Didacticiel sur Iptables, version 1.2.0

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières

Didacticiel sur Iptables, version 1.2.0

Oskar Andreasson

<oan@frozentux.net>

Dédicaces

Je voudrais dédier ce document à ma merveilleuse soeur pour m'avoir inspiré et donné ses conseils en retour. Elle est une source de joie et un rayon de soleil quand j'ai besoin d'elle. Merci !

Ensuite, j'aimerais dédicacer ce travail à tous les courageux développeurs et mainteneurs de Linux. Ce sont eux qui font exister ce fabuleux système d'exploitation.

A propos de l'auteur

Pour me présenter, je suis une personne qui détient beaucoup trop d'ordinateurs anciens. Je possède mon propre réseau local (`LAN`) et je veux conserver toutes mes machines connectées à Internet, en assurant en même temps que mon réseau reste sécurisé. De ce point de vue, le nouvel iptables est une amélioration intéressante de l'ancien ipchains. Avec ipchains, vous pouviez correctement sécuriser un réseau en détruisant les paquets non destinés à certains ports. Cependant, des échanges de type passif avec `FTP` ou sortants avec `DCC` sous `IRC` engendraient des problèmes. Ils attribuent des ports sur le serveur, en informent le client, puis laissent le client s'y connecter. Au départ, lorsque je me suis intéressé au code d'iptables, il avait quelques défauts de jeunesse, et à certains égards j'ai trouvé que ce code n'était pas tout à fait mûr pour une utilisation en production. Aujourd'hui, je recommande à tous les utilisateurs d'ipchains, ou même du vénérable ipfwadm, etc., d'effectuer une mise à jour – à moins qu'ils soient satisfaits des possibilités de leur code actuel et que celui−ci corresponde à leur besoin.

Exploration de ce document

Cet ouvrage a été rédigé simplement afin que tous puissent accéder au monde merveilleux d'iptables. Il n'a jamais été destiné à rassembler des informations de bogues de sécurité sur iptables ou Netfilter. Si vous trouvez des bogues ou des comportements insolites dans iptables ou une de ses composantes, vous devriez contacter la liste de diffusion de Netfilter et signaler le problème. On vous informera alors de la validité d'un bogue et de son éventuelle correction. Il y a très rarement de véritables bogues de sécurité identifiés dans iptables ou Netfilter, malgré tout, un ou deux peuvent se faufiler de temps en temps. Ils sont mis en évidence sur la page principale du site de Netfilter, et c'est là que vous devriez vous rendre pour rassembler de l'information sur ce sujet.

Ainsi, les exemples de règles fournis avec ce didacticiel ne sont pas écrits pour tenir compte des bogues actuels de Netfilter. L'objectif majeur est de rendre la méthode de configuration des règles suffisamment simple pour être en mesure de résoudre tous les problèmes rencontrés. Par exemple, ce didacticiel n'apprend pas comment fermer le port HTTP pour répondre à une vulnérabilité d'Apache dans la version 1.2.12 (ce cas est néanmoins traité mais pour une autre raison).

Ce document a été écrit afin de fournir à tous une introduction efficace et simple pour bien démarrer avec iptables, mais en même temps il a été créé pour être aussi complet que possible. Il ne contient aucune cible ou correspondance appartenant à patch−o−matic pour la simple raison qu'il serait trop fastidieux de tenir une telle liste à jour. Si vous êtes intéressés par les mises à jour de patch−o−matic, vous devriez lire les informations fournies avec, ainsi que les autres documentations disponibles sur le site de Netfilter.

Préalables

Des connaissances préalables sur Linux/Unix sont nécessaires, en particulier l'écriture de scripts shell, la compilation de son propre noyau, et quelques notions sur son fonctionnement interne.

J'ai essayé autant que possible d'éliminer tous les préalables nécessaires pour comprendre pleinement ce document, mais en pratique il est inévitable de posséder un minimum de connaissances.

Conventions utilisées dans ce document

Les conventions suivantes sont utilisées dans ce document lorsqu'il s'agit de commandes, de fichiers ou d'autres informations spécifiques.

- Les extraits de code ou les résultats de commandes sont affichés comme suit, avec une police de largeur constante, et les commandes entrées par l'utilisateur en caractères gras :

```
[blueflux@work1 neigh]$ ls
default  eth0  lo
[blueflux@work1 neigh]$
```

- Dans ce didacticiel, les noms de commandes et de programmes sont tous indiqués en ***caractères gras***.
- Les éléments du système, comme le matériel mais aussi les composantes internes du noyau comme l'interface de bouclage, sont tous indiqués en `caractères italiques`.
- Une sortie d'écran est mise en forme de `cette façon` dans le texte.
- Les noms de fichiers ou de chemins dans le système de fichiers sont indiqués comme `/usr/local/bin/iptables`.

Chapitre 1. Introduction

1.1. Motivations

A l'origine, j'ai constaté un vide important dans les guides pratiques (« Howto's ») disséminés un peu partout,

avec un manque d'informations notable sur les fonctions d'iptables et de Netfilter pour les nouveaux noyaux Linux de la famille 2.4.x. Par conséquent, je vais tenter de répondre à des interrogations courantes concernant de nouvelles possibilités comme la correspondance d'état. La plupart du temps, les situations seront appuyées par un fichier d'exemple rc.firewall.txt que vous pourrez utiliser dans vos scripts `/etc/rc.d/`. Effectivement, ce fichier était à l'origine issu du guide pratique du camouflage, pour ceux d'entre–vous qui l'auraient reconnu.

Par la même occasion, il existe un petit script que j'ai écrit au cas où vous peiniez autant que moi lors de la configuration. Il est disponible sous le nom rc.flush–iptables.txt.

1.2. Contenu

Initialement rédigé pour boingworld.com, qui fût un site de news consacré à Amiga/linux pour un petit nombre de personnes, y compris moi, il s'agissait d'un très petit didacticiel. En fonction du grand nombre de lecteurs et de commentaires que j'ai reçu, j'ai continué à écrire sur ce sujet. Le version originale faisait à peu près 10–15 pages au format A4 dans sa version imprimée. Un grand nombre de personnes m'ont aidé, pour la correction orthographique, bugs, etc. Au moment où j'écris ceci, le site http://iptables–tutorial.frozentux.net a enregistré plus de 600.000 connections.

Ce document est conçu pour vous guider pas–à–pas dans la méthode de configuration et il devrait vous aider à comprendre davantage le paquetage d'iptables. La plupart des exemples s'appuie sur le fichier rc.firewall, puisqu'il m'a semblé être un bon point de départ pour apprendre à se servir d'iptables. J'ai décidé de suivre simplement les chaînes fondamentales, et à partir de là, de poursuivre en approfondissant chacune des chaînes traversées dans l'ordre logique. Cette approche rend le didacticiel un peu plus difficile à suivre, mais elle a l'avantage d'être plus cohérente. Chaque fois que quelque–chose vous semble difficile à comprendre, replongez–vous dans ce didacticiel.

1.3. Termes spécifiques

Dans ce document, certains termes méritent des explications détaillées avant d'être abordés. Cette section cherche à couvrir les plus évidents et présente la façon dont ils sont utilisés ici.

Connexion – Se réfère généralement, dans ce document, à une série de paquets en relation entre eux. Ces paquets interragissent entre eux en établissant une sorte de connexion. Une connexion est en d'autres termes une série de paquets échangés.

DNAT – Traduction d'adresse réseau de destination ou « Destination Network Address Translation ». Le DNAT fait référence à la technique de traduction de l'adresse IP de destination d'un paquet. On l'utilise conjointement avec du SNAT pour permettre à plusieurs hôtes de partager une même adresse IP connectée à Internet, et pour continuer à offrir des services de type serveur. Typiquement, il suffit d'attribuer des ports différents avec une adresse IP utilisable sur Internet, puis de signaler au routeur Linux où expédier le trafic.

Espace noyau – C'est plus ou moins l'opposé de l'espace utilisateur. Ceci implique les actions effectuées dans le noyau, et non en dehors du noyau.

Flux (« Stream ») – Ce terme fait référence à une connexion qui envoie et reçoit des paquets qui sont d'une certaine manière en relation les uns avec les autres. Typiquement, j'ai employé ce terme pour toute connexion qui envoie deux paquets ou plus dans les deux sens. Pour le protocole TCP, ce terme peut désigner une connexion qui envoie un paquet SYN, puis répond avec un autre de type SYN/ACK; mais il peut aussi désigner une connexion qui envoie un paquet SYN, puis répond avec un paquet ICPM de type hôte inaccessible (« ICMP Host unreachable »). Bref, j'ai souvent utilisé ce terme avec inexactitude.

SNAT – Traduction d'adresse réseau de source ou « Source Network Address Translation ». Ce terme fait référence aux techniques mises en oeuvre pour traduire une adresse de source en une autre dans un paquet.

Ceci permet à plusieurs hôtes de partager une même adresse IP connectée à Internet, c'est utile pour compenser le manque d'adresses IP disponibles avec le protocole IPv4 (mais IPv6 vient résoudre ce problème).

État – Ce terme fait référence à l'état d'un paquet, en accord avec la RFC 793 – Transmission Control Protocol ou avec les états utilisateur utilisés dans Netfilter/iptables. Notez que les états utilisés, en interne et en externe, ne respectent pas scrupuleusement la spécification de la RFC 793. La raison principale provient du fait que Netfilter a dû faire plusieurs hypothèses sur les connexions et les paquets.

Espace utilisateur (« User space ») – Cette expression permet de d'assigner tout ce qui a lieu à l'extérieur du noyau. Par exemple, la commande ***iptables –h*** s'exécute en dehors du noyau, alors que ***iptables –A FORWARD –p tcp –j ACCEPT*** se déroule (en partie) à l'intérieur, puisqu'une nouvelle règle est ajoutée à la table de règles.

Domaine de l'utilisateur – Voir Espace utilisateur.

Paquet – Une unité envoyée sur le réseau, contenant une partie en–tête et une partie de données. Par exemple, un paquet IP sur un paquet TCP. Dans les RFC (Request For Comments) un paquet n'est pas généralisé ainsi, au lieu de cela les paquets sont appelés datagrammes, tandis que les paquets TCP sont appelés segments. J'ai choisi de tout nommer paquet dans ce document pour simplifier.

Segment – Un segment TCP est à peu près la même chose qu'un paquet, c'est en fait un paquet TCP.

Chapitre 2. Rappel TCP/IP

Iptables est un outil d'apprentissage très puissant. Parmi d'autres choses, vous devez avoir une très bonne compréhension du protocole TCP/IP.

Ce chapitre a pour but l'explication de ce que vous « devez savoir » sur TCP/IP avant de commencer à utiliser iptables. Les choses que nous allons aborder concernent les protocoles IP, TCP, UDP et ICMP, leurs en–têtes, et l'utilisation générale de chacun des ces protocoles et comment ils sont corrélés entre eux. Iptables fonctionne au niveau des couches Internet et transport, et à cause de ça, ce chapitre mettra l'accent sur ces couches.

Iptables peut aussi fonctionner sur des couches plus hautes, comme la couche application. Cependant, il n'a pas été conçu pour ça, et ne devrait pas être utilisé pour ce genre d'usage. J'en parlerai d'avantage dans le chapitre Introduction au filtrage IP.

2.1. Couches TCP/IP

Comme déja établi, TCP/IP est multi–couches. Ceci indique que nous avons une fonctionnalité sur un niveau, et une autre à un autre niveau, etc. La raison pour laquelle nous avons toutes ces couches est très simple.

La raison principale est que l'architecture globale est très extensible. Nous pouvons ajouter de nouvelles fonctionnalités aux couches application, par exemple, sans avoir a réimplémenter l'ensemble du code TCP/IP, ou inclure une pile TCP/IP complète dans l'application. Ainsi il est inutile de réécrire chaque programme chaque fois que nous installons une nouvelle carte d'interface réseau.

Note

Quand nous parlons de code TCP/IP lequel est intégré dans le noyau, nous parlons souvent de pile TCP/IP. La pile TCP/IP indique toutes les sous–couches utilisées, depuis la couche réseau jusqu'à la couche application.

Il existe deux architectures de base lorsque nous parlons de couches. Une des deux est le modèle OSI (Open System Interconnect) et consiste en 7 couches. Nous les verrons superficiellement ici, nous nous intéressons plus particulièrement aux couches TCP/IP. Cependant, sur un plan historique il est intéressant de le connaître, en particulier si vous travaillez avec plusieurs types de réseaux différents. Voir la liste dans le OSI Reference Model.

1. Couche Application
2. Couche Présentation
3. Couche Session
4. Couche Transport
5. Couche Réseau
6. Couche Liaison
7. Couche Physique

Un paquet que nous envoyons, parcourt du sommet à la base cette liste, chaque couche ajoutant ses propres *en−têtes* au paquet, ce que nous appelons la phase d'encapsulation. Lorsque le paquet rejoint sa destination il parcourt en sens inverse la liste et les en−têtes sont supprimés du paquet, un à un, chaque en−tête donnant à l'hôte de destination toute l'information nécessaire jusqu'à ce que le paquet joigne l'application ou le programme pour lequel il était destiné.

Le second et plus intéressant standard est le protocole TCP/IP, comme indiqué dans la liste TCP/IP architecture. Il n'y a pas d'accord universel en ce qui concerne le nombre de couches dans l'architecture TCP/IP. Cependant, on considère généralement qu'il y a de 3 à 5 couches disponibles, nous en verrons 4 pour simplifier.

1. Couche Application
2. Couche Transport
3. Couche Internet
4. Couche Réseau

Comme vous pouvez le voir, l'architecture du protocole TCP/IP est très proche du modèle OSI. De même qu'avec le modèle OSI, nous ajoutons et soustrayons les en−têtes pour chaque couche.

Par exemple, utilisons une des analogies les plus communes pour les machines modernes en réseau, la lettre par courrier postal. Chaque chose est effectuée par étape, identique en TCP/IP.

Vous désirez envoyer une lettre à quelqu'un en lui demandant comment il va, et qu'est−ce qu'il fait. Pour celà, vous devez poser des questions. Les questions seront situées à l'intérieur de la couche Application.

Après ceci vous écrirez les questions sur une feuille de papier que vous mettrez dans une enveloppe sur laquelle vous écrirez l'adresse de destination. Peut−être quelque chose comme :

Attn: John Doe

C'est l'équivalent de la couche Transport en TCP/IP.

À ce niveau nous écrivons l'adresse sur l'enveloppe, comme ceci :

V. Andersgardsgatan 2
41715 Gothenburg

Ça se passe dans l'analogie comme dans la couche Internet. La couche Internet contient les informations indiquant comment joindre le destinataire, ou l'hôte, dans un réseau TCP/IP. De la même façon qu'une enveloppe avec une adresse.

L'étape finale est de poster l'enveloppe dans une boîte aux lettres. Ce qui équivaut à peu près à envoyer un paquet dans la couche Réseau. La couche Réseau contient les fonctions et les routines pour accéder au réseau physique par lequel le paquet sera transporté.

Quand finalement nous recevons la lettre, nous la retirerons de l'enveloppe (décapsulation). La lettre que nous recevons peut parfois demander une réponse ou non. Dans certains cas il peut y avoir une réponse du destinataire, alors le destinataire devient expéditeur, et l'expéditeur devient destinataire.

Note

Il est très important de comprendre que Iptables est spécifiquement construit pour travailler à l'intérieur des en−têtes des couches Internet et Transport. Il est possible de créer quelques filtres très basiques avec Iptables dans les couches Application et Réseau, mais il n'a pas été conçu pour cela, ni approprié.

Par exemple, si nous utilisons une correspondance de chaîne et l'apparions pour une chaîne spécifique dans le paquet, disons ***get /index.html***. Ceci fonctionnera ? Normalement oui. Cependant, si la taille du paquet est très petite, cela ne marchera pas. La raison en est que Iptables est construit pour fonctionner sur une base *par paquet*, qui indique que si la chaîne est divisée en plusieurs paquets séparés, Iptables ne verra pas l'ensemble de la chaîne. Pour cette raison, il est mieux d'utiliser un proxy pour filtrer au niveau de la couche Application. Nous verrons ces problèmes en détail plus tard dans Introduction au filtrage IP.

Étant donné que Iptables et Netfilter opèrent principalement sur les couches Internet et Transport, ce sont les couches sur lesquelles nous insisterons le plus dans ce chapitre. Sous la couche Internet, nous verrons presque exclusivement le protocole IP. Il existe quelques ajouts à ceci, comme, par exemple, le protocole GRE, mais ils sont très rares. À cause de tous ces facteurs nous nous concentrerons sur le protocole IP de la couche Internet, et TCP, UDP, ICMP de la couche Transport.

Note

Le protocole ICMP est actuellement une sorte de mélange entre les deux couches. Il fonctionne dans la couche Internet, mais il possède exactement les mêmes en−têtes que le protocole IP, mais aussi quelques en−têtes supplémentaires. Nous verrons ceci plus en détail plus loin dans Caractéristiques ICMP.

2.2. Caractéristiques IP

Le protocole IP résides dans la couche Internet, comme nous l'avons déjà dit. Le protocole IP est le protocole dans la pile TCP/IP qui permet à votre machine, routeur, switch, etc. de savoir où un paquet spécifique est envoyé. Ce protocole est véritablement le coeur de toute la pile TCP/IP, et la base de tout sur Internet.

Le protocole IP encapsule le paquet de la couche Transport avec l'information du procole de la couche Transport, ainsi que d'autres informations utiles. Tout ceci, bien sûr, très précisément standardisé. Nous allons en parler dans ce chapitre.

Le protocole IP possède un couple de fonctionnalités de base qu'il doit être capable de traiter. Il doit être capable de définir le datagramme, lequel est le bloc de construction suivant créé par la couche Transport (ce qui en d'autres termes peut être TCP, UDP ou ICMP par esemple). Le Protocole IP définit aussi le système d'adressage Internet que nous utilisons aujourd'hui. Ceci indique que le protocole IP définit comment les hôtes peuvent se joindre entre eux, il indique aussi comment nous pouvons router les paquets, bien sûr. Les adresses dont nous parlons sont généralement appelées adresses IP. Usuellement, quand nous parlons d'adresses IP, nous parlons de chiffres avec des points (ex. 127.0.0.1). C'est principalement pour rendre l'adresse IP plus lisible pour l'oeil humain, car l'adresse IP est actuellement un champ de 32 bits de 1 et de 0 (127.0.0.1 pourrait désormais être lu comme 01111111000000000000000000000001 dans l'en−tête IP).

Le protocole IP doit aussi pouvoir décapsuler et encapsuler le datagramme IP (donnée IP) et envoyer ou recevoir le datagramme d'une couche Réseau, ou d'une couche Transport. Ceci peut sembler évident, mais parfois ce ne l'est pas. Au sommet de tout ça, il possède deux fonctions qu'il doit exécuter correctement, ce qui est particulièrement intéressant pour le pare−feu et le routage. Le protocole IP est responsable du routage des paquets depuis un hôte vers un autre. La plupart du temps sur des réseaux uniques, c'est un processus simple. Nous avons deux options différentes, soit le paquet est destiné au réseau local, soit passe par une passerelle. Mais lorsque vous commencez à travailler avec des pare−feux et des politiques de sécurité conjointement avec de multiples interfaces réseau et différentes routes, ce peut être casse−tête pour les administrateurs. La dernière des responsabilités du protocole IP est qu'il doit fragmenter et ré−assembler les datagrammes qui ont préalablement été fragmentés, ou qui nécessitent d'être fragmentés pour s'adapter à la taille du paquet pour la topologie du réseau où nous sommes connectés. Si ces fragments de paquet sont suffisamment petits, ils peuvent causer d'horribles maux de tête aux administrateurs réseau. Le problème est, qu'une fois qu'ils sont fragmentés, nous commençons à avoir des soucis pour lire même les en−têtes du paquet.

Astuce

Dans les séries 2.4 du noyau Linux, et Iptables, ceci ne représente pas un problème pour la plupart des pare−feux Linux. Le système de traçage de connexion utilisé par Iptables pour la vérification d'état, la traduction d'adresse, etc. doit être capable de lire les paquets défragmentés. À cause de ça, conntrack défragmente automatiquement tous les paquets avant qu'ils rejoignent la structure netfilter/iptables dans le noyau.

Le protocole IP est aussi un protocole en mode datagramme, ce qui indique que IP ne "négocie" pas une connexion. Un protocole orienté−connexion, en d'autres termes, négocie une "connexion" (appelée *poignée de main*) et lorsque toutes les données ont été envoyées, stoppe la connexion. TCP est un exemple de ce genre de protocole, cependant, il est implémenté au sommet du protocole IP. Il y a plusieurs raisons pour lesquelles il n'est pas orienté−connexion, mais parmi d'autres, une poignée de main n'est pas nécessaire à ce moment ce qui ne ferait qu'ajouter du temps système. Comme vous pouvez le voir, envoyer une requête et ensuite attendre un moment pour la réponse est préférable à envoyer un paquet pour dire que nous voulons établir une connexion, ensuite recevoir le réponse nous disant que la connexion est ouverte, et finalement accuser réception que nous sommes au courant que la connexion est ouverte, et *alors* envoyer la requête, et après renvoyer un autre paquet pour couper la connexion et attendre une autre réponse.

IP est également connu comme un *protocole incertain*, c'est à dire, il ne permet pas de savoir si un paquet a été reçu ou non. Il reçoit simplement un paquet depuis la couche transport et le passe à la couche réseau, et ne fait rien de plus. il peut recevoir un paquet en retour, lequel passe de la couche réseau au protocole IP et ensuite à la couche transport. Cependant, il ne vérifie pas si c'est un paquet en réponse ou si le paquet a été reçu dans un autre but. La même chose s'applique en terme d'incertitude IP comme pour le mode datagramme, ce qui nécessitera l'envoi d'un paquet supplémentaire en retour pour chaque paquet envoyé. Par exemple, considérons une consultation de table DNS. Nous envoyons une requête DNS au serveur de nom. Si nous ne recevons pas de réponse, nous savons que quelque chose ne fonctionne pas et renvoyons une requête de consultation, mais dans l'usage normal nous envoyons une requête et obtenons une réponse en retour. Ajouter de la fiabilité à ce protocole signifierait que la requête nécessite deux paquets (une requête et une confirmation que le paquet a été reçu) et ensuite deux paquets pour la réponse (une réponse et un accusé−réception comme quoi le paquet a été reçu). En d'autres termes, nous doublons le nombre de paquets nécessaires, et bien sûr doublons le nombre de données à transmettre.

2.3. En−têtes IP

Comme vous avez pû le comprendre dans l'introduction sur le protocole IP, un paquet IP contient différentes parties dans l'en−tête. Celui−ci est méticuleusement divisé en plusieurs parties, et chaque partie de l'en−tête est aussi petite que possible pour faire ce travail, ceci pour limiter le temps système au minimum. Vous verrez la configuration exacte d'une en−tête IP dans l'image En−têtes IP.

Note

Comprenez que les explications des différents en−têtes sont très brèves et que nous ne parlerons que des bases de ceux−ci. Pour chaque type d'en−tête dont nous parlons, nous indiquerons aussi sa RFC correspondante que vous devriez lire pour une meilleure compréhension et des explications techniques du protocole en question. En note marginale, les RFC (Request For Comments), ont aujourd'hui une signification totalement différente dans la communauté Internet. Elles définissent et standardisent l'ensemble de l'Internet, par rapport à ce pourquoi elles ont été écrites à l'origine. Au départ, il ne s'agissait que de simples RFC dont le but était de poser des questions pour avoir l'avis des autres chercheurs.

Le protocole IP est décrit principalement dans RFC 791 – Internet Protocol. Cependant, cette RFC est aussi mise à jour par la RFC 1349 – Type of Service in the Internet Protocol Suite, rendue obsolète par RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, mise à jour par RFC 3168 – The Addition of Explicit Congestion Notification (ECN) to IP et RFC 3260 – New Terminology and Clarifications for Diffserv.

Astuce

Comme vous pouvez le voir, tous ces standards peuvent être un peu difficiles à suivre. Un tuyau pour trouver les différentes RFC est d'utiliser les fonctions recherche disponibles sur RFC−editor.org. Dans le cas de IP, considérez que la RFC 791 est la RFC de base, toutes les autres sont simplement des mises à jour par rapport au standard. Nous parlerons de ceci plus en détail quand nous verrons les en−têtes spécifiques modifiés par ces nouvelles RFC.

Une chose à retenir est que, quelquefois, une RFC peut être obsolète (plus utilisée du tout). Normalement ceci signifie que la RFC a été si rigoureusement mise à jour, qu'il est mieux de tout revoir. Elle peut aussi devenir obsolète pour d'autres raisons. Quand une RFC devient obsolète, un champ est ajouté à la RFC d'origine qui pointe vers la nouvelle.

Internet Protocol

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Version	IHL	Type of Service/DSCP/ECN		Total Length			
2	Identification				Flags	Fragment Offset		
3	Time to Live		Protocol		Header checksum			
4	Source adress							
5	Destination adress							
6	Options						Padding	

Version – bits 0−3. C'est le numéro de version du protocole IP en binaire. IPv4 est nommé par 0100, tandis que IPv6 par 0110. Ce champ n'est généralement pas très utilisé pour le filtrage. La version décrite dans la RFC 791 est IPv4.

IHL (Longueur d'en−tête Internet) – bits 4−7. Ce champ nous indique la longueur de l'en−tête IP en 32 bits. Comme vous pouvez le voir, nous avons partagé l'en−tête (32 bits par ligne) dans l'image. Le champ Options est de longueur variable, ainsi nous ne pouvons jamais être absolument sûrs de la longueur totale de l'en−tête sans ce champ.

Type de Service, DSCP, ECN – bits 8−15. C'est une des zones les plus complexes de l'en−tête IP pour la simple raison qu'elle a été mise à jour 3 fois. Elle a toujours eu la même utilisation de base, mais l'implémentation a changé plusieurs fois. En premier le champ fut appelé Type de Service. Le Bit 0−2 du champ fut appelé champ Précédence. Le Bit était de délai normal/bas, le Bit 4 de débit normal/haut, le Bit 5 de fiabilité normale/haute et le Bit 6−7 réservé pour un usage futur. Ceci est toujours en usage sur de nombreux sites qui ont du matériel ancien, ce qui cause toujours certains problèmes pour l'Internet. Parmi d'autres choses, le Bit 6−7 est spécifié pour être placé à 0. Dans les mises à jour ECN (RFC, nous partons du principe que ces bits réservés sont désormais placés à la valeur 0. Mais nombre de pare−feux et routeurs ont été programmés pour vérifier si ces bits sont placés à 1, et si les paquets le sont, le paquet est supprimé. Aujourd'hui, c'est clairement une violation de la RFC, mais vous ne pouvez pas faire grand chose, excepté vous plaindre.

La seconde itération fut lorsque le champ a été modifié dans le champ DS comme spécifié dans la RFC 2474. DS pour Differentiated Services. Selon ce standard les bits 8−14 sont Differentiated Services Code Point (DSCP) et les deux bits restants (15−16) sont toujours inutilisés. Le champ DSCP est à peu près utilisé comme le champ ToS avant, pour marquer pour quel type de service ce paquet serait traité comme si le routeur en question ne faisait pas de différence entre eux. Un gros changement est que le matériel doit ignorer les bits inutilisés pour être pleinement conforme à la RFC 2474, ce qui signifie que nous somme délivrés des disputes comme expliqué précédemment, aussi longtemps que les fabriquants de matériel suivent cette RFC.

Le troisième, et pratiquement dernier, changement du champ ToS a été quand les deux bits inutilisés furent utilisés par ECN (Explicit Congestion Notification), comme défini dans la RFC 3168. ECn est utilisé pour permettre de savoir s'il existe une congestion des routeurs, avant il démarre la suppression des paquets, ainsi le noeud final pourra ralentir la transmission des données. Précédemment, la suppression de données était le seul moyen qu'avait un routeur pour annoncer qu'il était en surcharge, et les noeuds terminaux devaient faire un redémarrage lent pour chaque paquet supprimé, et ensuite accélérer de nouveau. Les deux bits sont appelés ECT (ECN Capable Transport) et CE (Congestion Experienced).

L'itération finale de l'ensemble est la RFC 3260 qui apporte quelques terminologies et clarifications nouvelles pour l'utilisation du système DiffServ. Elle n'améliore pas énormément de choses, sauf la terminologie. La RFC est également utilisée pour clarifier certains points qui ont été discutés entre développeurs.

Longueur totale – bits 16 – 31. Ce champ nous renseigne sur la taille des paquets en octets, incluant les en−têtes. La taille maximum est 65535 octets. La taille minimum d'un paquet est de 576 octets, sans prendre en compte si le paquet arrive en fragments ou non. Il est recommandé d'envoyer des paquets plus gros que cette limite seulement si il est garanti que l'hôte puisse les recevoir, selon la RFC 791. Cependant, actuellement la plupart des réseaux fonctionnent avec des tailles de paquets de 1500 octets. Ceci inclut la majorité des connexions ethernet et Internet.

Identification – bits 32 – 46. Ce champ est utilisé pour aider a réassembler les paquets fragmentés.

Fanions – bits 47 – 49. Ce champ contient des fanions mélangés appartenant à la fragmentation. Le premier bit est réservé, mais toujours inutilisé, et doit être placé à 0. Le second bit est placé à 0 si le paquet peut être fragmenté, et à 1 s'il ne peut pas être fragmenté. Le troisième et dernier bit peut être placé à 0 si il était le dernier fragment, et à 1 s'il n'y a pas de fragments supplémentaires de ce même paquet.

Fragment décalé – bits 50 – 63. Le champ fragment décalé indique de quel datagramme le paquet dépend. Les fragments sont calculés sur 64 bits, et le premier fragment a un décalage zéro.

Durée de vie – bits 64 – 72. Le champ TTL (Time To Live) nous indique le durée de vie d'un paquet, ou combien de "sauts" (hops) il peut réaliser sur l'Internet. Chaque processus qui touche le paquet doit supprimer un point du champ TTL, et si le TTL atteint zéro, le paquet doit être détruit ou écarté. C'est un usage de base fonctionnant comme une sécurité car si le paquet n'est pas supprimé/écarté il peut se transformer en boucle incontrôlable entre un ou plusieurs hôtes. Pour la destruction l'hôte renverra un message ICMP "Destination Unreachable" à l'expéditeur.

Protocole – bits 73 – 80. Dans ce champ le protocole du niveau de la couche suivante est indiqué. Par exemple, ce peut être TCP, UDP ou ICMP parmi d'autres. Tous ces nombres sont définis par la Internet Assigned Numbers Authority (IANA). Ces nombres peuvent être retrouvés sur le site Internet Assigned Numbers Authority.

Somme de contrôle d'en−tête – bits 81 – 96. C'est un contrôle de l'en−tête IP du paquet. Ce champ est recalculé au niveau de chaque hôte qui modifie l'en−tête, ce qui signifie à peu près chaque hôte que le paquet traverse, la modification se faisant le plus souvent au niveau du champ TTL.

Adresse de l'expéditeur – bits 97 – 128. C'est le champ adresse de la source. Elle est généralement écrite sur 4

octets, traduite du binaire en nombres décimaux avec des points entre eux. Par exemple, 127.0.0.1. Le champ permet au destinataire de connaître l'adresse d'expédition du paquet.

Adresse de destination – bits 129 – 160. Le champ adresse contient l'adresse de destination, et surprise, il est formaté de la même façon que l'adresse de l'expéditeur.

Options – bits 161 – 192 <> 478. Le champ options n'est pas optionnel, comme il pourrait le sembler. C'est un des champs les plus complexes dans les en–têtes IP. Le champ options contient divers réglages optionnels à placer dans l'en–tête, comme minuterie Internet, SACK ou enregistrement de route. Ces options sont toutes optionnelles, le champ options peut avoir des longueurs différentes, et donc l'en–tête IP complet. Cependant, nous calculons toujours l'en–tête IP sur 32 bits, nous devons toujours clore l'en–tête sur un même nombre, qui est un multiple de 32. Le champ peut contenir zéro ou plusieurs options.

Le champ options démarre avec un champ de 8 bits qui nous permet de savoir quelles options ont été utilisées dans le paquet. Les options sont toutes listées dans la table Options TCP, dans l'appendice Options TCP. Pour plus d'information sur les différentes options, lisez les RFC correspondantes. Pour une liste des options IP mises à jour, voir Internet Assigned Numbers Authority.

Remplissage – bits variables. C'est un champ de remplissage utilisé pour la fin de l'en–tête à la frontière des 32 bits. Le champ doit toujours être une ligne de zéros jusqu'à la fin.

2.4. Caractéristiques TCP

le protocole TCP réside au sommet du protocole IP. C'est un protocole architecture qui possède des fonctions natives pour vérifier si les données sont reçues correctement par les hôtes destinataires. Les buts principaux du protocole TCP sont de vérifier que les données sont envoyées et reçues de façon fiable, que les données sont transportées entre la couche Internet et la couche Application correctement, et que les paquets joignent le bon programme dans la couche application, et dans le bon ordre. Tout ceci est possible grâce aux en–têtes TCP du paquet.

Le protocole TCP considère les données comme un flux continu avec un signal de début et de fin. Le signal qui indique qu'un nouveau flux est en attente d'ouverture est appelé une poignée de main SYN à trois voies dans TCP, et consiste en un paquet envoyé avec le bit SYN. Les réponses se font soit avec SYN/ACK soit avec SYN/RST pour permettre au client de savoir si la connexion a été acceptée ou refusée respectivement. Si le client reçoit un paquet SYN/ACK, il peut de nouveau répondre, cette fois avec un paquet ACK. À ce moment, la connexion est établie et les données peuvent être envoyées. Pendant la poignée de main (handshake) initiale, toutes les options spécifiques qui seront utilisées à travers la connexion TCP sont aussi négociées, comme ECN, SACK, etc.

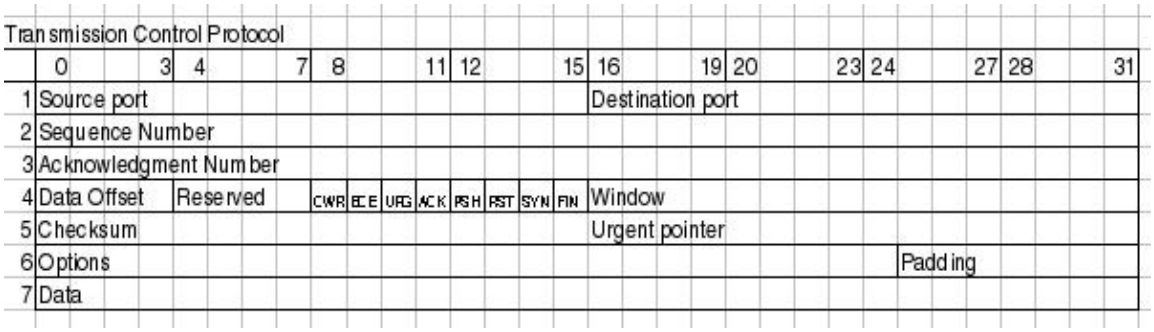
Tandis que le flux de données est actif, nous avons d'autres mécanismes à voir. C'est la partie fiabilité de TCP. Ceci est réalisé de façon simple, en utilisant un numéro d'interclassement (sequence) dans le paquet. Chaque fois que nous envoyons un paquet, nous donnons une nouvelle valeur au numéro d'interclassement, et quand le destinataire reçoit le paquet, il envoie en retour un paquet ACK à l'expéditeur. Le paquet ACK accuse réception que le paquet a été reçu correctement. Le numéro d'interclassement vérifie aussi que le paquet inséré dans un flux de données est en bon ordre.

Ensuite la connexion est fermée, ce qui est fait en envoyant un paquet FIN comme point final. Le destinataire répond alors en envoyant un paquet FIN/ACK. Plus aucune donnée ne peut alors être envoyée, mais l'expéditeur ou le destinataire peuvent toujours finir d'envoyer leurs données. Une fois que l'expéditeur ou le destinataire désire clore la connexion complètement, il envoie un paquet FIN en retour, et le correspondant répond avec un paquet FIN/ACK. Une fois cette procédure complète effectuée, la connexion est coupée proprement.

Comme nous le verrons plus tard, les en−têtes TCP contiennent des sommes de contrôle. La somme de contrôle consiste en une simple empreinte numérique du paquet. Avec cette empreinte numérique, nous pouvons avec une grande exactitude voir si la paquet a été corrompu d'une façon ou d'une autre pendant le transit entre les hôtes.

2.5. En−têtes TCP

Les en−têtes TCP doivent être capables d'exécuter toutes les tâches ci−dessus. Nous avons déjà expliqué quand et où certains de ces en−têtes sont utilisés, mais il y a encore d'autres zones que nous n'avons pas vu en détail. Ci−dessous vous avez une image de l'ensemble des en−têtes TCP. Il est formaté en mots de 32 bits par ligne, comme vous pouvez le voir.



Port expéditeur − bit 0 − 15. C'est le port source du paquet. Le port source était à l'origine lié directement au processus du système d'expédition. Aujourd'hui, nous utilisons une empreinte numérique entre les adresses IP, et les ports source et destination pour pouvoir les lier en une seule application ou programme.

Port destinataire − bit 16 − 31. C'est le port de destination du paquet TCP. Avec le port source, il était lié à l'origine directement au processus du système de réception. Aujourd'hui, une empreinte numérique est utilisée, qui nous permet d'avoir d'avantage de connexions ouvertes en même temps. Quand un paquet est reçu, les ports source et destination sont renversés dans la réponse à l'hôte d'origine, ainsi le port de destination est maintenant le port source, et le port source devient le port de destination.

Numéro d'interclassement − bit 32 − 63. le champ numéro d'interclassement est utilisé pour mettre en place un numéro sur chaque paquet TCP de façon que le flux TCP puisse être proprement ordonnancé (i.e. les paquets s'ordonnent dans le bon ordre). Le numéro d'interclassement est alors renvoyé dans le champ ACK pour accuser réception que le paquet a été correctement reçu.

Numéro d'accusé−reception − bit 64 − 95. Ce champ est utilisé quand nous accusons réception d'un paquet spécifique qu'un hôte a reçu. Par exemple, nous recevons un paquet avec un numéro d'interclassement, et si tout est ok avec le paquet, nous répondons avec un paquet ACK et le numéro d'interclassement identique au numéro d'interclassement d'origine.

Décalage de données − bit 96 − 99. Ce champ indique la longueur de l'en−tête TCP, et où la partie données du paquet démarre. Il est codé sur 4 bits, et mesure l'en−tête TCP en 32 bits. L'en−tête se termine toujours sur une limite de 32 bits, même avec différentes options.

Réservé − bit 100 − 103. Ces bits sont réservés pour un usage futur. Dans la RFC 793 les bits CWR et ECE sont également inclus. Selon la RFC 793 les bits 100−105 (i.e. ceci et les champs CWR et ECE) doivent être placés à zéro pour être pleinement compatibles. Plus tard, nous verrons cela quand nous commencerons l'introduction de ECN. Ceci a causé nombre de désagréments à cause de machines Internet comme des pare−feux et routeurs qui suppriment des paquets. C'est toujours vrai lors de la rédaction de ces lignes.

CWR − bit 104. ce bit a été rajouté dans la RFC 3268 et est utilisé par ECN. CWR pour Congestion Window Reduced, est utilisé par la partie des données envoyées au destinataire pour l'informer que la fenêtre d'encombrement a été réduite. Quand la fenêtre d'encombrement est réduite, nous envoyons moins de données

par unité de temps, pour pouvoir assurer la charge totale du réseau.

ECE – bit 105. Ce bit a aussi été rajouté avec la RFC 3268 et il est utilisé par ECN. ECE pour ECN Echo. Il est utilisé par la pile TCP/IP sur l'hôte destinataire qui permet à l'expéditeur de savoir si un paquet CE a été reçu. La même chose s'applique ici, comme pour le bit CWR, qui était à l'origine une partie du champ réservé et à cause de ça, certains matériels réseau supprimaient les paquets si les champs contenaient autre chiose que des zéros. C'est actuellement toujours vrai pour beaucoup de matériels malheureusement.

URG – bit 106. Ce bit nous indique si nous utilisons le champ Urgent Pointer ou non. S'il est placé à 0, pas d'utilisation de Urgent Pointer, s'il est placé à 1, utilisation de Urgent Pointer.

ACK – bit 107. Ce bit est placé dans un paquet pour indiquer qu'il s'agit d'une réponse à un autre paquet que nous avons reçu, et qui contient des données. Un paquet accusé–réception est toujours envoyé pour indiquer que nous avons reçu le paquet, et qu'il ne contient pas d'erreurs. Si ce bit est placé, l'expéditeur des données d'origine vérifiera le numéro d'accusé–réception pour voir quel paquet est actuellement en accusé–réception, et ensuite videra les tampons.

PSH – bit 108. Le fanion PUSH est utilisé pour prévenir le protocole TCP sur des hôtes intermédiaires d'envoyer les données à l'utilisateur actuel, incluant l'implémentation TCP sur l'hôte destinataire. Ceci expédie toutes les données.

RST – bit 109. Le fanion RESET est placé pour indiquer à l'hôte de relancer la connexion TCP. Ceci est dû à divers scénarios, les principales raisons étant que la connexion a été coupée, la connexion n'existe pas, ou le paquet contient des erreurs.

SYN – bit 110. Le SYN (Synchronize Sequence Numbers) est utilisé pendant l'établissement initial de la connexion. Il est placé dans deux circonstances, le paquet initial qui ouvre la connexion, et le paquet SYN/ACK en réponse. Il ne doit jamais être utilisé en dehors de ces cas.

FIN – bit 111. Le bit FIN indique que l'hôte qui envoie le bit FIN n'a plus de données à expédier. Quand l'hôte voit le bit FIN, il répond avec un un FIN/ACK. Une fois cela fait, l'expéditeur du bit FIN ne peut plus envoyer de données. Cependant, l'hôte peut continuer à expédier les données jusqu'à ce qu'il ait fini, et ensuite enverra un paquet FIN en retour, et attendra le FIN/ACK final, après ça la connexion est en état CLOSED.

Window – bit 112 – 127. Le champ fenêtre est utilisé par l'hôte destinataire pour dire à l'expéditeur combien de données il autorise à cet instant. Ceci est fait en envoyant un ACK en retour, qui contient un numéro d'interclassement nécessaire pour l'accusé–réception, le champ fenêtre contient alors les numéros d'interclassement maximum acceptés que l'expéditeur peut utiliser avant de recevoir le prochain paquet ACK. Le paquet ACK suivant met à jour la fenêtre que l'expéditeur peut utiliser.

Checksum – bit 128 – 143. Ce champ contient la somme de contrôle de l'en–tête TCP complet. C'est un complément d'une somme de chaque mot de 16 bits dans l'en–tête. Si l'en–tête ne finit pas sur une limite de 16 bits, le bit additionnel est placé à zéro. Tandis que la somme de contrôle est calculée, le champ somme de contrôle est placé à zéro. La somme de contrôle couvre également une pseudo en–tête de 96 bits contenant la Destination, Adresse source, protocole, et longueur TCP. Ceci pour des raisons de sécurité.

Pointeur urgent – bit 144 – 159. Pointeur placé à la fin des données considérées comme urgentes. Si la connexion a d'importantes données qui doivent être exécutées le plus tôt possible par le destinataire, l'expéditeur peut placer un drapeau URG pour indiquer où les données urgentes finissent.

Options – bit 160 – **. le champ Options est un champ de longueur variable qui contient des en–têtes optionnels. De façon basique, ce champ contient 3 sous–champs chaque fois. Un champ initial nous indique la longueur du champ Options, un second indique quelles options sont utilisées, et quand nous obtenons les options. Une liste complète de toutes les options TCP se trouve dans Options TCP.

Padding – bit **. Le champ remplissage complète l'en–tête TCP jusqu'à ce que tout l'en–tête atteigne la limite de 32 bits. Ceci assure que la partie données du paquet débute sur une limite de 32 bits, et qu'aucune donnée n'est perdue dans le paquet. Le remplissage ne contient toujours que des zéros.

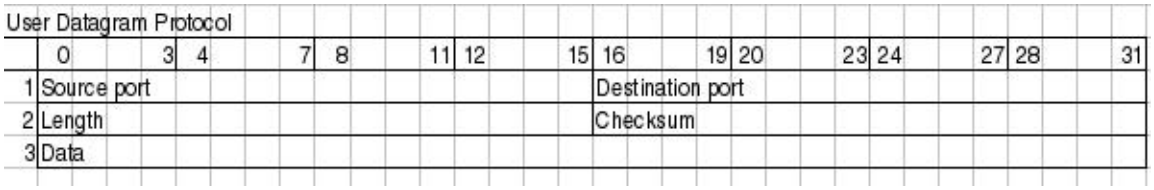
2.6. Caractéristiques UDP

Le User Datagram protocol (UDP protocol) est un protocole très basique et simple au sommet du protocole IP. Il a été développé pour permettre une transmission de données très simple sans détection d'erreur d'aucune sorte. Cependant, il est très bien adapté pour les applications de type requête/réponse, comme par exemple DNS, etc. car nous savons qu'à moins d'obtenir une réponse du serveur DNS, la requête sera perdue quelque part. Parfois il peut être utile de se servir du protocole UDP au lieu de TCP, comme lorsque nous voulons seulement une détection d'erreurs/pertes mais sans faire attention à l'ordre d'interclassement des paquets. Ceci supprime quelques en–têtes présents dans le protocole TCP. Nous pouvons aussi faire d'autres choses, créer notre propre protocole au sommet de UDP qui ne contient que l'interclassement, mais pas d'erreur ni de perte.

Le protocole UDP est spécifié dans la RFC 768 – User Datagram Protocol. C'est une très brève RFC.

2.7. En–têtes UDP

On peut dire que l'en–tête UDP est un en–tête TCP très basique et simplifié. Il contient la destination, le port source, la longueur d'en–tête et une somme de contrôle comme indiqué dans l'image ci–dessous.



Port source – bit 0–15. C'est le port source du paquet, décrivant où un paquet en réponse sera envoyé. Par exemple, quelquefois nous n'exigeons pas de paquet en réponse, le paquet peut alors être placé à zéro en port source. Dans la plupart des implémentations, il est placé à un nombre quelconque.

Port destination – bit 16–31. Le port de destination du paquet. Nécessaire pour tous les paquets, à l'opposé du port source d'un paquet.

Length – bit 32–47. Le champ longueur spécifie la taille de l'ensemble du paquet en octets, incluant les en–têtes et les données. Le plus petit paquet possible est de 8 octets.

Checksum – bit 48–63. La somme de contrôle est du même type que celle utilisée dans les en–têtes TCP, sauf qu'elle contient un ensemble de données différent. En d'autres termes, c'est un complément d'un complément de parties de l'en–tête IP, de l'ensemble de l'en–tête UDP, les données UDP et le remplissage avec des zéros lorsque nécessaire.

2.8. Caractéristiques ICMP

Les messages ICMP sont utilisés pour les types d'erreurs basiques rapportées entre hôtes, ou entre hôte et passerelle. Entre passerelles, un protocole appelé GGP (Gateway to gateway protocol) ou passerelle à passerelle doit normalement être utilisé pour le rapport d'erreur. Comme nous l'avons déjà vu, le protocole IP n'a pas été conçu pour parfaire le maniement d'erreur, mais les messages ICMP résolvent une partie de ces problèmes. Le gros problème de ce point de vue est que les en–têtes des messages ICMP sont plutôt compliqués, et diffèrent de message en message. Cependant, ce n'est pas un gros problème du point de vue filtrage la plupart du temps.

La forme de base du message contient l'en–tête IP standard, le type, le code et la somme de contrôle. Tous les messages ICMP contiennent ces champs. Le type spécifie le genre d'erreur du message de réponse, comme par exemple, destination injoignable, écho, réponse d'écho, ou message redirigé. Le code donne plus d'information si nécessaire. Si le paquet est de type destination injoignable, il y a plusieurs valeurs possibles comme réseau injoignable, hôte injoignable, ou port injoignable. La somme de contrôle est simplement une somme pour l'ensemble du paquet.

Comme vous avez pû le noter, j'ai mentionné explicitement l'en–tête IP pour la paquet ICMP. C'est dû au fait que l'en–tête IP actuel est une partie intégrale du paquet ICMP, et que le protocole ICMP est au même niveau, dans un sens, que le protocole IP. ICMP utilise le protocole IP comme s'il était à un niveau plus haut, mais en même temps ce n'est pas le cas. ICMP est une partie intégrale de IP, et ICMP doit être implémenté dans chaque implémentation de IP.

2.9. En–têtes ICMP

Comme déjà expliqué, les en–têtes de type ICMP diffèrent légèrement du type IP. La plupart des types ICMP permettent de les grouper par leurs en–têtes. À cause de cela, nous verrons en premier l'en–tête de base, et ensuite chaque groupe de type spécifique.

ICMP Basic Headers

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Version	IHL	Type of Service/DSCP/E		Total Length			
2	Identification				Flags	Fragment Offset		
3	Time to Live		Protocol		Header checksum			
4	Source adress							
5	Destination adress							
6	Type		Code		Checksum			

Tous les paquets contiennent des valeurs de base des en–têtes IP comme nous l'avons vu précédemment.

- Version – Doit toujours être placé à 4.
- Longueur en–tête Internet – Longueur de l'en–tête en mots de 32 bits.
- Type de Service – Voir au–dessus. Doit être placé à 0, seule légitimité selon la RFC 792 – Internet Control Message Protocol.
- Longueur totale – Longueur totale de l'en–tête et de la partie données du paquet, compté en octets.
- Décalages d'Identification, de fanions et fragments – Issu du protocole IP.
- Durée de vie – Nombre de sauts que le paquet peut effectuer.
- Protocole – Quelle version de ICMP est utilisée (doit toujours être à 1).
- En–tête somme de contrôle – Voir l'explication IP.
- Addresse source – L'adresse de la source qui a envoyé le paquet. Ce n'est pas entièrement vrai, car le paquet peut avoir une autre adresse source, que celle située sur la machine en question. Les types ICMP peuvent produire cet effet, ce sera noté si nécessaire.
- Addresse destination – L'adresse de destination du paquet.

Il existe aussi de nouvelles en–têtes utilisés par les types ICMP. Ce sont les suivantes :

- Type – Le champ type contient le type ICMP du paquet. Par exemple les paquets ICMP Destination Injoignable auront un type 3 placé. Pour une liste complète des différents types ICMP voir Types ICMP. Ce champ contient 8 bits au total.
- Code – Tous les types ICMP contiennent différents codes. Certains types ont un code unique, tandis que d'autres ont plusieurs codes qu'ils peuvent utiliser. Par exemple, le type ICMP Destination Injoignable peut avoir au moins les codes 0, 1, 2, 3, 4 ou 5. Chaque code aura un comportement différent selon le contexte. Pour une liste complète des différents codes, voir Types ICMP. Ce champ est de 8 bits de longueur totale. Nous verrons les différents codes un peu plus en détail plus tard dans cette section.

- Somme de contrôle – La somme de contrôle est un champ de 16 bits contenant un complément de complément des en–têtes démarrant avec le type ICMP. Tandis que le calcul de la somme de contrôle s'effectue, le champ de celle–ci sera placé à zéro.

À ce point les en–têtes peuvent présenter un visage différent. Nous décrirons les types ICMP les plus communs un par un, avec un bref apperçu de leurs différentes en–têtes et codes.

2.9.1. Écho requête/réponse ICMP

Internet Control Message Protocol – Echo/Echo Reply Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Identifier				Sequence Number			
3	Data...							

J'ai choisi de parler des paquets ICMP écho requête et réponse, ils sont très proches l'un par rapport à l'autre. La différence est que l'écho requête est de type 8, alors que l'écho réponse est de type 0. Quand un hôte reçoit un type 8, il répond avec un type 0.

Quand la réponse est envoyée, les adresses source et destination sont permutées. Après les deux changements effectués, la somme de contrôle est recalculée, et la réponse envoyée. Il y a un seul code pour les deux types, ils sont toujours placés à 0.

- Identifiant – Il est placé dans le paquet requête, et se retrouve en retour dans la réponse, il permet de synchroniser les différents pings de requête et de réponse.
- Checksum – Le numéro d'interclassement pour chaque hôte démarre généralement à 1 et est incrémenté de 1 pour chaque paquet.

Les paquets contiennent aussi une partie de données. Par défaut, la partie de données est généralement vide, mais elle peut contenir des données au hasard spécifiées par l'utilisateur.

2.9.2. Destination Injoignable ICMP

Internet Control Message Protocol – Destination Unreachable Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Unused							
3	Internet header + 64 bits of original data datagram							

Les trois premiers champs montrés dans l'image sont les mêmes que ceux précédemment décrits. Le type Destination Injoignable possède six codes de base qui peuvent être utilisés, comme indiqué ci–dessous.

- Code 0 – Réseau injoignable – Vous indique si un réseau spécifique est actuellement injoignable.
- Code 1 – Hôte injoignable – Un hôte spécifique est actuellement injoignable.
- Code 2 – Protocole injoignable – Ce code vous indique si un protocole spécifique (TCP, UDP, etc.) ne peut être joint pour l'instant.
- Code 3 – Port injoignable – Si un port (ssh, http, ftp, etc.) n'est pas joignable vous obtenez ce message.
- Code 4 – Fragmentation nécessaire et placement de DF – Si le paquet nécessite d'être fragmenté pour être délivré, mais que le bit "Do not Fragment" est placé dans le paquet, la passerelle retourne ce message.
- Code 5 – Échec de la route source – Si la route source échoue pour quelque raison, ce message est retourné.
- Code 6 – Destination réseau inconnue – S'il n'y a pas de route vers un réseau spécifique, ce message est retourné.

- Code 7 – Hôte de destination inconnu – S'il n'y a pas de route vers l'hôte spécifique, ce message est retourné.
- Code 8 – Hôte source isolé (obsolète) – Si l'hôte est isolé, ce message sera retourné. Ce code est obsolète aujourd'hui.
- Code 9 – Réseau de destination administrativement interdit – Si un réseau est bloqué au niveau de la passerelle et que votre paquet est incapable de le joindre à cause de ça, vous obtiendrez ce code ICMP en retour.
- Code 10 – Hôte de destination administrativement interdit – Si vous ne pouvez joindre l'hôte parce qu'il a été interdit administrativement (ex. administration du routage), vous obtenez ce message.
- Code 11 – Réseau injoignable pour TOS (Type de Service) – Si un réseau est injoignable à cause d'un mauvais TOS placé dans votre paquet, ce code sera généré en retour.
- Code 12 – Hôte injoignable pour TOS – Si votre paquet est incapable de joindre l'hôte à cause du TOS du paquet, ce message sera renvoyé.
- Code 13 – Communication administrativement interdite par filtrage – Si le paquet est interdit pour une raison (ex. pare–feu) de filtrage, vous obtenez le code 13 en retour.
- Code 14 – Violation de loi de précédence – Envoyé par le premier routeur pour notifier à un hôte connecté que la précédence utilisée n'est pas autorisée pour la combinaison spécifique source/destination.
- Code 15 – Effet de coupure de précédence – Le premier routeur peut envoyer ce message à un hôte si le datagramme reçu a un niveau de précédence trop bas.

Au sommet de tout ça, il existe également une petite partie "données", qui devrait être l'en–tête Internet et le datagramme IP d'origine en 64 bits. Si le protocole de niveau suivant contient des ports, etc. il est supposé que les ports seront disponibles dans les 64 bits supplémentaires.

2.9.3. Coupure de source

Internet Control Message Protocol – Source Quench Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Unused							
3	Internet header + 64 bits of original data datagram							

Un paquet coupure de source peut être envoyé à l'expéditeur d'un paquet ou d'un flux de paquets trop lents pour permettre de continuer à envoyer des données. Notez que la passerelle ou l'hôte que les paquets traversent peuvent aussi décharger les paquets sans prévenir, au lieu d'envoyer des paquets coupure de source.

Ces paquets ne contiennent pas d'en–têtes supplémentaires sauf la partie données, laquelle contient l'en–tête Internet plus les 64 bits du datagramme de données d'origine. Ceci est utilisé pour harmoniser le message de coupure source au processus correct, lequel envoie des données à travers la passerelle ou vers l'hôte de destination.

Tous les paquets coupure source ont leurs type ICMP placé à 4. Ils n'ont pas de codes sauf le 0.

Note

Aujourd'hui, il existe de nouveaux moyens de notifier à l'expéditeur ou au destinataire qu'une passerelle ou un hôte est en surcharge. Par exemple avec le système ECN (Explicit Congestion Notification).

2.9.4. Redirection

Internet Control Message Protocol – Redirect Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Gateway Internet adress							
3	Internet header + 64 bits of original data datagram							

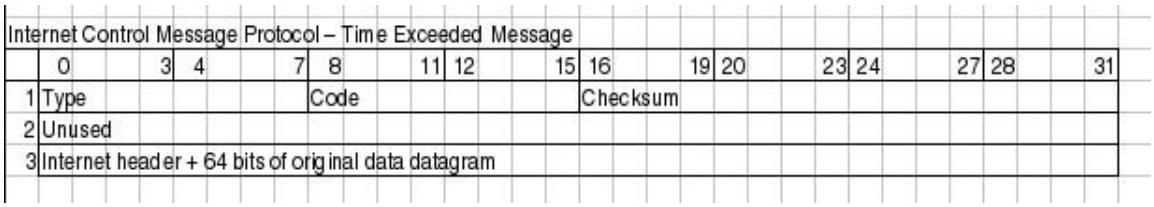
Le type redirection est envoyé dans un seul cas. Considerez ceci, vous avez un réseau (192.168.0.0/24) avec plusieurs clients et hôtes, et deux passerelles. Une passerelle sur un réseau 10.0.0.0/24, et une passerelle par défaut pour le reste de l'Internet. Maintenant considerez qu'un des hôtes soit sur le réseau 192.168.0.0./24 et n'ait pas de route vers 10.0.0.0/24, mais ait accès à la passerelle par défaut. Il envoie un paquet à la passerelle par défaut, laquelle bien sûr connait le réseau 10.0.0.0/24. Cette passerelle par défaut peut déduire qu'il est plus facile d'envoyer le paquet directement à la passerelle 10.0.0.0/24 car ce paquet entrera et quittera la passerelle par la même interface. La passerelle par défaut enverra désormais un paquet ICMP Redirect unique à l'hôte, à travers la passerelle 10.0.0.0/24. L'hôte saura maintenant que la passerelle la plus proche est 10.0.0.0/24, et l'utilisera dans le futur.

L'en−tête principale du type Redirect est le champ Gateway Internet Address. Ce champ indique à l'hôte la passerelle correcte, qui sera réellement utilisée. Le paquet contient aussi l'en−tête IP du paquet original, et les premiers 64 bits de données dans le paquet d'origine, lequel est utilisé pour se connecter au processus qui envoie les données.

Le type Redirection possède 4 codes, qui sont les suivants.

- Code 0 − Redirection pour le réseau − Seulement utilisé pour rediriger l'ensemble du réseau (voir l'exemple ci−dessus).
- Code 1 − Redirection pour l'hôte − Seulement utilisé pour les redirections d'un hôte spécifique.
- Code 2 − Redirection pour TOS et réseau − Seulement utilisé pour rediriger un Type de Service spécifique et vers un ensemble réseau. Utilisé comme le code 0, mais aussi basé sur TOS.
- Code 3 − Redirection pour TOS et hôte − Seulement utilisé pour rediriger vers un Type de Service spécifique vers un hôte spécifique. utilisé comme le code 1, mais aussi basé sur le TOS.

2.9.5. TTL égale 0



Le type ICMP TTL égale 0 est également connu comme "Time Exceeded Message" et possède le type 11, il a aussi 2 codes ICMP disponibles. Si le champ TTL atteint 0 pendant le transit à travers une passerelle ou un fragment réassemblé sur l'hôte de destination, le paquet doit être supprimé. Pour notifier ce problème à l'hôte expéditeur, nous pouvons envoyer un paquet ICMP TTL égale 0. L'expéditeur peut augmenter le TTL des paquets sortants si nécessaire.

Le paquet contient seulement la partie supplémentaire des données. Le champ données contient l'en−tête Internet plus 64 bits de données du paquet IP. Comme précédemment mentionné, le type TTL égale 0 peut avoir deux codes.

- Code 0 − TTL égale 0 pendant le transit − Envoyé par l'expéditeur si le paquet TTL d'origine atteint 0 quand il est transféré par une passerelle.
- Code 1 − TTL égale 0 pendant le ré−assemblage − Envoyé si le paquet d'origine était fragmenté, et que le TTL atteint 0 pendant le ré−assemblage des fragments. Ce code doit être envoyé uniquement depuis le destinataire.

2.9.6. Paramètre problème

Internet Control Message Protocol – Parameter Problem Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Pointer		Unused					
3	Internet header + 64 bits of original data datagram							

Le paramètre problème ICMP utilise le type 12 et possède deux codes qu'il peut utiliser indifféremment. Les messages du paramètre problème sont utilisés pour indiquer à l'hôte que la passerelle ou le destinataire ont des problèmes de compréhension sur des parties d'en−tête IP, ou nécessitent des options qui ont été omises.

Le type paramètre problème contient une en−tête spéciale, qui est un pointeur vers le champ qui a causé l'erreur dans le paquet d'origine, si le code est à 0. Les codes suivants sont disponibles:

- Code 0 − Mauvaise en−tête IP (catchall error) − Nous avons vu ce message d'erreur ci−dessus. Avec le pointeur ce code est utilisé pour indiquer quelle partie de l'en−tête IP contient l'erreur.
- Code 1 − Options nécessaires omises − Si une option IP nécessaire est omise, ce code est utilisé pour l'indiquer.

2.9.7. Horodatage requête/réponse

Internet Control Message Protocol – Timestamp/Timestamp Reply Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Identifier				Sequence Number			
3	Originate Timestamp							
4	Receive Timestamp							
5	Transmit Timestamp							

Le type horodatage est aujoud'hui obsolète, mais nous le verrons brièvement. La réponse et la requête ont un code unique (0). La requête est de type 13 tandis que la réponse est de type 14. Les paquets horodatage contiennent 3 fois 32 bits comptant les millisecondes depuis minuit en temps universel (UT).

Le premier horodatage est l'horodatage d'origine, qui contient la dernière information sur l'expéditeur du paquet. L'horodatage de réception est l'information sur le premier hôte qui a reçu le paquet et l'horodatage de transmission le dernier horodatage sur l'envoi du paquet.

Chaque message d'horodatage contient aussi les mêmes identifiants et numéros d'ordre que les paquets ICMP écho.

2.9.8. Requête/réponse information

Internet Control Message Protocol – Information Request/Information Reply Message

	0 3	4 7	8 11	12 15	16 19	20 23	24 27	28 31
1	Type		Code		Checksum			
2	Identifier				Sequence Number			

Les types requête/réponse information sont obsolètes depuis que les protocoles au sommet du protocole IP peuvent maintenant jouer ce rôle lorsque nécessaire (DHCP, etc.). La requête information génère une réponse depuis n'importe quel hôte interrogé sur le réseau.

