bisogno di una chiave diversa. Data la criticità della chiave questa deve essere generata direttamente con il programma stesso invocandolo come:

```
monk:~# openvpn --genkey --secret chiave.key
```

che genererà un file `chiave.key` con un contenuto del tipo:

chiave.key

```
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
f6703eb17814284ef14df4fb1cf79f42
1b5718e1c86a0c16d06b8a5d4901a88d
4ed0f5fb9393cf858653daa4ed6ab65d
ec6ede77ec657ca11be448b7572ccb0b
...
...
6126425a3f4fff2f1f9743c7fd44d647
ce5058749cc4a01caaa9dd7de82fd8e7
-----END OpenVPN Static key V1-----
```

La configurazione in questo caso è immediata, in quanto basta avere una modalità sicura di copiare il file su entrambi gli estremi della VPN per avere un canale cifrato funzionante, questo però indebolisce la sicurezza complessiva del sistema, in quanto si deve comunque eseguire la copia di un file che è noto su entrambe le macchine, ed utilizzando una chiave di cifratura che non viene mai cambiata (se non manualmente) ci espone, in caso di furto della stessa, alla possibilità che tutto il traffico (compreso quello precedente al furto) possa essere decifrato.

Per questo si usa in genere questo metodo in fase di test, e poi si passa al secondo metodo di autenticazione è quello basato sull'uso di SSL/TLS. In tal caso tutta la problematica di negoziazione e scambio della chiave per la cifratura del canale viene gestita da OpenVPN attraverso una sessione TLS, che permette sia di autenticare i due capi della connessione che di creare un canale di controllo su cui scambiare le chiavi di sessione usate per cifrare il canale dei dati.

Per poter utilizzare questa funzionalità ogni capo della comunicazione deve poter disporre di un suo certificato e della relativa chiave, firmati dalla stessa *certification authority*. Ciascun capo della connessione verificherà che il certificato presentato dall'altro capo sia debitamente firmato, e se il controllo riesce a questo punto l'autenticazione sarà considerata ottenuta e si procederà con lo scambio delle chiavi di sessione.

Il primo passo per utilizzare la modalità SSL/TLS è quello di designare quale dei due capi assumerà il ruolo di server e quale quello di client (nella modalità tradizionale questi ruoli servono solo allo scopo della creazione del canale di controllo su TLS, la trasmissione dei dati è sempre punto-punto). Questo deve essere utilizzando rispettivamente le direttive `tls-server` e `tls-client` sui due estremi della VPN.

Il passo successivo è ottenere tutti i certificati necessari per i due capi, le relative chiavi ed il certificato della *Certification Authority* usata per stabilirne la validità (per le problematiche relative si consulti sez. **??**). Una volta che questi siano disponibili basterà indicarli nella configurazione utilizzando le direttive `cert`, `key` e `ca`, ciascuna delle quali vuole come parametro il nome del rispettivo file in formato PEM.

Un'altra direttiva necessaria, ma solo sul lato server dell'autenticazione TLS, è `dh`, che specifica i parametri di *Diffie-Hellman*[31], un riassunto delle altre principali opzioni di configurazione di questa classe è stato riportato in tab. 2.6, al solito l'elenco completo è nella pagine di manuale.

[31] sono i parametri necessari ad ottenere la chiave di sessione con la procedura di *Diffie-Hellman* attivando il meccanismo della *Perfect Forward Security*, che assicura che anche se un attaccante venisse in possesso della chiave di uno dei certificati, non sarebbe in grado di decifrare il traffico precedente.

