IPtables for Fun – Implementare un firewall in linux

Carlo Contavalli – Copyright © 2000-2003 $Revision: 1.12 $ – $Date: 2003/07/29 3:12:46 $

Col passare del tempo, Linux sta diventando una piattaforma sempre più utilizzata nel mondo di internet. Con il maggior numero di utenti e con l'interesse suscitato ultimamente, le sue potenzialità come server di rete sono cresciute in maniera esponenziale. In particolare, il kernel Linux 2.4.x offre strumenti estremamente potenti e flessibili per la realizzazione di firewall avanzati. Questo documento si propone come scopo quello di introdurre gli utenti alla configurazione di linux come firewall utilizzando tutti gli strumenti più recenti messi a disposizione. L'ultima versione di questo documento può essere reperita in vari formati su https://github.com/ccontavalli/doc-it-iptables4fun

Contents

1 Copyright e note legali...

2 Prima di cominciare

Questa che state leggendo è la seconda versione (precisamente, $Revision: 1.12 $) di un documento che scrissi parecchio tempo fa, agli albori del kernel 2.4. Da allora, diverse versioni del kernel sono state rilasciate e nuove funzionalità sono state lentamente introdotte. Troverete quindi notevoli differenze tra questa versione e le versioni precedenti, sia in termini di contenuto tecnico sia in termini di linguaggio utilizzato.

Nelle versioni dalla 1.9 in poi, inoltre, il nome di questo tutorial è stato cambiato da "Iptables for Dummies" a "Iptables for Fun", al fine di sottolineare l'estraneità dello stesso dalla serie "for dummies" pubblicata in

libreria e di evitare problemi che si sarebbero potuti venire a creare in relazione a marchi, trademark o diritti della "Hungry Minds", diritti che non intendo in alcun modo violare.

A questo proposito, vorrei ringraziare alcuni lettori per aver segnalato il problema.

Spero che questo documento possa esservi utile e sarei felice di ricevere commenti, domande o richieste di chiarimenti (nonchè qualche invito a bere una birra in compagnia :-) al mio indirizzo di posta elettronica <ccontavalli at gmail.com>...

Devo inoltre ringraziare tutti coloro che mi hanno scritto aiutandomi a capire quali parti potevano essere di maggiore interesse, quali potevano essere migliorate o segnalando errori che spero ora aver corretto, o semplicemente incoraggiandomi nell'andare avanti, contribuendo a migliorare il documento che anche voi state leggendo.

Mi scuso se qualche volta non ho avuto il tempo di rispondere alle vostre email o se non sono sempre stato in grado di esservi di aiuto.

Purtroppo (più che altro per me), non riesco a dedicare a questo tutorial il tempo di cui avrebbe bisogno e le nuove sezioni su cui (ormai da mesi) sto lavorando verranno introdotte sempre più lentamente, quasi tanto quanto le correzioni che gentilmente mi avente inviato.

Comunque sia, versioni aggiornate di questo documento potranno essere reperite su https://github.com/ccontavalli/doc-it-iptables4fun .

Inoltre, nel corso di questo tutorial, quando si parlerà di IP e di protocolli di rete (se non esplicitamente indicato) mi riferirò alla versione 4 del protocollo IP (aka, IPv4).

Infine, se non vi interessa nulla di iptables, iproute, del kernel 2.4 e di tutte le funzioni avanzate messe a disposizione e volete proteggere semplicemente il vostro computer o la vostra piccola lan, potete limitarvi a digitare da riga di comando, come utente "root", qualcosa di simle a:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

ed eventualmente:

```
iptables -A FORWARD -s ip_rete_interna/24 -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -A POSTROUTING -s nostrareteinterna/24 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```

dove *ip_rete_interna* può, per esempio, essere sostituito con 10.0.0.0 o 192.168.0.0. Con queste ultime regole, consentirete ai computer della vostra rete che avranno impostato come gateway l'indirizzo ip del vostro firewall di condividere la connessione ADSL, ISDN o il vostro modem 56k (se siete abbastanza coraggiosi) ad esso collegati.

2.1 Ringraziamenti

Non c'è dubbio che per il tempo che mi hanno dedicato o per l'aiuto che mi è stato dato (anche solo inviandomi commenti o incoraggiandomi nell'andare avanti), debba ringraziare:

Andrea Ciancone, Guido Caruso, Gianluca D'incà, Giovanni Bassetto, Gabriele Biarese, Matteo Nastasi, Gianluca, Tony Arcucci, BlueRaven

e tutti gli amici del linux_var http://www.linuxvar.it , nonché coloro che ho dimenticato di citare in questa lista (mandatemi una mail!).

3 Nozioni di base sui firewall

Prima di entrare nel cuore della trattazione, conviene fare una piccola introduzione. Se già sapete cos'è un firewall e avete una discreta idea di come funzioni una rete, potete saltare senza troppi rimpianti questa sezione. Se invece non avete le idee tanto chiare o se per la prima volta vi trovate ad affrontare problematiche di questo tipo, vi consiglio di continuare...

Comunque sia, partendo da zero, un firewall linux non è altro che un semplice personal computer normalmente collegato fisicamente (con dei cavi) a più di una rete.

Un firewall avrà quindi installata una scheda di rete (interfaccia) per ogni rete cui sarà collegato. Inoltre, dovrà costituire l'unico punto di contatto tra le reti cui è connesso.

Per spiegarmi meglio, un firewall dovrà costituire l'unico punto di collegamento tra due o più reti. Un po' come un ponte che collega due o più isole. Senza entrare nei dettagli del protocollo ip, tutte le comunicazioni tra queste reti saranno quindi costrette a passare da questo computer.

Questo computer, una volta configurato come firewall, sarà in grado di decidere cosa far passare, cosa scartare o come reagire a determinati eventi. Tornando all'esempio del nostro ponte, configurare il computer come firewall può essere considerato equivalente a mettere una stanga e 2 finanzieri che controllano tutti i veicoli e le persone in transito. Questi 2 finanzieri saranno quindi in grado di agire secondo le istruzioni che gli verranno date dal ministero, bloccando veicoli, arrestando i contrabbandieri o sparando a vista ai ricercati.

In linux, la stanga e i finanzieri sono messi a disposizione direttamente dal kernel. Per controllare il kernel, per dare cioè le direttive ai finanzieri, vengono messi a disposizione diversi comandi: con il kernel 2.4, i principali e più avanzati sono comunque il comando "ip", "tc" e "iptables", i primi due inclusi nel pacchetto "iproute" mentre il terzo distribuito indipendentemente.

Procedendo con ordine, in linea di principio il comando "ip" vi consente di creare nuove isole e nuovi ponti, di gestire quindi le tabelle di routing e le interfacce di rete, il comando iptables di istruire i finanzieri dicendo cosa far passare e cosa non far passare o come reagire a determinate condizioni, consente cioè di creare dei filtri. Infine, il comando tc, vi consente di stabilire delle priorità e dei limiti sul traffico in ingresso ed in uscita. Ad esempio, un compito che esula dal filtraggio e che quindi riguarda tc potrebbe essere quello di riservare alcune corsie al passaggio di veicoli di emergenza o di servizio, limitando magari il traffico automobilistico, di riservare cioè determinate quantità di banda a determinate tipologie di traffico.

Riassumendo, quindi, si parlerà del comando

- ip – per configurare, attivare, disattivare o creare interfacce di rete, nonché per gestire le tabelle di routing (da solo rimpiazza i comandi ifconfig, route e arp).
- iptables – per bloccare, consentire o registrare il traffico, per effettuare cioè firewalling, ovvero per modificare alcune caratteristiche dei pacchetti in transito, per effettuare cioè NAT (Network Address Translation).
- tc – per assegnare delle priorità, imporre delle precedenze o dei limiti di banda (tc sta per "traffic classifier"), in poche parole, per effettuare dello shaping.

In questo documento si parlerà prevalentemente del comando iptables.

3.1 Tipologie di firewall

Adesso che abbiamo detto cos'è e a cosa serve un firewall, è importante parlare un pochettino delle tipologie di firewall che esistono sul mercato.

Questa non vuole essere né una trattazione completa né "esatta" dal punto di vista tecnico. Lo scopo è semplicemente quello di introdurre il lettore alla terminologia utilizzata normalmente parlando di firewall e di dare una breve panoramica sulle soluzioni disponibile per la configurazione, gestione e amministrazione delle reti.

3.1.1 Firewall stateless e firewall statefull

Prima di tutto, esistono firewall che vengono classificati come "stateless" ed altri che vengono chiamati "statefull". La differenza è abbastanza importante: firewall stateless prendono la decisione di bloccare o consentire una comunicazione soltanto sulla base delle caratteristiche della comunicazione stessa. Al contrario, firewall statefull sono in grado di riconoscere le connessioni e le trasmissioni, e, tenendone traccia, sono in grado di prendere decisioni in base ad altri parametri. Rifacendomi sempre all'esempio dei finanzieri, un finanziere "stateless" potrà bloccare o consentire un autoveicolo soltanto in base a caratteristiche come il numero di targa, il tipo di veicolo o il numero di passeggeri. Al contrario, un finanziere "statefull" sarà in grado di valutare caratteristiche come il numero di volte che un veicolo è passato, se fa parte di una comitiva di altri veicoli, piuttosto che se il veicolo dovrà fare altri viaggi per svolgere le sue mansioni. Non è proprio l'esempio migliore, ma credo che possa rendere bene l'idea. Parlando invece con un linguaggio più "tecnico", un firewall stateless conosce solo il concetto di pacchetto singolo, mentre un firewall statefull è in grado di riconoscere le connessioni e perfino alcuni protocolli. Firewall creati con il vecchio comando "ipchains" vengono considerati stateless, mentre quelli creati con "iptables" possono essere statefull.

3.1.2 Firewall e protocolli

I firewall possono essere poi classificati a secondo del "livello" (termine non strettamente corretto) a cui lavorano:

- Quelli che si intendono normalmente per firewall, lavorano solitamente al livello del protocollo ip o del protocollo tcp, filtrando il traffico in base al contenuto di queste intestazioni. Per esempio, sono in grado di filtrare il traffico in base all'indirizzo ip del mittente di un pacchetto, di destinazione o in base al protocollo utilizzato (dove per protocollo si intende la porta tcp o udp). Normalmente, un firewall di questo tipo è impostato come gateway delle reti cui è collegato.
- Firewall "trasparenti" lavorano invece ad un livello più basso del protocollo ip, molto vicino all'hardware, per cui vengono normalmente inseriti tra una rete e l'altra ma non devono essere impostati come gateway, non essendo visibili al livello del protocollo ip (è questo il motivo per cui vengono detti trasparenti - non sono visibili, e i client non devono essere configurati per usare il gateway).In linux, è possibile ottenere un firewall quasi trasparente utilizzando funzioni come il "proxy arp" in combinazione con qualche regola di iptables per modificare i pacchetti, oppure utilizzando il codice di "bridging" sempre in combinazione con "iptables". Attenzione però che attualmente il codice di bridging non è in grado di lavorare in coppia con iptables, a meno che non usiate un kernel sperimentale versione dal 2.5.45 in su o alcune patch disponibili su internet.
- Esistono poi firewall che lavorano a livelli più alti dell'ip, in grado ad esempio di censurare informazioni fornite con i vari protocolli o di consentire l'accesso solo a quegli url raggiungibili tramite link nelle pagine visitate.

Parlando poi di reti e di filtraggio, capita spesso che il discorso cada su proxy server e socks server.

3.1.3 Proxy server

Quando utilizzate un proxy server in rete (avete presente i parametri di configurazione di Internet Explorer o Mozilla?) il vostro browser non si collegherà direttamente al sito relativo alla pagina da voi richiesta,

bensì si collegherà al proxy richiedendo tale pagina. Il proxy, in vece dei client, si prodigherà per fornire questa pagina scaricandola da internet o leggendola dalla sua cache. In pratica, un proxy server non è altro che un software che tiene una memoria (cache) delle pagine visitate di recente, mettendo tale memoria a disposizione di tutti gli utenti collegati ad una rete il cui browser è configurato per utilizzare tale cache. Questo può portare ad un risparmio di banda davvero notevole, soprattutto in quegli ambienti dove molte persone si collegano più o meno agli stessi siti (e se lo spazio a disposizione per la cache è molto, sarà molto probabile che molti siti si trovino in cache).

Oltre a consentirvi di risparmiare banda (una percentuale spesso rilevante di banda), il vantaggio di utilizzare un proxy server è che vi consentono di creare delle regole molto avanzate e di interagire facilmente con gli utenti.

Con poche righe di configurazione per esempio, installando *squid* e *squidGuard* su una macchina linux è possibile filtrare le pagine web in base al contenuto, applicare delle espressioni regolari agli URL e redirigere gli utenti su particolari pagine in caso le politiche impostate vengano violate (cose estremamente difficili ed inefficienti da fare con un normale firewall). In più, i delay pools di squid consentono una distribuzione abbastanza equa della banda e con l'appoggio di un semplice firewall, un proxy server può essere reso trasparente (transparent proxy): in questo caso non ci sarà bisogno di configurare tutti i computer connessi alla rete (client) per utilizzare il proxy e tutte le connessioni verranno automaticamente deviate (redirette) sul proxy dal firewall.

3.1.4 Socks server

Esiste poi un fratello povero dei proxy server, di cui vi ho accennato prima: il socks server. Questo, invece di fare da cache, è un software che si occupa soltanto di applicare restrizioni imponendo dei limiti o dando privilegi in base a regole stabilite dall'amministratore.

Lo svantaggio principale di un socks server è che le applicazioni utilizzate devono essere fatte apposta per parlare la "lingua" di quest'ultimo e che non può essere automaticamente imposto a tutti i computer di una rete (non si può impostare un "transparent socks server"). Fortunatamente però, la maggior parte delle applicazioni attualmente disponibili sono a conoscenza di questa lingua.

Un buon motivo per utilizzare un socks server è che firewall statefull classici sono in grado di riconoscere e gestire soltanto i protocolli standard. Spesso capita quindi che protocolli inventati dai produttori di turno senza rispettare alcuno standard (o modificando gli standard a proprio piacimento) incontrino problemi non indifferenti. Configurando le applicazioni che fanno uso di questi protocolli in modo da collegarsi tramite un socks server li si costringerà a parlare una "lingua" standard che consentirà al socks server di far funzionare anche i protocolli più strani.

Normalmente quindi, utilizzando un firewall linux, a secondo delle esigenze degli utenti della rete che andrà a proteggere, si potrà installere e configurare anche un socks server per consentire a questi protocolli di funzionare.

E' importante notare però che i "socks server" in generale godono di una brutta fama per quanto concerne la sicurezza, non tanto perchè tutti i socks server siano pericolosi, quanto perchè c'è stato un periodo storico nel quale venivano preinstallati su determinati sistemi operativi senza imporre restrizione alcuna, consentendo ad utenti maliziosi di entrare nella vostra rete o di fare cose non proprio piacevoli.

Installando un socks server è quindi importante fare attenzione alle configurazioni imponendo delle regole il più possibile restrittive e cercando di evitare software notoriamente bacati. Ricordatevi infine che un socks server costituisce spesso l'unica alternativa per poter utilizzare alcuni programmi in rete, a meno di sviluppare un apposito modulo per il kernel.

4 Preparare il proprio sistema

Per poter usufruire delle nuove funzionalità messe a disposizione dal kernel di linux, è indispensabile che il vostro sistema soddisfi alcuni prerequisiti.

In particolare, dovrete avere a disposizione:

- Un kernel compilato con supporto per iptables, per il tracking (tracciamento) delle connessioni e per tutti i tipi di NAT (i kernel delle maggiori distribuzioni vengono distribuiti con i moduli necessari).
- Una versione di iptables compilata per il vostro kernel.
- Eventualmente, una versione del pacchetto "iproute", sempre compilato per essere usato con il vostro kernel.

Tutti questi strumenti sono solitamente preimpacchettati e disponibili nella maggior parte delle distribuzioni, per cui, probabilmente, non avrete nulla da fare se non qualche "apt-get install iproute iptables" o "rpm -i". Per verificare la loro presenza, potreste provare ad eseguire come "root" da riga di comando i comandi "iptables", "tc", ip. Se linux non vi restituisce un errore del tipo "commando not found" o "comando non trovato", allora sono già installati e probabilmente il kernel supporta tutte le funzionalità di cui avrete bisogno.

Se però volete dilettarvi con funzionalità particolarmente avanzate ed avere il massimo di flessibilità possibile, vi consiglio di ricompilare indipendentemente sia il kernel che iptables ed iproute. Come avrete capito, questa non è proprio una strada semplice da seguire, soprattutto se siete utenti linux alle prime armi. Non vi potrei biasimare quindi se decideste di usufruire degli strumenti messi a disposizione dalla vostra distribuzione e se decideste di saltare completamente questa sezione. In caso contrario, i prossimi paragrafi cercheranno di esservi di aiuto nel processo di ricompilare e nel capire alcuni meccanismi che stanno alla base del sistema di ricompilazione del codice di networking di linux.

Comunque sia, se avete deciso di andare avanti per la strada del "ricompilare" vi consiglio di fare tutto su una macchina che non sia il firewall: avrete la certezza di non dimenticare in giro tools come gcc, make, o le binutils. Attenzione però che senza usare i tool di impacchettamento messi a disposizione dalla vostra distribuzione, potrebbe diventare alquanto complesso e tedioso spostare i programmi appena compilati da una macchina all'altra... in questo documento verranno dati spunti per utilizzare i tool di Debian a questo proposito.

Infine, vi consiglio di eseguire le operazioni nell'ordine in cui sono presentate, onde evitare problemi di dipendenze.

4.1 Procurarsi i sorgenti

Prima di iniziare a ricompilare, potrebbe essere utile procurarsi i sorgenti dei vari programmi. Come prima cosa quindi, potreste voler scaricare:

- Gli ultimi sorgenti del kernel `http://www.kernel.org` di linux, oppure gli ultimi forniti con la vostra distribuzione (apt-cache search kernel-source; apt-get install kernel-source-2.4.x).
- L'ultima versione di iptables disponibile su `http://www.iptables.org` .
- L'ultima versione del patch-o-matic, disponibile su `ftp://ftp.netfilter.org/pub/patch-o-matic/` .
- L'ultima versione del pacchetto iproute, disponibile su `ftp://ftp.inr.ac.ru/ip-routing/` .

Ok, verificate quindi di avere tutto il necessario a disposizione per ricompilare (gcc, make, binutils, libncurses, ncurses-dev, modutils...) e date un'occhiata ai vari README, Changes o INSTALL per evitare di incappare in qualche incompatibilità e per verificare di avere eventuali librerie (o tools necessari) installati.

Spostatevi quindi in /usr/src o in una directory che vi fa comodo e decomprimete tutti i vari .tar.gz o .tar.bz2, con i soliti

```
# tar -xjf file.tar.bz2
```

oppure

```
# tar -xzf file.tar.gz
```

supponendo che stiate utilizzando una versione di GNU tar relativamente recente. Ricordatevi, inoltre, che per ricompilare il kernel vi servirà per lo meno gcc, make, le binutils e gli header delle libncurses (nonché qualche tool che sicuramente ho dimenticato).

4.1.1 Se usate debian...

Andate in /usr/src e date:

```
# apt-get install gcc make binutils libncurses5-dev
# apt-get install kernel-source-2.4.x
# apt-get build-dep kernel-source-2.4.x
# apt-get install kernel-package
# tar -xjf ./kernel-source-2.4.x.tar.bz2 &

# apt-get source iptables
# apt-get build-dep iptables

# apt-get source iproute
# apt-get build-dep iproute
```

4.2 Patchare e prepararsi a ricompilare...

Prima di ricompilare, è bene fare una precisazione. Il "netfilter" code del kernel di linux, ovvero la parte di kernel che si occupa di fare da firewall, è realizzato e rilasciato da un gruppo indipendente da quello che prettamente si occupa del kernel di linux. Capita quindi spesso che persino l'ultimissima release del kernel di linux sia rimasta indietro rispetto all'ultima versione del codice di filtraggio (il netfilter).

Considerando poi che molto velocemente vengono rilasciati bug fix e nuove funzioni vengono introdotte, una delle cose più importanti da fare è aggiornare iptables ed il kernel all'ultimissima versione, stando attenti però a mantenerli in sincronia. Per fare questo, il gruppo di sviluppo del netfilter ha messo a disposizione uno strumento estremamente potente quanto facile da usare: il patch-o-matic.

Col passare del tempo, però, questo strumento si è evoluto e modificato. Per cui, a secondo della versione di iptables che vorrete installare, dovrete seguire due percorsi diversi. In particolare, se i sorgenti di iptables che avete scaricato sono della versione 1.2.6 o minore, seguite le istruzioni indicate nella sezione "iptables 1.2.6", altrimenti seguite quelle indicate nella sezione "iptables 1.2.7a".

4.2.1 iptables 1.2.6 - Usare il patch-o-matic

Entrate quindi nella directory dei sorgenti di iptables, collegatevi ad internet ed iniziate dando un comando come:

