Universiteit Gent
Faculteit Toegepaste Wetenschappen
Vakgroep Electronica en Informatiesystemen
Voorzitter: Prof. dr. ir. J. Van Campenhout

Firewalls: Netfilter

door Bart De Schuymer

Promotor : prof. dr. ir. K. De Bosschere
Thesisbegeleiders : ir. R. Blomme en dr. ir. M. Ronsse

Afstudeerwerk ingediend tot het behalen van de graad van licentiaat in de informatica, optie: informatie- en communicatietechnologie

Academiejaar 2000-2001

Firewalls: netfilter

door
Bart De Schuymer

Afstudeerwerk ingediend tot het behalen van de graad van licentiaat in de informatica, optie: informatie- en communicatietechnologie.

Academiejaar 2000 - 2001
Universiteit Gent
Faculteit Toegepaste Wetenschappen

Promotor: prof. dr. ir. K. De Bosschere

Samenvatting

Het werk verricht voor deze thesis kan opgesplitst worden in twee grote delen. Eerst werd de broncode van de bestaande firewall-mogelijkheden voor Linux bestudeerd. Hierdoor werd een inzicht verworven in de werking van deze firewall en deze werking zal dan ook uitgelegd worden. Daarna werd een uitbreiding van de firewall-mogelijkheden geschreven. Naast het beschrijven van de resultaten zullen de algoritmes en denkwijzen die aan de basis liggen van de implementatie worden besproken. Dit zal weliswaar op een abstract niveau gebeuren, zonder het bestuderen van programmacode.

Dankwoord

Met dank aan:

Prof. dr. ir. K. De Bosschere, promotor van deze thesis,
Ir. R. Blomme en dr. ir. M. Ronsse voor de expertise,
De vakgroep ELIS voor het gebruik van de computerfaciliteiten,
Lennert Buytenhek, Paul Russel en Harald Welte voor het, af en toe, verschaffen van de nodige inzichten.

Toelating tot bruikleen

De auteur geeft de toelating dit afstudeerwerk voor consultatie beschikbaar te stellen en delen van het afstudeerwerk te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit dit afstudeerwerk.

Gent, 5 juni 2001

Bart De Schuymer

Lijst van figuren

Woordenlijst

Nederlands	**Engels**
aanpassen van pakketten	packet mangling
aanpassingentabel	mangle table
aansturingssoftware	driver
bediener	server
beheerder	maintainer
bijhouden van connecties	connection tracking
broncodeboom	source tree
brug	bridge
databank	database
doel	target
internetdienstenverlener	provider
elektronisch handboek	manual pages
elektronische post	e-mail
fitting	socket
gastheer	host
gebruikerscontext	user context
gebruikersprogramma	userspace program
gelijkenis(functie)	match
haak	hook
inleiding	preamble
kern	kernel
kernaanpassing	kernel patch
kernbeeld	kernel image
ketting	chain

kop	header
leesslot	readlock
map	directory
maskering	masquerading
netwerkadresvertaling (nav)	network address translation (nat)
omroepen	to broadcast
onderbreking	interrupt
onderbrekingsaanvraag	interrupt request (IRQ)
onderbrekingsafhandelaar	interrupt handler
opstartpartitie	boot partition
overloop	overflow
pakketfilter	packet filter
patstelling	deadlock
portaalcomputer	gateway
regel	rule
routeerder	router
routering	routing
schakelaar	switch
schrijfslot	writelock
slot	lock
slotvergrendeling	locking
software-onderbreking (SO)	soft-IRQ, softirq
spinslot	spinlock
standaardbeleid	policy
stroman	proxy
tabel	table
transparante firewall op een brug	transparent bridging firewall
van fouten zuiveren	to debug
webpagina	webpage
wekker	timer
werkverdeler	scheduler
wijzer	pointer

Inhoudsopgave

Leegte

hehe

Hoofdstuk 1

Inleiding

Vooraleer over te gaan tot de probleemschetsing en doelstelling van deze thesis zullen de verschillende begrippen die nodig zijn om de doelstelling te beschrijven worden geïntroduceerd. Van de lezer wordt een elementaire kennis van netwerken verwacht.
Voor een meer gedetailleerde beschrijving van netwerken en netwerkbeveiliging verwijzen we naar [2].

1.1 Het OSI-model

In 1977 werd het *Open Systems Interconnection Reference Model* (OSI-model) ontwikkeld door het ISO (*International Standards Organization*). Dit is een model voor netwerkcommunicatie. Het doel was niet een favoriete specifieke methode of communicatie aan te raden, maar wel het ontwikkelen van een set van richtlijnen die ervoor moet zorgen dat producten van verschillende bedrijven met elkaar kunnen samenwerken.
Het OSI-model bestaat uit zeven lagen (zie figuur 1.1). Elke laag beschrijft hoe haar deel van het communicatieproces moet werken. Wanneer een bedrijf een product maakt dat de specificaties van een laag volgt, zal het kunnen communiceren met producten van andere bedrijven die werken op een aangrenzende laag (en ook het OSI-model voor deze laag volgen).

Figuur 1.1: Het OSI-model

Bespreking van de verschillende lagen

- **Fysische laag**: Beschrijft de specificaties van de transportmedia, connectoren en signaalpulsen.
- **Datalinklaag**: Beschrijft de specificaties voor de topologie van en communicatie tussen lokale systemen. De eenheid van data die hier wordt getransporteerd bestaat uit een frame, wat een gestructureerde hoeveelheid bytes is. Ethernet is een goed voorbeeld van een datalinklaagspecificatie. Bruggen en schakelaars (zie punt 1.4) worden beschouwd als datalinktoestellen aangezien ze zich bewust zijn van frames, beide gebruiken ze specifieke informatie uit de framekop om het netwerkverkeer te regelen.
- **Netwerklaag**: Beschrijft hoe systemen, die zich op verschillende netwerken bevinden, elkaar vinden.
- **Transportlaag**: Houdt zich bezig met de manipulatie van de data. In deze laag wordt de data opgesplitst in pakketjes zodat deze pakketjes niet te groot zijn om in een frame te passen.
- **Sessielaag**: Houdt zich bezig met het opzetten en onderhouden van een verbinding tussen twee of meerdere systemen. Wanneer we bv. toegang tot een systeem trachten te krijgen m.b.v. een webbrowser zullen de sessielagen van beide systemen samenwerken om ervoor te zorgen dat we html-pagina's ontvangen en geen elektronische post.

- **Presentatielaag**: Zorgt ervoor dat de data wordt ontvangen in een formaat dat bruikbaar is voor programma's die zich op het systeem bevinden. Deze laag is bv. verantwoordelijk voor het encrypteren en decrypteren van data die verzonden wordt over een netwerk.

- **Applicatielaag**: Bestaat uit de programma's die gebruik maken van het netwerk.

In deze thesis staan de datalinklaag en de netwerklaag centraal.

1.2 Firewall

Een firewall is een softwarepakket dat zorgt voor de beveiliging van een computernetwerk tegenover invloeden van buitenaf. Deze software zal dus draaien op een computer die zich bevindt tussen het te beveiligen netwerk en de buitenwereld (bv. het internet of een ander netwerk). Het netwerkverkeer bestaat uit pakketjes die informatie bevatten. De firewallsoftware zal de pakketjes die willen passeren onderscheppen en aan een controle onderwerpen. Deze controle varieert van een eenvoudige controle op dit pakketje tot meer gesofisticeerde controles zoals het bijhouden van connecties. Hierbij vallen twee types firewalls te onderscheiden:

- **De stroman-firewall**:
  In een netwerk dat met een stroman werkt zullen alle aanvragen voor connecties naar de buitenwereld worden verstuurd naar de stroman. De computers binnen het netwerk (gastheren) kennen dus het adres van deze stroman. De stroman bekijkt de binnenkomende pakketten en beslist over welk type connectie het gaat (bv. http, ftp, telnet, ...). Indien het pakketje bv. het http-protocol gebruikt zal de stromansoftware dit pakketje doorgeven aan een speciaal programma dat zich enkel met het http-protocol bezighoudt. Dit programma zal beslissen of het pakketje zal worden doorgestuurd, indien nodig kan het bepaalde gegevens van dit pakketje wijzigen. Bron en bestemming zullen nooit rechtstreeks contact hebben met elkaar: de interne gastheer zowel als

de bestemming communiceren met elkaar via de stroman (er wordt dus met twee connecties gewerkt). Aangezien de protocolspecifieke software deze protocols volledig begrijpt kan men protocolspecifieke veiligheid aanbrengen. Er kan dus bv. een firewall op het niveau van het http-protocol worden gemaakt.