Direttiva	Significato
`secret`	specifica il file che contiene la chiave statica condivisa utilizzata per la cifratura del canale.
`cert`	specifica il file contenente il certificato locale firmato (in formato PEM).
`key`	specifica il file contenente la chiave del certificato locale (in formato PEM).
`ca`	specifica il file contenente il certificato della *Certification Authority* che firma i certificati locali (in formato PEM).
`dh`	specifica il file contenente i parametri necessari per la procedura di *Diffie-Hellman* sul lato `tls-server` (in formato PEM).
`pkcs12`	specifica il file che contiene il certificato locale, la relativa chiave ed il certificato della CA che li firma in formato PKCS12.
`tls-server`	abilita l'uso di SSL/TLS e facendo assumere al capo corrente il ruolo di server nella negoziazione.
`tls-client`	abilita l'uso di SSL/TLS e facendo assumere al capo corrente il ruolo di client nella negoziazione.
`tls-auth`	abilita una protezione da attacchi DoS richiedendo una autenticazione (a chiave condivisa) prima di iniziare la negoziazione SSL/TLS.
`askpass`	richiede la password della chiave del certificato locale prima di eseguire il demone (solo da riga di comando).
`reneg-sec`	specifica ogni quanti secondi rinegoziare le chiavi di sessione (analoghi sono `reneg-pkts` e `reneg-bytes`).
`comp-lzo`	abilita un algoritmo di compressione dei dati sul canale.
`status`	scrive lo stato delle operazioni sul file passato come argomento; prende un secondo argomento opzionale per indicare la frequenza di scrittura in secondi.

Tabella 2.6: Le principali opzioni di `openvpn` relative alla gestione dell'autenticazione e della cifratura.

Una terza classe di opzioni riguarda la capacità di eseguire delle operazioni ausiliarie da compiere nell'atto di attivare la VPN, come la possibilità di lanciare una serie di script ausiliari. In particolare sono definiti 5 momenti diversi, lungo la creazione di un canale, in cui il programma può eseguire degli script esterni in grado di compiere operazioni di appoggio. Questi sono riportati nell'ordine nelle prime cinque righe di tab. 2.7.

La configurazione più comune usando OpenVPN nella modalità tradizionale è, una volta stabiliti gli indirizzi dei due capi del tunnel con `ifconfig`, quella di utilizzare `up` per lanciare uno script non appena il canale si è attivato, che stabilisca le opportune rotte statiche all'avvio della VPN, in modo da usare i due capi del tunnel come gateway per le rispettive reti private. Questa, come le altre direttive di esecuzione, prendono come parametro il nome di un comando esterno che di norma è uno opportuno script di shell che verrà eseguito nel momento indicato in tab. 2.7.

Al momento dell'esecuzione al comando esterno saranno passati una serie di argomenti che specificano le proprietà del tunnel; questi dipendono dal tipo di dispositivo che si è indicato con `dev`, nel caso di `tun` saranno passati nell'ordine: il nome dell'interfaccia del tunnel, la MTU della stessa, la MTU del collegamento, l'indirizzo IP locale assegnato al tunnel e l'indirizzo IP remoto; nel caso invece si sia usata una interfaccia `tap` saranno passati il nome dell'interfaccia, la MTU

Direttiva	Significato
`up`	esegue il comando di shell passato come parametro una volta aperte con successo le interfacce del tunnel.
`tls-verify`	esegue il comando shell passato come parametro prima della verifica dell'autenticità dell'altro capo della connessione (eseguita con SSL/TLS).
`ipchange`	esegue il comando shell passato come parametro dopo l'autenticazione della connessione o quando un indirizzo remoto è cambiato.
`route-up`	esegue il comando shell passato come parametro dopo che sono state aggiunte le rotte alla tabella di routing.
`down`	esegue il comando di shell passato come parametro dopo chiusura delle interfacce del tunnel.
`route-delay`	attende il numero di secondi passato come parametro prima di inserire le rotte nella tabella di routing (un valore nullo, il default, indica l'inserimento immediato).
`route-gateway`	usa l'indirizzo passato come parametro come valore di default per il gateway di una rotta impostata con `route`.
`route`	aggiunge una voce alla tabella di routing una volta che è stata realizzata una connessione.
`mktun`	permette di creare un tunnel persistente (che viene mantenuto anche se OpenVPN non è attivo).
`rmtun`	rimuove un tunnel permanente.

Tabella 2.7: Altre opzioni di controllo di `openvpn`.

della stessa, la MTU del collegamento, l'indirizzo IP e la netmask del tratto di rete utilizzato. In entrambi i casi si potranno usare gli argomenti per eseguire (all'interno dello script) gli opportuni comandi per impostare l'instradamento dei pacchetti attraverso il tunnel.

Oltre all'uso di comandi esterni OpenVPN supporta anche una serie di direttive che fanno eseguire alcune operazioni direttamente al programma stesso. Fra queste una delle più importanti è `route` che permette di far impostare direttamente ad `openvpn` ulteriori rotte statiche una volta che la connessione viene stabilita, senza dover ricorrere all'uso di un comando esterno lanciato con `up`.