```
# make pending-patches KERNEL-DIR=path_assoluto/delvostrokernel
```

A questo punto, un'interfaccia a menù vi guiderà nell'aggiornamento del vostro kernel. Attenzione che è meglio (a volte indispensabile) specificare un path assoluto per il proprio kernel. In pratica, usate qualcosa come /usr/src/linux anziché ../linux o ~/src/linux.

Questo comando applicherà sia al kernel che ai vostri sorgenti di iptables tutte quelle patch che sarebbero dovute essere applicate prima del rilascio del kernel ufficiale, ma che per motivi vari non hanno fatto in tempo ad essere inserite (normalmente, si tratta di correzione di bug o piccole modifiche realizzate dopo il rilascio del kernel). Se state utilizzando i sorgenti del kernel della vostra distribuzione preferita, non dovreste preoccuparvi più di tanto. Il patch-o-matic è infatti in grado di capire da solo quali aggiornamenti sono già stati applicati e di evitare la maggior parte dei problemi.

Dopo questo primo passo, nel caso vi sentiste particolarmente coraggiosi ed inebriati dal successo appena ottenuto, potreste voler proseguire con un

```
# make most-of-pom KERNEL-DIR=path_assoluto/delvostrokernel
```

Come prima, apparirà un'interfaccia a menù che vi guiderà nell'aggiornamento dei vari sorgenti. A differenza di prima, però, questo comando tenterà di installare anche le estensioni che ancora non sono parte del kernel o classificate sperimentali. Tra queste, ce ne sono parecchie che la maggior parte delle distribuzioni includono perché ormai abbastanza testate, o che comunque saranno incluse nelle prossime versioni del kernel... c'è sempre un sacco di materiale interessante che spesso vale la pena di installare. Alcune funzioni poi, vi torneranno utili dopo, magari se vorrete dilettarvi con firewall trasparenti o imponendo limiti per orari o sul numero di connessioni parallele o ancora buttando via casualmente i pacchetti delle persone che più vi stanno antipatiche.

Non preoccupatevi poi se avete paura di applicare patch incompatibili o troppo instabili: most-of-pom è fatto in modo da evitarvi simili errori. In pratica, cercherà di evitere che vi facciate male da soli.

Se invece vi sentite ancor più coraggiosi e tutte le ultime feature del netfilter vi piacciono veramente tanto, potete provare un

```
# make patch-o-matic KERNEL-DIR=path_assoluto/delvostrokernel
```

che invece vi metterà a disposizione tutte le nuove feature, dalle più inoffensive a quelle più pericolose, dalle più testate a quelle più instabili, comprese quelle incompatibili tra di loro. In questo caso, starà alla vostra abilità ottenere qualcosa di funzionante.

4.2.2 iptables 1.2.7 - Usare il patch-o-matic

Ok, in questo caso, dovrete avere a disposizione sia i sorgenti di iptables, sia i sorgenti del patch-o-matic. Dall'1.2.7 in poi, infatti, i sorgenti del patch-o-matic vengono distribuiti indipendentemente dal codice di iptables.

Ipotizzando di aver decompresso tutto in /usr/src, dovrete iniziare entrando nella directory del patch o matic, ovvero

```
# cd patch-o-matic-xxxxxxxx
```

dove le 8 x non sono altro che la data di rilascio del patch-o-matic. Da qui, vi dovrebbe bastare eseguire:

```
# ./runme pending KERNEL_DIR=/path/assoluto/del/kernel
```

dove "/path/assoluto/del/kernel" è il path dove avete decompresso il kernel, per esempio "/usr/src/kernel-source-2.4.19". In questo caso, il "pending" sta ad indicare che volete installare tutte quelle patch che sarebbero dovute essere state inserite prima del rilascio del kernel ma che per un motivo o per l'altro non sono state applicate.

Come per iptlables 1.2.6, i coraggiosi potrebbero voler riprovare ad eseguire il comando di prima sostituendo a "pending" un "base" e poi un "extra".

Il "base" vi proporrà l'installazione di quelle patch addizionali che dovrebbero lavorare bene insieme e che comunque si sono dimostrate abbastanza stabili. Al contrario, un "extra" vi proporrà l'installazione di *tutte* le patch disponibili, anche di quelle che potrebbero causare problemi. Fra queste, alcune possono essere particolarmente utili, come il supporto per l'h323 o per protocolli o filtri un po' esotici, come lo "string" o l'"iplimit". In ogni caso, vi verrà presentata una simpatica e semplice da utilizzare interfaccia a menù.

iptables 1.2.7 - in Debian Il pacchetto Debian per iptables 1.2.7 è strutturato in modo che nella directory upstream vi siano alcuni file di documentazione ed i sorgenti originali di iptables (in formato tar.bz2), mentre nella directory debian vi siano i file necessari per ricompilare iptables.

Automaticamente quindi, il processo di compilazione (avviato utilizzando il solito "debuild" o "dpkg-buildpackage")si occuperà di prendere i sorgenti del kernel indicati nel file `debian/control` (campo Depends, che dovranno già essere installati sul sistema), decomprimerli e patcharli con il patch-o-matic contenuto nella directory "upstream" ed utilizzando tutte le patch elencate nel file "patch-o-matic.accepted.list".

Alla fine del processo di compilazione, dovreste quindi trovarvi con i nuovi pacchetti .deb per iptables ed un nuovo .tar.bz2 contenente i sorgenti del kernel patchati in automatico da "debuild". A questo punto, dovrete rimuovere i sorgenti precedentemente decompressi in `/usr/src` ed utilizzare quelli appena generati dal processo di compilazione di iptables, proseguendo poi con il processo di compilazione standard del kernel.

Se invece volete personalizzare il processo di compilazione, dovrete:

- Per utilizzare un kernel diverso da quello per cui è stato creato il pacchettino, vi consiglio di dare un comando, dalla directory dove avete decompresso i sorgenti (per esempio, `/usr/src/iptables-1.2.7a`) simile a:

  ```
  # perl -pi -e s/kernel-source-2.4.19/kernel-source-2.4.20/g ‘find ./ -type -f‘
  ```

  per utilizzare i sorgenti del 2.4.20 anziché quelli del 2.4.19. Questo comando, semplicemente, sostituisce in ogni file della directory la stringa "kernel-source-2.4.19" con la stringa "kernel-source-2.4.20". Abbiate fiducia, la stringa "kernel-source-2.4.19" appare solo dove viene utilizzata per decomprimere i sorgenti del kernel.

- Per utilizzare una versione diversa del patch-o-matic, scaricate tale versione in formato .tar.bz2 nella directory upstream, dopodichè date:

  ```
  # perl -pi -e s/patch-o-matic-20020825/patch-o-matic-20030220/g ‘find ./ -type -f‘
  ```

  per utilizzare la versione 20030220 al posto della versione 20020825.

- Infine, per elencare delle patch aggiuntive da applicare, potreste voler aggiungere delle nuove righe nel file "patch-o-matic.accepted.list" nello stesso formato in cui le altre righe sono indicate (in poche parole, non cambiate la directory, fate un copia & incolla cambiando solo il nome della patch).

Questa probabilmente non è la procedura più facile o più semplice da utilizzare, ma il pacchettino debian non è stato progettato per essere facilmente modificato dagli utenti.

In questi casi, la soluzione migliore potrebbe essere quella di aspettare la nuova versione del pacchetto.

Comunqe, per compilare i sorgenti, dovrete ancora una volta utilizzare "debuild" o "dpkg-buildpackage", ricordandovi di utilizzare la nuova versione del kernel che verrà generata sotto forma di .tar.bz2 dal processo di compilazione di iptables.

4.2.3 Altre patch

Ok, adesso avete l'ultimissima versione del netfilter e di iptables. Se state però configurando un firewall ad uso provider, reti di grosse/medie dimensioni o se avete intenzione di installarlo in posti particolarmente selvaggi con un minimo di banda a disposizione (non vale la pena se avete solo una ADSL o poco di più...), potreste voler essere in grado di effettuare dello shaping. Alcune tra le patch più famose per lo shaping sono:

- L'HTB, ovvero Hicerical Token Bucket filter, un class based queue molto più facile da usare e più preciso (nonché altrettanto potente) dell'equivalente CBQ (tra l'altro, molto ben documentato). E' incluso nei kernel dal 2.4.20 in su.
- Il wrr, o Weighted Round Robin, utile soprattutto se avete tanti utenti dove alcuni si divertono con flashget o equivalenti a saturare completamente la banda bloccando a tutti gli altri persino la navigazione. Disponibile su http://wipl-wrr.sourceforge.net , distribuisce equamente la banda in base alle persone collegate.
- Se invece non avete una connessione dedicata ma tante ADSL, potreste volere patchare il kernel in modo che possa essere in grado di fare il bilanciamento non solo per pacchetto (non funzionerebbe per via del NAT, probabilmente) ma per connessione. Non ricordo però l'URL della patch.

In questo documento, comunque, non tratteremo l'uso di queste patch, bensì solo la loro installazione.

Comunque sia, per queste e tutte le altre patch che vorrete applicare sul codice di filtraggio o di shaping, la procedura generica è questa:

1. Se la patch riguarda nuove capacità di filtraggio, probabilmente riguarda iptables, mentre se introduce nuovi metodi per gestire il traffico dei pacchetti o le tabelle di routing, probabilmente riguarda iproute.
2. Decomprimete la patch da qualche parte sul vostro disco. Dovrebbero venir fuori due file .diff: uno per iproute o iptables, l'altro per il vostro kernel.
3. Entrate nella directory del vostro kernel (per esempio, /src/linux/) e date un

```
# patch -p1 < path/file_del_kernel_patch.diff
```

4. Entrate nella directory di iptables o di iproute (per quanto riguarda il netfilter ed il codice di shaping, il 99% delle volte ad ogni patch del kernel corrisponde una di iptables o iproute) e date:

```
# patch -p1 < path/file_di_iptables_o_iproute_patch.diff
```

4.3 Preparare il kernel

Esistono centinaia di howto (se non migliaia) in giro per internet su come ricompilare il kernel e per questo mi limiterò a dare le istruzioni fondamentali per avere un buon kernel adatto per un firewall.

Ricordatevi però sempre di configurare il kernel **dopo** :-) aver applicato tutte le patch, onde evitare di faticare ad applicare delle patch che non verranno mai utilizzate per via dei parametri scelti con "menuconfig" (o "xconfig" o ...).

4.3.1 Configurarlo

Ok, a questo punto dovreste avere il vostro kernel decompresso in giro da qualche parte su una delle vostre partizioni. Entrate dunque nella directory dove è stato decompresso e preparatevi per compilarlo con il solito:

```
# make menuconfig
```

Se vi si presenta un errore qui, la maggior parte delle volte è perché vi mancano gli header delle ncurses o dei tool fondamentali. Rileggete la sezione sul *prepararsi a ricompilare* e su come *procurarsi i sorgenti*. Abilitate quindi tutto ciò che vi serve, con particolare attenzione però al menù delle "networking options". Abilitate dunque, tenendo le feature sperimentali disabilitate:

```
Packet socket
  Packet socket: mmapped IO
Netlink device emulation
Network Packet filtering
Socket Filtering
Unix Domain Sockets
TCP/IP networking
  IP: multicasting
  IP: advanced router
       e tutte le voci che appariranno
  IP: Tunnelling
  IP: GRE tunnels over IP
    IP: Broadcast GRE over IP
  IP: multicast routing
       e tutte le voci che appariranno

  IP: TCP syncookie support
       (solo se avete un kernel relativamente recente, non ricordo
       la versione corretta, ma ogni > 2.4.10 dovrebbe andare)
  IP: Netifilter configuration
    Connection Tracking
       e tutte le voci che appariranno
    IP Tables support
       e tutte le voci che appariranno
802.1d Ethernet Bridging
E tutte le voci sotto 'QoS' and fair queuing
```

Attenzione che la maggior parte di queste opzioni non sono indispensabili (anzi, probabilmente non le userete mai). Quando però mi capita di ricompilare un kernel per un firewall, mi piace pensare che abbia tutto ciò che al momento è disponibile, e spesso torno indietro abilitando anche le feature sperimentali, in modo da poter selezionare:

- Tutte quelle feature aggiunte tramite patch
- Il supporto per le VLAN
- Molti altri componenti già forniti col kernel, parte di iptables, ma considerati sperimentali.

Attenzione che il vostro kernel non influenzerà soltanto il comportamento del firewall in rete, ma sarà un componente indispensabile per la sicurezza e la stabilità del vostro sistema. Assicuratevi quindi di comprendere pienamente tutto ciò che selezionate (comprese le cose che vi ho indicato), e fatte le dovute

considerazioni, non sentitevi in colpa a fare diversamente da quanto suggerito. Nella peggiore delle ipotesi, dovrete ricompilare il kernel...

Ci sono quindi diverse cose che potreste considerare configurando il kernel del vostro firewall:

- Aggiungere delle patch per "rinforzare" il sistema (la openwall, www.openwall.com non appena sarà disponibile, oppure la LIDS, che però è molto invasiva).
- Disabilitare il supporto per i moduli (esiste almeno un rootkit basato su moduli del kernel, ed un kernel senza supporto per i moduli potrebbe complicare non di poco il lavoro di un possibile intruso).
- Abilitare file system come il reiser, l'ext3 o il devfs (per essere in grado di montare i dischi read-only e per avere una macchina che si comporti un po' come un videoregistratore: che si possa cioè tranquillamente spegnere e accendere senza avere alcuna competenza di linux, o che comunque in caso di mancanza di corrente non abbia alcun problema a ripartire velocemente).
- Abilitare il frame buffer, per riuscire a scrivere facilemente regole piú lunghe.

Questi sono soltanto spunti o suggerimenti, sta al vostro estro produrre un kernel il più utile e funzionale possibile.

Ricordatevi poi che in un firewall non esiste solo il networking, e non dimenticatevi quindi di scegliere opportunamente il processore, le periferiche e...

Infine, dopo aver ricompilato ed incluso tutte le funzioni che più vi stanno a cuore, ricordatevi di verificare di aver ottenuto qualcosa di stabile ed utilizzabile, provando il vostro firewall prima di metterlo in produzione.

4.3.2 Ricompilarlo

Per molti di voi sarà probabilmente superfluo dire di dare:

```
# make dep bzImage modules modules_install
```

Seguite poi la procedura della vostra distribuzione (o prevista dal vostro boot loader) per installare il nuovo kernel (lilo piuttosto che grub).

Se ci dovessero essere errori, l'unica cosa che posso suggerirvi è di provare a correggere gli errori o eventualmente disabilitare una patch per volta (via menuconfig) fino a riottenere qualcosa di ricompilabile...

In Debian Date:

```
# apt-get install kernel-package
# make-kpkg binary modules
# cd ..; dpkg -i kernel-image-2.4.x....deb; dpkg -i kernel-headers*.deb
```

4.4 Compilare iptables

Non dovreste avere grossi problemi... seguite le istruzioni indicate nel file INSTALL e tutto vi andrà bene, a meno che non abbiate abilitato qualche patch troppo sperimentale... comunque, se siete abbastanza pigri da non voler leggere un file in inglese, vi dovrebbe bastare qualcosa come (dalla directory di iptables):

```
# make install KERNEL_DIR=path_assoluto/delvostrokernel/
```

Se avete debian, dovreste già avere i pacchettini compilati nei precedenti passi. Questo è il momento buono per utilizzare "dpkg -i iptables_*.deb" per installarli.

4.5 Compilare iproute

iproute normalmente è un pochettino più problematico. Questo soprattutto perché ha bisogno che gli headers file del kernel installato sul sistema siano relativi al kernel che avete appena patchato e ricompilato. Se avete già provato a fare questa procedura, avrete senz'altro visto diversi errori relativi a dei tipi non definiti o errori in pak_sched.h (a meno che non abbiate applicato alcuna patch). Ma andiamo per gradi:

- Andate in /usr/include e fate sparire momentaneamente la directory "linux" (ad esempio, con un "mv ./linux /"). Attenzione! Cosí facendo potreste creare problemi ad eventuali altri utenti del sistema, e non è certo il metodo più corretto!
- Date un "ln -s /path_assoluto_sorgenti_nuovo_kernel/include/linux ./linux"

In questo modo avete messo a posto gli headers file per poter ricompilare iproute (esistono centinaia di modi più furbi per farlo, ma credo che questo sia uno dei più semplici).

Entrate quindi nella directory di iproute e provate a compilarlo seguendo le istruzioni fornite (dovrebbe bastare un semplice "make install"). Non vi spaventate se non va liscio la prima volta: alcune versioni di iproute citano in uno dei file .c una costante che non è più definita negli headers di linux.

Per risolvere il problema, basterà semplicemente aprire con un editor il file nella directory lib chiamato ll_proto.c (iproute/lib/ll_proto.c) e commentare via la riga (sul mio sistema la 36)

```
__PF(ECHO,echo)
```

sostituendola con la riga

```
/* __PF(ECHO,echo) */
```

Non vi preoccupate, non è una mia idea ed è un problema abbastanza documentato dove è prassi normale per risolverlo commentare via la riga indicata (tra l'altro, al momento del rilascio di questo documento, potrebbe essere un problema giá risolto).

Riprovate quindi a compilare... se ancora non vi va bene, controllate di aver sistemato correttamente `/usr/include/linux` e di aver applicato a iproute tutte le patch che avete applicato al vostro kernel e viceversa.

Se ancora non riuscite a compilare e non avete delle buone conoscenze del C, vi consiglio di riprovare a ricompilare iproute senza nessuna patch, poi con una, poi con due... finché non smette di funzionare. Se già dall'inizio non si ricompila, probabilmente vi manca qualche tool necessario per la compilazione o avete qualche header un po' particolare. Provate a chiedere sulle mailing-lists delle vostre distribuzioni.

Se invece non vi ha dato problemi, ed è andato tutto liscio al primo colpo, probabilmente siete stati fortunati... non vi preoccupate :-)...

4.5.1 In Debian

La procedura è la stessa di iptables: entrate nella directory di iproute e date

```
# debuild
# cd ..; dpkg -i *.deb
```

oppure

```
# dpkg-buildpackage
# cd ..; dpkg -i *.deb
```

per fabbricare i .deb contenenti la vostra versione personalizzata di iproute.

Se la procedura si blocca durante la compilazione, leggete le raccomandazioni nella sezione precedente, dove si descrive la procedura di installazione manuale, e verificate di aver già installato il nuovo kernel con dpkg.

5 Hardening the system

Con "hardening the system" si intende la procedura utilizzata per rendere il sistema il più resistente possibile ai vari tentativi di attacco. La procedura di per sé è abbastanza semplice, e si può riassumere in quattro semplici passi:

- Eliminare tutto ciò che è inutile al funzionamento del firewall, ovvero tutto ciò che potrebbe essere utilizzato contro di voi.
- Rendere difficilmente identificabile il sistema in modo da rendere più difficile per un attaccante capire che strumenti utilizzare per potervici entrare.
- Limitare le capacità del firewall in maniera da rendere la vita il più difficile possibile per un intruso (dischi readonly, mancanza di tool basilari, strette regole di firewalling anche per la catena in OUTPUT...).
- Mettersi nella posizione di poter rilevare facilmente un'intrusione.

In questo documento vi vengono date delle indicazioni molto sbrigative e sommarie in proposito, che dovrebbero essere interpretate più che altro come spunti. Vi consiglio di riferirvi a documentazione specifica in proposito.

5.1 Ridurre i punti di ingresso

L'idea di base è quella di eliminare tutto ciò che in rete potrebbe essere facilmente attaccato. Potreste quindi voler seguire le indicazioni contenute nelle prossime sezioni.

5.1.1 inetd.conf

In questo file sono indicati alcuni servizi di rete che dovranno essere attivi sulla vostra macchina. Alcune distribuzioni, invece dello standard inetd, utilizzano xinetd che fa uso di una struttura a directory e di un diverso file di configurazione. Se la vostra distribuzione utilizza inetd.conf, vi dovrebbe bastare commentare tutte le righe che servono per attivare servizi che non vi interessa offrire, inserendo un "#" all'inizio della riga stessa. Un tipico file inetd.conf al termine dell'installazione di debian (a secondo di quello che È stato installato) potrebbe essere simile al seguente:

```
# [...]
#
#:INTERNAL: Internal services
echo            stream  tcp     nowait  root    internal
echo            dgram   udp     wait    root    internal
chargen         stream  tcp     nowait  root    internal
chargen         dgram   udp     wait    root    internal
discard         stream  tcp     nowait  root    internal
discard         dgram   udp     wait    root    internal
daytime         stream  tcp     nowait  root    internal
daytime         dgram   udp     wait    root    internal
```

```
time            stream  tcp     nowait  root    internal
time            dgram   udp     wait    root    internal

#:STANDARD: These are standard services.
ftp             stream tcp nowait root /usr/sbin/tcpd /usr/sbin/proftpd

#:BOOT: Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
#tftp           dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/in.tftpd /boot