L'hôte qui désire recevoir l'information crée un paquet avec l'adresse source du réseau dans lequel il est lié (exemple, 192.168.0.0), et le réseau destinataire est placé à 0. La réponse contiendra l'information au sujet du masque de réseau et de l'adresse IP.

La requête information est lancée à travers un type ICMP 15 alors que la réponse est envoyée via un type 16.

2.10. Destination TCP/IP par routage

TCP/IP s'est accru en complexité quand il est devenu une partie du routage. Au début, la plupart des gens pensaient que la destination donnée par le routage était suffisante. Ces dernières années, c'est devenu de plus en plus complexe. Aujourd'hui, Linux peut router de façon basique chaque champ ou bit dans l'en−tête IP, et également les en−têtes basées sur TCP, UDP ou ICMP. Ceci est appelé gestion de réseau à base de règles, ou routage avancé.

Il ne s'agit ici que d'un survol du routage. Quand nous envoyons un paquet depuis un expéditeur, le paquet est créé. Après ça, l'ordinateur regarde l'adresse de destination du paquet et la compare à sa table de routage. Si l'adresse de destination est locale, le paquet est envoyé directement via l'adresse MAC du matériel (Ndt. interface réseau). Si le paquet est de l'autre côté de la passerelle il est envoyé via l'adresse MAC de la passerelle. Celle−ci regardera alors les en−têtes IP et verra l'adresse de destination du paquet. L'adresse de destination est consultée dans la table de routage, et le paquet envoyé à la passerelle suivante, etc. jusqu'à sa destination finale.

Comme vous pouvez le voir, ce routage est très basique. Avec le routage avancé, et la gestion de réseau à base de règles, ceci devient plus complexe. Nous pouvons router des paquets qui diffèrent dans leur adresse source par exemple, ou leur valeur TOS, etc.

2.11. Prochaine étape

Nous avons vu les points suivants :

- Structure TCP/IP
- Fonctionnalité du protocole IP et en−têtes.
- Fonctionnalité du protocole TCP et en−têtes.
- Fonctionnalité du protocole UDP et en−têtes.
- Fonctionnalité du protocole ICMP et en−têtes.
- Destination TCP/IP par routage.

Tout ceci sera revu plus tard quand nous aborderons les tables de règles des pare−feux. Toute ces informations s'imbriquent ensemble, pour permettre une meilleure configuration de pare−feu.

Chapitre 3. Introduction au filtrage IP

Dans ce chapitre nous verrons en détail la théorie du filtrage IP, ce que c'est, comment ça fonctionne et certaines choses basiques comme où placer les pare−feux, les règles de filtrage, etc.

Les questions de ce chapitre pourront être, où placer un pare−feu ? Dans la plupart des cas, c'est une question simple, mais dans des réseaux étendus cela peut devenir ardu. Quelles seraient les règles ? Qui aurait accès et où ? Qu'est−ce qu'un filtre IP ? Nous y répondrons ici.

3.1. Qu'est−ce qu'un filtre IP ?

Il est important de bien comprendre ce qu'est un filtre IP. Iptables est un filtre IP, et si vous ne comprenez pas complètement cela, vous irez au devant de sérieux problèmes dans la conception de vos pare−feux.

Un filtre IP opère principalement au niveau de la couche 2 de la pile de référence TCP/IP. Iptables cependant peut également travailler au niveau de la couche 3. Mais par définition un filtre IP travaille sur la seconde couche.

Si l'implémentation du filtre IP suit strictement la définition, il devrait être capable, en d'autres termes, de filtrer les paquets basés sur leurs en−têtes IP (adresses source et destination, TOS/DSCP/ECN, TTL, protocole, etc.). Toutes choses actuellement dans l'en−tête IP. Cependant, l'implémentation de Iptables n'est pas strictement en accord avec la définition, il est aussi capable de filtrer les paquets basés sur d'autres en−têtes se trouvant dans le paquet (TCP, UDP, etc.), et l'adresse MAC.

Il y a une chose cependant sur laquelle Iptables est très strict aujourd'hui. Il ne "suit" pas les flux ou les morceaux de données ensemble. Ce serait une grosse perte de temps. Il garde la trace des paquets et regarde si ils font partie du même flux de données (via numéros d'interclassement, numéros de port, etc.) à peu près comme la vraie pile TCP/IP. Ceci est appelé traçage de connexion, et grâce à ça nous pouvons effectuer de la translation (masquage/traduction) d'adresse source et destination (généralement appelé DNAT et SNAT), aussi bien que de la vérification d'état de paquets.

Comme nous avons vu ci−dessus, Iptables ne peut pas connecter des données provenant de différents paquets entre eux, et donc vous ne pourrez jamais être tout à fait certains que vous verrez la totalité des données à tout moment. Je mentione ça car il y a constamment des questions sur les différentes listes de discussion concernant netfilter et iptables à ce sujet. Par exemple, chaque fois qu'il survient un nouveau virus basé sur Windows, diverses personnes me demandent comment supprimer tous les flux contenant une chaîne spécifique. Par exemple, si nous regardons quelque chose comme ça :

cmd.exe

Que se passe−t−il si l'auteur du virus/exploit est suffisamment malin pour rendre la taille du paquet si petite que ***cmd*** tienne dans un seul paquet et ***.exe*** dans le paquet suivant ? Les fonctions d'appariement de chaîne sont incapables de passer à travers les frontières de paquet, le paquet continuera sa route.

Certains pourront se poser la question, pourquoi ne pas faire simplement des vérifications de chaîne ? C'est actuellement très simple. Ce serait trop coûteux en temps processeur. Le traçage de connexion prend déjà beaucoup de temps processeur. Ajouter une autre couche complexe au traçage de connexion, surchargerait la plupart des pare−feux. Sans parler de la quantité de mémoire supplémentaire qui serait nécessaire pour effectuer cette tâche pour chaque machine.

Il existe une seconde raison pour que cette fonctionnalité ne soit pas développée. Il existe une technique appelée proxy (mandataire). Les proxies ont été développés pour le traffic dans les couches les plus hautes, et donc répondent au mieux à ces besoins. Les proxies ont été créés à l'origine pour gérer les téléchargements et les pages les plus souvent utilisées et accélérer les transferts avec des connexions Internet lentes. Par exemple, Squid est un proxy pour le web. Une personne qui désire charger une page envoie la requête, le proxy soit extrait la requête soit reçoit la requête et ouvre la connexion au navigateur, et ensuite le connecte au serveur web et charge le fichier, et une fois chargé le fichier ou la page, l'envoie au client. Maintenant, si un second navigateur désire lire la même page, le fichier ou la page sont déja chargés par le proxy, et peuvent être envoyés directement, vous économisant du temps (et de la bande passante).

Comme vous pouvez le comprendre, les proxies ont un ensemble de fonctionnalités leur permettant de voir le contenu des fichiers qu'ils chargent. À cause de ça, ils sont bien meilleurs pour visualiser l'ensemble des flux, fichiers, pages, etc.

3.2. Termes et expressions du filtrage IP

Pour pleinement comprendre les chapitres suivants il y a quelques termes et expressions généraux que vous devez connaître, incluant nombre de détails par rapport au chapitre TCP/IP. Voici une liste des termes les plus couramment utilisés dans le filtrage IP.

- Effacement/refus (Drop/Deny) − Quand un paquet est effacé ou refusé, il est tout simplement supprimé. Aucune réponse n'est faite s'il est effacé, de même l'hôte destinataire du paquet ne sera pas

prévenu. Le paquet disparaît simplement.

- Rejet (Reject) – De façon basique, c'est la même chose que effacer/refuser, sauf qu'une notification est envoyée à l'hôte expéditeur du paquet rejeté. La réponse peut être spécifiée ou automatiquement calculée pour une certaine valeur. Actuellement, il n'y a malheureusement pas de fonctionnalité dans Iptables qui permette de prévenir l'hôte destinataire que le paquet a été rejeté. Ce serait très bien dans certaines circonstances, car l'hôte destinataire n'a pas la possibilité d'arrêter une attaque par DoS (Denial of Service) quand ça se produit.
- État (State) – Un état spécifique d'un paquet par rapport à l'ensemble d'un flux de paquets. Par exemple, si le paquet est le premier que voit le pare–feu ou dont il a connaissance, il est considéré comme nouveau (le paquet SYN dans une connexion TCP), ou s'il fait partie d'une connexion déja établie dont a connaissance le pare–feu. Les états sont connus par le traçage de connexion, qui garde les traces de toutes les sessions.
- Chaîne – Une chaîne contient un ensemble de règles qui sont appliquées aux paquets qui traversent la chaîne. Chaque chaîne a un but spécifique (ex. quelle table est connectée et à qui, qui spécifie que cette chaîne est habilitée à le faire), et une zone d'application spécifique (ex. seulement les paquets transmis, ou seulement les paquets destinés à un hôte). Dans Iptables il existe plusieurs chaînes différentes, comme nous le verrons plus loin.
- Table – Chaque table possède une fonctionnalité spécifique, et Iptables possède trois tables. Les tables nat, mangle et filter. Par exemple, la table filter est destinée à filtrer les paquets, tandis que la table nat est destinée aux paquets NAT (Network Address Translation).
- Match – Ce terme peut avoir deux sens différents quand il est employé avec iptables. Le premier sens est une simple correspondance qui indique à la règle ce que l'en–tête doit contenir. Par exemple, la correspondance ––source nous indique que l'adresse source doit être une plage réseau spécifique ou une adresse hôte. Le second indique si la règle entière est une correspondance. Si le paquet apparie toute la règle, les instructions saut et cible seront exécutées (ex. le paquet sera supprimé).
- Cible (Target) – C'est généralement une cible placée pour chaque règle dans une table de règles. Si la règle est pleinement appariée, la spécification de cible nous indique que faire avec le paquet. Par exemple, si nous l'effaçons, l'acceptons, le traduisons (nat), etc. Il existe aussi une chose appelée spécification de saut, pour plus d'information voir la description de saut dans la liste. Enfin, il peut ne pas y avoir de cible ou de saut dans chaque règle.
- Règle (Rule) – Une règle est un ensemble de correspondances avec cible unique dans la plupart des implémentations de filtres IP, incluant l'implémentation d'Iptables. Il existe certaines implémentations qui nous permettent d'utiliser plusieurs cibles/actions par règle.
- Table de règles (Ruleset) – Une table de règles est un ensemble complet de règles placé dans une implémentation de filtre IP. Dans le cas d'Iptables, ceci inclut toutes les règles placées dans le filtre, nat et mangle, et toutes les chaînes qui suivent. La plupart du temps, elles sont écrites dans un fichier de configuration.
- Saut (Jump) – L'instruction jump est très proche d'une cible. Une instruction jump est écrite exactement de la même façon qu'une cible dans Iptables, sauf que au lieu d'écrire un nom de cible, vous écrivez un nom de chaîne. Si la règle apparie, le paquet sera désormais envoyé à la seconde chaîne et exécuté comme d'habitude dans cette chaîne.
- Traçage de connexion – Un pare–feu qui implémente le traçage de connexion est capable de suivre les connexions/flux. La possibilité de faire ceci a un impact sur le processeur et l'usage mémoire. C'est malheureusement vrai également dans Iptables, mais beaucoup de travail a été fait sur ce sujet. Cependant, le bon côté de la chose est que le pare–feu sera beaucoup plus sécurisé avec un traçage de connexion bien utilisé par celui qui en établira les règles.
- Acceptation (Accept) – Pour accepter un paquet et le laisser passer à travers les règles du pare–feu. C'est l'opposé des cibles effacement et refus, de même pour la cible rejet.
- Gestion des règles (Policy) – Il existe deux sortes de gestion des règles dans l'implémentation d'un pare–feu. En premier nous avons la gestion des chaînes, qui indiquent le comportement par défaut du pare–feu. C'est l'usage principal du terme que nous utiliserons dans ce didacticiel. Le second type de gestion des règles est la gestion de sécurité établie par une stratégie d'ensemble d'une entreprise par exemple, ou pour un segment de réseau spécifique. Les stratégies de sécurité sont des documents a étudier de près avant de commencer l'implémentation d'un pare–feu.

3.3. Comment configurer un filtre IP ?

Une des premières choses à considérer lors de la configuration d'un pare−feu est son emplacement. Un des premiers endroits qui vient à l'esprit est la passerelle entre votre réseau local et l'Internet. C'est un endroit qui devrait être très sécurisé. Ainsi, dans les grands réseaux ce peut−être une bonne idée de séparer différentes divisions entre elles par des pare−feux. Par exemple, pourquoi l'équipe de développement aurait accès au réseau des ressources humaines, ou pourquoi ne pas protéger le département comptabilité des autres ?

Ceci indique que vous devriez configurer vos réseaux aussi bien que possible, et les planifier pour qu'ils soient isolés. Spécialement si le réseau est de taille moyenne ou grande (100 stations de travail ou plus, basé sur différents aspects du réseau). Dans les petits réseaux, essayez de configurer les pare−feux pour seulement autoriser le genre de trafic que vous désirez.

Ce peut être aussi une bonne idée de créer une "zone démilitarisée" (DMZ) dans votre réseau dans le cas où vous avez des serveurs qui sont connectés à l'Internet. Une DMZ est un petit réseau physique avec des serveurs, qui est isolé à l'extrême. Ceci réduit le risque que quelqu'un puisse s'introduire dans les machines de la DMZ, et également puisse y implanter des chevaux de Troie ou autre depuis l'extérieur. La raison pour laquelle nous l'appelons zone démilitarisée est qu'elle doit être joignable depuis l'intérieur et l'extérieur, et donc être une sorte de zone grise.

Il existe deux moyens pour configurer le comportement d'un pare−feu, et dans cette section nous verrons ce que vous devriez considérer avant d'implémenter votre pare−feu.

Avant de commencer, vous devriez comprendre que la plupart des pare−feux ont des comportements par défaut. Par exemple, si aucune règle n'est spécifiée dans une chaîne, elle peut−être acceptée ou effacée par défaut. Malheureusement, il y a seulement une gestion de règles par chaîne, mais il est souvent facile de les contourner si vous voulez avoir différentes gestions des règles par interface réseau, etc.

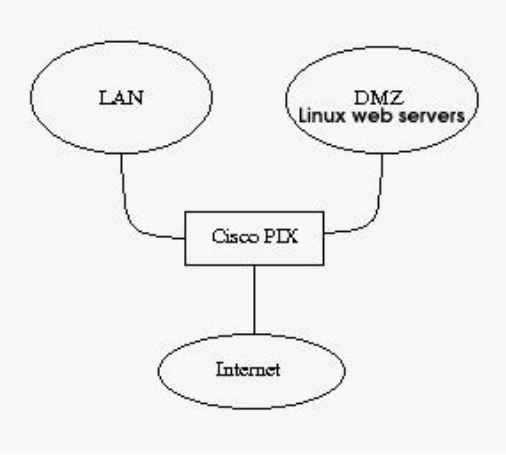
Il existe deux stratégies de base que nous utilisons habituellement. Soit nous supprimons (DROP) tout sauf ce que nous spécifions, soit nous acceptons tout excepté ce que nous spécifions comme devant être supprimé. La plupart du temps nous sommes principalement intéressés par la stratégie de suppression, et ensuite accepter ce que nous désirons de façon spécifique. Ceci indique que le pare−feu est plus sécurisé par défaut, mais il peut aussi indiquer que nous aurons plus de travail pour simplement obtenir un pare−feu qui fonctionne correctement.

Votre première décision à prendre est de simplement savoir quel type de pare−feu vous allez utiliser. Quel niveau de sécurité ? Quelles sortes d'applications pourront passer à travers le pare−feu ? Certaines applications sont horribles pour les pare−feux pour la simple raison qu'elles négocient les ports à utiliser pour les flux de données dans une session. Ceci rend extrêmement ardu pour le pare−feu de savoir quel port ouvrir. Les applications les plus communes fonctionnent avec Iptables, mais les rares autres ne fonctionnent pas à ce jour, malheureusement.

Note

Il y a aussi quelques applications qui fonctionnent en partie, comme ICQ. L'utilisation normale d'ICQ fonctionne correctement, mais pas les fonctions de chat (discussions) et d'envoi de fichiers, car elles nécessitent un code spécifique pour décrire le protocole. Comme les protocoles ICQ ne sont pas standardisés (ils sont propriétaires et peuvent être modifiés à tout moment) la plupart des filtres ont choisis soit de ne pas le gérer, soit d'utiliser un programme de correction (patch) qui peut être appliqué aux pare−feux. Iptables a choisi de les utiliser comme programmes séparés.

Ce peut être également une bonne idée d'appliquer des mesures de sécurité en couche, ce dont nous avons discuté plus haut. Ce qui veut dire, que vous pourriez utiliser plusieurs mesures de sécurité en même temps, et pas sur un seul concept de sécurité. En prenant ceci comme concept ça décuplera vos mesures de sécurité. Pour un exemple, voir la figure :



Comme vous pouvez le voir, dans cet exemple j'ai choisi de placer un pare–feu Cisco PIX dans le périmètre des trois connexions réseau. Je peux faire du NAT sur le réseau local (LAN), comme sur la DMZ si nécessaire. Il bloquera aussi tout le trafic sortant sauf les retours http ou ftp ou ssh. Il permet également le trafic http entrant depuis le LAN et l'Internet, de même pour le trafic ftp et ssh depuis le LAN. De plus, nous pouvons noter que chaque serveur web est basé sur Linux, avec Iptables et netfilter sur les machines avec les mêmes stratégies de sécurité.

En plus de tout ça, nous pouvons ajouter Snort sur chacune des machines. Snort est un excellent système de détection d'intrusion dans un réseau (NDIS) qui vérifie les signatures dans les paquets, et il peut soit envoyer un courrier à l'administrateur ou même répondre de façon active à l'attaque en bloquant l'IP expéditeur. Il faut noter que la réponse active ne doit pas être utilisée à la légère, car Snort a un comportement un peu extrême dans le rapport d'attaques (ex. rapport sur une attaque qui n'en est pas réellement une).

Une bonne idée est aussi d'ajouter un proxy en face des serveurs web pour capturer les mauvais paquets, avec la même possibilité pour toutes les connexions web générées localement. Avec un proxy web vous pouvez gérer de façon plus précise le trafic web de vos employés, aussi bien que restreindre leur usage à certaines extensions. Avec un proxy web sur vos propres serveurs, vous pouvez l'utiliser pour bloquer certaines connexions. Un bon proxy qu'il peut être intéressant d'utiliser est Squid.

Une autre précaution qui peut être prise est d'installer Tripwire. Qui est une excellente ligne de défense en dernier ressort pour ce type d'application. Il effectue une somme de contrôle sur tous les fichiers spécifiés dans le fichier de configuration, et ensuite l'exécute depuis cron pour vérifier s'ils sont identiques, ou n'ont pas changés de façon illégale. Ce programme est capable de savoir si quelqu'un a pénétré dans le système pour le modifer. Une suggestion est de l'exécuter sur tous les serveurs web.

Une dernière chose à noter est qu'il est toujours bon de suivre les standards. Comme vous l'avez déja vu avec ICQ, si vous ne suivez pas les systèmes standardisés, ça peut provoquer de grosses erreurs. Dans votre propre environnement ceci peut être ignoré pour certains domaines, mais si vous êtes sur une large bande ou une baie de modems, ça peut devenir très important. Les personnes qui se connectent doivent toujours pouvoir accéder aux services, et vous ne pouvez pas espérer que ces personnes utilisent le système d'exploitation de votre choix. Certaines personnes travaillent sous Windows, certaines autres sous Linux ou même VMS, etc. Si vous fondez votre sécurité sur des systèmes propriétaires, vous allez au devant de problèmes.

Un bon exemple de ceci est certains services à large bande qui sont apparus en Suède et qui fondent leur sécurité sur des ouvertures de session réseau Microsoft. Ceci peut sembler une bonne idée au départ, mais lorsque vous considerez d'autres systèmes d'exploitation, ce n'est pas une si bonne idée. Comment quelqu'un tournant sous Linux fera–t–il ? Ou VAX/VMS ? Ou HP/UX ? Avec Linux on peut le faire bien sûr, si l'administrateur réseau ne refuse pas à quiconque d'utiliser le service s'ils utilisent Linux. Cependant, ce didacticiel n'est pas une discussion théologique pour savoir qui est le meilleur, c'est simplement un exemple pour vous faire comprendre que c'est une mauvaise idée de ne pas utiliser les standards.

3.4. Au prochain chapitre

Ce chapitre vous a montré certains éléments de base du filtrage IP et des mesures de sécurité que vous pouvez appliquer pour sécuriser vos réseaux, stations de travail et serveurs. Les sujets suivants ont été approchés :

- Utilisation du filtrage IP
- Stratégies de filtrage IP
- Planification de réseau
- Planification de pare−feu
- Techniques de sécurité en couche
- Segmentation de réseau

Dans le prochain chapitre nous verrons rapidement la Translation d'Adresse Réseau (NAT), et ensuite regarderons de plus près Iptables et ses fonctionnalités.

Chapitre 4. Introduction à la Traduction d'adresse Réseau

Le NAT est une des plus grosses attractions de Linux et Iptables aujourd'hui. Au lieu d'utiliser une troisième solution coûteuse comme Cisco PIX, etc. nombre de petites entreprises et d'utilisateurs privés ont choisi de fonctionner avec ces solutions. Une des principales raisons est qu'elle est bon marché et sûre. Elle peut fonctionner sur un ordinateur ancien, une distribution Linux récente que vous pouvez télécharger gratuitement sur l'Internet, une carte réseau ou deux, et le cablage.

Ce chapitre décrira la théorie de base sur le NAT, comment il peut être utilisé, comment fonctionne−t−il et ce à quoi vous devez réfléchir avant de commencer à travailler sur ces sujets.

4.1. Comment le Nat est utilisé et termes et expressions de base

De façon basique, le NAT permet à un ou plusieurs hôtes de partager la même adresse IP. Par exemple, vous avez un réseau local composé de 5−10 clients. Leur adresse de passerelle par défaut pointe vers un serveur NAT. Normalement le paquet sera simplement transmis par la machine passerelle, mais dans le cas d'un serveur NAT c'est un peu différent.

Les serveurs NAT traduisent les adresses source et destination des paquets en adresses différentes. Le serveur NAT reçoit le paquet, réécrit les adresses source et/ou destination et ensuite recalcule la somme de contrôle du paquet. Une des utilisations les plus courantes du NAT est la fonction SNAT (Traduction d'Adresse Réseau Source). De façon basique, il est utilisé dans l'exemple au−dessus si nous n'avons pas les moyens ou ne voyons pas l'intérêt d'avoir une adresse IP publique pour chacun des clients. Dans ce cas, nous utilisons une de nos IP privées du réseau local (par exemple, 192.168.1.0/24). SNAT traduira toutes les adresses 192.168.1.0 en sa propre IP publique (par exemple, 217.115.95.34). De cette façon, il y aura 5−10 clients ou beaucoup plus qui utiliserons la même IP partagée.

Il existe aussi une chose appelée DNAT, qui peut être extrêment utile quand elle est utilisée avec la configuration des serveurs. En premier, vous pouvez en attendre le plus grand bien pour économiser de l'espace IP, ensuite, vous pouvez rendre plus ou moins impénétrable un pare−feu entre votre serveur et le serveur réel de façon aisée, ou simplement partager une IP entre plusieurs serveurs séparés physiquement. Par exemple, nous pouvons faire tourner un serveur web et ftp sur la même machine, tandis qu'il existe une machine physiquement distincte contenant différents services de chat que les employés travaillant à domicile ou étant sur la route peuvent utiliser en relation avec le personnel sur le site de l'entreprise. Nous pouvons alors faire fonctionner tous ces services sur la même IP depuis l'extérieur via DNAT.

L'exemple ci−dessus est aussi basé sur des ports séparés, souvent appelé PNAT. Nous ne nous référons pas très souvent à ce terme car il est inclu dans les fonctionnalités DNAT et SNAT de netfilter.

Dans Linux, il existe actuellement deux types séparés de NAT, Fast−NAT ou Netfilter−NAT. Fast−NAT est implémenté dans le code du routage IP du noyau Linux, tandis que Netfilter−NAT est aussi implémenté dans le noyau, mais à l'intérieur du code netfilter. Dans ce didacticiel nous ne verrons pas en détail le code du routage IP, sauf pour quelques notes. Fast−NAT est généralement appelé par ce nom car il est plus rapide que le code NAT netfilter. Il ne garde pas la trace des connexions. Le traçage de connexion prend de la puissance processeur, et le ralenti, ce qui est une des principales raisons pour laquelle Fast−NAT soit plus rapide que Netfilter−NAT. Comme nous l'avons dit, le mauvais côté de Fast−NAT est qu'il ne trace pas les connexions, ce qui indique qu'il ne sera pas capable de faire du SNAT correctement pour l'ensemble des réseaux, ni capable de faire du NAT sur des protocoles complexes comme FTP, IRC et d'autres que Netfilter−NAT est capable de faire très bien.

Un mot, pour finir, qui est un synonyme de SNAT, est le terme Masquerade (masquage/usurpation). Dans Netfilter, masquerade est à peu près la même chose que SNAT avec la différence que le masquerading traduira automatiquement la nouvelle IP source en adresse IP par défaut de l'interface réseau externe.

4.2. Divergences sur l'utilisation du NAT

Comme nous l'avons vu, il existe quelques divergences mineures sur l'utilisation du NAT. Le problème principal est que certains protocoles et applications peuvent ne pas fonctionner du tout. Heureusement, ces applications ne sont pas très communes dans les réseaux que vous administrez, et dans certains cas, ça ne créera pas de soucis.

Le second problème est que les applications et protocoles ne fonctionneront que partiellement. Ces protocoles sont plus communs que ceux qui ne fonctionnent pas du tout. Si des protocoles complexes continuent à être développés, c'est un problème continuel avec lequel nous devrons vivre. Spécialement si les protocoles ne sont pas standardisés.

Le troisième, et plus gros problème, à mon point de vue, est que l'utilisateur qui est situé derrière un serveur NAT pour accéder à une connexion Internet, aura du mal à exécuter son propre serveur. Il pourra le faire, mais ça lui coûtera beaucoup de temps et de travail pour le mettre en place. Dans les entreprises, ceci est préférable plutôt que d'avoir des tonnes de serveurs lancés par différents employés joignables depuis l'Internet, sans aucune supervision.

En dernière remarque sur les divergences à propos du NAT, il devrait être fait mention que le NAT est actuellement plus ou moins du bidouillage. Le NAT est une bouée de sauvetage car le IANA et d'autres organisations ont prévenus que l'Internet croissant de façon exponentielle, les adresses IP seront bientôt limitées. Le NAT est une solution au problème d'IPv4 (IP dont nous parlons jusqu'à présent fait partie d'IPv4, c'est à dire Internet Protocol version 4). La solution à long terme est le protocole IPv6, qui résout également beaucoup d'autres problèmes. IPv6 assigne les adresses sur 128 bits, tandis que IPv4 sur 32 bits seulement. C'est un énorme accroissement d'espace d'adressage. Il peut sembler ridicule d'avoir suffisamment d'adresses pour fournir une adresse IP à chaque atome de la planète, mais d'un autre côté, l'adressage IPv4 est trop réduit maintenant.

4.3. Exemple d'une machine NAT en théorie

Un petit scenario théorique dans lequel nous voulons faire du NAT entre 2 réseaux différents et une connexion Internet. Nous voulons connecter les deux réseaux ensemble, ces deux réseaux ayant accès l'un à l'autre et à l'Internet. Nous verrons les questions de matériel auxquelles vous devrez penser avant d'implémenter une machine NAT.

4.3.1. Ce qui est nécessaire pour une machine NAT

Avant d'aller plus loin, regardons d'abord quel type de matériel est nécessaire pour assembler une machine

Linux faisant du NAT. Pour la plupart des petits réseaux, ce n'est pas un problème, mais si vous considérez des grands réseaux ça peut en devenir un. Le plus gros problème avec le NAT est qu'il consomme des ressources très rapidement. Pour un petit réseau privé avec 1−10 utilisateurs, un 486 avec 32 Mo de RAM devrait suffire. Cependant, si vous avez 100 ou d'avantage d'utilisateurs, vous devrez reconsiderez le choix du matériel. Bien sûr, il faut aussi considérer la bande passante, et combien de connexions seront ouvertes en même temps. Généralement, des ordinateurs anciens et mis au rebus, devraient faire l'affaire. En utilisant un ordinateur ancien, vous aurez un pare−feu très bon marché en comparaison d'autres pare−feux.

Vous devez également penser au problème des cartes réseau. Combien de réseaux séparés seront connectés à votre machine NAT/pare−feu ? La plupart du temps il est simplement suffisant de connecter un réseau à une connexion Internet. Si vous vous connectez à l'Internet via ethernet, vous devrez en principe avoir deux cartes ethernet, etc. Ce peut être une bonne idée de choisir des cartes réseau 10/100 mbits de bonne marque pour l'extensibilité, mais la plupart des cartes réseau feront l'affaire pour autant que les pilotes soient intégrés dans le noyau Linux. Un mot sur ce sujet : évitez d'utiliser des cartes réseau qui n'ont pas de pilote intégré dans le noyau des distributions Linux. J'ai trouvé en plusieurs occasions des cartes réseau de marques qui distribuent séparément les drivers et qui fonctionnent mal. Ils ne sont souvent pas très bien maintenus, et si vous utilisez ce type de matériel vous avez une chance très mince qu'il fonctionne correctement avec les mises à jour prochaines des noyaux Linux. Ce qui veut dire, que vous avez intérêt à payer un petit peu plus cher pour vos cartes réseau, mais ça se justifiera.

Notez que, si vous installez un pare−feu sur du matériel très ancien, essayez au moins d'utiliser les bus PCI ou mieux si possible. En priorité, les cartes réseau pourront être utilisées dans le futur lorsque vous ferez des mises à jour. Ainsi, les bus ISA sont extrêmement lent et lourds en usage CPU.

Enfin, une chose à considérer est combien de mémoire vous mettrez sur la machine NAT/pare−feu. Le mieux est de mettre au moins 64 Mo, même s'il est possible de tourner avec 32 Mo. Le NAT ne consomme pas énormément de mémoire, mais il peut être prudent d'en ajouter juste dans le cas où le trafic serait plus important que prévu.

Comme vous pouvez le voir, il y a plusieurs choses à considérer quand on parle de matériel. Mais, pour être honnête, dans la plupart des cas ça ne posera pas de problème, à moins que vous installiez une machine NAT pour un gros réseau. Pour un usage domestique vous pouvez plus ou moins utiliser le matériel que vous avez sous la main.

4.3.2. Emplacement des machines NAT

Ceci pourrait sembler simple, cependant, ce peut être plus ardu qu'on ne le penserait sur des gros réseaux. En général, la machine NAT devrait être située en périphérie du réseau. Ceci, la plupart du temps, veut dire que les machines NAT et de filtrage sont les mêmes machines, bien sûr. Ainsi, si vous avez de très grands réseaux, il peut être utile de séparer le réseau en plusieurs réseaux plus petits et assigner une machine NAT pour chacun de ces réseaux. Le NAT prenant beaucoup de temps processeur, ceci vous aidera à conserver la durée de rotation (RTT, Round Trip Time : temps que met un paquet pour joindre sa destination et envoyer un paquet en retour), la plus courte possible.

Dans notre exemple de réseau décrit ci−dessus, avec deux réseaux et une connexion Internet, nous devrons considérer la taille des deux réseaux. Si nous les considérons comme petits, 200 clients ne seront pas un problème pour une machine NAT décente. D'un autre côté, nous auront à fractionner la charge sur plusieurs machines en plaçant les IP publiques sur des machines NAT plus petites, chacune maintenant leur propre segment de réseau et laissant se rassembler le trafic sur une seule machine de routage spécifique. Ceci bien sûr, prenant en considération le fait que vous devez avoir suffisamment d'IP publiques pour toutes vos machines NAT, et qu'elles sont routées à travers votre machine routeur.

4.3.3. Comment placer les proxies ?

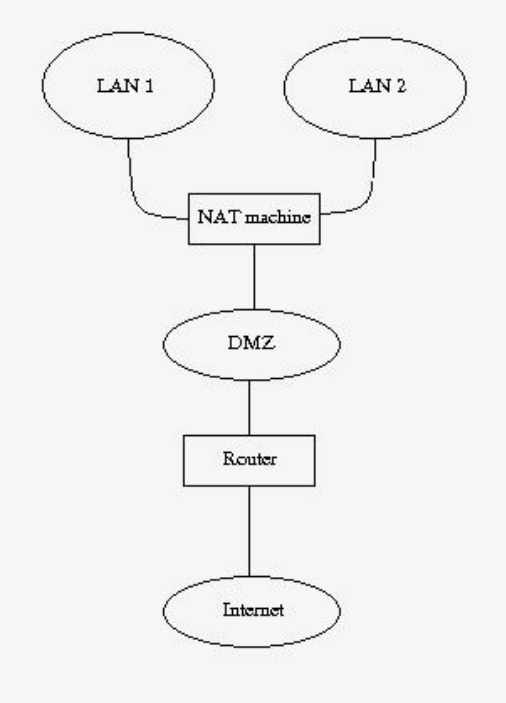
Les proxies sont généralement un problème lorsqu'ils sont accompagnés de NAT dans beaucoup de cas malheureusement, spécialement les proxies transparents. Un proxy normal ne causera pas trop de problèmes, mais en créant un proxy transparent c'est plus compliqué, spécialement dans les grands réseaux. Le premier problème est que les proxies prennent beaucoup de temps processeur, comme le NAT. Placer les deux sur la même machine n'est pas judicieux si vous avez à assurer un gros trafic réseau. Le second problème est que si vous traduisez l'IP source ainsi que l'IP destination, le proxy ne sera pas capable de savoir quel hôte contacter. Exemple, quel serveur est le client à contacter ? Localement, ceci a été résolu en ajoutant de l'information dans les structures internes de données créées pour les paquets, et ainsi les proxies comme Squid peuvent obtenir cette information.

Comme vous pouvez le voir, le problème est que vous n'avez pas beaucoup de choix si vous voulez utiliser un proxy transparent. Il existe, bien sûr, des possibilités mais elles ne sont pas réellement recommandables. Une possibilité est d'installer un proxy à l'extérieur du pare−feu et créer une entrée qui route tout le trafic web à travers cette machine, et localement NATer les paquets sur les bons ports pour le proxy. De cette façon, l'information est préservée pour le proxy et toujours disponible pour lui.

Le seconde possibilité est de simplement créer un proxy à l'extérieur du pare−feu, qui bloque tout le trafic web sauf le trafic venant du proxy. Comme ça, vous forcerez tous les utilisateurs à passer par le proxy. C'est un peu fruste comme moyen de faire, mais ça peut fonctionner.

4.3.4. Étape finale pour votre machine NAT

Dernière étape, réunissons toutes ces informations ensemble, et voyons comment nous pouvons installer une machine NAT. Regardons la figure des réseaux et voyons ce qu'elle dit. Nous avons décidé de placer un proxy à l'extérieur d'une machine NAT comme décrit plus haut. Cette zone peut être considérée comme une DMZ dans un sens, avec la machine NAT faisant routeur entre la DMZ et les deux réseaux de l'entreprise. Vous pouvez voir la topologie exacte dans l'image ci−dessous.



Tout le trafic normal provenant des réseaux NATés seront envoyés, à travers la DMZ, directement au routeur, qui adressera le trafic vers l'Internet. Sauf, le trafic web de la partie netfilter de la machine NAT, routé vers la machine proxy. De quoi parlons nous ? Un paquet http est analysé par la machine NAT. La table Mangle peut alors être utilisée pour marquer le paquet avec une marque de filtrage (nfmark). Même plus tard quand nous

voudrons router des paquets vers notre routeur, nous pourrons vérifier la marque de filtrage dans les tables de routage, et basé sur cette marque, nous pourrons choisir de router les paquets http vers le serveur proxy. Le serveur proxy fera alors son travail sur les paquets. Nous verrons ce sujet en détail plus loin.

La machine NAT possède une IP réelle disponible sur Internet, de même que le routeur et n'importe quelle autre machine présente sur l'Internet. Toutes les machines à l'intérieur de réseaux NATés auront des adresses IP privées, économisant de l'argent, et des adresses Internet.

4.4. Prochain chapitre

Nous avons expliqué dans ce chapitre le NAT et sa théorie. En particulier les divers points de vue, et quelques uns des problèmes pouvant survenir de l'utilisation du NAT et des proxies ensemble. les zones suivantes :

- Utilisation du NAT
- Composants du NAT
- Histoire du NAT
- Termes utilisés à propos du NAT
- Examens du matériel en rapport avec le NAT
- Problèmes avec le NAT

Tout ceci sera utilisé quand nous travaillerons avec Netfilter et Iptables. Le NAT est très largement utilisé dans les réseaux aujourd'hui, même si c'est seulement une solution intermédiaire pour un problème inattendu. Nous parlerons plus en détail du NAT plus loin lorsque nous verrons l'implémentation de Netfilter et Iptables.

Chapitre 5. Préparatifs

Ce chapitre est destiné à vous permettre de démarrer et vous aider à prendre conscience du rôle que Netfilter et ***iptables*** jouent aujourd'hui dans Linux. Idéalement, ce chapitre devrait vous conduire à configurer et finaliser l'expérimentation et l'installation de votre pare−feu. Avec du temps et de la persévérance, vous parviendrez à accomplir exactement ce que vous désirez.

5.1. Obtenir Iptables ?

Le paquetage ***iptables*** de l'espace utilisateur peut être téléchargé à partir de http://www.netfilter.org/. Le paquetage ***iptables*** nécessite des ressources de l'espace du noyau, qui doivent être configurées au sein de celui−ci pendant la phase ***make configure***. Sur ce sujet, les étapes indispensables seront approfondies un peu plus loin dans ce document.

5.2. Configuration du noyau

Pour exécuter les fonctions les plus élémentaires d'***iptables***, vous devez configurer les options suivantes dans le noyau, pendant la phase ***make config*** ou une autre commande apparentée:

`CONFIG_PACKET` – Cette option autorise les applications et les utilitaires à accéder directement aux périphériques réseau. Ces utilitaires sont par exemple tcpdump ou snort.

Note

Rigoureusement parlant, CONFIG_PACKET n'est pas indispensable pour faire fonctionner iptables, mais puisqu'il est énormément utilisé, j'ai choisi de l'inclure ici. Si vous ne le souhaitez pas, ne l'ajoutez pas.

`CONFIG_NETFILTER` – Cette option est nécessaire si vous comptez utiliser votre ordinateur en tant que pare−feu ou passerelle vers Internet. En définitive, c'est indispensable pour faire fonctionner tout ce qui se

trouve dans ce didacticiel. Je présume que vous le souhaitez, puisque vous lisez ceci.

Bien sûr, vous devez ajouter les pilotes spécifiques à votre interface pour obtenir un fonctionnement correct, i.e. pour les interfaces de type adaptateur Ethernet, PPP ou SLIP. Cette option ajoute seulement quelques–uns des organes élémentaires présents dans iptables. Pour être honnête, vous ne pourrez pas être véritablement productif car ceci n'ajoute qu'une architecture au noyau. Si vous voulez utiliser des options plus évoluées d'iptables, il vous faudra configurer les options adéquates dans votre noyau. Voici celles disponibles pour un simple noyau 2.4.9 accompagnées d'une courte explication:

CONFIG_IP_NF_CONNTRACK – Ce module permet de faire du traçage de connexion. Entre autres, le traçage de connexion est utilisé par le NAT et le camouflage. Si vous voulez protéger les machines d'un LAN derrière un pare–feu, vous devriez à coup sûr sélectionner cette option. Par exemple, ce module est obligatoire pour que le script rc.firewall.txt puisse fonctionner.

CONFIG_IP_NF_FTP – Ce module permet de faire du traçage de connexion sur du FTP. Comme il est habituellement difficile d'effectuer du traçage de connexion sur des connexions FTP, le module conntrack requiert le bien–nommé module d'assistance « helper ». Et cette option compile justement le module helper.

CONFIG_IP_NF_IPTABLES – Cette option est nécessaire pour effectuer n'importe quel type de filtrage, du camouflage ou du NAT. Elle insère dans le noyau toute l'architecture d'identification d'iptables. Sans cela, vous ne pourrez rien faire avec iptables.

CONFIG_IP_NF_MATCH_LIMIT – Ce module est facultatif, mais il est utilisé dans l'exemple rc.firewall.txt. Cette option fournit la correspondance LIMIT. Elle donne la possibilité de contrôler le nombre de paquets par minute avec lesquels autoriser la correspondance, suivant la définition d'une règle. Par exemple, la commande ***–m limit ––limit 3/minute*** autorise une correspondance avec un maximum de 3 paquets par minute. Ce module permet aussi d'éviter certaines attaques de type déni de service (DoS).

CONFIG_IP_NF_MATCH_MAC – Ceci permet de faire correspondre des paquets à partir des adresses MAC. Chaque adaptateur Ethernet possède sa propre adresse MAC. Il est possible de bloquer des paquets en identifiant l'adresse MAC utilisée et par conséquent, bloquer efficacement un ordinateur particulier, puisque l'adresse MAC est rarement modifiée. Cette option n'est utilisée ni dans l'exemple rc.firewall.txt, ni ailleurs.

CONFIG_IP_NF_MATCH_MARK – Ceci permet d'utiliser la correspondance ***MARK***. A titre d'exemple, on peut utiliser la cible ***MARK*** afin de marquer un paquet, et s'appuyer sur ce marquage plus loin dans la table pour éventuellement établir une correspondance. Cette option est la correspondance ***MARK***, et un peu plus loin sera décrite la cible ***MARK***.

CONFIG_IP_NF_MATCH_MULTIPORT – Ce module permet de faire correspondre des paquets sur un intervalle étendu de ports source ou destination. Normalement, c'est impossible, mais pas avec cette correspondance.

CONFIG_IP_NF_MATCH_TOS – Avec cette correspondance, on peut faire correspondre des paquets à partir du champ TOS qu'ils contiennent. TOS signifie *Type de Service* (« Type Of Service »). Il peut être défini par certaines règles dans la table mangle et grâce aux commandes ip/tc.

CONFIG_IP_NF_MATCH_TCPMSS – Cette option introduit la possibilité de faire correspondre les paquets TCP en fonction de leur champ MSS.

CONFIG_IP_NF_MATCH_STATE – Il s'agit d'une des plus importantes nouveautés vis–à–vis d'***ipchains***. Ce module permet de faire de la correspondance d'état sur les paquets. Par exemple, si vous avez déjà observé un trafic dans les deux directions sur une connexion TCP, les paquets concernés seront repérés par la mention ***ESTABLISHED***. Ce module est employé de manière intensive dans l'exemple rc.firewall.txt.

`CONFIG_IP_NF_MATCH_UNCLEAN` – Ce module introduit la possibilité d'établir une correspondance avec les paquets `IP`, `TCP`, `UDP` et `ICMP`, qui s'avèrent non–conformes à leur spécification ou invalides. Ces paquets pourront être détruits, mais il sera impossible alors de vérifier leur légitimité. Sachez que cette correspondance est encore expérimentale, donc qu'elle peut ne pas fonctionner parfaitement dans toutes les situations.

`CONFIG_IP_NF_MATCH_OWNER` – Cette option offre la possibilité d'établir une correspondance en se référant au propriétaire d'un connecteur réseau. A titre d'exemple, on peut autoriser l'accès Internet uniquement à l'utilisateur root. Ce module a été écrit à l'origine pour illustrer les possibilités du nouvel outil ***iptables***. Notez que cette correspondance est encore expérimentale, donc qu'elle pourrait ne pas fonctionner pour tout le monde.

`CONFIG_IP_NF_FILTER` – Ce module ajoute la table fondamentale `filter` qui permet d'effectuer le moindre filtrage `IP`. Dans la table `filter`, on trouve les chaînes `INPUT`, `FORWARD` et `OUTPUT`. Ce module est indispensable si vous envisagez de faire n'importe quel type de filtrage sur des paquets reçus ou envoyés.

`CONFIG_IP_NF_TARGET_REJECT` – Cette cible permet de spécifier qu'un message d'erreur `ICMP` doit être expédié en réponse à des paquets entrants, plutôt que de simplement les détruire. Gardez à l'esprit que les connexions `TCP`, à contrario des connexions `ICMP` et `UDP`, sont toujours réinitialisées ou refusées avec un paquet de type `TCP RST`.

`CONFIG_IP_NF_TARGET_MIRROR` – Ceci permet de renvoyer des paquets à leur expéditeur. Par exemple, si vous configurez une cible `MIRROR` sur le port destination `HTTP` dans votre chaîne `INPUT`, et que quelqu'un tente d'accéder à ce port, vous lui renverrez ses paquets, et il devrait probablement visualiser au final sa propre page web.

Avertissement

La cible `MIRROR` n'est pas à utiliser à la légère. Elle a été écrite à l'origine comme un module de test, et il serait sans doute très dangereux de l'utiliser (risque de DoS sérieux entre autre).

`CONFIG_IP_NF_NAT` – Ce module permet d'effectuer de la `traduction d'adresse réseau`, ou `NAT`, dans ses différentes formes. Il vous donne accès à la table nat d'iptables. Cette option est nécessaire pour réaliser de la redirection de port, du camouflage d'adresse IP, etc. Notez que cette option n'est pas indispensable pour installer un pare–feu et camoufler un réseau local, mais elle devrait vous être utile sauf si vous pouvez fournir une adresse IP unique pour chacun des hôtes. Par conséquent, cette option est nécessaire d'une part pour que le script d'exemple rc.firewall.txt puisse fonctionner correctement, et d'autre part pour votre réseau si vous n'êtes pas en mesure d'ajouter des adresses IP uniques.

`CONFIG_IP_NF_TARGET_MASQUERADE` – Ce module ajoute la cible ***MASQUERADE***. Par exemple, si vous ne connaissez pas l'adresse IP de votre connexion Internet, cette méthode permet de la récupérer en évitant le recours à du `DNAT` ou du `SNAT`. En d'autres termes, si vous utilisez `DHCP`, `PPP`, `SLIP` ou un autre moyen de connexion qui attribue lui–même l'adresse IP, vous aurez besoin d'utiliser cette cible plutôt que du `SNAT`. Le camouflage génère sur la machine une charge légèrement supérieure à du `NAT`, mais fonctionne sans connaître à l'avance l'adresse IP.

`CONFIG_IP_NF_TARGET_REDIRECT` – Cette cible est utile associée avec des `proxies d'application` par exemple. Au lieu de laisser passer un paquet directement, on peut le rediriger vers une machine locale. Autrement dit, on a la possibilité de réaliser un `proxy transparent` de cette manière.

`CONFIG_IP_NF_TARGET_LOG` – Ceci ajoute à ***iptables*** la cible ***LOG*** avec ses fonctionnalités. Ce module peut être employé pour journaliser des paquets dans syslogd, et découvrir ainsi ce qu'il advient d'eux. Cette possibilité se révèle inestimable dans le cas d'audits de sécurité, d'expertises ou pour déboguer un script en cours d'écriture.

CONFIG_IP_NF_TARGET_TCPMSS – Cette option permet de contrecarrer les Fournisseurs d'Accès à Internet (FAI) et les serveurs qui bloquent les paquets ICMP de type Fragmentation nécessaire (« Fragmentation Needed »). La conséquence de ceci est que des pages web ne passeront pas, des petits messages sont envoyés mais pas les gros, ssh fonctionne mais scp s'arrête après l'établissement de la liaison (« handshake »), etc. Dans cette situation, on peut utiliser la cible TCPMSS pour contourner cette difficulté en limitant le MSS (« Maximum Segment Size » ou taille maximum de segment) à la valeur du PMTU (« Path Maximum Transmit Unit » ou unité de transfert maximum de liaison). De cette façon, il est possible de surmonter ce que les auteurs de Netfilter appellent eux−mêmes les « FAI ou serveurs à tendance criminelle » dans l'aide de la configuration du noyau.

CONFIG_IP_NF_COMPAT_IPCHAINS – Ajoute un mode de compatibilité avec l'outil ***ipchains*** qui est devenu obsolète. Ne considérez pas ceci comme une solution sérieuse à long terme pour dénouer les problèmes de migration des noyaux Linux 2.2 vers 2.4, puisque ce mode pourrait bien disparaître dans le noyau 2.6.

CONFIG_IP_NF_COMPAT_IPFWADM – Ajoute un mode de compatibilité avec l'outil ***ipfwadm***, qui est également obsolète. Encore une fois, ne considérez pas ceci comme une solution sérieuse à long terme.

Comme vous le constatez, il existe un large éventail d'options. J'ai expliqué brièvement leur intérêt et ce qu'on pouvait attendre de chaque module. Cependant, seules sont décrites ici les options disponibles pour un noyau Linux 2.4.9 standard (saveur « vanilla »). Si vous souhaitez connaître d'autres options, je vous suggère de vous orienter vers les fonctions de patch-o-matic (POM) présentes dans l'espace utilisateur de Netfilter, qui apportent d'innombrables options supplémentaires. Les correctifs de POM sont des ajouts qu'il est envisagé d'intégrer au noyau à l'avenir, mais ils ne l'ont pas encore atteint. Les raisons sont variées – entre le patch qui n'est pas tout à fait stable, l'impossibilité à Linus Torvalds de le maintenir, ou son refus de l'ajouter à la branche principale de développement du noyau puisqu'il semble encore expérimental.

La liste d'options suivante devra être compilée dans votre noyau, ou ajoutée en tant que modules, pour que le script rc.firewall.txt fonctionne. Si vous avez besoin d'aide pour les options requises par les autres scripts, lisez la section sur les exemples de scripts de pare−feux.

- CONFIG_PACKET
- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_FTP
- CONFIG_IP_NF_IRC
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_TARGET_LOG
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_TARGET_MASQUERADE

Une dernière fois, tout ceci est indispensable pour le script rc.firewall.txt. Pour les autres scripts d'exemple, leurs conditions d'utilisation sont précisées dans leurs sections respectives. Pour l'instant, concentrez−vous sur le script principal que vous devriez déjà être en train d'étudier.

5.3. Configuration du domaine utilisateur

Avant tout, apprenons à compiler le paquetage ***iptables***. Il est important de réaliser que la configuration et la compilation d'iptables sont étroitement liées à celles du noyau. Certaines distributions sont fournies avec le paquetage ***iptables*** préinstallé, Red Hat en fait partie. Cependant, sous Red Hat, il est désactivé par défaut. Nous montrerons comment l'activer, et nous verrons d'autres distributions au cours de ce chapitre.

5.3.1. Compilation des applications

Tout d'abord, dépaquetez l'archive d'***iptables***. Dans le cas présent, le paquetage *iptables 1.2.6a* est utilisé, ainsi que le noyau 2.4 (vanilla). Dépaquetez le de manière classique, avec la commande ***bzip2 −cd iptables−1.2.6a.tar.bz2 | tar −xvf −*** (ou avec ***tar −xjvf iptables−1.2.6a.tar.bz2***, qui devrait aboutir au même résultat ; cependant, ça peut ne pas marcher avec d'anciennes versions de la commande ***tar***). Cette archive doit être dépaquetée dans un répertoire appelé `iptables-1.2.6a`. N'hésitez pas à lire le fichier `iptables-1.2.6a/INSTALL` qui contient des informations pertinentes sur la compilation et la préparation à l'exécution du programme.

Ensuite, vous avez la possibilité de configurer et installer les modules et options supplémentaires du noyau. L'étape décrite à présent vérifie et installe les patches standards en attente d'être intégrés au noyau. Il y a d'autres patches encore plus expérimentaux, qui devraient être disponibles seulement après certaines étapes.

Note

Certains de ces patches sont particulièrement expérimentaux et les installer pourrait ne pas être une très bonne idée. Pourtant, il y a une quantité de correspondances et de cibles extrèmement intéressantes lors de cette étape d'installation, donc n'ayez pas peur d'y jeter un oeil.

Pour finaliser cette étape, il suffit d'exécuter ceci à partir de la racine de l'archive d'iptables :

make pending−patches KERNEL_DIR=/usr/src/linux/

La variable `KERNEL_DIR` devrait pointer sur l'emplacement des sources du noyau. Normalement, il s'agit de `/usr/src/linux/`, mais ça peut changer et vous connaissez sûrement leur localisation.

On vous interroge seulement sur certains patches qui, de toute façon, sont presque entrés dans le noyau. Il peut y avoir davantage de patches et d'ajouts que les développeurs de Netfilter aimeraient voir ajouter au noyau, mais qui en sont encore un peu éloignés actuellement. Voici une façon de les installer :

make most−of−pom KERNEL_DIR=/usr/src/linux/

La commande précédente vous interroge sur les éléments à installer – ce que l'on appelle ***patch−o−matic*** dans le monde de Netfilter, mais éviter les patches les plus extrêmes, qui peuvent causer des ravages dans votre noyau. Observez qu'il est écrit « interroge », parce que c'est le comportement actuel de ces commandes. Elles vous interrogent avant de modifier quoi que ce soit dans les sources du noyau. Afin de forcer l'installation de *tous* les éléments de `patch-o-matic`, vous devez exécuter la commande suivante :

make patch−o−matic KERNEL_DIR=/usr/src/linux/

N'oubliez pas de lire attentivement l'aide de chaque patch avant de faire quoi que ce soit. Certains patches en détruisent d'autres, alors que d'autres encore détruisent votre noyau si vous les associez avec certains patches de `patch-o-matic`, etc.

Note

Vous pouvez ignorer complètement les étapes précédentes si vous ne souhaitez pas patcher votre noyau, autrement dit, elles ne sont pas obligatoires. Toutefois, quelques éléments de `patch-o-matic` sont tellement intéressants qu'ils méritent votre attention, et il n'y a aucun danger à exécuter ces commandes pour visualiser leur contenu.

Après cela, vous en avez fini avec l'installation des éléments de `patch-o-matic`. Vous pouvez maintenant compiler un nouveau noyau pour vous servir des nouveaux patches que vous avez inclus dans les sources. N'oubliez pas de reconfigurer le noyau puisque les nouveaux patches ne font certainement pas partie des options définies. Vous pouvez procéder à la compilation du noyau après celle du programme ***iptables*** de l'espace utilisateur, si ça vous chante.

Poursuivez en compilant l'application ***iptables***. Pour lancer cette compilation, vous entrez une simple commande comme ceci :

make KERNEL_DIR=/usr/src/linux/

L'application du domaine utilisateur devrait se compiler sans difficulté. Si ce n'est pas le cas, vous êtes face à vous−même, ou vous pouvez vous inscrire à la liste de diffusion de Netfilter , où vous avez la chance de pouvoir demander de l'aide sur vos problèmes. Il y a peu de choses qui peuvent mal tourner dans l'installation d'***iptables***, donc ne paniquez pas si ça ne fonctionne pas. Soyez logique et découvrez ce qui cloche, ou bien trouvez quelqu'un susceptible de vous aider.

Si tout s'est passé en douceur, vous êtes prêt désormais à installer les fichiers binaires. Pour ce faire, vous devez appliquer la commande suivante :

make install KERNEL_DIR=/usr/src/linux/

Soyons optimiste, tout doit maintenant fonctionner parfaitement dans le programme. Pour exploiter toute modification de l'application ***iptables***, vous devez à présent recompiler et réinstaller vos noyau et modules, si ce n'est pas déjà fait. Pour approfondir l'installation des applications à partir des sources, lisez le fichier `INSTALL` qui accompagne les sources et contient d'excellentes informations sur le sujet.

5.3.2. Installation sur Red Hat 7.1

Red Hat 7.1 est fournie avec un noyau 2.4.x précompilé avec `Netfilter` et ***iptables***. Il contient aussi tous les programmes élémentaires du domaine utilisateur et les fichiers de configuration exigés pour l'exécution. Cependant, l'équipe de Red Hat a désactivé la totalité en optant pour la rétrocompatibilité avec le module ***ipchains***. Ennuyé de répéter la même chose, et comme nombre de gens continuent à demander sur différentes listes de diffusion pourquoi ***iptables*** ne marche pas, abordons rapidement comment désactiver le module d'ipchains pour le remplacer par ***iptables***.

Note

L'installation par défaut de Red Hat 7.1 donne malheureusement une vieille version des applications de l'espace utilisateur. De fait, vous désirerez certainement compiler une nouvelle version des applications, associée à un noyau récent et personnalisé avant d'exploiter complètement ***iptables***.

En premier lieu, il faut arrêter le module ***ipchains*** de telle sorte qu'il ne démarre plus à l'avenir. Pour cela, quelques noms de fichiers doivent être changés dans l'arborescence `/etc/rc.d/`. La commande suivante devrait suffire :

chkconfig −−level 0123456 ipchains off

Avec ceci, tous les liens symboliques qui pointent vers le script `/etc/rc.d/init.d/ipchains` sont déplacés vers K92ipchains. La première lettre, S par défaut, indique de lancer le script de démarrage (« initscript ») correspondant. La conversion du S en K stipule d'interrompre (« Kill ») le service, ou de ne pas exécuter le script si le service n'a pas déjà démarré. Dorénavant, le script ne démarrera plus.

D'autre part, pour arrêter dès maintenant le service en cours d'exécution, il est nécessaire de lancer une autre commande. Il s'agit de la commande ***service*** qui permet de manipuler des services en cours d'exécution. Ainsi, pour stopper le service ***ipchains***, il suffit de faire :

service ipchains stop

Maintenant, il reste à démarrer le service ***iptables***. Tout d'abord, il faut connaître les niveaux d'exécution (« run−levels ») où l'on veut positionner ce service. Normalement, ça devrait être les niveaux 2, 3 et 5. Ils servent aux choses suivantes :

- 2. Multi−utilisateur sans NFS ou identique à 3 en l'absence de réseau.
- 3. Mode multi−utilisateur intégral, c.−à−d. le niveau d'exécution normal à lancer
- 5. X11. Utilisé si vous démarrez automatiquement sous Xwindow.

On impose de lancer ***iptables*** dans ces niveaux d'exécution avec la commande :

chkconfig −−level 235 iptables on

La commande ci−dessus permet de lancer le service ***iptables*** dans les niveaux d'exécution 2, 3 et 5. Si vous désirez qu'il en soit autrement, modifiez la commande en conséquence. Toutefois, aucun des autres niveaux d'exécution ne devrait être sélectionné, donc vous n'avez pas besoin d'activer ***iptables*** pour ces niveaux−là. Le niveau 1 concerne le mode un seul utilisateur, c−à−d. quand vous devez réparer une machine dysfonctionnante. Le niveau 4 devrait être inutilisé, et le niveau 6 est réservé à l'extinction de l'ordinateur.

Pour activer le service ***iptables***, lancez simplement la commande :

service iptables start

Initialement, il n'y a aucune règle dans le script ***iptables***. Pour ajouter des règles sur une Red Hat 7.1, il existe deux méthodes. Premièrement, vous pouvez éditer le script `/etc/rc.d/init.d/iptables`. Cette approche a un désagréable inconvénient, celui de voir toutes ses règles effacées si vous mettez à jour le paquetage iptables par RPM. La deuxième méthode consiste à charger le livre de règles, puis à le sauvegarder par le biais de la commande ***iptables−save***, et enfin à automatiser son chargement au démarrage avec les scripts de rc.d.

Tout d'abord, sera décrite la configuration d'***iptables*** avec des manipulations de copier/coller dans le script ***iptables*** du répertoire init.d. Pour ajouter des règles qui seront appliquées au démarrage du service, vous pouvez les insérer soit derrière la section "start)", soit à l'intérieur de la fonction "start()". Si vous choisissez la section "start)", vous devez penser à empêcher l'exécution de la fonction "start()" dans cette section. A propos, songez également à éditer la section "stop)" pour préciser au script les actions à entreprendre soit lorsqu'on éteint l'ordinateur, soit lorsqu'on active un niveau d'exécution qui ne nécessite pas ***iptables***. Par la même occasion, n'oubliez pas de vérifier les sections "restart" et "condrestart". Sachez que tout votre travail sera sûrement effacé si vous avez opté pour "Red Hat Network" qui met à jour automatiquement vos paquetages. Ce sera aussi le cas avec une mise à jour du paquetage RPM ***iptables***.

La seconde méthode de configuration est décrite ici. En premier lieu, créez un livre de règles qui répond à votre besoin, et écrivez−le dans un fichier de script shell ou utilisez−le directement avec ***iptables***, mais n'oubliez pas de l'expérimenter. Lorsque vous trouvez une configuration qui fonctionne sans problème et sans faille, utilisez la commande ***iptables−save***. Typiquement, vous pouvez faire ***iptables−save > /etc/sysconfig/iptables***, pour sauvegarder le livre de règles dans le fichier `/etc/sysconfig/iptables`. Ce fichier est lu automatiquement par le script ***iptables*** de rc.d pour restituer le livre de règles à la demande. Une autre possibilité est de sauvegarder le script en exécutant ***service iptables save***, qui sauvegarde automatiquement vers le fichier `/etc/sysconfig/iptables`. Au prochain démarrage de votre ordinateur, le script ***iptables*** de rc.d fera appel à la commande ***iptables−restore*** pour restituer le livre de règles à partir du fichier sauvegardé `/etc/sysconfig/iptables`. Ne mélangez pas ces deux méthodes, susceptibles de se nuire mutuellement et rendre votre pare−feu inopérant.

Une fois toutes ces étapes achevées, vous pouvez désinstaller les paquetages ***ipchains*** et ***iptables***. En effet, ceci permet d'éviter au système tout risque de confusion entre l'application ***iptables*** préinstallée et l'application ***iptables*** de l'espace utilisateur. Cette étape n'est utile que si vous installez ***iptables*** à partir des fichiers sources. Il n'y a rien d'inhabituel à voir le nouveau et l'ancien paquetage se mélanger, puisque l'installation à partir de rpm positionne les fichiers à des emplacements non standards qui ne seront pas écrasés par l'installation du nouveau paquetage ***iptables***. Pour procéder à la désinstallation, exécutez ceci :

rpm −e iptables

D'ailleurs, pourquoi conserver également ***ipchains*** s'il n'a plus d'utilité ? Supprimez−le de la même manière que les vieux fichiers binaires d'***iptables*** avec la commande :

rpm −e ipchains

Finalement, vous avez terminé la mise à jour du paquetage d'***iptables*** à partir des sources, en suivant les instructions d'installation. Maintenant, plus un seul fichier binaire, de bibliothèque ou de directive d'inclusion ne devrait résider sur le système.