La direttiva prende come primo parametro obbligatorio un indirizzo IP, che può essere di una singola macchina o di una rete, nel qual caso occorrerà aggiungere come secondo parametro la maschera di rete ed eventualmente un gateway per la stessa. Se non si specifica un gateway il default è di usare l'indirizzo associato al tunnel (quello specificato dal secondo parametro di `ifconfig`) oppure quello impostato con la direttiva `route-gateway`. In questo modo diventa possibile, all'attivazione di un tunnel, inserire automaticamente la rotta per gli indirizzi della rete che si trova dietro l'altro capo dello stesso.

Collegata a questa direttiva è anche `route-delay` che stabilisce il numero di secondi (passato come parametro) da aspettare prima di aggiungere una rotta alla tabella di routing. Normalmente queste vengono inserite immediatamente dopo l'apertura del dispositivo di rete (TAP o TUN che sia) e l'esecuzione dello script indicato da `up`, ma prima che vengano ceduti i privilegi di amministratore secondo quanto specificato dalle direttive `user` e `group`. In certi casi però[32] è necessario attendere un certo lasso di tempo, che può essere indicato da questa direttiva.

[32] ad esempio quando si usa il DHCP per ottenere un indirizzo sul dispositivo TAP, o su Windows per l'inizializzazione dell'adattatore TAP-Win32.

Un esempio tipico di file di configurazione tradizionale di un tunnel per una VPN basata su chiavi condivise è il seguente, che fa riferimento al capo *statico* della connessione, cioè quello che è posto su un indirizzo fisso, anche se in realtà non è necessario che l'indirizzo IP sia statico, è sufficiente che esso sia raggiungibile in maniera certa dalla macchina sull'altro capo tramite la direttiva `remote` (cosa che si può ottenere anche usando un servizio di DNS dinamico):

openvpn.conf

```
# Use a dynamic tun device.
dev tun
# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.2 10.1.0.1
# Our up script will establish routes
# once the VPN is alive.
up ./simone.up
# Our pre-shared static key
secret simone.key
# OpenVPN 2.0 uses UDP port 1194 by default
; port 1194
# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
# 9 -- verbose, good for troubleshooting
verb 3
```

si noti come con `dev` si sia scelto l'interfaccia di tipo TUN per usare l'incapsulamento su IP, poi si siano impostati gli indirizzi del tunnel con `ifconfig` ed usato `up` per invocare uno script di inizializzazione che si incarichi di impostare la rotta statica per raggiungere la rete privata dietro la nostra macchina. Con `secret` si è indicato il file su cui è salvata la chiave di accesso, e con `port` la porta da utilizzare (ogni tunnel dovrà usarne una diversa). Infine si è abilitato un adeguato livello di logging.

In corrispondenza alla precedente configurazione, quello che segue è l'estratto del file utilizzato sulla macchina all'altro capo del tunnel, che le consente di collegarsi (in questo caso da qualunque indirizzo) alla VPN:

openvpn.conf

```
# Use a dynamic tun device.
dev tun
# Our OpenVPN peer is the office gateway.
remote holland.truelite.it
# 10.1.0.2 is our local VPN endpoint (home).
# 10.1.0.1 is our remote VPN endpoint (office).
ifconfig 10.1.0.21 10.1.0.22
# Our up script will establish routes
# once the VPN is alive.
up ./simone.up
# OpenVPN 2.0 uses UDP port 1194 by default
; port 1194
# Verbosity level.
# 0 -- quiet except for fatal errors.
# 1 -- mostly quiet, but display non-fatal network errors.
# 3 -- medium output, good for normal operation.
```

```
# 9 -- verbose, good for troubleshooting
verb 3
```

e come si può notare la sola differenza è che gli indirizzi di `ifconfig` sono invertiti di ruolo, e che in questo secondo caso, essendo su una macchina senza indirizzo prestabilito, si è specificato con `remote` l'indirizzo dell'altro estremo della connessione. Si noti come queste configurazioni non facciano uso della direttiva `route`, affidandosi a degli script esterni.