# [...]
```

La maggior parte di questi servizi (il nome indicato nella prima colonna), sono servizi *nix standard, forniti per motivi di conformità con altri sistemi *nix e per lo più inutili.

Nel nostro caso, il firewall non offrirà alcun servizio, nel senso che non vi sarà installato alcun server, salvo, forse, per il server "ssh" per l'amministrazione remota. Dovremmo quindi provvedere a commentare tutte le righe inserendo un cancelletto all'inizio della riga, come mostrato per il servizio "tftp".

Nel caso decidiate di lasciare alcuni servizi aperti, fate attenzione alla quinta colonna, che contiene i privilegi con cui il programma verrà eseguito, e alla sesta, che contiene il programma da eseguire. A parte i programmi indicati come internal, la maggior parte delle volte vedrete in sesta colonna il comando "/usr/sbin/tcpd". Questo, in gergo, viene definito un "wrapper". Quello che fa questo comando, infatti, è semplicemente eseguire ciò che viene indicato nella colonna successiva. L'utilità del wrapper è che prima di eseguire il comando indicato, quando riceverà una connessione, questo si occuperà di effettuare alcune verifiche decidendo se bloccare o consentire l'utilizzo del servizio.

Le decisioni di questo wrapper sono normalmente basate sul contenuto del file `/etc/hosts.allow` e `/etc/hosts.deny`. Maggiori informazioni sul formato di questo file possono essere reperite sulla pagina di manuale di "hosts_access" e "hosts_options".

5.1.2 Directory /etc/init.d

In questa directory sono normalmente contenute due categorie di script:

- quelli necessari per effettuare alcune operazioni durante l'accensione e lo spegnimento del computer.
- quelli necessari per far partire o bloccare quei servizi che una volta attivati dovrebbero continuare a funzionare finchè non esplicitamente bloccati (facendo il logout non si fermano – demoni come apache, bind, sendmail o ...).

In Debian e in altre distribuzioni con un sistema di boot conforme a quello di System V, in `/etc` (o `/etc/rc.d`) esistono altre directory chiamate `rc0.d`, `rc1.d`, `rc2.d` e così via. Queste directory contengono dei link simbolici agli script in "init.d" che dovranno essere eseguiti quando si passa rispettivamente a runlevel 0, 1, 2 e così via.

Ogni runlevel, poi, può essere utilizzato in modo diverso e gli può venire assegnato un particolare significato. Ad esempio, l'operazione di shutdown consiste nel passaggio dal runlevel corrente al runlevel 0 (per cui, tutti gli script in rc0.d verranno utilizzati per bloccare i vari servizi e per eseguire le operazioni di spegnimento), il runlevel 2 è quello standard che viene eseguito quando si accende il computer, ed il 6 viene usato per il reboot.

Per rendere più sicuro il nostro firewall, dovremmo quindi prodigarci per evitare che durante l'accensione vengano avviati dei servizi che potrebbero essere utilizzati contro di noi.

Il metodo normalmente utilizzato è quello di eliminare i link simbolici (con un semplice rm) relativi a servizi che non ci interessano dalla directory relativa al runlevel utilizzato per avviare il sistema (normalmente "rc2.d", almeno in Debian). Alcune distribuzioni utilizzano metodi più esotici per gestire questi link simbolici, per cui vi conviene dare un'occhiata alla documentazione che vi è stata fornita.

In Debian per esempio, è possibile utilizzare lo script "update-rc.d", con qualcosa di simile a:

```
update-rc.d apache remove
```

per evitare che apache venga caricato durante l'avvio o lo shutdown. Un'altra alternativa potrebbe essere quella di rimuovere del tutto apache (probabilmente inutile se non lo si fa partire a boot-up), con il solito:

```
apt-get --purge remove apache
```

Per avere un elenco dei programmi attivi che offrono servizi in rete, potreste voler utilizzare il comando netstat, con

```
# netstat -npla |less
```

che vi dovrebbe indicare le porte rimaste aperte ed i relativi processi.

5.1.3 Demoni

In generale comunque, per i servizi che deciderete di lasciare attivi, ricordatevi dove possibile di:

- eliminare banner di benvenuto o comunque quelle informazioni che mostrino il nome e la versione dei programmi utilizzati.
- controllate mailing list di sicurezza, come per esempio bugtraq, per verificare che i demoni lasciati attivi non contengano bug pericolosi.
- non lasciate demoni attivi con privilegi di root.

Per ciò che offre servizi in rete, se vi voleste veramente divertire, potreste utilizzare delle porte diverse rispetto quelle standard: se metteste per esempio ssh sulla porta 42 piuttosto che sulla 54, potreste evitare tutti quegli scanner che mandano pacchettini in giro per la rete alla ricerca di particolari versioni di particolari software da poter facilmente bucare. Certo, questo puó creare parecchia confusione anche dal punto di vista dell'amministrazione (soprattutto se ogni volta utilizzate una porta diversa), ma a lungo andare potrebbe dimostrarsi una utile contromisura. Attenzione però che se il demone che state utilizzando contiene dei bachi di sicurezza, questo continuerà ad averne anche se deciderete di utilizzare una diversa porta.

Per cambiare impostazioni come la porta, i privilegi che il demone dovrà avere o per cambiare i banner di benvenuto, riferitevi alla documentazione specifica fornita col demone installato.

5.2 Ripulire il sistema

- Eliminate tutti i tool di compilazione (make, gcc...), gli header file, o tutti quei tool di debugging (gdb, objdump...) che potrebbero essere usati contro di voi. Se avete bisogno di ricompilare, potrete sfruttare una macchina di appoggio e i tool per creare i pacchetti messi a disposizione dalla vostra distribuzione. In questo modo, non rischierete di dimenticare tool pericolosi o sorgenti in giro.

- Cercate poi tutti quei programmi noti come "suid" root, cioè che possono prendere i privilegi dell'amministratore una volta caricati (find / -perm +4000), e togliete loro questa possibilità **valutando però caso per caso**. Ad esempio vorrete probabilmente lasciare /bin/login suid root, mentre a ping, mount & co. credo possa essere tolto tranquillamente questo privilegio con comandi del tipo "chmod ug-s nomefile".

- Configurate propriamente `fstab`, lasciando solo `/var` scrivibile e lasciando tutte le altre partizioni read-only, limitando persino la creazione di dispositivi o l'esecuzione di file suid root. Un esempio di fstab potrebbe essere il seguente:

```
proc          /proc     proc      defaults                        0 0
/dev/hda1     none      swap      sw                              0 0
/dev/hda5     /         ext2      ro,defaults,errors=remount-ro   0 1
/dev/hda2     /boot     ext2      ro,nosuid,noexec,nodev,nouser   0 3
/dev/hda7     /var      reiserfs  rw,nosuid,nodev,nouser          0 3
/dev/hda8     /home     reiserfs  rw,nosuid,nodev,nouser          0 3
```

  dove *nosuid* indica al kernel di non eseguire programmi con privilegi suid, *noexec* di non eseguire alcun programma, *nodev* di non consentire la creazione di dispositivi e infine *nouser* di non permettere agli utenti di montare o smontare la partizione indicata.Attenzione però che la vostra distribuzione potrebbe non essere troppo contenta di avere a che fare con dischi readonly, soprattutto per quanto riguarda la root. Una soluzione che ho visto funzionare abbastanza bene (salvo qualche directory) é stata quella di scrivere due script, mkro e mkrw, il primo per rendere la root read only ed il secondo per renderla read write, e di installarli in modo che venissero eseguiti rispettivamente come l'ultimo comando a startup ed il primo comando a shutdown. In Debian, è possibile fare questo copiando i due file nella directory `/etc/init.d` e dando i comandi "update-rc.d mkro start 99 2" e "update-rc.d mkrw stop 00 6".

  Così facendo, la partizione di root dovrebbe essere indicata come "rw" nel file fstab.

  Notate inoltre che in `/var` viene consentita l'esecuzione di programmi: questo perché la maggior parte delle distribuzioni utilizzano `/var` per conservare gli script di installazione/disinstallazione dei vari pacchettini che devono essere eseguiti.

  Infine, attenzione che la partizione che contiene la directory `/dev` deve essere montata come rw... una buona soluzione é quella di utilizzare o una partizione a parte, o una partizione creata da un ramdisk o shm, oppure utilizzare il devfs con il demone devfsd (soluzione sicuramente migliore).

  Per quanto riguarda `/tmp`, questa viene utilizzata da molti programmi e deve quindi essere scrivibile. Sempre che non abbiate creato una partizione indipendente per /tmp, una soluzione potrebbe essere quella di creare un symlink `/tmp` a `/var/tmp`, oppure sfruttare le nuove capacità del kernel 2.4 aggiungendo da qualche parte nei file eseguiti all'avvio un comando del tipo:

```
mount --bind /var/tmp /tmp
```

  che monterebbe in maniera molto particolare (estremamente simile ad un hard link) la directory `/var/tmp` su `/tmp`. Attenzione però che il FHS (lo standard per la struttura del file system), indica che in `/tmp` i file possono essere rimossi in qualsiasi istante dall'amministratore del sistema, e consiglia di ripulire /tmp durante i reboot. Al contrario, /var/tmp non deve essere ripulita durante i reboot e deve consentire la creazione di file "abbastanza persistenti". Per evitare problemi con la vostra distribuzione, è quindi consigliabile far puntare `/tmp` ad una sottodirectory di /var/tmp in modo che un'eventuale pulizia periodica di `/tmp` non rimuova tutti i file di `/var/tmp`. Per esempio, potreste fare qualcosa del tipo:

```
mkdir /var/tmp/removable/
mount --bind /var/tmp/removable/ /tmp
```

Piuttosto che

```
mkdir /var/tmp/removable/
cd /; ln -s /var/tmp/removable ./tmp
```

Infine, parlando di partizioni read only o read write, è bene dedicare un minimo di attenzione al file /etc/mtab. In questo file viene mantenuto l'elenco delle partizioni montate col comando "mount". In pratica, quando date il comando "mount" da solo, per vedere le partizioni montate, vi viene mostrato il contenuto di questo file. Il problema è che montando la root read only, mount non sarà in grado di aggiornare questo file e di capire quali partizioni sono montate, dando origine a degli errori piuttosto bizzarri. Una buona soluzione è quella di eliminare il file mtab completamente e di creare un link simbolico in /etc che punti al file /proc/mounts. Questo file mostra le informazione conosciute dal kernel a proposito dei dischi montati, e non può quindi contenere errori. In pratica date:

```
cd /etc
rm ./mtab
ln -s /proc/mounts ./mtab
```

- Per i file system ext2/3 potreste poi volere utilizzare il comando "chattr +i nomefile", per marcare i file a livello di filesystem come immodificabili. Questo è utile soprattutto perché si tratta di una modifica poco visibile (normalmente non si vede con ls -al) e non consente nemmeno a root di modificare il file, a meno che prima non esegua un comando come "chattr -i nomefile", sempre che chattr non sia stato preventivamente rimosso dal sistema :-).

- Date un'occhiata a /etc/crontab e alle varie directory, eliminando ciò che non serve. Ad esempio, su un sistema con dischi read-only non credo sia necessario aggiornare il database di "locate". Spesso poi, gli script eseguiti da cron interagiscono con il sistema e può capitare che possano essere imbrogliati costringendoli a fare cose non volute. Normalmente tutte le distribuzioni tendono ad evitare errori di questo tipo ed é molto raro trovarne, ma una verifica non costa nulla e può garantirvi un grado di sicurezza maggiore.

- Se sul sistema vi saranno altri utenti oltre root, date un'occhiata alle restrizioni imposte dalla pam (/etc/pam.d) agli utenti ed al file /etc/login.defs.

- Se avete un kernel relativamente recente, date un'occhiata al comando "lcap" (in Debian, installate l'omonimo pacchetto, con "apt-get install lcap"). lcap è un comando che vi consente di dire al kernel che **non** volete più usufruire di determinati servizi (capability, ovvero "capacità" del kernel). Una volta detto questo al kernel, le capability non potranno più essere acquisite dal kernel, a meno che la macchina non venga spenta e riavviata. Il fatto che il kernel di linux perda determinate "capacità" può quindi dare notevoli vantaggi dal punto di vista della sicurezza. Utilizzando il comando "lcap" senza alcun argomento, vi verrà data una lista delle "capacità" che il kernel può decidere di lasciare, come per esempio:

```
Current capabilities: 0xFFFFFEFF
   0) *CAP_CHOWN                1) *CAP_DAC_OVERRIDE
   2) *CAP_DAC_READ_SEARCH      3) *CAP_FOWNER
   4) *CAP_FSETID               5) *CAP_KILL
   6) *CAP_SETGID               7) *CAP_SETUID
   8)  CAP_SETPCAP              9) *CAP_LINUX_IMMUTABLE
  10) *CAP_NET_BIND_SERVICE    11) *CAP_NET_BROADCAST
  12) *CAP_NET_ADMIN           13) *CAP_NET_RAW
  14) *CAP_IPC_LOCK            15) *CAP_IPC_OWNER
  16) *CAP_SYS_MODULE          17) *CAP_SYS_RAWIO
  18) *CAP_SYS_CHROOT          19) *CAP_SYS_PTRACE
  20) *CAP_SYS_PACCT           21) *CAP_SYS_ADMIN
  22) *CAP_SYS_BOOT            23) *CAP_SYS_NICE
```

```
24) *CAP_SYS_RESOURCE          25) *CAP_SYS_TIME
26) *CAP_SYS_TTY_CONFIG
  * = Capabilities currently allowed
```

Ad esempio, con "lcap CAP_CHOWN" il kernel mollerebbe la capacità di far andare la chiamata di sistema CHOWN, rendendo impossibile il cambio di proprietario di un file. Eseguendo quindi il comando "chown", questo ci restituirebbe un errore del tipo "Operation not permitted". In breve, disattivare la capacità del kernel:

- CAP_CHOWN – inibisce il cambio di proprietario di un file, blocca il comando chown. Attenzione che, ad esempio in Debian, ogni volta che un utente completa il login viene fatto un chown sul dispositivo corrispondente alla sua console, onde evitare che altri (senza la sua autorizzazione) possano scrivere su tale dispositivo. Bloccare chown bloccherà quindi l'accesso a **tutti**, compreso l'amministratore, a meno che questa feture non venga disabilitata.
- CAP_NET_BIND_SERVICE – inibisce l'apertura di porte TCP o UDP inferirori alla 1024, impedendo il caricamento di nuovi servizi su porte standard.
- CAP_NET_ADMIN – inibisce: il cambio di configurazione delle interfacce di rete o la configurazione di nuove interfacce, le operazioni di amministrazione del codice di firewalling, nat, accounting, l'abilitazione del debugging sulle socket, la modifica delle tabelle di routing, l'abilitazione/disabilitazione del promiscuous mode e altre.
- CAP_SYS_MODULE – inibisce il caricamento o la rimozione di moduli.
- CAP_SYS_CHROOT – inibisce l'utilizzo di jail root, ovvero del comando chroot.
- CAP_SYS_PACCT – inibisce il cambio di configurazione del process accounting .
- CAP_SYS_BOOT – inibisce la possibilità di riavviare la macchina tramite comandi.
- CAP_SYS_RESOURCE – inibisce il cambio dei limiti di risorse imposti ai vari processi o delle quote imposte su disco (per fs ext2).
- CAP_SETUID – inibisce l'utilizzo di programmi suid.
- CAP_LINUX_IMMUTABLE – inibisce la possibilità di cambiare il flag immutabile tramite il comando "chattr"
- CAP_NET_RAW – inibisce la possibiltà di aprire PACKET o RAW socket, ovvero quelle socket che consentono a programmi come hping o nmap di scrivere direttamente i pacchetti sulle interfacce di rete.
- CAP_SYS_RAWIO – inibisce la possibilità di accedere direttamente ai dispositivi fisici. In pratica, non sarà più possibile accedere a /dev/hda, /dev/hdb e così via.
- CAP_SYS_PTRACE – inibisce l'utilizzo di "ptrace()", la chiamata che consente a comandi come strace, ltrace o gdb di seguire il comportamento di un processo.
- CAP_SYS_ADMIN – inibisce:
  * la configurazione di quote su disco, la creazione di dispositivi, l'amministrazione di /dev/random e urandom.
  * la configurazione dei messaggi inviati dal kernel a syslog, il cambio di nome di dominio, il cambio di hostname.
  * l'utilizzo di mount e umount, l'abilitazione disabilitazione dello swap, la configurazione di dispositivi raid ed il tuning dei parametri ide.

  e così via.

Per maggiori informazioni, potete dare un'occhiata a "lids.planetmirror.com" oppure cercare le "capacità" direttamente in google. L'utilizzo di "lcap" apre diversi nuovi orizzonti. Si potrebbe, per esempio, inserire in init.d uno script simile al seguente:

```
#!/bin/bash

lcap CAP_NET_BIND_SERVICE CAP_NET_ADMIN CAP_SYS_MODULE CAP_SYS_PACCT \
     CAP_SYS_RESOURCE CAP_SETUID CAP_LINUX_IMMUTABLE CAP_NET_RAW CAP_SYS_RAWIO \
     CAP_SYS_PTRACE CAP_SYS_ADMIN
```

in modo da disabilitare la maggior parte delle funzioni che potrebbero risultare pericolose. Allo stesso modo, utilizzando il comando "chattr +i" si potrebbero impostare come immutabili tutti i file coinvolti con il processo di boot, così da impedire che il sistema possa essere riavviato senza chiamare "lcap". Per avere ancora una maggior sicurezza, si potrebbe poi disabilitare il "reboot" del sistema, in modo che da remoto, anche ipotizzando che un attaccante possa riuscire a far sì che lcap non venga chiamato nel prossimo reboot, il sistema non possa essere riavviato senza la presenza fisica di una persona in sala macchine. In più, lasciare la capacità CAP_NET_ADMIN e CAP_NET_RAW impedirà all'attaccante di installare degli sniffer che impostino l'interfaccia in promiscuous mode ed il cambio delle regole di firewalling. Questo stesso script, potrebbe poi essere inserito in "runlevel 2", creando un link da /etc/rc2.d/S99-lcap a /etc/init.d/script_appena_creato, in modo che specificando qualcosa come "runlevel=3" dal prompt di lilo e di grub sia possibile accedere al sistema senza i limiti imposti da questo comando.

5.3 Rilevare le intrusioni

Beh, di tool per rilevare le intrusioni ce ne sono a decine disponibili sulla rete. Alcuni che vi posso consigliare sono:

- logcheck, per rilevare le differenze nei file di log in maniera più o meno intelligente, in modo da non aver mega e mega di log da leggere ogni giorno.
- tripwire o equivalenti (in debian sono disponibili debsums e debsig-verify, sicuramente molto utili e funzionali), per rilevare modifiche non volute ai file.
- msyslogd o syslog-ng, per essere sicuri che i log non vengano alterati (possono usare entrambi dei sistemi di checksumming crittografici per memorizzare i log). Inoltre, sono in grado di trasmettere in maniera abbastanza sicura i log da una macchina all'altra. Questo potrebbe darvi garanzie ancora maggiori sull'autenticità di questi log.
- snort, per rilevare scansioni o attacchi conosciuti dalla rete (viene considerato un IDS, ovvero, Intrusion Detection System).
- netacctd, per tenere traccia delle connessioni effettuate.
- arpwatch, per monitorare le workstation sulle varie reti ed eventualmente rilevare tentativi di arp spoofing.
- tcpdump, nel caso in cui il peggio accada, per loggare tutto il traffico e tentare di risalire al colpevole (creando, magari, uno script che lo carichi come demone con delle opzioni tipo -nei eth0 -s 2048 -w /var/log/traffic.dump).

E molti altri... (suggerimenti sono benvenuti).

5.3.1 Process accounting

Abilitando nel kernel l'opzione "General Setup/BSD Process Accounting" è possibile utilizzare il "process accounting". Il "process accounting" consente di mantenere delle statistiche su ogni comando eseguito sul

sistema, statistiche come l'utente che l'ha eseguito, la cpu che ha utilizzato, il tempo che e' stato eseguito e simili.

Per utilizzare il process accounting è possibile installare utility come "acct", "lastcomm" ed "sa", in Debian contenute tutte nel pacchetto "acct". Questo stesso pacchetto si occupa durante il processo di boot di chiamare "accton nomefile.log", che altro non fa che abilitare il process accounting dicendo al kernel di salvare le informazioni acquisite nel file di log indicato, generalmente "/var/account/pacct".

Una volta abilitato, sarà possibile utilizzare utility come "sa", per avere delle statistiche sugli ultimi comandi eseguiti e "lastcomm" per vedere i comandi eseguiti da ogni utente, divisi per tempo di esecuzione, terminale da cui sono stati lanciati e da alcuni flag indicanti come il comando sia stato eseguito.

6 Pericoli – ovvero, da cosa dovrà difendervi il firewall

Per rendere la trattazione più semplice, è meglio chiarirsi le idee fin dall'inizio. Il firewall che andrà a proteggere la vostra rete sarà in grado di filtrare il traffico secondo delle regole da voi stabilite. Lo scopo di queste regole sarà quindi principalmente quello di:

1. proteggere i vostri computer e i vostri server, rendendo accessibili solo alcuni servizi dall'esterno della vostra rete.
2. fare in modo che gli utenti della vostra rete non facciano operazioni non consentite.
3. evitare che il traffico consentito possa provocare dei danni. Per spiegarmi meglio, se aveste un web server interno alla vostra rete disponibile anche tramite internet, dovreste consentire il traffico sulla porta 80 ma al contempo fareste il possibile per evitare attacchi come syn flood o simili (vedi prossima sezione), conosciuti come "DoS" o "DDoS", rispettivamente "Denyal of Service" e "Distributed Denyal of Service", attacchi che mirano a sovraccaricare o a interrempore un servizio da voi offerto.
4. evitare che i vostri server o i vostri client possano essere utilizzati da qualcun altro per sferrare degli attacchi a danni di terzi.
5. evitare il più possibile i danni che vi potrebbe arrecare un attaccante nel caso riuscisse ad impossessarsi di una delle vostre macchine o dei vostri server.

Per quanto riguarda i primi 3 punti, non credo di aver detto nulla di originale. Normalmente lo scopo di un firewall è proprio quello o di proteggere servizi da noi offerti (o utilizzati internamente alla nostra rete), o limitare ciò che i nostri utenti possono fare.

Gli ultimi due punti, invece, vengono spesso sottovalutati o completamente ingorati, pur essendo fondamentali per almeno due motivi:

1. Per evitare un effetto "domino", per evitare cioè che "caduto" (bucato) un server, questo possa essere sfruttato per far "cadere" tutti gli altri. Spesso infatti, quando una ditta dispone di più server, vengono impostate delle politiche meno restrittive (se queste vengono impostate) per le comunicazioni da un server all'altro che potrebbero consentire ad un attaccante di bucare una macchina e da lì bucare tutte le altre.
2. Per evitare che la vostra macchina possa essere utilizzata per sferrare attacchi a terzi. Se tutti utilizzassero delle politiche di questo tipo sarebbe molto più facile tracciare eventuali attaccanti.

Una configurazione normalmente utilizzata che tiene in considerazione tutti i punti sopra elencati fa uso di quella che viene detta una DMZ, ovvero di una "Zona Demilitarizzata".

In pratica la rete di un'azienda viene divisa dal punto di vista hardware in tre segmenti, ed il firewall viene dotato di 3 schede di rete (ognuna collegata ad un segmento). Di queste schede, una viene collegata alla rete interna (utilizzata, per esempio, dai dipendenti), una collegata ad internet mentre l'ultima ad una piccola rete che conterrà soltanto i vostri server.

In questo modo, il firewall sarà in grado di proteggere i vostri server sia da attacchi provenienti dalla rete interna (e, da questo punto di vista, ci sono stati casi davvero eclatanti) sia da attacchi provenienti da internet.

Allo stesso modo, sarà in grado di proteggere la vostra rete interna da attacchi provenienti sia dai vostri server che da internet, e di proteggere internet da attacchi provenienti sia dai vostri server che dalla vostra rete interna. Il fatto che *sia in grado* non implica però che lo faccia sul serio. Spesso viene sottovalutata la pericolosità di attacchi provenienti dai propri server o dalla propria rete interna e vengono scritte delle regole meno restrittive o non vengono scritte affatto. Questo è un errore sicuramente da evitare. Un altro errore da evitare è quello di non sfruttrae le capacità di firewalling sui server, o di allentare le politiche di sicurezza "visto che c'è il firewall": è vero che il firewall c'è e funziona, ma essere un po' paranoici il più delle volte non guasta. Può capitare, per esempio, di fare una modifica sul firewall dimenticandosi del tale server, piuttosto che cambiare lo spinotto a cui era collegato.

6.1 Assunzioni errate

Esistono poi alcuni errori che è facile commettere o miti che è bene sfatare prima di fare delle assunzioni errate a proposito della propria rete. Tenete quindi ben presente che in una rete:

- **tutti i dati possono essere falsificati** – una "scheda di rete" non è altro che un dispositivo fatto per scrivere su di una rete, un po' come una testina è in grado di scrivere su dei dischi magnetici. I pacchetti non sono altro che dati che vengono scritti su di una rete, e possono essere quindi falsificati o inventati di sana pianta. Nessuno mi impedisce di "scrivere" con la mia scheda di rete dei pacchetti che indicano come mittente il computer di qualcun altro o delle informazioni assolutamente insensate. Scrivete quindi sempre delle regole che vietino o consentano anche ciò che può sembrare ovvio. Per esempio, se avete una intefaccia collegata alla rete 10.0.0.0/8, potreste scrivere come prima regola "vieta tutto ciò che viene dalla determinata scheda di rete ma che non ha 10.0.0.0/8 come indirizzo del mittente". Può sembrare stupido, ma questo vi potrebbe evitare un sacco di problemi. Cosa succederebbe ad esempio se riceveste dei pacchetti provenienti da 127.0.0.1? Con queste regole verrebbero immediatamente scartati, ma senza (e con l'rp_filter disabilitato, vedi prossima sezione) è difficile valutare cosa succederebbe (soprattutto su sistemi non linux – 127.0.0.1 è un indirizzo di loopback).

- **un computer collegato ad una lan è in grado di vedere tutto il traffico in transito dalla rete** – se *foo* è collegato allo stesso switch di *bar*, *foo* sarà in grado di vedere (sniffare, da "fare sniffing") tutto il traffico generato o ricevuto da *bar*, a partire dalle email fino ad arrivare alle password per accedere ai siti ftp preferiti. E' credenza comune che gli switch siano più sicuri degli hub e che non consentano questo tipo di operazioni. Si tratta però di un'assunzione il più delle volte errata. Per maggiori informazioni vi consiglio di dare un'occhiata alla sezione dedicata agli switch.

- **è possibile fare hijacking e spoofing** – spoofing consiste nel fingersi qualcun altro. In una rete locale, è possibile fare spoofing molto facilmente, per esempio con quello che viene definito "arp spoofing". Anche volendo utilizzare tecniche di spoofing più "classiche" basate sul protocollo tcp e ip non esistono molti problemi: su una rete locale, potendo vedere i pacchetti, non c'È bisogno di indovinare alcun parametro, e non è quindi necessario utilizzare tecniche di "blind spoofing", notoriamente più complesse. L'hijacking consiste invece nel "dirottare" le connessioni di qualcun altro, dopo che sono state stabilite. L'utilità dell'hijacking si dimostra ad esempio con protocolli non cifrati dove l'autenticazione viene effettuata in modo sicuro. Facendo sniffing non sarebbe possibile rubare la password, mentre utilizzando

l'hijacking si potrebbe rubare la connessione dopo che l'amministratore (o il nostro bersaglio) ha effettuato il login con successo, evitando il problema di aver bisogno di una password.

- **il protocollo udp è insicuro** – se il tcp fa uso di syn e ack per verificare e rendere le connessioni più sicure e affidabili, l'udp non fa uso di nessuno di questi sistemi. E' quindi estremamente facile falsificare delle trasmissioni udp, anche al di fuori di una lan.

- **il dns non è sicuro** – Il protocollo dns fa uso del protocollo udp. E' quindi difficile considerarlo un protocollo sicuro (nonostante qualche precauzione venga presa), soprattutto in una lan, dove è estremamente facile falsificare dei pacchetti. In più spesso si parte dall'assunzione che se la comunicazione tra client e dns è sicura allora i dati forniti dal dns sono corretti. Anche in questo caso esistono delle tecniche (usate per lo più in passato o con software dns di pessima qualità - compresi quelli inclusi in alcuni sistemi operativi) che consentono di inserire nel dns informazioni false (cache poisoning). Infine, l'amministratore del vostro dns (o qualcuno in grado di gestire il dns da voi utilizzato), può modificare quasi a piacimento le associazioni tra nomi di dominio ed indirizzi ip forniti da tale dns. Non si dovrebbe quindi mai fare affidamento sull'autenticità e correttezza delle informazioni fornite da un dns.

- **tutti i dati, se non cifrati, possono essere intercettati e letti da chiunque si trovi tra il mittente e il destinatario**

- **se un protocollo è cifrato, non è detto che sia sicuro** – un problema di internet e delle reti in generale non è soltanto il fatto che i dati possono essere facilmente intercettati. Anche ipotizzando che tutte le connessioni fossero cifrate, si creerebbe un problema di *autenticazione*. Per fare un esempio, se A deve mandare dei dati a B in maniera sicura, può creare un canale cifrato da A a B. Ma come fa A ad essere veramente sicura dell'identità di B? Se io fossi C, interessato ai dati trasmessi, potrei fingermi B (con tecniche di spoofing o simili). In questo caso, la connessione tra A e B sarebbe ancora cifrata, ma i dati arriverebbero in realtà a C. Il problema, in questo caso, è l'autenticazione del server. Esistono diverse soluzioni per questo problema. Con ssh, è consigliabile utilizzare sempre crittografia asimmetrica (le famose chiavi di ssh) anziché digitare le password e verificare sempre il "fingerprint", un numero che rappresenta la chiave del computer remoto e mostrato ad ogni connessione. Con protocolli come l'http, l'ftp, o la posta elettronica (sia pop che imap e smtp), è possibile fare uso di "certificati" emessi da particolari autorità (autorità che si fanno garanti della identità dell'utilizzatore del certificato) che provino l'identità del server.

  In Italia purtroppo, per risparmiare qualche euro, si è instaurata la pratica di generare da sé i propri certificati senza utilizzare quelli emessi da particolari autorità anche in strutture che per dimensioni, per necessità, e per competenze tecniche dovrebbero per lo meno porsi qualche problema (parlo di università, enti pubblici, e di alcune strutture che costituiscono lo scheletro di internet in Italia). Vorrei pertanto approfittare di questo documento per evidenziare che se un certificato non è emesso da una autorità riconosciuta o da un'autorità terza di cui avete ottenuto i certificati in maniera sicura (**non** scaricandoli da internet), la validità del certificato stesso dovrebbe essere considerata **nulla**. Non ignorate quindi la finestrella di IE o Mozilla che vi si presenta davanti dicendo che il certificato del sito non è valido: chiunque potrebbe aver generato tale certificato (man 1 openssl) ed essersi messo in mezzo tra voi ed il vostro sito. Non siate tolleranti e mandate email all'amministratore di tale rete finché non avrà acquistato un valido certificato!

  Protocolli di autenticazione come kerberos invece, partono dal principio che non è solo l'utente a dover provare la sua identità ma anche il server remoto. In questo caso, se il server non è chi dichiara di essere non sarà in grado di decifrare i dati dell'utente stesso.

- **un firewall non è in grado di garantirvi la sicurezza** – se sulla vostra rete usate sistemi operativi notoriamente insicuri, potreste trovarvi facilmente dei troiani (virus particolari) installati che consentono a terzi di controllare i vostri computer da remoto, senza che il vostro firewall possa farci nulla. Una politica di sicurezza deve quindi comprendere molto più di un singolo firewall.

- **un sistema non è sicuro solo se non possono rubarvi dati** – per esempio, potreste trovarvi nelle condizioni di avere un servizio critico per la vostra azienda offerto da una macchina estremamente sicura. Cosa succederebbe però se un attaccante riuscisse a sovraccaricarla di lavoro fino a bloccarla, pur senza riuscire ad entrarvici? Vi posso garantire che il vostro boss non sarebbe affatto felice... In alcuni ambiti poi, un'interruzione di servizio potrebbe portare molti più danni di quanti potrebbe causarne un'intrusione.

- **esiste sempre qualcuno più bravo, più competente o più furbo di voi** – per quante difese possiate approntare, esisterà sempre quel bug di cui non siete a conoscenza o quella persona in grado di "bucare" il vostro sistema. Se avrete usato per bene le carte a vostra disposizione, sarà per lui una dura partita ed il più delle volte mollerà prima di raggiungere l'obiettivo. Sarete inoltre in grado di rendervi conto di quello che sta succedendo e dei problemi che si sono venuti a creare.

- **non fidatevi** – sebbene possa sembrare un'osservazione faziosa e quasi paranoica, ricordatevi che avere il codice sorgente dei programmi che state utilizzando vi mette *direttamente* nelle condizioni di poter verificare l'assenza di backdoor (porte sul retro, accessi lasciati dai programmatori) o l'assenza di bug ed errori che potrebbero compromettere la sicurezza della vostra rete.Mettere a disposizione il codice sorgente di un determinato prodotto può quindi essere considerato come un atto di fiducia nei vostri confronti e costituire garanzia del buon operato dei programmatori coinvolti, che dimostrano di non aver nulla da nascondere né a voi "clienti" né ad eventuali attaccanti. Una cassaforte non dovrebbe essere considerata sicura perchè il suo meccanismo è tenuto segreto (un meccanismo si può sempre studiare), ma perchè pur rendendone pubblico il funzionamento *nessuno è in grado di aprirla* se non utilizzando la combinazione impostata.

  Quando comprate un apparecchio come uno switch, un router o un firewall, provate a cercare sul sito del produttore le procedure indicate per il recupero delle password dimenticate: alcune volte vi verrà detto di rendere l'apparecchiatura raggiungibile tramite internet e di contattare il servizio di assistenza (non è uno scherzo). Questo dovrebbe dirla lunga sull'assenza di backdoor e sulla sicurezza del prodotto.

  Infine, se la vostra preoccupazione maggiore è la sicurezza, ricordatevi che spesso questo non è l'interesse maggiore di chi vi vende un prodotto (e non sta a me ricordarvi che viviamo in un mondo dove l'economia è basata sul marketing e sulla massimizzazione dei guadagni). Non accontentatevi di belle parole o pubblicità accattivanti: se vi verrà rilasciato il codice sorgente sarete in grado di verificare in prima persona la qualità del prodotto.

Se può tranquillizzarvi, in questo momento non mi viene più in mente nulla da aggiungere. Spero che questa notte riusciate a dormire sonni senza incubi...

Le solite raccomandazioni, ormai, credo che siano superflue: non scambiate mail con password (è molto più facile falsificare una mail che non intercettare una comunicazione), scegliete password che non siano contenute in un dizionario e che contengano caratteri maiuscoli e minuscoli, cifre e simboli, utilizzate tecnologie di firma digitale e verificate l'identità delle persone che vi richiedono informazioni sensibili, magari richiamandole per telefono o utilizzando qualche sorta di Web Of Trust.

6.2 Sicurezza degli switch

Alcuni paragrafi fa avevo parlato di switch e della loro sicurezza. La differenza principale tra uno switch ed un hub, è che gli hub rimandano tutti i dati ricevuti a tutti i computer collegati all'hub stesso, mentre gli switch si limitano ad inviare i dati ricevuti ai soli computer coinvolti in una trasmissione. Teoricamente, quindi, utilizzando uno switch non sarebbe possibile in una rete effettuare lo "sniffing" dei dati trasmessi nelle connessioni tra altri computer.

In pratica però, gli switch sono in grado di isolare le connessioni mantenendo in memoria una sorta di tabella che associa ad ogni cavo un computer. Per cui, quando due computer devono comunicare, lo switch consulta

questa tabella ed in base a questa sceglie su quali cavi dovrà trasmettere il traffico.

Questo modello presenta diversi problemi:

- Prima di tutto, questa tabella viene creata in base al traffico che inizialmente viene visto transitare. Per cui, generando traffico fasullo, è possibile imbrogliare lo switch facendogli credere che un altro computer è collegato al cavo cui si è in realtà collegati (e se su due cavi risulta uno stesso computer, lo switch, normalmente, riinvierà i dati trasmessi su entrambi i cavi).
- In secondo luogo, gli switch sono collegabili in cascata. Questo comporta che nella tabella nella memoria dello switch ad ogni cavo possa essere associato più di un computer. La memoria di uno switch, però, ha dimensione limitata e la maggior parte degli switch si comporterà esattamente come un hub nel caso esaurisse la memoria e non fosse più in grado di mantenere tale tabella.Questo principalmente per evitare malfunzionamenti in reti di grosse dimensione.
- In terzo luogo, è possibile collegare più switch tra di loro utilizzando collegamenti multipli. Per esempio, una rete di medie dimensioni potrebbe utilizzare qualcosa come 20/30 switch, tutti collegati tra di loro. Se fossero tutti collegati in cascata, il guasto di uno switch comporterebbe il malfunzionamento di buona parte della rete (probabilmente, dividerebbe la rete in due parti).Normalmente quindi, si stabiliscono più connessioni tra gli stessi switch (stendendo più cavi) in modo che il guasto di un solo apparecchio comprometta la funzionalità dei soli computer connessi a tale apparecchio.

  Per consentire l'utilizzo di percorsi multipli, gli switch devono costruirsi una sorta di "mappa" della rete in modo da sapere che percorso far fare ai vari pacchettini per evitare che si vengano a creare dei loop (pacchetti che continuano a rimbalzare da uno switch ad un altro senza mai arrivare a destinazione). Per costruire questa mappa, gli switch devono comunicare tra di loro con un protocollo chiamato spanning tree protocol (IEEE 802.1d, scambiandosi quelli che vengono detti BPDU). In questo caso, quindi, un computer collegato allo switch potrebbe falsificare i dati del protocollo spanning tree in modo da far credere di essere a sua volta uno switch e di costituire il percorso più breve per raggiungere determinati computer, in modo da intercettare il traffico di tali computer.
- Infine, gli switch possono essere solitamente configurati tramite interfacce web o telnet, che non fanno uso di alcun protocollo crittografico o di particolari protezioni, se non un username ed una password.Allo stesso modo vengono spesso utilizzati protocolli come l'snmp ("Simple Network Management Protocol" - le prime versioni del protocollo, definito dall'RFC1157), che non richiedono nemmeno una password, se non un semplice nome di comunità.

Se la sicurezza della vostra rete dipende molto dalla sicurezza dei vostri switch, potreste considerare l'utilizzo di:

- VLAN (IEEE 802.1q) – questa tecnologia vi consente di assegnare un numero ad ogni porta di uno switch. Una volta fatto questo, solo le porte con lo stesso numero potranno comunicare tra di loro. In più, i collegamenti tra uno switch ed un altro passeranno per quelli che alcuni produttori chiamano "trunk", ovvero dei collegamenti dove tutti i dati, indipendentemente dal numero di VLAN, possono essere trasmessi. Attenzione però, che alcuni switch supportano una modalità di apprendimento, dove ad una porta è assegnata una particolare VLAN in base alle caratteristiche dei pacchetti trasmessi. Un utente malizioso, con questa funzionalità abilitata, potrebbe fingersi un altro switch e farsi inviare tutti i pacchetti delle VLAN che gli interessano. Inoltre, linux supporta il VLAN tagging (802.1Q), ovvero è in grado di creare delle interfacce virtuali su diverse VLAN, senza bisogno di hardware aggiuntivo.Un altro vantaggio delle VLAN consiste nel fatto che alcuni switch possono essere configurati in modo che esista una VLAN amministrativa: in questo caso, le configurazioni potranno essere cambiate e visualizzate soltanto da computer connessi ad una di queste VLAN.

- Alcuni switch supportano delle funzionalità di sicurezza relative alle porte ethernet. Per esempio, alcuni switch possono essere configurati in modo che se su una determinata porta viene visto un indirizzo ip o un mac address (o una coppia dei due) diversi da quelli impostati, la porta viene spenta. In più, possono essere normalmente indicate delle associazioni statiche tra porte ed indirizzi ip o mac address, senza che per questo venga bloccata la porta.
- A volte, lo spanning tree protocol può essere semplicemente disabilitato o perchè l'hardware utilizzato consente di impostare dei percorsi alternativi manualmente, o perchè non è necessario per la topologia della rete. In altre situazioni, alcuni switch supportano restrizioni (normalmente basate su VLAN) sull'invio e sulla ricezione di questi pacchetti.

Ricordatevi infine che un router od uno switch potrebbero costituire un obiettivo molto più succulento di quanto possiate pensare: controllando un router, infatti, si potrebbe creare un tunnel tra la vostra rete ed una rete bersaglio, piuttosto che impostare delle regole per deviare o fare delle modifiche al traffico.

7 Preparare il kernel ad andare in rete – overview di /proc

Adesso che siamo consapevoli di quello che ci può accadere e di quali siano i nostri obiettivi, iniziamo dando un'occhiata alla directory `/proc/sys/net`. In questa directory ci sono dei file e delle directory che ci consentono di modificare il comportamento della nostra macchina in rete, che ci consentono cioè di modificare molti dei parametri usati dal kernel per prendere le decisioni sui pacchetti che riceve o che deve trasmettere.

In pratica, ad ogni file contenuto in questa directory corrisponde una variabile, il cui valore regola il funzionamento di una determinata parte del kernel.

Le directory, invece, sono utilizzate semplicemente per raggruppare i file e per renderne più facile l'utilizzo.

7.1 Utilizzo dei file in /proc

Per conoscere il valore delle variabili impostate in tali file, è sufficiente leggere il contenuto del file stesso, utilizzando, per esempio, un "cat nomefile".

Per modificare tale valore, è sufficiente scrivere il nuovo valore all'interno del file utilizzando un comando simile a "echo nuovovlaore > nomefile".

Per convenzione, una variabile che può assumere come valore "vero" o "falso" oppure "attivo" o "inattivo" avrà rispettivamente come valori 1 o 0.

Per esempio, per disabilitare l'ecn (di cui si parlerà tra poco), basterà un comando simile a:

```
echo "0" > /proc/sys/net/ipv4/tcp_ecn
```

Le modifiche fatte in questo modo, però, non vengono conservate tra un reboot e l'altro. Per renderle definitive, è quindi necessario inserire i relativi comandi in un file eseguito all'avvio della macchina oppure usare il file di configurazione `/etc/sysctl.conf`. In questo file dovrete scrivere la "variabile" seguita da un "=" e dal valore che volete assegnare. Il nome della variabile corrisponde al nome del file nella directory di proc, togliendo però "/proc/sys" e sostituendo dei "." alle "/". Per esempio, per disabilitare per sempre l'utilizzo dell'ecn, sarà sufficiente aggiungere la riga:

```
net.ipv4.tcp_ecn=0
```

nel file `/etc/sysctl.conf`.

7.2 Alcuni file importanti

Alcune delle variabili in /proc/sys/net sono particolarmente importanti dal nostro punto di vista in quanto ci consentono di proteggere il kernel da diversi tipi di attacchi.

Di base, la directory che più ci interessa di `/proc/sys/net` è sicuramente `ipv4`.

In quest directory, dovreste trovare alcuni file tra cui:

- *icmp_echo_ignore_broadcasts* – questo file vi consente di dire al kernel se accettare o ignorare dei "ping" inviati ad indirizzi di broadcast. Un ping non è altro che il pacchetto generato dal comando "ping" per vedere se un computer è in funzione o meno. In questo caso, un valore di 0 (falso) dice al kernel di non ignorare i ping di broadcast, mentre un valore di 1 (vero) di ignorarli. Normalmente, è meglio impostare questo file ad "1". Provate ad immaginare cosa potrebbe succedere se una persona qualsiasi mandasse un ping (aka icmp echo request) a 16 milioni di computer, utilizzando come destinazione un indirizzo di broadcast e come mittente l'indirizzo ip del vostro computer.Con una cinquantina di byte, vi manderebbe contro circa 16 milioni di risposte al ping... (fortunatamente, la maggior parte dei computer in rete non rispondono a ping di broadcast).

  Questa variabile viene principalmente utilizzata all'interno di net/ipv4/icmp.c:icmp_rcv.

- *icmp_ignore_bogus_error_responses* – disabilita o abilita il logging di risposte errate mandate da alcune apparecchiature di rete. Utile soltanto per evitare di riempire i vostri log. Fa riferimento principalmente all'RFC1122 e alla funzione icmp_unreach chiamata da icmp_rcv in net/ipv4/icmp.c.

- *ip_forward* – questo file, invece, vi consente di abilitare o disabilitare il "forwarding". Il forwarding consiste nel consentire a linux di "copiare" i dati ricevuti da un'interfaccia di rete ad un'altra. Senza il forwarding abilitato, non potremmo utilizzare linux come firewall ed i pacchettini non potrebbero passare da un'interfaccia all'altra.

- *ipfrag_** – sebbene i default siano normalmente più che validi, è bene spendere due parole sui terribili frammenti. Alcune reti, a livello hardware, non riescono a trasmettere più di un numero di byte limitato per volta. Una rete ethernet per esempio, non consente di trasmettere più di 1500 (circa) bytes per volta (questo limite viene anche detto MTU, ovvero Max Transfer Unit). Esiste per questo un sistema per spezzettare i pacchetti ed inviarli come tante unità indipendenti. Questi vengono normalmente chiamati frammenti e possono creare diversi problemi:

  – Per esempio, inviando frammenti sufficientemente piccoli la parte iniziale del pacchetto che contiene le informazioni sul protocollo tcp o udp viene suddivisa su più pacchetti. Queste informazioni però, vengono utilizzate proprio dai firewall per capire la destinazione e lo scopo del pacchetto. In passato, alcuni firewall facevano semplicemente passare questi frammenti, altri si inceppavano, mentre altri ancora li buttavano via. L'approccio utilizzato attualmente da linux è quello di assemblare lui direttamente come firewall tutti i frammenti. In pratica, linux deciderà se far passare o meno un pacchetto solo dopo averlo completamente riassemblato. Questo proteggerà i nostri server anche da altri tipi di attacco.

  – Al contrario, dividendo il pacchetto in frammenti sufficientemente grossi, l'header tcp/udp viene inserito solo nel primo frammento. Così, se arrivano prima frammenti successivi al primo, il firewall deve decidere se far passare o meno il pacchetto senza avere tutte le informazioni necessarie a disposizione. Anche in questo caso, far assemblare i pacchetti al firewall può essere una buona soluzione.

  – Un altro esempio è quello di alcuni attacchi che prendevano dei grossi pacchetti, li spezzettavano in frammenti molto piccoli e li mandavono poi ad una macchina bersaglio in un ordine completamente casuale. Questa macchina si doveva quindi prodigare per ricomporre il puzzle riordinando e

riunendo i pacchetti nel corretto ordine, sovraccaricando notevolmente la cpu arrivando persino a bloccare la macchina.

Per rimediare, in linux si utilizzano proprio i file ipfrag_*. In pratica, si stabiliscono dei limiti oltrepassati i quali linux si "arrenderà" nel tentare di riordinare i pacchetti, evitando di bloccarsi su questo compito. I valori di default di tali limiti sono ben tarati, anche se potrebbe rendersi necessario abbassarli, a seconda della tipologia di macchina utilizzata e a secondo dell'attacco cui si è soggetti.

Capita raramente poi, che per migliorare le prestazioni di alcuni servizi, come per esempio NFS, tali limiti debbano essere alzati.

– Alcuni vecchi attacchi attualmente praticamente inoffensivi erano basati sul produrre frammenti che si sovrapponessero ("overlapping fragments") in modo da portare i sistemi operativi a sbagliare il calcolo della memoria da utilizzare, bloccando il sistema (CERT CA-1997-28).

Il significato dei file è quello classico di tutti i meccanismi basati su un limite (threshold). Quando la memoria utilizzata per riassemblare i frammenti supera ipfrag_high_thresh questi vengono bloccati fintanto che il valore non torna sotto ipfrag_low_thresh. ipfrag_time indica per quanto tempo al massimo un frammento può essere mantenuto in memoria. Questi limiti sono utilizzati principalmente in net/ipv4/ip_fragment.c:ip_defrag ed ip_evictor.

- *tcp_ ecn* – questo file regola l'utilizzo dell'ecn, ovvero "Explicit Congestion Notification protocol". Non che sia un rischio per la sicurezza, ma spesso firewall vecchi non conformi agli standard (RFC3168) tagliano via tutte le connessioni che utilizzano questo protocollo. Se da linux non riuscite a collegarvi a dei siti che invece non avete problemi a raggiungere con altri sistemi operativi, probabilmente dovete disabilitare l'ecn. Sicuramente quindi, almeno finché le cose non saranno cambiate, vi conviene tenerlo sempre disabilitato.

- *tcp_ max_ syn_ backlog e tcp_ syncookies* – un altro tipo di attacco, abbastanza pericoloso e abbastanza in voga di questi tempi viene detto syn flood, ovvero inondazione di syn (CERT CA-1996-21). Il giochetto anche in questo caso è molto semplice: viene inviato il pacchetto tcp per iniziare una connessione ma non ne viene mai confermata l'apertura, costringendo il sistema operativo a ricordarsi di questa connessione in attesa che una conferma venga inviata.Facendo un paio di grossolani conti, considerando che esistono 65535 porte e che per ogni porta devono poter essere accettate più connessioni, è abbastanza evidente che il kernel di un qualsiasi sistema operativo non può permettersi di ricordarsi tutti i pacchetti di inizio delle connessioni ricevuti. In pratica, per ogni porta viene creata una coda di dimensioni molto piccole (si parla di numeri nell'ordine delle decine o centinaia). Quando questa coda è piena, la porta non può più accettare pacchetti. A peggiorare la situazione, le rfc indicano un timeout abbastanza alto (nell'ordine dei minuti) prima che il kernel possa buttare via questi pacchetti, svuotando la coda e dedicandosi ad altro. E' quindi abbastanza facile dimostrare che un attaccante, mandando pochi kilobyte di syn al secondo con un semplice modem 56k, può rendere inutilizzabile una porta di un qualsiasi server, senza nemmeno alzare il carico della cpu né saturare la banda.

  Le soluzioni sono diverse, ma poche si sono rivelate veramente valide:

  – alcuni firewall limitano il numero massimo di syn al secondo inviati ad ogni server protetto, facendo in modo che quando questo limite viene sorpassato vengano fatti passare "equamente" (e ci sono diverse discussioni su cosa si debba intendere per "equamente") syn inviati da mittenti diversi. Il problema è che essendo sufficiente un basso numero di syn al secondo per bloccare un server, è difficile riconoscere un attacco da un comportamento "legale". Esistono diversi algoritmi per gestire queste situazioni, e su internet sono disponibili i benchmark sull'efficacia di questo tipo di protezione, divisi per firewall e per rivenditore (cercate con google!). Non si tratta comunque della soluzione migliore ed è stato mostrato che con attacchi di particolari dimensioni si tratta di una difesa molto poco efficace che può tagliare facilmente del traffico che invece dovrebbe essere consentito.

- un approccio più efficace adottato da altri firewall si è rivelato quello di far gestire le connessioni dal firewall stesso. In pratica, quando il firewall riceve un syn, questo "si finge" il server fino al ricevimento dell'ack di conferma della connessione. A questo punto, stabilisce una connessione "vera" con il server fingendosi il client dopodiché fa continuare la connessione come se il client ed il server si parlassero direttamente.In questo modo, un syn flood andrebbe a colpire solo il firewall proteggendo molto efficacemente i server.

  Firewall di questo tipo sono poi strutturati in modo da resistere molto bene a dei syn flood anche con un numero molto alto di syn al secondo, utilizzando tabelle più grosse di quanto si possa permettere un normale server e creando delle infrastrutture dinamiche in grado di gestire tali situazioni.

- molto originale invece è l'approccio utilizzato da linux e da molti altri sistemi. Il sistema viene chiamato syn cookie, "biscottini nel syn". Tornando a parlare di three way handshake e tcp, quando un syn viene ricevuto (e per syn si intende un pacchetto che contenga il numero iniziale di sequenza, o numero di sincronia, e il cui syn flag sia settato), il sistema che lo riceve deve rispondere a sua volta inviando un syn con un ack. Questo secondo syn è normalmente un numero casuale difficile da indovinare ed è proprio dopo aver inviato quest'ultimo che il sistema deve rimanere in attesa del famoso ack. Il sistema utilizzato dai syn cookies consiste nel prendere un numero non casuale da inviare che contenga quelle informazioni (o parte di quelle informazioni) che verrebbero memorizzate nella famosa coda (se ci fosse spazio a disposizione...). In pratica, una volta che la coda è piena, le informazioni che dovrebbero essere memorizzate vengono inviate nel syn ack di risposta. Se l'handshake viene completato correttamente dal client inviando l'ack finale (il terzo passo dell'handshake), le informazioni vengono estratte dal pacchetto ricevuto (l'ack di risposta viene sempre calcolato a partire dal syn, ed è quindi possibile partendo da un ack ricalcolare il syn originale), non risentendo quindi del problema.Il syn però, deve continuare ad essere difficilmente indovinabile ed aumentare di pacchetto in pacchetto (non avrebbe ragione di esistere altrimenti, e sono condizioni imposte dall'rfc 793). Vengono quindi normalmente utilizzate delle funzioni one-way o degli hash che rispecchino queste caratteristiche, e l'informazione inviata si riduce normalmente in un "connessione inizializzata correttamente" (per maggiori informazioni, potete vedere il file `net/ipv4/tcp_ipv4.c`, in particolare la funzione tcp_v4_conn_request, oppure il file `drivers/char/random.c` funzione secure_tcp_syn_cookie per sapere come un syn cookie viene calcolato). Il problema è che è stato dimostrato che in particolari condizioni con particolari versioni del kernel linux era possibile indovinare questi syn e fare spoofing delle connessioni tcp. Sebbene attacchi di questo tipo siano stati raramente (se mai) utilizzati (non mi riferisco allo spoofing in generale, pratica largamente utilizzata, ma all'indovinare i syn grazie a questo bug nella creazione dei syn cookies), è bene abilitare i syn cookies solo su kernel relativamente recenti (dal 2.4.6 in su funzionano sicuramente correttamente), o in particolari condizioni (quando si è sotto attacco, per esempio).

  Ultima cosa da dire a proposito dei syn cookies è che questi vengono considerati da alcuni violare lo standard del tcp (ci sono state diverse discussioni in proposito, si veda per esempio il sito http://cr.yp.to/syncookies.html, di Dan Bernstein) e possono quindi introdurre dei problemi. I syn cookies, se abilitati, verranno quindi utilizzati da linux soltanto nel caso in cui la famosa coda si riempia.

Tornando a parlare dei file in /proc, tcp_max_syn_backlog consente di specificare il numero massimo di connessioni che possono rimanere in coda per ogni porta (normalmente è impostato a 128 su sistemi con meno di 32 mega di ram e a 1024 su tutti gli altri sistemi - net/ipv4/tcp.c:tcp_listen_start, net/ipv4/tcp-ipv4.c:tcp_v4_conn_request e tcp_synq_is_full) mentre tcp_syncookies consente di abilitare o disabilitare i syn cookies.

Rimane soltanto da aggiungere una cosa: come si fa a rendersi conto di essere sotto syn flood? Beh, con i syn cookies disabilitati, potreste dare un comando come:

```
# netstat -npla |grep SYN_RECV
```

Normalmente, dovreste vedere non più di 1-2 connessioni provenienti da indirizzi ip diversi. Se ne vedete più di una trentina, allora è molto probabile che vi troviate sotto syn flood. Per esperienza, posso dirvi che con un kernel 2.2 senza i syn cookies abilitati con circa 110-120 connessioni in SYN_RECV la porta attaccata diventava irraggiungibile.

Per avere una ulteriore conferma, potreste infine verificare con tcpdump... se arrivano dei nuovi syn da uno stesso ip prima ancora che voi abbiate risposto con un altro syn-ack o dopo aver temporaneamente bloccato l'ip o il demone responsabile della porta, allora siete sotto syn-flood. Anche questa prova però non può essere considerata definitiva: il syn-flood viene considerato un blind attack, ovvero un attacco che non ha bisogno di ricevere le risposte per poter essere effettuato, per cui è estremamente semplice utilizzare dei "mittenti" falsificati nei pacchetti (ed utilizzare quindi ip multipli e differenti). L'ip che vedete in netstat o tcpdump potrebbe quindi essere quello di qualcuno che nulla ha a che vedere con l'attacco. L'unica condizione perchè un ip possa essere utilizzato è infatti che questo risulti irraggiungibile dalla vostra macchina (altrimenti manderebbe un reset dopo il vostro syn-ack). Infine, nel caso in cui il kernel si trovi a dover far uso dei syn cookies, dovreste vedere dei messaggi nei file di log.

- *directory conf* ok, qua dentro trovate i parametri per le singole interfacce di rete. A parte "all" e "default", che consentono rispettivamente di cambiare i parametri di tutte le interfacce o i default per le future interfacce, ogni sottodirectory contiene i parametri di una singola interfaccia di rete.Diamo quindi un'occhiata alla directory all (o eth0, o lo, o...):
  - *accept_redirects* fa in modo che vengano accettati redirect dalla particolare interfaccia. Un redirect consiste "in un suggerimento" per un percorso "migliore" per raggiungere un determinato punto (RFC792, pagina 12). Il problema è che è facile dare questi "suggerimenti" deviando il traffico a proprio piacimento (o comunque in modo da poter fare cose poco piacevoli). Un tempo i redirect erano molto pericolosi, adesso vengono accettati solo se determinate condizioni vengono verificate. Per un firewall, è comunque bene disabilitare questi redirects, o se necessario affidarsi a demoni come zebra o bird e a protocolli più evoluti per aggiornare dinamicamente le tabelle di routing.
  - *secure_redirects e send_redirects* consentono rispettivamente di accettare i redirect che rispettino condizioni molto più stringenti di quelle normali e di generare i redirect per gli altri host. In pratica, viene verificato che il redirect venga inviato da un gateway di default conosciuto (net/ipv4/fib_semantics.c:ip_fib_check_default chiamato da route.c:ip_rt_redirect).
  - *accept_source_route* fa in modo che vengano accettati "source route", ovvero pacchetti che chiedono che la risposta passi da determinati routers. Sebbene questo non sia un problema in se stesso, è meglio disabilitare questa opzione per rendere la vita più difficile a coloro che tentino la strada dello spoofing. Uno dei problemi principali dello spoofing è infatti quello di mettersi in grado di poter ricevere le risposte dei pacchetti falsificati, ed il source_route facilita notevolmente il raggiungimento di questo obiettivo. La maggior parte dei provider, infatti, bloccano pacchetti che utilizzano questa opzione del protocollo ip.
  - *forwarding e mc_forwarding* abilitano o disabilitano rispettivamente il forwarding ed il forwarding dei pacchetti di multicast per la singola interfaccia (lo stesso forwarding di cui parlavamo prima). Attenzione però che il forwarding dei pacchetti di multicast ha bisogno di un demone di supporto.
  - *rp_filter* abilita o disabilita il "reversed path filter". Con questo filtro abilitato, viene verificata la coerenza dell'indirizzo ip del mittente di un pacchetto rispetto l'interfaccia su cui tale pacchetto è stato ricevuto e le informazioni contenute nelle tabelle di routing. In caso le informazioni non risultino coerenti, il pacchetto viene scartato. Quest'opzione può creare problemi con tabelle di routing particolarmente complesse (utilizzando ip rule, per esempio, e tabelle di routing multiple) o quando si fa uso di link asimmetrici, ovvero dove le richieste escono da un'interfaccia

mentre le risposte ritornano su un'altra (con collegamenti satellitari, per esempio, dove le richieste escono dal telefono e le risposte arrivano sulla parabolica). Utilizzato principalmente in sys/net/ipv4/fib_frontend.c:fib_validate_source.

– *log_martians* fa in modo che pacchetti strani, improbabili o impossibili vengano registrati nei file di log. Il problema è che a volte, magari con particolari configurazioni, un po' troppi pacchetti vengono loggati.

8 Filtrare il traffico con iptables

Di base, tutte le distribuzioni mettono a disposizione dei tool per configurare la rete. Se avete molta fiducia nella vostra distribuzione o gli script sono particolarmente ben fatti, potete tranquillamente usare quelli... per quanto mi riguarda, pur avendo molta fiducia negli script forniti con debian, preferisco escludere completamente il supporto della distribuzione in maniera da non avere limiti e da poter usare con tranquillità le patch sopra installate ed in modo da evitare interazioni non volute con script o programmi di cui spesso non si ha il pieno controllo. Nelle prossime sezioni configureremo un firewall "from scratch", aggiungendo cioè i nostri script in /etc/init.d in modo che vengano caricati automaticamente all'accensione della vostra macchina.

8.1 Preambolo

Ok, il nostro primo script si chiamerà conffw. Ovviamente non dovrete scrivere i numeri di riga, usati soltanto per rendere la trattazione più semplice.

Iniziamo allora con:

```
0: #!/bin/bash
1: echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
2: for a in /proc/sys/net/ipv4/conf/*/rp_filter; do
3:     echo 1 > $a
4: done
5: echo 1 > /proc/sys/net/ipv4/tcp_syncookies
6: echo 0 > /proc/sys/net/ipv4/tcp_ecn
```

In queste prime righe, ci limitiamo a dire al kernel di ignorare i ping di broadcast e di abilitare l'rp_filter su tutte le interfacce. Infine, abilitiamo i syn cookies e disabilitiamo l'ecn (vedi sezione precedente per una spiegazione più approfondita). In maniera perfettamente equivalente avremmo potuto scrivere in /etc/sysctl.conf:

```
net.ipv4.icmp_echo_ignore_broadcasts=1
net.ipv4.conf.all.rp_filter=1
net.ipv4.conf.default.rp_filter=1
net.ipv4.tcp_syncookies=1
net.ipv4.tcp_ecn=0
```

8.2 Configurare le interfacce

Da adesso in poi proverò ad andare avanti configurando un ipotetico firewall collegato ad un'ipotetica rete e spiegando man mano i vari passi. Inizierò quindi ora descrivendo l'ipotetica rete che verrà utilizzata nei prossimi esempi. Nel nostro caso, quindi, il firewall sarà equipaggiato con:

- una interfaccia eth0 collegata alla rete interna, con indirizzo 192.168.200.254 e netmask 255.255.255.0.

- una interfaccia eth1 collegata alla nostra DMZ, con indirizzo 123.45.67.9 con netmask 255.255.255.248, dove per DMZ si intende la rete dove verranno attaccati tutti quei server disponibili sia dall'interno che dall'esterno della nostra rete.
- ed infine una interfaccia eth2 collegata con il mondo esterno, con indirizzo 123.45.68.1, netmask 255.255.255.0 e con gateway su 123.45.68.254.

Ok, una volta definito il comportamento di base del firewall e la configurazione hardware, possiamo andare avanti configurando le varie schede di rete tramite il comando "ip":

```
8: ip addr add 192.168.200.254/24 dev eth0
9: ip addr add 123.45.67.9/29 dev eth1
10: ip addr add 123.45.68.1/24 dev eth2
```

Vi rimando ad altra documentazione disponibile sul mio sito per una discussione sul comando ip. Vi basti sapere che in questo modo abbiamo configurato le schede di rete senza attivarle come indicato sopra.

A questo punto, possiamo dedicarci ad iptables e ad impostare le regole di firewalling vere e proprie.

E' importante non abilitare le interfacce di rete in modo da evitare che i pacchetti inizino a fluire prima che le regole di firewalling siano state aggiunte.

8.3 Il comando iptables

Sicuramente se avete sentito parlare di linux e di firewalling, avete anche sentito parlare di ipchains e di ipfwadm. Questi comandi consentivano l'impostazione delle regole di firewalling rispettivamente nei kernel 2.2 e 2.0. Nei kernel dal 2.4 in poi è stato introdotto un nuovo comando: iptables, anche se il supporto sia per ipchains che ipfwadm è rimasto.

Dovendo parlare di iptables, è indispensabile una piccola introduzione. Il parente più stretto di iptables è sicuramente ipchains. ipchains aveva introdotto il concetto di "catena". iptables ha ripreso il concetto di catena per estenderlo tramite l'introduzione di tabelle.

Prima di parlare di tabelle però, ci conviene chiarire il concetto di catena. L'idea di base è abbastanza semplice: un computer collegato in rete con una o più interfacce ha a che vedere con tre tipi di pacchetti:

- Pacchetti destinati al firewall stesso
- Pacchetti che devono transitare dal firewall ma destinati a qualcun'altro
- Pacchetti originati dal firewall stesso

Di default ipchains ed iptables mettono quindi a disposizioni rispettivamente 3 catene:

- La catena di input
- La catena di forward
- La catena di output

In un computer normale (o in un server) verrebbero principalmente utilizzate le catene di input e di output per filtrare il traffico ricevuto e generato dalla macchina presa in considerazione.

In un firewall invece, i pacchetti transiterebbero da una scheda di rete all'altra, passando quindi attraverso la catena di forward.

Ogni catena è poi costituita da un insieme di regole. Ad ogni regola, è associata un'azione (target). Ogni volta che un pacchetto transita per una catena, tutte le regole vengono controllate in ordine, dalla prima inserita fino all'ultima. Quando una regola risulta applicabile al pacchetto (viene trovato un match) viene applicata al pacchetto l'azione specificata e viene interrotto il processo di ricerca.

Se non viene trovata una regola applicabile al pacchetto, la sua sorte (l'azione d'applicare) viene scelta in base alla politica (policy) della catena stessa. In pratica, la policy indica cosa fare di tutti i pacchetti che non rientrano in nessuna regola.

Sia ipchains che iptables poi, consentono di creare o rimuovere catene e di specificare come azione quella di percorrere un'altra catena.

Detto così può sembrare tutto molto complesso, ma vedrete che poi nella pratica diventerà molto più semplice.

8.4 Definizione delle policy

Iniziamo quindi a parlare di iptables, ed iniziamo proprio definendo la "politica" di default delle catene di base, cioè l'azione che deve essere presa nel caso in cui nessuna regola si possa applicare al particolare pacchetto. Per fare questo, basta dare:

```
12: iptables -P INPUT DROP
13: iptables -P OUTPUT DROP
14: iptables -P FORWARD DROP
```

Qui vediamo proprio che se un match non viene trovato, i pacchetti devono essere "buttati via" (DROP), proprio come se non fossero mai stati ricevuti (o richiesto l'invio, a secondo della catena).

Oltre a DROP, esistono molte altre azioni che è possibile specificare, dove le principali sono:

- ACCEPT – lascia passare il pacchetto
- QUEUE – passa il pacchetto ad un programma in userspace

Di base, le due azioni che userete di più sono ACCEPT e DROP. QUEUE difficilmente lo userete. Spesso però vi capiterà di usare delle "target extensions", ossia dei particolari target che sono forniti tramite dei moduli esterni (vi ricordate tutte le patch applicate col patch-o-matic? Molte hanno dato origine a dei nuovi target).

E' buona abitudine mettere come politica di default quella di buttare via i pacchetti, dopodichè consentire esplicitamente tutto ciò che vogliamo fare passare. Questo principalmente per due motivi:

- è facile rendersi conto di cosa si è dimenticato chiuso (molti dei vostri utenti vi telefoneranno fino alla saturazione delle linee telefoniche pur di vedere il loro client preferito funzionare), mentre è molto più difficile vedere cosa si è dimenticato aperto.
- non è facile riconoscere tutto ciò che c'è da bloccare. Per esempio, Usando una politica di ACCEPT, è estremamente difficile prevedere ciò che potrà essere utilizzato per aggirare le restrizioni, contando che spesso non è obbligatorio seguire gli standard e una persona un po' sveglia potrebbe modificare il kernel di due macchine linux per utilizzare un protocollo che nulla ha a che vedere col tcp e che mai noi ci saremmo immaginati di dover bloccare (trattandosi, magari, di un protocollo inventato di sana pianta)...

Un comando che vi può essere utile nello scrivere le regole si chiama "nstreams": questo stamperà a video l'elenco di tutte le connessioni in corso su una rete...

8.5 La nostra configurazione

Prima di andare avanti e definire le regole di firewalling, dobbiamo avere un'idea chiara di cosa vogliamo fare con la nostra rete. Prima di procedere, è quindi fondamentale un minimo di pianificazione. Nel nostro caso, abbiamo una dmz, una lan ed un collegamento ad internet.

Abbiamo un firewall con tre interfacce di rete, ognuna collegata ad una rete diversa e abbiamo quindi una connessione ad internet relativamente lenta (rispetto alle altre due reti). L'idea di base è quella di mettere nella dmz i server e di fare in modo da risparmiare banda, inserendo un proxy server.

Sulla nostra dmz, ci troveremo quindi:

- un web server, ad uso sia interno che esterno
- un mail server (pop3 e smtp), ad uso sia interno che esterno
- un proxy server, soltanto ad uso interno
- un dns, ad uso sia interno che esterno

Per quanto riguarda le altre restrizioni da imporre alle comunicazioni tra le varie reti (ad esempio, non vogliamo che gli utenti della lan controllino altre caselle di posta elettronica oltre quelle fornite dal nostro mail server), ne parleremo in ogni specifica sezione.

8.6 Prime catene

Ok, due sezioni fa abbiamo detto al kernel di buttare via tutto quanto può avere a che fare col nostro computer. Per andare avanti nella nostra configurazione, dobbiamo quindi indicare cosa invece lasciare passare. Questo si fa aggiungendo delle regole. Una soluzione potrebbe essere quella di aggiungere tutte le regole relative al traffico che vogliamo lasciare passare nelle 3 catene di base. Funzionerebbe, ma vi posso garantire che sarebbe molto facile commettere degli errori e vi posso garantire che altrettanto facilmente diventerebbero così tante da non poter essere più gestibili.

Uno degli approcci più semplici nella gestione delle regole di firewalling è quindi quello di dividere il traffico in "flussi" di dati dopodichè considerare indipendentemente ogni singolo flusso, evitando così di avere tutte le regole insieme e formalizzandosi in modo da evitare gli errori più comuni.

Analizzando il nostro esempio, potremmo quindi dividere il nostro traffico in 6 grandi flussi:

- tutto ciò che dalla lan va alla dmz
- tutto ciò che dalla lan va su internet
- tutto ciò che dalla dmz va su internet
- tutto ciò che dalla dmz va sulla lan
- tutto ciò che da internet va sulla nostra dmz
- tutto ciò che da internet vuole andare sulla nostra lan

Possiamo poi, per esempio, creare per ogni flusso una catena e ad ogni catena associare un suo insieme di regole indipendenti da quello di tutte le altre catene.

Ma andiamo per gradi: iniziamo a creare delle nuove catene. Per fare questo dobbiamo eseguire iptables seguito da un "-N" e da un nome per la catena:

```
16: iptables -N landmz       #dalla scheda di rete eth0 alla scheda di rete eth1
17: iptables -N laninet      #dalla scheda di rete eth0 alla scheda di rete eth2
18: iptables -N dmzinet      #dalla scheda di rete eth1 alla scheda di rete eth2
19: iptables -N dmzlan       #dalla scheda di rete eth1 alla scheda di rete eth0
20: iptables -N inetdmz      #dalla scheda di rete eth2 alla scheda di rete eth1
21: iptables -N inetlan      #dalla scheda di rete eth2 alla scheda di rete eth0
```

Per il kernel, però, il nome che abbiamo dato ed il commento non hanno alcun significato. Dobbiamo quindi trovare un modo per indicare quali catene devono essere percorse in quali situazioni. Per fare questo, possiamo specificare 6 semplici regole. Trattandosi di regole per traffico che non è né originato né destinato al firewall, dovremo inserirle nella catena di FORWARD. Per esempio, potremmo dare dei comandi come:

```
23: iptables -A FORWARD -i eth0 -o eth1 -j landmz
24: iptables -A FORWARD -i eth0 -o eth2 -j laninet
25: iptables -A FORWARD -i eth1 -o eth2 -j dmzinet
26: iptables -A FORWARD -i eth1 -o eth0 -j dmzlan
27: iptables -A FORWARD -i eth2 -o eth1 -j inetdmz
28: iptables -A FORWARD -i eth2 -o eth0 -j inetlan
```

Il significato di queste righe è molto semplice: aggiungi (-A, append) alla catena FORWARD, una regola per cui se un pacchetto proviene dalla scheda di rete eth0 (-i) ed è destinato ad uscire dalla scheda eth1 (-o), la sua sorte deve essere decisa (-j) dalla catena landmz e così via. In pratica, abbiamo diviso tutti i pacchetti in transito nella catena di forward in 6 categorie, dove ogni categoria ha associata una catena e quindi una serie di regole. Riassumendo:

- *-N* crea una nuova catena
- *-A* aggiunge una regola ad una catena
- *-i* consente, in una regola, di discriminare i pacchetti in base all'interfaccia fisica da cui sono entrati
- *-o* in base all'interfaccia fisica da cui usciranno (scelta in base alle tabelle di routing da noi impostate)
- *-j* di mandare i pacchetti ad un'altra catena

Abbiamo visto quindi come una regola può identificare dei pacchetti in base alle schede di rete coinvolte. E' però possibile utilizzare molti altri criteri, per esempio:

- il protocollo (-p)
- l'indirizzo ip sorgente (-s)
- l'indirizzo ip destinazione (-d)
- se si tratta di un frammento (-f)
- o se non lo è (! -f)

Dove ogni protocollo può aggiungere dei criteri di classificazione. Per esempio, se specifichiamo "-p tcp" per indicare il protocollo tcp, possiamo poi dividere i pacchetti in base anche

- alla porta sorgente (–sport)
- alla porta destinazione (–dport)
- ai flag del tcp (–tcp-flags SYN, ACK, FIN...)

- alle opzioni (–tcp-option)

Anche in questo caso, oltre alle estensioni fornite dai protocolli, è possibile utilizzare delle estensioni fornite da moduli.

A questo punto però, vi sarà venuto spontaneo chiedervi perché effettuare una prima classificazione in base all'hardware e non, ad esempio, in base all'indirizzo ip sorgente o all'indirizzo ip destinazione. Ancora una volta, la risposta è molto semplice: il contenuto di un pacchetto si può falsificare. Il fatto che arrivi su un'interfaccia piuttosto che un'altra, no. Utilizzando almeno per le prime regole le interfacce di input e quelle di output, avremo quindi la certezza che i pacchetti saranno valutati dalle regole contenute nella catena corretta e che nessuno potrà imbrogliarci utilizzando ip fasulli (ancora, cosa succederebbe se basassimo le nostre regole solo sugli indirizzi ip e qualcuno ci mandasse dei pacchetti provenienti da 127.0.0.1 con l'rp_filter disabilitato?).

8.6.1 Dalla LAN alla DMZ

Ricordandoci della configurazione della nostra ipotetica rete, abbiamo deciso che dalla nostra lan vogliamo che tutte le richieste per pagine web vengano deviate sul nostro server proxy (transparent proxy) in maniera trasparente, che i nostri utenti non controllino altre caselle di posta elettronica se non quelle da noi fornite, che possano collegarsi al nostro server web direttamente e che siano in grado di scaricare file con ftp direttamente da internet. Dall'esterno, vogliamo soltanto rendere accessibile il nostro dns, il server di posta elettronica, il nostro server web ed il server ftp. Ma lavoriamo anche qua per flussi, in modo da semplificare un po' le cose.

Iniziamo allora a creare le regole per la nostra lan, lasciando indietro (per ora) ciò che riguarda il "deviare". Dalla lan alla dmz, vogliamo quindi consentire:

- le connessioni al nostro server web (www)
- le connessioni per spedire la posta (smtp)
- le connessioni per ricevere la posta (pop3)
- le connessioni verso il server proxy (webcache)
- nonché le connessioni necessarie per utilizzare il nostro dns (domain)

Aggiungiamo quindi le regole corrette, esattamente come indicato qua sopra:

```
30: iptables -A landmz -s ! 192.168.200.0/24 -j DROP
31: iptables -A landmz -p tcp -d nostro.server.web --dport www -j ACCEPT
32: iptables -A landmz -p tcp -d nostro.server.smtp --dport smtp -j ACCEPT
33: iptables -A landmz -p tcp -d nostro.server.pop3 --dport pop3 -j ACCEPT
34: iptables -A landmz -p tcp -d nostro.server.proxy --dport webcache -j ACCEPT
35: iptables -A landmz -p tcp -d nostro.dns --dport domain -j ACCEPT
36: iptables -A landmz -p udp -d nostro.dns --dport domain -j ACCEPT
```

Prima di tutto, qua dentro sarebbe molto meglio indicare direttamente degli indirizzi ip al posto dei nomi dei computer. Come abbiamo già detto, infatti, il protocollo dei DNS deve essere considerato insicuro. Se proprio volessimo usare i nomi dei computer, potremmo utilizzare il vecchio file degli hosts, in /etc/hosts, dove ogni linea è formata dall'indirizzo ip di un computer seguito dal suo nome (e separati da spazi).

Come potete vedere, questa volta le regole le "appendiamo" (-A) non più alla catena di FORWARD bensì alla catena landmz, trattandosi di regole che andranno a discriminare il traffico tra la nostra lan e la nostra dmz.

Ora, la prima regola indica che non vogliamo che passi niente che non abbia un indirizzo ip proveniente dalla nostra rete interna. Questa regola eviterà quindi che qualcuno dei nostri utenti possa fare spoofing verso la nostra dmz o che comunque tenti di imbrogliarci con dei giochi strani sull'indirizzo ip. E' una regola un po' superflua, avendo già abilitato la protezione del kernel, ma a volte è meglio aggiungere qualche regola in più piuttosto che avere qualche regola in meno (e poi, siete sicuri che nelle prossime versioni del kernel il filtro sarà ancora disponibile o che si comporterà sempre nello stesso modo?).

Le altre sono ancora regole molto semplici. Le uniche novità introdotte rispetto prima sono la negazione (il !), che indica che perché una regola venga applicata una certa condizione **non** deve essere soddisfatta, ed il fatto che finalmente potete vedere diversi criteri messi in pratica. -s, specifica un ip sorgente e può essere seguito da un indirizzo ip o da un indirizzo di rete (indicato come x.x.x.x/y) o dal nome di un host. -p può essere seguito dal nome o dal numero di un protocollo (file `/etc/protocols` conserva le associazioni). -d, ancora, indica la destinazione di un pacchettino e può essere seguito sempre da un indirizzo ip, da un nome di host o da un indirizzo di rete (con lo stesso formato indicato prima). –dport, invece, indica una porta di destinazione e può essere seguito da un numero o dal nome di una porta (il file `/etc/services` conserva le associazioni). In tutti i casi, è possibile usare la negazione dopo l'indicazione del criterio.

Infine, ACCEPT dice ad iptables di accettare tali pacchetti nel caso in cui la regola risulti applicabile. Per configurare un firewall, è necessaria una buona conoscenza dei vari protocolli di rete: l'ultima riga è stata aggiunta in quanto il protocollo per la comunicazione con i dns utilizza il più delle volte connessioni udp, mentre utilizza connessioni tcp solo in condizioni particolari. Senza questa regola, il dns avrebbe funzionato sempre tranne qualche volta, e sarebbe stato estremamente difficile trovare il problema (se mai ce ne fossimo resi conto).

Per chi di voi si fosse invece chiesto come si fa a specificare una rete o che senso ha il /24, basti sapere che si tratta di un sistema estremamente comodo per indicare le netmask. Per esempio, una netmask 255.255.255.0 indica che se i primi 24 bit di due indirizzi ip sono uguali, allora i due si trovano sulla stessa rete (255.255.255.0 scritto in notazione binaria sarebbe 11111111.11111111.11111111.00000000, con 24 uno). 255.255.255.0 è quindi equivalente a /24 (ogni ottetto - 255 - sono 8 bit).

Visto però che si tratta delle prime regole "serie" della nostra trattazione, vediamo comunque di descriverle a parole, un po' come abbiamo fatto prima:

- 30: Butta via ogni pacchetto in transito nella catena landmz che **non** venga da un indirizzo ip facente parte la nostra lan.
- 31: Accetta ogni pacchetto tcp in transito dalla catena landmz destinato alla porta 80 (www) del nostro.server.web
- 32: Accetta ogni pacchetto tcp in transito dalla catena landmz destinato alla porta 25 (smtp) del nostro.server.smtp
- 33: Accetta ogni pacchetto tcp in transito dalla catena landmz destinato alla porta 110 (pop3) del nostro.server.pop3
- 34: Accetta ogni pacchetto tcp in transito dalla catena landmz destinato alla porta 8080 (webcache) del nostro.server.proxy
- 35: Accetta ogni pacchetto udp in transito dalla catena landmz destinato alla porta 53 (dns) del nostro.dns
- 36: Accetta ogni pacchetto tcp in transito dalla catena landmz destinato alla porta 53 (dns) del nostro.dns
- (Butta via tutto il resto, come definito dalla policy)

Ad alcuni di voi sarà venuto spontaneo chiedersi come mai è stata messa come prima regola un "butta via" con una negazione, anzichè una più semplice accetta (qualcosa del tipo: accetta ogni pacchetto in transito nella catena landmz che viene da un indirizzo ip facente parte della nostra lan). Il problema è che il controllo delle regole si ferma alla prima regola soddisfatta. Se fosse stata quindi un'accept, la regola sarebbe stata soddisfatta per ogni pacchetto proveniente dalla lan, e la scansione delle regole non sarebbe andata avanti a quelle successive, vanificando buona parte del nostro lavoro. Un'altra possibilità sarebbe stata quella di omettere la prima regola e di specificare ogni volta esplicitamente l'indirizzo ip, aggiungendo alle righe dalla 31 alla 36 qualcosa come "-s 192.168.200.0/24". Siccome però sono una persona molto pigra, ho scelto il metodo più veloce da scrivere.

L'ultima regola indicata tra parentesi, poi, non è stata da noi scritta: è implicitamente aggiunta dalla policy (DROP) che abbiamo deciso di utilizzare.

L'esperienza però mi insegna che a questo punto potrebbe essere conveniente inserire come regola 38 qualcosa di simile a:

```
38: iptables -A landmz -p tcp -j REJECT --reject-with tcp-reset
```

Questo ha a che vedere con i meandri del tcp. Ma vediamo di spiegarla in poche parole: quando avviene una connessione TCP/IP, ha luogo il solito "three way handshake". In pratica, per avere la certezza che i pacchetti arrivino a destinazione, il computer A manda un messaggio del tipo "questo è il mio numero, e se ci sei, mandami il tuo", dopodichè il computer B risponde con un messaggio del tipo "ho ricevuto il tuo numero, questo è il mio", infine il computer A risponde dicendo "ho ricevuto il tuo numero" e la comunicazione ha inizio. Normalmente però, rispettando il protocollo TCP/IP, se il computer A dovesse tentare di collegarsi ad una porta chiusa (ad un servizio non disponibile) su B, allora B dovrebbe rispondere con "la porta è chiusa".

Ma vediamo cosa succede nel nostro caso:

- Tra A e B c'è il nostro firewall F.
- A manda il messaggio a B ad una porta non consentita.
- senza regola 38, F riceve il messaggio e lo butta via.
- A rimane in attesa di una risposta per parecchi secondi e ci riprova, sempre con lo stesso risultato.
- A questo punto, A è rimasto in attesa di un tempo valutabile in minuti, mentre B è rimasto inconsapevole di tutto.

Bene, la regola 38 dice che per ogni connessione tcp arrivata in fondo alla catena (essendo stata aggiunta per ultima), invece di essere buttato via il pacchetto, deve essere utilizzata l'estensione (modulo esterno) REJECT per mandare un "tcp-reset", il messaggio che indica il fatto che la porta è chiusa.

Vediamo però che il target REJECT è stato specificato semplicemente con un -j, come per tutti gli altri target. Quando si parla di target, infatti, non c'è alcuna differenza tra "estensioni" esterne e target forniti direttamente da iptables, se non la pagina di manuale.

Alcuni di voi si potranno chiedere a questo punto "ma che diavolo, cosa mi interessa se l'altro tentando di accedere ad una porta non consentita rimane in attesa?? Così impara per la prossima volta...". Beh, ci sono almeno tre buoni motivi per non lasciarlo in attesa:

- A volte, quando vi collegate per esempio ad un server di posta elettronica, alcuni sistemi fanno quello che viene definito un "ident lookup", cercano cioè di capire chi siete collegandosi alla porta 113 della vostra macchina, dove teoricamente ci dovrebbe essere un demone in ascolto che fornisce questo tipo di informazioni.Il problema è che questi sistemi non vi fanno accedere al servizio fino a quando non

scoprono la vostra identità o non appurano che il vostro sistema non offre questo servizio (rispondendo che la porta è chiusa). In entrambi i casi, il sistema remoto rimane in attesa di una risposta, e se questa non arriva, può ipotizzare due cose:

1. che la richiesta sia andata persa
2. che il vostro computer sia improvvisamente morto

Proveranno quindi a rimandare il pacchetto per diverse volte prima di arrendersi e farvi entrare (ma come, il vostro computer non era morto? Non ha risposto quando abbiamo tentato di contattarlo...), introducendo spesso dei delay valutabili nell'ordine delle decine di secondi o addirittura dei minuti.

L'esempio più classico è questo, ma altri protocolli usano un approccio simile, e vi posso garantire che una regola come la 38 può evitarvi diversi delay nell'utilizzo della rete altrimenti difficilmente spiegabili.

- Molti "scanner" sono tratti in inganno da questa regola che fa credere loro che la porta sia effettivamente chiusa rendendo quindi più difficile l'identificazione del firewall (vi siete mai chiesti come nmap faccia ad indicare che una porta è "filtered" piuttosto che "closed"?). Attenzione però che scanner più intelligenti potrebbero rilevare le differenze di TTL (il firewall si trova un passo prima del vero client), rendendo vano tale tentativo (se volete divertirvi, date un'occhiata al target TTL, che vi consente di evitare persino quest'effetto).

Questa regola la vedrete apparire molto spesso nelle catene seguenti, sempre per lo stesso motivo... Dovremmo quindi aver finito con il traffico dalla LAN alla nostra DMZ.

8.6.2 Dalla DMZ alla LAN

Ok, qua (a dirsi) le cose sono molto più semplici: dobbiamo consentire soltanto quei pacchettini in risposta alle richieste partite dalla LAN (i server non si devono connettere di loro spontanea volontà ai nostri client – teoricamente, nessuno dovrebbe usarli).

Fare questo in ipchains era una cosa abbastanza complessa: bisognava specificare delle regole per consentire ogni tipo di pacchetto che ci sarebbe potuto tornare in risposta (giocando con l'opzione –syn, che seleziona i pacchetti che stabiliscono nuove connessioni) e si doveva quindi avere una buona conoscenza (se non ottima) dei vari protocolli. iptables facilita molto le cose tramite l'introduzione di "moduli per il tracciamento delle connessioni" (conntrack – uno dei molti vantaggi dei firewall statefull rispetto quelli stateless).

Vediamo quindi una delle soluzioni che potremmo adottare:

```
40: iptables -A dmzlan -s ! 123.45.67.9/29 -j DROP
41: iptables -A dmzlan -m state --state ESTABLISHED,RELATED -j ACCEPT
42: iptables -A dmzlan -p tcp -j REJECT --reject-with tcp-reset
```

Bene, il primo comando è la solita regola per evitare lo spoofing. Il secondo invece, è diverso da ogni altro comando finora incontrato: -m chiede ad iptables di caricare un modulo esterno, in questo caso il modulo state (si ricorda lo stato delle connessioni), per il tracciamento delle connessioni. L'opzione –state relativa al modulo state è quello che ci permette di identificare i pacchetti: in pratica, la regola ha come significato quello di consentire tutti i pacchetti facenti parte connessioni già stabilite (ESTABLISHED, e se sono già stabilite vuol dire che sono state consentite) e tutte quelle connessioni relative a connessioni già esistenti (RELATED).

In questo caso, la parolina magica è proprio RELATED. Ci consente cioè di scaricare sul modulo state la responsabilità di identificare i pacchettini che non soltanto sono risposte a pacchetti già inviati, ma anche quelli che fanno parte dello stesso protocollo.

Vediamo però qualche esempio:

- Quando volete scaricare un file tramite ftp il vostro browser (o qualsivoglia programma stiate utilizzando) apre prima una connessione di controllo sul server remoto, sulla classica porta 21. Dopodiché si mette d'accordo con il computer remoto su una porta da utilizzare per creare una nuova connessione per l'invio dei dati. Ma proprio qui sorge il problema: per passare dal nostro firewall (che ha una politica di DROP), questa connessione dovrebbe essere esplicitamente consentita a priori con una regola che termina con un -j ACCEPT. A priori, però, non possiamo creare questa regola in quanto non sappiamo le porte che verranno utilizzate (possono essere liberamente negoziate sia dal client che dal server). La soluzione normalmente adottata prima dell'introduzione dei moduli di tracciamento era molto semplice: consentire tutte le connessioni salvo quelle a porte problematiche. Il modulo state, invece, ci viene in aiuto aggiungendo e rimuovendo regole dinamicamente secondo quanto necessario.
- Il protocollo ip lavora in collaborazione con il protocollo ICMP che viene utilizzato per la segnalazione e la gestione degli errori. Ebbene, senza utilizzare il modulo state con l'opzione RELATED, avremmo dovuto impostare per ogni connessione consentita una regola del tipo "consenti tutti i messaggi di errore relativi a questa connessione", oppure consentire l'entrata di tutti i pacchetti ICMP di alcuni tipi. Bene, il modulo state si prende in carico anche di questo aggiungendo e togliendo regole dinamicamente ed in maniera trasparente.

La differenza è quindi questa: ESTABLISHED, fa passare tutti i pacchetti che fanno parte di una connessione già stabilita, mentre RELATED fa passare tutti i pacchetti di controllo (errori, piuttosto che...) o connessioni relative alle connessioni conosciute.

Se state configurando un firewall, però, e la vostra preoccupazione principale è la sicurezza, vi sarà sorta spontanea una domanda: "e se qualcuno imbrogliasse il modulo di state, facendo aprire connessioni non volute?". Bene, in passato qualche problema di questo tipo è stato sollevato, ma il codice del netfilter dovrebbe essere attualmente abbastanza maturo per evitare questo tipo di problemi.

Vi consiglio però di seguire le mailing list dedicate, per essere eventualmente prontamente avvisati di possibili problemi, e di leggere qualche documento in più sul funzionamento del protocollo ftp.

Attenzione però che perché le regole sopra elencate funzionino, potrebbe essere necessario caricare dei moduli nel kernel, con comandi del tipo "modprobe ip_conntrack", "modprobe ip_conntrack_ftp", "modprobe ip_conntrack_altri_moduli_di_protocolli_che_volete_utilizzare".

Per quanto riguarda le prestazioni, il modulo state utilizza un po' più di risorse rispetto alle regole manuali, ma comunque nulla di rilevante se rapportato ai vantaggi che offre (alcune cose, non è proprio possibile farle con firewall stateless, a meno di non aprire migliaia di porte).

Un'altra cosa da dire è che prima, nella definizione della catena dalla LAN alla DMZ, ho volutamente dimenticato la regola 37:

```
37: iptables -A landmz -m state ESTABLISHED,RELATED -j ACCEPT
```

che sarebbe stata troppo prematura da discutere. In pratica, anche dalla lan alla dmz è importante far passare tutti quei dati relativi a connessioni già stabilite o comunque i messaggi di errore. Probabilmente non molto importante per reti di piccole dimensioni, ma rilevante in reti più grosse o con regole più complicate. Anche questa regola la vedrete apparire molto spesso.

8.6.3 Dalla LAN ad Internet

Dalla nostra lan ad internet vogliamo invece essenzialmente che passi traffico ftp (che non può essere deviato sul proxy), in modo che gli utenti possano scaricare direttamente il loro materiale. Per fare questo, basta aggiungere le regole:

```
44: iptables -A laninet -s ! 192.168.200.0/24 -j DROP
45: iptables -A laninet -p tcp --dport ftp -j ACCEPT
48: iptables -A laninet -m state ESTABLISHED,RELATED -j ACCEPT
49: iptables -A laninet -p tcp -j REJECT --reject-with tcp-reset
```

Dove però non c'è nulla di nuovo di cui parlare...

8.6.4 Da Internet alla LAN

Infine, per terminare con la nostra lan:

```
50: iptables -A inetlan -s 192.168.200.0/24 -j DROP
51: iptables -A inetlan -s 123.15.67.9/29 -j DROP
52: iptables -A inetlan -m state --state ESTABLISHED,RELATED -j ACCEPT
53: iptables -A inetlan -p tcp -j REJECT --reject-with tcp-reset
```

In questo caso, le prime due regole bloccano tutti quei pacchetti che hanno come mittente l'indirizzo di una delle nostre due reti.

La terza regola, invece, consente a tutte le connessioni create dall'interno di poter ricevere una risposta, senza dover scrivere troppe regole.

All'epoca di ipchains invece, non c'era un modo di "tenere traccia" delle connessioni ed era quindi necessario aprire molte più porte per consentire a ftp di funzionare, cosa che diminuiva notevolmente l'efficacia del firewall.

8.6.5 Da Internet alla DMZ

Andiamo avanti ora con la configurazione della nostra dmz. Abbiamo detto che da internet saranno accessibili i seguenti servizi:

- server web
- server smtp
- server dns
- server ftp

Che si traduce linearmente in qualcosa del tipo:

```
54: iptables -A inetdmz -s 192.168.200.0/24 -j DROP
55: iptables -A inetdmz -s 123.15.67.9/29 -j DROP
56: iptables -A inetdmz -p tcp -d nostro.server.web --dport www -j ACCEPT
57: iptables -A inetdmz -p tcp -d nostro.server.smtp --dport smtp -j ACCEPT
58: iptables -A inetdmz -p tcp -d nostro.dns --dport domain -j ACCEPT
59: iptables -A inetdmz -p udp -d nostro.dns --dport domain -j ACCEPT
60: iptables -A inetdmz -p tcp -d nostro.ftpserver --dport ftp -j ACCEPT
61: iptables -A inetdmz -m state --state ESTABLISHED,RELATED
62: iptables -A inetdmz -p tcp -j REJECT --reject-with tcp-reset
```

In questo caso, la prima regola è un po' superflua ma estremamente importante. I nostri server, infatti, potrebbero essere configurati per essere un po' meno restrittivi nei confronti dei nostri utenti. Se qualcun altro riuscisse da internet ad usare i nostri indirizzi ip, sarebbe in grado di usufruire di questi benefici.

8.6.6 Dalla DMZ ad Internet

Dalla DMZ ad Internet, invece

```
63: iptables -A dmsinet -s ! 123.15.67.9/29 -j DROP
64: iptables -A dmzinet -p tcp -s nostro.server.smtp --dport smtp -j ACCEPT
65: iptables -A dmzinet -p udp -s nostro.server.dns --dport domain -j ACCEPT
66: iptables -A dmzinet -p tcp -s nostro.server.dns --dport domain -j ACCEPT
67: iptables -A dmzinet -p tcp -s nostro.server.proxy --dport www -j ACCEPT
68: iptables -A dmzinet -m state --state ESTABLISHED,RELATED -j ACCEPT
69: iptables -A dmzinet -p tcp -j REJECT --reject-with tcp-reset
```

Qui qualche commento è doveroso farlo...

La prima regola consente al server smtp di inviare la posta elettronica dei nostri utenti.

Dello stesso tipo sono la seconda e terza regola, che consentono invece al dns server di effettuare delle query ricorsive ed eventualmente degli zone transfer.

La quarta regola, infine, è quella che consente al proxy server di uscire all'esterno a procurarsi le pagine non in cache. Attenzione, però, che a secondo delle configurazioni del vostro proxy server, potrebbe essere necessario aprire più porte (per fare comunicare più proxy tra di loro, per consentire l'utilizzo al proxy del protocollo ftp...).

Le ultime due sono le solite regole: consenti le risposte alle richieste effettuate, chiudi le connessioni tcp vietate.

8.6.7 Traffico da e per il firewall

Come ultima cosa, ci rimane da filtrare il traffico originato o destinato al firewall direttamente. Giusto a titolo esemplificativo, ecco alcune regole:

```
71: iptables -A INPUT -m limit --limit 10/min -p tcp --syn --dport ssh -j ACCEPT
72: iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
73: iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
74: iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset

75: iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Buona norma sarebbe quello di consentire il meno possibile verso il firewall, e di bloccare tutto ciò che può da lui essere originato. In questo caso, viene accettato il traffico ssh ed i ping (non sempre una buona idea) e tutto il traffico relativo, mentre in uscita vengono consentite solo le risposte ai pacchettini ricevuti. Se utilizzate demoni come zebra o simili, potrebbe essere necessario aprire più porte ed allentare un po' la cintura soprattutto per quanto riguarda icmp. Ricordatevi inoltre che se non viene specificata esplicitamente una interfaccia, la catena di INPUT e di OUTPUT determineranno il comportamento di **input e output** da tutte le interfacce (compreso il loopback!!!). State quindi molto attenti a quello che fate.

Potete vedere anche l'utilizzo di un nuovo modulo: il limit, che consente di impostare dei limiti di frequenza, e per la prima volta dell'opzione (tcp) –syn, che seleziona soltanto le nuove connessioni (i famosi pacchettini "mandami il tuo numero"). Questo per evitare un problema con ssh segnalato diverso tempo fa (e probabilmente già corretto) per cui in determinate condizioni e con reti molto veloci era possibile fare hijacking di una connessione tentando di indovinare alcuni parametri.

Ultima cosa importante da dire prima di passare ad altro: a differenza da ipchains, in iptables ciò che non è esplicitamente consentito non passa (o vice versa, a secondo della policy). Per esempio, nel nostro caso un

client interno alla rete non potrebbe "pingare" alcun server all'esterno né sarebbe in grado di utilizzare altri messaggi ICMP che non siano imparentati (related) con connessioni esistenti.

Con questo paragrafo viene chiuso l'argomento filtraggio. Nel prossimo paragrafo si parlerà infatti del NAT e di come fare quelle "deviazioni" di cui ci siamo temporaneamente dimenticati.

Lo scopo era quello di introdurre l'utente all'uso di iptables e di metterlo nelle condizioni di poter leggere facilmente le pagine del manuale, dove può trovare un elenco di tutti i possibili match che si possono effettuare (divisi per protocollo e modulo) e di tutti i target che si possono utilizzare. Per le target extension installate con il patch-o-matic, riferitevi alla documentazione fornita.

9 NAT con iptables

Col paragrafo precedente abbiamo completato la configurazione di una piccola rete per quanto riguarda il filtraggio.

Argomento di questa sezione è il NAT, ovvero "Network Address Translation".

Il NAT consente ad un firewall o router linux di non limitarsi a bloccare o consentire i pacchetti in transito, bensì anche di modficarli secondo delle regole ben definite.

Il codice che in linux gestisce il NAT è stato completamente ridisegnato per iptables ed il risultato è stato un sistema molto più pulito e flessibile rispetto quello utilizzato da ipchains. ipchains, per effettuare modifiche ai pacchetti, utilizzava semplicemente dei target diversi come MASQ (per mascherare i pacchetti). Questo però creava molta confusione sul meccanismo di scansione delle regole ed andava ad interferire con le regole di filtraggio. In iptables, invece, la parte di filtraggio dei pacchetti è completamente indipendente da quella di modifica e di gestione delle regole di nat. Per raggiungere questa indipendenza è stato infatti introdotto il concetto di "tabella" di cui fin'ora non avevamo parlato. Ebbene, le 3 catene principali di cui abbiamo parlato fin'ora (INPUT, OUTPUT, FORWARD) si trovano nella tabella "filter". La tabella di filter è quella che viene scelta in automatico se non viene specificata un'altra tabella (per questioni di compatibilità).

Esistono quindi altre tabelle in iptables, ognuna delle quali mette a disposizione diverse catene di base:

- la tabella *filter*, le cui catene principali sono **INPUT, OUTPUT, FORWARD**
- la tabella *nat*, le cui catene principali sono **PREROUTING, OUTPUT, POSTROUTING**
- la tabella *mangle*, le cui catene principali sono **PREROUTING, OUTPUT**

Beh, ci sono 3 tabelle in totale. La prima, che normalmente non viene mai indicata esplicitamente, che serve per filtrare i pacchetti. La seconda, quella di nat, che serve normalmente per modificare ciò che ha a che vedere col mittente o il destinatario di un pacchetto, ed infine la tabella di mangle che consente di modificare tutte le altre caratteristiche dei pacchetti.

La distinzione tra mangle e nat non è così netta, si tratta soltanto di un criterio di base, ed il manuale vi saprà sicuramente indicare la tabella migliore da utilizzare a secondo delle vostre necessità.

Ma guardiamo meglio i nomi delle catene: nella tabella filter, ci troviamo le solite INPUT, OUTPUT, FORWARD, col significato che ormai conosciamo. Nella tabella di nat abbiamo invece 3 nuove catene: la catena di PREROUTING, la catena di OUTPUT e la catena di POSTROUTING. La catena di OUTPUT mantiene lo stesso significato che aveva nella tabella di filter. La catena di PREROUTING e quella di POSTROUTING classificano invece i pacchetti diversamente: non più in base alla direzione dei pacchetti, bensì al fatto che sia già stata presa una decisione di routing su questi pacchetti. Vediamo di spiegare meglio l'idea: quando un pacchetto di qualsiasi protocollo deve uscire da una macchina connessa ad una rete (poco importa la provenienza del pacchetto) deve essere presa una decisione di routing. Deve cioè essere stabilito

da che interfaccia dovrà uscire questo pacchetto e con quali caratteristiche dovrà essere mandato (es, il TTL dovrebbe essere decrementato, l'MTU potrebbe essere diverso e necessitare una frammentazione, l'indirizzo hardware specificato potrebbe essere quello di un gateway...). Dal punto di vista logico, quindi, ogni pacchetto che deve uscire da un'interfaccia di rete passa prima dalla catena di PREROUTING, viene presa una decisione sull'interfaccia da cui fare uscire il pacchetto, dopodichè passa dalla catena di POSTROUTING.

PREROUTING e POSTROUTING sono state introdotte proprio per consentire di realizzare diversi comportamenti: ad esempio, se si vuole cambiare l'indirizzo di destinazione di un pacchetto sarà importante farlo prima che venga deciso da dove fare uscire il pacchetto (se così non fosse, il pacchetto uscirebbe dall'interfaccia sbagliata, a meno che questo non sia l'effetto desiderato). Mentre, al contrario, quando si cambia il mittente di un pacchetto potrebbe essere importante farlo dopo che una decisione di routing è stata presa (in questo caso però l'errore non sarebbe così palese).

Comunque sia, in ognuna di queste tabelle è possibile specificare dei target differenti utilizzando però le stesse opzioni e gli stessi moduli per creare delle regole. Di base, i target più conosciuti per la tabella di NAT sono proprio:

- **SNAT** consente di cambiare l'indirizzo ip sorgente di un pacchetto (POSTROUTING)
- **DNAT** consente di cambiare l'indirizzo ip destinazione di un pacchetto (PREROUTING)
- **MASQUERADE** consente di cambiare l'indirizzo ip sorgente di un pacchetto (verrà spiegata in seguito la differenza - POSTROUTING).
- **REDIRECT** consente di cambiare la porta di destinazione di un pacchetto (PREROUTING)

Torniamo ora al nostro esempio: la nostra lan utilizza degli indirizzi ip del tipo 192.168, e come tali non possono viaggiare su internet (192.168 è una classe di indirizzi ip riservati per reti private). Per consentire quindi ai computer della nostra rete di navigare, sarebbe necessario o un proxy (che però gestisca tutti i protocolli) o fare in modo che l'indirizzo ip sorgente dei pacchetti originati dalla nostra rete venga cambiato con un indirizzo ip valido per viaggiare in internet (come quello del nostro firewall, per esempio).

Realizzare questo a livello di regole di iptables è estremamente facile. Bisogna però fare una piccola introduzione e dare un minimo di spiegazioni. A molti di voi, ad esempio, sarà sorto spontaneo chiedersi come diavolo può fare un firewall a nascondere 20 computer (per esempio) dietro un unico indirizzo ip. Voglio dire, in uscita funziona bene: il client FOO dietro il firewall manda un pacchettino al gateway (il nostro firewall), il nostro firewall cambia l'indirizzo ip sorgente col suo (valido su internet) e manda avanti il pacchettino, il server remoto risponde, il pacchettino torna indietro, e... il pacchettino che gli torna indietro ha come mittente il server remoto, come destinatario il firewall... non sembra funzionare tanto... come fa il firewall a capire a quale dei nostri client deve rimandare il pacchetto, se l'indirizzo di destinazione è il suo indirizzo esterno? E anche ipotizzando di avere un indirizzo ip solo per i client, come farebbe a mandare la risposta al client corretto? Se guardasse l'indirizzo ip del server remoto, solo un client per volta potrebbe navigare su un determinato sito... non proprio funzionale. Una soluzione potrebbe essere quella di avere tanti ip esterni quanti client. Ma allora perché usare il nat? La sicurezza potrebbe essere un buon motivo, ma normalmente il nat si usa quando ci sono meno (o molti meno) indirizzi esterni a disposizione rispetto ai client della propria rete (attualmente, gli indirizzi ip hanno un costo, e non indifferente).

Beh, la soluzione che è stata trovata per risolvere questo problema si basa sull'idea di porta. Quando stabilite una connessione tcp verso un server, l'unica porta veramente importante è quella di destinazione, che identifica univocamente un servizio. Quella mittente, invece, non viene quasi mai guardata e spesso viene scelta a caso.

Con il NAT, il nostro firewall linux cambierà (se necessario) la porta sorgente dei pacchetti in uscita ed utilizzerà quella per identificare univocamente un client. Se la soluzione vi convince poco, provate a pensare al flusso dei pacchetti... il client A manda un pacchetto proveniente da A porta x al server B porta 80. Il

firewall riceve il pacchetto, cambia x in y (dove x può essere uguale a y, se tale porta non è utilizzata da nessuno), si segna che y è associata solo ad A e cambia l'indirizzo ip sorgente con il suo, il pacchetto arriva al server remoto R porta 80 come proveniente da F:y, e risponde creando un pacchetto proveniente da R:80 e destinato al firewall F porta y. Firewall riceve il pacchetto, vede che la connessione non è stata creata direttamente da lui (ogni sistema operativo sa quali connessioni ha iniziato) cerca nella tabella e vede che la porta y era in origine x ed era associata al client A. Prende il pacchetto, modifica il destinatario in modo che indichi A porta x e rimanda il pacchetto ad A. Sembra funzionare, no? L'idea è abbastanza furba e permette di identificare univocamente un client utilizzando qualche bit in più di informazioni. Il problema è che a volte questa tecnica non funziona:

- ad esempio, quando la trasmissione non fa uso del concetto di porta (solo le trasmissioni tcp e udp hanno delle porte associate). Fortunatamente però, oltre all'icmp, non esistono protocolli realmente utilizzati che non usino il concetto di porta, e gli errori sono facilmente associabili (non sono specifici al client: se un server non è raggiungibile, questo non sarà raggiungibile per *tutti* i client dietro al NAT box). Cioè, questi protocolli esistono ma vengono comunque normalmente gestiti non dal kernel ma da demoni dedicati per cui il problema non si pone.

- ad esempio, quando c'è bisogno che i pacchettini provengano da una porta specifica: un client, nascosto dal firewall, non saprà mai in realtà che porta sorgente utilizzerà il firewall per rispedire i suoi pacchetti.