- **De pakketfilter**:
  Pakketfilters en stromanfirewalls verschillen sterk van elkaar. Bij pakketfilters praten bron en bestemming rechtstreeks met elkaar, ze weten niet dat een pakketfilter hun verkeer modereert. Wanneer een pakketje verstuurd wordt van een interne gastheer naar een externe bestemming zal de pakketfilter dit onderscheppen. Op dit pakketje zullen controles worden uitgevoerd om te bepalen of het wel mag doorgestuurd worden. Eenmaal beslist is dat het pakketje mag doorgestuurd worden zal de pakketfilter dit versturen naar de bestemming.
  Nog een verschilpunt is dat de pakketfilters aan routering doen en stroman-firewalls niet (zie volgend punt 1.3).

Wie meer over firewalls wil weten wordt verwezen naar [3] en [4].

1.3 Routeerder en portaalcomputer

Een routeerder is een toestel (of programma) dat het volgende netwerkpunt bepaalt waarnaar een pakket moet worden doorgezonden om uiteindelijk op zijn bestemming te geraken. Routering is een functie die geassocieerd wordt met de netwerklaag.
Een portaalcomputer is een toestel dat als een ingang tot een netwerk dient. Via deze toestellen komen we dus in andere netwerken terecht, het zijn connectoren van verschillende netwerken. De plaats waar routeerders of stromannen worden aangebracht zal op zulke portaalcomputers zijn.

1.4 Brug/Schakelaar

Een brug/schakelaar wordt gebruikt indien we netwerken (die hetzelfde datalinkprotocol gebruiken) met elkaar willen verbinden op een intelligente manier en zodat zo weinig mogelijk configuraties moeten worden gewijzigd. Een brug is een schakelaar tussen twee netwerken. Een schakelaar kan dus meer dan twee netwerken verbinden. In het Linux-jargon spreekt men echter altijd van een brug, alhoewel de Linux-brugcode meer dan twee netwerken kan verbinden. De brug bekijkt alle pakketjes en zal pakketjes die van netwerk A komen en als bestemming netwerk B hebben op netwerk B versturen. Pakketjes waarvan zowel bron als bestemming in netwerk A liggen worden niet in het andere netwerk verstuurd. Zo vermijdt men dus overtollig verkeer in de netwerken en moet niet met een portaalcomputer worden gewerkt. Om zijn functie te vervullen houdt de brug een tabel bij waarin voor een bepaalde bestemming (een datalinklaagadres) wordt aangegeven in welk netwerk pakketten met dit adres als bestemming moeten worden verstuurd. Indien de brug niet weet in welk netwerk de bestemming zich bevindt, zal het pakket in alle netwerken worden verstuurd, behalve in het netwerk waarin de bron zich bevindt (anders zou dit pakket tweemaal in dit netwerk zijn verstuurd). De gastheren in de verschillende netwerken zijn zich niet bewust van het bestaan van deze brug. Een brug verschilt van een routeerder omdat een brug op de datalinklaag actief is terwijl een routeerder op de netwerklaag actief is. Een brug gebruikt bv. ethernet (MAC-adressen) terwijl een routeerder bv. het ip-protocol gebruikt.
Zie [10] voor de webpagina van de linuxbrug.

1.5 Transparente firewall op een brug

Indien we een brug met een pakketfilter combineren verkrijgen we een transparente firewall. Deze opstelling verschilt van de normale pakketfilter aangezien de routering bij de normale pakketfilter op de netwerklaag gebeurt i.p.v. op de datalinklaag. De firewall is in beide gevallen echter dezelfde.
We noemen deze combinatie transparant omdat een aanvaller niet kan we-

ten op welke computer de firewall zich bevindt. Een normale pakketfilter bevindt zich, zoals gezegd, op een portaalcomputer. De aanvaller weet het adres van de portaalcomputer en kan makkelijk raden dat de firewall zich hier zal bevinden. De transparante firewall bevindt zich op een brug en de aanvaller kan niet raden welke computer als brug functioneert.

Hoofdstuk 2

Probleemschetsing en doelstelling

Om de veiligheid van het ELIS-netwerk nog te verbeteren onderzoeken we het uitbreiden van de transparante firewallmogelijkheden voor een brug in Linux.

2.1 Het originele netwerk en het vernieuwde netwerk

Voor een meer volledige beschrijving verwijzen we naar [1]. In het kort komt het erop neer dat ELIS bestaat uit twee subnetten (157.193.67.0 en 157.193.83.0) en men voor deze twee netten een firewall wou installeren zonder dat de ip-configuratie van de gastheren in ELISnet daarvoor dienden veranderd te worden. Hiervoor had men twee mogelijke oplossingen: ofwel een transparente firewall op een brug, ofwel het installeren van een portaalcomputer die als firewall en routeerder dienst doet. Deze tweede oplossing was praktisch gezien moeilijker dan de eerste oplossing. Een transparante firewall is ook veiliger, aangezien er niet kan achterhaald worden welke machine precies de firewall-functie uitvoert (zie punt 1.4). Daarom werd geopteerd voor de eerste oplossing.

Als besturingssysteem voor deze transparante firewall op een brug werd gekozen voor Linux, draaiende op een kern met versienummer 2.2.x.

2.2 Doelstelling

De kernen met versienummers 2.2.x gebruiken een ander systeem voor pakketfiltering dan de kernen met versienummers 2.4.x, die tegenwoordig de meest recente kernen zijn. Bij de aanvang van de thesis werd gewenst deze nieuwe versie van de pakketfilter te bestuderen en indien nodig belangrijke aanpassingen aan te brengen. Na een grondige studie bleek dat deze nieuwe code zeer goed in elkaar zit en dat alle belangrijke functionaliteiten reeds waren geïmplementeerd. Deze filter controleert echter enkel pakketten voor de protocollen IPv4 en IPv6, welke zich in het OSI-model bevinden in de netwerklaag. Er werd vastgesteld dat de brugcode alle pakketten van andere protocollen doorlaat. Na grondig onderzoek werd geen code voor de Linux-kern gevonden die hieraan kan verhelpen en werd besloten deze code zelf te schrijven. Deze uitbreiding is de uiteindelijke doelstelling van de thesis.

2.3 Uiteindelijke implementatie

Er werd dus besloten om een filter te schrijven die op basis van het gebruikte protocol van het pakketje beslissingen kan nemen. Voorbeelden van zulke protocollen zijn: IPv4, IPv6, ARP, RARP, NetBEUI, IPX. Deze implementatie kreeg de toepasselijke naam brepf, wat staat voor *brug ethernet-protocolfilter*.

2.4 Indeling van de thesis

Hoofdstukken 3 t.e.m. 7 beschrijven de bestaande infrastructuur van de Linux-kern. Een groot deel van het werk voor deze thesis bestond in het bestuderen van de code van deze infrastructuur. Hoe deze code werkt en wat ze doet wordt dus grondig bekeken. We wensen op te merken dat de studie van deze code niet triviaal was. Gedetailleerde literatuur over de in deze

thesis besproken infrastructuren, bestaat nl. niet. Er diende dus diep in de C-code gegraven te worden om de werkwijze ervan te achterhalen.
In hoofdstuk 8 wordt de zelf geschreven code onder de loepe genomen. Deze code omvat de volledige brepf-code (zowel de kerncode als het gebruikersprogramma). Aan de code die in hoofdstug 9 wordt besproken werd meegeholpen, alhoewel het meeste werk door L. Buytenhek werd gedaan. Voor de broncode verwijzen we naar de bijhorende cd-rom.
In hoofdstuk 10 worden de behaalde resultaten beschreven.

Hoofdstuk 3

Waarom het bestaande systeem veranderen?