Quelle appena illustrate sono le configurazioni per la creazione di un tunnel, ma occorrerà prevedere anche una opportuna configurazione del firewall che consenta di far passare i pacchetti relativi al traffico eseguito sul canale cifrato. Per questo, assunto che OpenVPN sia attivo sul firewall stesso, oltre al traffico diretto del tunnel, si dovrà anche consentire il traffico dalla rete interna verso le interfacce del tunnel e viceversa. Un insieme di regole necessarie al funzionamento di OpenVPN in modalità punto-punto è il seguente:

firewall.sh

```
iptables -A INPUT -p udp --dport 1194 -m state --state NEW -j ACCEPT
iptables -A INPUT -i tun+ -m state --state NEW  -j ACCEPT
iptables -A FORWARD -i tun+ -m state --state NEW -j ACCEPT
```

dove è dato per scontato che si accettino i pacchetti in stato `ESTABLISHED` e `RELATED` e si è assunta una sola istanza che lavora sulla porta standard. La prima regola permette di accettare le connessioni per il funzionamento del tunnel e le altre due consentono le connessioni verso la macchina locale e verso la rete locale attraverso il tunnel. Ovviamente in presenza di più istanze si dovranno aprire le ulteriori porte da esse utilizzate.

Come accennato in precedenza quando si opera con OpenVPN in modalità punto-punto ci sono una serie di operazioni che devono essere compiute al di fuori dal programma tramite gli opportuni script. Tratteremo queste operazioni nel caso particolare di una VPN realizzata con l'interfaccia `tun` illustrato dai due estratti di file di configurazione; l'uso di `tap` infatti è normalmente più complesso e meno performante dovendo costruire un livello di incapsulamento in più.

Una volta che si è creato un tunnel IP tutto quello che si ha è una comunicazione cifrata fra i due estremi della VPN che si scambiano pacchetti attraverso una interfaccia come `tun0`; in modalità punto-punto OpenVPN si limita a questo, per ottenere una vera VPN (cioè una canale di comunicazione fra due tratti di reti private) sono necessari alcuni passi ulteriori, che come accennato sono realizzati tramite gli opportuni script specificati come parametro per la direttiva `up`.

Una volta che il canale è attivo quello che serve è aggiungere nella tabella di routing di ciascun estremo della VPN una rotta statica che consenta, dalle macchine nella rete privata dietro lo stesso, di raggiungere quelle della rete dietro l'altro estremo, facendo riferimento alla configurazione di esempio di fig. 2.5, sull'estremo di destra dovremo avere uno script del tipo di:

up.sh

```
#!/bin/bash
route add -net 192.168.1.0 netmask 255.255.255.0 gw $5
```

Si noti come si sia usato come gateway per il raggiungimento dell'altra rete il quinto argomento passato allo script; questo è l'indirizzo IP associato all'interfaccia `tun` dell'altro capo del

tunnel. In questo modo, fintanto che la nostra macchina riceve i pacchetti diretti verso la rete al di là della VPN, questi verranno instradati verso l'interfaccia del tunnel, e attraverso di questa saranno cifrati e poi trasmessi su internet usando il socket di collegamento di `openvpn`. Ovviamente perché il meccanismo funzioni occorre che la stessa operazione venga ripetuta sull'altro capo della VPN, altrimenti i pacchetti non avranno una strada per tornare indietro.

2.3.3 La configurazione in modalità server

Le opzioni di configurazione trattate finora sono le stesse sia per la versione 1.0 che per la versione 2.0 di OpenVPN, quest'ultima però, come accennato, ha introdotto una nuova modalità di funzionamento che consente di utilizzare una sola istanza del programma per fare da *server* nei confronti di un numero arbitrario di client, e che insieme a questo permette una gestione molto più sofisticata (e comoda) del canale cifrato.

La direttiva che stabilisce quale modalità di operazione utilizzare è `mode`, che prende come argomenti il valore `p2p` per indicare il meccanismo classico, con una istanza del programma per canale, o il valore `server` per indicare la nuova modalità di funzionamento con una unica istanza che fa da server, introdotta con la versione 2.0. Se non la si specifica viene assunto come default il comportamento classico, il che consente di riutilizzare i file di configurazione della versione 1.0 senza modifiche.

Il principale vantaggio della modalità server è che è sufficiente usare una sola porta (la 1194 UDP) ed un unico file di configurazione per gestire l'accesso alla propria rete privata da parte di un numero arbitrario di client. Essa inoltre supporta un meccanismo che consente al server di inviare ai client i dati necessari affinché questi possano configurare la rete in maniera corretta, senza dover ricorrere a degli script ad hoc come fatto in precedenza.