Chapitre 6. Traversée des tables et des chaînes

Ce chapitre décrit la façon dont les paquets traversent les différentes chaînes, et dans quel ordre. De même, il explique l'ordre dans lequel les tables sont traversées. Vous percevrez l'importance de ce fonctionnement plus loin, lors de l'écriture de vos propres règles. D'autres points seront examinés, liés à des éléments dépendants du noyau, car ils entrent également en considération dans ce chapitre. Entre autres, les différentes décisions de routage. C'est particulièrement utile si vous voulez écrire des règles pour ***iptables*** qui peuvent modifier les consignes/règles de routage des paquets, c−à−d. pourquoi et comment les paquets sont routés; le ***DNAT*** et le ***SNAT*** sont des exemples caractéristiques. Bien sûr, il ne faut pas oublier les bits de `TOS`.

6.1. Généralités

Quand un paquet arrive pour la première fois dans un pare−feu, il rencontre le niveau matériel, puis il est recueilli par le pilote de périphérique approprié au sein du noyau. Ensuite, le paquet enchaîne une succession d'étapes dans le noyau, avant d'être envoyé à l'application adéquate (localement), ou expédié à un autre hôte − ou quoi que ce soit d'autre.

D'abord, analysons un paquet destiné à la machine locale. Il enchaîne les étapes suivantes avant d'être réellement délivré à l'application qui le reçoit :

Tableau 6.1. Hôte local destinataire (votre propre machine)

Étape	Table	Chaîne	Commentaire
1			Sur le câble (ex. Internet)
2			Arrive sur l'interface (ex. eth0)
3	mangle	PREROUTING	Cette chaîne sert normalement à modifier les paquets, i.e. changer les bits de `TOS`, etc.
4	nat	PREROUTING	Cette chaîne sert principalement au `DNAT`. Évitez de filtrer dans cette chaîne puisqu'elle est court−circuitée dans certains cas.
5			Décision de routage, i.e. le paquet est−il destiné à notre hôte local, doit−il être réexpédié et où ?
6	mangle	INPUT	Ici, il atteint la chaîne `INPUT` de la table mangle. Cette chaîne permet de modifier les paquets, après leur routage, mais avant qu'ils soient réellement envoyés au processus de la machine.
7	filter	INPUT	C'est l'endroit où est effectué le filtrage du trafic entrant à destination de la machine locale. Notez bien que tous les paquets entrants et destinés à votre hôte passent par cette chaîne, et ceci quelle que soit leur interface ou leur provenance d'origine.

8			Processus/application local (i.e. programme client/serveur)

Remarquez que cette fois, le paquet est transmis à travers la chaîne `INPUT` au lieu de la chaîne `FORWARD`. C'est parfaitement logique. Et c'est certainement la seule chose logique à vos yeux dans le parcours des tables et des chaînes pour le moment, mais si vous continuez d'y réfléchir, vous trouverez ceci de plus en plus clair au fur et à mesure.

À présent, analysons les paquets sortant de notre hôte local et les étapes qu'ils enchaînent.

Tableau 6.2. Hôte local source (votre propre machine)

Étape	Table	Chaîne	Commentaire
1			Processus/application local (i.e. programme client/serveur)
2			Décision de routage. Quelle adresse source doit être utilisée, quelle interface de sortie, et d'autres informations nécessaires qui doivent être réunies.
3	mangle	OUTPUT	C'est là où les paquets sont modifiés. Il est conseillé de ne pas filtrer dans cette chaîne, à cause de certains effets de bord. C'est aussi où le traçage de connexion généré localement prend place, nous verrons celà dans le chapitre La machine d'état.
4	nat	OUTPUT	Cette chaîne permet de faire du NAT sur des paquets sortant du pare−feu.
5			Décision de routage, comment les modifications des mangle et nat précédents peuvent avoir changé la façon dont les paquets seront routés.
6	filter	OUTPUT	C'est de là que les paquets sortent de l'hôte local.
7	mangle	POSTROUTING	La chaîne `POSTROUTING` de la table mangle est principalement utilisée lorsqu'on souhaite modifier des paquets avant qu'ils quittent la machine mais après les décisions de routage. Cette chaîne est rencontrée d'une part par les paquets qui ne font que transiter par le pare−feu, d'autre part par les paquets créés par le pare−feu lui−même.
8	nat	POSTROUTING	C'est ici qu'est effectué le `SNAT`. Il est conseillé de ne pas filtrer à cet endroit à cause des effets de bord, certains paquets peuvent se faufiler même si un comportement par défaut a été défini pour la cible ***DROP***.
9			Sort par une certaine interface (ex. eth0)
10			Sur le câble (ex. Internet)

Dans cet exemple, on suppose que le paquet est destiné à un autre hôte sur un autre réseau. Le paquet parcourt les différentes étapes de la façon suivante :

Tableau 6.3. Paquets redirigés

Étape	Table	Chaîne	Commentaire
1			Sur le câble (ex. Internet)
2			Arrive sur l'interface (ex. eth0)
3	mangle	PREROUTING	Cette chaîne est typiquement utilisée pour modifier les paquets, i.e. changer les bits de `TOS`, etc. C'est ici aussi que le traçage de connexion généré non−localement prend place, nous verrons celà dans le chapitre La machine d'état.

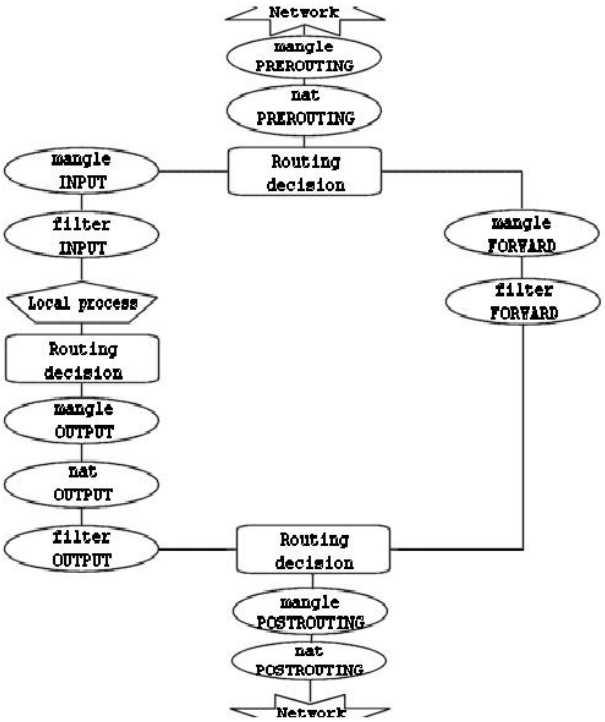
4	nat	PREROUTING	Cette chaîne sert principalement à réaliser du `DNAT`. Le `SNAT` est effectué plus loin. Evitez de filtrer dans cette chaîne car elle peut être court−circuitée dans certains cas.
5			Décision de routage, c−à−d. le paquet est−il destiné à votre hôte local, doit−il être redirigé et où ?
6	mangle	FORWARD	Le paquet est alors envoyé à la chaîne `FORWARD` de la table mangle. C'est utile pour des besoins très spécifiques, lorsque l'on souhaite modifier des paquets après la décision de routage initiale, mais avant la décision de routage finale effectuée juste avant l'envoi du paquet.
7	filter	FORWARD	Le paquet est routé vers la chaîne `FORWARD`. Seuls les paquets réexpédiés arrivent ici, et c'est ici également que tout le filtrage est effectué. Notez bien que tout trafic redirigé passe par ici (et pas seulement dans un sens), donc vous devez y réfléchir en rédigeant vos règles.
8	mangle	POSTROUTING	Cette chaîne est employé pour des formes particulières de modification de paquets, que l'on veut appliquer postérieurement à toutes les décisions de routage, mais toujours sur cette machine.
9	nat	POSTROUTING	Cette chaîne est employé pour des formes particulières de modification de paquets, que l'on veut appliquer postérieurement à toutes les décisions de routage, mais toujours sur cette machine
10			Sort par l'interface de sortie (ex. eth1).
11			Sort de nouveau par le câble (ex. LAN).

Comme vous pouvez le constater, il y a de nombreuses étapes à franchir. Un paquet peut être arrêté dans n'importe quelle chaîne d'***iptables***, et même ailleurs s'il est malformé. Pourtant, il est intéressant de se pencher sur le sort du paquet vu par ***iptables***. Remarquez qu'aucune chaîne ou table spécifique n'est définie pour des interfaces différentes, ou quoi que ce soit de semblable. La chaîne `FORWARD` est systématiquement parcourue par les paquets qui sont redirigés par l'intermédiaire de ce pare−feu/routeur.

Attention

N'utilisez pas la chaîne `INPUT` pour filtrer dans le scénario précédent ! `INPUT` n'a de sens que pour des paquets destinés à votre hôte local, autrement dit qui ne seront routés vers aucune autre destination.

Maintenant, vous avez découvert comment les différentes chaînes sont traversées selon trois scénarios distincts. On peut en donner une représentation graphique :



Pour être plus clair, ce dessin mérite quelques explications. Si un paquet atteignant la première décision de routage n'est pas destiné à la machine locale, il sera orienté vers la chaîne `FORWARD`. Par contre, s'il est destiné à une `adresse IP` que la machine écoute, ce paquet sera envoyé vers la chaîne `INPUT`, et donc à la machine locale.

Il est important de remarquer que même si des paquets sont destinés à la machine locale, leur adresse de destination peut être modifiée à l'intérieur de la chaîne `PREROUTING` par une opération de `NAT`. En effet, puisque ceci a lieu avant la première décision de routage, le paquet ne sera examiné qu'après un éventuel changement. A cause de cette particularité, le routage peut être altéré avant que la décision de routage ne soit prise. Notez bien que *tous* les paquets transiteront par l'un ou l'autre des chemins de ce dessin. Si vous réalisez du `DNAT` sur un paquet pour le renvoyer sur le réseau duquel il provient, il continuera malgré tout sa route à travers les chaînes restantes jusqu'à ce qu'il retourne sur le réseau externe.

Astuce

Si vous pensez avoir besoin d'informations supplémentaires, vous pouvez utiliser le script rc.test–iptables.txt. Ce script de test devrait vous procurer des règles suffisantes pour expérimenter et comprendre de quelle façon les tables et les chaînes sont traversées.

6.2. La table mangle

Comme il a déjà été précisé, le rôle principal de cette table devrait être de modifier des paquets. En d'autres termes, vous pouvez utiliser en toute liberté les correspondances de la table mangle, qui permettent de changer les champs de `TOS` (type de service), et d'autres.

Attention

Vous avez été suffisamment prévenus de ne pas utiliser cette table pour effectuer du filtrage ; de même, les opérations de ***DNAT***, de ***SNAT*** ou de ***camouflage*** ne fonctionnent pas dans cette table.

Les cibles suivantes sont valides uniquement dans la table mangle :

- TOS

- TTL
- MARK

La cible ***TOS*** permet de définir et/ou modifier le champ de `Type de Service` d'un paquet. C'est utile pour définir des stratégies réseau concernant le choix de routage des paquets. Sachez que, d'une part ceci n'a pas été perfectionné, d'autre part ce n'est pas vraiment implémenté sur Internet car la majorité des routeurs ne se préoccupe pas de ce champ, et quelquefois même, ils adoptent un comportement erroné. Bref, ne configurez pas ce champ sur les paquets qui naviguent sur Internet, sauf si vous souhaitez leur appliquer des décisions de routage, avec iproute2.

La cible ***TTL*** permet de modifier le champ durée de vie ou `TTL` ("Time To Live") d'un paquet. Il est possible par exemple de spécifier aux paquets d'avoir un champ `TTL` spécifique. Ceci peut se justifier lorsque vous ne souhaitez pas être rejeté par certains Fournisseurs d'Accès à Internet (FAI) trop indiscrets. En effet, il existe des FAI qui désapprouvent les utilisateurs branchant plusieurs ordinateurs sur une même connexion, et de fait, quelques−uns de ces FAI sont connus pour inspecter si un même hôte génère différentes valeurs `TTL`, supposant ainsi que plusieurs machines sont branchées sur la même connexion.

La cible ***MARK*** permet d'associer des valeurs de `marquage` particulières aux paquets. Elles peuvent ensuite être identifiées par les programmes ***iproute2*** pour appliquer un routage différent en fonction de l'existence ou l'absence de telle ou telle `marque`. On peut ainsi réaliser de la restriction de bande passante et de la gestion de priorité ("Class Based Queuing").

6.3. La table nat

Cette table devrait être utilisée seulement pour effectuer de la traduction d'adresse réseau (`NAT`) sur différents paquets. Autrement dit, elle ne devrait servir qu'à traduire le champ de l'adresse source d'un paquet ou celui de l'adresse destination. Précisons à nouveau que seul le premier paquet d'un flux rencontrera cette chaîne. Ensuite, les autres paquets subiront automatiquement le même sort que le premier. Voici les cibles actuelles capables d'accomplir ce genre de choses :

- DNAT
- SNAT
- MASQUERADE
- REDIRECT

La cible ***DNAT*** est généralement utile dans le cas où vous détenez une adresse IP publique et que vous désirez rediriger les accès vers un pare−feu localisé sur un autre hôte (par exemple, dans une zone démilitarisée ou `DMZ`). Concrètement, on change l'adresse de destination du paquet avant de le router à nouveau vers l'hôte désigné.

La cible ***SNAT*** est quant à elle employée pour changer l'adresse de source des paquets. La plupart du temps, vous dissimulerez votre réseau local ou votre `DMZ`, etc. Un très bon exemple serait donné par un pare−feu pour lequel l'adresse externe est connue, mais qui nécessite de substituer les adresses IP du réseau local avec celle du pare−feu. Avec cette cible, le pare−feu effectuera automatiquement sur les paquets du ***SNAT*** dans un sens et du ***SNAT inverse*** dans l'autre, rendant possible les connexions d'un `réseau local` sur Internet. A titre d'exemple, si votre réseau utilise la famille d'adresses 192.168.0.0/masque_réseau, les paquets envoyés sur Internet ne reviendront jamais, parce que l'IANA (institut de régulation des adresses) a considéré ce réseau (avec d'autres) comme privé, et a restreint son usage à des `LAN`s isolés d'Internet.

La cible ***MASQUERADE*** s'utilise exactement de la même façon que la cible ***SNAT***, mais la cible ***MASQUERADE*** demande un peu plus de ressources pour s'exécuter. L'explication vient du fait que chaque fois qu'un paquet atteint la cible ***MASQUERADE***, il vérifie automatiquement l'adresse IP à utiliser, au lieu de se comporter comme la cible ***SNAT*** qui se réfère simplement à l'unique adresse IP configurée. Par conséquent, la cible ***MASQUERADE*** permet de faire fonctionner une système d'adressage IP dynamique sous

DHCP, que votre FAI devrait vous procurer pour des connexions à Internet de type PPP, PPPoE ou SLIP.

6.4. La table filter

La table filter sert principalement à filtrer les paquets. On peut établir une correspondance avec des paquets et les filtrer comme on le désire. C'est l'endroit prévu pour intervenir sur les paquets et analyser leur contenu, c'est–à–dire les détruire (avec la cible ***DROP***) ou les accepter (avec ***ACCEPT***) suivant leur contenu. Bien entendu, il est possible de réaliser préalablement du filtrage ; malgré tout, cette table a été spécialement conçue pour ça. Presque toutes les cibles sont utilisables dans celle–ci. D'autres informations seront données sur la table filter, cependant vous savez maintenant que c'est l'emplacement idéal pour effectuer votre filtrage principal.

Chapitre 7. La machine d'état

Ce chapitre aborde et explique en détails la machine d'état. Après avoir lu ceci, vous devriez avoir pleinement compris son fonctionnement. Vous parcourerez un nombre important d'exemples sur la façon dont les états sont traités par la machine d'état elle–même. Ces cas concrets devraient vous éclairer parfaitement.

7.1. Introduction

La machine d'état correspond à une partie spéciale à l'intérieur d'iptables. En fait, elle porte très mal son nom puisqu'il s'agit en réalité d'une machine de traçage de connexion. Cependant, la plupart des gens la connaissent sous la première appellation. Au cours de ce chapitre, les deux noms sont utilisés indistinctement comme ils sont synonymes. Ceci ne devrait pas trop vous perturber. Le traçage de connexion est effectué afin que l'architecture de Netfilter puisse connaître l'état d'une connexion spécifique. Les pare–feux qui implémentent ceci sont habituellement appelés pare–feux à état. Un pare–feu à état est généralement bien plus sûr qu'un pare–feu sans état, puisqu'il impose une plus grande rigueur sur l'écriture des livres de règles.

Dans iptables, les paquets peuvent être reliés aux connexions tracées dans quatre états différents, qui sont connus sous les noms de ***NEW***, ***ESTABLISHED***, ***RELATED*** et ***INVALID***. Chacun de ces états sera approfondi plus loin. Avec la correspondance ***––state***, il est facile de contrôler qui, ou ce qui, est autorisé à démarrer de nouvelles sessions.

L'intégralité du traçage de connexion est effectué par une structure particulière à l'intérieur du noyau appelée conntrack. La structure conntrack peut soit être chargée comme un module, soit être interne au noyau. La plupart du temps, on a besoin de fonctions supplémentaires de traçage de connexion autres que celles proposées par défaut dans le moteur conntrack. De ce fait, des parties spécifiques de conntrack prennent en charge les protocoles TCP, UDP et ICMP. Ces modules capturent des informations spécifiques et uniques sur les paquets, afin de pouvoir tracer chaque flux de données. L'information récupérée par conntrack lui permet de connaître l'état dans lequel se trouve chaque flux actuellement. Par exemple, un flux UDP est, en général, identifié uniquement par son adresse IP destination , son adresse IP source, son port destination et son port source .

Dans les noyaux précédents, il était possible d'activer ou désactiver la défragementation. Cependant, depuis qu'iptables et Netfilter ont été incorporés avec, en particulier, le traçage de connexion, cette option a disparu. La raison en est simple, le traçage de connexion ne peut pas fonctionner correctement sans défragmenter les paquets, par conséquent la défragmentation a été intégrée dans conntrack, et elle est réalisée automatiquement. Elle ne peut donc plus être désactivée, sauf en désactivant le traçage de connexion. En définitive, la défragmentation a toujours cours si le traçage de connexion est actif.

Le traçage de connexion est entièrement pris en charge dans la chaîne PREROUTING, sauf pour les paquets générés en local, qui sont pris en charge dans la chaîne OUTPUT. Ceci signifie qu'iptable effectue tous les calculs d'état dans la chaîne PREROUTING. Si on envoie le premier paquet d'un flux, l'état est défini comme

NEW dans la chaîne `OUTPUT`, et quand on reçoit un paquet de réponse, l'état passe à ***ESTABLISHED***, et ainsi de suite. Si le premier paquet n'est pas envoyé par nous−mêmes, l'état ***NEW*** est naturellemt défini dans la chaîne `PREROUTING`. Ainsi, tous les changements d'état et calculs sont réalisés dans les chaînes `PREROUTING` et `OUTPUT` de la table nat.

7.2. Les entrées de conntrack

Examinons rapidement le contenu d'une donnée d'entrée de conntrack et lisons−la dans `/proc/net/ip_conntrack`. Ce lien contient une liste de toutes les entrées actuelles de la base de données de conntrack. Si vous avez chargé le module `ip_conntrack`, faites un ***cat*** de `/proc/net/ip_conntrack` pour obtenir quelque−chose comme ceci :

```
tcp      6 117 SYN_SENT src='1'92.168.1.6 dst='1'92.168.1.9 sport=32775 \
     dport='2'2 [UNREPLIED] src='1'92.168.1.9 dst='1'92.168.1.6 sport='2'2 \
     dport=32775 [ASSURED] use='2'
```

Cet exemple contient toute l'information gérée par le module conntrack pour savoir dans quel état se trouve une connexion. Tout d'abord, il y a le protocole, ici tcp. Ensuite, encore le protocole mais codé en décimal. Après cela, on voit combien de temps doit survivre cette entrée de conntrack. La valeur à cet instant est de 117 secondes, et elle est decrémentée régulièrement jusqu'à ce qu'on voit à nouveau du trafic pour cette connexion. Cette valeur est alors réinitialisée à la valeur par défaut pour l'état en question à cet instant donné. Ensuite vient l'état actuel de cette entrée. Dans le cas présenté ci−dessus, on visualise une connexion qui est dans l'état `SYN_SENT`. La valeur interne d'une connexion est légèrement différente de celles utilisées en externe avec ***iptables***. La valeur `SYN_SENT` indique que cette connexion a seulement vu un paquet `TCP SYN` dans une direction. Puis, on voit l'`adresse IP source`, l'`adresse IP destination`, le `port source` et le `port destination`. Arrivé à ce niveau, on voit un mot−clé spécifique qui signale qu'aucun trafic n'a été observé en retour pour cette connexion. Enfin, on voit ce qui est attendu pour les paquets de réponse. Entre autres, l'`adresse IP source` et l'`adresse IP destination` (qui sont inversées, puisque le paquet attendu doit être dirigé dans l'autre sens). La même chose s'applique aux `port source` et `port destination` de la connexion. Ces valeurs nous intéressent particulièrement.

Les entrées du traçage de connexion peuvent prendre un ensemble de valeurs différentes, toutes spécifiées dans les en−têtes de conntrack et disponibles dans les fichiers `linux/include/netfilter−ipv4/ip_conntrack*.h`. Ces valeurs dépendent du sous−protocole `IP` qu'on utilise. Les protocoles `TCP`, `UDP` et `ICMP` correspondent à des valeurs fixées et spécifiées dans le fichier `linux/include/netfilter−ipv4/ip_conntrack.h`. Ceci sera analysé plus en détails lors de l'analyse de chaque protocole ; cependant, ils ne seront pas employés intensivement dans ce chapitre, puisqu'ils ne sont pas utilisés en dehors du fonctionnement interne de conntrack. Ainsi, en fonction de l'évolution de cet état, on change la valeur du temps restant avant la destruction de la connexion.

Note

Récemment, un nouveau patch est devenu disponible dans patch−o−matic, appelé tcp−window−tracking. Il ajoute, entre autres, toutes les temporisations précitées aux variables spéciales sysctl, ce qui signifie qu'elles peuvent être modifiées à la volée, alors que le système est toujours en fonctionnement. Par conséquent, il ne devient plus indispensable de recompiler le noyau à chaque changement dans les temporisations.

Tout ceci peut être modifié par le biais d'appels système spécifiques, disponibles dans le répertoire `/proc/sys/net/ipv4/netfilter`. Vous devriez regarder en particulier les variables `/proc/sys/net/ipv4/netfilter/ip_ct_*`.

Quand une connexion a observé du trafic dans les deux directions, l'entrée de conntrack efface le fanion [UNREPLIED] , et donc le réinitialise. Elle le remplace par le fanion [ASSURED], vers la fin. Il signale que cette connexion est confirmée, donc elle ne sera pas supprimée si on atteint le maximum de connexions tracées possible. En fait, les connexions estampillées [ASSURED] ne seront pas supprimées, au contraire des connexions non confirmées (sans le fanion [ASSURED]). Le nombre maximum de connexions gérées par la table de traçage de connexion dépend d'une variable qui peut être définie à l'aide de la fonction ip−sysctl dans les noyaux récents. La valeur par défaut prise en charge varie fortement avec la quantité de mémoire disponible. Avec 128 Mo de RAM, vous pourrez avoir 8192 entrées possibles, et avec 256 Mo, ce sera 16376 entrées. Vous pouvez lire et définir vos réglages à l'aide de /proc/sys/net/ipv4/ip_conntrack_max.

Un moyen différent de faire ceci, plus efficace, est de placer la taille de la fonction de hachage pour le module ip_conntrack une fois qu'il est chargé. Dans des circonstances normales ip_conntrack_max égale 8 * la taille de la fonction de hachage. En d'autres termes, placer cette taille à 4096 fera que ip_conntrack_max aura 32768 entrées conntrack. Un exemple de ce qui pourrait être :

```
work3:/home/blueflux# modprobe ip_conntrack hashsize=4096
work3:/home/blueflux# cat /proc/sys/net/ipv4/ip_conntrack_max
32768
work3:/home/blueflux#
```

7.3. États de l'espace utilisateur

Comme vous l'avez vu, les paquets peuvent prendre différents états à l'intérieur du noyau, en fonction du protocole considéré. Cependant, à l'extérieur du noyau, seuls 4 états sont disponibles, comme c'est expliqué précédemment. Ces états peuvent être associés à la correspondance state qui est capable de sélectionner les paquets à partir de l'état actuel du traçage de connexion. Les états valides sont ***NEW***, ***ESTABLISHED***, ***RELATED*** et ***INVALID***. Le tableau suivant décrit brièvement chacun d'eux.

Tableau 7.1. États de l'espace utilisateur

État	Explication
NEW	L'état ***NEW*** indique que le paquet est le premier de la connexion. Cela signifie que, pour une connexion donnée, le premier paquet que le module conntrack aperçoit est sélectionné. Par exemple, si on rencontre un paquet SYN et que c'est le premier paquet d'une connexion, on établit la correspondance. Cependant, le paquet peut aussi ne pas être de type SYN et considéré tout de même dans l'état ***NEW***. Ceci peut s'avérer problématique dans certaines situations, mais peut aussi être extrêmement utile quand on doit récupérer les connexions perdues issues d'autres pare−feux, ou quand une connexion a dépassé son temps de survie, mais n'est pas réellement fermée.
ESTABLISHED	L'état ***ESTABLISHED*** résulte de l'observation d'un trafic dans les deux sens, et donc établit une correspondance avec ce type de paquets. Les connexions établies (***ESTABLISHED***) sont particulièrement simple à comprendre. La seule condition pour entrer dans l'état ***ESTABLISHED*** est qu'un hôte ayant envoyé un paquet reçoive plus tard une réponse de l'hôte destinataire de ce paquet. A la réception du paquet de réponse, l'état ***NEW*** est transformé en ***ESTABLISHED***. Les messages d'erreur et de redirection ICMP peuvent aussi être considérés comme ***ESTABLISHED***, si on a généré un paquet qui en retour génère le message ICMP.
RELATED	L'état ***RELATED*** est un des états les plus astucieux. Une connexion est considérée comme ***RELATED*** quand elle est liée à une autre connexion déjà établie, donc dans l'état ***ESTABLISHED***. Ainsi, pour qu'une connexion soit identifiée comme ***RELATED***, on doit tout d'abord disposer d'une autre dans l'état ***ESTABLISHED***. La connexion

	ESTABLISHED crée alors une connexion extérieure à la connexion principale. La nouvelle connexion créée est donc considérée comme ***RELATED***, si le module conntrack l'identifie comme en relation avec l'autre. Les exemples suivants peuvent être considérés comme ***RELATED*** : les connexions `FTP-data` sont liées au port `FTP control`, et les connexions `DCC` interviennent par l'intermédiaire de `IRC`. Ceci permet d'utiliser des réponses `ICMP`, des transferts `FTP` et des `DCC` pour travailler convenablement à travers un pare–feu. Remarquez que la plupart des protocoles `TCP` et certains protocoles `UDP` qui reposent sur ce mécanisme sont particulièrement complexes. Ils envoient des informations de connexion à l'intérieur des données utiles, qui par conséquent requièrent des modules helper spécifiques pour être correctement interprétées.
INVALID	L'état ***INVALID*** signifie que le paquet ne peut pas être identifié ou qu'il n'a aucun état connu. Il peut y avoir plusieurs raisons à cela : par exemple, un système en dépassement de mémoire ou des messages d'erreur `ICMP` ne répondant à aucune connexion connue. Généralement, il est préférable de détruire tout ce qui se trouve dans cet état.

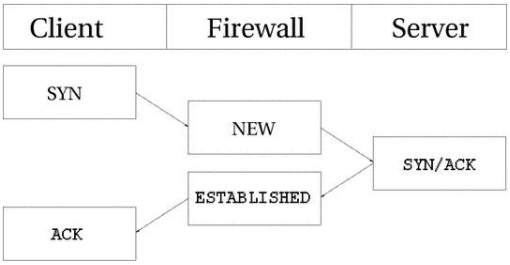
Ces états peuvent être utilisés avec la correspondance ***––state*** pour sélectionner des paquets à partir de leur état de traçage de connexion. C'est ce qui rend la machine d'état si puissante et efficace pour votre pare–feu. Auparavant, nous devions souvent ouvrir tous les ports supérieurs à 1024 pour permettre le trafic inverse à destination de notre réseau local. Avec la machine d'état à l'oeuvre, ce n'est plus nécessaire, puisqu'il est possible d'ouvrir le pare–feu pour les réponses sans l'ouvrir pour le reste du trafic.

7.4. Connexions TCP

Dans cette section et les suivantes, nous examinons davantage les états et comment ils sont gérés pour les trois protocoles élémentaires `TCP`, `UDP` et `ICMP`. Nous verrons aussi comment les états sont gérés par défaut, s'ils ne peuvent être assimilés à un quelconque de ces trois protocoles. Nous choisissons de démarrer avec le protocole `TCP` comme c'est un protocole à état en lui–même, avec nombre de caractéristiques intéressantes en rapport avec la machine d'état d'iptables.

Une connexion `TCP` démarre toujours avec l'établissement d'une liaison en 3 phases, qui met en place et négocie la connexion qui servira pour le transfert des données. Toute la session commence par un paquet `SYN`, suivi d'un paquet `SYN/ACK` et se termine par un paquet `ACK` pour accuser réception de l'établissement de la session. A cet instant, la connexion est établie et prête à envoyer des données. La problème est le suivant : comment le traçage de connexion peut–il s'accrocher à cette étape ? En fait, très simplement.

En ce qui concerne l'utilisateur, le traçage de connexion fonctionne de façon identique pour tout type de connexion. Observez le schéma ci–dessous pour comprendre dans quel état se trouve le flux aux différentes étapes de la connexion. Comme vous le voyez, le code du traçage de connexion ne suit pas vraiment les étapes de la connexion `TCP` du point de vue de l'utilisateur. Dès qu'un paquet `SYN` arrive, il considère la connexion comme nouvelle (***NEW***). Dès qu'un paquet de réponse `SYN/ACK` est observé, il considère la connexion comme établie (***ESTABLISHED***). Si vous réfléchissez une seconde, vous comprendrez pourquoi. Avec cette implémentation particulière, vous pouvez autoriser des paquets ***NEW*** et ***ESTABLISHED*** à quitter votre réseau local, et autoriser seulement des connexions ***ESTABLISHED*** en retour, et ça fonctionnera sans problème. A contrario, si la machine de traçage de connexion considérait l'établissement complet de la connexion comme ***NEW***, il serait impossible d'arrêter les connexions issues de l'extérieur vers le réseau local, puisqu'il faudrait à nouveau autoriser le retour de paquets ***NEW***. Pour rendre les choses encore plus compliquées, il existe de nombreux autres états internes qui sont utilisés pour les connexions `TCP` à l'intérieur du noyau, mais qui ne sont pas disponibles dans l'espace utilisateur. Brièvement, ils respectent les états standards spécifiés dans le document RFC 793 – Transmission Control Protocol aux pages 21–23.



Comme vous pouvez le voir, c'est tout à fait simple, du point de vue de l'utilisateur. Cependant, en regardant la construction complète du point de vue du noyau, c'est un peu plus difficile. Voyons un exemple. Regardons exactement comment les états de connexion changent dans la table `/proc/net/ip_conntrack`. Le premier état est rapporté sur le reçu d'un premier paquet SYN dans la connexion.

```
tcp      6 117 SYN_SENT src='1'92.168.1.5 dst='1'92.168.1.35 sport='1'031 \
     dport='2'3 [UNREPLIED] src='1'92.168.1.35 dst='1'92.168.1.5 sport='2'3 \
     dport='1'031 use='1'
```

Comme nous l'indique l'entrée ci−dessus, nous avons un état précis dans lequel un paquet SYN a été envoyé, (le drapeau `SYN_SENT` est placé), et pour lequel aucune réponse n'a été envoyée (en témoigne le drapeau `[UNREPLIED]`). L'état interne suivant sera joint quand nous verrons un autre paquet dans l'autre direction.

```
tcp      6 57 SYN_RECV src='1'92.168.1.5 dst='1'92.168.1.35 sport='1'031 \
     dport='2'3 src='1'92.168.1.35 dst='1'92.168.1.5 sport='2'3 dport='1'031 \
     use='1'
```

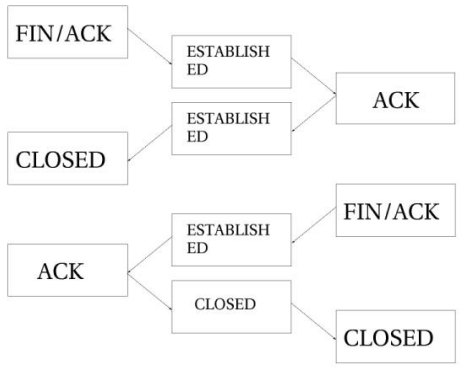
Maintenant nous avons reçu un SYN/ACK correspondant en retour. Dès que ce paquet a été reçu, l'état change encore une fois, cette fois vers `SYN_RECV`. `SYN_RECV` nous indique que le SYN d'origine a été délivré correctement et que le SYN/ACK en retour passe aussi à travers le pare−feu proprement. D'ailleurs, cette entrée de traçage de connexion a vu le trafic dans les deux directions et considère désormais qu'il y a été répondu. Ceci n'est pas explicite, mais nous assumons, comme l'était le drapeau `[UNREPLIED]` au−dessus. L'étape finale sera atteinte lorsque nous aurons vu le ACK dans l'établissement d'une liaison à trois voies.

```
tcp      6 431999 ESTABLISHED src='1'92.168.1.5 dst='1'92.168.1.35 \
     sport='1'031 dport='2'3 src='1'92.168.1.35 dst='1'92.168.1.5 \
     sport='2'3 dport='1'031 [ASSURED] use='1'
```

Dans le dernier exemple, nous avons obtenu le ACK final dans l'établissement d'une liaison à trois voies et la connexion a pris l'état ***ESTABLISHED***, pour autant que les mécanismes internes de Iptables soient avisés. Normalement, le flux de données sera `ASSURED` maintenant.

Une connexion peut aussi être dans l'état ***ESTABLISHED***, mais pas `[ASSURED]`. Ceci survient lorsque nous avons une connexion captée ouverte (nécessite la patch tcp−window−tracking et le ip_conntrack_tcp_loose placé à 1 ou plus haut). Par défaut, sans la patch tcp−window−tracking, c'est ce comportement qui s'applique, et il n'est pas modifiable.

Quand une connexion TCP est close, elle est faite de cette façon et prend les états suivants.



Comme vous pouvez le voir, la connexion n'est jamais réellement fermée jusqu'à ce que le ACK soit envoyé. Notez que cette figure décrit simplement comment elle est fermée dans des circonstances normales. Une connexion peut aussi, par exemple, être fermée par l'envoi d'un RST (reset), si celle−ci a été refusée. Dans ce cas, la connexion sera fermée immédiatement.

Quand la connexion TCP a été fermée, elle entre dans l'état `TIME_WAIT`, qui est, par défaut, de 2 minutes. Ceci est utilisé pour que tous les paquets sortis puissent encore passer à travers notre table de règles, même après que la connexion soit fermée. Ceci est employé comme une sorte de tampon pour que les paquets qui ont été immobilisés dans un ou d'autres routeurs engorgés puissent encore passer le pare−feu, ou vers une autre fin de connexion.

Si la connexion est réinitialisée par un paquet RST, l'état est modifié en `CLOSE`. Ce qui indique que la connexion a, par défaut, 10 secondes avant que la connexion complète soit fermée définitivement. Les paquets RST ne sont pas reconnus dans aucun sens, et couperont la connexion directement. Il existe aussi d'autres états que ceux dont nous avons parlé. Voici une liste complète des états qu'un flux TCP peut prendre, et leurs valeurs de délai d'attente.

Tableau 7.2. États internes

État	Délai
NONE	30 minutes
ESTABLISHED	5 jours
SYN_SENT	2 minutes
SYN_RECV	60 secondes
FIN_WAIT	2 minutes
TIME_WAIT	2 minutes
CLOSE	10 secondes
CLOSE_WAIT	12 heures
LAST_ACK	30 secondes
LISTEN>	2 minutes

Ces valeurs ne sont pas absolues. Elles peuvent changer en fonction des versions du noyau, et également par le système de fichiers dans les variables `/proc/sys/net/ipv4/netfilter/ip_ct_tcp_*`. Les valeurs par défaut seront, cependant, justes en pratique. Ces valeurs sont établies en secondes.

Note

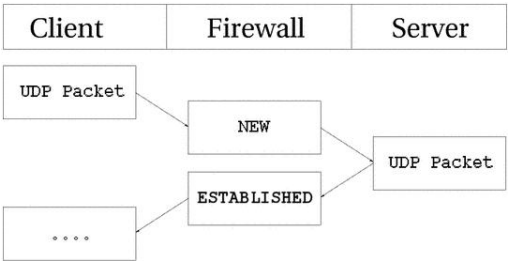
Notez aussi que le domaine utilisateur de la machine d'état ne doit pas voir les drapeaux TCP (i.e., RST, ACK et SYN sont des drapeaux) placés dans les paquets TCP. C'est généralement pas recommandé, car

vous pourriez autoriser les paquets dans l'état ***NEW*** à traverser le pare–feu, mais quand vous spécifiez le drapeau ***NEW***, vous indiquez les paquets SYN.

Ce n'est pas ce qui se produit avec l'implémentation de l'état; ce qui veut dire, même un paquet avec aucun bit de placé ou un drapeau ACK, sera compté comme ***NEW***. Ceci peut être utilisé pour faire de la redondance de pare–feu, mais c'est en général très déconseillé pour un réseau domestique, dans lequel vous avez un seul pare–feu. Pour en savoir plus vous pouvez utiliser la commande expliquée dans la section Paquets état NEW sans bit SYN placé de l'annexe Problèmes et questions courants. Un autre moyen est d'installer l'extension ***tcp–window–tracking*** depuis ***patch–o–matic***, et de placer `/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_loose` à zéro, le pare–feu effacera tous les paquets NEW en ayant le drapeau SYN placé.

7.5. Connexions UDP

Les connexions UDP ne sont pas des connexions à part entière, mais plutôt des connexions "apatrides". Il y a plusieurs raisons pour cela, la principale parce qu'elles ne contiennent aucun établissement ou fermeture de connexion; la plupart n'ont pas de liaison. En recevant deux datagrammes UDP dans un ordre spécifique, on ne peut savoir dans quel ordre ils ont été envoyés. Cependant, il est toujours possible de placer des états sur les connexions dans le noyau. Voyons comment une connexion peut être tracée et à quoi elle ressemblerait dans le conntrack.



Comme on peut le voir, la connexion circule presque exactement de la même façon qu'une connexion TCP. Ceci, du point de vue de l'utilisateur. En interne, l'information conntrack est un peu différente, mais intrinsèquement les détails sont les mêmes. Premièrement, regardons les entrées après que le paquet UDP initial ait été envoyé.

```
udp      17 20 src='1'92.168.1.2 dst='1'92.168.1.5 sport='1'37 dport='1'025 \
     [UNREPLIED] src='1'92.168.1.5 dst='1'92.168.1.2 sport='1'025 \
     dport='1'37 use='1'
```

Comme vous pouvez le voir entre la première et deuxième valeur, c'est un paquet UDP. Le premier est le nom du protocole, le second le numéro du protocole. C'est identique pour une connexion TCP. La troisième valeur indique combien de secondes cet état s'est maintenu. Après cela, nous avons les valeurs du paquet que nous avons vu et les probabilités que les paquets sur cette connexion nous joignent depuis l'expéditeur. Ce sont la source, la destination, le port source et le port destination. À ce point le drapeau `[UNREPLIED]` nous indique qu'il n'y a pas eu de réponse au paquet. Enfin, nous avons une courte liste de probabilités de retour de paquets. Notez que les dernières entrées sont en ordre inverse des premières valeurs. Le délai d'attente est de 30 secondes dans ce cas, par défaut.

```
udp      17 170 src='1'92.168.1.2 dst='1'92.168.1.5 sport='1'37 \
     dport='1'025 src='1'92.168.1.5 dst='1'92.168.1.2 sport='1'025 \
     dport='1'37 [ASSURED] use='1'
```

À ce point le serveur a vu une réponse au premier paquet envoyé et la connexion est maintenant considérée comme ***ESTABLISHED***. Ce n'est pas indiqué dans le traçage de connexion comme vous pouvez le voir. La principale différence est que le drapeau `[UNREPLIED]` est maintenant fourni. D'ailleurs, le temps d'attente par défaut a changé pour passer à 180 secondes – mais dans cet exemple il a été décrémenté pour passer à 170

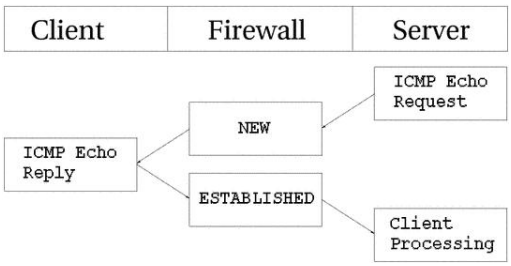
secondes – avec un délai de 10 secondes, il sera de 160 secondes. Une chose qui a été oubliée et qui peut changer légèrement est le drapeau [ASSURED] décrit ci–dessus. Pour placer le fanion [ASSURED] sur un traçage de connexion, il doit y avoir un paquet en réponse au paquet NEW.

```
udp      17 175 src='1'92.168.1.5 dst='1'95.22.79.2 sport='1'025 \
     dport=53 src='1'95.22.79.2 dst='1'92.168.1.5 sport=53 \
     dport='1'025 [ASSURED] use='1'
```

Ici, la connexion est assurée. La connexion a exactement le même aspect que dans l'exemple précédent. Si cette connexion n'est pas utilisée dans les 180 secondes, elle est hors délai. 180 secondes est comparativement une valeur basse, mais qui devrait être suffisante dans la plupart des cas. Cette valeur est réinitialisée complètement pour chaque paquet qui correspond à la même entrée et passe à travers la pare–feu, comme pour tous les états internes.

7.6. Connexions ICMP

Les paquets ICMP sont loin d'être un flux véritable, car ils sont seulement utilisés pour le contrôle et n'établissent jamais aucun type de connexion. Il existe quatre types ICMP qui généreront cependant des paquets en retour, et qui possèdent deux états différents. Ces messages ICMP peuvent prendre les états ***NEW*** et ***ESTABLISHED***. Les types ICMP dont nous parlons sont les requêtes et réponse Echo, les requêtes et réponses Timestamp, les requêtes et réponses Information, et enfin les requêtes et réponses de masque d'Adresse. En dehors de ça, les requêtes d'horodatage et d'information sont obsolètes et seront probablement effacées. Cependant, les messages Echo sont utilisés dans plusieurs cas comme le ping sur des hôtes. Les requêtes de masque d'adresse ne sont pas utilisées souvent, mais peuvent être utiles quelques fois. Pour en avoir une idée, voyons l'image suivante.



Comme vous pouvez le voir dans l'image ci–dessus, l'hôte envoie une requête écho vers la cible, laquelle est considérée comme ***NEW*** par le pare–feu. La cible répond alors avec un écho que le pare–feu considère comme état ***ESTABLISHED***. Quand la première requête écho a été vue, les entrées état suivantes se dirigent dans le ip_conntrack.

```
icmp     1 25 src='1'92.168.1.6 dst='1'92.168.1.10 type=8 code=0 \
     id=33029 [UNREPLIED] src='1'92.168.1.10 dst='1'92.168.1.6 \
     type=0 code=0 id=33029 use='1'
```

Ces entrées semblent un peu différentes des états standards pour TCP et UDP. Le protocole est présent, et la temporisation, de même que les adresses source et destination. Le problème survient après. Nous avons maintenant trois nouveaux champs appelés type, code et id. Il n'y a rien de spécial, le champ type contient le type ICMP et le champ code contient le code ICMP. Toutes les variables sont présentées dans l'annexe Types ICMP. Le dernier champ id, contient l'ID ICMP. Chaque paquet ICMP obtient un ID quand il est envoyé, et lorsque le destinataire reçoit le message ICMP, il place le même ID dans le nouveau message ICMP, que l'expéditeur reconnaîtra dans la réponse et pourra se connecter avec la requête ICMP correcte.

Dans le champ suivant, nous avons reconnu le drapeau [UNREPLIED]. Ce fanion nous indique que nous observons une entrée de traçage de connexion, dont le trafic est dans une direction. Enfin, nous voyons la probabilité de réponse à un paquet ICMP, qui est l'inversion des adresses IP source et destination. Comme

pour le type et le code, ils ont été modifiés en valeurs correctes pour le paquet de retour, ainsi une requête écho est changée en réponse écho et ainsi de suite. L'ID ICMP est préservé depuis le paquet requête.

Le paquet réponse est considéré comme ***ESTABLISHED***, ainsi que nous l'avons déjà expliqué. Cependant, nous pouvons savoir avec certitude qu'après la réponse ICMP, il n'y aura plus de trafic régulier dans la même connexion. Pour cette raison, l'entrée de traçage de connexion est détruite une fois que la réponse soit passée à travers la structure de Netfilter.

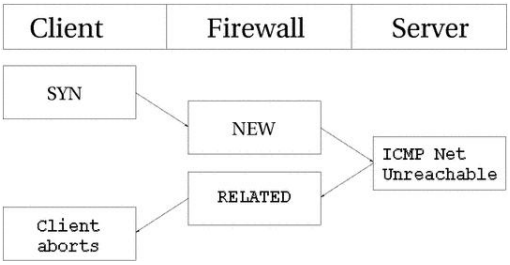
Dans chacun des cas ci−dessus, la requête est considérée comme ***NEW***, tandis que la réponse est considérée comme ***ESTABLISHED***. Voyons cela de plus près. Quand le pare−feu voit un paquet requête, il le considère comme ***NEW***. Quand l'hôte envoit un paquet en réponse à la requête il est considéré comme ***ESTABLISHED***.

Note

Notez que cela indique que le paquet réponse doit apparier le critère donné par l'entrée de traçage de connexion pour être considéré comme établi, de la même façon que tous les autres types de trafic.

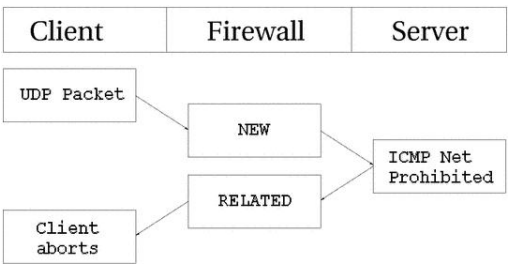
Les requêtes ICMP ont un délai par défaut de 30 secondes, que vous pouvez modifier dans l'entrée `/proc/sys/net/ipv4/netfilter/ip_ct_icmp_timeout`. C'est, en général, une bonne valeur de temps d'attente, car elle permet de capturer la plupart des paquets en transit.

Une autre partie très importante de ICMP est le fait qu'il est utilisé pour indiquer aux hôtes les événements au niveau des connexions spécifiques UDP et TCP ou des tentatives de connexions. Pour cette raison, les réponses ICMP seront très souvent reconnues comme ***RELATED***. Un exemple simple est le `ICMP Host unreachable` ou `ICMP Network unreachable`. Ceci est toujours généré en retour vers notre hôte si il tente une connexion vers quelque autre hôte, mais que le réseau ou l'hôte en question ne soit pas connecté, ainsi le dernier routeur essayant de joindre le site répondra par un message ICMP nous l'indiquant. Dans ce cas, la réponse ICMP est considérée comme un paquet ***RELATED***. L'image suivante explique ceci.



Dans l'exemple ci−dessus, nous envoyons un paquet SYN vers une adresse spécifique. Ceci est considéré comme une connexion ***NEW*** par le pare−feu. Cependant, le réseau que le paquet essaie de joindre est injoignable, donc un routeur va nous renvoyer une erreur ICMP indiquant que le réseau est injoignable. Le code de traçage de connexion peut reconnaître ce paquet comme ***RELATED***, ainsi la réponse ICMP est correctement envoyée au client. En attendant, le pare−feu a détruit l'entrée de traçage de connexion depuis qu'il a eu connaissance du message d'erreur.

Le même comportement que ci−dessus est observé avec les connexions UDP avec des problèmes identiques que précédemment. Tous les messages ICMP envoyés en réponse aux connexions UDP sont considérés comme ***RELATED***. Voyons l'image suivante.



Un paquet UDP est envoyé à un hôte. Cette connexion UDP est considérée comme ***NEW***. Cependant, le

réseau est interdit administrativement par un pare−feu quelconque ou un routeur. Donc, notre pare−feu reçoit un "ICMP Network Prohibited" en retour. Le pare−feu sait que ce message d'erreur ICMP est en relation avec la connexion UDP déjà ouverte et envoit un paquet ***RELATED*** au client. À ce niveau, le pare−feu détruit les entrées de traçage de connexion, et le client reçoit le message ICMP.

7.7. Connexions par défaut

Dans certains cas, la machine conntrack ne sait pas comment traiter un protocole spécifique. Ceci se produit lorsqu'elle ne connait pas le protocole en particulier, ou ne sait pas comment il fonctionne. Dans ces cas là, elle utilise un comportement par défaut. Ce comportement est utilisé sur, par exemple, NETBLT, MUX et EGP. Ce comportement ressemble au traçage de connexion UDP. Le premier paquet est considéré comme ***NEW***, et le trafic en réponse comme ***ESTABLISHED***.

Quand ce comportement par défaut est utilisé, tous les paquets ont la même valeur de délai par défaut. Ceci peut être paramétré via la variable `/proc/sys/net/ipv4/netfilter/ip_ct_generic_timeout`. La valeur par défaut est ici de 600 secondes, soit 10 minutes. En fonction du trafic, ceci peut avoir besoin d'être modifié. Spécialement si vous renvoyez le trafic par satellite, ce qui peut prendre un long moment.

7.8. Protocoles complexes et traçage de connexion

Certains protocoles sont plus complexes que d'autres. Ce qui fait que, lorsqu'ils interfèrent avec du traçage de connexion ils peuvent être plus difficiles à tracer correctement. De bons exemples en sont les protocoles ICQ, IRC et FTP. Chacun des ces protocoles transporte l'information dans les données utiles des paquets, et donc nécessitent un traçage de connexion spécial pour fonctionner correctement.

Voici une liste des protocoles complexes supportés par le noyau Linux, et la version du noyau dans laquelle il a été implémenté.

Tableau 7.3. Support des protocoles complexes

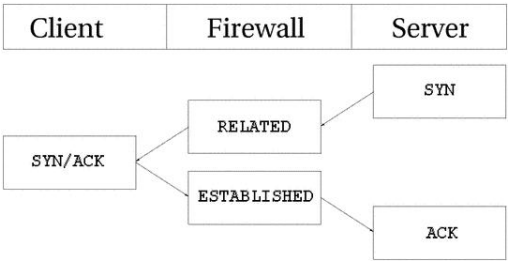
Nom du protocole	Version du noyau
FTP	2.3
IRC	2.3
TFTP	2.5
Amanda	2.5

- FTP
- IRC
- TFTP

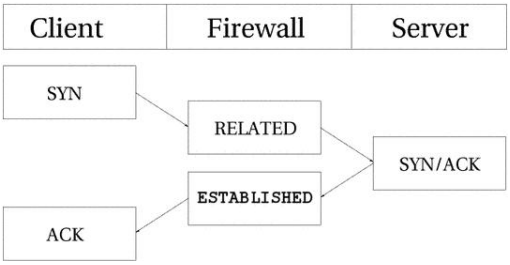
Prenons le protocole FTP comme premier exemple. Ce protocole ouvre en premier une seule connexion appelée contrôle de session FTP. Quand nous lançons des commandes dans cette session, d'autres ports sont ouverts pour transporter le reste des données relatives à ces commandes spécifiques. Ces connexions peuvent être réalisées de deux manières, de façon active ou passive. Quand une connexion est en mode actif, le client FTP envoie au serveur un port et une adresse IP pour se connecter. Après cela, le client FTP ouvre le port et le serveur, le connecte depuis un domaine de ports non privilégiés (>1024) et envoie les données sur celui−ci.

Le problème est que le pare−feu ne connaît pas ces connexions, car elles ont été négociées dans les données utiles du protocole. À cause de ça, le pare−feu sera incapable de savoir s'il doit laisser le serveur connecté au client sur ces ports spécifiques.

La solution est d'ajouter une aide spéciale au module de traçage de connexion qui scannera les données dans le contrôle de connexion à la recherche d'information et de syntaxes spécifiques. Quand il fonctionne avec l'information correcte, il indique cette information spécifique comme ***RELATED*** et le serveur sera capable de tracer la connexion. Voyons l'image suivante pour comprendre les états quand le serveur FTP a envoyé le retour de connexion au client.



Le FTP passif fonctionne de la façon inverse. Le client FTP indique au serveur qu'il désire certaines données spécifiques, le serveur lui répond avec l'adresse IP à connecter et le port. Le client pourra, au reçu de ces données, se connecter à ce port depuis son propre port 20 (port FTP), et obtenir les données en question. Si vous avez un serveur FTP derrière votre pare−feu, vous aurez besoin de ce module en supplément de vos modules standards Iptables pour permettre aux clients sur l'Internet de se connecter au serveur FTP correctement. C'est la même chose si vous êtes très restrictif pour vos utilisateurs, et les laissez se connecter seulement aux serveurs HTTP et FTP sur l'Internet, et que vous bloquez tous les autres ports. Regardons l'image suivante à propos du FTP passif.



Certains assistants conntrack sont déjà disponibles dans le noyau lui−même. Plus spécifiquement, les protocoles FTP et IRC possèdent des assistants. Si vous ne trouvez pas les assistants conntrack dont vous avez besoin dans le noyau, vous devriez regarder l'arborescence patch−o−matic dans le domaine utilisateur Iptables. L'arborescence patch−o−matic peut contenir d'avantage d'assistants conntrack, comme ceux des protocoles ntalk ou H.323. S'ils ne sont pas disponibles dans l'arborescence patch−o−matic, vous avez plusieurs options. Soit vous cherchez dans le source CVS d'Iptables, si il a été récemment modifié, soit vous contactez la liste de diffusion Netfilter−devel et demandez leurs disponibilités. Sinon, et s'il n'y a pas de projet en vue, lisez le Rusty Russell's Unreliable Netfilter Hacking HOW−TO dont le lien est dans Autres ressources et liens.

Les assistants conntrack peuvent être soit compilés statiquement dans le noyau, soit en modules. S'ils sont compilés en modules, vous pouvez les charger avec les commandes suivantes :

```
modprobe ip_conntrack_ftp
modprobe ip_conntrack_irc
modprobe ip_conntrack_tftp
modprobe ip_conntrack_amanda
```

Notez que la traçage de connexion n'a rien à voir avec le NAT, et donc vous pouvez avoir besoin de d'avantage de modules si vous NATez les connexions. Par exemple, si vous voulez NATer et tracer les connexions FTP, vous aurez besoin du module NAT. Tous les assistants NAT débutent avec ip_nat_ et suivant; exemple, l'assistant FTP NAT sera nommé ip_nat_ftp et le module IRC nommé ip_nat_irc. Les assistants conntrack suivent la même convention, et donc l'assistant IRC conntrack s'appellera

ip_conntrack_irc, alors que l'assistant FTP conntrack sera nommé ip_conntrack_ftp. Comme dans la sortie d'écran ci−dessus.

Chapitre 8. Sauvegarde et restauration des tables de règles importantes

Le paquetage ***iptables*** est fourni avec deux outils très utiles, spécialement si vous utilisez des tables de règles importantes. Ces deux outils sont appelés ***iptables−save*** et ***iptables−restore*** et sont utilisés pour la sauvegarde et la restauration des tables de règles dans un format spécifique qui semble un peu différent du code shell standard que nous avons vu dans ce didacticiel.

8.1. Considérations de vitesse

Une des plus importantes raisons d'utiliser ***iptables−save*** et ***iptables−restore*** est qu'elles améliorent la vitesse de chargement et de restauration des tables de règles importantes. Le problème principal lors du lancement de scripts shell contenant des règles ***iptables*** est que chaque invocation d'***iptables*** dans le script extraira l'ensemble des règles de l'espace Netfilter du noyau, et après cela, inserrera les règles, ou toute autre action en réponse à la commande spécifique. Enfin, ajoutera les nouvelles règles issues de sa propre mémoire dans l'espace noyau. L'utilisation d'un script shell, créé pour chacune des règles que vous voulez insérer, prend plus de temps pour l'extraction et l'insertion de la table de règles.

Pour résoudre ce problème, il existe les commandes ***iptables−save*** et ***restore***. La commande ***iptables−save*** est utilisée pour sauvegarder la table de règles dans un fichier texte au format spécial, et la commande ***iptables−restore*** est utilisée pour charger ce fichier à nouveau dans le noyau. Le plus intéressant de ces commandes est qu'elles chargent et sauvegardent la table de règles en une seule action. ***iptables−save*** récupérera la table de règles du noyau et la sauvegardera dans un fichier en une seule action. ***iptables−restore*** enverra la table de règles au noyau en une seule action pour chaque table. En d'autres termes, au lieu d'effacer la table de règles du noyau quelques 30 000 fois, ce qui arrive pour les tables de règles importantes, et ensuite la renvoyer au noyau, nous pouvons maintenant sauvegarder cet ensemble dans un fichier en une ou deux actions.

Comme vous pouvez le comprendre, ces outils sont définitivement faits pour vous si vous utilisez un vaste ensemble de règles. Cependant, elles ont des inconvénients, comme nous le verrons dans la section suivante.

8.2. Inconvénients avec restore

Comme vous avez pû vous en étonner, ***iptables−restore*** fonctionne−t−il avec toutes sortes de scripts ? De loin, non, il ne pourra probablement jamais le faire. C'est le principal défaut d' ***iptables−restore*** car il n'est pas capable de faire beaucoup de choses avec ces fichiers. Par exemple, si vous avez une connexion qui utilise une IP dynamique et que vous voulez récupérer cette IP à chaque démarrage de la machine et ensuite insérer cette valeur dans vos scripts ? Avec ***iptables−restore***, c'est plus ou moins impossible.

Une possibilté pour faire ceci est de créer un petit script qui récupère les valeurs que vous voulez utiliser, ensuite faire un sed du fichier ***iptables−restore*** pour ces mots−clé spécifiques et les remplacer avec les valeurs collectées via le petit script. À ce point, vous pouvez les sauvegarder dans un fichier temporaire, et ensuite utiliser ***iptables−restore*** pour charger ces nouvelles valeurs. Ceci provoque cependant certains problèmes, et vous serez incapables de vous servir de ***iptables−save*** correctement car il effacera probablement vos mots−clé ajoutés à la main dans le script de restauration. C'est une solution maladroite.

Une autre solution est de charger les scripts ***iptables−restore*** en premier, et ensuite charger les scripts qui insèrent les règles plus dynamiques à leur place. Bien sûr, comme vous pouvez le comprendre, c'est juste une reprise maladroite de la première solution. ***iptables−restore*** n'est tout simplement pas très bien adapté pour les configurations dans lesquelles les adresses IP sont assignées dynamiquement.

Un autre inconvénient avec ***iptables–restore*** et ***iptables–save*** est qu'il ne sont pas aussi complètement fonctionnels que si vous faites un script. Le problème est simplement que pas beaucoup de personnes l'utilisent et donc qu'il n'y a pas énormément de rapports de bugs. Même si ces problèmes existent, je vous recommande fortement d'utiliser ces outils qui fonctionnent très bien pour la plupart des tables de règles tant qu'elles ne contiennent pas certains nouveaux modules ou cibles qu'ils ne savent pas gérer correctement.

8.3. iptables–save

La commande ***iptables–save*** est, comme nous l'avons déjà expliqué, un outil pour sauvegarder dans la table de règles un fichier que ***iptables–restore*** peut utiliser. Cette commande est tout à fait simple, et prend seulement deux arguments. Regardons l'exemple suivant pour comprendre la syntaxe :

`iptables-save` [–c] [–t *`table`*]

L'argument ***–c*** indique à ***iptables–save*** de conserver les valeurs spécifiées dans les compteurs de bits et de paquets. Ce qui pourrait être utile si vous voulez redémarrer votre pare–feu principal, mais sans perdre les compteurs de bits et de paquets que nous pourrions utiliser dans un but de statistiques. Exécuter une commande ***iptables–save*** avec l'argument ***–c*** nous permet de redémarrer sans briser les routines de statistique et de comptage. La valeur par défaut est, bien sûr, de ne pas garder les compteurs intacts quand cette commande est exécutée.

L'argument ***–t*** indique à la commande ***iptables–save*** quelle table sauvegarder. Sans cet argument toutes les tables disponibles dans le fichier seront automatiquement sauvegardées. Ci–dessous, un exemple de ce que donne une commande ***iptables–save*** sans avoir chargé de table de règles.

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*filter
:INPUT ACCEPT [404:19766]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [530:43376]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*mangle
:PREROUTING ACCEPT [451:22060]
:INPUT ACCEPT [451:22060]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [594:47151]
:POSTROUTING ACCEPT [594:47151]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [3:450]
:OUTPUT ACCEPT [3:450]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
```

Les commentaires débutent avec la signe #. Chaque table est marquée par *<table–name>, par exemple, `*mangle`. Dans chaque table nous avons les spécifications de chaînes et les règles. Une spécification de chaîne ressemble à : <chain–name> <chain–policy> [<packet–counter>:<byte–counter>]. Le `chain-name` peut être, par exemple, `PREROUTING`, la règle d'action est décrite avant et peut être, par exemple, `ACCEPT`. Enfin les compteurs d'octets et de paquets sont les mêmes que dans la sortie de la commande ***iptables –L –v***. Chaque déclaration de table se termine avec un mot–clé `COMMIT`. Le mot–clé `COMMIT` indique qu'à ce niveau toutes les règles seront envoyées au noyau par l'opérateur de transfert de données.

L'exemple ci–dessus est tout à fait basique, et je crois qu'il est approprié de montrer un bref exemple qui contient un petit Iptables–save. Si nous voulons lancer ***iptables–save*** sur celui–ci, la sortie de la commande sera :

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*filter
:INPUT DROP [1:229]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth1 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*mangle
:PREROUTING ACCEPT [658:32445]
:INPUT ACCEPT [658:32445]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [891:68234]
:POSTROUTING ACCEPT [891:68234]
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*nat
:PREROUTING ACCEPT [1:229]
:POSTROUTING ACCEPT [3:450]
:OUTPUT ACCEPT [3:450]
-A POSTROUTING -o eth0 -j SNAT --to-source 195.233.192.1
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
```

Comme on peut le voir, chaque commande a été préfixée avec les compteurs d'octets et de paquets car nous avons utilisé l'argument ***–c***. Excepté pour ceci, la ligne de commande est tout à fait identique au script. Le seul problème, est de savoir comment sauvegarder la sortie dans un fichier. Vraiment simple, et vous devriez savoir le faire si vous avez utilisé Linux auparavant. Il suffit d'utiliser un "pipe" (canal de communication) pour enregistrer la sortie de la commande dans le fichier. Ça ressemblera à cela :

iptables–save –c > /etc/iptables–save

La commande ci–dessus fera une sauvegarde de toute la table de règles appelée `/etc/iptables-save` avec les compteurs d'octets et de paquets toujours intacts.