3.1 Kern 2.4.x versus 2.2.x

De huidige firewall in het ELIS-netwerk draait, zoals gezegd, op een Linux-machine met een kernversie 2.2.x met ipchains als pakketfilter. Tussen de versies 2.2.x en 2.4.x is heel wat veranderd, bv. betere smp-ondersteuning (symmetrische multi-processing), software-onderbrekingen (zie punt 8.4.5) en betere slotvergrendeling. Ook de pakketfiltercode is heel wat veranderd, ze is zelfs volledig van nul herschreven (zie [5]). Dit werd gedaan door Paul 'Rusty' Russel, de ip-firewall-beheerder voor Linux. Dit heeft hem 12 maanden tijd gevergd en hiervoor werd hij betaald door een firewall-bedrijf. Zelf legt hij het als volgt uit: 'Watchgard, an excellent firewall company who sell the really nice plug-in Firebox, offered to pay me to do nothing, so I could spend all my time writing this'. De ipchains-code heeft enkele grote tekortkomingen die door de nieuwe iptables-code werden verholpen:

- Er is geen mogelijkheid om pakketten door te geven naar gebruikersprogramma's. Daardoor moet alles in de kern gebeuren. Hoe meer door gebruikersprogramma's kan gedaan worden (zonder dat de prestaties te sterk afnemen), hoe beter. Wanneer de kerncode niet werkt, crasht de volledige kern meestal, terwijl het crashen van een gebruikers-

programma de kern niet doet crashen. Een ander belangrijk voordeel is dat hierdoor uitbreidingen kunnen geschreven worden waarvoor de kern niet hoeft te worden gehercompileerd. Daarnaast maakt het de kern ook kleiner en is het gewoon veel makkelijker om een gebruikersprogramma te schrijven.

- Ip-maskering (zie punt 4.3) hangt volledig vast aan pakketfiltering, terwijl de twee onafhankelijk van elkaar zouden moeten werken.
- De ipchains-code is zeer moeilijk uitbreidbaar omdat ze niet modulair is opgebouwd.
- Connecties kunnen niet worden bijgehouden. Dit mogelijk maken (zie punt 7.1.7) in de ipchains-kern zou zeer lastig zijn aangezien de structuur niet modulair is. De code zou, net zoals de ip-maskeringscode, verweven zijn in de filteringcode.

Net doordat de iptables-code speciaal ontworpen is om makkelijk uitbreidbaar te zijn, is er veel meer functionaliteit aanwezig en zullen nieuwe functionaliteiten sneller worden toegevoegd. De modulaire aanpak maakt de code ook veel mooier en makkelijker leesbaar en begrijpbaar, wat zorgt voor meer betrouwbare code. Een groot voordeel van het gebruik van de kern 2.4.x is de netfilter-architectuur. Dit is een architectuur die door alle firewalls en andere software die pakketten willen bestuderen kan worden gebruikt. Iptables gebruikt deze architectuur en van deze architectuur is ook dankbaar gebruik gemaakt bij het ontwerp van de brepf-software. Voor een beschrijving van deze architectuur verwijzen we naar hoofdstuk 6.

3.2 Waarom een ethernet-protocolfilter op een brug?

Een brug dient om aan routering te doen (transparante routering). Enkel het bron- en bestemmingsadres (MAC-adres bij ethernet) van een pakketje is belangrijk voor een brug. Om pakketfiltering toe te laten op de brug zal

de kern moeten gehercompileerd worden nadat een programmacorrectie werd toegepast op de kern. Dit omdat pakketfiltering op bruggen niet standaard in de kern zit.
Indien we aan pakketfiltering met iptables doen op deze brug, zullen de pakketjes die binnenkomen en die het ip-protocol gebruiken, worden verstuurd naar de pakketfilteringcode. Deze code beslist over het lot van het pakketje. Andere pakketjes zullen echter gewoon de routeringscode van de brug doorlopen (zonder filtering) en zullen worden doorgestuurd. Dit komt omdat dit net de functie van een brug is en we extra stappen moeten ondernemen om voor filtering te zorgen. Deze extra stap is in ons geval het schrijven van een ethernet-protocolfilter. Deze filter is nuttig voor volgende zaken:

- **Veiligheid en zekerheid**:
  Stel dat we twee netwerken met elkaar wensen te verbinden via een brug. Het eerste netwerk is een windows NT netwerk (met IBM PC's) dat NetBEUI (*NetBIOS Extended User Interface*) gebruikt, het tweede netwerk is een Unix-netwerk. Een - misschien paranoïde, misschien goede? - netwerkbeheerder zal honderd procent zeker willen stellen dat bv. enkel ARP- (*address resolution protocol*) en IPv4- (*internet protocol versie 4*) pakketten tussen de twee netwerken worden uitgewisseld. Unix gebruikt nl. het NetBEUI-protocol niet, dus horen zulke pakketjes daar niet thuis.

- **Monitoring**:
  In de brepf-software zijn tellers voorzien en ook een databank die bijhoudt welke protocollen op welke interfaces toekomen en vertrekken. De tellers werken als volgt: telkens een pakketje voldoet aan de voorwaarden van een regel van een ketting (zie punt 4.1) zal de corresponderende teller van die regel worden verhoogd. Deze teller kan dan afgedrukt worden op het scherm. Zowel de databank als de tellers kunnen uitgeschakeld worden.

Hoofdstuk 4

Iptables

Ip-pakketfiltering staat los van bruggen. Het is zelfs zo dat (zoals gezegd) om pakketfiltering te kunnen doen op een brug, er een speciale kernaanpassing moet worden uitgevoerd op de standaard Linux-kern. We vergeten nu dus even bruggen en concentreren ons op iptables. Iptables bestaat niet alleen uit een pakketfilter, het kan nl. ook aan nav (netwerkadresvertaling) en *mangling* (het veranderen van pakketjes) doen en heeft hierdoor in totaal drie tabellen, zie verder. Bij ip-pakketfiltering staat de ip-kop centraal (zie figuur 4.1). Voor praktische voorbeelden verwijzen we naar het elektronische handboek van iptables en naar [11]. Hier concentreren we ons op de huidige mogelijkheden van iptables.

<table>
<tr><td>4-bit
version</td><td>4-bit
HL</td><td>8-bit
TOS</td><td colspan="2">16-bit total length of packet</td></tr>
<tr><td colspan="3">16 bit identification</td><td>3-bit flags</td><td>13-bit fragment offset</td></tr>
<tr><td colspan="2">8-bit time-to-live</td><td>8-bit protocol</td><td colspan="2">16-bit header checksum</td></tr>
<tr><td colspan="5">32-bit source IP address</td></tr>
<tr><td colspan="5">32-bit destination IP address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
</table>

Figuur 4.1: Ip-kop

4.1 Tabellen, kettingen en regels

Zoals gezegd bestaat iptables uit drie tabellen. Elke tabel bestaat op haar beurt uit een aantal kettingen. Een ketting is een aaneenschakeling van regels. Een regel van een ketting heeft deze vorm: 'als een pakketje er zo uitziet doe dan dit'. Wanneer een pakketje gelijkenis vertoont met een regel zal dus een bepaalde actie (het doel) worden uitgevoerd. Tabellen onderscheiden faciliteiten van elkaar die totaal verschillen, terwijl de kettingen te maken hebben met de plaats waar het pakket zich bevindt (is het net binnengekomen, is het klaar om doorgestuurd te worden, ...).

Wanneer een ip-pakketje door een bepaalde ketting gaat, zal het pakketje worden onderworpen aan een controle door de regels. Indien het pakketje en de regel bij elkaar passen (gelijkenis vertonen) beslist de regel wat met het pakketje zal gebeuren. Indien de regel besluit dat het pakketje niet mag doorgaan, zal het pakketje zijn weg in het netwerk niet mogen vervolgen, de ketting wordt verlaten. Indien de regel besluit dat het pakketje wel mag doorgaan, zal het pakketje de ketting verlaten en zijn weg vervolgen. Daarnaast kan de regel ook beslissen dat een andere actie moet worden uitgevoerd: er kan bv. naar een door de gebruiker zelf gedefinieerde ketting worden gesprongen, of het pakketje kan veranderd worden. Na deze acties wordt de ketting niet noodzakelijk verlaten.

Er kunnen dus zelf kettingen aangemaakt worden, naar deze kettingen wordt gesprongen via andere kettingen. Indien de regels niet hebben beslist wat er met het pakketje moet gebeuren, zal het standaardbeleid van de ketting worden gevolgd.