La scelta della modalità server però porta in maniera sostanzialmente obbligata anche all'uso di SSL per la creazione del tunnel, dato che altrimenti si sarebbe costretti all'uso di una singola chiave segreta identica per tutti i client, con gli ovvi problemi di sicurezza che questo comporta, configurazione che pertanto non prenderemo neanche in considerazione.

La direttiva principale per l'uso della modalità server è appunto `server`, che richiede due parametri, indirizzo e netmask della rete all'interno della quale saranno scelti gli indirizzi assegnati nella creazione dei singoli tunnel. In realtà, come illustrato nella pagina di manuale, questa non è altro che una direttiva riassuntiva fornita allo scopo di semplificare la configurazione sul server, che invoca automaticamente una serie di altre direttive.

In questa modalità infatti è compito dell'istanza che fa da server definire quali sono gli IP da assegnare ai capi di ciascun tunnel, che vengono scelti all'interno di un *pool* (la direttiva sottostante è in realtà `ifconfig-pool`) analogamente a quanto avviene per il DHCP. Specificando una rete con la direttiva `server` quello che accade è che al server verrà comunque associato il primo indirizzo della rete, e sarà creata una rotta statica per la suddetta rete in modo che tutti gli indirizzi dei tunnel siano raggiungibili. Per compatibilità con Windows l'allocazione degli indirizzi viene comunque effettuata all'interno di reti `/30`, per cui su una rete di classe C si avranno a disposizione un massimo di 64 tunnel.[33]

[33] in realtà a partire da OpenVPN 2.1 esiste la possibilità di assegnare la rete degli indirizzi interni dei client con diverse metodologie tramite la direttiva `topology`, ma il funzionamento richiede versioni aggiornate di tutti i software.

Come accennato il vantaggio della modalità server è che buona parte delle configurazioni dei client possono essere amministrate direttamente dal server. Questo avviene grazie alla direttiva `push` che consente di inviare ai client che si collegano al server una serie di direttive di configurazione. L'insieme delle configurazioni inviabili ad un client è limitato per motivi di sicurezza e fattibilità, le principali sono: `route`, `route-gateway`, `route-delay`; per un elenco più dettagliato si consulti la pagina di manuale. La direttiva `push` richiede un singolo parametro, per cui si deve aver cura di proteggere la direttiva che si intende inviare scrivendone il relativo testo fra virgolette.

Quando un client intende connettersi in modalità server a OpenVPN dovrà a sua volta utilizzare la direttiva `pull`, che abilita la ricezione delle direttive di configurazione dal server; queste ultime saranno poi applicate come se fossero presenti nel file di configurazione. In generale si usa al suo posto la direttiva generica `client` che oltre ad abilitare `pull` qualifica l'istanza di OpenVPN come client anche nella negoziazione della connessione SSL.

L'uso più comune di `push` è per inviare ai vari client delle direttive `route` che permettono di configurarne automaticamente la tabella di routing, inserendovi le rotte statiche relative alle varie reti private che sono raggiungibili attraverso il server. In questo modo se si aggiunge una nuova rete dietro al server non è più necessario modificare le configurazioni di tutti i client per aggiornare gli script di `up`.

Un secondo meccanismo molto utile fornito dalla modalità server è quello che consente di modificare dinamicamente la configurazione del server stesso in corrispondenza al collegamento di un client. Questo è governato dalla direttiva `client-config-dir`, che permette di indicare una directory (relativa a `/etc/openvpn`) in cui sono mantenute le opzioni di configurazione relative a ciascun client.

Per poter utilizzare questa funzionalità però è obbligatorio usare la gestione del canale con SSL, infatti per identificare un client il server utilizza il *Common Name* scritto nel certificato con cui esso si presenta in fase di connessione. Se all'interno della directory specificata da `client-config-dir` viene trovato un file con lo stesso nome[34] presente nel certificato una volta completata la connessione verranno eseguite le direttive di configurazione in esso contenute; anche in questo caso è disponibile solo un sottoinsieme limitato di direttive, le principali delle quali sono `iroute` e `push` (per l'elenco completo si faccia riferimento alla pagina di manuale). Se non viene trovato nessun file corrispondente ma è presente il file `DEFAULT`, verranno usate le direttive presenti in quest'ultimo.