8.4. iptables–restore

La commande ***iptables–restore*** est exécutée pour restaurer la table de règles de ***iptables*** qui a été sauvegardée par la commande ***iptables–save***. Elle prend toutes les entrées standard mais ne peut faire de restauration depuis un fichier de règles écrit à la main (script), malheureusement. La syntaxe de cette commande :

`iptables-restore` [–c] [–n]

L'argument ***–c*** restaure les compteurs d'octets et de paquets et doit être utilisé si vous voulez garder les compteurs précédemment enregistrés avec ***iptables–save***. Cet argument peut aussi s'écrire avec sa forme de nom long ***––counters***.

L'argument ***–n*** indique à ***iptables–restore*** de ne pas écraser les règles précédemment écrites dans la table, ou les tables. Le comportement par défaut de ***iptables–restore*** est d'effacer et supprimer toutes les règles inscrites auparavant. L'argument court ***–n*** peut être remplacé par son format long ***––noflush***.

Pour charger une table de règles avec la commande ***iptables−restore***, il existe plusieurs solutions, mais nous ne verrons que la plus simple et la plus commune :

```
cat /etc/iptables-save | iptables-restore -c
```

Ceci fonctionnera également :

```
iptables-restore -c < /etc/iptables-save
```

Ceci concaténera la table de règles située dans le fichier `/etc/iptables-save` et ensuite l'enverra vers ***iptables−restore*** qui récupérera cette table de règles sur l'entrée standard et la restaurera, en incluant les compteurs d'octets et de paquets. Cette commande peut varier à l'infini et nous pourrions montrer les diverses possibilités de "piping", cependant, c'est un peu hors du sujet de ce chapitre, et nous laisserons ceci comme exercice pour le lecteur.

La table de règles devrait maintenant être chargée correctement dans le noyau et fonctionnelle. Sinon, vous avez peut être fait une erreur dans ces commandes.

Chapitre 9. Création d'une règle

Ce chapitre présentera en détail comment créer vos propres règles. Une règle peut être décrite comme une directive au pare−feu et donc, son comportement au niveau du blocage ou de de l'autorisation des différentes connexions et paquets dans une chaîne spécifique. Chaque ligne que vous écrivez est insérée dans une chaîne qui sera considérée comme une règle. Nous verrons également les modules de base disponibles, comment les utiliser, de même que les diverses cibles et la façon de créer de nouvelles cibles (i.e., nouvelles sous−chaînes).

Ce chapitre montrera les lignes de base, comment une règle est créée, comment vous devez l'écrire de façon à ce qu'elle soit acceptée par le programme domaine utilisateur de ***iptables***, les différentes tables, de même que les commandes à exécuter. Après ça nous verrons dans le chapitre suivant tous les modules disponibles pour ***iptables***, et en détail chaque type de cible et de saut.

9.1. Bases de la commande iptables

Comme ceci a déjà été evoqué, chaque règle est une ligne lue par le noyau pour en déduire ce qu'il convient de faire d'un paquet. Si tous les critères − ou les correspondances − sont remplis, on exécute l'instruction donnée par la cible − ou le saut. Normalement, on écrit les règles dans une syntaxe qui ressemble à celle−ci :

`iptables` [−t *`table`*] commande [correspondance] [cible/saut]

Rien ne vous oblige à mettre l'instruction de cible en fin de la ligne. Toutefois, vous devriez préférer cette syntaxe qui améliore la lisibilité. En tout cas, la plupart des règles que vous découvrirez sont écrites de cette façon. Ainsi, si vous lisez le script de quelqu'un d'autre, vous reconnaitrez très probablement la syntaxe et comprendrez plus facilement la règle.

Si vous voulez utiliser une autre table que la table standard, insérer la spécification de table à la place de la mention [table]. Cependant, il n'est pas indispensable de déclarer explicitement la table à utiliser, puisqu'***iptables*** utilise par défaut la table `filter` sur laquelle sont implémentées toutes les commandes. Vous n'êtes pas non plus obligé de spécifier la table à cet endroit dans la règle. Elle peut tout aussi bien être placée ailleurs dans la ligne. Malgré tout, il est plus ou moins habituel de placer la spécification de table au début.

Il y a une chose à garder à l'esprit : la commande devrait toujours être en première position, ou à la rigueur juste après la spécification de table. La "commande" indique au programme ce qu'il doit faire, par exemple

insérer une règle, ajouter une règle en fin de chaîne, ou encore supprimer une règle. Tout ceci est approfondi ultérieurement.

La correspondance est la partie de la règle qui est envoyée au noyau pour identifier la caractéristique particulière du paquet, c'est−à−dire ce qui le distingue de tous les autres paquets. On peut donc spécifier l'adresse IP dont provient le paquet, de quelle interface réseau, l'adresse IP à atteindre, un port, un protocole ou quoi que ce soit d'autre. Il existe un éventail de correspondances que l'on peut utiliser et qui sont développées au cours de ce chapitre.

Enfin, on trouve la cible du paquet. Si toutes les correspondances sont satisfaites pour un paquet, on informe le noyau de l'action à accomplir. Par exemple, vous pouvez stipuler au noyau d'envoyer le paquet à une autre chaîne créée par vous−même, et qui appartient à cette table. Vous pouvez aussi notifier au noyau de détruire le paquet et de ne faire aucun autre traitement, ou encore d'envoyer une réponse particulière à l'expéditeur. Comme pour les autres aspects de cette section, la cible est analysée en profondeur plus loin dans ce chapitre.

9.2. Les tables

L'option ***−t*** précise la table à utiliser. Par défaut, il s'agit de la table `filter`. On peut spécifier une des tables suivantes avec l'option ***−t***. Remarquez que c'est une présentation extrêmement rapide de contenu du chapitre Traversée des tables et des chaînes.

Tableau 9.1. Les tables

Table	Explication
nat	La table `nat` sert principalement à faire de la `traduction d'adresse réseau`. Les paquets soumis au NAT voient leur adresse modifiée, en accord avec les règles concernées. Les paquets d'un flux ne traversent cette table qu'une seule fois. En effet, le sort du premier paquet d'un flux conditionne celui des autres. Si le premier paquet est accepté, les autres paquets du flux sont soumis automatiquement au `NAT` (ou au camouflage, etc.), donc subissent les mêmes actions que le premier paquet. Par conséquent, ils ne passeront pas par cette table, mais seront néanmoins traités de la même façon que le premier paquet du flux. C'est pour cette raison qu'on déconseille le filtrage dans cette table. La chaîne `PREROUTING` permet de modifier les paquets dès qu'ils entrent dans le pare−feu. La chaîne `OUTPUT` permet de modifier les paquets générés localement (c−à−d. dans le pare−feu) avant qu'ils n'accèdent à la décision de routage. En dernier lieu, la chaîne `POSTROUTING` offre la possibilité de modifier les paquets juste avant qu'ils ne quittent le pare−feu.
mangle	Cette table sert à transformer les paquets. Entre autres, on peut modifier le contenu de differents paquets et celui de leurs en−têtes. Par exemple, on peut changer les champs ***TTL***, ***TOS*** ou ***MARK***. Notez que le champ ***MARK*** n'est pas à proprement parlé un changement sur le paquet, mais une valeur de marquage définie dans l'espace du noyau. D'autres règles ou programmes peuvent s'appuyer sur ce marquage à l'intérieur du pare−feu pour filtrer ou opérer un routage évolué, tc en est un exemple. Cette table est constituée de cinq chaînes pré−définies, qui sont nommées `PREROUTING`, `POSTROUTING`, `OUTPUT`, `INPUT` et `FORWARD`. La chaîne `PREROUTING` permet de modifier les paquets juste quand ils entrent dans le pare−feu mais avant qu'ils n'atteignent la décision de routage. La chaîne `POSTROUTING` permet de modifier les paquets juste après toutes les décisions de routage. La chaîne `OUTPUT` s'utilise pour transformer les paquets générés localement avant qu'ils ne sollicitent la décision de routage. La chaîne `INPUT` permet de modifier les paquets une fois qu'ils ont été routés vers la machine locale, mais avant que l'application de l'espace utilisateur n'ait réceptionné les données. La chaîne `FORWARD` permet de modifier les paquets après la première décision de routage mais avant la dernière. Notez que la table mangle ne peut en aucun cas servir à une forme de `traduction d'adresse réseau` ou de `camouflage`, la table nat a été conçue pour ce genre d'opérations.

filter	La table `filter` devrait être exclusivement consacrée à filtrer les paquets. Par exemple, elle permet de détruire (***DROP***), journaliser (***LOG***), accepter (***ACCEPT***) ou rejeter (***REJECT***) des paquets sans aucun problème, de la même manière que dans les autres tables. Cette table est constituée de trois chaînes pré−définies. La première se nomme `FORWARD` et s'applique à tous les paquets qui ne sont ni générés localement, ni destinés à l'hôte local (c−à−d. le pare−feu). La chaîne `INPUT` s'applique à tous les paquets destinés à l'hôte local (le pare−feu), et au final, `OUTPUT` s'applique à tous les paquets générés localement.

Les explications précédentes devraient avoir fourni des notions fondamentales concernant les trois tables disponibles. Celles−ci s'emploient dans des situations complètement différentes, et vous devez maîtriser l'utilisation des différentes chaînes qui composent ces tables. Si vous ne comprenez pas leur utilisation, vous pourriez créer une faille dans votre pare−feu, dans laquelle quelqu'un peut s'engouffrer et vous nuire s'il la découvre. Les tables et chaînes indispensables ont déjà été présentées en détails dans le chapitre Traversée des tables et des chaînes. Si vous ne l'avez pas parfaitement assimilé, je vous conseille de vous y replonger.

9.3. Commandes

Cette section expose les différentes commandes et ce qu'elles permettent de réaliser. La commande signifie à ***iptables*** ce qu'il faut faire du reste de la règle qui est envoyée à l'interpréteur. Typiquement, il s'agit soit d'ajouter, soit d'effacer quelque−chose dans une table quelconque. Les commandes suivantes sont disponibles pour ***iptables*** :

Tableau 9.2. Commandes

Commande	***−A***, ***−−append***
Exemple	***iptables −A INPUT ...***
Explication	Cette commande ajoute une règle à la fin d'une chaîne. La règle sera donc placée en dernière position dans la table de règles, et par conséquent vérifiée en dernier, sauf si vous ajoutez par la suite des règles supplémentaires.
Commande	***−D***, ***−−delete***
Exemple	***iptables −D INPUT −−dport 80 −j DROP***, ***iptables −D INPUT 1***
Explication	Cette commande supprime une règle dans une chaîne. Il existe deux moyens de le faire ; soit en précisant la règle complète (comme dans le premier exemple), soit en indiquant le numéro de la règle que vous visez. Si vous optez pour la première méthode, votre règle doit correspondre exactement avec celle présente dans la chaîne. Avec la seconde méthode, vous devez pointer le numéro de la règle à effacer. Les règles sont numérotées à partir de 1 en commençant au début de chaque chaîne.
Commande	***−R***, ***−−replace***
Exemple	***iptables −R INPUT 1 −s 192.168.0.1 −j DROP***
Explication	Cette commande remplace la règle présente à la ligne indiquée. Elle fonctionne comme la commande ***−−delete***, mais au lieu de supprimer complètement la règle, elle la remplace par une nouvelle. Cette commande est particulièrement commode dans une phase d'expérimentation d'***iptables***.
Commande	***−I***, ***−−insert***
Exemple	***iptables −I INPUT 1 −−dport 80 −j ACCEPT***
Explication	Cette commande insère une règle quelque−part dans une chaîne. La règle est insérée à l'emplacement donné par le numéro spécifié. En l'occurence, l'exemple précédent insère dans la chaîne `INPUT` la règle numéro 1, qui devient ainsi la toute première règle de la chaîne.

Commande	***–L***, ***––list***
Exemple	***iptables –L INPUT***
Explication	Cette commande dresse la liste des entrées de la chaîne donnée. Dans l'exemple précédent, on liste toutes les règles de la chaîne `INPUT`. Il est aussi possible de ne spécifier aucune chaîne. Dans ce cas, la commande listera toutes les chaînes de la table spécifiée (pour spécifier la table, voir la section Les tables). La sortie exacte dépend des autres options envoyées à l'interpréteur, par exemple les options ***–n*** et ***–v***, etc.
Commande	***–F***, ***––flush***
Exemple	***iptables –F INPUT***
Explication	Cette commande vide la chaîne donnée de toutes ses règles. Elle équivaut à effacer les règles une à une, mais se révèle un peu plus rapide. Appelée sans option, cette commande revient à supprimer toutes les règles de toutes les chaînes dans la table spécifiée.
Commande	***–Z***, ***––zero***
Exemple	***iptables –Z INPUT***
Explication	Cette commande permet de mettre à zéro tous les compteurs dans une chaîne spécifique ou dans toutes les chaînes. Si vous utilisez l'option ***–v*** avec la commande ***–L***, vous afficherez le compteur de paquets au début de chaque champ. Pour mettre à zéro le compteur de paquets, utilisez l'option ***–Z***. Elle fonctionne de la même façon que ***–L***, sauf que ***–Z*** ne liste pas les règles. Si ***–L*** et ***–Z*** sont utilisées ensemble (ce qui est autorisé), les chaînes sont dans un premier temps listées, puis les compteurs de paquets sont mis à zéro.
Commande	***–N***, ***––new–chain***
Exemple	***iptables –N allowed***
Explication	Cette commande stipule au noyau de créer une nouvelle chaîne avec le nom indiqué dans la table spécifiée. Dans l'exemple ci–dessus, on crée une chaîne nommée ***allowed***. Notez qu'aucune chaîne ou cible de ce nom ne doit préalablement exister.
Commande	***–X***, ***––delete–chain***
Exemple	***iptables –X allowed***
Explication	Cette commande efface de la table la chaîne spécifiée. Pour que cette commande fonctionne, il ne doit subsister aucune règle faisant référence à la chaîne à effacer. Autrement dit, vous devez remplacer ou supprimer toutes les règles qui pourraient se référer à la chaîne concernée avant de pouvoir l'effacer. Appelée sans option, cette commande efface toutes les chaînes de la table spécifiée, exceptées les chaînes pré–définies.
Commande	***–P***, ***––policy***
Exemple	***iptables –P INPUT DROP***
Explication	Cette commande indique au noyau de configurer une cible par défaut – ou une stratégie – sur une chaîne. Ceci conditionne le comportement par défaut de la chaîne. Les paquets qui n'établissent de correspondance avec aucune règle sont contraintes de suivre la stratégie de la chaîne. Les seules cibles autorisées sont ***DROP*** et ***ACCEPT*** (il pourrait y en avoir d'autres, écrivez–moi si c'était le cas).
Commande	***–E***, ***––rename–chain***
Exemple	***iptables –E allowed disallowed***
Explication	La commande ***–E*** stipule à ***iptables*** de modifier le nom d'une chaîne du premier nom vers le second. Dans l'exemple fourni, on change le nom de la chaîne `allowed` en `disallowed`.

	Remarquez que ceci n'affecte en rien le fonctionnement actuel de la table. C'est juste une modification cosmétique du contenu de la table.

Chaque ligne de commande doit être saisie entièrement, sauf si vous souhaitez seulement bénéficier de l'aide en ligne d'***iptables*** ou obtenir la version de la commande. Utilisez l'option ***–v*** pour afficher la version, et ***–h*** pour visualiser l'aide. Ci–dessous sont présentées quelques options utilisables avec des commandes diverses. On décrit l'effet des options et les commandes qu'elles concernent. Notez que les options incluses ici n'affectent ni les règles, ni les correspondances. Néanmoins, les correspondances et les cibles sont examinées dans une section ultérieure de ce chapitre.

Tableau 9.3. Options

Option	***–v***, ***––verbose***
Commandes compatibles	***––list***, ***––append***, ***––insert***, ***––delete***, ***––replace***
Explication	Cette option correspond au mode verbeux qui délivre un affichage détaillé. Elle est surtout utilisée avec la commande ***––list*** ; dans ce cas, elle affiche l'adresse de l'interface, les options des règles et les masques de TOS. Elle inclut également un compteur d'octets et de paquets pour chaque règle. Ces compteurs intègrent les suffixes multiplicateurs K (x1000), M (x1.000.000) et G (x1.000.000.000). Vous pouvez suspendre ce mode et obtenir les valeurs exactes avec l'option ***–x***, décrite plus loin. Quand l'option ***––verbose*** est utilisée avec une des commandes ***––append***, ***––insert***, ***––delete*** ou ***––replace***, le programme fournit des informations détaillées sur la façon dont la règle est interprétée, si elle est insérée correctement, etc.
Option	***–x***, ***––exact***
Commandes compatibles	***––list***
Explication	Cette option affine les valeurs numériques. En d'autres termes, la commande ***––list*** n'affiche plus les suffixes multiplicateurs K, M ou G. A la place, elle délivre les valeurs exactes des compteurs donnant le nombre de paquets et d'octets qui ont correspondu à chaque règle. Notez que cette option ne fonctionnne qu'avec la commande ***––list***, et qu'elle n'est applicable avec aucune autre commande.
Option	***–n***, ***––numeric***
Commandes compatibles	***––list***
Explication	Cette option indique à ***iptables*** de sortir les informations en format numérique. Les adresses IP et numéros de port sont affichés sous forme de valeurs numériques au lieu des noms d'hôtes, de réseaux ou d'applications. Cette option s'applique seulement à la commande ***––list***. Elle se substitue au comportement par défaut qui tente de résoudre toutes les valeurs numériques en noms d'hôtes ou de services dès que c'est possible.
Option	***––line–numbers***
Commandes compatibles	***––list***
Explication	L'option ***––line–numbers*** associée à la commande ***––list*** permet d'afficher aussi les numéros de ligne. Avec cette option, chaque règle est donc affichée avec son numéro, ce qui s'avère pratique pour identifier les règles et leur numéro avant d'insérer d'autres règles. Cette option ne fonctionne qu'avec la commande ***––list***.
Option	***–c***, ***––set–counters***

Commandes compatibles	***--insert***, ***--append***, ***--replace***
Explication	Cette option est utile lors d'une création de règle ou de certaines modifications. Cette option permet d'initialiser les compteurs de paquets et d'octets pour une règle. La syntaxe ressemble à ***--set-counters 20 4000***, ce qui stipule au noyau de mettre le compteur de paquets à 20 et le compteur d'octets à 4000.
Option	***--modprobe***
Commandes compatibles	Toutes
Explication	L'option ***--modprobe*** permet de préciser à ***iptables*** quel module utiliser lorsqu'il détecte les modules ou qu'il cherche à en ajouter au noyau. Elle est aussi pratique si la commande ***modprobe*** n'est pas accessible par défaut sur votre système. Dans certains cas, spécifier cette option est nécessaire pour que le programme sache quoi faire lorsqu'un module utile n'est pas chargé. Elle est utilisable avec toutes les commandes.

Chapitre 10. Correspondances

Cette section permet d'approfondir les correspondances. Elles sont intentionnellement classées en cinq catégories distinctes. En premier, on trouve les *correspondances génériques* qui s'emploient avec toutes les règles. Ensuite, il y a les *correspondances TCP* qui ne s'appliquent qu'aux paquets `TCP`. De même, il y a les *correspondances UDP* qui ne s'appliquent qu'aux paquets `UDP`, et les *correspondances ICMP* qui ne s'appliquent qu'aux paquets `ICMP`. Et à la fin, on décrit les *correspondances spéciales*, comme les correspondances d'état, de propriétaire, de limite, etc... Ces dernières correspondances sont réparties en autant de sous−catégories, même si elles ne se révèlent pas singulièrement si différentes. J'espère que cette répartition est suffisamment cohérente pour être compréhensible.

Comme vous l'avez peut être déjà compris si vous avez lu les chapitres précédents, une correspondance est quelque chose qui spécifie une condition spéciale dans le paquet et qui doit être vraie (ou fausse). Une seule règle peut contenir plusieurs correspondances de cette sorte. Par exemple, nous voulons apparier des paquets issus d'un hôte spécifique sur notre réseau local, et seulement des ports particuliers sur cet hôte. Nous utilisons alors les correspondances qui indiquent la règle à appliquer à la cible − ou saut − sur les paquets qui ont une adresse source spécifique, arrivant sur l'interface connectée au réseau local et ces paquets doivent être sur un des ports spécifiés. Si une de ces correspondances est erronée (ex. l'adresse source est incorrecte, mais le reste est correct), la règle complète échoue et la règle suivante est testée sur le paquet. Si toutes les correspondances sont vraies, la cible spécifiée par la règle est appliquée.

10.1. Correspondances génériques

Les *correspondances génériques* désignent un type de correspondance toujours disponible, et ce quel que soit le protocole concerné ou les extensions de correspondances chargées. Autrement dit, ces correspondances ne requièrent aucun paramètre particulier. La correspondance ***--protocol*** a été délibérément incluse ici, bien qu'elle s'adresse spécifiquement aux protocoles. Par exemple, si vous désirez utiliser une *correspondance TCP*, vous devez appeler la correspondance ***--protocol*** et lui fournir `TCP` pour option. Pourtant, ***--protocol*** est également en elle−même une correspondance générique, puisqu'elle permet d'établir une correspondance avec des protocoles différents. Les correspondances suivantes sont donc toujours disponibles.

Tableau 10.1. Correspondances génériques

Correspondance	***-p***, ***--protocol***
Noyau	2.3, 2.4, 2.5 et 2.6

Exemple	***iptables –A INPUT –p tcp***
Explication	Cette correspondance permet de vérifier le type de protocole, par exemple `TCP`, `UDP` ou `ICMP`. De plus, le protocole doit nécessairement soit faire partie des protocoles définis en interne comme TCP, UDP ou ICMP, soit prendre une valeur spécifiée dans le fichier `/etc/protocols`, ce qui, si elle ne s'y trouve pas, retourne une erreur. Le protocole peut aussi être entré sous forme d'un nombre entier. A titre d'exemple, le protocole ICMP est identifié par la valeur entière 1, TCP par la valeur 6 et UDP par 17. Et finalement, le protocole peut aussi prendre la valeur ALL. *ALL* signifie tous, donc il établit une correspondance avec tous les protocoles TCP, UDP et ICMP. La commande accepte aussi une liste de protocoles séparés par des virgules, telle que ***udp,tcp*** qui permet d'établir une correspondance avec tous les paquets `UDP` et `TCP`. Si on désigne le protocole par la valeur zéro (0), ceci est équivalent à ALL, soit tous les protocoles, qui est aussi la valeur par défaut si la correspondance ***––protocol*** est omise. Cette correspondance peut également être inversée à l'aide du symbole ***!***. Dans ce cas, ***––protocol ! tcp*** identifie les protocoles différents de `TCP`, et établit donc une correspondance avec `UDP` et `ICMP`.
Correspondance	***–s***, ***––src***, ***––source***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –s 192.168.1.1***
Explication	C'est la correspondance de source. Elle sert à sélectionner les paquets à partir de leur adresse IP source. La forme principale permet d'établir une correspondance avec des adresses IP uniques, telles que *192.168.1.1*. Mais il est possible d'employer un masque réseau sous une forme binaire de type CIDR, en spécifiant le nombre de "1" dans la partie gauche du masque réseau. Par exemple, ajouter */24* signifie utiliser le masque réseau *255.255.255.0*. Ainsi, un intervalle complet d'adresses IP peut être détecté, comme celui d'un réseau local ou d'un sous–réseau derrière un pare–feu. La commande ressemble alors à *192.168.0.0/24*, qui établit une correspondance avec les paquets de l'intervalle *192.168.0.x*. Une autre méthode consiste à utiliser un masque réseau ordinaire de la forme *255.255.255.255*, ce qui donne au final *192.168.0.0/255.255.255.0*. On peut également inverser la sélection avec un ***!*** comme précédemment. Ainsi, avec une correspondance du type ***––source ! 192.168.0.0/24***, on établit une correspondance avec tous les paquets dont l'adresse source n'appartient pas à l'intervalle *192.168.0.x*. Le comportement par défaut sélectionne toutes les adresses IP.
Correspondance	***–d***, ***––dst***, ***––destination***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –d 192.168.1.1***
Explication	La correspondance ***––destination*** est utilisée pour sélectionner les paquets à partir de leur(s) adresse(s) destination. Ceci fonctionne sensiblement comme la correspondance ***––source*** et avec la même syntaxe, excepté qu'on s'intéresse ici à la destination des paquets. Pour correspondre avec un intervalle d'adresses IP, on peut ajouter un masque réseau soit sous sa forme exacte, soit avec le nombre de 1 compris dans la partie gauche du masque réseau sous forme binaire. voici des exemples : *192.168.0.0/255.255.255.0* et *192.168.0.0/24*. Les deux sont parfaitement équivalents. Il est toujours possible d'inverser la sélection à l'aide du signe ***!*** comme précédemment.

	−−destination ! 192.168.0.1 établit une correspondance avec tous les paquets sauf ceux qui sont destinés à l'adresse IP *192.168.0.1*.
Correspondance	***−i***, ***−−in−interface***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −i eth0***
Explication	Cette correspondance est destinée à sélectionner les paquets issus d'une certaine interface. Remarquez que cette option n'est autorisée que dans les chaînes `INPUT`, `FORWARD` et `PREROUTING`, et qu'elle retourne une erreur si elle est utilisée ailleurs. Si aucune interface n'est spécifiée, le comportement par défaut présuppose que le caractère **+** a été omis. Ce caractère permet d'établir une correspondance avec une chaîne de caractères (composée de lettres et chiffres). Un simple **+** stipule au noyau de reconnaître tous les paquets sans identifier leur interface d'origine. Le caractère **+** peut également être juxtaposé au type d'interface, donc ***eth+*** désigne tous les périphériques Ethernet. Le sens de cette option peut être inversée à l'aide du symbole ***!***. Une ligne dont la syntaxe est ***−i ! eth0*** cherche à correspondre à toutes les interfaces d'entrée, sauf `eth0`.
Correspondance	***−o***, ***−−out−interface***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A FORWARD −o eth0***
Explication	La correspondance ***−−out−interface*** permet de sélectionner les paquets en fonction de l'interface par laquelle ils sortent. Remarquez que cette correspondance n'est disponible que pour les chaînes `OUTPUT`, `FORWARD` et `POSTROUTING`, à l'opposé de la correspondance ***−−in−interface***. A part ça, elle fonctionne presque de la même façon. L'extension **+** traduit une correspondance avec des périphériques similaires, ainsi ***eth+*** établit une correspondance avec tous les périphériques de type `eth`, et ainsi de suite. Pour inverser le sens de la sélection, utilisez le signe ***!*** exactement comme pour la correspondance ***−−in−interface***. Si aucune interface de sortie n'est spécifiée avec ***−−out−interface***, le comportement par défaut accepte tous les périphériques, indépendemment de la direction prise par les paquets.
Correspondance	***−f***, ***−−fragment***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −f***
Explication	Cette correspondance est destinée à sélectionner le deuxième fragment et les suivants d'un paquet fragmenté. En fait, dans le cas d'un paquet fragmenté, il est impossible de connaître ni les ports source ou destination des fragments, ni les types `ICMP`, ni d'autres choses encore. Ainsi, les paquets fragmentés peuvent être utilisés dans des cas très particuliers pour organiser des attaques contre des ordinateurs. De tels fragments ne correspondent à aucune autre règle, ce qui a conduit à créer celle−ci. Cette option peut aussi être employée avec le symbole ***!*** ; mais exceptionnellement ici, le signe ***!*** doit précéder la correspondance, c'est−à−dire ***! −f***. Quand cette correspondance est inversée, elle sélectionne tous les fragments d'en−tête et/ou tous les paquets non fragmentés. Ceci signifie qu'on établit une correspondance avec tous les premiers fragments des paquets fragmentés, et pas avec les deuxièmes, troisièmes, et ainsi de suite. On établit aussi une correspondance avec les

	paquets qui n'ont pas été fragmentés pendant le transfert. Notez qu'il y a d'excellentes options de défragmentation dans le noyau, et qui peuvent se substituer à cette correspondance. Notez également que si vous utilisez du traçage de connexion, vous ne verrez aucun paquet fragmenté, puisqu'ils sont pris en compte avant d'atteindre les chaînes ou les tables dans ***iptables***.

10.2. Correspondances implicites

Cette section se charge de décrire les correspondances chargées implicitement. Les *correspondances implicites* sont sous–jacentes, acquises et automatiques. Par exemple, lorsqu'on établit une correspondance avec ***––protocol tcp*** sans autre critère. Il y a actuellement trois types de correspondances implicites pour trois différents protocoles : les *correspondances TCP*, les *correspondances UDP* et les *correspondances ICMP*. Les correspondances basées sur `TCP` contiennent un ensemble de critères uniquement valables pour les paquets `TCP`. De même pour les correspondances `UDP` et `ICMP`. D'un autre côté, il peut aussi y avoir des correspondances explicites, c'est–à–dire chargées explicitement. Les *correspondances explicites* ne sont ni sous–jacentes, ni automatiques, vous devez obligatoirement les spécifier. Pour celles–ci, utilisez l'option ***–m*** ou ***––match***, qui est abordée dans la section suivante.

10.2.1. Correspondances TCP

Ces correspondances sont dédiées à un protocole, et en l'occurence elles sont seulement disponibles pour des paquets ou des flux `TCP`. Pour utiliser ces correspondances, vous devez ajouter ***––protocol tcp*** à la ligne de commande avant de vous en servir. Notez bien que ***––protocol tcp*** doit précéder (donc être situé à gauche) les correspondances spécifiques au protocole. Celles–ci peuvent être chargées implicitement de la même façon que peuvent l'être les *correspondances UDP* et *ICMP*. Les autres correspondances sont développées à la suite de cette section.

Tableau 10.2. Correspondances TCP

Correspondance	***––sport***, ***––source–port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp ––sport 22***
Explication	La correspondance ***––source–port*** permet de sélectionner des paquets à partir de leur port source. Sans cela, on sous–entend tous les ports source. La correspondance accepte indifféremment un nom de service ou un numéro de port. Si vous spécifiez un nom de service, celui–ci doit figurer dans le fichier `/etc/services`, parce qu'***iptables*** s'appuie sur ce fichier pour identifier le service. Si vous spécifiez le port par son numéro, la règle sera chargée légèrement plus vite, puisqu' ***iptables*** n'a pas à valider le nom du service. Cependant, la correspondance risque d'être un peu plus difficile à lire qu'avec un nom de service. Si vous écrivez une table de règles constituée de plus de 200 règles, vous devriez utiliser les numéros de port, car la différence devient sensible (sur une machine lente, ceci peut conduire à un écart de 10 secondes, si vous avez défini une table de règles contenant au moins 1000 règles). La correspondance ***––source–port*** permet aussi de sélectionner n'importe quel intervalle de ports. Par exemple, ***––source–port 22:80*** établit une correspondance avec tous les ports source compris entre 22 et 80. Si vous omettez la spécification du premier port, le port 0 est implicitement considéré. Ainsi, ***––source–port :80*** permet d'établir une correspondance avec les ports de 0 à 80. Et si vous omettez la spécification du dernier port, le port 65535 est considéré. Ainsi, ***––source–port 22:*** permet d'établir une correspondance avec tous les ports de 22 à 65535. Si vous intervertissez les ports de l'intervalle, ***iptables*** corrige automatiquement en réordonnant les numéros. Donc, écrire ***––source–port 80:22*** est naturellement interprété ***––source–port 22:80***. Une correspondance peut être inversée en ajoutant le symbole ***!***. Par exemple,

	−−source−port ! 22 signifie établir une correspondance avec tous les ports sauf le port 22. L'inversion peut s'appliquer aussi à un intervalle de ports, comme par exemple ***−−source−port ! 22:80*** qui établit une correspondance avec tous les ports sauf ceux de l'intervalle 22 à 80. Notez que cette correspondance n'accepte pas plusieurs ports ou intervalles de ports distincts. Pour plus d'information sur cette possibilité, consultez l'extension de correspondance multiport.
Correspondance	***−−dport***, ***−−destination−port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −−dport 22***
Explication	Cette correspondance permet de sélectionner des paquets `TCP` en fonction de leur port destination. Elle s'appuie sur la même syntaxe que la correspondance ***−−source−port***. Elle comprend les spécifications de ports et d'intervalle de ports, ainsi que l'option d'inversion. De même, elle intervertit si nécessaire les premier et dernier ports dans la spécification d'intervalle, comme ci−dessus. Cette correspondance considère également par défaut les valeurs de ports de 0 et 65535 si les extrémités d'intervalle sont omises. En définitive, elle fonctionne exactement selon la même syntaxe que ***−−source−port***. Notez que cette correspondance n'accepte pas plusieurs ports ou intervalles de ports distincts. Pour plus d'information sur cette possibilité, consultez l'extension de correspondance multiport.
Correspondance	***−−tcp−flags***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −p tcp −−tcp−flags SYN,FIN,ACK SYN***
Explication	Cette correspondance permet de sélectionner les paquets à partir de leurs fanions `TCP`. En premier, la correspondance requiert une liste de fanions à tester (un masque) suivie de la liste des fanions qui doivent être positionnés à 1 (donc activés). Dans les deux listes, les fanions sont séparés par des virgules. La correspondance reconnaît les fanions `SYN`, `ACK`, `FIN`, `RST`, `URG` et `PSH`. Elle accepte aussi les mots `ALL` (tous) et `NONE` (aucun) dont le sens est plutôt intuitif : ALL équivaut à tous les fanions et NONE à aucun. Typiquement, ***−−tcp−flags ALL NONE*** vérifie tous les fanions `TCP` et établit une correspondance si aucun n'est activé (donc positionné à 1). Cette option peut également être inversée à l'aide du signe ***!***. Par exemple, spécifier ***! SYN,FIN,ACK SYN*** revient à faire correspondre les paquets qui possèdent les bits `ACK` et `FIN` activés, mais pas le bit `SYN`. Notez que la séparation des fanions par des virgules ne doit inclure aucune espace, comme vous pouvez le voir dans l'exemple ci−dessus.
Correspondance	***−−syn***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −p tcp −−syn***
Explication	La correspondance ***−−syn*** est plus ou moins une relique du règne d'ipchains. Elle perdure pour garantir une certaine rétro−compatibilité et simplifier la transition vers ***iptables***. Elle permet d'établir une correspondance avec des paquets s'ils possèdent le bit `SYN` activé et les bits `ACK` et `RST` désactivés. Cette commande se comporte rigoureusement comme la correspondance ***−−tcp−flags SYN,RST,ACK SYN***. Les paquets de ce type servent principalement aux demandes de connexion en provenance de serveurs. Si vous bloquez ces paquets, vous devriez effectivement empêcher toutes les tentatives de connexions entrantes. Toutefois, vous ne bloquerez pas les connexions sortantes, qui sont mises à profit aujourd'hui par de nombreux exploits (par exemple, détourner un service légitime pour installer localement un programme ou créer une liaison à partir d'une connexion existante sur votre hôte au lieu d'ouvrir un nouveau port). Cette correspondance peut

	également être inversée à l'aide du signe ***!***. Ainsi, ***! --syn*** correspond à tous les paquets ayant les bits `RST` ou `ACK` activés, autrement dit les paquets appartenant à une connexion déjà établie.
Correspondance	***--tcp-option***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -p tcp --tcp-option 16***
Explication	Cette correspondance permet d'établir une correspondance avec des paquets suivant leurs options `TCP`. Une `option TCP` identifie une partie spécifique de l'en-tête des paquets. Cette partie contient 3 champs différents. Le premier a une longueur de 8 bits et décrit les options utilisées dans ce flux ; le deuxième s'étend aussi sur 8 bits et précise la longueur du champ des options. L'information de longueur du champ doit son existence au caractère optionnel des `options TCP`. Pour être conforme aux standards, il n'est pas utile d'implémenter toutes les options, il suffit de les identifier. Si elles ne sont pas prises en charge, on lit seulement l'information de longueur afin de sauter par-dessus ces données. Cette correspondance permet de sélectionner plusieurs options `TCP` en fonction de leurs valeurs numériques. Elle peut également être inversée avec le signe ***!***, de telle sorte que la correspondance s'établisse avec toutes les options `TCP` sauf celle passée en paramètre. Pour obtenir la liste complète des options, consultez le site Internet Engineering Task Force qui contient une liste de toutes les valeurs standardes employées sur Internet.

10.2.2. Correspondances UDP

Cette section décrit les correspondances qui fonctionnent seulement avec des paquets `UDP`. Elles sont chargées implicitement lorsque la correspondance ***--protocol UDP*** est spécifiée et elles ne sont effectivement disponible qu'après cette spécification. Notez que les paquets `UDP` ne sont pas orientés connexion, et par conséquent ils ne possèdent pas de fanions particuliers pour informer du rôle joué par le datagramme tel que l'ouverture ou la fermeture d'une connexion, ou encore le simple envoi de données. Les paquets `UDP` ne nécessitent aucun accusé de réception. S'ils s'égarent sur le réseau, ils n'engendrent aucune action (aucun message d'erreur de type `ICMP` n'est expédié). Autrement dit, il existe nettement moins de correspondances associées aux paquets `UDP` qu'aux paquets `TCP`. Notez que la machine d'état fonctionne sur tous les types de paquets, même si les paquets `UDP` et `ICMP` appartiennent à des protocoles sans connexion. La machine d'état fonctionne quasiment de la même façon pour les paquets `UDP` que pour les paquets `TCP`.

Tableau 10.3. Correspondances UDP

Correspondance	***--sport***, ***--source-port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p udp --sport 53***
Explication	Cette correspondance fonctionne exactement comme son équivalent pour `TCP`. Elle permet d'établir des correspondances avec des paquets à partir de leurs ports source `UDP`. Elle prend en charge les intervalles de ports, les ports uniques et les inversions de ports selon la même syntaxe. Pour spécifier un intervalle de ports `UDP`, vous pouvez utiliser 22:80 qui établit une correspondance avec les ports `UDP` de 22 à 80. Si le premier numéro est omis, il est considéré par défaut comme étant le port 0. Si le dernier numéro est omis, le port 65535 est pris par défaut. Si le port le plus grand est mis avant le plus petit, les numéros sont intervertis automatiquement. Dans le cas d'un port `UDP` unique, la syntaxe se calque sur l'exemple ci-dessus. Pour inverser la correspondance de port, il suffit d'insérer le signe ***!***. Dans ***--source-port ! 53***, la correspondance s'établit avec tous les ports sauf le numéro 53. Cette correspondance comprend les noms de service, du moment qu'ils sont disponibles dans le fichier `/etc/services`. Notez que cette correspondance n'accepte

	pas les ports et les intervalles de ports distincts. Pour davantage d'information, consultez l'extension de correspondance multiport.
Correspondance	***--dport***, ***--destination-port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p udp --dport 53***
Explication	Cette correspondance s'apparente fortement à ***--source-port*** décrite ci-dessus. Elle est aussi très proche de la correspondance `TCP` équivalente, sauf qu'elle s'applique aux paquets `UDP`. Elle établit une correspondance à partir du port destination `UDP`. Elle accepte les intervalles de ports, les ports uniques et les inversions. Pour sélectionner un port unique, vous pouvez utiliser par exemple ***--destination-port 53*** ; pour l'inverser, ce sera plutôt ***--destination-port ! 53***. La première commande sélectionnne tous les paquets `UDP` en direction du port 53, alors que la seconde sélectionne tous les paquets sauf ceux destinés au port 53. Pour spécifier un intervalle de ports, utilisez par exemple ***--destination-port 9:19*** pour établir une correspondance avec tous les paquets destinés aux ports `UDP` compris entre 9 et 19. Si le premier port est omis, on considère le port 0 par défaut. Si le second est omis, on considère le port 65535 par défaut. Si le port le plus grand est placé avant le plus petit, ils sont interchangés automatiquement pour que le plus petit port précède le plus grand. Notez que cette correspondance n'accepte pas les ports et intervalles de ports distincts. Pour plus d'information, consultez l'extension de correspondance multiport.

10.2.3. Correspondances ICMP

Abordons maintenant les *correspondances ICMP*. Les paquets `ICMP` sont de nature éphémère, c'est-à-dire qu'ils ont la vie courte, plus courte que les paquets `UDP` dans le sens où ils sont sans connexion. Le protocole `ICMP` sert principalement aux messages d'erreur, aux contrôles de connexion, et d'autres choses du même acabit. `ICMP` n'est pas un protocole subordonné au protocole IP, mais plutôt qui enrichit le protocole IP et concourt à la gestion des erreurs. L'en-tête des paquets `ICMP` ressemble à celle des paquets IP, mais diffère sur certains aspects. La caractéristique primordiale de ce protocole provient du type d'en-tête, qui traduit la raison d'être du paquet. A titre d'exemple, si on tente d'accéder à une adresse IP inaccessible, on récupère normalement en retour un `ICMP host unreachable` (machine inaccessible). Pour visualiser la liste complète des types `ICMP`, consultez l'annexe Types ICMP. Une seule correspondance `ICMP` spécifique est disponible pour les paquets `ICMP`, et heureusement, elle devrait suffire. Cette correspondance est chargée implicitement quand on spécifie ***--protocol ICMP***, et on en dispose automatiquement. Notez que toutes les correspondances génériques sont utilisables, et qu'elles permettent par exemple de sélectionner les adresses source et destination.

Tableau 10.4. Correspondances ICMP

Correspondance	***--icmp-type***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p icmp --icmp-type 8***
Explication	Cette correspondance permet de spécifier le `type ICMP` à sélectionner. Les `types ICMP` peuvent être définis soit par leur valeur numérique, soit par leur nom. Les valeurs numériques sont spécifiés dans le RFC 792. Pour afficher la liste complète des noms `ICMP`, exécutez la commande ***iptables --protocol icmp --help*** ou consultez l'annexe Types ICMP. Cette correspondance peut être inversée en insérant le signe ***!*** de cette façon : ***--icmp-type ! 8*** ou ***--icmp-type 8/0***. Pour une liste complète des noms, tapez ***iptables -p icmp --help***.

10.3. Correspondances explicites

Les correspondances explicites ont besoin d'être chargés spécifiquement à l'aide de l'option ***–m*** ou ***––match***. Par exemple, les correspondances d'état requiert la directive ***–m state*** avant d'entrer la véritable correspondance à prendre en compte. Certaines de ces correspondances sont spécifiques à un protocole. Certaines peuvent aussi être détachées de tout protocole spécifique – par exemple les états de connexion. Ils sont identifiés par ***NEW*** (pour le premier paquet d'une connexion non encore établie), ***ESTABLISHED*** (pour une connexion déjà enregistrée dans le noyau), ***RELATED*** (pour une nouvelle connexion créée par une connexion plus ancienne et déjà établie), etc. Parmi ces correspondances explicites, quelques–une peuvent avoir évolué pour des questions de test ou d'expérimentation, ou simplement pour mettre en évidence les capacités d'***iptables***. Par conséquent, ceci signifie que l'intégralité de ces correspondances n'est pas à première vue indispensable. Néanmoins, il y a de grandes chances que vous trouviez certaines des ces correspondances explicites particulièrement utiles. Et de nouvelles apparaissent en permanence, lors de chaque nouvelle version d'***iptables***. Que vous leur découvriez ou non une utilisation dépend de votre imagination et de vos besoins. Pour comprendre la différence entre une correspondance chargée implicitement et une chargée explicitement, il faut savoir que la première est chargée automatiquement quand par exemple vous établissez une correspondance avec une propriété des paquets `TCP`, alors que la seconde n'est jamais chargée automatiquement – c'est à vous de révéler et d'activer une correspondance explicite.

10.3.1. Correspondance AH/ESP

Ces correspondances sont utilisées pour les protocoles IPSEC AH et ESP. IPSEC sert à créer des tunnels sécurisés par dessus une connexion Internet non sécurisée. Les protocoles AH et ESP sont utilisés par IPSEC pour créer ces connexions sécurisées. Les correspondances AH et ESP sont deux correspondances séparées, mais elles sont toutes les deux décrites ici car elles se ressemblent beaucoup, et toutes les deux ont le même usage.

Je ne rentrerai pas dans les détails d'IPSEC ici, regardez les pages suivantes pour plus d'information :

- RFC 2401 – Security Architecture for the Internet Protocol
- FreeS/WAN
- IPSEC Howto
- Linux Advanced Routing and Traffic Control HOW–TO

Il existe aussi des tonnes de documentation sur l'Internet à ce sujet.

Pour utiliser les correspondances AH/ESP, vous devrez vous servir de ***–m ah*** pour charger les correspondances AH, et ***–m esp*** pour charger les correspondances ESP.

Note

Dans les noyaux 2.2 et 2.4, Linux utilise une chose appelée FreeS/WAN pour l'implémentation de IPSEC, mais à partir des noyaux 2.5.47 et supérieurs, ceux–ci ont une implémentation directe de IPSEC et donc ne nécessitent pas de patcher le noyau. C'est une réécriture complète de l'implémentation de IPSEC dans Linux.

Tableau 10.5. Options des correspondances AH

Correspondance	***––ahspi***
Noyau	2.5 et 2.6
Exemple	***iptables –A INPUT –p 51 –m ah ––ahspi 500***
Explication	Ceci vérifie le numéro de l'Index du Paramètre de Sécurité (SPI) des paquets AH. Notez que vous devez spécifier le protocole, car AH s'exécute sur un protocole différent des

	standards TCP, UDP et ICMP. Le numéro SPI est utilisé en conjonction avec les adresses source et destination et les clés secrètes pour créer une association de sécurité (SA). SA identifie chacun des tunnels IPSEC pour tous les hôtes. SPI est utilisé uniquement pour distinguer chaque tunnel IPSEC connecté entre deux identiques. Utiliser la correspondance ***−−ahspi***, nous permet de vérifier un paquet basé sur le SPI des paquets. Cette correspondance peut vérifier une chaîne complète de valeur SPI en utilisant un signe :, comme 500:520, qui vérifiera toute la chaîne des SPI.

Tableau 10.6. Options des correspondances ESP

Correspondance	***−−espspi***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p 50 −m esp −−espspi 500***
Explication	La contrepartie de l'Index des Paramètres de Sécurité (SPI) est utilisée de la même façon que la variante AH. La correspondance semble exactement la même, avec seulement la différence esp/ah. Bien sûr, cette correspondance peut apparier un ensemble complet de numéros SPI de la même façon que la variante AH de la correspondance SPI, comme −−espi 200:250 qui apparie la totalité de la chaîne des SPI.

10.3.2. Correspondance conntrack

La correspondance ***conntrack*** est une version étendue de la correspondance état, qui rend possible l'appariement des paquets de façon un peu plus grossière. Ce qui vous permet d'avoir l'information directement disponible dans un système de traçage de connexion, sans applications frontales, comme dans la correspondance état. Pour plus de détails sur le système de traçage de connexion, regardez le chapitre La machine d'état.

Il existe nombre de différents appariements dans la correspondance conntrack, pour différents champs dans le système de traçage de connexion. Ils sont indiqués ensemble dans la liste ci−dessous. Pour charger ces correspondances, vous devez spécifier ***−m conntrack***.

Tableau 10.7. Options de correspondance conntrack

Correspondance	***−−ctstate***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctstate RELATED***
Explication	Cette correspondance est utilisée pour apparier l'état d'un paquet, selon l'état conntrack. Elle est utilisée pour apparier plus finement les mêmes états que dans la correspondance ***state*** d'origine. Les entrées valides pour cette correspondance sont : ♦ INVALID ♦ ESTABLISED ♦ NEW ♦ RELATED ♦ SNAT ♦ DNAT Les entrées peuvent être utilisées l'une avec l'autre en les séparant par une virgule. Par exemple, ***−m conntrack −−ctstate ESTABLISHED,RELATED***. Elles peuvent aussi être interverties en mettant un ! avant ***−−ctstate***. Exemple : ***−m conntrack ! −−ctstate ESTABLISHED,RELATED***, qui apparie tout sauf les états ***ESTABLISHED*** et

	RELATED.
Correspondance	***−−ctproto***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctproto TCP***
Explication	Ceci apparie le protocole, de la même façon que le fait ***−−protocol***. Il peut prendre les mêmes types de valeurs, et on peut l'intervertir en utilisant le signe !. Exemple, ***−m conntrack ! −−ctproto TCP*** apparie tous les protocoles sauf `TCP`.
Correspondance	***−−ctorigsrc***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctorigsrc 192.168.0.0/24***
Explication	***−−ctorigsrc*** est une correspondance basée sur la spécification de la source IP d'origine de l'entrée conntrack en rapport avec le paquet. La correspondance peut être inversée en utilisant le ! entre le ***−−ctorigsrc*** et la spécification IP, comme ***−−ctorigsrc ! 192.168.0.1***. Elle peut aussi prendre un masque de réseau de forme CIDR, comme ***−−ctorigsrc 192.168.0.0/24***.
Correspondance	***−−ctorigdst***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctorigdst 192.168.0.0/24***
Explication	Cette correspondance est utilisée de la même façon que ***−−ctorigsrc***, sauf qu'elle apparie sur le champ destination de l'entrée conntrack. Elle possède la même syntaxe.
Correspondance	***−−ctreplsrc***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctreplysrc 192.168.0.0/24***
Explication	La correspondance ***−−ctreplysrc*** est utilisée pour l'appariement basé sur la réponse source du conntrack d'origine du paquet. C'est à peu près la même chose que le ***−−ctorigsrc***, mais nous apparions la réponse source attendue des paquets envoyés. Cette cible peut, bien sûr, être inversée et adresser une chaîne complète d'adresses, de la même façon que la cible précédente dans cette classe.
Correspondance	***−−ctrepldst***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctreplydst 192.168.0.0/24***
Explication	La correspondance ***−−ctreplydst*** est la même que ***−−ctreplysrc***, avec la différence qu'elle apparie la réponse de destination de l'entrée conntrack qui a apparié la paquet. Elle peut être inversée, et accepte les chaînes, comme la correspondance ***−−ctreplysrc***.
Correspondance	***−−ctstatus***
Noyau	2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m conntrack −−ctstatus RELATED***
Explication	Ceci apparie les statuts de la connexion, comme décrit dans la chapitre La machine d'état. Ces statuts sont les suivants : ♦ NONE – La connexion ne possède aucun statut.

	♦ EXPECTED – Cette connexion est en attente et a été ajoutée par les gestionnaires d'attente. ♦ SEEN_REPLY – La connexion a vu une réponse mais n'en est cependant pas assurée. ♦ ASSURED – La connexion est certaine et ne sera pas supprimée tant que le délai d'attente ne sera pas atteint ou qu'elle sera interrompue d'une autre façon. Elle peut aussi être inversée par le signe !. Exemple, ***–m conntrack ! ––ctstatus ASSURED*** qui apparie tout sauf le statut ASSURED.
Correspondance	***––ctexpire***
Noyau	2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m conntrack ––ctexpire 100:150***
Explication	Cette correspondance sert à apparier les paquets basés sur la longueur du temps d'expiration de l'entrée conntrack, mesuré en secondes. Elle peut soit prendre une seule valeur, ou une chaîne comme dans l'exemple au–dessus. Elle peut aussi être inversée avec le signe !, comme ***–m conntrack ! ––ctexpire 100***. Ceci apparie chaque temps d'expiration, qui n'est pas exactement de 100 secondes.

10.3.3. Correspondance DSCP

Cette correspondance est utilisée pour apparier les paquets basés sur leur champ DSCP (Differentiated Services Code Point). C'est documenté dans la RFC RFC 2638 – A Two–bit Differentiated Services Architecture for the Internet. La correspondance est chargée en spécifiant ***–m dscp***. Elle peut prendre deux options mutuellement incompatibles, décrites ci–dessous.

Tableau 10.8. Options de correspondance DSCP

Correspondance	***––dscp***
Noyau	2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m dscp ––dscp 32***
Explication	Cette option prend une valeur DSCP soit en décimal soit en hexadécimal. Si la valeur de l'option est en décimal, elle sera écrite comme 32 ou 16, etc. Si elle est écrite en hexadécimal, elle pourrait être préfixée avec des 0x, comme ça : 0x20. Elle peut aussi être inversée par le signe !, comme : ***–m dscp ! ––dscp 32***.
Correspondance	***––dscp–class***
Noyau	2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m dscp ––dscp–class BE***
Explication	La correspondance ***––dscp–class*** sert à l'appariement d'une classe Diffserv d'un paquet. Les valeurs peuvent être l'une des classes BE, EF, AFxx ou CSxx comme spécifié dans les diverses RFC. Elle peut être inversée de la même façon qu'avec l'option ***––dscp***.

Note

Notez que les options de classes ––dscp et dscp–class sont mutuellement exclusives et ne peuvent pas être utilisées conjointement l'une avec l'autre.

10.3.4. Correspondance ECN

ECN est utilisé pour l'appariement sur les différents champs ECN dans les en–têtes TCP et IPv4. ECN est décrit en détail dans la RFC RFC 3168 – The Addition of Explicit Congestion Notification (ECN) to IP. La correspondance est chargée par ***–m ecn*** dans la ligne de commande. Elle prend trois options différentes,

comme décrit ci–dessous.

Tableau 10.9. Options de la correspondance ECN

Correspondance	***––ecn***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m ecn ––ecn–tcp–cwr***
Explication	Cette correspondance est utilisée pour apparier le bit CWR (Congestion Window Received), s'il a été placé. Le fanion CWR est placé pour notifier l'autre point limite de la connexion reçu (ECE), et qui a été réactivé. Par défaut elle vérifie si le bit CWR est placé, mais la correspondance peut aussi être inversée par le signe !.
Correspondance	***––ecn–tcp–ece***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m ecn ––ecn–tcp–ece***
Explication	Cette correspondance peut être utilisée pour apparier le bit `ECE` (`ECN-Echo`). Le ECE est placé une fois que les points limite aient reçu un paquet avec un bit CE placé par un routeur. Le point limite place alors le ECE en renvoyant la paquet ACK, pour le notifier à l'autre point limite. Cet autre point limite envoie alors un paquet CWR comme décrit dans l'explication de ***––ecn–tcp–cwr***. C'est le comportement par défaut si le bit ECE est placé, mais peut être interverti par le signe !.
Correspondance	***––ecn–ip–ect***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m ecn ––ecn–ip–ect 1***
Explication	***––ecn–ip–ect*** est utilisée pour apparier les codes caractères ECT (ECN Capable Transport). Les codes caractères ECT possèdent plusieurs types. Principalement, ils sont utilisés pour savoir si la connexion a les possibilités ECN en plaçant un des deux bits à 1. ECT est aussi utilisé par les routeurs pour indiquer qu'ils sont en processus d'engorgement, en plaçant les deux points limite ECT à 1. les valeurs ECT sont toutes disponibles dans la table Champ ECN dans IP ci–dessous. La correspondance peut être inversée par un !, exemple ***! ––ecn–ip–ect 2*** qui va apparier toutes les valeurs ECN sauf le point limite ECT(0). la chaîne de valeur correcte dans Iptables est de 0 à 3. Voir ci–dessous pour les valeurs :

Tableau 10.10. Champ ECN dans IP

Valeur Iptables	ECT	CE	[Obsolète] Noms RFC 2481 pour les bits ECN
0	0	0	Not–ECT, ie. pas de possibilité de connexion non–ECN
1	0	1	ECT(1), nouvelle convention de nommage des points limite ECT dans la RFC 3168
2	1	0	ECT(0), nouvelle convention de nommage des points limite ECT dans la RFC 3168
3	1	1	CE (Congestion Experienced), utilisé pour notifier les points limite pour l'engorgement.

10.3.5. Correspondance Helper

C'est plutôt une correspondance pas très orthodoxe en comparaison des autres, dans ce sens qu'elle utilise une syntaxe spécifique de bit. Elle est utilisée pour apparier les paquets, basés sur l'assistant conntrack en relation avec le paquet. Par exemple, regardons une session FTP. La session Contrôle est ouverte, et les ports/connexion sont négociés pour la session Données dans la session Contrôle. Le module assistant ***ip_conntrack_ftp*** va trouver cette information, et créer une entrée dans la table conntrack. Maintenant, quand un paquet entre, nous pouvons voir quel protocole est en relation, et pouvons apparier le paquet dans nos propres tables de règles basées sur l'assistant qui a été utilisé.

Tableau 10.11. Options de correspondance Helper

Correspondance	***--helper***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p tcp -m helper --helper ftp-21***
Explication	L'option ***--helper*** est utilisée pour spécifier une valeur de chaîne, indiquant à la correspondance quelle assistant conntrack apparier. Dans sa forme basique, elle peut ressembler à ***--helper irc***. C'est l'endroit où la syntaxe démarre en variant par rapport à la syntaxe normale. Nous pouvons aussi choisir d'apparier seulement les paquets basés sur tel port que l'original a pris. Exemple, la session Contrôle FTP est normalement transférée sur le port 21, mais il peut aussi bien être sur le port 954 ou un autre. Nous pouvons alors spécifier quel port sera utilisé, comme ***--helper ftp-954***.

10.3.6. Correspondance de plage IP

La correspondance de plage IP est utilisée pour apparier les plages IP, comme les correspondances ***--source*** et ***--destination*** peuvent le faire. Cependant, cette correspondance ajoute une sorte d'appariement différent dans le sens qu'elle peut apparier dans la forme IP à IP, ce que les correspondances ***--source*** et ***--destination*** sont incapables de faire. Ceci peut être nécessaire dans certains réglages de réseaux spécifiques, et elle est légèrement plus souple.

Tableau 10.12. Options de correspondance de plage IP

Correspondance	***--src-range***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p tcp -m iprange --src-range 192.168.1.13-192.168.2.19***
Explication	Ceci apparie une plage d'adresses source IP. La plage inclut chaque adresse depuis la première jusqu'à la dernière, ainsi l'exemple ci-dessus inclut toutes les adresses depuis 192.168.1.13 jusqu'à 192.168.2.19. Elle peut aussi être inversée avec le !. L'exemple du dessus ressemblera alors à ***-m iprange ! --src-range 192.168.1.13-192.168.2.19***, qui va apparier chaque adresse, sauf celles spécifiées.
Correspondance	***--dst-range***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables -A INPUT -p tcp -m iprange --dst-range 192.168.1.13-192.168.2.19***
Explication	***--dst-range*** fonctionne exactement de la même façon que la correspondance ***--src-range***, sauf qu'elle apparie la destination IP au lieu de la source IP.

10.3.7. Correspondance Length

La correspondance Length est utilisée pour apparier les paquets basés sur leur longueur. C'est très simple. Si vous voulez limiter la longueur des paquets pour quelque étrange raison, ou bloquer ce qui ressemble à un ping−of−death, utilisez cette correspondance.

Tableau 10.13. Options de correspondance Length

Correspondance	***−−length***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −m length −−length 1400:1500***
Explication	L'exemple ***−−length*** va apparier tous les paquets de longueur comprise entre 1400 et 1500 octets. Elle peut être inversée en utilisant le signe ***!***, comme ça : ***−m length ! −−length 1400:1500***. Elle peut aussi être utilisée pour apparier certaines tailles seulement, supprimant le signe :, comme ceci : ***−m length −−length 1400***. La plage est, bien sûr, inclusive, ce qui inclut tous les paquets dont la longueur est comprise dans les valeurs que vous avez spécifiées.

10.3.8. Correspondance Limit

L'extension de correspondance ***limit*** doit être chargée explicitement avec l'option ***−m limit***. Cette correspondance peut être employée avantageusement pour limiter la journalisation de certaines règles, etc. Par exemple, vous pouvez établir une correspondance avec tous les paquets qui n'excèdent pas une quantité donnée, et au−delà de ce seuil, limiter la journalisation de l'évènement en question. Considérez ce seuil comme une limite temporelle : vous pouvez limiter le nombre de correspondances d'une certaine règle dans un certain laps de temps, par exemple pour atténuer l'impact des attaques de type déni de service (*DoS*). C'est d'ailleurs la principale application qu'on en fait, mais naturellement, il en existe d'autres. La correspondance ***limit*** peut également être inversée en ajoutant le symbole ***!*** juste après le mot ***limit***. Elle s'exprime alors sous la forme ***−m limit ! −−limit 5/s***, autrement dit tous les paquets sont sélectionnés quand la limite est dépassée.

Pour décrire plus précisément la correspondance ***limit***, c'est essentiellement un filtre à seau de jetons ("token bucket filter"). Considérez un seau percé qui laisse fuir N paquets par unité de temps. N est défini en fonction du nombre de paquets que nous voulons sélectionner, ainsi si nous voulons 3 paquets, le seau laisse fuir 3 paquets par unité de temps. L'option ***−−limit*** détermine le nombre de paquets qui peuvent remplir le seau par unité de temps, alors que l'option ***−−limit−burst*** définit la contenance initiale du seau. Par conséquent, en définissant ***−−limit 3/minute −−limit−burst 5***, puis en recevant 5 correspondances, le seau sera vidé. Après une attente de 20 secondes, le seau est rempli d'un nouveau jeton, et ainsi de suite jusqu'à ce que le paramètre ***−−limit−burst*** soit atteint ou jusqu'à ce que les jetons soient tous utilisés.

Considérez l'exemple ci−dessous pour approfondir sur ce fonctionnement.

1. On définit une règle avec les paramètres −m limit −−limit 5/second −−limit−burst 10/second. Le paramètre limit−burst du seau à jetons est fixé initialement à 10. Chaque paquet qui établit une correspondance avec la règle consomme un jeton.
2. On reçoit alors des paquets qui correspondent à la règle, 1−2−3−4−5−6−7−8−9−10, tous arrivent dans un intervalle de 1/1000ème de seconde.
3. Le seau de jetons se retrouve complètement vide. Et puisque le seau est vide, les paquets qui rencontrent la règle ne peuvent plus correspondre et poursuivent leur route vers la règle suivante, ou subissent le comportement par défaut de la chaîne.
4. Pour chaque tranche de 1/5ème de seconde sans qu'un paquet ne corresponde, le compteur de jetons augmente de 1, jusqu'à un maximum de 10. Et 1 seconde après avoir reçu 10 paquets, on aura de nouveau 5 jetons de moins.
5. Et naturellement, le seau sera vidé d'1 jeton par paquet reçu.