Voor een beter inzicht moet geweten zijn dat een standaardketting zal ingehaakt zijn op een bepaalde haak (zie figuur 4.3). Zo zal de FORWARD-ketting van de pakketfiltertabel ingehaakt zijn op de FORWARD-haak. De ip-pakketjes die deze FORWARD-haak passeren zullen dus in de FORWARD-ketting worden onderzocht. Haken komen uitgebreider aan bod in hoofdstuk 6.

4.2 De pakketfiltertabel

4.2.1 Wat is een pakketfilter en wat is het nut ervan?

Een pakketfilter is een stuk software dat naar de kop van een pakketje kijkt en met deze informatie over het lot van het hele pakketje beslist. Deze software wordt om drie redenen gebruikt:

- **Controle**:
  Aangezien het bron- en doeladres van een ip-pakket in de kop van het pakketje zitten, kunnen we ervoor zorgen dat bv. bepaalde internetpagina's niet toegankelijk zijn van binnen het netwerk.

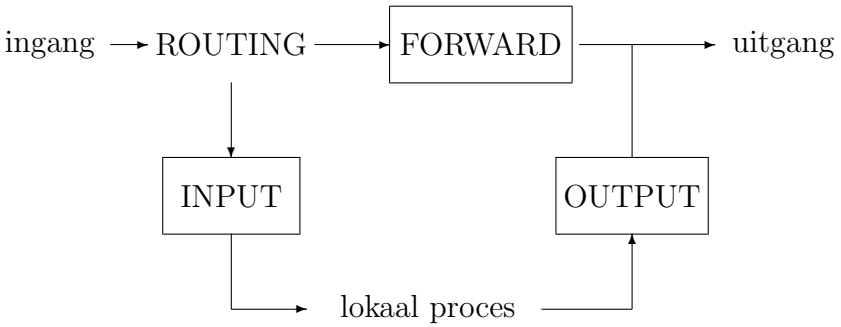
- **Veiligheid**:
  Het kan bv. aangewezen zijn dat men van buiten het netwerk telnet niet kan gebruiken om in te loggen op een computer in het netwerk. Telnet is nl. een onveilige inlogmanier aangezien het paswoord ongeëncrypteerd wordt verstuurd.

- **Oplettendheid**:
  Een slecht geconfigureerde lokale machine kan bv. ongecontroleerd pakketten het internet opsturen. Het is handig als iptables iets laat weten indien dit gebeurt, misschien kan het probleem zo sneller opgelost worden.

4.2.2 Hoe doorlopen ip-pakketten de filter?

De drie standaardkettingen zijn INPUT, OUTPUT en FORWARD (zie figuur 4.2). Deze kettingen zitten ingehaakt op een haak (zie punt 6.2.1). Voor ip bestaan er 5 haken (zie figuur 4.3), waarvan de pakketfilter er dus 3 gebruikt.

Vooraleer een pakket de firewallcode passeert zal dit pakket de routeringscode passeren. Hierin wordt beslist naar waar het pakket moet worden verstuurd. Indien het pakket bestemd is voor de computer waarop de pakketfilter draait, komt het in de INPUT-ketting terecht. Indien daarentegen het



Figuur 4.2: Pakketfilterhaken

pakket een andere bestemming heeft, komt het in de FORWARD-ketting terecht. Pakketten met als oorsprong de machine waarop de pakketfilter draait kunnen ook worden verzonden, deze komen in de OUTPUT-ketting terecht. In elke ketting worden de pakketten vergeleken met de regels van die ketting. De verschillende standaardacties (doelen) die men kan ondernemen zijn: ACCEPT, DROP, QUEUE, RETURN en naar een zelfgemaakte ketting springen. Deze zijn ook de standaardacties in de twee andere tabellen. ACCEPT laat het pakket zijn weg vervolgen (het verlaat de ketting) terwijl DROP ervoor zorgt dat het pakket zijn bestemming niet zal bereiken. QUEUE zet het pakket in een wachtrij voor verdere verwerking door gebruikersprogramma's (dit is interessant omdat de verwerking van het pakket niet meer in de kern gebeurt, zie punt 3.1). RETURN betekent dat er terug moet worden gesprongen naar de aanroepende ketting. Indien we in één van de drie standaardkettingen zitten, zal het standaardbeleid worden gevolgd. Daarnaast bestaat er nog een actie die een extensie is, nl. REJECT. Dit doet hetzelfde als DROP, maar zendt hierbij een errorpakket naar de bron. Naast deze extensie zijn er nog meer, deze komen in de volgende punten ter sprake.
Wanneer geen enkele regel past bij het pakketje en we in een standaardketting zitten, zal het standaardbeleid voor die ketting worden gevolgd, wat een te nemen actie omvat. Indien we in een door de gebruiker gedefinieerde ketting zitten, wordt teruggesprongen naar de aanroepende ketting.

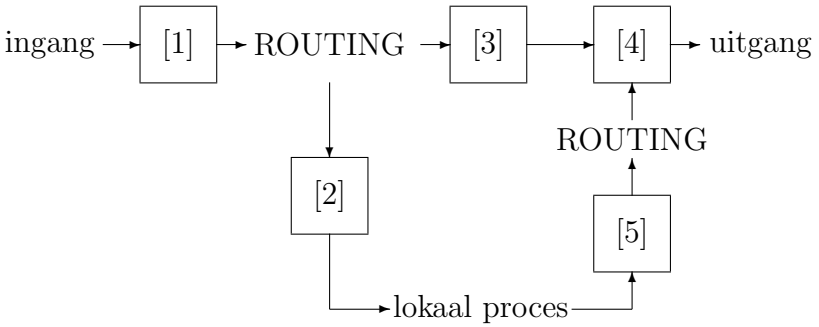
4.2.3 Mogelijkheden bij het aanmaken van regels

De mogelijkheden bij iptables zijn zeer uitgebreid. Bij de regels kunnen we o.a. volgende standaard zaken specificeren:

- protocol: bv. tcp, udp, icmp, ...
- ip-adres van de bron en bestemming.
- interfaces waar het pakket is binnengekomen of waardoor het pakket zal worden verzonden.
- aanduiden dat de regel enkel geldt (of niet geldt) voor het eerste fragment van een ip-pakket (dit is alleen maar nuttig indien we geen connecties bijhouden). Een ip-pakket kan nl. gefragmenteerd toekomen, dit bv. als de pakketgrootte groter is dan de maximale framegrootte van het transportmedium. Elk fragment bestaat wel uit een ip-kop, maar andere informatie wordt niet telkens meegegeven. Zo is het poortnummer bij tcp (*transmission control protocol*) bv. enkel uit het eerste fragment af te leiden, dit omdat dit poortnummer deel uitmaakt van de transportlaag en zich dus niet in de ip-kop bevindt. Het feit dat ip-pakketjes in de verkeerde volgorde kunnen toekomen geeft hiervoor geen problemen aangezien in de ip-kop met een fragment-offset wordt gewerkt (zie figuur 4.1).

Aangezien iptables modulair is opgebouwd kunnen er makkelijk extensies voor worden geschreven. Extensies die nu al geschreven zijn omvatten o.a.:

- **tcp**: de bron- en bestemmingspoort en de tcp-vlaggen (SYN, ACK, FIN, RST, URG, PSH, ALL, NONE) kunnen worden gespecificeerd. Dit is dus firewalling die zich op de transportlaag afspeelt (deze informatie maakt geen deel uit van de ip-kop).
- **udp**: (*user datagram protocol*) de bron- en bestemmingspoort kunnen worden gespecificeerd.
- **icmp**: (*internet control message protocol*) men kan het icmp-type van een icmp-pakket specificeren.



[**1**] : PRE_ROUTING [**2**] : LOCAL_IN (=INPUT)
[**3**] : FORWARD [**4**] : POST_ROUTING
[**5**] : LOCAL_OUT (=OUTPUT)

Figuur 4.3: Alle ip-haken

- **mac**: het ethernet MAC-bronadres kan worden opgegeven. Dus enkel pakketten die van deze ethernet-kaart afkomstig zijn zullen voldoen aan dit deel van de regel.