L'uso più comune di questa direttiva è quello che consente di rendere visibili fra loro la rete dietro il server e una eventuale sottorete presente dietro al client. In questo caso però non basterà aggiungere (ovviamente usando la direttiva `route`) la rete posta dietro il client alla tabella di routing del server in modo che i relativi pacchetti siano inviati sull'interfaccia di tunnel;[35] si dovrà anche dire ad `openvpn` a quale fra i vari client eventualmente connessi esso deve instradare (internamente) detti pacchetti.[36] Questo viene fatto dalla direttiva `iroute` che prende come

[34] si faccia attenzione che alcuni nomi non vengono riconosciuti correttamente, ad esempio se si usa un indirizzo di posta (tipo `piccardi@truelite.it`) si possono avere dei problemi.

[35] si suppone che il server ed il client facciano da default gateway per le loro reti, in caso contrario si dovranno aggiungere le rotte statiche sulle singole macchine.

[36] questo problema non si pone con una configurazione punto-punto, in quanto ciascuna istanza usa una interfaccia diversa; però in tal caso occorre gestire l'inserimento di tutte le rotte su entrambi i capi della comunicazione con degli opportuni script da invocare esternamente.

argomenti l'indirizzo IP della rete e la relativa netmask;[37] ovviamente questa direttiva dovrà essere inserita nel file di configurazione specifico del client dietro il quale detta rete è posta.

La presenza della direttiva `iroute` è dovuta al fatto che quando opera in modalità server OpenVPN può ricevere il traffico proveniente da diversi client su una unica interfaccia, pertanto viene ad assumere, nei confronti di detto traffico, il ruolo di un router. Il comportamento di default del programma è quello di non instradare detti pacchetti, per cui ciascun client sarà in grado di "vedere" su tale interfaccia soltanto il server; si può però far agire OpenVPN come un vero router utilizzando la direttiva `client-to-client`. In tal caso infatti OpenVPN si incaricherà di instradare anche il traffico diretto ad altri client ad esso collegati, e questi potranno anche comunicare fra di loro attraverso il server.

Direttiva	**Significato**
`mode`	Specifica la modalità di funzionamento di OpenVPN, prende come parametro `p2p` o `server`.
`server`	Configura OpenVPN per operare in modalità server con tunnel di tipo TUN; prende due parametri (numero IP e netmask) che indicano la rete su cui vengono allocati i numeri IP usati per gli estremi dei vari tunnel.
`topology`	Specifica la modalità di assegnazione degli indirizzi dei client nella rete indicata da `server`; i valori possibili sono `net30` (il default) per l'uso di una sottorete `/30` per ciascun client, `p2p` per un singolo indirizzo per client, non supportata da client Windows, `subnet` sempre per un singolo indirizzo per client, funzionante anche con versioni recenti di Windows.
`push`	Invia ad un client che si connette la direttiva di configurazione passata come parametro.
`pull`	Indica ad un client di accettare le direttive di configurazione inviategli da un server con la direttiva `push`.
`ifconfig-pool`	Definisce un intervallo di indirizzi IP, i cui estremi sono indicati dai due parametri passati alla direttiva, all'interno dei quali sono scelti dinamicamente gli indirizzi da assegnare i capi di ciascun tunnel.
`ifconfig-pool-persist`	Definisce un file, passato come parametro, dove sono registrate le corrispondenze fra client ed IP assegnati.
`client-to-client`	Abilita l'instradamento del traffico fra diversi client così che questi possano vedersi fra loro; non richiede nessun parametro.
`client-config-dir`	specifica la directory contenente le configurazioni specifiche dei singoli client.
`iroute`	Imposta una rotta interna su uno specifico client, viene usata per inviare i pacchetti destinati ad una certa sottorete al tunnel relativo ad uno specifico client; si usa all'interno delle configurazioni specifiche dei client.
`client`	Indica una configurazione di tipo client, è equivalente all'uso delle due direttive `pull` e `tls-client`.
`duplicate-cn`	Consente il collegamento anche in presenza di più client che hanno un certificato con lo stesso common name.
`max-clients`	Consente il collegamento ad un numero massimo di client passato come parametro.