Tableau 10.14. Options de la correspondance limit

Correspondance	***−−limit***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m limit −−limit 3/hour***
Explication	Ceci définit le taux de correspondance moyen maximum pour la correspondance ***limit***. Il est spécifié par un nombre suivi éventuellement d'une unité de temps. Les unités suivantes sont actuellement reconnues : ***/second***, ***/minute***, ***/hour*** et ***/day***. La valeur par défaut est fixée à 3 correspondances par heure, soit ***3/hour***. Ceci indique à la correspondance ***limit*** combien de fois la correspondance avec un paquet est autorisé par unité de temps (par exemple par minute).
Correspondance	***−−limit−burst***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m limit −−limit−burst 5***
Explication	Ceci définit la *réserve maximale* (ou la salve) de la correspondance ***limit***. Il indique à ***iptables*** le nombre maximum de paquets pouvant correspondre pendant l'unité de temps donnée. Ce nombre est décrémenté de 1 après chaque unité de temps (spécifiée par l'option ***−−limit***) pendant laquelle l'évènement ne s'est pas produit, jusqu'à atteindre la plus faible valeur, 1. Si l'évènement se produit de façon répétée, le compteur est alors incrémenté jusqu'à atteindre la valeur de réserve maximale, et ainsi de suite. La valeur par défaut de ***−−limit−burst*** est 5. Pour comprendre simplement comment ceci fonctionne, utilisez le script d'exemple Limit−match.txt composé d'une règle. Grâce à ce script, vous pouvez voir vous−mêmes comment fonctionne le règle limit, en envoyant des paquets d'écho (de type ping) à des intervalles différents et en différentes rafales. Toutes les `réponses d'écho` seront bloquées jusqu'à ce que le seuil de réserve maximale soit de nouveau atteint.

10.3.9. Correspondance MAC

La correspondance `MAC` (`Ethernet Media Access Control`) permet de sélectionner des paquets à partir de leur `adresse MAC source` . Lors de l'écriture de ce document, cette correspondance s'avère quelque−peu limitée, cependant elle pourrait être plus évoluée à l'avenir, donc plus utile.

Note

Remarquez que l'utilisation de ce module impose de le charger explicitement avec l'option ***−m mac***. Il est nécessaire de le rappeler ici vu le nombre de personne croyant pouvoir l'invoquer seulement par ***−m mac−source***, ce qui n'est pas possible.

Tableau 10.15. Options de la correspondance MAC

Correspondance	***−−mac−source***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m mac −−mac−source 00:00:00:00:00:01***
Explication	Cette correspondance permet de sélectionner des paquets à partir de leur `adresse MAC source`. L'`adresse MAC` doit être spécifiée de la forme *XX:XX:XX:XX:XX:XX*, autrement elle n'est pas valide. La correspondance peut être inversée avec le signe ***!*** et ressemble alors à ***−−mac−source ! 00:00:00:00:00:01***. Autrement dit, ceci inverse le sens de la corrrespondance, en sélectionnant tous les paquets sauf ceux possédant l'`adresse MAC` spécifiée. Notez que comme les `adresses MAC` ne sont utilisées que dans les réseaux de type Ethernet, cette correspondance ne s'applique qu'aux interfaces Ethernet. La

	correspondance ***MAC*** est valide seulement dans les chaînes `PREROUTING`, `FORWARD` et `INPUT`, et nulle–part ailleurs.

10.3.10. Correspondance mark

L'extension de correspondance ***mark*** permet de sélectionner des paquets à partir de leur marquage. Le ***marquage*** désigne un champ particulier, pris en charge uniquement au sein du noyau, et lié aux paquets circulant à travers la machine. Le marquage peut être employé par différentes routines du noyau pour des tâches comme de la régulation de trafic ou du filtrage. Aujourd'hui, il n'existe qu'un seul moyen de définir un marquage sous Linux, c'est la cible ***MARK*** dans ***iptables***. Auparavant, il s'agissait de la cible ***FWMARK*** dans ***ipchains***, et c'est pourquoi nombre de gens se réfère encore à ***FWMARK*** dans les documentations avancées sur le routage. Le champ de marquage est actuellement défini comme un entier non signé, soit 4294967296 valeurs possibles sur un système 32 bits. En d'autres termes, vous ne risquez pas de sitôt de dépasser cette limite.

Tableau 10.16. Options de la correspondance mark

Correspondance	***––mark***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –t mangle –A INPUT –m mark ––mark 1***
Explication	Cette correspondance permet de sélectionner des paquets qui ont été préalablement marqués. Les marquages peuvent être positionnés avec la cible ***MARK*** développée dans la section suivante. Tous les paquets transitant par `Netfilter` se voient affectés d'un `champ de marquage` spécial. Notez que ce `champ de marquage` n'est propagé en aucune manière, que ce soit à l'intérieur ou à l'extérieur du paquet. Il reste exclusivement à l'intérieur de la machine qui l'a créé. Si le `champ de marquage` est égal à la valeur de l'option ***––mark***, il y a correspondance. Le `champ de marquage` est un entier non signé, par conséquent 4294967296 différents marquages peuvent exister. Vous pouvez également utiliser un masque avec le marquage. Dans ce cas, la spécification de marquage ressemble, par exemple, à ***––mark 1/1***. Quand un masque est spécifié, on effectue un ET logique avec le marquage spécifié avant de réaliser la comparaison réelle.

10.3.11. Correspondance multiport

L'extension de correspondance ***multiport*** permet de spécifier plusieurs ports et intervalles et ports destination. Sans cette possibilité, vous devriez utiliser différentes règles du même genre, juste pour établir une correspondance avec différents ports.

Note

Vous ne pouvez pas utiliser simultanément la correspondance de port standard et la correspondance multiport. Par exemple, il est inutile d'écrire : ***––sport 1024:63353 –m multiport ––dport 21,23,80***, car ça ne marchera pas. Si vous le faites quand même, iptables considèrera le premier élément de la règle et ignorera l'instruction multiport.

Tableau 10.17. Options de la correspondance multiport

Correspondance	***––source–port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m multiport ––source–port 22,53,80,110***
Explication	Cette correspondance coïncide avec plusieurs ports source. Au maximum, 15 ports peuvent être spécifiés. Les ports doivent être séparés par des virgules, comme dans l'exemple ci–dessus. Cette correspondance ne peut être employée qu'avec les

	correspondances ***–p tcp*** ou ***–p udp***. C'est principalement une version améliorée de la correspondance classique ***––source–port***.
Correspondance	***––destination–port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m multiport ––destination–port 22,53,80,110***
Explication	Cette correspondance coïncide avec plusieurs ports destination. Elle fonctionne exactement de la même façon que la correspondance de port source mentionnée précédemment, excepté qu'elle s'applique aux ports destination. Elle dispose aussi d'une limite de 15 ports, et ne peut être employée qu'avec ***–p tcp*** et ***–p udp***.
Correspondance	***––port***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m multiport ––port 22,53,80,110***
Explication	Cette extension de correspondance permet de sélectionner les paquets à partir à la fois de leur port destination et de leur port source. Elle fonctionne de la même façon que les correspondances ***––source–port*** et ***––destination–port*** présentées ci–dessus. Elle accepte 15 ports au maximum, et ne peut être employée qu'avec ***–p tcp*** et ***–p udp***. Notez que la correspondance ***––port*** ne peut sélectionner que les paquets qui viennent et vont vers le même port, par exemple, du port 80 au port 80, du port 110 au port 110, etc.

10.3.12. Correspondance owner

L'extension de correspondance ***owner*** permet de sélectionner des paquets à partir de l'identité du processus qui les a créés. Le propriétaire ("***owner***") peut être spécifié comme étant l'identifiant de l'utilisateur qui a lancé la commande en question, de son groupe, du processus, de la session, ou bien de la commande elle–même. A l'origine, cette extension a été écrite pour donner un exemple des utilisations possibles d'***iptables***. La correspondance ***owner*** fonctionne seulement dans la chaîne `OUTPUT` pour des raisons évidentes : il est presque impossible d'extraire des éléments d'information sur l'identité de l'instance ayant envoyé un paquet à partir de l'autre extrémité, et où a lieu le saut intermédiaire avant la destination finale. Et même dans la chaîne `OUTPUT`, ce n'est pas vraiment fiable puisque certains paquets peuvent ne pas avoir de propriétaire. Les célèbres paquets de ce genre sont, entre–autres, les `réponses ICMP`. Donc les `réponses ICMP` ne correspondront jamais.

Tableau 10.18. Options de la correspondance owner

Correspondance	***––uid–owner***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A OUTPUT –m owner ––uid–owner 500***
Explication	Avec cette correspondance, le paquet est sélectionné s'il a été créé par l'`identifiant d'utilisateur` (UID) donné. Ceci permet d'établir une correspondance avec les paquets sortants basée sur celui qui les a créés. Une utilisation possible serait d'empêcher tout utilisateur autre que root d'ouvrir de nouvelles connexions extérieures au pare–feu. Une autre possibilité pourrait être d'empêcher tout le monde sauf l'utilisateur `http` d'envoyer des paquets à partir du port `HTTP`.
Correspondance	***––gid–owner***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A OUTPUT –m owner ––gid–owner 0***

Explication	Cette correspondance permet de sélectionner des paquets à partir de leur `identifiant de groupe` (GID). Ainsi on établit une correspondance avec tous les paquets associés au groupe auquel appartient l'utilisateur ayant créé le paquet. Par exemple, ceci permet d'empêcher tous les utilisateurs sauf ceux appartenant au groupe `network` de naviguer sur Internet, ou comme précédemment, d'autoriser seulement les membres du groupe `http` à créer des paquets sortants par le port `HTTP`.
Correspondance	***--pid-owner***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A OUTPUT -m owner --pid-owner 78***
Explication	Cette correspondance permet de sélectionner des paquets à partir de l'`identifiant de processus` (PID) qui est responsable d'eux. Cette correspondance est un peu plus difficile à utiliser, mais il est possible, par exemple, d'autoriser seulement le `PID` 94 à envoyer des paquets par le port `HTTP` (si le processus `HTTP` n'est pas un fil d'exécution, bien sûr). Une autre alternative serait d'écrire un petit script qui récupère le `PID` à partir de la sortie d'une commande ***ps*** pour un "daemon" spécifique, et qui ajoute ensuite une règle pour le numéro récupéré. Pour donner un exemple, vous pouvez élaborer une règle comme celle présente dans l'exemple .
Correspondance	***--sid-owner***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A OUTPUT -m owner --sid-owner 100***
Explication	Cette correspondance permet de sélectionner des paquets à partir de l'`identifiant de session` (SID) utilisé par le programme en question. La valeur du `SID` d'un processus est celle du processus lui-même et de tous les processus découlant du processus d'origine. Ces derniers peuvent être un fil d'exécution ("thread"), ou un processus fils du processus d'origine. Donc par exemple, tous les processus `HTTPD` devraient posséder le même `SID` que leur processus parent (le processus `HTTPD` d'origine), si le `HTTPD` appartient à un fil d'exécution (comme la plupart des processus `HTTPD`, Apache et Roxen par exemple). Ceci est illustré par un petit script qui s'appelle Sid-owner.txt. Celui-ci pourrait éventuellement être lancé toutes les heures et enrichi de code supplémentaire pour vérifier que l'exécution du processus `HTTPD` est toujours en cours et le redémarrer sinon, avant de vider et redéfinir la chaîne `OUTPUT` si nécessaire.

10.3.13. Correspondance type de paquet

Cette correspondance sert à apparier les paquets basés sur leur type. C'est à dire, ceux destinés à une personne précise, à tout le monde ou à un groupe de machines spécifique ou encore des utilisateurs. Ces trois groupes sont généralement appelés unicast, broadcast et multicast, comme nous l'avons vu dans le chapitre Rappel TCP/IP. La correspondance est chargée par la commande : ***-m pkttype***.

Tableau 10.19. Options de correspondance type de paquet

Correspondance	***--pkttype***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A OUTPUT -m owner --pkttype unicast***
Explication	La correspondance ***--pkttype*** sert à indiquer quel type de paquet apparier. Elle peut prendre soit `unicast`, `broadcast` ou `multicast` comme argument, voir l'exemple. Elle peut aussi être inversée par ***!*** comme ceci : ***-m pkttype --pkttype ! broadcast***, qui va apparier tous les autres types de paquets.

10.3.14. Correspondance Recent

La correspondance Recent est une système plutôt important et complexe, qui nous permet d'apparier des paquets basés sur les événements récents. Exemple, si nous voulons voir une sortie de connexion IRC, nous pouvons placer l'adresse IP dans une liste d'hôtes, et avoir une autre règle qui permet des requêtes d'identification en retour d'un serveur IRC dans les 15 secondes de veille du paquet d'origine.

Avant nous pouvons avoir une vue plus précise de cette correspondance, et voyons comment elle fonctionne. En premier, nous utilisons différentes règles pour utiliser l'appariement récent. Celui−ci se sert de plusieurs listes d'événements récents. Par défaut la liste utilisée est DEFAULT. Nous créons une nouvelle entrée dans la liste avec cette option, ainsi une fois qu'une règle est complètement appariée (l'option placée est toujours une correspondance), nous pouvons ajouter une entrée dans la liste récente spécifiée. L'entrée de la liste contient un horodatage, et l'adresse source IP utilisée dans le paquet qui déclenche l'option. Une fois ceci fait, nous pouvons utiliser une série d'options différentes pour apparier cette information, comme la mise à jour des entrées d'horodatage, etc.

Enfin, si nous voulons pour quelque raison supprimer une entrée de la liste, nous pouvons le faire en supprimant la correspondance du module récent. Toutes les règles utilisant la correspondance Recent, doivent charger ce module (***−m recent***). Voyons en les options.

Tableau 10.20. Options de la correspondance Recent

Correspondance	***−−name***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A OUTPUT −m recent −−name examplelist***
Explication	L'option "name" donne le nom de la liste à utiliser. Par défaut la liste DEFAULT est utilisée, ce qui n'est probablement pas ce que nous voulons si nous nous servons de plus d'une liste.
Correspondance	***−−set***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A OUTPUT −m recent −−set***
Explication	Ceci crée une nouvelle entrée dans la liste récente, qui contient un horodatage et l'adresse source IP de l'hôte qui a déclenché la règle.
Correspondance	***−−rcheck***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A OUTPUT −m recent −−name examplelist −−rcheck***
Explication	L'option ***−−rcheck*** vérifie si l'adresse IP source du paquet est dans la liste nommée. Si c'est le cas, la correspondance renvoit un "vrai", dans le cas contraire elle renverra un "faux". Cette option peut être inversée avec le signe !. Dans ce dernier cas, elle renverra vrai si l'adresse IP source n'est pas dans la liste, et faux si elle est dans la liste.
Correspondance	***−−update***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A OUTPUT −m recent −−name examplelist −−update***
Explication	Cette correspondance est vraie si la source est disponible dans la liste spécifiée et met à jour le dernier horodatage dans la liste. Elle peut aussi être inversée par le ! devant le module. Exemple, ***! −−update***.

Correspondance	***−−remove***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−remove***
Explication	Cette correspondance essaie de trouver l'adresse source du paquet dans la liste, et renvoit un vrai si le paquet est présent. Elle supprimera aussi l'entrée de liste correspondante de la liste. Cette commande peut être inversée avec le signe ***!***.
Correspondance	***−−seconds***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−check −−seconds 60***
Explication	Cette correspondance n'est valide seulement qu'avec les commandes ***−−check*** et ***−−update***. Le module ***−−seconds*** est utilisé pour spécifier le délai de mise à jour de la colonne "dernier apperçu" dans la liste récente. Si la colonne dernier apperçu est plus ancienne qu'un certain nombre de secondes, la correspondance renvoit faux. Si la correspondance récent fonctionne anormalement, l'adresse source doit toujours être dans la liste pour un retour vrai de la correspondance.
Correspondance	***−−hitcount***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−check −−hitcount 20***
Explication	La correspondance ***−−hitcount*** doit être utilisée avec les commandes ***−−check*** ou ***−−update***, elle limitera la vérification aux seuls paquets vus par le compteur. Si cette correspondance est utilisée avec la commande ***−−seconds***, cela nécessite que le compteur de paquets spécifié soit vu dans le bloc de temps. Elle peut être inversée par le signe ***!*** devant la commande. Avec la commande ***−−seconds***, elle indique le maximum de paquets qui peuvent avoir été vus durant le bloc de temps spécifié. Si les deux correspondances sont inversées, alors un maximum de paquets peuvent avoir été vus durant le dernier minimum de secondes.
Correspondance	***−−rttl***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−check −−rttl***
Explication	La correspondance ***−−rttl*** vérifie que la valeur TTL du paquet est la même que celle du paquet original utilisé pour placer l'entrée dans la liste récente. Ceci peut être utilisé pour vérifier que les adresses sources de personnes n'ont pas été mystifiées (spoofing) pour interdire aux autres l'accès à leur serveurs en faisant usage de la correspondance recent.
Correspondance	***−−rsource***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−rsource***
Explication	La correspondance ***−−rsource*** indique au module recent de sauvegarder l'adresse source et les ports dans la liste recent. C'est le comportement par défaut.
Correspondance	***−−rdest***
Noyau	2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m recent −−name example −−rdest***

Explication	***--rdest*** est l'opposé de ***--rsource*** en ce qu'elle indique à la correspondance recent d'enregistrer l'adresse et le port de destination dans la liste recent.

j'ai créé un petit exemple de script sur la façon d'utiliser la correspondance recent, vous pouvez le trouver dans la section Recent-match.txt.

En bref, c'est une pauvre variation de la machine d'état disponible dans netfilter. Cette version fut créé avec à l'esprit un serveur http, mais qui fonctionnera avec n'importe quelle connexion TCP. En premier, nous avons créé deux chaînes nommées http-recent et http-final. La chaîne http-recent est utilisée aux étapes du démarrage de la connexion, et pour la transmission des données, tandis que la chaîne http-final est utilisée pour les derniers FIN, FIN/ACK dans l'établissement de la liaison.

Avertissement

C'est une très mauvaise alternative pour la machine d'état et elle ne dispose pas de toutes les possibilités de la machine d'état. Cependant, c'est un bon exemple de ce qui peut être fait avec la correspondance Recent sans être trop spécifique. N'utilisez pas cet exemple en production. Il est lent, gère mal les cas spéciaux, et ne doit être jamais utilisé que comme un exemple.

Par exemple, il ne gère pas les ports fermés dans une connexion, les établissements de liaison FIN asynchrones (où une des parties connectée se ferme, tandis que l'autre continue d'envoyer des données), etc.

Suivons un paquet à travers l'exemple de la table de règles. D'abord le paquet entre dans la chaîne `INPUT`, et nous l'envoyons à la chaîne `http-recent`.

1. Le premier paquet sera un paquet SYN, et n'aura pas de bit ACK, FIN ou RST placé. Il est apparié en utilisant la ligne ***--tcp-flags SYN,ACK,FIN,RST SYN***. À ce niveau nous ajoutons la connexion à httplist avec la ligne ***-m recent --name httplist --set***. Enfin nous acceptons le paquet.
2. Après le premier paquet nous recevons un paquet SYN/ACK indiquant que le paquet SYN a été reçu. Ceci peut être apparié en utilisant la ligne ***--tcp-flags SYN,ACK,FIN,RST SYN,ACK***. FIN et RST sont illégaux à ce niveau. Nous mettons à jour l'entrée dans httplist par ***-m recent --name httplist --update*** et finalement nous avons l'ACCEPT du paquet.
3. Maintenant nous obtenons un paquet final ACK, venant du créateur de la connexion, nous permettant de savoir que le SYN/ACK a été envoyé par le serveur. SYN, FIN et RST sont illégaux à ce point de la connexion, et la ligne ressemblera à ***--tcp-flags SYN,ACK,FIN,RST ACK***. Nous mettons à jour la liste de la même façon que dans l'étape précédente, et nous avons l'ACCEPT.
4. À ce niveau, la transmission de données peut démarrer. La connexion ne contiendra jamais aucun paquet SYN maintenant, mais contiendra des paquets ACK pour permettre de savoir que les données sont envoyées. Chaque fois que nous voyons un paquet comme celui-là, nous mettons à jour la liste et ACCEPT les paquets.
5. La transmission peut être terminée de deux façons, la plus simple est le paquet RST. RST réinitialisera la connexion et la coupera. Avec FIN, la connexion sera coupée sans plus envoyer de données. Le destinataire du FIN, pourra toujours envoyer des données, et nous arrivons à l'étape finale de la connexion.
6. Dans les chaînes http-recent-final nous vérifions si le paquet est toujours dans la httplist, et si c'est le cas, nous l'envoyons à la chaîne http-recent-final1. Dans cette chaîne nous supprimons la connexion de la httplist l'ajoutons à la liste http-recent-final. Si la connexion a déjà été supprimée et déplacée vers la liste http-recent-final, nous envoyons le paquet vers la chaîne http-recent-final2.
7. Dans la chaîne http-recent-final2, nous attendons que la partie non fermée finisse d'envoyer ses données, et fermons ensuite la connexion. Une fois ceci fait, la connexion est tout simplement supprimée.

Comme nous l'avons vu la liste recent peut devenir tout à fait complexe, mais elle nous donne un vaste éventail de possibilités si nécessaire. Encore une fois, nous ne réinventons pas la roue. Si la fonctionnalité que vous désirez est déjà implémentée, utilisez la au lieu d'essayer de créer votre propre solution.

10.3.15. Correspondance state

L'extension de correspondance ***state*** est associée au code de traçage de connexion dans le noyau. La correspondance d'état accède à l'état du traçage de connexion des paquets grâce à la machine de "conntracking". Elle permet de savoir dans quel état se trouve la connexion, et fonctionne pour quasiment tous les protocoles y−compris les protocoles sans état tels que `ICMP` et `UDP`. Dans tous les cas, la connexion est sujette à un dépassement de temps établi par défaut ("default timeout") et sera, le cas échéant, supprimée de la base de données du traçage de connexion. Cette correspondance exige d'être chargée explicitement en ajoutant la directive ***−m state*** à la règle. Vous disposerez alors d'une nouvelle correspondance appelée ***state***. Le concept de correspondance d'état est couvert plus en détail dans le chapitre La machine d'état, étant donné que le sujet est assez vaste.

Tableau 10.21. Correspondances de state

Correspondance	***−−state***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −m state −−state RELATED,ESTABLISHED***
Explication	Cette option indique à la correspondance ***state*** dans quels états doivent être les paquets pour être sélectionnés. Actuellement, 4 états sont disponibles : ***INVALID***, ***ESTABLISHED***, ***NEW*** et ***RELATED***. ***INVALID*** signifie que le paquet n'est associé à aucun flux, ni à aucune connexion connus, et qu'il peut contenir des données ou des en−têtes erronés. ***ESTABLISHED*** signifie que le paquet est lié à une connexion déjà établie, qui a vu passer des paquets dans les deux directions et qui est considérée valide. ***NEW*** signifie que le paquet a démarré ou démarrera une nouvelle connexion, ou bien qu'il est associé à une connexion qui n'a pas vu passer des paquets dans les deux directions. Enfin, ***RELATED*** signifie que le paquet démarre une nouvelle connexion et qu'il est associé à une connexion déjà établie. Ceci peut évoquer par exemple un `transfert de données par FTP`, ou une `erreur ICMP` associée à une connexion `TCP` ou `UDP`. Notez que l'état ***NEW*** n'examine pas les bits `SYN` des paquets `TCP` qui tentent de démarrer une nouvelle connexion. Par conséquent, cet état ne devrait pas être utilisé tel quel dans les situations où il n'existe qu'un seul pare−feu, ou quand il n'y a pas d'équilibrage de charge entre les différents pare−feux. Cependant, cet état se révèle utile dans certains cas. Pour en savoir plus, consultez le chapitre La machine d'état.

10.3.16. Correspondance TCPMSS

La correspondance ***tcpmss*** est utilisée pour apparier un paquet basé sur la `Maximum Segment Size` (Tailles maximum de segment) dans `TCP`. Ceci vérifie seulement la validité des paquets `SYN` et `SYN/ACK`. Pour une explication plus détaillée de la valeur MSS, voir l'appendice, Options TCP la RFC 793 – Transmission Control Protocol et la RFC 1122 – Requirements for Internet Hosts – Communication Layers. Cette correspondance est chargée en utilisant ***−m tcpmss*** et prend uniquement cette option.

Tableau 10.22. Options de correspondance TCPMSS

Correspondance	***−−mss***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables −A INPUT −p tcp −−tcp−flags SYN,ACK,RST SYN −m tcpmss −−mss 2000:2500***
Explication	L'option ***−−mss*** indique à la correspondance tcpmss quel `Maximum Segment Sizes` apparier. Ceci peut être soit une simple valeur MSS, soit une plage de valeurs MSS séparées par ***:***. La valeur peut être inversée par le signe ***!***, comme dans l'exemple suivant :

	–m tcpmss ! ––mss 2000:2500 Cet exemple vérifiera toutes les valeurs MSS, sauf les valeurs comprises dans la plage de 2000 à 2500.

10.3.17. Correspondance TOS

La correspondance ***TOS*** peut servir à sélectionner les paquets à partir de leur `champ de TOS`. `TOS` signifie `type de service` ; il est constitué de 8 bits et se situe dans l'en–tête IP. Cette correspondance est chargée explicitement en ajoutant ***–m tos*** à la règle. Elle est normalement utilisée afin d'informer les hôtes intermédiaires de l'ordre de priorité du flux et de son contenu (ce n'est pas vraiment le cas, mais il informe des besoins spécifiques au flux, comme une réexpédition aussi rapide que possible, ou un impératif de débit). Les différents routeurs et administrateurs gèrent ces valeurs de façon variable. La plupart ne s'en préoccupent pas du tout, alors que d'autres font de leur mieux pour servir les paquets en question et les données qu'ils contiennent.

Tableau 10.23. Correspondance TOS

Correspondance	***––tos***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables –A INPUT –p tcp –m tos ––tos 0x16***
Explication	Cette correspondance s'utilise tel que décrit ci–dessus. Elle sélectionne les paquets à partir de leur `champ de TOS` et de sa valeur. Ceci peut être employé avec le programme ***iproute2*** et les fonctions avancées de routage de Linux pour effectuer un ***marquage*** des paquets pour une usage ultérieur. La correspondance prend en option une valeur hexadécimale ou numérique, ou éventuellement un des noms fournis par la commande '***iptables –m tos –h***'. Actuellement, elle donne les noms suivants : `Minimize-Delay 16 (0x10)`, `Maximize-Throughput 8 (0x08)`, `Maximize-Reliability 4 (0x04)`, `Minimize-Cost 2 (0x02)`, et `Normal-Service 0 (0x00)`. `Minimize-Delay` signale de minimiser le retard pour les paquets qui traversent – les services classiques qui requièrent ceci peuvent être, par exemple, `telnet`, `SSH` et `FTP-control`. `Maximize-Throughput` précise de trouver un chemin qui offre le plus haut débit possible – un protocole typique est `FTP-data`. `Maximize-Reliability` indique de maximiser la fiabilité de la connexion, donc d'utiliser des lignes aussi fiables que possible – deux exemples typiques sont `BOOTP` et `TFTP`. `Minimize-Cost` signale de minimiser le coût des paquets qui traversent tous les liens vers le client ou le serveur ; par exemple, déterminer la route qui offre le voyage le moins cher de bout en bout. Des exemples de protocoles classiques qui peuvent l'utiliser sont `RTSP` ("Real Time Stream Control Protocol" ou protocole de contrôle de flux temps–réel) et d'autres protocoles de flux vidéo/radio. Enfin, `Normal-Service` désigne tout protocole classique n'ayant aucun besoin particulier.

10.3.18. Correspondance TTL

La correspondance ***TTL*** permet de sélectionner les paquets à partir de leur champ `TTL` ("Time To Live" ou durée de vie) localisé dans l'en–tête IP. Le `champ TTL` contient 8 bits de données, et il est décrémenté de 1 à chaque fois qu'il est traité par un hôte intermédiaire entre le client et l'hôte destinataire. Si le `TTL` atteint 0, un message `ICMP` type 11 code 0 (TTL égal à 0 pendant le transit) ou code 1 (TTL égal à 0 pendant le réassemblage) est transmis à l'expéditeur du paquet pour l'informer du problème. Cette correspondance est utilisée seulement pour sélectionner les paquets à partir de leur `TTL`, et non pour effectuer un changement quel qu'il soit. Celui–ci, soit dit en passant, s'applique à tout type de correspondance. Pour charger cette correspondance, vous devez ajouter ***–m ttl*** à la règle.

Tableau 10.24. Correspondances TTL

Correspondance	***--ttl***
Noyau	2.3, 2.4, 2.5 et 2.6
Exemple	***iptables -A OUTPUT -m ttl --ttl 60***
Explication	Cette option de correspondance permet de spécifier la valeur `TTL` à sélectionner. Cette option requiert une valeur numérique et établit une correspondance avec cette valeur dans le paquet. Aucune inversion n'est disponible et il n'y a rien d'autre, en particulier, à sélectionner. Mais ceci peut être utile, par exemple, pour déboguer votre réseau local – c'est-à-dire les hôtes de votre LAN qui semblent présenter des problèmes de connexion avec un hôte sur Internet – ou pour trouver d'éventuelles entrées de chevaux de Troie, etc. Les possibilités de cette option sont relativement limitées, cependant son intérêt dépend essentiellement de votre imagination. Un exemple pourrait être de trouver des hôtes avec de mauvaises valeurs par défaut de `TTL` (pouvant être la conséquence d'un pile `TCP/IP` mal implémentée, ou simplement d'un défaut de configuration).

10.3.19. Correspondance unclean

La correspondance ***unclean*** ne prend aucune option et ne nécessite rien de plus qu'un chargement explicite si vous souhaitez l'utiliser. Notez que cette option est considérée comme expérimentale, qu'elle peut ne pas fonctionner en toutes circonstances et qu'elle ne prendra pas en charge tous les paquetages ou problèmes relatifs à unclean. La correspondance ***unclean*** tente de sélectionner les paquets qui paraissent malformés ou inhabituels, comme des paquets avec des en-têtes ou des sommes de contrôle ("checksums") erronés. Elle peut être utilisée pour rejeter des connexions (avec la cible ***DROP***) et pour rechercher les flux douteux, par exemple. Cela dit, vous devez être conscient qu'il existe un risque d'interruption de connexions saines.

Chapitre 11. Iptables cibles et sauts

Target/jump indique à la règle que faire avec un paquet qui est parfaitement apparié avec la section correspondance de la règle. Il existe deux cibles de base, les cibles ***ACCEPT*** et ***DROP***, que nous verrons en premier. Cependant, avant, jetons un bref regard sur la façon dont un saut est construit.

La spécification saut est faite exactement de la même façon que la définition cible, sauf qu'elle nécessite une chaîne dans la même table. Pour faire un saut vers une chaîne spécifique, il faut bien sûr que la chaîne existe. Comme nous l'avons déjà expliqué, une chaîne définie par l'utilisateur est créée avec la commande ***-N***. Par exemple, nous créons une chaîne dans la table filtre appelée ***tcp_packets***, comme ceci :

```
iptables -N tcp_packets
```

Nous pouvons alors lui ajouter une cible saut comme :

```
iptables -A INPUT -p tcp -j tcp_packets
```

Nous pourrons alors faire un saut depuis la chaîne ***INPUT*** vers la chaîne ***tcp_packets*** et commencer à traverser la chaîne. Quand nous atteignons la fin de cette chaîne, nous retournons vers la chaîne ***INPUT*** et le paquet démarre sa traversée de la règle une étape après qu'il ait fait le saut vers l'autre chaîne (tcp_packets dans ce cas). Si le paquet est ***ACCEPT*** dans une des sous-chaînes, elle sera ***ACCEPT*** dans la chaîne de sur-ensemble également et ne traversera plus aucune des chaînes de sur-ensemble. Cependant, notez que le paquet traversera toutes les autres chaînes des autres tables. Pour plus d'information sur la traversée des tables et des chaînes, voir le chapitre Traversée des tables et des chaînes.

D'un autre côté, les cibles spécifient une action a effectuer sur le paquet en question. Nous pouvons, par exemple, ***DROP*** ou ***ACCEPT*** selon ce que nous voulons faire. Il existe aussi plusieurs autres actions que nous pouvons effectuer, que nous décrirons plus tard dans cette section. Certaines cibles stopperont le paquet dans sa traversée des chaînes, comme décrit au−dessus. De bons exemples de ces règles sont ***DROP*** et ***ACCEPT***. les règles qui sont stoppées, ne passeront plus à travers aucune règle suivante sur la chaîne ou sur une chaîne supérieure. D'autres cibles, peuvent avoir une action sur le paquet, lequel ensuite continuera à traverser les règles suivantes. Un bon exemple de ceci peuvent être les cibles ***LOG***, ***ULOG*** et ***TOS***. Ces cibles peuvent journaliser les paquets, les analyser et les passer à d'autres règles dans le même ensemble de chaînes. Nous pouvons, par exemple, de plus vouloir analyser les valeurs TTL et TOS d'un paquet/flux spécifique. Certaines cibles accepterons des options supplémentaires (quelle valeur TOS utiliser, etc.), tandis que d'autres n'en ont pas nécessairement besoin − mais peuvent en inclure si nous le souhaitons (journaliser les préfixes, masquer les ports, etc.). Nous essaierons de couvrir tous ces sujets quand nous verrons la description des cibles. Regardons de quelles sortes de cible il s'agit.

11.1. Cible ACCEPT

Cette cible ne nécessite pas d'autre option. Aussitôt que la spécification de correspondance pour un paquet a été pleinement satisfaite, et que nous spécifions ACCEPT comme cible, la règle est acceptée et ne traversera pas la chaîne ou aucune autre chaîne dans la même table. Notez cependant, qu'un paquet qui a été accepté dans une chaîne peut toujours circuler à travers les chaînes dans d'autres tables, et peut toujours être supprimé à cet endroit là. Il n'y a rien de spécial concernant cette cible, et il n'est pas nécessaire d'y ajouter des options. Pour utiliser cette cible, spécifiez simplement ***–j ACCEPT***.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.2. Cible CLASSIFY

la cible ***CLASSIFY*** peut servir à classer les paquets de façon à ce qu'ils puissent être utilisés par deux ou plusieurs qdiscs (Queue Disciplines). Par exemple, atm, cbq, dsmark, pfifo_fast, htb. Pour plus d'information sur qdiscs et le contrôle de trafic, voir la page Linux Advanced Routing and Traffic Control HOW−TO.

La cible ***CLASSIFY*** est valide uniquement dans la chaîne `POSTROUTING` de la table `mangle`.

Tableau 11.1. Options de la cible CLASSIFY

Option	***−−set−class***
Exemple	***iptables −t mangle −A POSTROUTING −p tcp −−dport 80 −j CLASSIFY −−set−class 20:10***
Explication	La cible ***CLASSIFY*** prend seulement un argument, le ***−−set−class***. Il indique à la cible comment classer le paquet. Ce classement prend deux valeurs séparées par le signe deux points (:), comme ceci MAJOR:MINOR. Encore une fois, si vous voulez plus d'information, regardez la page Linux Advanced Routing and Traffic Control HOW−TO.

Note

Fonctionne avec les noyaux Linux 2.5 et 2.6.

11.3. Cible DNAT

La cible ***DNAT*** est utilisée pour la Traduction d'Adresse Réseau de Destination, ce qui veut dire qu'elle sert à réécrire l'adresse `IP de Destination` du paquet. Si un paquet est apparié, et qu'il est la cible de la règle, ce paquet et tous les paquets suivants du même flux seront traduits, et ensuite routés vers le matériel, l'hôte ou le réseau appropriés. Cette cible peut être extrêmement utile, par exemple, quand vous avez un hôte avec un

serveur web dans un *LAN*, mais pas d'IP réelle routable sur l'Internet. Vous pouvez alors indiquer au pare−feu de transférer tous les paquets allant vers son propre port HTTP, vers le serveur web réel dans le *LAN*. Vous pouvez aussi spécifier une plage d'adresses IP de destination, et le mécanisme ***DNAT*** choisira l'adresse IP de destination au hasard pour chaque flux. Nous pourrons donc réaliser une sorte d'équilibrage de charge en faisant ça.

Notez que la cible ***DNAT*** est disponible uniquement dans les chaînes `PREROUTING` et `OUTPUT` de la table nat. Les chaînes contenant des cibles ***DNAT*** ne peuvent pas être utilisées depuis d'autres chaînes, comme la chaîne `POSTROUTING`.

Tableau 11.2. Cible DNAT

Option	***−−to−destination***
Exemple	***iptables −t nat −A PREROUTING −p tcp −d 15.45.23.67 −−dport 80 −j DNAT −−to−destination 192.168.1.1−192.168.1.10***
Explication	L'option ***−−to−destination*** indique au mécanisme `DNAT` quelle `Destination IP` placer dans l'en−tête IP, et où sont envoyés les paquets qui sont appariés. L'exemple ci−dessus enverra sur tous les paquets destinés à l'adresse IP 15.45.23.67 dans une plage IP de réseau local comprise entre 192.168.1.1 jusqu'à 192.168.1.10. Notez que, comme décrit précédemment, un simple flux utilisera toujours le même hôte, et chaque flux aura une adresse IP attribuée au hasard, et qui sera toujours en direction de quelque part, dans ce flux. Nous pouvons aussi avoir à spécifier une seule adresse IP, dans ce cas nous serons toujours connectés au même hôte. Notez aussi que nous pouvons ajouter un port ou une plage de ports vers lequel le trafic sera redirigé. Ceci se fait en ajoutant, par exemple, un :80 à l'adresse IP pour laquelle nous voulons traduire les paquets. Une règle peut alors ressembler à ***−−to−destination 192.168.1.1:80*** par exemple, ou ***−−to−destination 192.168.1.1:80−100*** si nous voulons spécifier une plage de ports. Comme vous pouvez le voir, la syntaxe est à peu près la même que la cible ***SNAT***, même si elles font deux choses totalement différentes. Les spécifications de port sont valides uniquement pour les règles qui précisent les protocoles TCP ou UDP avec l'option ***−−protocol***.

Comme ***DNAT*** nécessite pas mal de travail pour fonctionner correctement, j'ai décidé d'ajouter une explication plus complète sur ce sujet. Prenons un bref exemple pour comprendre comment les choses se passent normalement. Nous voulons publier notre site web via notre connexion Internet. Nous ne possédons qu'une seule adresse IP, et le serveur HTTP est situé dans notre réseau interne. Notre pare−feu possède l'adresse IP externe ***$INET_IP***, et notre serveur HTTP a l'adresse IP interne ***$HTTP_IP*** et enfin le pare−feu a l'adresse IP interne ***$LAN_IP***. La première chose à faire est d'ajouter la simple règle suivante à la chaîne PREROUTING dans la table nat :

```
iptables −t nat −A PREROUTING −−dst $INET_IP −p tcp −−dport 80 −j DNAT \
−−to−destination $HTTP_IP
```

Maintenant, tous les paquets provenant de l'Internet et allant vers le port 80 sur notre pare−feu, sont redirigés (ou ***DNAT***és) vers notre serveur HTTP interne. Si vous testez ceci depuis L'Internet, tout devrait fonctionner parfaitement. mais, que se passe−t−il si vous essayez de vous connecter depuis un hôte sur le même réseau local que le serveur HTTP ? Il ne fonctionnera tout simplement pas. C'est un réel problème avec le routage. Commençons par voir ce qui se passe dans un cas normal. La machine externe possède une adresse IP ***$EXT_BOX***, pour conserver la lisibilité.

1. Le paquet quitte l'hôte connecté allant vers ***$INET_IP*** et la source ***$EXT_BOX***.
2. Le paquet atteint le pare−feu.
3. Le pare−feu ***DNAT*** le paquet et envoit celui−ci à travers les différentes chaînes, etc.
4. Le paquet quitte la pare−feu pour aller vers le $HTTP_IP.

5. Le paquet atteint le serveur HTTP, et la machine HTTP répond en retour à travers le pare−feu, si c'est cette machine que la base de routage a entré comme passerelle pour ***$EXT_BOX***. Normalement, ça devrait être la passerelle par défaut du serveur HTTP.
6. Le pare−feu Un−***DNAT*** le paquet de nouveau, ainsi le paquet semble provenir du pare−feu lui−même.
7. Le paquet en réponse transite vers le client ***$EXT_BOX***.

Maintenant, voyons ce qui se passe si le paquet est généré par un client sur le même réseau que le serveur HTTP lui−même. Le client possède l'adresse IP ***$LAN_BOX***, tandis que les autres machines ont les mêmes réglages.

1. Le paquet quitte la ***$LAN_BOX*** vers ***$INET_IP***.
2. Le paquet atteint le pare−feu.
3. Le paquet est ***DNAT***é, et toutes les autres actions requises sont prises, cependant, le paquet n'est pas ***SNAT***é, ainsi la même adresse source IP est utilisée pour le paquet.
4. le paquet quitte le pare−feu et atteint le serveur HTTP.
5. Le serveur HTTP essaie de répondre au paquet, et voit dans les tables de routage que le paquet provient d'une machine locale sur le même réseau, et donc tente d'envoyer le paquet directement à l'adresse source IP d'origine (qui devient alors l'adresse IP de destination).
6. Le paquet atteint le client, et le client est dans la confusion car le paquet en retour ne provient pas de l'hôte qui a envoyé la requête d'origine. Donc, le client supprime le paquet, et attend une réponse "réelle".

La solution la plus simple à ce problème est de ***SNAT***er tous les paquets entrant dans le pare−feu sortant vers un hôte ou une IP sur lequel nous faisons du ***DNAT***. Exemple, regardons la règle ci−dessus. Nous SNATons les paquets entrants dans notre pare−feu qui sont destinés à ***$HTTP_IP*** port 80 et ainsi il est vu que des paquets proviennent d'une ***$LAN_IP***. Ceci force le serveur HTTP à envoyer ces paquets vers notre pare−feu, lequel Un−***DNAT*** ceux−ci et les envoit au client. La règle ressemble à ceci :

```
iptables -t nat -A POSTROUTING -p tcp --dst $HTTP_IP --dport 80 -j SNAT \
--to-source $LAN_IP
```

Souvenez vous que la chaîne POSTROUTING est exécutée en dernier, et donc le paquet sera déjà ***DNAT***é une fois qu'il joint cette chaîne spécifique. C'est la raison pour laquelle nous apparions les paquets basés sur une adresse interne.

Avertissement

Cette dernière règle nuira sérieusement à votre journalisation, ainsi il n'est pas recommandé d'utiliser cette méthode, mais l'ensemble de l'exemple est valide. Que se passe−t−il alors, le paquet provient de l'Internet, est SNATé et DNATé et finalement atteint le serveur HTTP (par exemple). Le serveur HTTP voit maintenant les requêtes comme si elles provenaient du pare−feu, et donc les journalisera *toutes* comme telles.

Ceci peut avoir également d'autres implications plus graves. Prenons un serveur SMTP sur un LAN, qui autorise les requêtes depuis le réseau interne, et vous avez un pare−feu paramétré pour transférer le trafic SMTP vers ce serveur. Vous avez donc créé un serveur SMTP en relais ouvert, avec une journalisation horrible !

Une solution à ce problème est de tout simplement rendre la règle ci−dessus plus précise dans sa partie appariement, et de travailler seulement sur les paquets qui proviennent du LAN. En d'autres termes, ajoutez un ***−i $LAN_IFACE*** à l'ensemble de la commande. Ceci fera que la règle ne fonctionnera que sur les flux provenant du LAN, et donc n'affectera pas la source IP, ainsi les journaux seront corrects, sauf pour les flux venant du LAN.

Vous auriez mieux fait, en d'autres termes, de résoudre ces problèmes soit en paramétrant un serveur DNS (serveur de nom) séparé pour votre LAN, soit en paramétrant une DMZ séparée, la dernière étant préférable si vous avez les moyens.