- **state**: een grote verbetering t.o.v. de vroegere pakketfiltercode is dat connecties kunnen worden bijgehouden (indien de code die hiervoor zorgt in de kern is gecompileerd). Met state kunnen we eisen dat een pakket één van de volgende connectie-statussen heeft:
  INVALID: het pakket hoort niet bij een gekende connectie;
  ESTABLISHED: het pakket hoort bij een connectie waarvan bidirectioneel verkeer is gezien;
  NEW: het pakket heeft een nieuwe connectie gestart, ofwel maakt het deel uit van een connectie die enkel verkeer in één richting heeft gezien;
  RELATED: het pakket start een nieuwe connectie, maar is geassocieerd met een reeds bekende connectie. Dit is bv. nodig bij het ftp-protocol (dit is een protocol dat het tcp-protocol gebruikt voor de overdracht van data tussen twee computers). Dit protocol gebruikt nl. twee connecties: een controleconnectie en een dataconnectie. De controleconnectie wordt eerst opgezet. De dataconnectie wordt pas opgezet wanneer een bestand moet worden verzonden. Het is duidelijk dat deze dataconnec-

tie geassocieerd is met de controleconnectie;

- **tos**: de *type of service* opgeven waaraan het ip-pakket moet voldoen (dit is een veld in de ip-kop, zie figuur 4.1).

Voor een volledige dissertatie over de mogelijkheden van iptables verwijzen we naar het elektronische handboek.

4.3 De nav-tabel

4.3.1 Dynamische vs. statische nav

Netwerkadresvertaling (nav) omvat alle ip-adresvertalingen die op pakketjes gebeuren. Een adresvertaling bestaat erin dat het bron- en/of bestemmingsadres van een pakketje wordt veranderd.
Er wordt onderscheid gemaakt tussen statische en dynamische nav. Bij statische nav zullen de adresvertalingsregels altijd worden uitgevoerd, zonder rekening te houden met bestaande verbindingen. Dit maakt statische nav zeer eenvoudig, aangezien verbindingen niet moeten worden bijgehouden.
Bij dynamische nav zullen connecties onthouden worden. Wanneer de software de ip-adressen van een pakketje verandert zal de software bijhouden hoe dit gebeurde. De ip-adressen van pakketjes die antwoorden op dit eerste pakket zullen dan in de omgekeerde richting worden veranderd.
Nav met de Linux-kernen die iptables gebruiken zal altijd dynamische nav zijn. Hiervoor maakt de nav-code gebruik van de code die connecties bijhoudt.

4.3.2 Waarom nav gebruiken?

- **internetconnecties**: Indien we met meerdere computers op het internet willen, maar enkel over één ip-adres beschikken, zullen we maskering (zie punt 4.3.3) willen toepassen.

- **verschillende bedieners**: We kunnen een netwerk maken van computers die het werk van binnenkomende pakketten verdelen (*load-sharing*). Een nav-box zal dan het bestemmingsadres veranderen in dat van één van de computers in het netwerk zodat de pakketjes mooi verspreid worden over alle computers.

- **transparante stroman**: Deze werkt als een normale stroman (zie punt 1.2) met het verschil dat het lokale netwerk niet weet dat het met een stroman werkt.

4.3.3 Hoe werkt nav in iptables?

De nav-tabel bestaat uit 3 ingebouwde kettingen: PREROUTING, OUTPUT en POSTROUTING (zie figuur 4.3). Pakketten die binnenkomen in de computer passeren eerst de PREROUTING-ketting vooraleer ze de routeringscode binnenkomen. Analoog zullen de pakketten die op het punt staan verder gestuurd te worden eerst nog de POSTROUTING-ketting moeten passeren. Net zoals bij de filtertabel komen pakketjes die afkomstig zijn van de firewall-machine zelf terecht in de OUTPUT-ketting.
Het moge duidelijk zijn dat pakketjes, die veranderd worden door de PREROUTING-ketting van de nav-tabel, in de FORWARD-ketting van de filtertabel in hun veranderde vorm zullen terechtkomen. De filtercode zal dus de originele vorm van het pakketje niet kunnen achterhalen.
Nav kan opgesplitst worden in bron-nav (bnav) en bestemming-nav (dnav). Bij bnav wordt het ip-adres van de bron veranderd. bnav kan enkel gebruikt worden in de POSTROUTING-ketting (en in zelfgedefinieerde kettingen die enkel worden opgeroepen vanuit deze ketting). Indien bnav in de PREROUTING-ketting zou gebeuren, dan zou de pakketfilter het echte bron-ip-adres niet weten, terwijl bnav in de OUTPUT-ketting geen zin heeft (waarom het ip-adres van de firewallmachine zelf veranderen?).
Analoog wordt bij dnav het ip-adres van de bestemming veranderd. Dnav kan enkel gebruikt worden in de PREROUTING- en OUTPUT-kettingen (en in zelfgedefinieerde kettingen die enkel worden opgeroepen vanuit één van deze kettingen). Indien we dnav in de POSTROUTING-ketting zouden

toelaten, dan zouden we de firewall (de pakketfiltertabel) kunnen misleiden, wat niet de bedoeling kan zijn. Een speciaal geval van dnav is REDIRECT. Hiermee wordt het ip-adres van de bestemming veranderd in het ip-adres van de machine (waarop iptables draait) zelf, meerbepaald het ip-adres van de interface waarop het pakketje binnenkwam.
Een speciaal geval van nav is maskering, wat een combinatie is van dnav en bnav. Deze techniek wordt meestal gebruikt indien men maar één ip-adres is toegewezen door de internetdienstenverlener maar toch met meerdere computers tegelijkertijd toegang tot het internet wil hebben. De computers binnen het netwerk worden ip-adressen toegewezen waarvan we zeker zijn dat deze adressen nergens anders op het internet gebruikt worden (deze adressen zijn door het IANA vastgelegd). De machine die aan maskering doet krijgt het ip-adres dat verkregen werd door de provider.
Het IANA (*Internet Assigned Numbers Authority*) heeft de volgende drie blokken van ip-adressen gereserveerd voor private netwerken:

klasse A	10.0.0.0	-	10.255.255.255
klasse B	172.16.0.0	-	172.31.255.255
klasse C	192.168.0.0	-	192.168.255.255

Wanneer een connectie wordt opgezet tussen een lokale computer en een bediener ergens in het internet, zal de maskeringsbox het bron-ip-adres van alle pakketjes die van de lokale computer verzonden worden veranderen in het ip-adres van de maskeingsbox. De externe bediener denkt dus dat hij praat met de maskeringsbox. Wanneer deze bediener pakketjes terugstuurt zal de maskeringsbox het doeladres vervangen door dat van de specifieke lokale gastheer.

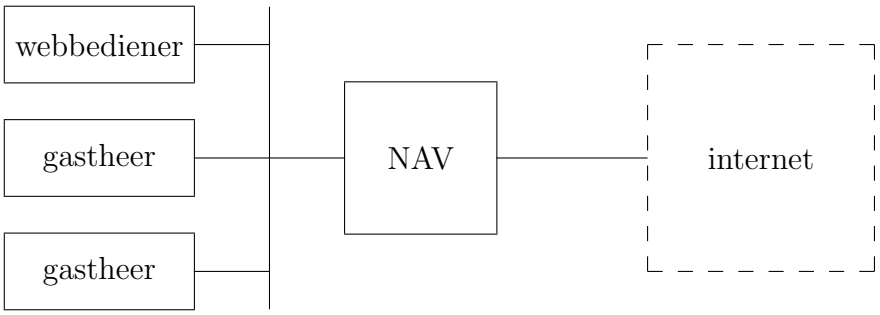
4.3.4 Een nav-voorbeeld

Een veel gebruikte techniek bij webbedieners is het volgende (zie figuur 4.4): het internet maakt een connectie op poort 80 (de http-poort) van de nav-box. Deze nav-box is echter geen webbediener. Het pakket zal doorgestuurd worden naar de echte webbediener die zich binnen het lokale netwerk bevindt, gebruik makend van nav. Het ip-adres van de bestemming zal in dat van