Tabella 2.8: Le opzioni di controllo di `openvpn` per la modalità server.

[37] qualora si intenda inserire una singola macchina invece di una rete la netmask può essere tralasciata.

Con l'uso di questa direttiva diventa anche possibile unire più reti private, presenti dietro vari client, in modo che esse possano comunicare fra loro. Questo può sempre essere fatto in modalità punto-punto, ma il costo è quello di inviare i pacchetti da una istanza all'altra di OpenVPN, e di dover predisporre per ciascuna istanza gli opportuni script per l'inserimento di tutte le relative rotte statiche. Lo svantaggio (a parte la minore efficienza) è che la riconfigurazione va fatta per tutti i client, per cui l'aggiunta di una nuova rete dietro un nuovo client comporta la modifica delle configurazioni di tutti gli altri.

La caratteristica interessante della modalità server è che invece, una volta attivata la direttiva `client-to-client`, detta configurazione può essere realizzata operando solo sul server. In tal caso infatti oltre ai passi precedentemente illustrati per "pubblicare" la rete presente dietro il nuovo client, basterà usare la direttiva `push` per inviare a tutti i client la rotta della nuova rete.[38]

In tab. 2.8 si sono riportate le principali opzioni relative alla configurazione di OpenVPN in modalità server (sia per il server che per il client). Al solito l'elenco completo delle varie opzioni/direttive è disponibile sulla pagina di manuale del comando `openvpn`.

Vediamo allora un esempio tipico di configurazione per OpenVPN in modalità server. Partiamo dal file di configurazione sul lato server, per una macchina che fa da gateway per la rete `192.168.1.0` (ufficio) cui si collega un client che fa da gateway per la rete `192.168.0.0` (casa), una volta eliminati commenti e righe vuote si avrà:

server.conf

```
port 1194
proto udp
dev tun
ca /etc/ssl/certs/Truelite-cacert.pem
cert vpncert.pem
key vpnkey.pem   # This file should be kept secret
dh dh4096.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "route 192.168.1.0 255.255.255.0"
client-config-dir ccd
route 192.168.0.0 255.255.255.0
keepalive 10 120
comp-lzo
user nobody
group nogroup
persist-key
persist-tun
status openvpn-status.log
verb 3
```

In questo caso si è usata la porta standard, per il resto le prime 6 direttive sono analoghe a quelle che sarebbero presenti in modalità punto-punto. Rispetto agli esempi illustrati in sez. 2.3.2 questa volta si è predisposto un tunnel creato con SSL per cui si sono usate le direttive `ca`, `cert`, `dh` e `key` al posto di `secret` per dichiarare i file contenenti i dati necessari.

La prima differenza con una configurazione punto-punto è la presenza della direttiva `server` che dichiara la rete su cui saranno allocati gli indirizzi IP dei tunnel. L'impostazione della rete

[38] fra questi ovviamente non ci sarà il client dietro cui tale rete si trova; questo viene curato automaticamente da OpenVPN, che se una rete è citata in una direttiva `iroute` nella configurazione di un client, modifica il comportamento di `push` in modo che essa non gli invii una rotta statica relativa a detta rete.

presente dietro il server è invece effettuata dalla direttiva `push`, che invierà la relativa direttiva di configurazione a tutti i client. Per poter accedere alla rete `192.168.0.0` invece si usa direttamente la direttiva `route` sul server, preceduta dall'impostazione della directory per le configurazioni dei singoli client con `client-config-dir`.

Per potersi collegare al server della configurazione precedente, si potrà invece usare una configurazione client come quella del seguente esempio, in cui di nuovo si sono rimosse righe vuote e commenti:

client.conf

```
client
dev tun
proto udp
remote holland.truelite.it 1194
resolv-retry infinite
port 1194
user nobody
group nogroup
persist-key
persist-tun
ca  Truelite-cacert.pem
cert havnor-cert.pem
key  havnor-key.pem
comp-lzo
verb 3
```

Si noti come le direttive per SSL siano (ad eccezione di `dh`) sostanzialmente analoghe, e come una volta usata la direttiva `client` sia stato sufficiente indicare con `remote` l'indirizzo pubblico del server, senza avere più la necessità di predisporre script da lanciare con `up`.

Appendice A

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this

License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History

section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

A.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License or any later version applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.