Vous pouvez penser que c'est suffisant, et c'est vrai, sauf à considérer un dernier aspect du scénario. Que se passe–t–il si le pare–feu lui–même essaie d'accéder au serveur HTTP, où va–t–il ? Il tentera malheureusement d'accéder à son propre serveur HTTP, et pas au serveur situé sur ***$HTTP_IP***. Pour parer à ça, nous devons rajouter une règle ***DNAT*** à la chaîne `OUTPUT`. Suivant l'exemple ci–dessus, ça ressemblerait à quelque chose comme :

```
iptables -t nat -A OUTPUT --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

Tous les réseaux séparés qui ne sont pas situés sur le même réseau que le serveur HTTP fonctionneront sans soucis, tous les hôtes sur le même réseau que le serveur HTTP pourront s'y connecter et enfin, le pare–feu pourra exécuter ses connexions correctement. Maintenant, tout fonctionne et aucun problème ne devrait arriver.

Note

Tout le monde devrait réaliser que ces règles affectent seulement la façon dont le paquet est DNATé et SNATé. En plus de ces règles, nous avons aussi besoin de règles supplémentaires dans la table filtre (chaîne FORWARD) pour permettre aux paquets de traverser ces chaînes. N'oubliez pas que tous les paquets sont déjà passés par la chaîne PREROUTING, et donc ont vu leur adresse de destination réécrite par DNAT.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.4. Cible DROP

La cible ***DROP*** fait exactement ce qu'elle veut dire, elle efface des paquets et n'effectue aucun autre processus supplémentaire. Un paquet qui apparie parfaitement un règle et est ensuite effacé sera bloqué. Notez que cette action peut avoir, dans certains cas, des effets inattendus, car elle peut laisser des interfaces de connexions mortes sur quelque hôte. Une meilleure solution dans ces cas là serait d'utiliser la cible ***REJECT***, spécialement quand vous voulez bloquer le balayage (scan) de ports pour ne pas donner trop d'informations, ou le filtrage de ports, etc. Notez également que si le paquet subit l'action ***DROP*** dans une sous–chaîne, ce paquet ne sera traité dans aucune des chaînes principales, soit dans la table présente ou dans une quelconque autre table. Le paquet est, en d'autres termes, totalement mort. Comme nous l'avons vu précédemment, la cible n'enverra aucune autre sorte d'information dans aucune direction, ni par des intermédiaires comme les routeurs.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.5. Cible DSCP

C'est une cible qui modifie les repères DSCP (Differentiated Services Field) dans un paquet. La cible ***DSCP*** peut placer n'importe quelle valeur DSCP dans un paquet TCP, ce qui est un moyen d'indiquer aux routeurs la priorité du paquet en question. Pour plus d'information sur DSCP, voyez la RFC RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.

De façon basique, DSCP est un moyen de différencier divers services de catégories séparées, et leur donner différentes priorités à travers les routeurs. De cette façon, vous pouvez donner à des sessions TCP interactives (comme telnet, SSH, POP3) une très grande vitesse de connexion, ceux–ci pouvant ne pas être très appropriés

pour des tranferts importants. Si la connexion est de plus faible importance (SMTP, ou ce que vous voulez classer en basse priorité), vous pouvez employer un temps de latence plus important, ce qui est meilleur marché que d'utiliser des connexions en haute ou basse latence.

Tableau 11.3. Options de la cible DSCP

Option	***--set-dscp***
Exemple	***iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp 1***
Explication	Ceci place la valeur DSCP à la valeur spécifiée. Les valeurs peuvent être placées soit par class, voir ci-dessous, soit avec le ***--set-dscp***, qui prend une valeur entière ou une valeur hexadécimale.
Option	***--set-dscp-class***
Exemple	***iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF***
Explication	Place le champ DSCP selon une classe Diffserv prédéfinie. Certaines des valeurs possibles sont EF, BE et les valeurs CSxx et AFxx disponibles. Vous pouvez trouver plus d'information sur le site Implementing Quality of Service Policies with DSCP. Notez que les commandes ***--set-dscp-class*** et ***--set-dscp*** sont mutuellement exclusives, ce qui veut dire que vous ne pouvez pas les utiliser ensemble dans la même commande !

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.6. Cible ECN

Cette cible peut être extraordinaire, utilisée correctement. Simplement placée, la cible ***ECN*** peut être utilisée pour réinitialiser les bits ECN depuis l'en-tête IPv4, ou les réinitialiser à 0 au moins. ECN est une chose relativement nouvelle sur le net, et il y a quelques problèmes avec elle. Par exemple, elle utilise 2 bits définis dans la RFC originale du protocole TCP comme devant être à 0. Certains routeurs et autres serveurs Internet ne transfèrent pas les paquets dont les bits sont placés à 1. Si vous voulez faire usage d'une partie au moins des fonctionnalités de ECN depuis vos hôtes, vous pourrez par exemple réinitialiser les bits ECN à 0 pour les réseaux spécifiques dont nous savons qu'il y a des problèmes de connexion à cause de ECN.

Note

Notez qu'il n'est pas possible d'activer ECN au milieu d'un flux. Ce n'est pas autorisé selon la RFC, et ne sera possible en aucune façon. Les deux points limite d'un flux doivent négocier l'ECN. Si nous l'activons, un des hôtes n'est pas informé de cela, et ne peux répondre proprement aux notifications ECN.

Tableau 11.4. Options de la cible ECN

Option	***--ecn-tcp-remove***
Exemple	***iptables -t mangle -A FORWARD -p tcp --dport 80 -j ECN --ecn-tcp-remove***
Explication	La cible ECN prend un seul argument, ***--ecn-tcp-remove***. Ceci indique à la cible de supprimer les bits ECN des en-têtes TCP. Voir au-dessus pour plus d'information.

Note

Fonctionne avec les noyaux Linux 2.5 et 2.6.

11.7. Options de la cible LOG

La cible ***LOG*** est spécialement destinée à journaliser des informations détaillées sur les paquets. Ceci peut, par exemple, être considéré comme illégal. Ou la journalisation peut servir à la recherche de bogues et

d'erreurs. La cible ***LOG*** renverra une information spécifique sur les paquets, comme les en−têtes IP et autre détails considérés comme intéressants. Ceci se réalise par les fonctionnalités de journalisation du noyau, normalement ***syslogd***. Cette information peut alors être lue directement avec la commande ***dmesg***, ou depuis les journaux ***syslogd***, ou avec d'autres programmes ou applications. C'est une excellente cible utilisée comme débogage des tables de règles, ainsi vous pouvez voir où vont les paquets et comment les règles sont appliquées et sur quels paquets. Notez que ce peut être une très bonne idée d'utiliser la cible ***LOG*** au lieu de la cible ***DROP*** lorsque vous testez une règle dont vous n'êtes pas sûrs à 100% de son efficacité dans un pare−feu en production, car une erreur de syntaxe dans la table de règles pourrait causer de sévères problèmes de connectivité entre vos utilisateurs. Notez aussi que la cible ***ULOG*** peut être intéressante si vous utilisez réellement une journalisation extensive, car ***ULOG*** supporte directement la journalisation dans les bases de données `MySQL` et d'autres.

Note

Notez que si vous obtenez une sortie de journalisation directement vers les consoles, ce n'est pas un problème de ***iptables*** ou Netfilter, mais plutôt un problème causé par votre configuration de syslogd − probablement `/etc/syslog.conf`. Pour en savoir plus ***man syslog.conf***.

Vous pourriez aussi désirer revoir les paramétrages de ***dmesg***. ***dmesg*** est la commande qui permet de voir sur une console les erreurs envoyées par le noyau. ***dmesg −n 1*** enverra tous les messages sur la console, sauf les messages de panique. Les niveaux de message de dmesg apparient exactement les niveaux de syslogd, et fonctionnent seulement sur les messages de journalisation depuis les fonctionnalités du noyau. Pour plus d'information voir ***man dmesg***.

La cible ***LOG*** prend actuellement cinq options qui peuvent être intéressantes si vous recherchez une information précise, ou désirez placer différentes options pour certaines valeurs. Elles sont présentées ci−dessous.

Tableau 11.5. Options de la cible LOG

Option	***−−log−level***
Exemple	***iptables −A FORWARD −p tcp −j LOG −−log−level debug***
Explication	C'est l'option qui indique à ***iptables*** et ***syslog*** quel niveau de journalisation utiliser. Pour une liste complète des niveaux de journalisation lisez le manuel `syslog.conf`. Normalement il y a les niveaux de journalisation suivants, ou les priorités qui s'y réfèrent : `debug`, `info`, `notice`, `warning`, `warn`, `err`, `error`, `crit`, `alert`, `emerg` et `panic`. le mot−clé `error` est le même que `err`, `warn` est le même que `warning` et `panic` le même que `emerg`. Notez que tous les trois sont obsolètes, en d'autres termes n'utilisez pas `error`, `warn` et `panic`. La priorité définit le niveau de rigueur des messages journalisés. Tous les messages sont journalisés par les fonctionnalités du noyau. En d'autres termes, placer ***kern.=info /var/log/iptables*** dans votre fichier `syslog.conf` et ensuite laisser tous vos messages de ***LOG*** dans ***iptables*** utilise le niveau info, ce qui fera que tous vos messages apparaîtront dans le fichier `/var/log/iptables`. Notez qu'il peut y avoir d'autres messages provenant d'autres parties du noyau qui utilisent la priorité info. Pour plus d'information sur la journalisation, je vous recommande de lire les pages de manuel de ***syslog*** et `syslog.conf` comme les autres HOWTO, etc.
Option	***−−log−prefix***
Exemple	***iptables −A INPUT −p tcp −j LOG −−log−prefix "INPUT packets"***
Explication	Cette option indique à ***iptables*** de préfixer tous les messages de journalisation avec un préfixe spécifique, qui peut être facilement combiné avec ***grep*** ou d'autres outils qui permettent de tracer ces problèmes et les sorties des différentes règles.

	Le préfixe peut avoir jusqu'à 29 lettres de long, incluant les espaces et autres symboles spéciaux.
Option	***--log-tcp-sequence***
Exemple	***iptables -A INPUT -p tcp -j LOG --log-tcp-sequence***
Explication	Cette option journalisera les numéros des `Séquences TCP`, avec le message de journalisation. Les numéros de Séquences TCP sont des nombres spéciaux qui identifient chaque paquet et qu'ils ajustent dans une séquence TCP, et permettent de savoir comment le flux sera réassemblé. Notez que cette option constitue un risque de sécurité si les journaux sont lisibles par des utilisateurs non autorisés, ou par tout le monde.
Option	***--log-tcp-options***
Exemple	***iptables -A FORWARD -p tcp -j LOG --log-tcp-options***
Explication	L'option ***--log-tcp-options*** journalise les différentes options des en-têtes des paquets TCP et peuvent être utiles lors du débogage. Cette option ne prend aucun champ de variable, comme beaucoup d'options ***LOG***.
Option	***--log-ip-options***
Exemple	***iptables -A FORWARD -p tcp -j LOG --log-ip-options***
Explication	L'option ***--log-ip-options*** journalisera la plupart des options des en-têtes de paquets IP. Elle fonctionne exactement comme l'option ***--log-tcp-options***, mais sur les options IP. Ces messages de journalisation peuvent être utiles pour le débogage ou le traçage, comme dans l'option précédente.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.8. Cible MARK

La cible ***MARK*** sert à placer les valeurs de marquage ***Netfilter*** qui sont associées à des paquets spécifiques. Cette cible n'est valide que dans la table mangle, et ne fonctionne pas en dehors de celle-ci. Les valeurs ***MARK*** peuvent être utilisées conjointement avec les possibilités de routage avancé de Linux pour envoyer différents paquets à travers différentes routes et indiquer d'utiliser différentes disciplines de files d'attente (qdisc), etc. Pour plus d'information sur le routage avancé, voyez le Linux Advanced Routing and Traffic Control HOW-TO. Notez que la valeur de marquage n'est pas incluse dans le paquetage actuel, mais est associée au paquet dans le noyau. En d'autres termes, vous ne pouvez pas placer une ***MARK*** pour un paquet et ensuite espérer que la ***MARK*** sera toujours présente sur un autre hôte. Si c'est ce que vous voulez, vous feriez mieux d'utiliser la cible ***TOS*** qui analysera la valeur TOS dans l'en-tête IP.

Tableau 11.6. Options de la cible MARK

Option	***--set-mark***
Exemple	***iptables -t mangle -A PREROUTING -p tcp --dport 22 -j MARK --set-mark 2***
Explication	L'option ***--set-mark*** est nécessaire pour placer une marque. ***--set-mark*** prend une valeur entière. Par exemple, nous pouvons placer la marque à 2 sur un flux spécifique de paquets, ou sur tous les paquets provenant d'un hôte précis et ensuite faire du routage avancé sur cet hôte, pour augmenter ou diminuer la bande passante du réseau, etc.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.9. Cible MASQUERADE

La cible ***MASQUERADE*** est utilisée (de façon basique) comme la cible ***SNAT***, mais ne nécessite aucune option ***--to-source***. La raison de ceci est que la cible ***MASQUERADE*** a été créée pour fonctionner avec, par exemple, des connexions en dial-up (accès par ligne commutée), ou en DHCP, qui récupèrent des adresses IP dynamiques lors de la connexion au réseau. Ceci veut dire que vous n'utiliserez la cible ***MASQUERADE*** qu'avec des connexions fournissant des adresses IP dynamiques. Si vous avez une adresse IP statique, vous utiliserez dans ce cas la cible ***SNAT***.

Quand vous masquez une connexion, ça indique que vous placez l'adresse IP utilisée sur une interface réseau spécifique au lieu de l'option ***--to-source***, et l'adresse IP est automatiquement récupérée depuis cette interface spécifique. La cible ***MASQUERADE*** a également pour effet que les connexions sont abandonnées quand une interface est coupée, ce qui est extrêmement intéressant si nous coupons une interface spécifique. Ceci est, en général, le comportement correct avec les lignes en dial-up qui ont sans doute des IP assignées à chaque connexion. Lorsque une IP différente est attribuée, la connexion est perdue, et il est idiot d'en conserver les entrées.

Il est toujours possible d'utiliser la cible ***MASQUERADE*** au lieu de ***SNAT*** même si vous avez une IP statique, cependant, ce n'est pas très intéressant car ça ajoute un surdébit, et peut aller à l'encontre de vos scripts et les rendre "inutilisables".

Notez que la cible ***MASQUERADE*** n'est valide que dans la chaîne `POSTROUTING` de la table nat, comme la cible ***SNAT***. ***MASQUERADE*** ne prend qu'une option spécifiée ci-dessous, et qui est optionnelle.

Tableau 11.7. Cible MASQUERADE

Option	***--to-ports***
Exemple	***iptables -t nat -A POSTROUTING -p TCP -j MASQUERADE --to-ports 1024-31000***
Explication	L'option ***--to-ports*** est utilisée pour placer le port source ou des ports sur des paquets sortants. Soit vous pouvez spécifier un seul port comme ***--to-ports 1025*** soit une plage de ports comme ***--to-ports 1024-3000***. En d'autres termes, les délimitations des plages de ports la plus basse et la plus haute séparées par un tiret. Ceci modifie la sélection de port par défaut de SNAT comme décrit dans la section Cible SNAT. L'option ***--to-ports*** n'est valide que si la section de correspondance de la règle spécifie les protocoles TCP ou UDP avec la correspondance ***--protocol***.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.10. Cible MIRROR

Avertissement

Attention, ***MIRROR*** est dangereux et n'a été développé que comme exemple de code pour le nouveau conntrack et NAT. Elle peut provoquer des failles dangereuses, et de très sérieux DDoS/DoS sont possibles si elle est utilisée improprement. Évitez de l'utiliser dans tous les cas ! Elle a été supprimée dans les noyaux 2.5 et 2.6 à cause de ses implications dans la sécurité.

La cible ***MIRROR*** est expérimentale, et vous êtes prévenus qu'il peut en résulter de sérieux problèmes de Denial of Service. ***MIRROR*** est utilisée pour inverser les champs source et destination dans l'en-tête IP, avant de retransmettre le paquet. Ceci peut causer quelques effets comiques, un cracker a cracké sa propre machine en l'utilisant. Plaçons une cible ***MIRROR*** sur le port 80 d'un ordinateur A. Si l'hôte B vient de yahoo.com, et essaie d'accéder au serveur HTTP de A, le cible ***MIRROR*** renverra l'hôte yahoo à sa propre page web (car c'est de là que vient la requête).

Notez que la cible ***MIRROR*** n'est valide que dans les chaînes `INPUT`, `FORWARD` et `PREROUTING`, et les chaînes définies par l'utilisateur. Notez aussi que les paquets sortants sont le résultat de la cible ***MIRROR*** et ne sont vus par aucune des chaînes normales du filtre, les tables nat ou mangle, qui peuvent provoquer des boucles et autres problèmes. Ceci peut faire que la cible soit la cause de maux de têtes inattendus. Par exemple, un hôte peut envoyer un paquet de mystification vers un autre hôte qui utilise la commande ***MIRROR*** avec un TTL de 255, en même temps il mystifie son propre paquet, comme s'il semblait qu'il venait d'un troisième hôte utilisant cette commande ***MIRROR***. Le paquet sera alors renvoyé sans arrêt, pour le nombre de sauts nécessaires pour qu'il soit complété. S'il n'y a qu'un seul saut, le paquet reviendra 240–255 fois. C'est intéressant pour un cracker, en d'autres termes, envoyer 1500 octets de données consomme 380 ko de votre connexion. Notez que ceci est le meilleur scenario pour un cracker.

Note

Fonctionne avec les noyaux Linux 2.3 et 2.4. A été supprimé dans les noyaux 2.5 et 2.6 à cause de problèmes de sécurité. N'utilisez pas cette cible !

11.11. Cible NETMAP

NETMAP est une implémentation nouvelle des cibles ***SNAT*** et ***DNAT*** où la partie hôte de l'adresse IP n'est pas changée. Elle procure une fonction `NAT` 1:1 pour l'ensemble des réseaux qui n'ont pas de fonctions ***SNAT*** et ***DNAT*** standards. Par exemple, nous avons un réseau de 254 hôtes utilisant des adresses IP privées (un réseau /24), et nous avons un nouveau réseau /24 d'adresses IP publiques. Au lieu de changer les IP de chacun des hôtes, il sera plus simple d'utiliser la cible ***NETMAP*** comme –j NETMAP –to 10.5.6.0/24, tous les hôtes seront vus comme 10.5.6.x quand ils quitteront le pare–feu. Exemple, 192.168.0.26 deviendra 10.5.6.26.

Tableau 11.8. Options de la cible NETMAP

Option	***––to***
Exemple	***iptables –t mangle –A PREROUTING –s 192.168.1.0/24 –j NETMAP ––to 10.5.6.0/24***
Explication	C'est la seule option de la cible ***NETMAP***. Dans l'exemple précédent, les hôtes 192.168.1.x seront directement traduits en 10.5.6.x

Note

Fonctionne avec les noyaux Linux 2.5 et 2.6.

11.12. Cible QUEUE

La cible ***QUEUE*** sert à mettre les paquets en attente pour les programmes et applications du domaine utilisateur. Elle est utilisée conjointement avec des programmes ou des utilitaires étrangers à Iptables et qui peuvent être utilisés, par exemple, pour des réseaux comptables, ou pour des applications avancées et spécifiques qui filtrent ou mettent en cache les paquets. Nous ne parlerons pas de cette cible en détail, car le codage de certaines applications est hors de sujet de ce didacticiel. En premier, ça nous prendrait trop de temps, ensuite cette documentation n'a pas grand chose à faire avec l'aspect programmation de Netfilter et Iptables. Voir le Netfilter Hacking HOW–TO.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.13. Cible REDIRECT

La cible ***REDIRECT*** est utilisée pour rediriger les paquets et les flux vers la machine elle–même. Ceci veut dire que nous pouvons, par exemple ***REDIRECT*** tous les paquets destinés aux ports `HTTP` vers un proxy HTTP comme Squid, sur notre propre machine. Les paquets générés localement sont mappés vers les adresses 127.0.0.1. En d'autres termes, elle réécrit les adresses de destination vers votre propre machine pour les paquets qui sont transmis, ou quelque chose comme ça. La cible ***REDIRECT*** est très utile quand vous voulez,

par exemple, faire du proxy transparent, où l'hôte du *LAN* n'a pas connaissance du proxy.

Notez que la cible ***REDIRECT*** est uniquement valide dans les chaînes `PREROUTING` et `OUTPUT` de la table nat. Elle est aussi valide dans les chaînes définies par l'utilisateur. ***REDIRECT*** ne prend qu'une option, comme décrit ci–dessous.

Tableau 11.9. Cible REDIRECT

Option	***––to–ports***
Exemple	***iptables –t nat –A PREROUTING –p tcp ––dport 80 –j REDIRECT ––to–ports 8080***
Explication	L'option ***––to–ports*** spécifie le port de destination, ou la plage de ports, à utiliser. Sans ***––to–ports***, le port de destination n'est jamais modifié. Ceci est spécifié, comme au–dessus ***––to–ports 8080*** dans les cas où nous voulons seulement préciser un seul port. Si nous voulons spécifier une plage de ports, nous écrirons ***––to–ports 8080–8090***, qui indique à la cible ***REDIRECT*** de rediriger les paquets vers les ports 8080 jusqu'à 8090. Cette option n'est disponible que dans les règles spécifiant le protocole TCP ou UDP avec le module ***––protocol***.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.14. Cible REJECT

La cible ***REJECT*** fonctionne à la base comme la cible ***DROP***, mais elle renvoit un message d'erreur à l'hôte qui a envoyé le paquet. ***REJECT*** n'est valide que dans les chaînes `INPUT`, `FORWARD` et `OUTPUT` ou leurs sous–chaînes. Après tout, ce sont les seules chaînes dans lesquelles il soit sensé de placer cette cible. Notez que toutes les chaînes qui utilisent ***REJECT*** ne peuvent être invoquées que par `INPUT`, `FORWARD`, et `OUTPUT`, sinon elles ne fonctionnent pas. Il n'y a qu'une option qui contrôle le fonctionnement de cette cible.

Tableau 11.10. Cible REJECT

Option	***––reject–with***
Exemple	***iptables –A FORWARD –p TCP ––dport 22 –j REJECT ––reject–with tcp–reset***
Explication	Cette option indique à la cible ***REJECT*** quelle réponse envoyer à l'hôte qui a expédié le paquet qui a été rejeté. Quand nous sommes en présence d'un paquet qui apparie une règle dans laquelle nous avons spécifié cette cible, notre hôte envoie la réponse associée, et le paquet est ensuite supprimé, comme pour la cible ***DROP***. Les types suivants de rejet sont valides : `icmp-net-unreachable`, `icmp-host-unreachable`, `icmp-port-unreachable`, `icmp-proto-unreachable`, `icmp-net-prohibited` et `icmp-host-prohibited`. Le message d'erreur par défaut expédie un ***port–unreachable*** à l'hôte. Tous sont des messages d'erreur `ICMP` et peuvent être paramétrés comme vous le voulez. Vous trouverez plus d'information dans l'annexe Types ICMP. Enfin, il existe une option supplémentaire appelée ***tcp–reset***, qui peut être utilisée seulement avec le protocole TCP. L'option ***tcp–reset*** qui indique le ***REJECT*** envoie un paquet TCP RST en réponse à l'hôte expéditeur. Les paquets TCP RST sont utilisés pour clore les connexions TCP. Pour plus d'information sur TCP RST voir la RFC 793 – Transmission Control Protocol. Comme indiqué dans le manuel d'***iptables***, elle est principalement utilisée pour bloquer les sondeurs d'identité.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.15. Cible RETURN

La cible ***RETURN*** stoppe un paquet traversant la chaîne dans laquelle la règle est placée. Si c'est une sous−chaîne d'une autre chaîne, le paquet continuera sa route vers les chaînes supérieures comme si rien ne s'était passé. Si cette chaîne est la chaîne principale, par exemple la chaîne INPUT, le paquet aura le comportement par défaut. Ce comportement par défaut est normalement ***ACCEPT***, ***DROP*** ou similaire.

Exemple, un paquet entre dans la chaîne INPUT et rencontre une règle qui l'apparie et indique ***−−jump EXAMPLE_CHAIN***. Ce paquet traversera alors ***EXAMPLE_CHAIN***, et soudain il rencontre une règle qui a la cible ***−−jump RETURN***. Il retournera alors vers la chaîne `INPUT`. Un autre exemple, si le paquet rencontrera une règle ***−−jump RETURN*** dans la chaîne INPUT. Il sera alors droppé selon le comportement par défaut décrit plus haut, et plus aucune action sera faite dans cette chaîne.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.16. Cible SAME

La cible ***SAME*** fonctionne à peu près comme ***SNAT***, mais diffère légèrement. À la base, ***SAME*** tentera d'utiliser la même adresse IP en sortie pour toutes les connexions initiées par un hôte sur le réseau. Par exemple, vous avez un réseau en /24 (192.168.1.0) et 3 adresses IP (10.5.6.7−9). Maintenant, si 192.168.1.20 sort de l'adresse .7 une première fois, le pare−feu tentera de conserver à cette machine toujours la même adresse IP.

Tableau 11.11. Optiond de la cible SAME

Option	***−−to***
Exemple	***iptables −t mangle −A PREROUTING −s 192.168.1.0/24 −j SAME −−to 10.5.6.7−10.5.6.9***
Explication	Comme vous pouvez le voir, l'argument −−to prend deux adresses IP liées ensemble par un −. Ces adresses IP, et toutes les autres entre, sont des adresses que nous NATons pour utiliser l'algorithme ***SAME***.
Option	***−−nodst***
Exemple	***iptables −t mangle −A PREROUTING −s 192.168.1.0/24 −j SAME −−to 10.5.6.7−10.5.6.9 −−nodst***
Explication	Au cours d'une action normale, la cible ***SAME*** calcule le suivi de connexions basé sur les adresses IP source et destination. L'option −−nodst, permet de n'utiliser que l'adresse IP source pour savoir de quelle IP la fonction NAT se sert pour la connexion spécifique. Sans cet argument, elle utilise une combinaison de l'adresse IP source et destination.

Note

Fonctionne avec les noyaux Linux 2.5 et 2.6.

11.17. Cible SNAT

La cible ***SNAT*** est utilisée pour la Traduction d'Adresse Réseau Source, ce qui veut dire que cette cible réécrira l'adresse IP source dans l'en−tête IP du paquet. Exemple, quand plusieurs hôtes doivent partager une connexion Internet. Nous pouvons alors activer le transfert d'IP (IP Forwarding) dans le noyau, et écrire une règle ***SNAT*** qui traduira tous les paquets sortants du réseau local vers l'***IP source*** de notre connexion Internet. Sans cela, le monde extérieur ne saurait pas où envoyer les paquets en réponse, car les réseaux locaux utilisent la plupart du temps des adresses IP spécifiées par le IANA et qui sont allouées aux ***LAN***. Si nous transférons

les paquets tels quels, personne sur l'Internet ne saura qu'ils proviennent de nous. La cible ***SNAT*** fait toutes les traductions nécessaires pour réaliser ce genre de chose, permettant à tous les paquets quittant notre ***LAN*** d'être vus comme provenant d'un hôte unique, qui pourrait être notre pare−feu.

SNAT n'est valide que dans la table nat, à l'intérieur de la chaîne POSTROUTING. C'est, en d'autres termes, la seule chaîne dans laquelle vous pouvez utiliser ***SNAT***. Seul le premier paquet d'une connexion est analysé par ***SNAT***, et ensuite tous les paquets utilisant la même connexion seront également ***SNAT***és. De plus, les règles initiales de la chaîne POSTROUTING seront appliquées à tous les paquets du même flux.

Tableau 11.12. Options de la cible SNAT

Option	***−−to−source***
Exemple	***iptables −t nat −A POSTROUTING −p tcp −o eth0 −j SNAT −−to−source 194.236.50.155−194.236.50.160:1024−32000***
Explication	L'option ***−−to−source*** est utilisée pour spécifier quelle source le paquet doit utiliser. Cette option, la plus simple, prend l'adresse IP que nous voulons utiliser pour adresse ***IP source*** dans l'***en−tête IP***. Si nous voulons faire ceci entre plusieurs adresses IP, nous pouvons utiliser une plage d'adresses, séparées par un tiret. Les numéros IP −−to−−source peuvent alors ressembler à notre exemple ci−dessus : 194.236.50.155−194.236.50.160. L'IP source pour chaque flux que nous ouvrons sera allouée aléatoirement, et un flux utilisera toujours la même adresse IP pour tous les paquets transitants dans ce flux. Nous pouvons aussi spécifier une plage de ports à utiliser par ***SNAT***. Tous les ports source seront alors confinés aux ports spécifiés. Le bit de port de la règle ressemblera alors à notre exemple, :1024−32000. Ce n'est valide que si −p tcp ou −p udp sont spécifiés quelque part dans la correspondance de la règle en question. Iptables essaiera toujours d'éviter de modifier les ports si possible, mais si deux hôtes tentent d'utiliser les mêmes ports, Iptables redirigera un de ceux−là vers un autre port. Si aucune plage de ports n'est précisée, et si elles sont requises, tous les ports source au dessous de 512 seront redirigés vers d'autres ports en dessous de 512. Ceux entre les ports source 512 et 1023 seront redirigés en dessous de 1023. Tous les autres ports seront redirigés vers 1024 et au dessus. Comme établi précédemment, Iptables tentera toujours de conserver les ports source utilisés par la machine établissant la connexion. Notez que ceci n'a rien à voir avec les ports destination, si un client essaie de prendre contact avec un serveur ***HTTP*** en dehors du pare−feu, il ne sera pas redirigé vers le port ***FTP control***.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.18. Cible TCPMSS

La cible ***TCPMSS*** peut être utilisée pour modifier la valeur MSS (Maximum Segment Size) des paquets TCP SYN que le pare−feu examine. La valeur MSS sert à contrôler la taille maximum des paquets d'une connexion spécifique. Dans des circonstances normales, ceci indique la taille de la valeur MTU (Maximum Transfert Unit), moins 40 octets. Elle est utilisée pour éviter que certains fournisseurs d'accès ou serveurs bloquent la fragmentation ICMP des paquets, ce qui peut provoquer des problèmes mystérieux, qui peuvent être décrits principalement par le fait que tout fonctionne parfaitement au niveau de notre routeur/pare−feu, mais que nos hôtes locaux derrière le pare−feu ne peuvent échanger des paquets importants. Ceci peut se traduire par certaines choses comme des serveurs de courrier capables d'envoyer des petits mails, mais pas des gros, des navigateurs web qui se connectent mais ensuite se figent en ne recevant aucune donnée, une connexion ssh propre, mais dont le scp est suspendu après l'établissement de la liaison. Autrement dit, tout ce qui utilise des paquets importants sera incapable de fonctionner.

La cible TCPMSS est capable de résoudre ces problèmes, en changeant la taille des paquets sortants d'une connexion. Notez que nous avons uniquement besoin de placer le MSS sur le paquet SYN, les hôtes s'occupant du MSS après ça. La cible prend deux arguments.

Tableau 11.13. Options de la cible TCPMSS

Option	***−−set−mss***
Exemple	***iptables −t mangle −A POSTROUTING −p tcp −−tcp−flags SYN,RST SYN −o eth0 −j TCPMSS −−set−mss 1460***
Explication	L'argument ***−−set−mss*** place une valeur MSS spécifique pour tous les paquets sortants. Dans l'exemple ci−dessus, nous plaçons le MSS de tous les paquets SYN sortants sur l'interface eth0 à 1460 octets −− le MTU normal pour l'ethernet est de 1500 octets, moins 40 octets soit 1460 octets. MSS doit seulement être placé correctement dans le paquet SYN, ensuite les hôtes pairs s'occupent du MSS automatiquement.
Option	***−−clamp−mss−to−pmtu***
Exemple	***iptables −t mangle −A POSTROUTING −p tcp −−tcp−flags SYN,RST SYN −o eth0 −j TCPMSS −−clamp−mss−to−pmtu***
Explication	***−−clamp−mss−to−pmtu*** place automatiquement le MSS à la bonne valeur, et désormais vous n'aurez plus besoin de le disposer explicitement. Il est mis de façon automatique en PMTU (Path Maximum Transfer Unit) moins 40 octets, ce qui est une valeur raisonnable pour la plupart des applications.

Note

Fonctionne avec les noyaux Linux 2.5 et 2.6.

11.19. Cible TOS

La cible ***TOS*** sert à disposer le champ Type de Service dans une en−tête IP. Le champ TOS consiste en 8 bits utilisés pour aider au routage de paquets. C'est un des champs qui peut être utilisé directement dans ***iproute2*** et son sous−système pour les stratégies de routage. Notez de plus, que si vous maintenez plusieurs pare−feux et routeurs séparés, c'est le seul moyen pour propager les informations de routage dans les paquets entre ces routeurs et pare−feux. Comme noté précédemment, la cible ***MARK*** – laquelle dispose un ***MARK*** associé à un paquet spécifique – est disponible seulement dans le noyau, et ne peut pas être propagée avec le paquet. Si vous avez besoin de propager des informations de routage pour un paquet spécifique ou un flux, vous devrez donc placer le champ TOS, qui a été créé pour ça.

Il existe un grand nombre de routeurs sur Internet qui font du mauvais travail à ce sujet, ce qui fait qu'il peut être moins utile maintenant d'essayer de faire de l'analyse TOS avant d'envoyer les paquets sur Internet. Dans le meilleur des cas les routeurs ne font pas attention au champ TOS. Dans le pire, ils examineront le champ TOS et feront de mauvaises choses. Cependant, le champ TOS peut être placé avec une grande utilité si vous avez un grand WAN ou LAN avec de multiples routeurs. Vous avez la possibilité de donner aux paquets différentes routes et préférences, basées sur leur valeur TOS.

Attention

La cible ***TOS*** ne place que des valeurs spécifiques, ou valeurs nommées sur des paquets. Ces valeurs prédéfinies peuvent être trouvées dans les fichiers include du noyau, ou plus précisément le fichier `Linux/ip.h`. Les raisons sont multiples, et vous n'aurez actuellement besoin de placer aucune autre valeur; cependant, il existe d'autres moyens par rapport à cette limitation. Pour contourner cette limitation de ne pouvoir placer que des valeurs nommées sur les paquets, vous pouvez utiliser le patch FTOS disponible sur le site Paksecured Linux Kernel patches maintenu par Matthew G. Marsh. Attention, soyez prudents avec ce patch ! Vous ne devriez pas avoir besoin d'autre chose que les valeurs par défaut, sauf dans des cas extrêmes.

Note

Cette cible n'est valide que dans la table `mangle` et ne peut être utilisée hors de celle−ci.

Note

Notez aussi que certaines anciennes versions (1.2.2 ou avant) d'Iptables fournissaient une implémentation de cette cible qui ne fixait pas la somme de contrôle dans l'analyse, ce qui corrompait les paquets et provoquait des connexions qui n'aboutissaient pas.

TOS ne prend qu'une option décrite ci–dessous.

Tableau 11.14. Cible TOS

Option	***––set–tos***
Exemple	***iptables –t mangle –A PREROUTING –p TCP ––dport 22 –j TOS ––set–tos 0x10***
Explication	L'option ***––set–tos*** indique à l'analyseur ***TOS*** quelle valeur placer sur les paquets qui sont appariés. L'option prend une valeur numérique, soit en hexadécimal soit en décimal. Comme la valeur TOS consiste en 8 bits, la valeur peut être 0–255, ou en hexadécimal 0x00–0xFF. Notez que dans la cible TOS standard vous êtes limités à l'utilisation des valeurs nommées disponibles (qui sont plus ou moins standards), comme mentionné précédemment. Ces valeurs sont `Minimize-Delay` (valeur décimale 16, valeur hexadécimale 0x10), `Maximize-Throughput` (valeur décimale 8, valeur hexadécimale 0x08), `Maximize-Reliability` (valeur décimale 4, valeur hexadécimale 0x04), `Minimize-Cost` (valeur décimale 2, valeur hexadécimale 0x02), ou `Normal-Service` (valeur décimale 0, valeur hexadécimale 0x00). La valeur par défaut pour la plupart des paquets est `Normal-Service`, ou 0. Notez que vous pouvez, bien sûr, utiliser les noms au lieu des valeurs hexadécimales pour placer la valeur TOS; en fait, c'est généralement recommandé, car les valeurs associées avec les noms peuvent être changées par la suite. Pour une liste complète des "valeurs descriptives", tapez la commande : ***iptables –j TOS –h***.

Note

Fonctionne avec les noyaux 2.3, 2.4, 2.5 et 2.6.

11.20. Cible TTL

Attention

Ce patch nécessite le ***TTL*** du patch–o–matic disponible dans le répertoire de http://www.netfilter.org/.

La cible ***TTL*** modifie le champ Durée de Vie (Time To Live) dans l'en–tête IP. Une application très utile de ceci est de pouvoir changer toutes les valeurs de durée de vie en une valeur identique pour tous les paquets sortants. Une raison de faire ça peut être que, vous avez un fournisseur d'accès un peu rigide qui ne vous permet pas d'avoir plus d'une machine connectée à la même connexion Internet. En mettant toutes les valeurs ***TTL*** à la même valeur, il sera plus difficile pour lui de voir ce que vous faites. Nous pouvons alors réinitialiser la valeur ***TTL*** de tous les paquets sortants à une valeur standard, comme 64 ainsi que spécifié dans le noyau Linux.

Pour plus d'information pour savoir comment placer la valeur par défaut utilisée dans Linux, lisez le ip–sysctl.txt, que vous pouvez trouver dans l'annexe Autres ressources et liens.

La cible ***TTL*** n'est valide que dans la table mangle, et nulle part ailleurs. Elle prend trois options, décrites ci–dessous.

Tableau 11.15. Cible TTL

Option	***––ttl–set***
Exemple	***iptables –t mangle –A PREROUTING –i eth0 –j TTL ––ttl–set 64***

Explication	L'option ***--ttl-set*** indique à la cible ***TTL*** quelle valeur placer sur le paquet en question. Une bonne valeur serait aux alentours de 64. Ce n'est ni trop long ni trop court. Ne placez pas cette valeur trop haut, car elle peut affecter votre réseau. Cette cible peut être utilisée pour limiter la distance de vos clients. Un bon exemple de ceci peuvent être les serveurs DNS, où nous ne voulons pas que les clients soient trop éloignés.
Option	***--ttl-dec***
Exemple	***iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-dec 1***
Explication	L'option ***--ttl-dec*** indique à la cible ***TTL*** de décrémenter la valeur TTL d'un montant précis après le ***--ttl-dec***. En d'autres termes, si le TTL d'un paquet entrant était de 53 et que nous avons ajouté ***--ttl-dec 4***, le paquet quittera l'hôte avec une valeur de 49. La raison en est que le code réseau décrémentera automatiquement la valeur TTL par 1, donc le paquet sera décrémenté en 4 étapes, de 53 à 49. Ceci peut être utilisé quand nous voulons limiter l'éloignement de clients utilisant nos services. Exemple, les hôtes utilisent toujours un DNS proche, et donc nous pouvons apparier tous les paquets quittant notre serveur DNS et réduire la distance en plusieurs étapes. Bien sûr, le ***--set-ttl*** peut être une meilleure idée pour cet usage.
Option	***--ttl-inc***
Exemple	***iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1***
Explication	L'option ***--ttl-inc*** indique à la cible ***TTL*** d'incrémenter la valeur `Time To Live` d'une valeur spécifiée avec ***--ttl-inc***. Ceci indique que nous voulons augmenter le TTL avec une valeur spécifiée dans l'option ***--ttl-inc***, et que si nous spécifions ***--ttl-inc 4***, un paquet entrant avec un TTL de 52 quittera l'hôte avec un TTL de 56. Notez que la même chose dans l'exemple ***--ttl-dec*** s'applique ici, dans lequel le code réseau décrémentait automatiquement la valeur TTL par 1, ce qui est toujours le cas. Ceci peut être utilisé pour rendre notre pare-feu un peu plus furtif pour les traceroutes parmi d'autres choses. En mettant le TTL à une valeur plus haute pour tous les paquets entrants, nous rendons effectivement le pare-feu plus dissimulé pour les traceroutes. Les traceroutes sont des choses adorables et détestables, car elles fournissent d'excellentes informations sur les problèmes de connexion et indiquent à quel endroit ils se produisent, mais en même temps ils fournissent au hacker/cracker des informations dans le sens montant s'il nous a pris pour cible. Pour un bon exemple de son utilisation, voir le script Ttl-inc.txt.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

11.21. Cible ULOG

La cible ***ULOG*** sert à la journalisation des paquets appariés de l'espace utilisateur. Si un paquet est apparié et la cible ***ULOG*** placée, l'information du paquet est multidiffusée avec le paquet complet à travers une interface de connexion réseau. Un ou plusieurs processus espace utilisateur peuvent souscrire aux divers groupes de multidiffusion et recevoir le paquet. C'est une possibilité de journalisation plus complète et plus sophistiquée qui est utilisée par Iptables et Netfilter. Cette cible nous permet de journaliser l'information dans des bases de données MySQL, ou autres bases, rendant plus simple la recherche de paquets spécifiques, et groupant les entrées de journal. Vous pouvez trouver les applications domaine utilisateur ULOGD à ULOGD project page.

Tableau 11.16. Cible ULOG

Option	***--ulog-nlgroup***
Exemple	***iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-nlgroup 2***
Explication	L'option ***--ulog-nlgroup*** indique à la cible ***ULOG*** à quel groupe netlink envoyer les paquets. Il existe 32 groupes netlink, qui sont simplement spécifiés 1-32. Si nous voulons joindre le

	groupe netlink 5, nous écrirons simplement ***−−ulog−nlgroup 5***. Le groupe netlink par défaut est 1.
Option	***−−ulog−prefix***
Exemple	***iptables −A INPUT −p TCP −−dport 22 −j ULOG −−ulog−prefix "SSH connection attempt: "***
Explication	L'option ***−−ulog−prefix*** fonctionne comme la valeur de préfixe de la cible ***LOG***. Cette option préfixe toutes les entrées du journal avec un préfixe utilisateur. Il peut être de 32 caractères de longueur, et est le plus utilisé pour distinguer différents messages de journal et d'où ils proviennent.
Option	***−−ulog−cprange***
Exemple	***iptables −A INPUT −p TCP −−dport 22 −j ULOG −−ulog−cprange 100***
Explication	L'option ***−−ulog−cprange*** indique à la cible ***ULOG*** combien d'octets du paquet envoyer au démon de l'espace utilisateur de ***ULOG***. Si nous spécifions 100 ou plus, nous copierons 100 octets du paquet vers l'espace utilisateur, ce qui inclura l'en−tête complet en l'espérant, plus certaines données. Si nous spécifions 0, le paquet complet sera copié vers l'espace utilisateur, sans faire attention à la taille des paquets. La valeur par défaut est 0, ainsi l'ensemble du paquet sera copié vers l'espace utilisateur.
Option	***−−ulog−qthreshold***
Exemple	***iptables −A INPUT −p TCP −−dport 22 −j ULOG −−ulog−qthreshold 10***
Explication	Cette option ***−−ulog−qthreshold*** indique à la cible ***ULOG*** combien de paquets seront en attente dans le noyau avant d'envoyer les données vers l'espace utilisateur. Par exemple, si nous indiquons un seuil de 10 ou plus, le noyau accumulera 10 paquets, et les transmettra ensuite à l'espace utilisateur comme un simple message netlink multiparties. La valeur par défaut ici est à 1 à cause de la compatibilité de l'affichage précédent, le démon de l'espace utilisateur ne connaissant pas le nombre de messages multiparties précédents.

Note

Fonctionne avec les noyaux Linux 2.3, 2.4, 2.5 et 2.6.

Chapitre 12. Débogage des scripts

Un des aspects les plus importants et sous−estimés dans l'écriture de vos propres tables de règles est de savoir comment les déboguer, et retrouver les erreurs. Ce chapitre vous montrera quelques étapes de base dans le débogage de vos scripts, ainsi que certaines choses un peu plus élaborées pour éviter les problèmes de connexion à votre pare−feu lorsque vous avez accidentellement exécuté une règle incorrecte.

Tous ce que nous écrivons ici est fondé sur la supposition que les tables de règles de vos scripts sont écrites en shell Bash, mais elles peuvent facilement s'appliquer à d'autres environnements. Les tables de règles qui ont été sauvegardées avec ***iptables−save*** sont une autre partie du code. Les fichiers ***iptables−save*** sont simples et ne contiennent pas de code écrit à la main qui créent des règles spécifiques, ils sont aussi plus simples à déboguer.

12.1. Déboguer, une nécessité

Déboguer est plus ou moins une nécessité avec Iptables et Netfilter et les pare−feux en général. Le problème avec 99% des pare−feux est qu'à la fin c'est un comportement humain qui décide des stratégies et de la façon dont les tables de règles sont créées, et je peux vous promettre, qu'il est facile de faire une erreur dans l'écriture de vos tables. Parfois, ces erreurs sont très difficiles à voir à l'oeil nu, ou découvrir les trous laissés par le pare−feu. Les trous que nous ne connaissons pas ou qui ne sont pas intentionnels peuvent causer des dégâts dans nos réseaux, et créer de faciles entrées pour les attaquants. La plupart de ces trous peuvent être

découverts aisément avec quelques bons outils.

En dehors de ça, nous pouvons écrire des bogues dans nos scripts, qui peuvent nous interdire d'ouvrir une session sur le pare−feu. Ceci peut également être résolu avec un peu de dextérité avant d'exécuter les scripts. Utiliser la pleine puissance du langage de script et de l'environnement système peut se révéler incroyablement utile, avec toutes les expériences que les administrateurs Unix ont déjà consignées, et c'est tout ce que nous faisons lors du débogage de nos scripts.

12.2. Débogage en Bash

Il y a certaines choses qui peuvent être faites avec Bash pour nous aider à déboguer nos scripts contenant les tables de règles. Un des premiers problèmes quand on trouve un bogue est de savoir dans quelle ligne il se trouve. Ceci peut être résolu de deux façons différentes, soit en utilisant le fanion −x du Bash, soit en tapant simplement ***echo*** pour trouver l'endroit où le problème apparaît. Idéalement, vous pouvez avec ***echo***, ajouter quelque chose comme les états suivants à intervalles réguliers dans le code :

```
...
echo "Debugging message 1."
...
echo "Debugging message 2."
...
```

Dans mon cas, j'utilise des messages insignifiants, tant qu'ils ont quelque chose d'unique je peux retrouver le message d'erreur par un simple grep ou une recherche dans le script. Maintenant, si le message d'erreur apparaît après le "Debugging message 1", mais avant le "Debugging message 2", nous savons alors que la ligne de code erronée est quelque part entre les deux messages. Comme vous pouvez le comprendre, Bash n'est pas réellement mauvais, et a au moins l'idée de continuer à exécuter les commandes même si il y a une erreur dans une commande précédente. Dans Netfilter, ceci peut provoquer certains problèmes très intéressants. L'idée d'utiliser les états echo pour trouver les erreurs est très simple, mais vous pouvez en même temps cerner l'ensemble du problème à une seule ligne de code.

La seconde possibilité pour trouver le problème est d'utiliser la variable Bash −x, comme dit précédemment. Ceci peut bien sûr être gênant particulièrement si votre script est important, et que le tampon de votre console n'est pas assez grand. Ce que la variable −x indique est très simple, elle précise au script d'envoyer un écho pour chaque ligne de code de votre script vers la sortie standard du shell (généralement la console). Pour cela changer la ligne de début du script qui doit se présenter comme :

```
#!/bin/bash
```

En la ligne suivante :

```
#!/bin/bash -x
```

Comme vous pouvez le voir, ceci ne modifie peut être que deux lignes, dans le total important de données en sortie. Le code vous indique chaque ligne de commande qui est exécutée, et avec quelles valeurs de variables, etc. Chaque ligne exécutée est affichée sur votre écran. Une chose intéressante, est que les lignes de sortie Bash sont préfixées par un signe +. Ceci rend plus facile de discerner les messages d'erreur et d'avertissement provenant du script.

L'option −x est aussi très intéressante pour déboguer les problèmes communs qu'on rencontre dans les tables de règles plus complexes. Le premier est de trouver ce qui se passe avec ce que vous pensiez être une simple boucle. Voyons l'exemple ci−dessous.

```
#!/bin/bash
iptables="/sbin/iptables"
$iptables -N output_int_iface
cat /etc/configs/machines | while read host; do
  $iptables -N output-$host
  $iptables -A output_int_iface -p tcp -d $host -j output-$host

  cat /etc/configs/${host}/ports | while read row2; do
    $iptables -A output-$host -p tcp --dport $row2 -d $host -j ACCEPT
  done
done
```

Ces règles peuvent sembler simples, mais le problème existe toujours. Nous obtenons les messages d'erreur suivants que nous savons provenir du code ci–dessus en utilisant un simple écho comme méthode de débogage.

```
work3:~# ./test.sh
Bad argument `output-'
Try `iptables -h' or 'iptables --help' for more information.
cat: /etc/configs//ports: No such file or directory
```

Activons l'option –x du Bash et regardons la sortie. Celle–ci est indiquée ci–dessous, comme vous pouvez le voir il y a quelque chose de très mystérieux. Il y a un couple de commandes où les variables $host et $row2 ne sont remplacées par rien. En regardant de plus près, nous voyons que c'est seulement la dernière itération du code qui cause problème. Soit nous avons fait une erreur de programmation, soit il y a quelque chose d'étrange avec la donnée. Dans ce cas c'est une simple erreur avec la donnée, qui contient un saut de ligne en trop à la fin du fichier. Ceci crée une boucle d'itération. En supprimant simplement ce saut de ligne du fichier, le problème est résolu. Ceci peut ne pas être une solution très élégante, mais pour un usage privé c'est suffisant. D'un autre côté nous avons ajouté du code qui indique qu'il y a certaines données dans les variables $host et $row2.

```
work3:~# ./test.sh
+ iptables=/sbin/iptables
+ /sbin/iptables -N output_int_iface
+ cat /etc/configs/machines
+ read host
+ /sbin/iptables -N output-sto-as-101
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-101 -j output-sto-as-101
+ cat /etc/configs/sto-as-101/ports
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 21 -d sto-as-101 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 22 -d sto-as-101 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 23 -d sto-as-101 -j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-sto-as-102
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-102 -j output-sto-as-102
+ cat /etc/configs/sto-as-102/ports
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 21 -d sto-as-102 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 22 -d sto-as-102 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 23 -d sto-as-102 -j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-sto-as-103
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-103 -j output-sto-as-103
```

```
+ cat /etc/configs/sto-as-103/ports
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 21 -d sto-as-103 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 22 -d sto-as-103 -j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 23 -d sto-as-103 -j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-
+ /sbin/iptables -A output_int_iface -p tcp -d -j output-
Bad argument `output-'
Try `iptables -h' or 'iptables --help' for more information.
+ cat /etc/configs//ports
cat: /etc/configs//ports: No such file or directory
+ read row2
+ read host
```

Le troisième et dernier problème peut être partiellement résolu avec l'aide de l'option –x si vous exécutez le script du pare–feu par SSH, la console se suspend au milieu de l'exécution du script, elle ne vous rend pas la main, ni vous ne pouvez vous connecter par SSH à nouveau. Dans 99% des cas, ceci indique qu'il y a certains problèmes dans le script avec un couple de règles. En activant l'option –x, vous verrez exactement à quelle ligne ça bloque. Il y a une ou deux circonstances où ce n'est pas vrai, malheureusement. Par exemple, si le script initialise une règle qui bloque le trafic entrant, mais que le serveur ssh/telnet envoit un écho en premier comme trafic sortant, netfilter enregistrera la connexion, et donc permettra le trafic entrant de toutes destinations si vous avez une règle qui maintient les états de connexion.

Comme vous pouvez le voir, il peut être tout à fait complexe de déboguer vos tables de règles. Cependant, ce n'est pas impossible. Vous pouvez aussi avoir noté, si vous travaillez à distance sur votre pare–feu via SSH par exemple, que le pare–feu peut se figer quand vous charger des règles incorrectes. Une des choses supplémentaires que vous pouvez faire dans ces circonstances, est de faire une sauvegarde par jour. Cron est un excellent moyen pour faire ça. Exemple, vous travaillez sur un pare–feu en étant à 50 km de distance, vous ajoutez et supprimez des règles. Le pare–feu se bloque, et vous ne pouvez plus rien faire. le seul moyen est de vous déplacer à l'endroit où se trouve physiquement le pare–feu et régler le problème, à moins que vous n'ayez pris des précautions de ce genre !

12.3. Outils système pour le débogage

Une des meilleures précautions que vous pouvez prendre contre un pare–feu qui se bloque, est de tout simplement utiliser le programme cron pour ajouter un script qui se lance toutes les 5 minutes ou qui relance le pare–feu, et ensuite supprimer la ligne du cron une fois que vous êtes sûrs que l'installation fonctionne bien. La ligne du cron peut ressembler à l'exemple ci–dessous et elle sera installée par la commande ***crontab –e***.

```
*/5 * * * * /etc/init.d/rc.flush-iptables.sh stop
```

Soyez absolument sûrs, que la ligne fonctionne et fait exactement ce que vous en attendez sinon ça peut bloquer le serveur.

Un autre outil qui est constamment utilisé pour déboguer les scripts est la fonction syslog. C'est ce qui journalise tous les messages générés par beaucoup de programmes différents. En fait, la plupart des gros programmes supportent la journalisation par syslog, y compris le noyau. Tous les messages envoyés à syslog ont deux variables de base dont il est très important de se souvenir, la fonction et le niveau/priorité de la journalisation.

La fonction indique au serveur syslog de quel endroit provient l'entrée du journal, et quoi journaliser. Il existe

plusieurs fonctions spécifiques, mais celle qui nous intéresse est la fonction Kern, ou fonction kern. Tous les messages générés par netfilter sont envoyés par cette fonction.

Le niveau de journalisation indique à syslog quelle priorité ont les messages. Il y a plusieurs priorités, voir ci−dessous.

1. debug
2. info
3. notice
4. warning
5. err
6. crit
7. alert
8. emerg

En fonction de ces priorités, nous pouvons les envoyer vers différents fichiers journaux en utilisant le syslog.conf. Par exemple, pour envoyer tous les messages provenants de la fonction kern avec une priorité warning vers un fichier appelé /var/log/kernwarnings, nous ferons comme ci−dessous. La ligne sera placée dans /etc/syslog.conf.

```
kern.warning                        /var/log/kernwarnings
```

Comme vous pouvez le voir, c'est tout à fait simple. Maintenant, vous trouverez vos journaux de netfilter dans le fichier /var/log/kernwarnings (après redémarrage, ou en faisant un HUP sur le serveur syslog). Bien sûr, ceci dépend du niveau de journalisation que vous avez mis dans vos règles de netfilter. Le niveau de journalisation peut être placé avec l'option −−log−level.

Ces journaux vous fourniront l'information que vous désirez via les règles de journalisation spécifiques dans la table de règles. Avec elles, vous pouvez voir si il existe un problème quelque part. Par exemple, vous pouvez placer vos règles de journalisation à la fin des chaînes pour voir s'il y a des paquets qui ont transité jusqu'à la frontière de vos chaînes. Une entrée de journal peut ressembler à l'exemple ci−dessous, et inclure les informations suivantes.

```
  Oct 23 17:09:34 localhost kernel: IPT INPUT packet died: IN=eth1 OUT=
MAC=08:00:09:cd:f2:27:00:20:1a:11:3d:73:08:00 SRC=200.81.8.14 DST=217.215.68.146
LEN=78 TOS=0x00 PREC=0x00 TTL=110 ID=12818 PROTO=UDP SPT=1027 DPT=137 LEN=58
```

Comme vous pouvez le comprendre, syslog peut réellement vous aider à déboguer vos tables de règles. Regarder ces journaux peut vous aider à comprendre pourquoi les ports que vous voulez ouvrir ne fonctionnent pas.

12.4. Débogage d'Iptables

Iptables peut être parfois difficile à déboguer, car les messages d'erreur provenant d'iptables lui−même ne sont pas toujours conviviaux. Pour cette raison, ce peut être une bonne idée de regarder les messages d'erreur les plus fréquents venant d'iptables, et pourquoi vous les obtenez.

Un des premiers messages d'erreur à regarder est le "Unknown arg". Il peut apparaître pour différentes raisons. Exemple ci−dessous.

```
work3:~# iptables -A INPUT --dport 67 -j ACCEPT
iptables v1.2.9: Unknown arg `--dport'
Try `iptables -h' or 'iptables --help' for more information.
```

Cette erreur est simple à corriger, car nous n'avons utilisé qu'un seul argument. Normalement, nous pouvons avoir utilisé une très longue commande et obtenir ce message. Le problème dans le scenario ci–dessus est que nous avons oublié d'utiliser la correspondance ––protocol, et à cause de ça, le module ––dport n'est pas disponible. En ajoutant la correspondance ––protocol nous résoudrons le problème. Soyez absolument certains que vous n'oubliez aucune pré–condition spéciale nécessaire pour utiliser une correspondance spécifique.

Une autre erreur très commune est que vous oubliez un tiret (–) quelque part dans la ligne de commande, comme en–dessous. La solution est de rajouter simplement ce tiret, et la commande fonctionnera.

```
work3:~# iptables -A INPUT --protocol tcp -dport 67 -j ACCEPT
Bad argument `67'
Try `iptables -h' or 'iptables --help' for more information.
```

Et enfin, le simple oubli, ce qui est le plus courant. Voir ci–dessous. Le message d'erreur, est exactement le même que lorsque vous oubliez d'ajouter une correspondance pré–requise à votre règle, aussi il est nécessaire de les regarder de près.

```
work3:~# iptables -A INPUT --protocol tcp --destination-ports 67 -j ACCEPT
iptables v1.2.9: Unknown arg `--destination-ports'
Try `iptables -h' or 'iptables --help' for more information.
```

Il existe aussi une cause plus probable quand vous obtenez l'erreur "Unknown arg". Si l'argument est écrit correctement, et qu'il n'y a pas d'erreur dans les pré–requis, il est possible que la cible/correspondance/table ne soit pas compilée dans le noyau. Par exemple, nous oublions de compiler la table filter dans le noyau, ce qui ressemblera à ça.

```
work3:~# iptables -A INPUT -j ACCEPT
iptables v1.2.9: can't initialize iptables table `filter': Table does not exist
(do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

Normalement, iptables est capable de charger automatiquement un module spécifique qui n'est pas déjà dans le noyau, c'est généralement le signe que soit vous n'avez pas fait un depmod correct après avoir redémarré avec le nouveau noyau, soit vous avez simplement oublié le(s) module(s). Si le module problématique est une correspondance, le message d'erreur sera plus crypté et difficile à interpréter. Par exemple, regardons ce message d'erreur.

```
work3:~# iptables -A INPUT -m state
--state ESTABLISHED -j ACCEPT
iptables: No chain/target/match by that name
```

Dans ce cas, nous avons oublié de compiler le module, et comme vous avez vu, le message d'erreur n'est pas très facile à interpréter. Mais il fait allusion à ce qui est faux. Finalement, nous avons la même erreur de nouveau, mais cette fois, la cible est omise. Comme vous l'avez compris en regardant le message, il est plus compliqué car c'est exactement le même pour les deux erreurs (correspondance et/ou cible oubliée).

```
work3:~# iptables -A INPUT -m state
--state ESTABLISHED -j REJECT
iptables: No chain/target/match by that name
```

Le moyen le plus simple de savoir si vous avez oublié de faire un depmod, ou si le module est manquant, est de regarder dans le répertoire où se trouvent les modules. C'est le répertoire

`/lib/modules/2.6.4/kernel/net/ipv4/netfilter`. Tous les fichiers ipt_* écrits en majuscules sont des cibles, et tous les autres en minuscules sont des correspondances. Par exemple, ipt_REJECT.ko est une cible, tandis que ipt_state.ko est une correspondance.

Note

Dans les noyaux 2.4 et plus anciens, l'extension de fichier pour tous les modules du noyau étaient .o, qui a été changée pour les fichiers des noyaux 2.6 en .ko.

Une autre façon d'obtenir de l'aide d'iptables lui−même est de commenter une chaîne entière dans votre script pour savoir si ça résoud le problème. En supprimant la chaîne et mettant à la place une stratégie par défaut (ACCEPT), et ensuite en testant, si ça marche mieux, c'est que c'était cette chaîne qui causait les problèmes. Si rien ne s'améliore, alors c'est une autre chaîne, et vous devez chercher le problème ailleurs.

12.5. Autres outils de débogage

Il existe bien sûr d'autres outils qui peuvent être extrêmement utiles pour déboguer vos scripts. Cette section présente brièvement les plus communs qui sont utilisés pour savoir comment apparaît votre pare−feu dans les deux sens (interne et externe). Les outils que j'ai choisi sont ici Nmap et Nessus.

12.5.1. Nmap

Nmap est un excellent outil vu dans une perspective de pare−feu, pour trouver quels ports sont ouverts et un niveau d'information de plus bas niveau. Il possède un support d'empreinte système, plusieurs méthodes différentes de balayage de port, le support de IPv6 et IPv4 et le balayage de réseau.

La forme de base du balayage est exécutée avec une syntaxe de ligne de commande très simple. N'oubliez pas de spécifier les ports à scanner avec l'option −p, par exemple −p 1−1024. Voir ci−dessous.

```
blueflux@work3:~$ nmap -p 1-1024 192.168.0.1

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-03-18 17:19 CET
Interesting ports on firewall (192.168.0.1):
(The 1021 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
587/tcp open  submission

Nmap run completed -- 1 IP address (1 host up) scanned in 3.877 seconds
```

Il est aussi capable de deviner le système d'exploitation de l'hôte scanné par l'empreinte système (fingerprinting). Le fingerprinting nécessite les privilèges administrateur (root), mais il peut aussi être très intéressant à utiliser pour savoir ce que la plupart des gens voient sur l'hôte. Fingerprinting peut ressembler à ça :

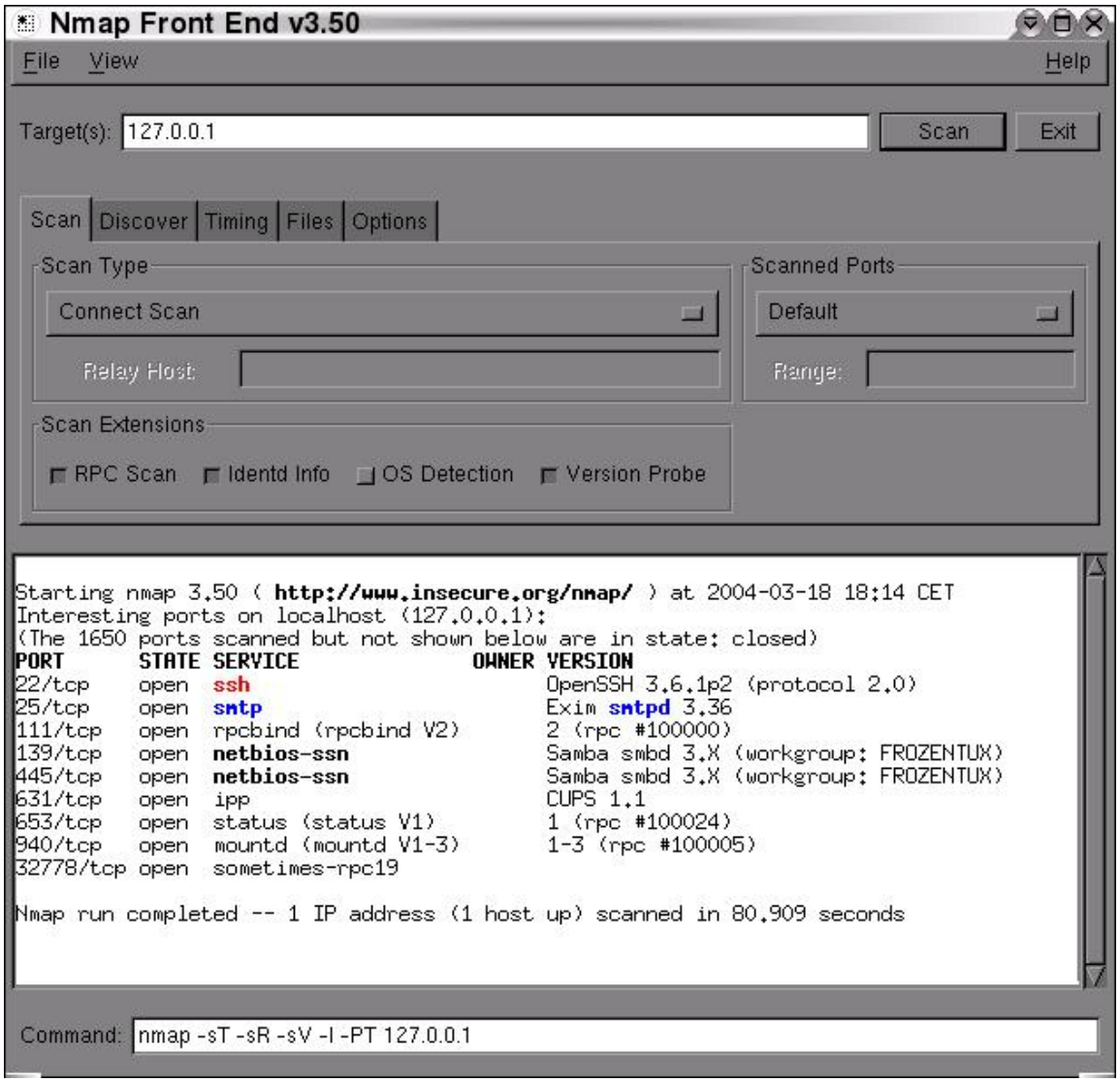
```
work3:/home/blueflux# nmap -O -p 1-1024 192.168.0.1

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-03-18 17:38 CET
Interesting ports on firewall (192.168.0.1):
(The 1021 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
587/tcp open  submission
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 6.201 days (since Fri Mar 12 12:49:18 2004)
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 14.303 seconds
```

Le fingerprinting n'est pas parfait, comme vous pouvez le voir, mais il est très utile, pour vous et pour un attaquant. Il faut le savoir. La meilleure chose à faire, est de montrer le moins de choses possibles, donc avec cet outil vous saurez ce qu'un attaquant peut voir de votre machine.

Nmap possède également une interface graphique, appelée nmapfe (Nmap Front End). C'est une excellente interface, et si vous avez besoin de faire des recherches un peu plus complexes, vous pourrez l'utiliser avec bonheur. Un exemple de capture d'écran :



Bien sûr, Nmap a d'autres fonctions que vous pouvez découvrir sur le site. Pour plus d'information, regardez Nmap.

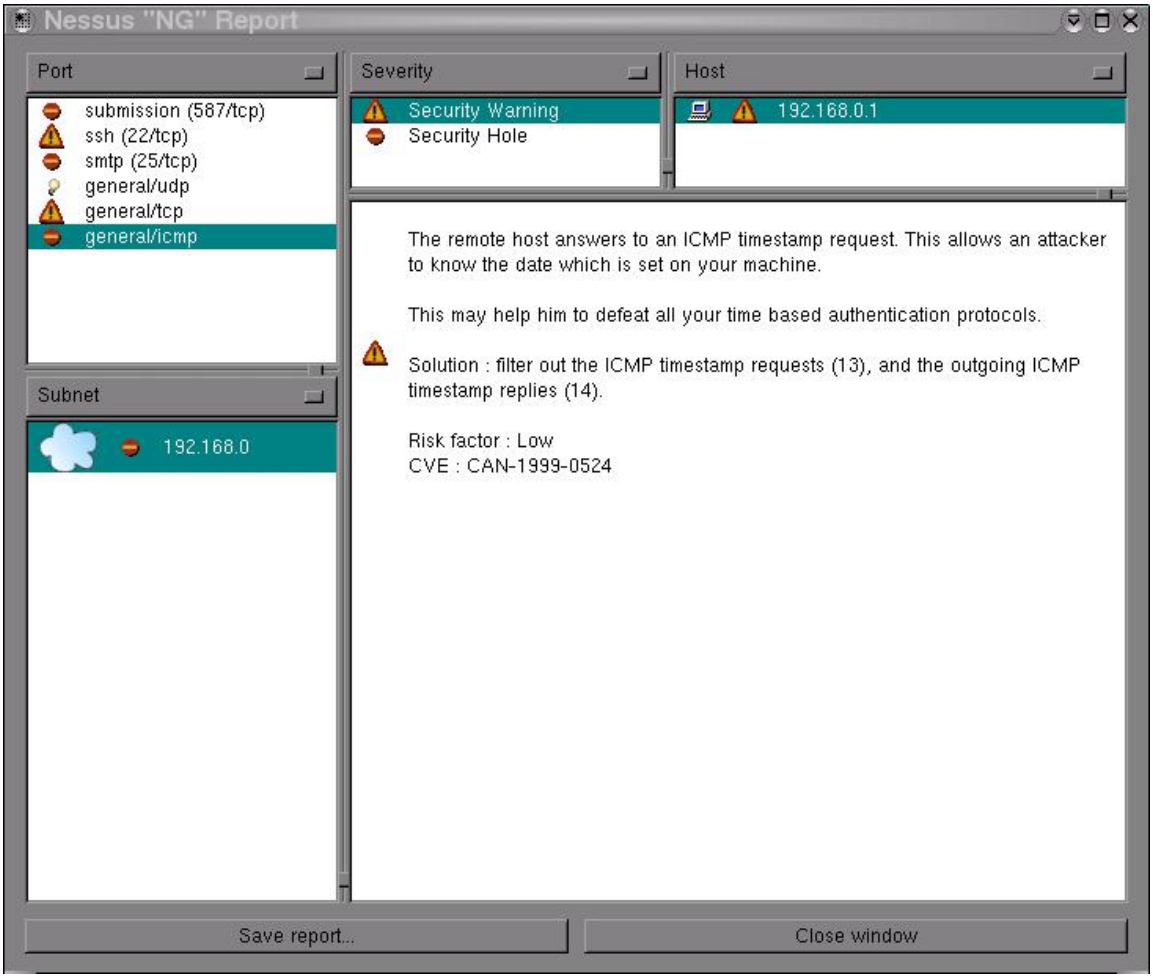
Comme vous l'avez compris, c'est un excellent outil pour tester votre hôte, et trouver les ports ouverts et ceux qui sont fermés. Par exemple, après avoir terminé vos réglages, utilisez nmap pour savoir si ce que vous avez fait correspond à votre attente. Obtenez vous les réponses correctes des bons ports, et ainsi de suite.

12.5.2. Nessus

Alors que Nmap est plus un scanner de bas niveau, indiquant les ports ouverts, etc., le programme Nessus est un scanner de sécurité. Il tente de se connecter à différents ports, et de trouver en plus, les versions des

serveurs en activité. Nessus effectue cela une étape plus loin, en trouvant les ports ouverts, ce qui est actif et sur quels ports, quel programme et quelle version, et ensuite teste les diverses menaces pour la sécurité dans les programmes, et finalement crée un rapport complet de toutes les menaces concernant la sécurité.

Comme vous pouvez le comprendre, c'est un outil extrêmement utile. Le programme fonctionne sur le modèle client/serveur, ainsi il est plus facile d'en connaître d'avantage sur votre pare−feu depuis l'extérieur en utilisant le démon Nessus, ou en interne de cette manière. Le client est une interface graphique dans laquelle vous vous connectez au démon, réglez vos paramètres, et spécifiez quel hôte scanner. Le rapport généré peut ressembler à l'exemple ci−dessous.



Attention

Nessus devrait être utilisé avec certaines précautions, car il peut faire "planter" une machine ou un service spécifié par une attaque. Ces attaques sur les machines sont désactivées par défaut, heureusement.

12.6. Le chapitre suivant

Dans ce chapitre nous avons vu diverses techniques que vous pouvez utiliser pour déboguer vos scripts de pare−feu. Le débogage de scripts peut devenir fatiguant à la longue, mais c'est une nécessité. Si vous utilisez les exemples ça peut être très facile. Nous avons vu les techniques suivantes en particulier :

- Débogage par le Bash
- Outils système pour le débogage
- Débogage d'Iptables
- Autres outils pour le débogage

Chapitre 13. Fichier rc.firewall

Ce chapitre présente un exemple de configuration de pare−feu et un fichier script. Nous avons utilisé une des configurations de base et regardé comment ça fonctionne et ce que nous pouvons faire avec. Ceci vous donnera une idée de comment résoudre les différents problèmes. Il peut être utilisé tel quel en changeant les variables, mais je ne le vous conseille pas car il peut ne pas fonctionner avec la configuration de votre réseau.

Note

Notez qu'il y a des moyens plus efficaces de configurer des tables de règles, cependant, le script a été écrit pour plus de lisibilité sans devoir connaître en détail la syntaxe du Bash.

13.1. Exemple de rc.firewall

Cet exemple rc.firewall.txt (également inclu dans Example scripts code−base) est assez important mais n'a pas beaucoup de commentaires. Au lieu de regarder les commentaires, je suggère que vous lisiez le script pour voir à quoi il ressemble, et ensuite revenir ici pour le détailler.

13.2. Explication du rc.firewall

13.2.1. Options de configuration

La première section de l'exemple rc.firewall.txt est la section de configuration. Elle devra toujours être modifiée car elle contient des informations vitales pour votre configuration. Par exemple, votre adresse IP changera. Le ***$INET_IP*** devra toujours être une adresse IP totalement valide, si vous en avez une (sinon regardez de plus près le rc.DHCP.firewall.txt). Également, la variable ***$INET_IFACE*** devra pointer vers le matériel utilisé pour votre connexion Internet. Ce peut être eth0, eth1, ppp0, tr0, etc., pour citer quelques noms d'interfaces.

Ce script ne contient aucune option de configuration spéciale pour DHCP ou PPPoE, c'est pourquoi ces sections sont vides. Même chose pour toutes les sections vides, elles sont cependant indiquées, ainsi vous pouvez voir la différence entre les scripts de façon plus efficace. Si vous avez besoin de ces parties, vous pouvez toujours créer un mélange des différents scripts, ou créer le votre entièrement.

La section concernant votre réseau local contient la plupart des options de configuration nécessaires. Par exemple, vous avez besoin de spécifier l'adresse IP d'une interface physique connectée au LAN de même que la plage IP que le LAN utilise et l'interface par laquelle la machine est connectée au réseau.

Ainsi, vous pouvez voir qu'il y a une section pour la configuration de l'hôte local. Nous vous la fournissons, cependant vous n'effecturez à 99% aucun changement dans ces valeurs car on utilise presque toujours l'adresse IP 127.0.0.1 et l'interface nommée lo. Juste après la configuration de l'hôte local, vous trouverez une brève section qui appartient à iptables. Cette section concerne les variables ***$IPTABLES***, qui pointent le script vers l'endroit exact où se trouve l'application ***iptables***. Ceci peut varier un peu, et l'endroit par défaut lors de la compilation à la main est `/usr/local/sbin/iptables`. Cependant, plusieurs distributions placent l'application à un autre endroit comme `/usr/sbin/iptables` ou `/sbin/iptables`.

13.2.2. Chargement initial des modules supplémentaires

Premièrement, regardons si les fichiers des dépendances des modules sont à jour en exécutant la commande ***/sbin/depmod −a***. Après ça chargeons les modules nécessaires au script. Évitez toujours de charger des modules dont vous n'avez pas besoin. C'est pour des raisons de sécurité, car il sera plus difficile d'établir des règles supplémentaires de cette façon. Maintenant, par exemple, si vous voulez avoir le support des cibles ***LOG***, ***REJECT*** et ***MASQUERADE*** et ne les avez pas compilées statiquement dans le noyau, vous devrez charger ces modules comme suit :

```
/sbin/insmod ipt_LOG
/sbin/insmod ipt_REJECT
/sbin/insmod ipt_MASQUERADE
```

Attention

Dans ces scripts, nous chargeons les modules de force, ce qui peut conduire à des problèmes. Si un module ne se charge pas, ce qui peut dépendre de plusieurs facteurs, il enverra un message d'erreur. Si certains modules les plus basiques ne se chargent pas, l'erreur la plus probable est que le module, ou la fonctionnalité, est compilée statiquement dans le noyau. Pour plus d'information sur ce sujet, lisez la section Problèmes de chargement des modules dans l'annexe Problèmes et questions courants.

Ensuite c'est le module `ipt_owner` à charger, qui peut, par exemple, être utilisé pour permettre à certains utilisateurs de réaliser certaines connexions, etc. Je n'utilise pas ce module dans l'exemple, mais vous pouvez autoriser seulement root à se connecter en FTP et HTTP à redhat.com et ***DROP*** tous les autres. Vous pouvez aussi interdire tous les utilisateurs sauf vous et root de se connecter depuis votre machine à l'Internet. Ça peut être ennuyeux pour les autres, mais vous aurez plus de sécurité par rapport aux attaques où le hacker utilise seulement votre machine comme hôte intermédiaire. Pour plus d'information sur `ipt_owner` regardez la section Correspondance owner dans le chapitre Création d'une règle.

Vous pouvez aussi charger des modules supplémentaires pour le code de correspondance d'état ici. Tous les modules additionnels au code de correspondance d'état et au code de traçage de connexion sont appelés ip_conntrack_* et ip_nat_*. Les assistants de traçage de connexion sont des modules spéciaux qui indiquent au noyau comment tracer correctement les connexions spécifiques. Sans ces assistants, le noyau ne sait pas quoi chercher quand il essaie de tracer des connexions. Les assistants NAT d'un autre côté, sont des extensions des assistants de traçage de connexion qui indiquent au noyau que rechercher dans des paquets spécifiques et comment traduire ceux–ci dans les connexions en cours. Par exemple, FTP est un protocole complexe par définition, il envoit des informations de connexion dans les données utiles du paquet. Donc, si une de vos machines NATées se connecte à un serveur FTP sur l'Internet, elle enverra sa propre adresse IP du réseau local dans les données utiles du paquet, et indiquera au serveur FTP de se connecter à cette adresse. Étant donné que adresse du réseau locale n'est pas valide en dehors de votre propre réseau, le serveur FTP ne saura pas que faire avec elle et la connexion sera coupée. Les assistants FTP NAT font les traductions qui permettent au serveur FTP de savoir où se connecter. La même chose s'applique pour les transferts de fichiers en DCC dans les chats irc. Créer ce genre de connexions nécessite une adresse IP et des ports à envoyer au protocole IRC, lequel en retour demande que certaines traductions soient faites. Sans ces assistants, FTP et IRC fonctionneront sans doute, cependant, certaines autres choses ne marcheront pas. Par exemple, vous pouvez recevoir des fichiers par DCC, mais pas en envoyer. Ceci est dû à la façon dont DCC démarre une connexion. En premier, vous indiquez au destinataire que vous voulez envoyer un fichier et où il devra se connecter. Sans les assistants, la connexion DCC ressemblera à une tentative du receveur de connecter certains hôtes au propre réseau local de ce receveur. La connexion sera brisée. Cependant, ça peut fonctionner sans défaut, car l'expéditeur vous enverra (probablement) la bonne adresse pour vous connecter.

Note

Si vous rencontrez des problèmes avec le DCC de mIRC et que tout fonctionne correctement avec d'autres clients IRC, lisez la section Problèmes avec le DCC de mIRC dans l'appendice Problèmes et questions courants.

Comme nous l'avons écrit, c'est seulement l'option de chargement des modules qui ajoute le support pour les protocoles FTP et IRC. Pour une explication plus détaillée des modules conntrack et nat, lisez l'annexe Problèmes et questions courants. Il existe aussi les assistants conntrack H.323 dans le patch–o–matic, comme d'autres assistants conntrack et NAT. Pour pouvoir vous en servir, vous devez utiliser le patch–o–matic et compiler votre propre noyau. Pour une explication plus complète, voir le chapitre Préparatifs.

Note

Notez que vous devrez charger ip_nat_irc et ip_nat_ftp si vous voulez que la traduction d'adresse réseau fonctionne correctement avec les protocoles FTP et IRC. Vous aurez également besoin de charger les modules ip_conntrack_irc et ip_conntrack_ftp avant de charger les modules NAT. Ils sont utilisés de la même façon que les modules conntrack, mais ils vous permettront de faire du NAT sur ces deux protocoles.

13.2.3. Réglage du proc

Ici nous démarrons le `IP forwarding` par un écho à 1 sur `/proc/sys/net/ipv4/ip_forward` de cette façon :

echo "1" > /proc/sys/net/ipv4/ip_forward

Avertissement

Il peut être intéressant de réfléchir où et quand nous devons placer l'IP forwarding (transfert IP). Dans ce script, et dans tous les autres de ce didacticiel, nous le plaçons avant de créer les autres filtres IP (i.e., les règles de ***iptables***). Ceci conduit à une brève période de temps pendant laquelle le pare−feu accepte le transfert de tout le trafic, entre une milliseconde et une minute selon le script. Ceci peut permettre à des personnes malicieuses d'essayer de passer le pare−feu. En d'autres termes, cette option doit être activée *après* la création de toutes les règles, cependant, j'ai choisi de l'activer avant de charger d'autres règles pour des raisons de concordance avec le script.

Dans le cas où vous avez besoin du support d'adresse IP dynamique, par exemple vous utilisez SLIP, PPP ou DHCP, vous devez acitver l'option `ip_dynaddr` en faisant :

echo "1" > /proc/sys/net/ipv4/ip_dynaddr

S'il y a d'autres options que vous voulez activer vous suivez cette procédure. Il existe d'autres documentations sur ces sujets mais c'est hors du sujet de ce didacticiel. Il existe un bon mais bref document sur le système proc disponible dans le noyau, également disponible dans l'annexe Autres ressources et liens. L'annexe Autres ressources et liens est généralement un bon endroit pour rechercher l'information que vous ne trouvez pas ici.

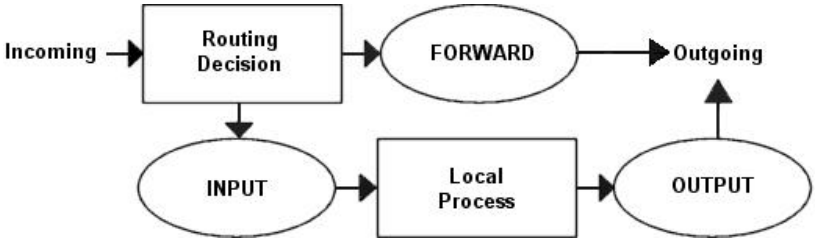
Note

Le script `rc.firewall.txt`, et tous les autres scripts de ce didacticiel, contient une petite section sur la mise en place des proc qui ne sont pas requises. Ce peut être là qu'il faut regarder quand quelque chose ne fonctionne pas comme vous le voulez, cependant, ne changez pas les valeurs avant de savoir ce qu'elles représentent.