de webbediener worden veranderd en het pakketje zal op het lokale netwerk worden verstuurd. Het doeladres van de antwoorden die de webbediener verstuurt, wordt vervangen door het adres van de externe gastheer die de connectie opstartte. Op deze manier wordt de webbediener beschermd van de buitenwereld aangezien men het ip-adres ervan niet weet en de bediener zich achter een firewall bevindt (de firewall draait op de nav-box).
Om dit te doen werken, moeten we ervoor zorgen dat alle pakketjes die voor deze connectie verzonden worden langs de nav-box passeren. Voor connecties tussen het internet en de webbediener is dit geen probleem aangezien de nav-box tussen beide in staat. Voor connecties tussen lokale gastheren en de webbediener geeft dit echter wel een probleem (dat weliswaar makkelijk zal opgelost worden). Stel dat gastheer A een connectie met de webbediener aanvraagt via poort 80 op de nav-box. De nav-box verandert het bestemmingsadres naar dat van de echte webbediener. De webbediener zal een antwoord sturen naar het bronadres van het eerste pakketje, dus rechtstreeks naar gastheer A. De nav-box zal het pakketje niet ontvangen. Gastheer A zal de connectie niet herkennen want het bron-adres zal dat zijn van de webbediener, niet van de nav-box. Dit kan makkelijk opgelost worden door nog een nav-regel toe te voegen in de nav-box. Deze regel luidt dat alle bronadressen van pakketjes die toekomen op tcp-poort 80 moeten veranderd worden in het ip-adres van de nav-box (zodat de webbediener dus zeker alle pakketten verstuurt naar de nav-box). De twee iptables-regels zien er als volgt uit:

```
iptables -t nat -A PREROUTING -d <nav-box> -p tcp
--dport 80 -j DNAT --to <webbediener>
iptables -t nat -A POSTROUTING -d <webbediener>
-s <lokale gastheer> -p tcp --dport 80 -j SNAT --to <nav-box>
```

Hierbij moeten `<nav-box>` en `<webbediener>` door hun respectievelijke ip-adressen veranderd worden; `<lokale gastheer>` zal een netwerkadres of netwerkadressen zijn (bv. 192.68.1.0/24). Merk op dat we genoeg hebben aan twee regels aangezien nav in Linux altijd dynamisch gebeurt.



Figuur 4.4: Netwerktopologie in het voorbeeld

4.4 De aanpassingentabel

In deze tabel gebeuren alle veranderingen op het pakket, behalve de veranderingen van bron- en bestemmingsadres. Zo kan men bv. het ToS-veld (*type of service*) in de kop van het ip-pakket veranderen (zie figuur 4.1).

Hoofdstuk 5

De Linux-kern

5.1 De geboorte van Linux

Het verhaal van Linux begint in Finland in het jaar 1991 wanneer een zekere Linus B. Torvalds, toen nog student aan de universiteit van Helsinki, een PC met een 386-processor koopt om de werking ervan te bestuderen. Omdat MS/DOS de mogelijkheden van de 386 niet volledig gebruikte, wendde Linus zich tot een ander commercieel besturingssysteem: Minix, hoofdzakelijk door Andrew Tanenbaum ontwikkeld. Het Minix-systeem is een miniatuurversie van het Unix-systeem. Doordat het systeem veel tekortkomingen had, begon Linus bepaalde delen van de software te herschrijven om meer functionaliteit en mogelijkheden aan te brengen. Zijn resultaten verspreidde hij gratis over het internet, onder de titel Linux (een concatenatie van Linus en Unix). Hiermee was Linux geboren. De eerste versie van Linux, versie 0.01, werd uitgebracht in augustus 1991. Deze versie was nog in embryonale staat. Er was zelfs geen officiële aankondiging. De eerste officiële versie werd op 5 oktober 1991 gepubliceerd (versie 0.02). Door de jaren heen is het aantal ontwikkelaars van Linux blijven groeien. In het begin waren er maar enkele enthousiastelingen die het systeem hadden opgemerkt en erin geïnteresseerd raakten. Nu wordt Linux ontwikkeld door tientallen mensen verspreid over de hele wereld, die elkaar meestal nog nooit hebben ontmoet. Het internet heeft een belangrijke rol gespeeld bij de ontwikkeling van het systeem. Beeldt u bv. een ontwikkelaar in die Linux installeert op zijn machine en een fout

vindt. Hij verbetert deze fout en zendt de code naar Linus. Enkele dagen later kan de nieuwe kern worden verdeeld via het internet.

5.2 De versienummers van de kern

Het tweede cijfer van een versienummer vertelt ons meer over het soort kern. De evolutie van Linux gebeurt nl. in twee fases:

- Een **ontwikkelingsfase**: hier is de kern niet betrouwbaar, hier is het de bedoeling functionaliteit toe te voegen, optimalisaties aan te brengen en nieuwe ideeën te testen. Kernen in deze fase worden gekenmerkt door een oneven getal als tweede cijfer. Voorbeelden van zulke kernen zijn 2.3.x, 1.1.x. In deze periode wordt het meeste werk verricht op de kern. In de 2.3.x kernen werd de netfilter/iptables-architectuur bv. ontwikkeld.

- Een **stabiliseringsfase**: de zogenaamde testfases. Deze dragen versienummers zoals 2.4.0-test12. In deze fase kan geen nieuwe functionaliteit meer worden toegevoegd. Enkel foutcorrecties worden toegevoegd. Deze testfases worden dan nog gevolgd door een *prerelease*, zoals 2.4.0-prerelease. Eenmaal stabiliteit is bereikt zal een stabiele kern worden uitgebracht.

Stabiele kernen krijgen als tweede nummer een even nummer en bestaan uit 3 cijfers: op het moment dat dit geschreven wordt, heeft de meest recente stabiele Linux-kern versie 2.4.3.

5.3 Topologie van de kern

Wanneer we naar de broncodeboom van Linux kijken vinden we volgende mappen:

- **/arch**: bevat alle code die afhankelijk is van de computerarchitectuur waarop de kern zal draaien, bv. sparc (sun) t.o.v. i386 (Intel).

- **/drivers**: bevat alle aansturingssoftware; indien een bepaald randapparaat ondersteund wordt door Linux, komt dit doordat de broncode van de aansturingssoftware voor dat apparaat zich in deze map bevindt.
- **/fs**: de code van alle bestandssystemen (*file systems*) die worden ondersteund door Linux bevindt zich hier.
- **/init**: bevat de code die uitgevoerd wordt bij het opstarten van de kern.
- **/ipc**: bevat de code voor inter-procescommunicatie.
- **/kern**: bevat de centrale kerncode, zoals werkverdeling en wekkers.
- **/mm**: bevat de code voor het geheugenbeheer (*memory management*).
- **/net**: bevat alle code die te maken heeft met netwerken, behalve de aansturingssoftware voor netwerkapparatuur want deze horen thuis in de `/drivers`-map.

De plaats waar de brepf-code moet komen is natuurlijk in de `/net`-map. Meer bepaald onder `/net/bridge/netfilter/`. Dit omdat we een aanvulling schrijven voor de brugcode van Linux die gebruik maakt van netfilter.
De netfilter-code zelf alsook de iptables-code bevinden zich onder `/net/ipv4/netfilter/` en `/net/ipv6/netfilter/`.

5.4 Een Linux-kern compileren

De standaard webpagina waarlangs de kernbroncode kan worden binnengehaald is [12]. Op hun ftp-bediener kan de nieuwste versie alsook alle andere versies teruggaand tot versie 1.0 (die dateert uit 1994) worden afgehaald. De nieuwste versie wordt altijd vermeld door het bestand dat getiteld is met 'LATEST-IS-x.x.x' (bv. LATEST-IS-2.4.3). De kernversies zijn altijd in twee formaten binnen te halen: `.tar.gz`, of `.tar.bz2`. De bestanden in het eerste formaat werden gecomprimeerd met gzip, de andere bestanden met bzip2. Bestanden van het tweede formaat zijn kleiner omdat bzip2 een beter

compressie-algoritme gebruikt. Kernbroncode komt in Linux altijd onder de `/usr/src/` map.

```
cd /usr/src
```

Eenmaal het bestand is afgehaald moet het gedecomprimeerd worden. Daarna kan het bestand uitgepakt worden met `tar` (**t**ape **ar**chive). Zorg er wel op voorhand voor dat er geen `/usr/src/linux` map bestaat, aangezien het bestand in deze map zal worden uitgepakt. Hernoem indien nodig deze map naar een andere (betekenisvolle) naam, bv.

```
mv /usr/src/linux /usr/src/linux-(versienummer).
```

Indien het gz-formaat werd afgehaald doet men dit zo:

```
tar -xvzf linux-2.4.3.tar.gz
```