- ad esempio, quando il protocollo si basa su connessioni effettuate da un server esterno verso un client interno. Per vedere dove sta il problema, proviamo a vedere un esempio. Ipotizziamo che il solito client FOO si colleghi a uno strano server con uno strano protocollo. Per via di questo strano protocollo, il server deve aprire una connessione verso la porta 789 del client e tenta quindi di collegarsi. Quello che però per lui è l'indirizzo del client è in realtà l'indirizzo del firewall, per via del NAT che ha cambiato l'indirizzo del mittente.

  Così facendo, il server tenterà di collegarsi al firewall, che però questa volta non avrà nessuna associazione nella sua tabella (l'associazione si crea solo per le connessioni che escono, e non è possibile indovinarle per quelle che entrano, a meno che non vengano stabilite a priori dall'amministratore) e sarà quindi costretto a buttare via il pacchetto rispondendo al server con un messaggio icmp con scritto qualcosa del tipo "la porta è chiusa e io non so chi tu sia".

Fortunatamente però il primo caso è estremamente raro (non credo sia stato mai nemmeno preso in considerazione il problema), il secondo capita ancora molto raramente, mentre per il terzo esistono delle soluzioni: il tracciamento delle connessioni.

Infatti, i moduli del kernel prima caricati che si occupavano di aprire o chiudere dinamicamente le porte sono anche in grado di aggiungere o rimuovere associazioni dalla famosa tabella, o di fare delle piccole modifiche al protocollo in modo che questo possa funzionare. Per fare questo però, se non avete compilato staticamente il kernel, dovete caricare anche i moduli "ip_nat_ftp", "ip_nat_irc" o "ip_nat_protocollo_strano_che volete usare" con il solito "modprobe".

Il problema che a volte si incontra, però, è che questi moduli non sono proprio facili da scrivere, per cui è relativamente facile trovarne per i protocolli standard, mentre non esistono per quei protocolli inventati dalla fantasia dei venditori di software di turno, che non sentono il dovere morale di rispettare degli standard.

Normalmente i problemi si incontrano solo per parti dei vari "protocolli" (ad esempio, dietro un firewall linux di questo tipo, col messanger sarebbe difficile farsi mandare dei file da altri utenti, pur funzionando per tutte le altre cose). E se per ftp e IRC (anche netmeeting, ma sperimentale) sono stati scritti dei moduli, per altri protocolli meno utilizzati credo che sarà difficile che qualcuno si occupi di realizzarli.

In questi casi, un "socks server" vi consentirà il più delle volte di rendere felici i vostri utenti. Il prezzo da pagare è che un socks server di base è molto aperto ed una configurazione errata o superficiale di questo

potrebbe vanificare completamente i vostri sforzi. In più, preparatevi a ricevere chiamate a tutte le ore del tipo "come si fa ad impostare il socks server su fuffolo 3.0 piuttosto che kazzat 2.8"...

9.1 Dalla teoria alla pratica

Passando dalla teoria alla pratica, occorre prima di tutto parlare in maniera un pochettino più approfondita dei target di default messi a disposizione dalla tabella di NAT.

- SNAT: serve per modificare il mittente di un pacchetto (l'indirizzo ip sorgente). Viene utilizzato nella catena di POSTROUTING proprio perché il cambiare il mittente non interferisce sulla scelta dell'interfaccia da utilizzare per raggiungere il destinatario. Consente anche di modificare manualmente la porta sorgente di un pacchetto. Attenzione! Funziona solo con indirizzi ip statici!
- DNAT: serve per modificare il destinatario di un pacchetto (l'indirizzo ip destinatario). Viene utilizzato nella catena di PREROUTING in quanto la modifica deve essere effettuata prima di decidere da che parte fare uscire il pacchettino. E' possibile inoltre specificare una diversa porta di destinazione.
- MASQUERADE: è un tipo particolare di SNAT: fa in modo che i pacchettini abbiano come mittente l'indirizzo IP della interfaccia di rete dalla quale usciranno. Si utilizza nella tabella di POSTROUTING (modifica il mittente) e viene utilizzato al posto dello SNAT su interfacce con IP assegnato dinamicamente, ad esempio tramite dhcp o bootp.
- REDIRECT: è una versione semplificata del DNAT mantenuta più che altro per motivi di compatibilità e fa in modo che il pacchettino venga rediretto sulla macchina locale (127.0.0.1) ad una porta specifica. Equivale a qualcosa del tipo: -j DNAT –to 127.0.0.1:porta <–> -j REDIRECT –to-ports porta.

Esistono molti altri target che possono essere aggiunti tramite il patch-o-matic adatti a tutte le esigenze. Date un'occhiata al manuale di iptables ed alle informazioni mostrate sullo schermo durante l'aggiornamento di iptables per maggiori informazioni.

Tornando invece al nostro esempio (presentato nei paragrafi sul filtraggio), è per noi indispensabile utilizzare il NAT in quanto:

- dobbiamo "deviare" tutti le connessioni web verso internet sul nostro proxy server che si trova nella dmz (transparent proxy).
- dobbiamo fare in modo che la nostra rete interna che utilizza indirizzi ip di classe 192.168.200.x (indirizzi per reti private) possa navigare su internet.

Avendo però spesso a che fare con reti di discrete dimensioni dove il dhcp è raramente utilizzato, mi piace aggiungere alcune regole per rendere le macchine "indipendenti" dalle modifiche del provider di turno o eventuali riconfigurazioni della rete (vi è mai capitato di dover cambiare l'ip del dns su n macchine diverse, dove n tende ad infinito?).

Per fare questo normalmente si possono utilizzare diversi approcci:

- Se la rete è divisa in diverse sottoreti (subnet), si possono "inventare" due indirizzi ip che non risiedano su nessuna delle nostre reti ed impostare tutti i client per utilizzare quegli indirizzi ip come dns primario e secondario.
- Si possono semplicemente deviare **tutti** i pacchetti destinati ad un dns verso il dns reale (c'è il problema del dns secondario, però).
- Semplicemente ignorare il problema inizialmente ed in caso di cambio di ISP aggiungere due regole per deviare il traffico sui nuovi dns.

Trattandosi comunque di regole abbastanza semplici ed interessanti dal punto di vista didattico, verranno presentate tutte e tre le soluzioni (la prima è quasi perfettamente equivalente alla terza).

Vediamo quindi le nostre prime regole di NAT:

```
77: iptables -t nat -A PREROUTING -p tcp -i eth0 --dport www -j DNAT --to nostro.proxy.server:8080
78: iptables -t nat -A POSTROUTING -o eth2 -s 192.168.200.0/24 -j SNAT --to 123.45.68.1
```

Ok, la prima regola dice, a parole, di inserire una regola nella tabella di nat per intercettare tutti i pacchetti provenienti da eth0 e destinati ad un server www (porta 80 – prima che venga deciso da che interfaccia farli uscire) e di deviarli sul nostro.proxy.server porta 8080.

Attenzione però che proxy server come squid devono essere configurati per poter accettare connessioni deviate in questo modo, in quanto non seguono perfettamente lo standard utilizzato dai proxy (leggete la documentazione di squid!).

Tornando a parlare delle regole, la seconda dice di modificare il mittente (l'indirizzo ip sorgente) dei pacchetti uscenti da eth2 e provenienti dalla nostra LAN (192.168.200.0/24) con l'indirizzo ip esterno del nostro firewall.

A proposito di queste due regole ci sono alcune cose importanti da dire:

- Prima di tutto, nella prima regola si è reso necessario indicare la scheda di rete sorgente in modo da non deviare le richieste del nostro proxy server. Per la cronaca: quando facciamo una richiesta al nostro proxy server, questo se non ha le pagine richieste in cache dovrà collegarsi al server web richiesto, e dobbiamo quindi stare attenti a non deviare anche le richieste provenienti dal proxy (si creerebbe una sorta di circolo vizioso (loop) né molto bello né molto positivo per la salute della nostra rete).
- Nella seconda regola, invece, è importante fare in modo che soltanto i pacchetti provenienti dalla nostra rete interna (e non dalla DMZ!) vengano modificati e soltanto quelli uscenti verso internet (quelli destinati al nostro proxy non devono essere toccati: saremo così in grado di fare un minimo di statistiche sui siti più interessanti per i nostri utenti).
- Nella seconda regola, è fondamentale fare in modo che i pacchetti abbiano come mittente uno degli indirizzi ip esterni del firewall. Per chi si accontenta di poche parole, basti dire che in caso contrario non funzionerebbe. Per tutti gli altri, invece, il motivo sta nell'arp, address resolution protocol. Ovvero, se l'indirizzo ip non fosse uno di quelli del firewall (aggiunti con ip o ifconfig, per intenderci), il firewall non risponderebbe alle query ARP, ed i pacchetti uscirebbero felici dalla rete ma il gateway di default del nostro firewall non sarebbe in grado di rimandare i pacchetti indietro. Come sempre, linux lascia la massima libertà: sta all'amministratore evitare di farsi male con errori di questo tipo.
- Riferendosi al funzionamento del SNAT come prima descritto, a volte (in reti particolarmente affollate) potrebbe essere indispensabile utilizzare più indirizzi ip per effettuare lo SNAT (il numero di porte disponibili ed utilizzabili su un host per fare il giochettino descritto è abbastanza limitato). Il target fortunatamente consente di indicare più indirizzi ip (date un'occhiata al manuale di iptables). L'esperienza però mi insegna che in tali reti si supererebbe molto prima il numero massimo di connessioni tracciabili rispetto alle porte disponibili. Per aumentare questo numero, basta dare un:

  ```
  echo 16384 > /proc/sys/net/ipv4/ip_conntrack_max
  ```

  o, in maniera del tutto equivalente, scrivere qualcosa come

  ```
  net.ipv4.ip_conntrack_max=16384
  ```

  nel file /etc/sysctl.conf. Comunque sia, tenete presente che già una rete di 100-200 computer potrebbe aver bisogno di essere in grado di tracciare più connessioni, mentre non mi è ancora capitato di dover utilizzare più di un indirizzo ip in uscita.

Il kernel comunque, vi avviserà in caso fosse necessario aumentare questi limiti, con un messaggio nei log per ogni connessione scartata.

- Nell'ultima regola, infine, se avessimo avuto a disposizione una connessione dial-up (come isdn) o tramite adsl con indirizzi ip assegnati dinamicamente, si sarebbe dovuto usare come target "MASQUERADE", senza però specificare alcun indirizzo ip sorgente. Attenzione! Lo SNAT proprio non funziona su interfacce con ip dinamici assegnati! Potete vedere un esempio di come utilizzare il target MASQUERADE in uno dei paragrafi introduttivi...
- Per quanto riguarda le regole di NAT e filtraggio, è fondamentale notare (per poter ottenere un firewall funzionante) che le **regole di firewalling, inserite nella tabella di filter, vedranno i pacchetti con i loro indirizzi reali**. Nel caso di DNAT, per esempio, il filtro vedrà pacchetti con la destinazione modificata (in questo caso, il proxy server). In caso invece di SNAT, il filtro vedrà come mittente l'indirizzo reale del mittente (non quello modificato), come se il codice di filtraggio fosse inserito dopo la catena di PREROUTING ma prima della catena di POSTROUTING.

Volendo fare il giochettino con il DNS, le due possibilità sono realizzabili utilizzando o queste regole:

```
80: iptables -t nat -A PREROUTING -p tcp --dport domain -j DNAT --to nostro.dns
81: iptables -t nat -A PREROUTING -p udp --dport domain -j DNAT --to nostro.dns
```

che deviano tutto il traffico verso un qualsiasi dns su nostro.dns, o queste altre, che similmente intercettano il traffico verso un dns da noi specificato e deviano il traffico verso il nostro dns reale:

```
83: iptables -t nat -A PREROUTING -p tcp -d un.dns --dport domain -j DNAT --to nostro.dns
84: iptables -t nat -A PREROUTING -p udp -d un.dns --dport domain -j DNAT --to nostro.dns
```

A questo punto, non vi rimane che divertirvi con i pacchetti che fluiscono dalle vostre reti...

10 Mettere tutto insieme

Ok, abbiamo visto come configurare dei filtri ed il NAT. Mancano però ancora un paio di cosettine di cui sarebbe il caso di parlare.

Prima di tutto, soprattutto quando staremo correggendo gli errori nelle nostre regole, caricheremo e ricaricheremo il nostro script finché non vedremo queste regole funzionare (molte volte). Il fatto è che creando delle catene, non chiediamo di svuotarle nel caso queste esistano già, per cui continueremmo ad aggiungere regole su regole, non semplificandoci affatto la correzione degli errori.

Potreste quindi voler aggiungere qualcosa come:

```
for table in nat mangle filter
do
   iptables -t $table -F
   iptables -t $table -X
done
```

all'inizio del vostro script... che elimina tutte le catene e tutte le regole inserite in ogni tabella. Allo stesso modo, potreste voler aggiungere qualcosa come

```
(
for interface in eth0 eth1 eth2 eth3
do
```

```
  ip route flush dev $interface
  ip addr flush dev $interface
  ip link set down dev $interface
done;
) &>/dev/null
```

Sempre all'inizio del nostro script, per riportare tutte le interfacce e le impostazioni ad uno stato conosciuto, in modo che non si vadano in continuazione ad aggiungere route o indirizzi, che, aggiunti per ultimi, verrebbero ignorati in favore di vecchie configurazioni che si combinerebbero e mischierebbero in maniera poco prevedibile. Comunque, "ip route flush dev ethx" elimina tutti i route relativi ad ethx, "ip addr flush" elimina tutti gli indirizzi dell'interfaccia ed infine "ip link set down" spegnere la scheda di rete.

Infine, dobbiamo attivare le nostre interfacce di rete, configurare la tabella di routing ed abilitare il "forwarding", dicendo al kernel di iniziare a trasmettere i pacchetti da una scheda di rete all'altra:

```
86: ip link set up dev eth0
87: ip link set up dev eth1
88: ip link set up dev eth2

89: ip route add default via 123.45.68.254

90: echo 1 > /proc/sys/net/ipv4/ip_forward
```

In questo caso, ip_forward non può essere aggiunto al file sysctl.conf, in quanto vogliamo che il forwarding venga attivato solo dopo che abbiamo impostato correttamente le regole di firewalling.

Non ci resta quindi che creare in /etc/rcS.d/ o rc2.d (a secondo della distribuzione) un link al file appena completato, in modo che questo venga eseguito ad ogni reboot, con qualcosa come "ln -s ../init.d/ifconffw ./S40ifconffw" oppure "update-rc.d ifconffw start 40 S" in Debian.

10.1 Configurazione dei client

Se il firewall ha un indirizzo ip diverso per ogni interfaccia ed utilizza come gateway il gateway fornito dal vostro provider, i vostri client dovranno utilizzare come gateway, come cancello che gli permetta di connettersi ad altre reti, l'indirizzo ip del vostro firewall relativo alla rete cui il vostro client è collegato.

Un server collegato alla dmz dovrà quindi utilizzare come gateway l'indirizzo ip della vostra interfaccia collegata alla DMZ. Allo stesso modo un client collegato alla LAN dovrà utilizzare come gateway l'indirizzo ip dell'interfaccia collegata alla lan. Per il resto, la configurazione dei client dipenderà principalmente dai server presenti (proxy, socks...) e dai giochini che avrete deciso di fare col NAT.

11 Modifiche on the fly - iptables da riga di comando

Sebbene sia veramente poco piacevole, è possibile utilizzare iptables direttamente da riga di comando.

Nonostante questo, alcune di queste funzioni possono risultare molto utili in fase di testing. Ad esempio, dovendo fare delle misure di banda può essere utile consentire temporaneamente solo il traffico tra determinate macchine, e non è il caso di perdere tempo a modificare file di configurazione o script dalla provata affidabilità. Ok, iniziamo quindi col vedere le regole attive sul kernel da me correntemente utilizzate, col comando "iptables -nL", dove "n" dice di non risolvere i nomi (passando da indirizzi ip a nomi di domini) e "L" di "listare" o elencare (in italiano) tutte le regole:

```
Chain INPUT (policy DROP)
```

```
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0          tcp dpt:22
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0          state RELATED,ESTABLISHED
REJECT     tcp  --  0.0.0.0/0            0.0.0.0/0          reject-with tcp-reset

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

-n lo uso principalmente perché altrimenti ci metterebbe molto tempo a generare l'output (tentando di connettersi al database), per il resto, credo che l'output in se stesso credo che risulti abbastanza autoesplicativo... un problema di "iptables -nL" (rispetto ad usare un file di configurazione) è che molti dettagli non vengono mostrati e comunque la leggibilità è molto minore.

Per avere più dettagli, comunque, è possibile utilizzare l'opzione "-v" o "-vv", con un comando simile a "iptables -v -nL":

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
41512   83M ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0          tcp dpt:22
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0          state RELATED,ESTAB
    0     0 REJECT     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0          reject-with tcp-res

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 41513 packets, 83M bytes)
 pkts bytes target     prot opt in     out     source               destination
```

ancora, l'output è abbastanza autoesplicativo. Se state leggendo il pdf notate però come le righe vengono troncate. Immaginate su un terminale con le classiche 80 colonne cosa verrebbe fuori.

Se siete interessati ad un'unica catena, potete specificarne il nome dopo il "-nL". Con qualcosa di simile a "iptables -vv -nL INPUT" vedreste solo la catena di "INPUT" ed un dump delle informazioni del kernel. Come altre opzioni generiche, potete utilizzare "-x", che mostrerà i valori esatti dei contatori, mentre "-Z" li azzererà, oppure "-t nometabella" per visualizzare altre tabelle.

11.1 Manipolare le catene

Già conosciamo i parametri:

- *-N* – per creare nuove catene
- *-X* – per eliminare una catena (tutte, se non viene specificato un argomento)
- *-F* – per eliminare tutte le regole di una catena (tutte, se non viene specificato un argomento)
- *-A* – per aggiungere regole alle catene
- *-P* – per impostare le politiche

In più, vi possono tornare utili i comandi

- *-I* – per inserire una nuova regola in una determinata posizione di una catena
- *-R* – per sostituire una regola con un'altra
- *-D* – per rimuovere una regola

Cerchiamo però di vedere questi parametri nella pratica. Ipotizzando di avere una situazione come quella mostrata sopra con "iptables -nL", immaginiamo di voler inserire una regola dopo quella relativa ad ssh (catena di INPUT, porta 22, riga 2) per consentire l'accesso ad un web server installato sul mio portatile. In questo caso, potremmo dare un comando come

```
iptables -I INPUT 3 -p tcp --dport 80 -j ACCEPT
```

Il 3 indica che questa regola deve essere inserita nella terza riga, spostando tutte le altre righe più in basso. Dopo questo comando, "iptables -nL INPUT" mostrerebbe qualcosa come

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0          tcp dpt:22
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0          tcp dpt:80
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0          state RELATED,ESTABLISHED
REJECT     tcp  --  0.0.0.0/0            0.0.0.0/0          reject-with tcp-reset
```

proprio come volevamo. Immaginiamo ora di spostare il web server sulla porta 443, e di voler rimpiazzare la regola appena inserita con qualcosa come "-p tcp –dport 443 -j ACCEPT". In questo caso sarebbe sufficiente utilizzare

```
iptables -R INPUT 3 -p tcp --dport 443 -j ACCEPT
```

per ottenere qualcosa come

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0          tcp dpt:22
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0          tcp dpt:443
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0          state RELATED,ESTABLISHED
REJECT     tcp  --  0.0.0.0/0            0.0.0.0/0          reject-with tcp-reset
```

Infine, per rimuovere la regola, basterebbe qualcosa come

```
iptables -D INPUT 3
```

Se siete meno pigri di me, potete utilizzare anche

```
iptables -D INPUT -p tcp --dport 443 -j ACCEPT
```

che porterebbe esattamente allo stesso risultato.

11.2 Salvare le modifiche

Versioni relativamente recenti di iptables mettono anche a disposizione due tools: iptables-save ed iptables-restore. Il primo, stampa a video in un formato standard tutte le regole di firewalling, mentre il secondo è in grado di leggere delle regole in questo formato standard da standard input e di impostarle nel kernel. Ad esempio, per salvare delle regole correntemente utilizzate dal kernel, è possibile utilizzare il comando:

```
# iptables-save > nomefile
```

mentre per ripristinarle il comando

```
# iptables-restore < nomefile
```

In Debian, è possibile utilizzare "/etc/init.d/iptables save nomefile" oppure "/etc/init.d/iptables load nomefile". Utilizzando come nomefile "active", le regole verranno automaticamente caricate ad ogni accensione della macchina. In quest'ultimo caso, potreste voler dare un'occhiata a `/etc/default/iptables`.

12 Conclusioni

Le funzioni viste qua sopra non costituiscono probabilmente neanche il 25% di tutto ciò che il kernel 2.4 può fare. Mi basti ricordarvi che delle tre tabelle a disposizione ne abbiamo utilizzate soltanto 2 e che molte delle nuove funzionalità sono state soltanto accennate, senza dimenticare poi che nulla è stato visto in proposito del QoS e del fair queuing. Giusto per stuzzicare la vostra curiosità, la tabella di mangle serve per "maneggiare" i pacchetti in maniera più sottile, in modo che comandi come ip e tc possano interagire tra di loro distinguendo pacchetti scelti da iptables o per modificare i parametri del pacchetto (modulo ttl & co.).

Giusto per fare un accenno, il fair queuing si gestisce attraverso il comando tc, e consente di suddividere tutti i pacchetti in transito in classi, dove ogni classe può avere una certa larghezza di banda riservata o una certa priorità. Si potrebbe poi parlare di funzioni come il proxy_arp, che in combinazione con il modulo ttl consentono di creare dei firewall (praticamente) invisibili.