13.2.4. Déplacement des règles vers différentes chaînes

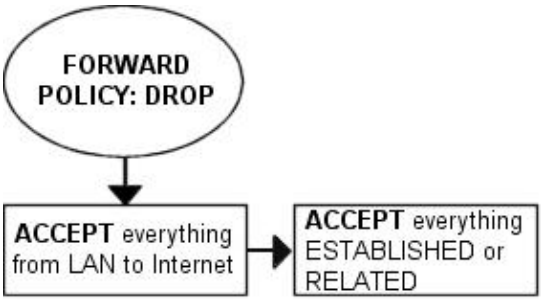
Cette section décrit brièvement mes choix en fonction des scripts spécifiques du `rc.firewall.txt`. Certains des chemins d'accès indiqués ici peuvent être faux selon un ou un autre aspect. Aussi, cette section jette en bref regard en arrière sur le chapitre Traversée des tables et des chaînes. Nous nous souviendrons de la façon dont les tables et les chaînes sont traversées dans un exemple réel.

J'ai modifié toutes les différentes chaînes utilisateur de façon à économiser le plus possible de CPU, mais en même temps en mettant l'accent principal sur la sécurité et la fiabilité. Au lieu de laisser un paquet TCP traverser les règles ICMP, UDP et TCP, j'ai simplement apparié tous les paquets TCP et laissé ces paquets TCP traverser les chaînes spécifiées par l'utilisateur. De cette façon nous ne consommons pas trop de temps système. L'image ci−dessous tente d'expliquer comment un paquet entrant traverse Netfilter. Avec ces images et explications, j'essaie de clarifier les buts de ce script. Nous ne verrons cependant aucun détail spécifique, ceci sera fait plus loin dans le chapitre. C'est un image plutôt triviale en comparaison de celle du chapitre Traversée des tables et des chaînes où nous parlons de la traversée complète des tables et des chaînes en profondeur.

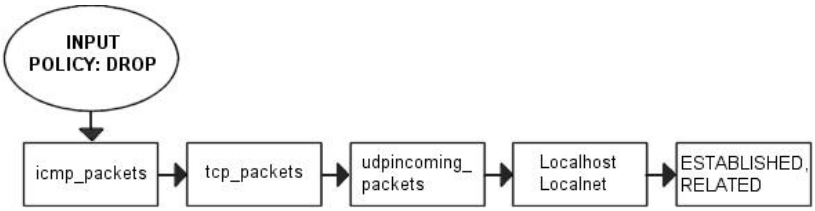


Nous fondant sur cette image, clarifions nos buts. C'est un exemple complet de script basé sur la supposition que notre scenario contient un réseau local, un pare−feu et une connexion Internet connectée à ce pare−feu. Cet exemple est aussi basé sur la supposition que nous avons une adresse IP statique vers l'Internet (à l'opposé de DHCP, PPP, SLIP et autres). Dans ce cas, nous voulons autoriser le pare−feu à agir comme serveur pour certains services sur l'Internet, et nous faisons pleinement confiance à notre réseau local et donc ne bloquons pas le trafic provenant de celui−ci. Pour faire ceci, nous plaçons les stratégies des chaînes par défaut à DROP. Ce qui effectivement bloquera toutes les connexions et tous les paquets que nous n'avons pas explicitement autorisés dans notre réseau ou notre pare−feu.

Dans le cas de ce scenario, nous laisserons notre réseau local se connecter à l'Internet. Comme nous avons pleine confiance dans notre réseau, nous permettons toute sorte de trafic depuis ce réseau local vers l'Internet. Cependant, l'Internet n'est pas un réseau de confiance et donc nous voulons bloquer les connexions venant de celui−ci et allant vers notre réseau. En fonction de ces suppositions, regardons ce que nous devons faire et ne pas faire.



En priorité, nous voulons que notre réseau local puisse se connecter à l'Internet, bien sûr. Pour cela, nous devons NATer tous les paquets car aucune de nos machines locales n'a d'adresse IP routable. Tout ceci est effectué dans la chaîne POSTROUTING, créée en dernier dans le script. Nous devons aussi faire du filtrage dans la chaîne FORWARD car nous devons permettre un accès complet à notre réseau local. Nous avons confiance dans notre réseau local, et pour ça nous autorisons tout le trafic provenant de celui−ci et allant vers l'Internet. Comme personne sur l'Internet ne sera autorisé à se connecter aux ordinateurs de notre réseau local, nous bloquerons tout le trafic provenant de l'Internet vers le réseau local sauf les connexions déjà établies, qui autorisent le trafic en réponse.



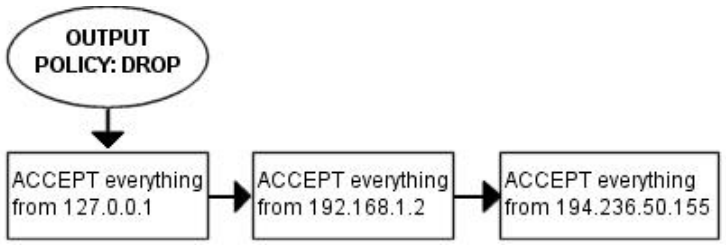
Comme pour notre pare−feu, nous n'avons peut−être pas trop de moyens, et ne voulons offrir que quelques services sur l'Internet. Nous avons décidé de permettre les accès HTTP, FTP, SSH et IDENTD à notre pare−feu. Tous ces protocoles sont disponibles dans le pare−feu, et seront donc autorisés par la chaîne INPUT, ainsi que nous autoriserons le trafic en retour à travers la chaîne OUTPUT. Cependant, nous avons pleinement confiance dans notre réseau local, et le matériel local et l'adresse IP sont également sûrs. Donc, nous pouvons ajouter des règles spéciales pour permettre le trafic depuis le réseau local comme depuis la boucle locale. De même, nous n'autoriserons pas certains paquets spécifiques, ni certaines plages d'adresses IP à joindre le pare−feu depuis l'Internet. Par exemple, la plage d'adresses IP 10.0.0.0/8 est réservée à notre réseau local et donc nous ne voulons pas autoriser les paquets provenant d'une de ces adresses car ils risqueraient à 99% une usurpation d'adresse. Cependant, avant d'implémenter ceci, nous devons noter que

certains fournisseurs d'accès Internet (FAI) utilisent ces plages d'adresses IP dans leur propre réseau. Pour plus de détails, voir le chapitre Problèmes et questions courants.

Comme nous avons un serveur FTP actif, et que nous voulons traverser certaines règles, nous ajoutons une règle qui permet le trafic établi et relié au début de la chaîne INPUT. Pour la même raison, nous divisons les règles en sous−chaînes. En faisant ça, nos paquets n'auront besoin de traverser que quelques chaînes. En traversant moins de chaînes, nous consommons moins de temps pour chaque paquet, et réduisons la latence dans le réseau.

Dans ce script, nous choisissons de diviser les différents paquets par leur famille de protocole, par exemple TCP, UDP ou ICMP. Tous les paquets TCP traversent une chaîne spécifique nommée tcp_packets, qui contient les règles pour tous les ports et protocoles TCP que nous voulons autoriser. Ainsi, si nous voulons faire certaines vérifications supplémentaires sur les paquets TCP, nous devrons créer une sous−chaîne pour tous les paquets acceptés qui utilisent des numéros de port valides vers le pare−feu. Cette chaîne que nous choisissons d'appeler chaîne autorisée, contiendra certaines vérifications supplémentaires avant d'accepter le paquet. Pour les paquets ICMP, ils traversent la chaîne icmp_packets. Quand nous avons décidé de créer cette chaîne, nous n'avons pas vu le besoin de vérifications supplémentaires avant d'accepter les paquets s'ils sont conformes au code ICMP, et donc les acceptons directement. Enfin, nous avons les paquets UDP qui doivent être distribués avec. Nous envoyons ces paquets vers la chaîne udp_packets qui traite tous les paquets UDP entrants. Tous les paquets UDP entrants doivent être envoyés à cette chaîne, et s'ils sont d'un type autorisé nous les acceptons immédiatement sans vérification supplémentaire.

Comme nous sommes sur un réseau relativement petit, cette machine étant également utilisée comme station de travail secondaire, nous voulons autoriser certains protocoles spécifiques à joindre le pare−feu, comme ***speak freely*** et ***ICQ***.



Enfin, nous avons la chaîne OUTPUT. Comme nous faisons confiance à notre pare−feu, nous autorisons tout le trafic quittant celui−ci. Nous ne bloquons aucun utilisateur, ni aucun protocole spécifique. Cependant, nous ne voulons pas que des personnes utilisent cette machine pour usurper les paquets quittant le pare−feu, et donc autorisons uniquement le trafic depuis les adresses assignées au pare−feu. Nous implémenterons ceci en ajoutant des règles qui ACCEPT tous les paquets quittant le pare−feu lorsque ceux−ci proviennent des adresses assignées, s'ils ne sont pas supprimés par défaut dans la chaîne OUTPUT.

13.2.5. Mise en place des actions par défaut

Très tôt dans le processus de création de nos règles, nous avons placé nos stratégies par défaut. Nous implémentons nos stratégies par défaut dans les différentes chaînes avec une commande très simple, comme décrite ci−dessous :

```
iptables [−P {chain} {policy}]
```

La stratégie par défaut est utilisée chaque fois que les paquets n'apparient pas une règle dans une chaîne. Par exemple, nous avons un paquet qui n'apparie aucune règle dans notre table de règles. Si ça se produit, nous devons décider quoi faire du paquet en question, et c'est là qu'intervient la stratégie par défaut. Elle est utilisée

sur tous les paquets qui ne s'apparient avec aucune règle dans notre table de règles.

Attention

Faîtes attention avec la stratégie par défaut que vous placez sur des chaînes dans d'autres tables, car elle n'est pas conçue pour le filtrage, et peut provoquer des comportements étranges.

13.2.6. Implémentation des chaînes utilisateur dans la table filtre

Maintenant que nous avons une bonne image de ce que nous voulons faire avec ce pare–feu, voyons l'implémentation de la table de règles. C'est le moment de faire attention à l'implémentation des règles et des chaînes que nous voulons créer, de même que les tables de règles dans les chaînes.

Après cela, nous créons les différentes chaînes spéciales que nous voulons utiliser avec la commande ***–N***. Les nouvelles chaînes sont créées et implémentées sans aucune règle à l'intérieur. Les chaînes que nous utilisons sont, comme précédemment décrit, icmp_packets, tcp_packets, udp_packets et les chaînes autorisées, qui sont utilisées par la chaîne tcp_packets. Les paquets entrants sur ***$INET_IFACE***, de type ICMP, seront redirigés vers la chaîne icmp_packets. Les paquets de type TCP, seront redirigés vers la chaîne tcp_packets et les paquets entrants de type UDP venant de ***$INET_IFACE*** iront vers la chaîne udp_packets. Tout ceci sera expliqué en détail dans la section Chaîne INPUT ci–dessous. Créer une chaîne est tout à fait simple et consiste seulement en une déclaration de chaîne comme ceci :

`iptables` [–N *`chain`*]

Dans les sections suivantes nous verrons les chaînes définies par l'utilisateur que nous avons créées. Regardons à quoi elles ressemblent, quelles règles elles contiennent et ce que nous pouvons faire avec.

13.2.6.1. La chaîne bad_tcp_packets

La chaîne bad_tcp_packets est destinée à contenir les règles qui vérifient les paquets entrants avec des en–têtes mal formés ou d'autres problèmes. Nous avons choisi d'inclure seulement un filtre de paquet qui bloque tous les paquets TCP entrants qui sont considérés comme ***NEW*** mais n'ont pas le bit SYN placé, et une règle qui bloque les paquets SYN/ACK considérés comme NEW. Cette chaîne peut être utilisée pour vérifier toutes les contradictions possibles, comme ci–dessus ou les balayages de port *XMAS*, etc. Nous pourrions de même ajouter des règles pour l'état ***INVALID***.

Si vous voulez pleinement comprendre le NEW non SYN, regardez la section Paquets état NEW sans bit SYN placé dans l'annexe Problèmes et questions courants en relation avec NEW et les paquets non–SYN. Ces paquets seront autorisés dans certaines circonstances mais dans 99% des cas nous n'en aurons pas besoin. Nous pouvons les journaliser et ensuite les supprimer.

La raison pour laquelle nous rejetons les paquets SYN/ACK qui sont considérés comme NEW est très simple. C'est décrit en détail dans la section SYN/ACK et les paquets NEW de l'annexe Problèmes et questions courants.

13.2.6.2. La chaîne autorisée

Si un paquet de type TCP arrive sur l'interface ***$INET_IFACE***, il traverse la chaîne tcp_packets et si la connexion est sur un port sur lequel nous voulons autoriser le trafic, nous ferons certaines vérifications finales sur ce port pour savoir s'il est actuellement autorisé ou non. Toutes ces vérifications finales sont faites dans la chaîne autorisée.

En premier, nous vérifions si le paquet est un paquet SYN. Si c'est le cas, il y a de fortes chances pour que ce soit le premier paquet d'une nouvelle connexion, nous l'autorisons. Ensuite nous vérifions si le paquet provient d'une connexion ***ESTABLISHED*** ou ***RELATED***, et si c'est encore le cas nous l'autorisons. Une connexion ***ESTABLISHED*** est une connexion qui a observé le trafic dans les deux sens, et donc nous avons un paquet

SYN, cette connexion doit être dans l'état ***ESTABLISHED***, selon la machine d'état. La dernière règle dans cette chaîne ***DROP*** tout le reste. Dans ce cas ceci indique que tout le trafic n'a pas été forcément observé dans les deux directions, i.e., nous n'avons pas répondu au paquet SYN, ou qu'il y a eu une tentative de connexion sans paquet SYN. Il n'y a *pas*, dans la pratique, de démarrage de connexion sans paquet SYN, sauf dans le cas où des personnes font du balayage de port. Actuellement, il n'y a pas d'implémentation TCP/IP qui supporte l'ouverture d'une connexion TCP avec autre chose qu'un paquet SYN à ma connaissance, donc nous faisons un ***DROP*** car nous sommes à 99% sûrs qu'il s'agit alors d'un balayage de port.

13.2.6.3. La chaîne TCP

La chaîne tcp_packets spécifie quels ports provenant de l'Internet sont autorisés dans le pare−feu. Il y a cependant, quelques vérifications supplémentaires à faire, ainsi nous envoyons chaque paquet vers la chaîne autorisée, comme décrit précédemment.

−A tcp_packets indique à ***iptables*** dans quelle chaîne ajouter la nouvelle règle, celle−ci étant ajoutée à la fin de la chaîne. ***−p TCP*** indique d'apparier les paquets TCP et ***−s 0/0*** apparie toutes les adresses source provenant de 0.0.0.0 avec un masque de réseau de 0.0.0.0, en d'autres termes *toutes* les adresses source. C'est le comportement par défaut mais je l'utilise ici pour rendre les choses le plus clair possible. ***−−dport 21*** indique le port de destination 21, si le paquet est destiné au port 21 il est aussi vérifié. Si tous les critères sont appariés, le paquet sera dirigé vers la chaîne autorisée. S'il n'apparie aucune des règles, elles seront renvoyées à la chaîne qui a expédié le paquet vers la chaîne tcp_packets.

Comme cela maintenant, il autorise le port TCP 21, ou le port de contrôle FTP, qui sert à contrôler les connexions FTP et plus tard les connexions ***RELATED***, ainsi nous autorisons les connexions PASSIVE et ACTIVE car le module ip_conntrack_ftp est chargé. Si nous ne voulons pas du tout autoriser le FTP, nous pouvons décharger le module ip_conntrack_ftp et supprimer la ligne ***$IPTABLES −A tcp_packets −p TCP −s 0/0 −−dport 21 −j allowed*** du fichier `rc.firewall.txt`.

Le port 22 est le port SSH, qu'il est beaucoup mieux d'utiliser que de permettre le telnet sur le port 23 si vous voulez autoriser quelqu'un de l'extérieur à utiliser un shell sur votre machine. Notez que c'est toujours une mauvaise idée de permettre à quelqu'un d'autre que vous même d'avoir accès à une machine pare−feu.

Le port 80 est le port HTTP, en d'autres termes votre serveur web, supprimez le si vous ne voulez pas exécuter un serveur web directement sur votre pare−feu.

Enfin, nous autorisons le port 113, qui est le IDENTD et peut être nécessaire pour certains protocoles comme IRC, etc. Notez qu'il peut être intéressant d'utiliser le paquetage ***oidentd*** si vous faites du NAT sur plusieurs hôtes de votre réseau local. ***oidentd*** possède un support pour faire du relaying des requêtes IDENTD vers les bonnes machines de votre réseau local.

Si vous voulez ajouter d'autres ports dans ce script, c'est le moment. Simplement, copiez et coller une des autres lignes de la chaîne tcp_packets et modifiez la en fonction des ports que vous voulez ouvrir.

13.2.6.4. La chaîne UDP

Si nous obtenons un paquet UDP dans la chaîne INPUT, nous l'envoyons alors vers udp_packets où il sera de nouveau apparié pour le protocole UDP avec ***−p UDP*** et ensuite vérifié avec l'adresse source 0.0.0.0 et le masque de réseau 0.0.0.0. Sauf que cette fois, nous n'acceptons que les ports UDP spécifiques que nous voulons ouvrir pour les hôtes de l'Internet. Notez que nous ne créons pas de trous sur le port source des hôtes expéditeurs, car il en sera pris soin par la machine d'état. Nous n'avons besoin d'ouvrir des ports sur notre hôte que si nous devons faire tourner un serveur sur un port UDP, comme le DNS, etc. Les paquets entrants dans le pare−feu et qui font partie d'une connexion déjà établie (par notre réseau local) seront automatiquement acceptés par les règles −−state ESTABLISHED,RELATED au début de la chaîne INPUT.

Ainsi, nous ne plaçons pas le ***ACCEPT*** sur les paquets UDP entrants provenant du port 53, celui qui est utilisé pour le DNS. La règle existe mais elle est commentée par défaut. Si vous voulez que votre pare−feu agisse comme serveur DNS, décommentez la.

j'ai personnellement autorisé le port 123, port NTP ou network Time Protocol. Ce protocole est utilisé pour synchroniser l'horloge de votre machine avec des serveurs de temps qui sont *très* précis. La plupart d'entre vous n'utilise sans doute pas ce protocole et je ne l'ai donc pas autorisé par défaut. Il suffit aussi de décommenter la règle pour l'activer.

Nous n'autorisons pas le port 2074, utilisé par certains programmes *multimedia* comme ***speak freely*** qui servent à parler avec d'autres personnes en temps réel en utilisant des haut−parleurs et des microphones, ou même un casque d'écoute. Si vous voulez vous en servir décommentez simplement la ligne.

Le port 4000 est celui du protocole ICQ. C'est un protocole très bien connu qui est utilisé par le programme Mirabilis nommé ***ICQ***. Il existe au moins 2 ou 3 clones de ***ICQ*** pour Linux et c'est un des programmes de chat les plus utilisés dans le monde. Je doute qu'il soit besoin d'en expliquer d'avantage.

À ce point, deux règles supplémentaires sont disponibles si vous avez fait l'expérience de certaines entrées de journaux dans certaines circonstances. La première règle bloque la diffusion des paquets vers les ports de destination 135 à 139. Ils sont utilisés par NETBIOS, ou SMB pour les utilisateurs de Microsoft. Ceci bloque toutes les entrées de journaux provenant de iptables qui journalise l'activité de réseaux Microsoft à l'extérieur de notre pare−feu. La seconde règle a été créée pour prévenir les problèmes de journalisation excessive, et prend soin des requêtes DHCP provenant de l'extérieur. Ceci est particulièrement vrai si votre réseau extérieur est de type Ethernet non−commuté, dans lequel les clients obtiennent leur adresses IP par DHCP. Dans ces circonstances vous pouvez avoir beaucoup d'entrées de journal juste pour ça.

Note

Notez que ces deux dernières règles sont désactivées car certaines personnes peuvent être intéressées par ce genre de logs. Si vous rencontrez des problèmes avec une journalisation excessive, essayez de supprimer ce type de paquetages à ce niveau. Il y a aussi beaucoup de règles de ce type juste avant les règles de log dans la chaîne INPUT.

13.2.6.5. La chaîne ICMP

C'est là que nous décidons quels types ICMP autoriser. Si un paquet de type ICMP arrive sur eth0 dans la chaîne INPUT, nous le redirigeons vers la chaîne `icmp_packets` comme expliqué plus haut. Ici nous consignons quels types ICMP autoriser. Pour le moment, j'autorise seulement les requêtes écho ICMP entrantes, TTL égale 0 pendant le transit et TTL égale 0 pendant le réassemblage. La raison pour laquelle nous n'autorisons aucun autre type ICMP par défaut, est que la plupart des autres types ICMP seront pris en charge par les règles d'état RELATED.

Note

Si un paquet ICMP est envoyé en réponse à un paquet déjà existant il est considéré comme RELATED par rapport au flux d'origine. Pour plus d'information sur les états, voir le chapitre La machine d'état.

La raison pour laquelle j'autorise ces paquets ICMP est la suivante, les Requêtes Écho servent aux réponses écho, utilisées principalement pour "pinguer" d'autres hôtes, pour voir s'ils sont disponibles sur les réseaux. Sans cette règle, d'autres hôtes ne pourraient pas nous "pinguer" pour vérifier que nous sommes présent dans une connexion réseau. Notez que certaines personnes ont tendance à supprimer cette règle, car ils ne veulent pas être vus sur Internet. Supprimer cette règle rend effectivement inefficace tous les pings vers notre pare−feu depuis l'Internet car le pare−feu ne répondra tout simplement pas.

Time Exceeded (i.e., TTL égale 0 pendant le transit et TTL égale 0 pendant le réassemblage), est autorisé dans le cas où nous voulons faire du traçage de route sur certains hôtes ou si un paquet a un TTL pacé à 0, nous obtiendrons une réponse en retour. Par exemple, quand vous faites un traceroute sur quelqu'un, vous

commencez avec un TTL = 1, et il obtient en retour un 0 au premier saut de son chemin, et un Time Exceeded est envoyé depuis la première passerelle de la route vers l'hôte que vous voulez tracer, ensuite le TTL = 2 et la seconde passerelle envoie un Time Exceeded, et ainsi de suite jusqu'à ce que vous obteniez une réponse de l'hôte que vous vouliez joindre. De cette façon nous obtenons une réponse de chaque hôte sur notre chemin, et pouvons voir quel hôte ne répond pas.

Pour une liste complète de tous les types ICMP, voir l'appendice Types ICMP. Pour plus d'information sur ICMP lisez les documents et rapports :

- RFC 792 – Internet Control Message Protocol par J. Postel.

Note

Une erreur peut apparaître chez vous quand vous bloquez certains types ICMP, mais dans mon cas tout fonctionne parfaitement quand je bloque tous les types ICMP non autorisés.

13.2.7. Chaîne INPUT

La chaîne INPUT, utilise la plupart du temps les autres chaînes pour faire le plus gros du travail. De cette façon nous n'avons pas trop de charge provenant d'iptables, et il fonctionnera mieux sur les machines lentes. Ceci est fait en vérifiant les détails spécifiques qui peuvent être identiques pour beaucoup de paquets différents, en ensuite en envoyant ces paquets dans les chaînes spécifiées par l'utilisateur. En faisant ça, nous réduisons notre table de règles qui ne contient que le nécessaire pour le transit des paquets, et donc le pare–feu aura moins de charge pour filtrer les paquets.

En premier nous vérifions les mauvais paquets. C'est réalisé en envoyant tous les paquets TCP vers la chaîne bad_packets. Cette chaîne contient des règles qui vérifient les paquets mal formés ou d'autres anomalies. Pour une explication complète sur la chaîne bad_tcp_packets, regardez dans la section La chaîne bad_tcp_packets de ce chapitre.

À ce niveau nous recherchons le trafic généré par les réseaux de confiance. Ce qui inclut l'adaptateur réseau et tout le trafic prevenant de celui–ci, ainsi que le trafic entrant et sortant de la boucle locale (loopback), avec toutes les adresses IP assignées (toutes les adresses y compris notre adresse IP Internet). Ainsi, nous avons choisi de placer la règle qui autorise l'activité du LAN vers le pare–feu en premier, car notre réseau local génère plus de trafic de la connexion Internet. Ceci permet d'avoir moins de charge système pour apparier chaque paquet avec chaque règle, et c'est toujours une bonne idée de regarder quel type de trafic traverse principalement le pare–feu. En faisant cela nous rendons les règles plus efficaces, avec moins de charge sur le pare–feu et moins de congestion sur notre réseau.

Avant de nous attaquer aux règles "réelles" dans lesquelles nous déciderons quoi autoriser depuis l'Internet, nous avons placé une règle pour réduire la charge système. C'est une règle d'état qui autorise tous les paquets d'un flux ESTABLISHED ou RELATED vers l'adresse IP Internet. Cette règle a une équivalence dans la chaîne autorisée, qui est redondante à celle–ci. Cependant, la règle ***––state ESTABLISHED,RELATED*** dans la chaîne autorisée a été conservée pour plusieurs raisons, vous pouvez donc copier–coller cette fonction.

Après ça, nous apparions tous les paquets TCP de la chaîne INPUT qui arrivent dans l'interface ***$INET_IFACE***, et les envoyons vers `tcp_packets`, comme précédemment décrit. Nous faisons maintenant la même chose pour les paquets UDP sur l'interface ***$INET_IFACE*** et les envoyons vers la chaîne udp_packets, ensuite tous les paquets ICMP sont envoyés vers la chaîne icmp_packets. Normalement, un pare–feu devrait être plus difficile à attaquer par des paquets TCP, que par des paquets UDP et ICMP. C'est le cas normal, mais souvenez vous, ce peut être différent pour vous. La même chose peut être observée ici, comme avec les règles réseau spécifiques. Lesquelles génèrent le plus de trafic ? Sur les réseaux générant un important volume de données, c'est une absolue nécessité de vérifier cela, car une machine de type Pentium III peut être saturée par une simple table de règles contenant 100 règles avec une carte réseau ethernet 100 Mbit fonctionnant à sa pleine capacité, si la table de règles est mal écrite. Il est important de regarder ça de près.

Ici nous avons une règle supplémentaire, qui est par défaut désactivée, et qui peut être utilisée pour évitez une journalisation excessive dans le cas où nous avons un réseau Microsoft à l'extérieur de notre pare−feu Linux. Les clients Microsoft ont la mauvaise habitude d'envoyer des tonnes de packets multicast vers la plage 224.0.0.0/8, donc nous avons la possibilité de bloquer ces paquets ici. Il existe deux autres règles faisant à peu près la même chose sur la chaîne udp_packets décrite dans La chaîne UDP.

Avant de tester la stratégie par défaut de la chaîne INPUT, nous la journalisons pour savoir s'il existe des problèmes/bugs. Ce peut être soit un paquet que nous ne voulons pas autoriser, soit une chose qui peut se révéler néfaste pour nous, ou finalement un problème dans notre pare−feu qui n'autorise pas le trafic qui devrait être autorisé. Nous ne journalisons pas plus de 3 paquets par minute car nous ne voulons pas surcharger nos journaux, ainsi nous plaçons un préfixe pour toutes les entrées de journalisation et savons donc d'où ils proviennent.

Tout ce qui n'a pas été capturé sera ***DROP***é par la stratégie par défaut de la chaîne INPUT. Voir la section Mise en place des actions par défaut dans ce chapitre.

13.2.8. Chaîne FORWARD

La chaîne FORWARD contient quelques règles dans notre scenario. Nous avons une seule règle qui envoie tous les paquets vers la chaîne bad_tcp_packets, laquelle est également utilisée dans la chaîne INPUT comme décrit précédemment. La chaîne bad_tcp_packets est construite de façon qu'elle puisse être utilisée dans plusieurs chaînes, sans regarder quel paquet la traverse.

Après cette vérification des mauvais paquets TCP, nous avons les règles principales dans la chaîne FORWARD. La première règle autorise tout le trafic depuis notre ***$LAN_IFACE*** vers n'importe quelle autre interface librement, sans restrictions. En d'autres termes, cette règle autorise tout le trafic depuis le LAN vers l'Internet. La seconde règle autorise le trafic en retour ***ESTABLISHED*** et ***RELATED*** à travers le pare−feu. Ce qui veut dire qu'elle autorise les connexions initiées par notre réseau local à circuler librement dans le LAN. Ces règles sont nécessaires pour que notre réseau local puisse accéder à l'Internet, car la stratégie par défaut de la chaîne FORWARD est placée à ***DROP***. C'est adroit, car elle autorise les hôtes de notre réseau local à se connecter à des hôtes sur Internet, mais en même temps elle bloque les hôtes depuis Internet leur interdisant de se connecter aux hôtes de notre réseau interne.

Enfin, nous avons également une chaîne de journalisation pour tous les paquets qui ne sont pas autorisés dans un sens ou dans l'autre à traverser la chaîne FORWARD. Ceci concerne principalement les paquets mal formés ou autre problème. Une cause peut être une attaque de hacker, et une autre des paquets mal formés. C'est exactement la même règle que celle utilisée dans la chaîne INPUT sauf pour le préfixe de journalisation, ***"IPT FORWARD packet died: "***. Le préfixe de journalisation est principalement utilisé pour séparer les entrées de journaux, et peut être utilisé pour savoir d'où les paquets ont été journalisés et connaître certaines options d'en−tête.

13.2.9. Chaîne OUTPUT

Comme nous utilisons notre machine en partie comme pare−feu et en partie comme station de travail, nous autorisons tout ce qui sort de cette machine qui a une adresse source ***$LOCALHOST_IP***, ***$LAN_IP*** ou ***$STATIC_IP***. Enfin nous journalisons tout ce qui est DROPé. S'il y a des paquets DROPés, nous voulons savoir quelle action entreprendre contre ce problème. Soit c'est une erreur, soit c'est un paquet mystérieux qui peut être usurpé. Enfin nous ***DROP***ons le paquet dans la stratégie par défaut.

13.2.10. Chaîne PREROUTING de la table nat

La chaîne PREROUTING fait à peu près ce qu'elle indique, elle traduit les adresses réseau sur les paquets avant la décision de routage qui les envoie vers les chaînes INPUT ou FORWARD dans la table de filtrage. La seule raison que nous avons de parler de cette chaîne ici est que nous ne faisons aucun filtrage dans

celle−ci. La chaîne PREROUTING est traversée seulement par le premier paquet d'un flux, ce qui veut dire que tous les autres paquets ne seront pas vérifiés dans cette chaîne. Dans ce script, nous n'utilisons pas du tout la chaîne PREROUTING, cependant, c'est le bon endroit si nous voulons faire du DNAT sur des paquets spécifiques, par exemple si nous voulons héberger notre serveur web dans notre réseau local. Pour plus d'information sur la chaîne PREROUTING, lire le chapitre Traversée des tables et des chaînes.

Attention

La chaîne PREROUTING ne doit pas être utilisée pour quelque filtrage que ce soit, car parmi d'autres choses, elle n'est traversée que par le premier paquet d'un flux. Elle devrait être utilisée uniquement pour la traduction d'adresse réseau, à moins que vous ne sachiez réellement ce que vous faites.

13.2.11. Démarrage de SNAT et la chaîne POSTROUTING

Notre dernière mission est d'activer la traduction d'adresse réseau. En premier nous ajoutons une règle à la table nat, dans la chaîne POSTROUTING qui NAT tous les paquets provenant de notre interface et allant vers Internet. Pour moi c'est eth0. Cependant, il existe des variables spécifiques ajoutées aux scripts d'exemples qui peuvent être utilisées automatiquement pour configurer cela. L'option ***−t*** indique à ***iptables*** dans quelle table insérer la règle, dans notre cas c'est la table nat. La commande −A indique que nous voulons lier une nouvelle règle à une chaîne existante nommée POSTROUTING et ***−o $INET_IFACE*** nous dit d'apparier tous les paquets sortants sur l'interface ***INET_IFACE*** (ou eth0, par défaut dans ce script) et enfin nous plaçons la cible pour faire du ***SNAT*** sur les paquets. Ainsi tous les paquets qui apparient cette règle seront SNATés pour vérifier qu'ils viennent de l'interface Internet. Notez que vous devez indiquer l'adresse IP à donner aux paquets sortants avec l'option ***−−to−source*** envoyée à la cible SNAT.

Dans ce script nous avons choisi d'utiliser la cible ***SNAT*** au lieu de ***MASQUERADE*** pour deux raisons. La première est que ce script est supposé s'exécuter sur un pare−feu qui possède une adresse IP statique. La raison suivante est qu'il est plus rapide et plus efficace d'utiliser la cible SNAT si possible. Bien sûr, nous l'utilisons aussi pour montrer comment elle fonctionne dans un exemple réel. Si nous n'avons pas d'adresse IP statique, nous utiliserons la cible ***MASQUERADE*** car elle offre des fonctions simples et faciles pour faire du NAT, mais elle récupère automatiquement l'adresse IP qui sera utilisée. Ceci consomme un peu plus de temps système, mais c'est très avantageux si vous utilisez DHCP. Si vous voulez avoir une vue plus détaillée de la cible ***MASQUERADE***, regardez le script rc.DHCP.firewall.txt.

Chapitre 14. Exemples de scripts

L'objectif de ce chapitre est de vous fournir une brève explication de chaque script disponible avec ce didacticiel, et quels services ils fournissent. Ces scripts ne sont en aucun cas parfaits, et peuvent ne pas correspondre tout à fait à ce que vous en attendez. C'est une aide pour vous assister dans la création de scripts selon vos besoins. La première section indique la structure que j'ai établie dans chaque script, ansi nous pourrons retrouver notre chemin plus facilement.

14.1. Structure du script rc.firewall.txt

Tous les scripts de ce didacticiel ont été écrits pour une structure spécifique. La raison pour ça est qu'ils sont assez similaires entre eux ce qui permet de façon aisée de voir les différences. Cette structure est à peu près bien documentée dans ce bref chapitre. Il vous donnera un idée de pourquoi ces scripts ont été écrits, et pourquoi j'ai choisi cette structure.

Note

Même si c'est la structure que j'ai choisi, notez qu'elle peut ne pas être la meilleure pour vos scripts. Elle vise une lecture et une compréhension faciles pour nos besoins.

14.1.1. La structure

C'est la structure de tous les scripts de ce didacticiel. S'ils diffèrent quelque part c'est probablement une erreur de ma part, sauf si spécifié explicitement.

1. *Configuration* – En premier lieu nous avons les options de configuration que le reste du script utilisera. Les options de configuration seront toujours les premières dans chaque script.
    1. *Internet* – C'est la section de configuration qui concerne la connexion Internet. Vous pouvez la passer si vous n'avez pas de connexion Internet. Notez qu'il pourrait y avoir d'avantage de sous–sections ici, mais nous n'indiquons que celles concernant l'Internet.
        1. *DHCP* – Si nécessaire nous ajouterons les options de configuration spécifique à DHCP ici.
        2. *PPPoE* – Si l'utilisateur désire ce script spécifique, et qu'il utilise une connexion PPPoE, nous ajouterons les options ici.
    2. *LAN* – S'il y a un réseau local derrière le pare–feu, nous ajouterons les options le concernant ici. C'est le cas la plupart du temps, donc cette section sera toujours disponible.
    3. *DMZ* – Si nécessaire, nous ajouterons la configuration de la DMZ ici. Beaucoup de scripts n'ont pas cette section, principalement parce que dans un réseau domestique, ou pour une petite entreprise il n'y en a pas.
    4. *Localhost* – Cette section concerne l'hôte local. Ces options ne changent pratiquement jamais.
    5. *iptables* – Section qui concerne la configuration spécifique d'iptables. Dans la plupart des cas elle ne nécessite qu'une variable qui nous indique où iptables est situé.
    6. *Other* – S'il y a d'autres options et variables spécifiques, elles devront être placées dans la sous–section concernée (si elles appartiennent à la connexion Internet, elles seront placées dans la sous–section Internet, etc.). Si elles ne vont nulle part elles seront placées dans les sous–sections des options de configuration.
2. *Module loading* – Cette section contient une liste de modules. La première partie concerne les modules nécessaires, la seconde les modules non nécessaires.

    
    Note

    Notez que certains modules peuvent accroître la sécurité, ou ajouter certaines possibilités, et donc peuvent être ajoutés même s'ils ne sont pas obligatoires. Ils seront indiqués dans certains cas dans les scripts.
    1. *Required modules* – Cette section contient les modules obligatoires et, peut être, des modules spéciaux qui ajoutent à la sécurité ou des services supplémentaires pour l'administrateur ou les clients.
    2. *Non–required modules* – Section qui contient les modules non obligatoires pour les opérations normales. Tous ces modules peuvent être commentés par défaut, si vous voulez ajouter le service en question décommentez le.
3. *proc configuration* – Cette section concerne toute configuration particulière nécessaire pour le système de fichiers proc. Si certaines de ces options sont obligatoires, elles seront listées ici, elles sont commentées par défaut, et indiquées comme configurations proc non obligatoires. Beaucoup de configurations proc utiles seront indiquées, mais pas toutes et de loin.
    1. *Required proc configuration* – Section qui contient les configurations proc obligatoires pour que le script fonctionne. Elle peut aussi contenir des configurations qui accroissent la sécurité, ou ajoutent des services supplémentaires pour l'administrateur ou les clients.
    2. *Non–required proc configuration* – Cette section pourrait contenir les configurations proc non obligatoires mais qui peuvent être utiles. Elles sont toutes commentées, car elles ne sont pas nécessaires pour l'instant pour que le script fonctionne. Cette liste n'est de loin pas complète.

rules set up – Maintenant le script est prêt pour y insérer la table de règles. J'ai choisi de diviser toutes les règles en noms de table et de chaîne dans la table de règles, pour rendre plus facile à lire ce qui suit. Toutes les chaînes utilisateur spécifiées sont créées avant de faire quoi que ce soit d'autre. J'ai aussi choisi de placer les chaînes et leur spécifications de règles dans le même ordre que la sortie de la commande ***iptables –L***.

1. *Filter table* – En premier nous voyons la table filter et son contenu. En priorité nous configurons toutes les stratégies de la table.

    1. *Set policies* – Configuration des stratégies par défaut pour les chaînes système. Normalement je met les stratégies à DROP pour les chaînes de la table filtre, et spécifie ACCEPT les services et les flux que je veux autoriser. De cette façon nous nous débarrassons de tous les ports que nous ne voulons pas autoriser.
    2. *Create user specified chains* – Ici nous créons toutes les chaînes utilisateur que nous voulons utiliser dans cette table. Nous ne pourrons pas utiliser ces chaînes dans les chaînes système si elles ne sont pas déjà créées, le mieux est de le faire le plus tôt possible.
    3. *Create content in user specified chains* – Après avoir créé les chaînes utilisateur nous pouvons rentrer toutes les règles dans ces chaînes. Vous pouvez aussi les rentrer plus tard dans le script, c'est comme vous voulez.
    4. *INPUT chain* – Ici nous ajouterons toutes les règles de la chaîne INPUT.

        
        Note

        Nous utiliserons le modèle de sortie de la commande ***iptables –L*** comme vous pourrez le voir. Il n'y a pas de raison pour que vous conserviez cette structure, cependant, essayez d'éviter de mélanger les données provenant de différentes tables et chaînes car elles deviendraient plus difficiles à lire et à résoudre les problèmes.
    5. *FORWARD chain* – Ici nous ajoutons les règles de la chaîne FORWARD.
    6. *OUTPUT chain* – En dernier, nous ajoutons les règles de la chaîne OUTPUT.
2. *nat table* – Après la table filtre occupons nous de la table nat. Nous le faisons après la table filtre pour plusieurs raisons. La première c'est que nous ne voulons pas activer l'ensemble du mécanisme de forwarding et les fonctions NAT trop tôt, ce qui pourrait conduire les paquets à traverser le pare–feu au mauvais moment (i.e., quand le NAT est activé, mais que les règles de filtre ne le sont pas). Ainsi, je vois la table nat comme une sorte de couche qui se lie à la table filter et en quelque sorte l'entoure. La table filter sera donc le noyau, tandis que la table nat agira comme une couche autour de la table filter, et enfin la table mangle entourera la table nat comme une seconde couche. Ceci peut être faux dans certaines perspectives mais pas trop loin de la réalité.

    1. *Set policies* – En premier nous plaçons les stratégies par défaut dans la table nat. Normalement, avec la stratégie ACCEPT placée au début ce sera suffisant. Cette table n'est pas utilisée pour le filtrage, et les paquets ne seront pas DROP ici, car certaines choses dangereuses peuvent survenir dans certains cas. Je laisse ces chaînes à ACCEPT car il y a aucune raison de ne pas le faire.
    2. *Create user specified chains* – Ici nous créons les chaînes utilisateur que nous voulons insérer dans la table nat. Normalement je n'en ai pas, mais j'ai ajouté cette section juste au cas où. Notez que les chaînes utilisateur doivent être créées avant qu'elles soient utilisées dans les chaînes système.
    3. *Create content in user specified chains* – Maintenant il est temps d'ajouter toutes les règles des chaînes utilisateur dans la table nat. C'est la même chose que pour les chaînes utilisateur dans la table filter. Nous les ajoutons ici car il n'y a aucune raison de ne pas le faire.

4. 4. *PREROUTING chain* – La chaîne PREROUTING est utilisée pour faire du DNAT sur les paquets quand nous en avons besoin. Dans beaucoup de scripts cette fonctionnalité n'est pas utilisée, ou alors elle est désactivée. La raison en étant que nous ne voulons pas créer de gros trous dans notre réseau local sans savoir ce que nous faisons. Dans certains scripts nous l'avons activé par défaut car le seul but de ces scripts et de procurer certains services.
   5. *POSTROUTING chain* – La chaîne POSTROUTING sera utilisée par les scripts que j'ai écrit car la plupart d'entre eux dépendent du fait que nous avons un ou plusieurs réseaux locaux que nous voulons protéger de l'Internet. Principalement nous essaierons d'utiliser la cible SNAT, mais dans certains cas nous devrons utiliser la cible MASQUERADE.
   6. *OUTPUT chain* – Cette chaîne est à peine utilisée dans les scripts. Je n'ai trouvé aucune bonne raison de m'en servir.
3. *mangle table* – La dernière table est la table mangle. Normalement je n'utilise pas cette table, sauf pour des besoins spécifiques, comme masquer toutes les machines pour utiliser le même TTL ou pour changer les champs TOS, etc. J'ai choisi de laisser ces parties du script plus ou moins vides, avec quelques exceptions dans lesquelles j'ai ajouté des exemples.

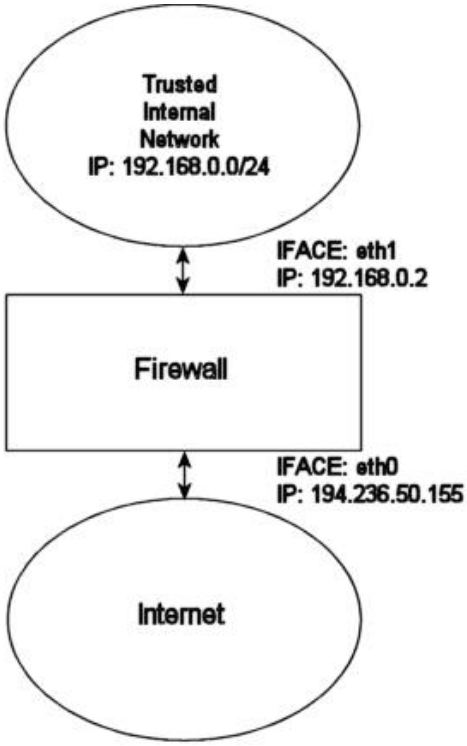
   1. *Set policies* – Place les stratégies par défaut dans la chaîne. C'est la même chose que pour la table nat, à peu près. Cette table n'est pas faite pour le filtrage. Je n'ai placé aucune stratégie dans aucun des scripts de la table mangle, et vous êtes encouragés à en faire autant.
   2. *Create user specified chains* – Crée toutes les chaînes utilisateur. Comme j'ai laissé vide la table mangle, je n'ai créé aucune chaîne ici. Cependant, cette section a été ajoutée juste au cas où vous en auriez besoin dans le futur.
   3. *Create content in user specified chains* – Ici plus aucun script de ce didacticiel ne contiendra des règles.
   4. *PREROUTING* – Ici plus aucun script de ce didacticiel ne contiendra des règles.
   5. *INPUT chain* – Ici plus aucun script de ce didacticiel ne contiendra des règles.
   6. *FORWARD chain* – Ici plus aucun script de ce didacticiel ne contiendra des règles.
   7. *OUTPUT chain* – Ici plus aucun script de ce didacticiel ne contiendra des règles.
   8. *POSTROUTING chain* – Ici plus aucun script de ce didacticiel ne contiendra des règles.

Nous expliquerons en détail comment chaque script est structuré et pourquoi.

Attention

Notez que ces descriptions sont assez brèves, et doivent être vues comme une rapide explication.

14.2. rc.firewall.txt

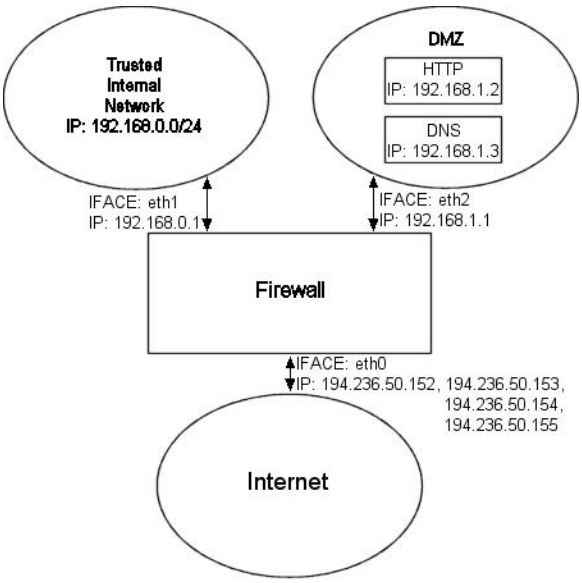


Le rc.firewall.txt est le noyau sur lequel le reste des scripts est basé. Le chapitre Fichier rc.firewall expliquera chaque détail du script. Il a été écrit principalement pour un réseau domestique dual. Par exemple, vous avez un LAN et une connexion Internet. Ce script suppose également que vous avez une IP fixe vers l'Internet, et donc que vous n'utilisez pas DHCP, PPP ou SLIP ou un autre protocole qui assigne les IP automatiquement. Si vous cherchez un script pour cela, regardez de plus près rc.DHCP.firewall.txt.

Le script `rc.firewall.txt` nécessite que les options suivantes soient compilées statiquement dans le noyau, ou comme modules. Sans cela des parties du script seront inutilisables. Vous pourrez avoir besoin de d'avantage d'options, tout dépend de ce que vous voulez utiliser.

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

14.3. rc.DMZ.firewall.txt



Le script rc.DMZ.firewall.txt a été écrit pour les personnes qui ont un réseau de confiance, une DMZ et une connexion Internet. La DMZ est dans ce cas NATée pair–à–pair et nécessite de faire de l'alias d'IP dans le pare–feu, i.e., la machine doit reconnaître les paquets de plus d'une IP. Il existe plusieurs moyens de faire cela, un de ceux–ci est de placer le NAT pair–à–pair, un autre est de créer un sous–réseau, donnant au pare–feu une IP interne et une externe. Vous pouvez alors placer ces IP sur la machine DMZ comme vous le voulez. Notez que ça vous "occupera" deux adresses, une pour l'adresse de diffusion et l'autre pour l'adresse réseau. C'est à vous de décider de l'implémenter ou non. Ce didacticiel vous donne les outils pour créer un pare–feu et faire du NAT, mais ne vous dira pas exactement tout en fonction de vos besoins spécifiques.

Le script rc.DMZ.firewall.txt nécessite que ces options soient compilées dans votre noyau, soit de façon statique soit comme modules. Sans ces options vous ne pourrez pas utiliser les fonctionnalités de ce script. Vous obtiendriez des erreurs sur les modules et les cibles/sauts ou les correspondances. Si vous envisagez de faire du contrôle de trafic ou quelque chose comme ça, vous devez vérifier que toutes les options obligatoires sont compilées dans votre noyau.

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

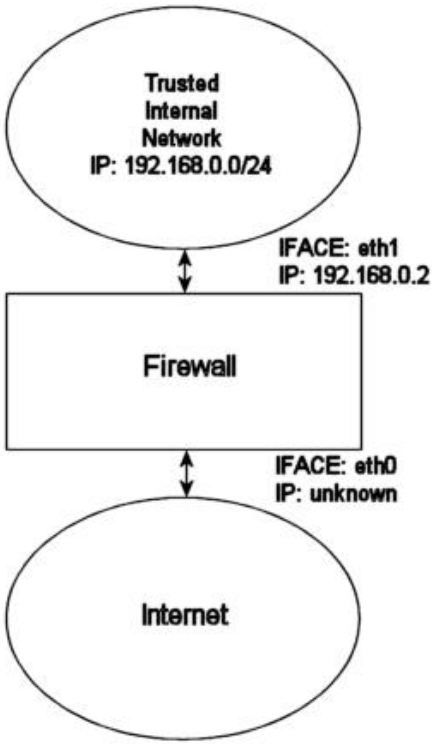
Vous devez avoir deux réseaux internes pour ce script comme vous pouvez le voir sur l'image. L'un utilise la plage IP 192.168.0.0/24 et correspond au réseau de confiance. L'autre utilise la plage IP 192.168.1.0/24 et est la DMZ dans laquelle nous faisons du NAT pair–à–pair. Par exemple, si quelqu'un sur l'Internet envoit un paquet vers notre DNS_IP, nous utilisons DNAT pour expédier ce paquet vers notre DNS sur le réseau DMZ. Quand le DNS voit le paquet, il sera destiné à l'adresse IP du réseau interne DNS, et pas vers l'IP DNS externe. Si le paquet n'était pas traduit, le DNS ne répondrait pas à ce paquet. Voyons à quoi ressemble le code DNAT :

```
$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $DNS_IP \
--dport 53 -j DNAT --to-destination $DMZ_DNS_IP
```

En premier, DNAT ne peut être exécuté que dans la chaîne PREROUTING de la table nat. Le protocole TCP sur $INET_IFACE avec une destination IP qui apparie $DNS_IP, est dirigé vers le port 53, qui est le port TCP pour la zone de transferts entre serveurs de noms. Ensuite nous spécifions où nous voulons envoyer le paquet avec l'option ***--to-destination*** et lui donnons la valeur de la `$DMZ_DNS_IP`, en d'autres termes l'IP de notre réseau DNS ou DMZ. C'est du DNAT de base. Après ça la réponse au paquet DNATé est envoyée au pare-feu, qui le "déNATe" automatiquement.

Nous devrions en avoir suffisamment compris pour pouvoir saisir l'ensemble de ces scripts. S'il y a quelque chose que vous ne comprenez pas dans ce didacticiel, faites moi un mail c'est sans doute une erreur de ma part.

14.4. rc.DHCP.firewall.txt



Le script rc.DHCP.firewall.txt est à peu près identique au rc.firewall.txt. Cependant, il n'utilise pas la variable ***STATIC_IP***, ce qui est la principale différence avec le script rc.firewall.txt. La raison est qu'il ne fonctionne pas avec une connexion IP dynamique. Les modifications à effectuer sur le script d'origine sont minimes, cependant, certaines personnes m'ont demandé si ce script est une bonne solution. Il permet d'utiliser des connexions DHCP, PPP et SLIP pour l'Internet.

Le script `rc.DHCP.firewall.txt` nécessite que les options suivantes soient compilées statiquement dans le noyau, ou comme modules, pour fonctionner correctement.

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_MASQUERADE
- CONFIG_IP_NF_TARGET_LOG

le principal changement dans ce script consiste en la suppression de la variable `STATIC_IP` et à supprimer toute référence à cette variable. Le script filtrera maintenant sur la variable `INET_IFACE`. En d'autres termes ***–d $STATIC_IP*** a été changé en ***–i$INET_IFACE***. C'est la seule modification qu'il est réellement nécessaire de faire.

Il y a plusieurs choses à penser. Nous ne pouvons pas faire de filtrage sur ce qui dépend de la chaîne INPUT, par exemple, ***––in–interface $LAN_IFACE ––dst $INET_IP***. Ceci nous force à faire du filtrage uniquement sur les interfaces dans le cas où les machines internes doivent accéder à une IP Internet adressable. Un bon exemple est si nous faisont tourner un serveur HTTP sur notre pare–feu. Si nous allons sur la page principale (i.e., http://192.168.0.1/), qui contient des liens statiques vers le même hôte (i.e., http://foobar.dyndns.net/fuubar.html), qui pourrait être une solution dyndns, nous rencontrons un problème. La machine NATée cherchera le DNS pour l'IP du serveur HTTP, et ensuite tentera d'accéder à cette IP. Dans le cas où nous filtrons sur l'interface et l'IP, la machine NATée sera incapable d'accéder au HTTP car la chaîne INPUT ***DROP*** les paquets. Ceci s'applique aussi dans le cas où nous avons une IP statique, mais dans ces cas nous pouvons contourner le problème en ajoutant des règles qui apparient les paquets de l'interface LAN pour notre `INET_IP`, et les plaçons à ***ACCEPT***.

Comme vous l'avez vu plus haut, ce peut être une bonne idée de faire un script qui traite les IP dynamiques d'une meilleure façon. Nous pouvons par exemple faire un script qui récupère l'IP depuis ***ifconfig*** et l'ajoute à une variable, dans l'initialisation de la connexion Internet. Un bon moyen pour faire ça, serait d'utiliser, par exemple, les scripts `ip-up` fournis par ***pppd*** ou tout autre programme. Voir sur le site linuxguruz.org qui possède une quantité de scripts disponibles en téléchargement. Le lien est dans l'annexe Autres ressources et liens.

Note

Ce script peut être un peu moins sûr que le `rc.firewall.txt`. Je vous préviens qu'il est d'avantage ouvert aux attaques depuis l'extérieur.

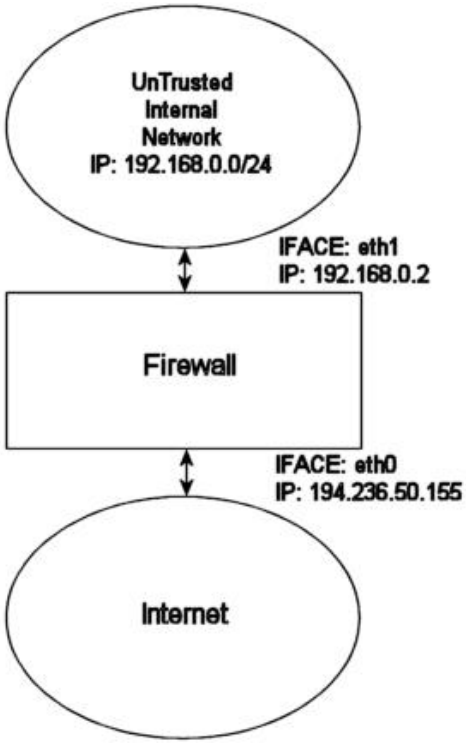
Il est également possible d'ajouter certaines choses comme cela dans votre script :

```
INET_IP=`ifconfig $INET_IFACE | grep inet | cut -d : -f 2 | \
cut -d ' ' -f 1`
```

La commande ci–dessus récupère automatiquement l'adresse IP de la variable ***$INET_IFACE***, affiche la ligne qui contient l'adresse IP et la transforme en une adresse IP gérable. Pour une façon plus élaborée de faire ceci, vous pouvez appliquer des bouts de code disponibles dans le script retreiveip.txt, qui récupère automatiquement votre adresse IP Internet quand vous lancez le script. Notez que cette façon de faire peut conduire à un comportement un peu aléatoire, comme le blocage des connexions depuis votre pare–feu en interne. Les comportements étranges les plus courants sont décrits dans la liste suivante.

1. Si le script est lancé depuis un script exécuté par, par exemple, le démon PPP, il suspendra toutes les connexions actives à cause des règles NEW non–SYN (voir la section Paquets état NEW sans bit SYN placé).
2. Si vous avez des règles statiques, il est plus difficile d'ajouter et d'enlever ces règles tout le temps, sans modifier celles déjà existantes. Par exemple, si vous voulez bloquer l'accès des hôtes de votre LAN au pare–feu, mais en même temps exécuter un script depuis le démon PPP, comment ferez vous sans effacer vos règles actives qui bloquent le LAN ?
3. Ce n'est pas nécessairement compliqué, mais peut conduire à des compromis sur la sécurité. Si le script est très simple, il est facile de corriger les problèmes.

14.5. rc.UTIN.firewall.txt



Le script rc.UTIN.firewall.txt bloque le LAN qui est situé derrière nous. En d'autres termes, nous ne faisons pas confiance aux réseaux auxquels nous sommes connectés. Nous n'autorisons personne de notre LAN à se connecter à l'Internet, sauf pour des tâches spécifiques. Les seules choses autorisées sont les accès POP3, HTTP et FTP. Nous ne faisons également pas confiance aux utilisateurs internes pour accéder au pare–feu comme pour les utilisateurs sur l'Internet.

Le script `rc.UTIN.firewall.txt` nécessite que les options suivantes soient compilées en statique dans le noyau, ou en modules. Sans une ou plusieurs des ces options, le script ne fonctionnera pas correctement ou sera même inutilisable. Si vous modifiez ce script vous aurez peut être besoin d'options supplémentaires qui devront aussi être compilées dans le noyau.

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

Le script suit la règle d'or de ne faire confiance en personne, pas même en vos propres employés. C'est malheureux à dire, mais une grande partie du hacking/cracking dans une entreprise provient du personnel interne. Ce script vous donne quelques clés pour remédier à ça. Il n'est pas très différent du script `rc.firewall.txt`.

14.6. rc.test–iptables.txt

Le script rc.test–iptables.txt peut être utilisé pour tester toutes les différentes chaînes, mais il peut nécessiter quelques adaptations en fonction de votre configuration, comme l'activation de ***ip_forwarding***, ou le masquerading, etc. Il fonctionnera dans la plupart des cas, si vous avez une configuration des tables de base chargées dans le noyau. Certaines cibles ***LOG*** sont activées ce qui permet de journaliser les requêtes et les

réponses aux pings. De cette façon vous aurez des informations sur les chaînes traversées et dans quel ordre. Par exemple, lancez ce script et faites :

```
ping -c 1 host.on.the.internet
```

Et ***tail –n 0 –f /var/log/messages*** pendant que vous exécutez la première commande. Ceci vous indiquera les diverses chaînes utilisées, et dans quel ordre, jusqu'à ce que les entrées du journal s'arrêtent pour quelque raison.

Note

Ce script a été écrit dans un but de test uniquement. En d'autres termes, ce n'est pas une bonne idée d'avoir des règles comme celles–là qui journalisent tout car vos fichiers de log se rempliront très vite et il pourrait être confronté à une attaque de type DoS.

14.7. rc.flush–iptables.txt

Le script rc.flush–iptables.txt ne devrait pas être appelé script à proprement parler. Ce script rc.flush–iptables.txt réinitialise toutes les tables et les règles. Il commence en activant par défaut les stratégies en mode ***ACCEPT*** sur les chaînes INPUT, OUTPUT et FORWARD de la table filter. Après ça nous réinitialisons les stratégies des chaînes PREROUTING, POSTROUTING et OUTPUT de la table nat. Nous faisons ça en premier ainsi nous ne sommes pas gênés par les fermetures de connexion. Ce script a pour but la mise en place de votre pare–feu et de le tester.

Après cela nous réinitialisons toutes les chaînes, en premier la table filter et ensuite la table NAT. De cette façon nous savons qu'il n'y a pas de règles redondantes. Quand tout ceci est fait, nous passons à la section suivante dans laquelle nous supprimons toutes les chaînes utilisateur dans les tables NAT et filter. Quand cette étape est terminée, nous considérons que le script est achevé. Vous pouvez ajouter des règles pour réinitialiser votre table mangle si vous l'utilisez.

Note

Un dernier mot. Certaines personnes m'ont demandé de mettre ce script dans la syntaxe du rc.firewall original utilisé par Red Hat Linux où vous tapez quelque chose comme rc.firewall start et le script démarre. Cependant, je ne l'ai pas fait car il s'agit d'un didacticiel destiné à vous donner des idées, et il ne devra pas grossir démesurément avec des syntaxes particulières. Ajouter des syntaxes et autres scripts shell peut aussi le rendre plus difficile à lire.

14.8. Limit–match.txt

Le script limit–match.txt est un mirroir du script test qui vous permet de tester la correspondance limit et de voir comment elle fonctionne. Chargez ce script, et ensuite envoyez des paquets à différents intervalles. Toutes les réponses seront bloquées jusqu'à ce que le seuil limite soit atteint.

14.9. Pid–owner.txt

Le script pid–owner.txt est un petit exemple qui indique comment vous pouvez utiliser la correspondance PID. Il ne fait rien de réel, mais vous permet une fois exécuté la commande ***iptables –L –v*** de savoir quelle règle est actuellement appariée.

14.10. Recent–match.txt

Ce script recent–match.txt indique comment la correspondance recent est utilisée. Pour une explication complète regardez la section Correspondance Recent du chapitre Correspondances.

14.11. Sid–owner.txt

Le sid–owner.txt est un petit exemple montrant comment utiliser la correspondance SID. Il n'a rien de réel, en lançant la commande ***iptables –L –v*** vous verrez les règles appariées actuellement.

14.12. Ttl–inc.txt

Un petit exemple ttl–inc.txt. Il indique comment rendre invisible le pare–feu/routeur aux traceroutes, lesquels révèlent beaucoup d'informations utiles aux attaquants possibles.

14.13. Iptables–save

Un petit example script utilisé dans le chapitre Sauvegarde et restauration des tables de règles importantes pour illustrer comment iptables–save peut être utilisé. Ce script ne doit être utilisé que comme une référence, il ne fonctionne pas.