Anders dient bzip2 eerst uitgevoerd te worden:

```
bzip2 -d linux-2.4.3.tar.bz2 | tar -xvf -
```

Vervolgens moeten we aanduiden welke faciliteiten we in de nieuwe kern wensen. Indien er al een gecompileerde kern bestaat en we de opties van deze kern willen houden moeten we het bestand .config kopiëren van de map van de oude kern naar die van de nieuwe. Vervolgens voeren we `make oldconfig` uit en krijgen we enkel de nieuwe faciliteiten van de kern te zien en kunnen we kiezen welke we wensen en welke niet.
Anders kunnen we kiezen tussen `make menuconfig` (consolemenu) en `make xconfig` (grafisch menu in xwindows) om onze kern samen te stellen. Sommige opties (zoals de meeste aansturingssoftware) kunnen zowel resident tot de kern behoren of als module worden gecompileerd. In het eerste geval zal de code in de kern altijd aanwezig zijn en maakt het de kern dus groter. In het tweede geval wordt de module maar in de kern geladen indien daar om gevraagd is. Men kan bv. de aansturingssoftware voor ethernet-kaarten (bv. VIA-RHINE en DEC Tulip) als modules gebruiken. Deze worden dan ingeladen wanneer we de ethernet-kaarten willen gebruiken (dus typisch bij het opstarten van de kern). Wanneer we dan geen netwerk hoeven voor de

computer, kunnen we de modules uit de kern laden waardoor die minder geheugen inneemt. Er moet voor gezorgd worden dat het juiste processortype is gekozen (momenteel staat dit standaard op Pentium III). Er moet ook gezorgd worden dat de aansturingssoftware voor de harde schijf waarop de opstartpartitie staat in de kern is geladen.
Eenmaal we de configuratie van de kern hebben gekozen moet `make dep` uitgevoerd worden. Dit is een hulpprogramma dat de afhankelijkheden (*dependencies*) van de C-bestanden bepaalt en ze in een bestand zet dat .depend heet. Voor elke map zal `make dep` de C-broncodebestanden bekijken en eruit afleiden welke *headerbestanden* deze bestanden nodig hebben. Bij het compileren van de kern zal dan gebruik worden gemaakt van deze .depend-bestanden.
Daarna volgt de eigenlijke compilatie, met `make bzImage`. Indien gewerkt wordt op een smp-computer kan dit geparallelliseerd worden door

```
make -jN bzImage
```

Hierbij is N gelijk aan het aantal processors plus 1. Indien we de kern willen compileren terwijl we niet als *root* ingelogd willen blijven kunnen we de volgende opdracht gebruiken:

```
nohup make bzImage&
```

Indien we bepaalde opties als module hebben geselecteerd, moeten we nog het volgende doen:

```
make modules
make modules_install
```

Als laatste stap dient het kernbeeld te worden gekopieerd naar de `/boot`-map. In `/etc/lilo.conf` maken we een nieuwe structuur aan en voeren daarna `lilo` uit.

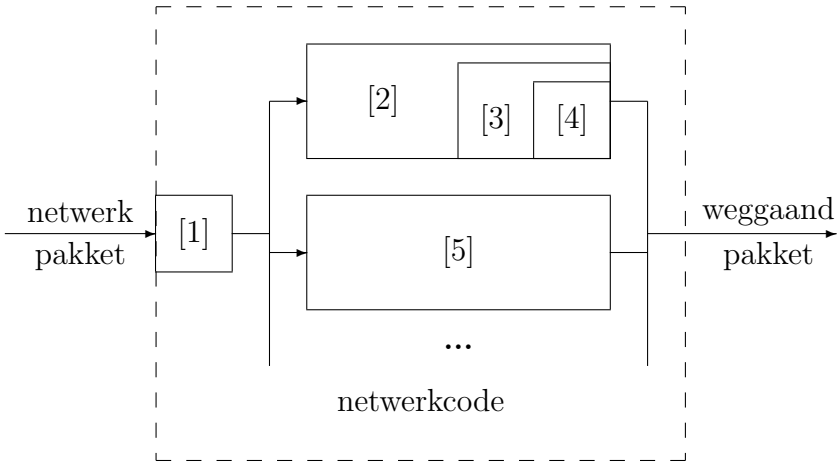
```
cp arch/i386/boot/bzImage /boot/vmlinuz-(versie)
pico /etc/lilo.conf lilo
```

Hoofdstuk 6

De netfilter-architectuur

6.1 Iptables vs. netfilter

Alhoewel iptables meestal in één adem wordt vernoemd met netfilter zijn het toch twee verschillende zaken. Netfilter is een algemeen te gebruiken infrastructuur terwijl iptables software is die deze netfilter-infrastructuur gebruikt. De reden waarom ze met elkaar verward worden, is waarschijnlijk omdat de meest bekende software in de kern die de netfilter-infrastructuur gebruikt iptables is.
Beide werden tijdens de ontwikkelingskernen versie 2.3.x ontwikkeld. Voor een vergelijking tussen de oude firewalling en de nieuwe verwijzen we naar de figuren 6.1 en 6.2. De gestreepte rechthoeken stellen de verzameling van code voor die netwerkcommunicatie mogelijk maakt voor. Zoals op de figuur te zien, maakt de brugcode en ip-code hier deel van uit. Deze verzameling is dus een onderdeel van de volledige Linux-kern. Zoals op figuur 6.1 te zien is, zitten de ip-code, ipchains-code en maskeringscode in elkaar vernesteld, dus niet modulair. In figuur 6.2 zien we dat dankzij de netfilter-architectuur de code nu wel modulair is. De volle pijlen stellen de weg voor die een pakket volgt. De gestreepte pijlen in figuur 6.2 stellen de beslissing voor die de functie teruggeeft aan netfilter (bv. NF_ACCEPT).



[1] : netprotocolcheck **[2]** : ip-code
[3] : ipchains-code **[4]** : maskeringscode
[5] : andere netwerkcode, bv. brugcode

Figuur 6.1: Netwerkcodestructuur van kernen met versie lager dan 2.3.0

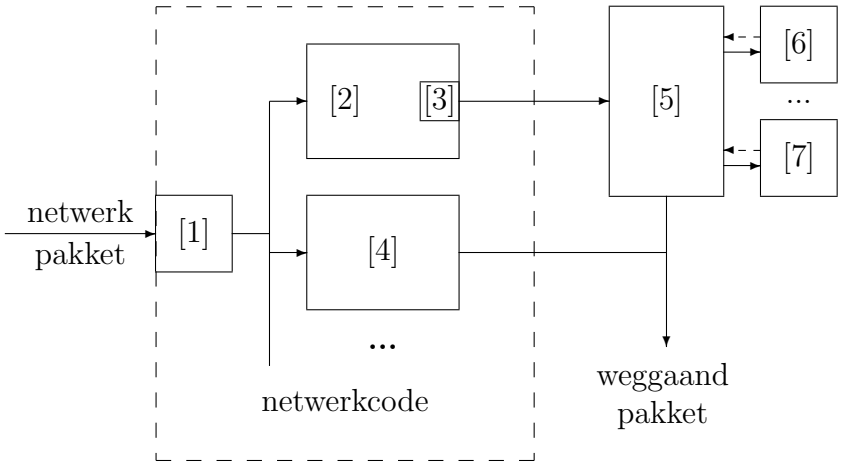
6.2 De netfilter-infrastructuur

6.2.1 Haken

Feitelijk is de netfilter-architectuur vrij eenvoudig. Ze biedt een interface tussen de netwerkcode die voor de verzending van pakketten zorgt (bv. de ip-code of de brugcode) en de software die deze pakketten wil onderzoeken.

De functie-aanroep

De netfilter-code werkt met haken. Haken zijn plaatsen in de netwerkcode (bv. de ip-code) waar gesprongen wordt naar de netfilter-code. Dit gebeurt eenvoudig door een functieaanroep die luistert naar de naam `NF_HOOK()`. Wanneer de netwerkcode deze functie aanroept, staat het de volledige controle van het pakketje af aan de netfilter-architectuur. Om alles duidelijker te maken, is het handig om de functieaanroep van dichterbij te bekijken.