Chapitre 15. Interfaces utilisateur graphiques pour Iptables/netfilter

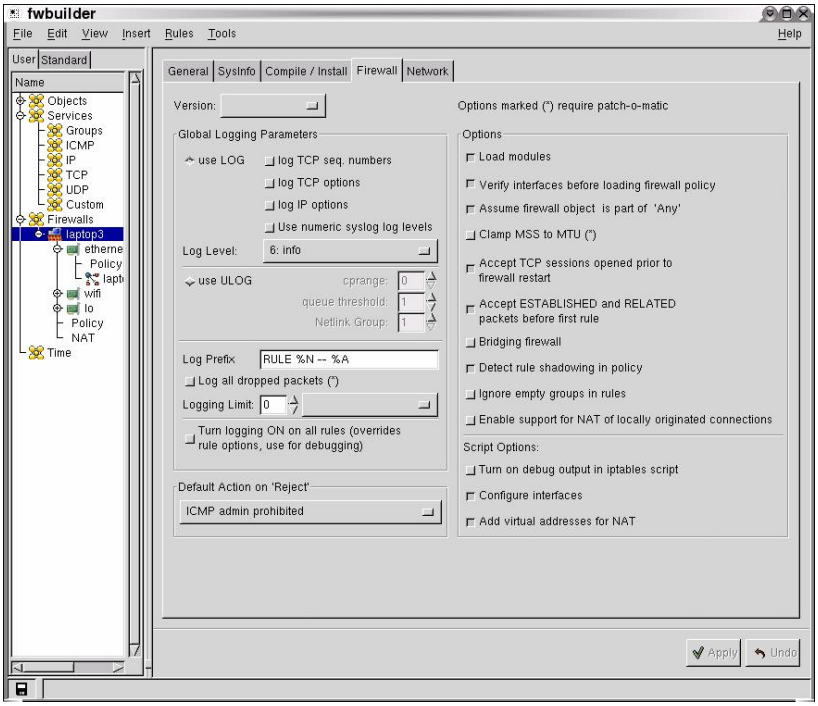
Un aspect de iptables et netfilter que nous n'avons pas encore vu, est l'interface utilisateur graphique. Un des principaux problèmes avec ces interfaces est que netfilter est très complexe, ce qui peut produire des effets étranges. Pour cette raison, ce peut être un tâche décourageante de créer ce type d'interfaces.

Plusieurs personnes et organismes ont essayé de créer des interfaces graphiques pour netfilter et iptables, certaines avec succès, d'autres ont abandonné après un certain temps. Ce chapitre est une petite compilation de certaines interfaces graphiques qu'il peut être intéressant de regarder.

15.1. fwbuilder

Firewall Builder, ou simplement fwbuilder, est un outil extrêmement souple et puissant qui peut être utilisé pour créer vos propres pare–feux, ou pour maintenir plusieurs pare–feux. Il peut être utilisé pour créer plusieurs stratégies de pare–feux différentes, incluant iptables (Linux 2.4 et 2.6), ipfilter (FreeBSD, NetBSD, etc.), OpenBSD pf et, avec un module qui doit être acheté, Cisco PIX.

Fwbuilder a beaucoup de succès et continue d'être développé. Il fonctionne sur un système hôte séparé, sur lequel vous créez vos fichiers de stratégies, il les copie ensuite et les exécute sur le système cible. Il peut maintenir depuis une simple table de règles jusqu'à de plus importantes et compliquées. Il a également des possibilités d'extensions selon les différentes versions d'iptables, en fonction des cibles/correspondances disponibles sur chaque système, etc. Le résultat peut être sauvegardé dans un fichier xml, ou un fichier de configuration (ex. les scripts pare–feux).

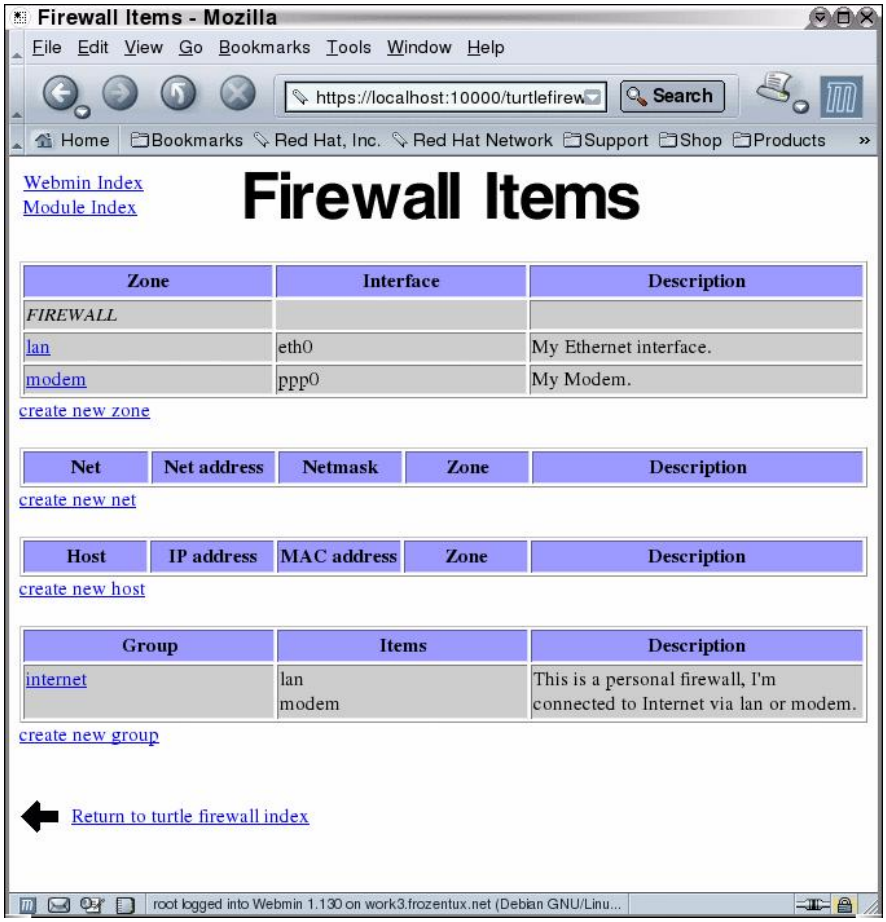


Vous pouvez voir la "configuration" du pare−feu dans l'exemple suivant, et les principaux menus de fwbuilder. Fwbuilder peut être trouvé sur http://www.fwbuilder.org.

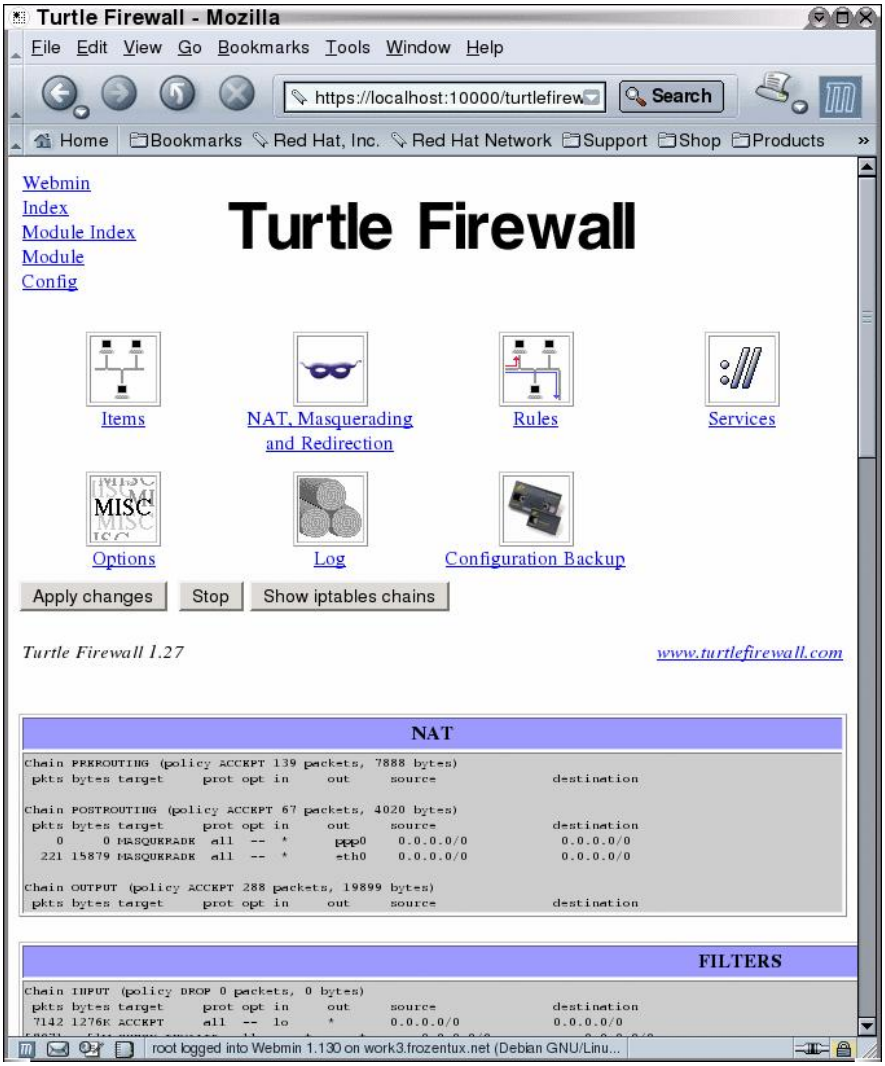
15.2. Projet Turtle Firewall

Turtle Firewall est une excellente, mais simple, interface pour iptables. Il est intégré dans Webmin (une interface d'administration web). Il est assez basique, et n'est ni complexe ni ne peut supporter des modifications complexes comme fwbuilder, mais il est très capable de maintenir des plus simples pare−feux à certains plus avancés.

Un gros avantage avec Turtle Firewall est qu'il est basé sur une interface web, et donc peut être contrôlé à distance d'une manière complètement différente de fwbuilder et de la plupart des autres outils. Bien sûr, il présente plus de risques en termes de sécurité car Webmin est un service supplémentaire séparé fonctionnant sur le pare−feu lui−même.



La capture d'écran ci–dessus montre les rubriques de Turtle Firewall, dans lesquelles vous pouvez configurer les interfaces réseau, et d'autres choses.



La dernière capture d'écran montre l'écran principal de Turtle Firewall, avec les règles en bas de page. L'ensemble des règles n'est pas affiché, comme vous pouvez le voir, mais vous avez une idée générale de son fonctionnement.

Vous trouverez Turtle Firewall sur http://www.turtlefirewall.com/.

15.3. Integrated Secure Communications System

Integrated Secure Communications System, ou en plus court ISCS, est encore en phase de développement, et aucune version publique n'est disponible. Cependant, il semblerait être un outil très utile une fois finalisé. Le développeur a des exigences très élevées, et c'est la principale raison pour laquelle il n'est pas encore terminé. ISCS intègre diverses fonctionnalités dans une seule interface d'administration. De façon basique, ceci indique qu'une fois le projet réalisé, pour pourrez configurer tous vos pare–feux depuis un point centralisé en utilisant une seule interface graphique, incluant les VPN, VLAN, tunnel, syscontrol, etc.

Le but principal du développeur de ISCS est de simplifier l'administration est de supprimer les tâches fastidieuses pour les administrateurs, et ainsi économiser des heures de travail. Ceci est réalisé en joignant les stratégies, ensuite le programme crée les tables de règles et les "envoie" vers des "points d'exécution" (ex. pare–feux, proxies, etc.). L'administrateur n'a pas à écrire les tables de règles, mais simplement définir les stratégies qui seront exécutées par ISCS.

Cet outil n'est pas encore achevé, comme je l'ai écrit. Cependant, j'ai été en contact avec le principal développeur du projet, et c'est réellement un travail très important. Quand il sera fini, je crois qu'il sera un des meilleurs outils du marché. Vous pouvez trouver le projet ISCS sur http://iscs.sourceforge.net/.

Note

Le principal développeur, John Sullivan, m'a dit de demander aux personnes intéressées de l'aider dans le développement. Le projet est très lourd, et il a vraiment besoin d'aide. Si vous êtes capables de le faire, votre aide sera la bienvenue.

15.4. IPMenu

IPMenu est un programme très utile, il est cependant simple à utiliser et ne demande pas trop de ressources ou de bande passante. C'est un programme en mode console, ainsi il fonctionne parfaitement avec une connexion SSH par exemple. Il s'exécute très bien sur des machines possédant un vieux modem.

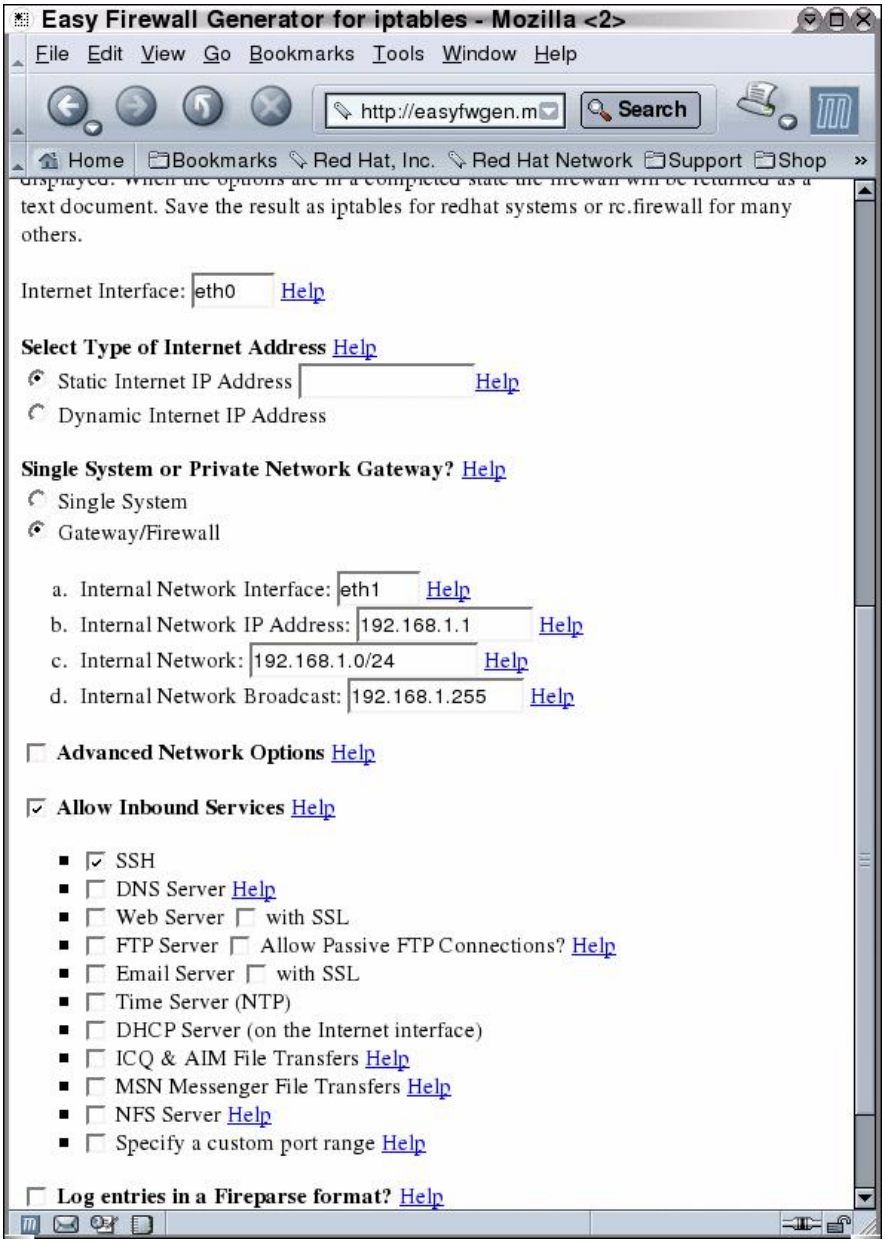
Comme vous pouvez le voir sur l'image, il contient toutes les fonctionnalités d'iptables, incluant le filtrage, mangle et nat. Il peut aussi maintenir les tables de routage et la bande passante et sauvegarder et restaurer les tables de règles.

Comme vous l'avez vu dans l'image précédente, le programme est plutôt basique, mais il convient dans la plupart des cas. En premier, il est très simple, et peut être utilisé pour l'administration à distance, et fonctionne avec SSH via une console standard, il est également sécurisé. Vous pouvez trouver le programme sur http://users.pandora.be/stes/ipmenu.html.

15.5. Easy Firewall Generator

Easy Firewall Generator est un autre développement intéressant. De façon basique, c'est une page web PHP dans laquelle vous spécifiez les options de votre pare−feu, ensuite la configuration se fait en cliquant sur un bouton, ce qui génère une table de règles iptables que vous pouvez utliser.

Le script contient toutes les règles de base, avec en plus certaines autres destinées à contenir des modèles insolites dans les paquets. Il contient aussi les modifications sysctl IP spécifiques qui peuvent être nécessaires, le chargement des modules, etc. La table de règles est écrite dans le format init.d de red Hat.



Cette capture d'écran montre une des étapes finales de la configuration du script du pare–feu créé par le programme. Vous pouvez trouver plus d'information sur http://easyfwgen.morizot.net/.

15.6. Partie suivante

Dans ce chapitre nous avons vu ce que nous pouvons faire avec certaines interfaces graphiques. Notez qu'il existe beaucoup plus d'interfaces sur le marché. La plupart d'entre elles sont open source et libres d'utilisation, tandis que certaines sont des applications commerciales qui nécessitent d'être achetées pour obtenir un support ou une fonctionnalité complets.

Annexe A. Explication détaillée des commandes spéciales

A.1. Affichage de votre table de règles

Pour lister votre table de règles vous devez passer une option spéciale à la commande ***iptables***, dont nous avons brièvement parlé dans le chapitre Création d'une règle. Ceci pourrait ressembler à ça :

iptables –L

Cette commande affichera votre table de règles active, et la traduira dans une forme lisible. Par exemple, elle traduira tous les différents ports selon le fichier `/etc/services` aussi bien que le DNS de toutes les adresses IP pour en obtenir des enregistrements. Cette dernière peut poser un léger problème. Par exemple, elle tentera de résoudre les adresses IP LAN, i.e., `192.168.1.1`, en quelque chose de plus fonctionnel. `192.168.0.0/16` est une plage d'adresses privées et la commande semblera se figer. Pour résoudre ce problème nous ferons comme suit :

iptables –L –n

Une autre chose qui peut être intéressante est de voir quelques statistiques concernant chaque stratégie, règle et chaîne.

iptables –L –n –v

N'oubliez pas qu'il est également possible d'afficher les tables nat et mangle. Ceci est fait avec l'option –t, comme ça :

iptables –L –t nat

Il y a aussi quelques fichiers qu'il serait intéressant de regarder dans le système de fichiers `/proc`. Par exemple, savoir quelles connexions sont en cours dans la table conntrack. Cette table contient toutes les connexions tracées et sert de table de base, ainsi nous pouvons toujours connaître l'état de nos connexions. Cette table ne peut être éditée, et même si c'était le cas, ce ne serait pas une bonne idée. Pour voir la table exécutez la commande :

cat /proc/net/ip_conntrack | less

La commande ci–dessus indique toutes les connexions traçées même si ça peut être un peu difficile à tout comprendre.

A.2. Mise à jour et vidange des tables

Si à un certain moment vous sabotez votre ***iptables***, il existe des commandes pour les vidanger, ainsi vous n'aurez pas à rebooter. J'ai abordé cette question plusieurs fois, ainsi je crois que la réponse ici sera correcte. Si vous faites une erreur dans une règle, vous n'avez juste qu'à changer les paramètres de ***–A*** en ***–D*** dans la ligne qui contient l'erreur. ***iptables*** trouvera la ligne erronée et l'effacera pour vous, dans le cas où vous avez de multiples lignes avec des erreurs identiques dans la chaîne, il supprimera la première instance et fera l'appariement de votre règle. Si ce n'est pas le comportement voulu vous pouvez essayer d'utiliser l'option ***–D*** comme dans ***iptables –D INPUT 10*** laquelle effacera la dixième règle de la chaîne INPUT.

Il y a aussi certains cas où vous voudrez vider une chaîne complète, dans ces cas vous exécuterez l'option ***–F***. Par exemple, ***iptables –F INPUT*** supprimera la chaîne INPUT en totalité, mais ne modifiera pas la stratégie par défaut, ainsi si elle est placée à DROP vous bloquerez la chaîne INPUT. Pour réinitialiser la stratégie de la chaîne, qui était placée à DROP, faites par exemple, ***iptables –P INPUT ACCEPT***.

J'ai écrit un rc.flush–iptables.txt (disponible dans l'annexe) qui vide et réinitialise votre ***iptables*** que vous pouvez utiliser lors de l'écriture de votre fichier `rc.firewall.txt`. Une chose encore; si vous démarrez avec une table mangle qui contient des erreurs, le script ne les supprimera pas, il est plus simple d'ajouter quelques lignes pour les supprimer. je ne l'ai pas fait car la table mangle n'est pas utilisée dans mon script `rc.firewall.txt`.

Annexe B. Problèmes et questions courants

B.1. Problèmes de chargement des modules

Vous pouvez rencontrer quelques problèmes lors du chargement des modules. Par exemple, obtenir des messages indiquant qu'il n'existe pas de module de ce nom, etc. Ils peuvent ressembler à ça :

```
insmod: iptable_filter: no module by that name found
```

Ces modules peuvent avoir été compilés statiquement dans le noyau. C'est la première des choses à regarder pour résoudre ce problème. Le moyen le plus simple pour vérifier si ces modules sont déjà chargés ou sont compilés en statique dans le noyau, est d'essayer une commande qui utilise ces fonctionnalités. Dans le cas ci–dessus, nous ne pouvons pas charger la table filter. Si cette fonctionnalité n'est pas présente, nous ne pourrons pas utiliser la table filter. Pour vérifier ceci, faites :

iptables -t filter -L

Ce qui afficherait soit la liste de toutes les chaînes de la table filtre, soit un échec. Si tout est correct, regarder si vous avez des règles insérées ou non.

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

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

Si la table filtre n'est pas chargée, vous pouvez obtenir une erreur de ce genre :

```
iptables v1.2.5: can't initialize iptables table `filter': Table \
     does not exist (do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

Ceci est un peu plus sérieux car, en premier lieu, la fonctionnalité n'est pas compilée dans le noyau, en second lieu, le module n'est pas trouvé dans le chemin normal. Ce qui peut indiquer que soit vous avez oublié d'installer les modules, ou vous avez oublié d'exécuter un ***depmod –a*** pour mettre à jour la base de données de vos modules, soit vous n'avez pas compilé cette fonctionnalité en module ou statiquement dans le noyau. Il peut y avoir d'autres raisons pour que le module ne soit pas chargé, mais c'est une des principales. La plupart de ces problèmes sont aisément solvables. Le premier peut être réglé en faisant un simple ***make modules_install*** dans le répertoire source du noyau (si le source a déjà été compilé et que les modules sont présents). Le second problème est résolu en exécutant ***depmod –a*** et en regardant si ça fonctionne ensuite. Le troisième cas est un peu hors sujet ici. Vous trouverez plus d'information sur Linux Documentation Project.

Une autre erreur qui peut survenir est celle–ci :

```
iptables: No chain/target/match by that name
```

Cette erreur nous indique qu'il n'y a pas de chaîne, cible ou correspondance. Ceci peut dépendre de beaucoup de facteurs, le plus courant étant que vous avez mal nommé la chaîne, cible ou correspondance en question.

Également, ça peut arriver si vous essayez d'utiliser une correspondance non disponible, soit parce que le bon module n'est pas chargé, ou non compilé dans le noyau, soit iptables n'arrive pas à charger automatiquement le module. En général, il faut vérifier avec les solutions ci−dessus, mais aussi regarder si les cibles sont bien nommées dans vos règles.

B.2. Paquets état NEW sans bit SYN placé

Il y a certaines *fonctionnalités* d'***iptables*** qui ne sont pas très bien documentées et qui peuvent être laissées de côté par certaines personnes (y compris moi). Si vous utilisez l'état ***NEW***, les paquets avec le bit SYN non placé traverseront votre pare−feu. Cette fonctionnalité existe car dans certains cas nous pouvons considérer qu'un paquet peut faire partie d'une connexion déjà ***ESTABLISHED*** sur, par exemple, un autre pare−feu. Cette fonctionnalité offre la possibilité d'avoir deux ou plusieurs pare−feux, et un des pare−feux peut être désactivé sans perte de données. Le pare−feu du sous−réseau peut alors être remplacé par un pare−feu secondaire. Ceci peut conduire cependant au fait que l'état ***NEW*** autorisera toute sorte de connexion TCP, sans se soucier si c'est un établissement de liaison à trois voies ou non. Pour surveiller ce problème nous ajoutons les règles suivantes à nos pare−feux dans les chaînes INPUT, OUTPUT et FORWARD.

```
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG \
     --log-prefix "New not syn:"
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
```

Attention

Les règles ci−dessus surveillent ce problème. C'est un comportement mal documenté du projet ***Netfilter/iptables*** qui devrait être mis en avant.

Notez qu'il existe certains problèmes avec les règles ci−dessus et les mauvaises implémentations TCP/IP de Microsoft. Ces règles peuvent provoquer dans certaines conditions que des paquets générés par des produits Microsoft soient labellisés avec l'état ***NEW*** et soient journalisés et supprimés. À ma connaissance, ça ne produit cependant pas de coupure de connexion. Le problème survient lorsque la connexion est fermée, le FIN/ACK final est envoyé, la machine d'état de ***Netfilter*** ferme la connexion et n'apparaît plus dans la table conntrack. À ce moment l'implémentation Microsoft envoie un autre paquet considéré comme ***NEW*** mais il manque le bit SYN, il est donc apparié par les règles. En d'autres termes, ne vous inquiétez pas trop à propos de cette règle, ou alors placez l'option ***−−log−headers*** dans la règle et journalisez les en−têtes, ainsi vous aurez une meilleure vision de ce à quoi ressemble le paquet.

Il y a un problème plus connu avec ces règles. Si quelqu'un est connecté au pare−feu, depuis le LAN, et que vos scripts sont prévus pour s'activer lors du lancement d'une connexion PPP. Dans ce cas, quand vous démarrez une connexion PPP, la personne connectée depuis le LAN sera plus ou moins supprimée. Ceci s'applique seulement quand vous travaillez avec les codes nat et conntrack en modules, et que les modules sont chargés et déchargés chaque fois que vous lancez le script. Une autre façon de rencontrer ce problème est de lancer le script `rc.firewall.txt` par une connexion en telnet depuis un hôte non présent sur ce pare−feu. Pour l'ajouter simplement, connectez vous en ***telnet*** ou autre type de connexion. Démarrez la connexion en traçant les modules, ensuite chargez les règles de paquet ***NEW*** non−SYN. Enfin, le ***client telnet*** ou le ***démon*** tentera d'envoyer quelque chose. Le code du traçage de connexion ne reconnaîtra pas cette connexion comme légale car il n'a pas vu les paquets dans aucune direction auparavant, ainsi il n'y aura aucun bit SYN de placé car ce n'est pas le premier paquet de la connexion. Ce paquet sera apparié avec les règles, journalisé et ensuite supprimé.

B.3. SYN/ACK et les paquets NEW

Certaines attaques par mystification TCP utilisent une technique appelée Sequence Number Prediction. Dans ce type d'attaque, l'attaquant mystifie certaines adresses IP d'hôtes, et essaie de prédire la suite de chiffres

utilisée par l'hôte.

Regardons un cas typique de mystification TCP (TCP spoofing) par prédiction de suite de chiffres. Les joueurs : "l'attaquant" [A], tente d'envoyer des paquets à la "victime" [V], prétendant être un "autre hôte" [O].

1. [A] envoie un SYN vers [V] avec l'adresse IP source de [O].
2. [V] répond à [O] par un SYN/ACK.
3. donc [O] répondra à un SYN/ACK inconnu par RST et l'attaque sera réussie, mais nous supposons que [O] est déconnecté.
4. [A] peut maintenant parler à [V] en prétendant être [O] tant qu'il peut prédire correctement la séquence de chiffres.

Tant que nous n'envoyons pas le paquet RST au SYN/ACK inconnu à l'étape 3, nous permettons à [V] d'être attaqué, et nous mêmes seront incriminés. La courtoisie serait désormais, d'envoyer le RST à [V] de façon correcte. Si nous utilisons les règles NEW non−SYN spécifiées dans la table de règles, les paquets SYN/ACK seront supprimés. Nous aurons donc les règles suivantes dans la chaîne bad_tcp_packets, juste au−dessus des règles NEW non−SYN :

```
iptables -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
```

Une chance serait que [O] dans ce scenario soit relativement petit, mais ces règles seront sûres dans la plupart des cas. Sauf quand vous utilisez plusieurs pare−feux redondants qui prennent la suite des paquets ou des flux de chacun des autres. Dans ces cas là, certaines connexions peuvent être bloquées, même si elles sont légales. Cette règle peut aussi autoriser certains balayages de port, pour voir ce qui apparaît au niveau de notre pare−feu, mais elle ne pourra pas en dévoiler d'avantage.

B.4. Fournisseurs d'accès Internet qui utilisent des adresses IP assignées

J'ai ajouté ceci, car un ami m'a dit quelque chose que j'avais complètement oublié. Certains fournisseurs d'accès Internet stupides utilisent des adresses IP assignées par l'*IANA* pour leurs réseaux locaux sur lesquels vous vous connectez. Par exemple, le suédois Telia utilise ce processus sur ses serveurs DNS, lesquels se servent de la plage d'adresses IP 10.x.x.x. Un problème courant que vous pouvez rencontrer lors de l'écriture de vos scripts, est que vous n'autorisez pas les connexions depuis la plage d'adresses IP 10.x.x.x vers vous−même, à cause des possibilités de spoofing. C'est malheureusement un de ces exemples avec lesquels vous pouvez avoir des problèmes avec ces règles. Vous pouvez juste insérer une règle ***ACCEPT*** au−dessus de la section concernant le spoofing pour autoriser le trafic depuis ces serveurs DNS, ou désactiver cette partie du script. Ça ressemble à ceci :

```
/usr/local/sbin/iptables -t nat -I PREROUTING -i eth1 -s \
     10.0.0.1/32 -j ACCEPT
```

Je voudrai prendre un moment pour parler de ces fournisseurs d'accès Internet (FAI). Ces plages d'adresses IP ne sont pas destinées pour votre usage à discrétion, du moins à ma connaissance. Pour de gros sites d'entreprises c'est plus que d'accord, ou pour votre réseau local, mais vous n'êtes pas supposés nous forcer à les utiliser juste par caprice de votre part. Vous êtes de gros FAIs, et si vous ne pouvez pas vous payer 3−4 adresses IP pour vos serveurs DNS, je n'ai pas beaucoup confiance en vous.

B.5. Laissez les requêtes DHCP traverser iptables

C'est réellement une tâche facile, une fois que vous savez comment DHCP fonctionne, cependant, vous devez prendre des précautions sur ce que vous laissez passer ou non. En premier lieu, nous devons savoir que DHCP fonctionne sur le protocole UDP. Donc, c'est la première chose à voir. En second lieu, nous devons vérifier depuis quelle interface les requêtes sont envoyées et reçues. Par exemple, si notre interface eth0 est activée par DHCP, nous n'autoriserons pas les requêtes DHCP sur eth1. Pour rendre la règle un peu plus précise, nous n'autorisons que les ports UDP utilisés par DHCP, qui sont les ports 67 et 68. Ce sont les critères que nous choisissons pour apparier les paquets, et que nous autorisons.

```
$IPTABLES  -I INPUT -i $LAN_IFACE -p udp --dport 67:68 --sport \
     67:68 -j ACCEPT
```

Notez que nous autorisons tout le trafic depuis et vers les ports 67 et 68, cependant, ce n'est pas un gros problème car nous n'acceptons que les requêtes des hôtes établissant la connexion depuis les ports 67 et 68. Cette règle peut, bien sûr, être encore plus restrictive, mais elle semble suffisante pour accepter les requêtes DHCP sans ouvrir de larges failles.

B.6. Problèmes avec le DCC de mIRC

mIRC utilise un réglage spécial qui permet de se connecter à travers un pare−feu pour établir une connexion DCC sans que le pare−feu en ait connaissance. Si cette option est utilisée avec iptables et en particulier avec les modules ip_conntrack_irc et ip_nat_irc, il ne fonctionnera pas. Le problème est que mIRC NATe automatiquement les paquets pour vous, et quand ces paquets atteignent le pare−feu, celui−ci ne sait pas quoi en faire. mIRC ne s'attend pas à ce que le pare−feu soit suffisamment sensible pour analyser ceci en expédiant une requête au serveur IRC au sujet de cette adresse IP et en envoyant les requêtes DCC avec cette adresse.

Activer l'option de configuration "je suis derrière un pare−feu" et utiliser les modules ip_conntrack_irc et ip_nat_irc permettra à Netfilter de créer des entrées de logs avec le contenu suivant "Forged DCC send packet".

La solution la plus simple est de désactiver cette option de mIRC et de laisser iptables faire le travail. Ce qui veut dire, que vous indiquerez à mIRC qu'il n'est *pas* derrière un pare−feu.

Annexe C. Types ICMP

Voici un liste complète des types ICMP. Notez la référence qui pointe vers la RFC ou la personne qui a introduit le type et le code. Pour une liste complètement à jour des types et des codes ICMP, voir le document icmp−parameters sur Internet Assigned Numbers Authority.

Tableau C.1. Types ICMP

TYPE	CODE	Description	requête	Erreur	Référence
0	0	Echo Reply	x		RFC792
3	0	Network Unreachable		x	RFC792
3	1	Host Unreachable		x	RFC792
3	2	Protocol Unreachable		x	RFC792
3	3	Port Unreachable		x	RFC792

3	4	Fragmentation needed but no frag. bit set		x	RFC792
3	5	Source routing failed		x	RFC792
3	6	Destination network unknown		x	RFC792
3	7	Destination host unknown		x	RFC792
3	8	Source host isolated (obsolete)		x	RFC792
3	9	Destination network administratively prohibited		x	RFC792
3	10	Destination host administratively prohibited		x	RFC792
3	11	Network unreachable for TOS		x	RFC792
3	12	Host unreachable for TOS		x	RFC792
3	13	Communication administratively prohibited by filtering		x	RFC1812
3	14	Host precedence violation		x	RFC1812
3	15	Precedence cutoff in effect		x	RFC1812
4	0	Source quench			RFC792
5	0	Redirect for network			RFC792
5	1	Redirect for host			
5	2	Redirect for TOS and network			RFC792
5	3	Redirect for TOS and host			RFC792
8	0	Echo request	x		RFC792
9	0	Router advertisement – Normal router advertisement			RFC1256
9	16	Router advertisement – Does not route common traffic			RFC2002
10	0	Route selection			RFC1256
11	0	TTL equals 0 during transit		x	RFC792
11	1	TTL equals 0 during reassembly		x	RFC792
12	0	IP header bad (catchall error)		x	RFC792
12	1	Required options missing		x	RFC1108
12	2	IP Header bad length		x	RFC792
13	0	Timestamp request (obsolete)	x		RFC792
14		Timestamp reply (obsolete)	x		RFC792
15	0	Information request (obsolete)	x		RFC792
16	0	Information reply (obsolete)	x		RFC792
17	0	Address mask request	x		RFC950
18	0	Address mask reply	x		RFC950
20–29		Reserved for robustness experiment			Zaw–Sing Su
30	0	Traceroute	x		RFC1393
31	0	Datagram Conversion Error		x	RFC1475

32	0	Mobile Host Redirect			David Johnson
33	0	IPv6 Where−Are−You	x		Bill Simpson
34	0	IPv6 I−Am−Here	x		Bill Simpson
35	0	Mobile Registration Request	x		Bill Simpson
36	0	Mobile Registration Reply	x		Bill Simpson
39	0	SKIP			Tom Markson
40	0	Photuris			RFC2521

Annexe D. Options TCP

Cette annexe est une simple et brève liste des options TCP officiellement reconnues. Ces références et chiffres sont tirés du site Internet Assigned Numbers Authority. Le fichier principal peut être trouvé sur http://www.iana.org/assignments/tcp−parameters. Les détails pour le contact avec les personnes référencées dans ce document ont été supprimés, pour leur permettre d'avoir moins de charge de travail, heureusement.

Tableau D.1. Options TCP

Copie	Classe	Nombre	Valeur	Nom	Référence
0	0	0	0	EOOL – End of Options List	[RFC791,JBP]
0	0	1	1	NOP – No Operation	[RFC791,JBP]
1	0	2	130	SEC – Security	[RFC1108]
1	0	3	131	LSR – Loose Source Route	[RFC791,JBP]
0	2	4	68	TS – Time Stamp	[RFC791,JBP]
1	0	5	133	E−SEC – Extended Security	[RFC1108]
1	0	6	134	CIPSO – Commercial Security	[???]
0	0	7	7	RR – Record Route	[RFC791,JBP]
1	0	8	136	SID – Stream ID	[RFC791,JBP]
1	0	9	137	SSR – Strict Source Route	[RFC791,JBP]
0	0	10	10	ZSU – Experimental Measurement	[ZSu]
0	0	11	11	MTUP – MTU Probe	[RFC1191]*
0	0	12	12	MTUR – MTU Reply	[RFC1191]*
1	2	13	205	FINN – Experimental Flow Control	[Finn]
1	0	14	142	VISA – Experimental Access Control	[Estrin]
0	0	15	15	ENCODE – ???	[VerSteeg]
1	0	16	144	IMITD – IMI Traffic Descriptor	[Lee]
1	0	17	145	EIP – Extended Internet Protocol	[RFC1385]
0	2	18	82	TR – Traceroute	[RFC1393]
1	0	19	147	ADDEXT – Address Extension	[Ullmann IPv7]
1	0	20	148	RTRALT – Router Alert	[RFC2113]

1	0	21	149	SDB – Selective Directed Broadcast	[Graff]
1	0	22	150	NSAPA – NSAP Addresses	[Carpenter]
1	0	23	151	DPS – Dynamic Packet State	[Malis]
1	0	24	152	UMP – Upstream Multicast Pkt.	[Farinacci]

Annexe E. Autres ressources et liens

Ici vous avez une liste de ressources et de liens avec lesquels vous pouvez obtenir une information spécifique :

- ip–sysctl.txt – depuis le noyau 2.4.14. Un peu court mais une bonne référence pour le contrôle de réseau IP et ce qu'il fait avec le noyau.
- RFC 768 – User Datagram Protocol – RFC officielle décrivant comment le protocole UDP doit être utilisé, en détail, avec tous ses en–têtes.
- RFC 791 – Internet Protocol – La spécification IP toujours utilisée sur l'Internet, avec les ajouts et mises à jour. La base est toujours la même pour ipv4.
- RFC 792 – Internet Control Message Protocol – Ressource définitive pour toute information sur les paquets ICMP. Toute information technique dont vous avez besoin sur le protocole ICMP. Écrite par J.Postel.
- RFC 793 – Transmission Control Protocol – C'est la ressource pour savoir comment se comporte TCP sur tous les hôtes. Ce document a été le standard de TCP depuis 1981 et après. Extrêmement technique, mais incontournable pour quiconque veut connaître TCP dans le détail. C'est à l'origine un rapport écrit par J.Postel pour le Department of Defense.
- RFC 1122 – Requirements for Internet Hosts – Communication Layers – Cette RFC définit les pré–requis pour les logiciels s'exécutant sur un hôte Internet, spécifiquement pour les couches communication.
- RFC 1349 – Type of Service in the Internet Protocol Suite – RFC décrivant certains changements et clarifications du champ TOS dans l'en–tête IP.
- RFC 2401 – Security Architecture for the Internet Protocol – RFC sur l'implémentation et la standardisation de IPSEC. À lire si vous travaillez avec IPSEC.
- RFC 2474 – Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers – Dans ce document vous trouverez comment DiffServ fonctionne, et également l'information nécessaire sur les additions/modifications du protocole TCP/IP requises pour faire fonctionner le protocole DiffServ.
- RFC 2638 – A Two–bit Differentiated Services Architecture for the Internet – RFC qui décrit un méthode d'implémentation de deux architectures DiffServ en une seule. Décrites à l'origine par D.Clark et van Jacobsen au meeting IETH de Munich en 1997.
- RFC 3168 – The Addition of Explicit Congestion Notification (ECN) to IP – RFC qui décrit comment ECN est utilisé au niveau technique et comment il est implémenté dans les protocoles TCP et IP. Écrit par K.Ramakrishnan, S.Floyd et D.Black.
- RFC 3260 – New Terminology and Clarifications for Diffserv – Ce mémo présente les actes du groupe de travail DiffServ concernant la nouvelle terminologie, et indique certaines clarifications techniques.
- ip_dynaddr.txt – Depuis le noyau 2.4.14. Une courte référence aux réglages de ip_dynaddr disponibles via sysctl et le système de fichiers proc.
- iptables.8 – Page de manuel de iptables 1.3.1. Version HTML de la page de manuel qui est une excellente référence pour la lecture/écriture de tables de règles iptables. À avoir toujours sous la main.
- Ipsysctl tutorial – Un autre didacticiel que j'ai écrit sur le contrôle du système IP dans Linux. Une tentative pour faire une liste complète de toutes les variables IP qui peuvent être insérées au vol dans Linux.
- Policy Routing Using Linux – Excellent livre qui a maintenant été publié sur l'internet en rapport à la politique de routage dans Linux. Écrit par Matthew G. Marsh.
- Firewall rules table – Un petit document en PDF gracieusement fourni pour ce projet par Stuart Clark, qui présente en formulaire toute l'information nécessaire pour votre pare–feu, de manière simple.

- http://www.netfilter.org/ – Le site officiel de ***Netfilter*** et ***iptables***. C'est un must pour quiconque implémente ***iptables*** et ***Netfilter*** dans Linux.
- http://www.insecure.org/nmap/ – Nmap est un des meilleurs, et plus connus, scanneur de ports disponible. Très utile lors du débogage de vos scripts de pare−feux.
- http://www.netfilter.org/documentation/index.html#FAQ – La *Frequently Asked Questions* (FAQ) officielle de ***Netfilter***. Un bon endroit pour démarrer avec ***iptables*** et ***Netfilter***.
- http://www.netfilter.org/unreliable−guides/packet−filtering−HOWTO/index.html – Le Unreliable Guide de Rusty Russells sur le filtrage de paquet. Excellente documentation sur le fitrage de paquet avec ***iptables*** écrite par un des développeurs de ***iptables*** et ***Netfilter***.
- http://www.netfilter.org/unreliable−guides/NAT−HOWTO/index.html – Unreliable Guide de R.Russells sur la traduction d'adresse réseau. Excellente documentation sur la NAT dans ***iptables*** et ***Netfilter***.
- http://www.netfilter.org/unreliable−guides/netfilter−hacking−HOWTO/index.html – Unreliable Netfilter Hacking HOW−TO de R.Russells. Une des rares documentations sur la façon d'écrire du code dans ***Netfilter*** du code−base dans l'espace utilisateur ***iptables*** et l'espace noyau.
- http://www.linuxguruz.org/iptables/ – Excellente page avec des liens vers beaucoup de pages sur l'Internet à propos de ***iptables*** et ***Netfilter***. Contient aussi une liste de scripts iptables pour différents projets.
- Implementing Quality of Service Policies with DSCP – Un lien sur l'implémentation par Cisco de DSCP. Indique certaines classes utilisées en DSCP.
- IPSEC Howto – Howto IPSEC officiel pour les noyaux 2.6 Linux. Décrit comment IPSEC fonctionne avec les noyaux 2.6, cependant, ce n'est pas l'endroit où vous trouverez comment fonctionne IPSEC avec les noyaux 2.2 et 2.4 Linux. Voir le site FreeS/WAN pour cela.
- FreeS/WAN – Site officiel de FreeS/WAN, une implémentation IPSEC pour les noyaux 2.2 et 2.4 Linux. Ce site contient la documentation et tout le nécessaire à télécharger pour l'implémentation de IPSEC. Cet travail a été discontinu pour plusieurs raisons expliquées sur la page, mais les efforts ont toujours été portés sur la correction de bogues, la documentation et les forums. Pour une implémentation de IPSEC dans les noyaux 2.6 Linux, voir le IPSEC Howto.
- http://www.islandsoft.net/veerapen .html – Excellente discussion sur l'amélioration de ***iptables*** et comment faire des modifications qui vous permettent automatiquement d'ajouter les sites hostiles dans une liste de banissement spéciale dans ***iptables***.
- `/etc/protocols` – Un exemple de fichier de protocoles pris sur une distribution Slackware. Peut être utilisé pour retrouver les numéros de protocole, comme IP, ICMP ou TCP.
- `/etc/services` – Exemple de fichier service pris dans une distribution Slackware. C'est très utile de le lire au moins une fois, en particulier si vous voulez savoir quel protocole tourne sur quel port.
- Internet Assigned Numbers Authority – La IANA est l'organisation responsable de l'attribution de tous les numéros des différents protocoles. Si quelqu'un a un ajout spécifique à faire pour un protocole (par exemple, ajouter une nouvelle option TCP), il doit prendre contact avec IANA, qui assignera les numéros demandés. En d'autres termes, ce site est très important.
- RFC−editor.org – Excellent site pour trouver les documents RFC de façon rapide et ordonnée. Fonctions de recherche de RFC, et information générale sur la communauté RFC (errata, nouvelles,...).
- Internet Engineering Task Force – C'est un des groupes les plus importants pour l'implémentation et la maintenance des standards Internet. Ils sont les seuls à maintenir le dépôt des RFC, et consiste en un ensemble d'entreprises et de particuliers qui travaillent conjointement pour assurer l'interopérabilité de l'Internet.
- Linux Advanced Routing and Traffic Control HOW−TO – Ce site accueille le Linux Advanced Routing and Traffic Control HOWTO. C'est un des plus importants et meilleurs documents concernant le routage avancé sous Linux. maintenu par B.Hubert.
- Paksecured Linux Kernel patches – Site contenant tous les patches du noyau écrits par M.G. Marsh. Parmi d'autres, le patch FTOS est disponible ici.
- ULOGD project page – Page d'accueil du site ULOGD.
- Le Linux Documentation Project est un super site pour la documentation. La plupart des documents les plus importants pour Linux sont disponibles ici.

- Snort – C'est un excellent "système de détection d'intrusion réseau" (NIDS) qui cherche les signatures dans les paquets, et si il voit une signature ressemblant à celle d'une attaque il peut faire diverses actions prédéfinies.
- Tripwire – Tripwire est un excellent outil de sécurité qui peut être utilisé pour les intrusions. Il effectue des sommes de contrôle de tous les fichiers spécifiés, et ensuite prévient l'administrateur.
- Squid – Un des proxies web les plus connus disponibles sur le marché. Il est Open Source, et gratuit. Il peut faire plusieurs tâches de filtrage en amont de votre serveur web, de même que du cache standard pour vos réseaux.
- http://kalamazoolinux.org/presentations/20010417/conntrack.html – Cette présentation contient une excellente explication des modules conntrack et de leur fonction dans Netfilter.
- http://www.docum.org – Excellente information sur `CBQ`, et les commandes ***tc*** et ***ip*** dans Linux. Un des rares sites ayant de l'information sur ces programmes. Maintenu par S. Coene.
- http://lists.samba.org/mailman/listinfo/netfilter – La liste de discussion officielle de Netfilter. Extrêmement utile si vous avez des questions sur des aspects non couverts par ce document.

Et bien sûr le source ***iptables***, la documentation et les personnes qui m'ont aidé.

Annexe F. Remerciements

je voudrais remercier les personnes suivantes pour leur aide à propos de ce document :

- *Fabrice Marie*, pour d'importantes corrections sur mon horrible grammaire et syntaxe. De même que pour la mise à jour de ce didacticiel au format DocBook.
- *Marc Boucher*, pour son aide sur certains aspects de l'utilisation du code.
- *Frode E. Nyboe*, pour les grandes améliorations des règles du `rc.firewall` lors de la réécriture des tables de règles.
- *Chapman Brad*, *Alexander W. Janssen*, tous les deux m'ont permis de me rendre compte de mes erreurs sur le transit des paquets dans les tables NAT et filter.
- *Michiel Brandenburg*, *Myles Uyema*, pour leur aide sur le code de correspondance état.
- *Kent `Artech' Stahre*, pour leur aide sur les graphiques, ainsi que pour la vérification des erreurs dans ce didacticiel.
- *Anders 'DeZENT' Johansson*, pour m'avoir averti du comportement étrange de certains FAIs et de leur usage des réseaux privés sur l'Internet.
- *Jeremy `Spliffy' Smith*, pour son aide à propos des erreurs dans ce document.

Et bien sûr toute personne à qui j'ai demandé des avis sur ce document, je m'excuse de ne pas citer tout le monde.

Annexe G. History

Version 1.2.0 (20 July 2005)
http://iptables–tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Corey Becker, Neil Perrins, Watz and Spanish translation team.

Version 1.1.19 (21 May 2003)
http://iptables–tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Peter van Kampen, Xavier Bartol, Jon Anderson, Thorsten Bremer
and Spanish Translation Team.

Version 1.1.18 (24 Apr 2003)
http://iptables–tutorial.frozentux.net

By: Oskar Andreasson
Contributors: Stuart Clark, Robert P. J. Day, Mark Orenstein and Edmond Shwayri.

Version 1.1.17 (6 Apr 2003)
http://iptables−tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Geraldo Amaral Filho, Ondrej Suchy, Dino Conti, Robert P. J. Day, Velev Dimo, Spencer Rouser, Daveonos, Amanda Hickman, Olle Jonsson and Bengt Aspvall.

Version 1.1.16 (16 Dec 2002)
http://iptables−tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Clemens Schwaighower, Uwe Dippel and Dave Wreski.

Version 1.1.15 (13 Nov 2002)
http://iptables−tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Mark Sonarte, A. Lester Buck, Robert P. J. Day, Togan Muftuoglu, Antony Stone, Matthew F. Barnes and Otto Matejka.

Version 1.1.14 (14 Oct 2002)
http://iptables−tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Carol Anne, Manuel Minzoni, Yves Soun, Miernik, Uwe Dippel, Dave Klipec and Eddy L O Jansson.

Version 1.1.13 (22 Aug 2002)
http://iptables−tutorial.haringstad.com
By: Oskar Andreasson
Contributors: Tons of people reporting bad HTML version.

Version 1.1.12 (19 Aug 2002)
http://www.netfilter.org/tutorial/
By: Oskar Andreasson
Contributors: Peter Schubnell, Stephen J. Lawrence, Uwe Dippel, Bradley Dilger, Vegard Engen, Clifford Kite, Alessandro Oliveira, Tony Earnshaw, Harald Welte, Nick Andrew and Stepan Kasal.

Version 1.1.11 (27 May 2002)
http://www.netfilter.org/tutorial/
By: Oskar Andreasson
Contributors: Steve Hnizdur, Lonni Friedman, Jelle Kalf, Harald Welte, Valentina Barrios and Tony Earnshaw.

Version 1.1.10 (12 April 2002)
http://www.boingworld.com/workshops/linux/iptables−tutorial/
By: Oskar Andreasson
Contributors: Jelle Kalf, Theodore Alexandrov, Paul Corbett, Rodrigo Rubira Branco, Alistair Tonner, Matthew G. Marsh, Uwe Dippel, Evan Nemerson and Marcel J.E. Mol.

Version 1.1.9 (21 March 2002)
http://www.boingworld.com/workshops/linux/iptables−tutorial/

By: Oskar Andreasson
Contributors: Vince Herried, Togan Muftuoglu, Galen Johnson, Kelly Ashe, Janne Johansson, Thomas Smets, Peter Horst, Mitch Landers, Neil Jolly, Jelle Kalf, Jason Lam and Evan Nemerson.

Version 1.1.8 (5 March 2002)
http://www.boingworld.com/workshops/linux/iptables−tutorial/
By: Oskar Andreasson

Version 1.1.7 (4 February 2002)
http://www.boingworld.com/workshops/linux/iptables−tutorial/
By: Oskar Andreasson
Contributors: Parimi Ravi, Phil Schultz, Steven McClintoc, Bill Dossett, Dave Wreski, Erik Sjölund, Adam Mansbridge, Vasoo Veerapen, Aladdin and Rusty Russell.

Version 1.1.6 (7 December 2001)
http://people.unix−fu.org/andreasson/
By: Oskar Andreasson
Contributors: Jim Ramsey, Phil Schultz, Göran Båge, Doug Monroe, Jasper Aikema, Kurt Lieber, Chris Tallon, Chris Martin, Jonas Pasche, Jan Labanowski, Rodrigo R. Branco, Jacco van Koll and Dave Wreski.

Version 1.1.5 (14 November 2001)
http://people.unix−fu.org/andreasson/
By: Oskar Andreasson
Contributors: Fabrice Marie, Merijn Schering and Kurt Lieber.

Version 1.1.4 (6 November 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson
Contributors: Stig W. Jensen, Steve Hnizdur, Chris Pluta and Kurt Lieber.

Version 1.1.3 (9 October 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson
Contributors: Joni Chu, N.Emile Akabi−Davis and Jelle Kalf.

Version 1.1.2 (29 September 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson

Version 1.1.1 (26 September 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson
Contributors: Dave Richardson.

Version 1.1.0 (15 September 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson

Version 1.0.9 (9 September 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson

Version 1.0.8 (7 September 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson

Version 1.0.7 (23 August 2001)
http://people.unix−fu.org/andreasson
By: Oskar Andreasson
Contributors: Fabrice Marie.

Version 1.0.6
http://people.unix−fu.org/andreasson
By: Oskar Andreasson

Version 1.0.5
http://people.unix−fu.org/andreasson
By: Oskar Andreasson
Contributors: Fabrice Marie.

Annexe H. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

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

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

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

1. APPLICABILITY AND DEFINITIONS

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

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

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

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

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

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

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

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

2. VERBATIM COPYING

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

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

3. COPYING IN QUANTITY

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

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

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

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

4. MODIFICATIONS

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

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

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

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

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

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

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

5. COMBINING DOCUMENTS

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

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

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

6. COLLECTIONS OF DOCUMENTS

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

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

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the

Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self−contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

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

8. TRANSLATION

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

9. TERMINATION

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

10. FUTURE REVISIONS OF THIS LICENSE

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

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

. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Annexe I. GNU General Public License

Version 2, June 1991

> Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software−−to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

1. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.
2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    A. Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    B. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    C. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

    The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

    If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other

circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.
11. NO WARRANTY

    BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

2. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse−clicks or menu items−−whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this

License.

Annexe J. Example scripts code–base

J.1. Example rc.firewall script

```
#!/bin/sh
#
# rc.firewall - Initial SIMPLE IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

###############################################################################
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#
```

```
#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

##############################################################################
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
```

```
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

#########################################################################
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
```

```
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 53 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 123 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 2074 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 4000 -j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught properly
# otherwise.
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j ACCEPT

#
```

```
# Rules for incoming packets from the internet.
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udp_packets
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall, you may
# also get flooded by Multicasts. We drop them so we do not get flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT
```

```
#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#

$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source $INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
```

```
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

J.2. Example rc.DMZ.firewall script

```
#!/bin/sh
#
# rc.DMZ.firewall - DMZ IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

###############################################################################
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.152"
HTTP_IP="194.236.50.153"
DNS_IP="194.236.50.154"
INET_IFACE="eth0"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#
```

```
LAN_IP="192.168.0.1"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#

DMZ_HTTP_IP="192.168.1.2"
DMZ_DNS_IP="192.168.1.3"
DMZ_IP="192.168.1.1"
DMZ_IFACE="eth2"

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
#
# 2. Module loading.
#

#
# Needed to initially load modules
#
/sbin/depmod -a



#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc
```

```
##############################################################################
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

##############################################################################
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP
```

```
#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# ICMP rules
#

# Changed rules totally
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Packets from the Internet to this box
#

$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# Packets from LAN, DMZ or LOCALHOST
#

#
# From DMZ Interface to DMZ firewall IP
#

$IPTABLES -A INPUT -p ALL -i $DMZ_IFACE -d $DMZ_IP -j ACCEPT

#
# From LAN Interface to LAN firewall IP
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -d $LAN_IP -j ACCEPT

#
# From Localhost interface to Localhost IP's
#

$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught properly
# otherwise.
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j ACCEPT

#
# All established and related packets incoming from the internet to the
```

```
# firewall
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state ESTABLISHED,RELATED \
-j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A INPUT -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A INPUT -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# If you have a Microsoft Network on the outside of your firewall, you may
# also get flooded by Multicasts. We drop them so we do not get flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets


#
# DMZ section
#
# General rules
#

$IPTABLES -A FORWARD -i $DMZ_IFACE -o $INET_IFACE -j ACCEPT
$IPTABLES -A FORWARD -i $INET_IFACE -o $DMZ_IFACE -m state \
--state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $LAN_IFACE -o $DMZ_IFACE -j ACCEPT
$IPTABLES -A FORWARD -i $DMZ_IFACE -o $LAN_IFACE -m state \
--state ESTABLISHED,RELATED -j ACCEPT

#
# HTTP server
#

$IPTABLES -A FORWARD -p TCP -i $INET_IFACE -o $DMZ_IFACE -d $DMZ_HTTP_IP \
```

```
--dport 80 -j allowed
$IPTABLES -A FORWARD -p ICMP -i $INET_IFACE -o $DMZ_IFACE -d $DMZ_HTTP_IP \
-j icmp_packets

#
# DNS server
#

$IPTABLES -A FORWARD -p TCP -i $INET_IFACE -o $DMZ_IFACE -d $DMZ_DNS_IP \
--dport 53 -j allowed
$IPTABLES -A FORWARD -p UDP -i $INET_IFACE -o $DMZ_IFACE -d $DMZ_DNS_IP \
--dport 53 -j ACCEPT
$IPTABLES -A FORWARD -p ICMP -i $INET_IFACE -o $DMZ_IFACE -d $DMZ_DNS_IP \
-j icmp_packets

#
# LAN section
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#
```

```
#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $HTTP_IP --dport 80 \
-j DNAT --to-destination $DMZ_HTTP_IP
$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $DNS_IP --dport 53 \
-j DNAT --to-destination $DMZ_DNS_IP
$IPTABLES -t nat -A PREROUTING -p UDP -i $INET_IFACE -d $DNS_IP --dport 53 \
-j DNAT --to-destination $DMZ_DNS_IP

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#

$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source $INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

J.3. Example rc.UTIN.firewall script

```
#!/bin/sh
#
# rc.UTIN.firewall - UTIN Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

##############################################################################
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"
```

```
#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

##############################################################################
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

```
#########################################################################
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed
```

```
#
# UDP ports
#

#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 123 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 2074 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 4000 -j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Rules for incoming packets from anywhere.
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -j tcp_packets
$IPTABLES -A INPUT -p UDP -j udp_packets
$IPTABLES -A INPUT -p ICMP -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall, you may
# also get flooded by Multicasts. We drop them so we do not get flooded by
# logs
#
```

```
#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -p tcp --dport 21 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 80 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 110 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
```

```
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#

$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source $INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

J.4. Example rc.DHCP.firewall script

```
#!/bin/sh
#
# rc.DHCP.firewall - DHCP IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

###############################################################################
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IFACE="eth0"

#
# 1.1.1 DHCP
#

#
# Information pertaining to DHCP over the Internet, if needed.
#
# Set DHCP variable to no if you don't get IP from DHCP. If you get DHCP
# over the Internet set this variable to yes, and set up the proper IP
# address for the DHCP server in the DHCP_SERVER variable.
#

DHCP="no"
DHCP_SERVER="195.22.90.65"

#
# 1.1.2 PPPoE
#

# Configuration options pertaining to PPPoE.
#
# If you have problem with your PPPoE connection, such as large mails not
# getting through while small mail get through properly etc, you may set
# this option to "yes" which may fix the problem. This option will set a
# rule in the PREROUTING chain of the mangle table which will clamp
# (resize) all routed packets to PMTU (Path Maximum Transmit Unit).
#
# Note that it is better to set this up in the PPPoE package itself, since
# the PPPoE configuration option will give less overhead.
#
```

```
PPPOE_PMTU="no"

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
#

/sbin/modprobe ip_conntrack
/sbin/modprobe ip_tables
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_MASQUERADE

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
```

```
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

###############################################################################
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

###############################################################################
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#
```

```
$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
if [ $DHCP == "yes" ] ; then
 $IPTABLES -A udp_packets -p UDP -s $DHCP_SERVER --sport 67 \
 --dport 68 -j ACCEPT
fi

#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 123 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 2074 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 4000 -j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
```

```
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught properly
# otherwise.
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j ACCEPT

#
# Rules for incoming packets from the internet.
#

$IPTABLES -A INPUT -p ALL -i $INET_IFACE -m state --state ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udp_packets
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall, you may
# also get flooded by Multicasts. We drop them so we do not get flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
```

```
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -o $INET_IFACE -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

if [ $PPPOE_PMTU == "yes" ] ; then
 $IPTABLES -t nat -A POSTROUTING -p tcp --tcp-flags SYN,RST SYN \
 -j TCPMSS --clamp-mss-to-pmtu
fi
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j MASQUERADE

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
```

```
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

J.5. Example rc.flush–iptables script

```
#!/bin/sh
#
# rc.flush-iptables - Resets iptables to default values.
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA

#
# Configurations
#
IPTABLES="/usr/sbin/iptables"

#
# reset the default policies in the filter table.
#
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
```

```
$IPTABLES -P OUTPUT ACCEPT

#
# reset the default policies in the nat table.
#
$IPTABLES -t nat -P PREROUTING ACCEPT
$IPTABLES -t nat -P POSTROUTING ACCEPT
$IPTABLES -t nat -P OUTPUT ACCEPT

#
# reset the default policies in the mangle table.
#
$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P POSTROUTING ACCEPT
$IPTABLES -t mangle -P INPUT ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT
$IPTABLES -t mangle -P FORWARD ACCEPT

#
# flush all the rules in the filter and nat tables.
#
$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
#
# erase all chains that's not default in filter and nat table.
#
$IPTABLES -X
$IPTABLES -t nat -X
$IPTABLES -t mangle -X
```

J.6. Example rc.test–iptables script

```
#!/bin/bash
#
# rc.test-iptables - test script for iptables chains and tables.
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

#
# Filter table, all chains
#
iptables -t filter -A INPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter INPUT:"
iptables -t filter -A INPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter INPUT:"
```

```
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter FORWARD:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter FORWARD:"

#
# NAT table, all chains except OUTPUT which don't work.
#
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat OUTPUT:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat OUTPUT:"

#
# Mangle table, all chains
#
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle PREROUTING:"
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle PREROUTING:"
iptables -t mangle -I FORWARD 1 -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle FORWARD:"
iptables -t mangle -I FORWARD 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle FORWARD:"
iptables -t mangle -I INPUT 1 -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle INPUT:"
iptables -t mangle -I INPUT 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle INPUT:"
iptables -t mangle -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle OUTPUT:"
iptables -t mangle -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle OUTPUT:"
iptables -t mangle -I POSTROUTING 1 -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle POSTROUTING:"
iptables -t mangle -I POSTROUTING 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle POSTROUTING:"
```