[**1**] : netprotocolcheck [**2**] : ip-code
[**3**] : ip-haken [**4**] : netwerkcode die iptables niet gebruikt, bv. brugcode
[**5**] : netfilter [**6**] : iptables [**7**] : *connection tracking*

Figuur 6.2: Netwerkcodestructuur van nieuwere kernen
Netwerkcodestructuur van kernen met versie 2.3.0 of hoger

```
NF_HOOK(pf, haak, pakket, in_dev, uit_dev, accepteer_functie);
```

Beschrijving van de argumenten van de functie NF_HOOK (voor de C-kenners: NF_HOOK is eigenlijk geen functie, maar een macro):

- **pf**: De protocolfamilie. Dit is bv. IPv4 of de brugprotocolfamilie. Elke protocolfamilie kan dus haar eigen haken definiëren.
- **haak**: Het haaknummer. Het maximaal aantal haken is in de kern vastgelegd op 8. Met het haaknummer kunnen de verschillende aanroepen van `NF_HOOK()` binnen dezelfde protocolfamilie zich van elkaar onderscheiden.
- **pakket**: Het volledige pakket. Dit omvat zowel de kop behorende bij de datalinklaag, de kop van de netwerklaag als de data van de daarboven liggende lagen. Indien het een ip-pakket betreft dat vanop een ethernet-netwerk binnenkwam, omvat de variabele `pakket` dus de ethernet-kop, de ip-kop en de eigenlijke data.

- **in_dev**: de naam (en de informatie) van het apparaat (bv. een ethernet-kaart) waarop het pakket is binnengekomen.
- **uit_dev**: de naam (en de informatie) van het apparaat waarlangs het pakket verder zal worden gestuurd. Indien het pakket voor de lokale computer is bestemd, zal deze variabele leeg zijn.
- **accepteer_functie**: de functie die de netfilter-architectuur moet aanroepen indien er beslist is dat het pakket haar weg mag vervolgen. De netfilter-architectuur is dus verantwoordelijk voor het oproepen van deze functie indien het pakket haar weg mag vervolgen. In de POST_ROUTING-ip-haak bv. zorgt deze functie dat het pakket verder wordt verzonden richting haar bestemming.

Alle mogelijke informatie van het pakket wordt hierdoor meegegeven aan de netfilter-architectuur en het is de verantwoordelijkheid van deze architectuur dat de pakketten al of niet op hun bestemming komen.

De begrippen *haak* en *vasthaken*

Wanneer we over een haak praten, hebben we het dus over deze functie `NF_HOOK()`. Een haak in de netwerkcode is dus niets meer dan een aanroep van deze functie, met de juiste parameters. Wanneer we praten over een functie die zich vasthaakt aan een bepaalde haak, bedoelen we dat de functie aan de netfilter-architectuur meedeelt dat ze wenst te worden uitgevoerd wanneer een pakket op deze haak passeert (dit gebeurt in de initialisatiecode van deze functie, zie hoofdstuk 7). Wanneer een functie zich vasthaakt aan een haak, bedoelen we dus niet dat ze zich rechtstreeks bindt aan die haak. De functie verbindt zich aan de haak via de netfilter-architectuur. Om alles uit te leggen is het echter makkelijker te werken met de abstracte begrippen *haak* en *vasthaken*.

6.2.2 Netfilter: een interface

De netfilter-architectuur zorgt voor een gestandaardiseerde en modulaire aanpak van pakketonderzoekers (zoals firewalls), overzichtelijkheid is troef.

Figuur 6.2 geeft een grafisch beeld van de architectuur. Hierbij valt op te merken dat naast iptables en *connection tracking* ook andere software kan gebruik maken van netfilter. Netwerkcode die geen gebruik maakt van de netfilter-architectuur (bv. de brugcode in de standaardkern) zal geen firewallcode bevatten. Om firewalling op de brugcode te kunnen doen, zullen haken moeten gedefinieerd worden en zal de firewallsoftware moeten worden geschreven. Dit is net wat de kernaanpassing voor ip-firewalling op een brug doet.
De netfilter-architectuur houdt per netwerkfamilie een lijst van functies bij die ingehaakt zijn op een bepaalde haak. Het is duidelijk dat de meeste van die lijsten ledig zijn aangezien in de standaardkern enkel voor de netwerkfamilies IPv4, IPv6 en DECnet haken zijn gedefinieerd. Indien de lijst leeg is, wordt de functie `accepteer_functie()` aangeroepen en wordt er teruggekeerd naar de netwerkfamiliecode.
Indien de lijst niet leeg is zullen de functies die zich hebben vastgehaakt aan deze specifieke haak worden aangeroepen, deze functies krijgen alle informatie over het pakketje. De beslissing over de volgorde waarin deze functies moeten worden aangeroepen, wordt bepaald door de prioriteit van de functies. In de lijst zullen zich geen twee functies met dezelfde prioriteit bevinden (zie ook punt 7.1.5). Hoe hoger de prioriteit van de functie, hoe eerder ze zal worden uitgevoerd door netfilter.
Deze functies kunnen de informatie van het pakketje naar believen veranderen en bestuderen. Wanneer ze hiermee klaar zijn, geven ze hun beslissing terug aan de netfilter-architectuur. Deze beslissing is één van volgende waarden:

- **NF_ACCEPT**: Van deze functie krijgt het pakket het recht om verder te gaan.
- **NF_DROP**: De netfilter-code moet ervoor zorgen dat het pakketje wordt genegeerd en niet tot haar bestemming raakt.
- **NF_STOLEN**: Dit duidt aan dat de functie zich verantwoordelijk heeft gesteld voor de toekomst van het pakketje en dat de netfilter-code het pakketje moet negeren. De netfilter-code zal het pakketje

wel niet verwijderen uit het geheugen, daar moet de functie die er nu verantwoordelijk voor is, voor zorgen.

- **NF_QUEUE**: Het pakketje moet in een wachtlijst worden geplaatst en zal door gebruikersprogramma's verder worden behandeld.
- **NF_REPEAT**: Dezelfde functie moet terug aangeroepen worden voor hetzelfde pakketje.

Wanneer de functie NF_ACCEPT teruggeeft, zal de volgende functie uit de lijst op het pakketje worden losgelaten. Het is dus zo dat een pakketje maar zal mogen verdergaan indien alle functies die op de haak zijn ingeschakeld het erover eens zijn dat het mag verder gaan.
Daarentegen is het genoeg dat één functie NF_DROP teruggeeft. De andere functies uit de lijst worden niet meer uitgevoerd en het pakketje zal verwijderd worden.

6.2.3 Vasthaken op een haak

Software die geschreven wordt om zich vast te haken op netfilter-haken zal in haar initialisatiecode een oproep doen naar de netfilter-architectuur om zich te registreren op die haak. Deze software zal dus enkel worden uitgevoerd wanneer netfilter deze oproept. Hierbij moet speciaal aandacht worden besteed aan de prioriteit die aan de software moet worden gegeven (deze prioriteit wordt vastgelegd tijdens de registratie van de functie op de haak). Er moet gezorgd worden dat er geen andere software zich op dezelfde haak zal plaatsen met dezelfde prioriteit aangezien de software die zich het laatst probeert te registreren dan niet zal geregistreerd worden. Ook komt het soms voor dat de software zeker moet uitgevoerd worden voordat andere software die op de haak is vastgehaakt, wordt uitgevoerd. Dit is bv. het geval bij het bijhouden van connecties (zie punt 7.1.7).

Hoofdstuk 7

De iptables-architectuur

Indien bepaalde zaken onduidelijk lijken kan het interessant zijn om hoofdstuk 4 (en eventueel hoofdstuk 6) nog eens te bekijken. Iptables zit immers nogal ingewikkeld in elkaar.

7.1 Het kerngedeelte

7.1.1 De initialisatie van iptables

Indien iptables als een module wordt gebruikt, zal de initialisatie ervan gebeuren wanneer de module wordt ingeladen met `insmod`. Indien de code resident in de kern aanwezig is, gebeurt de initialisatie bij het opstarten van de kern.
De code zal twee lijsten initialiseren. De eerste lijst houdt alle doelen bij die in de kern zijn geladen. Deze lijst wordt geïnitialiseerd met de standaarddoelen, die altijd in de kern aanwezig zijn (zie punt 4.2.2). De tweede lijst houdt alle gelijkenisfuncties bij. Een gelijkenisfunctie maakt deel uit van een regel en beschrijft hoe een pakket eruit moet zien. Alle gelijkenisfuncties zijn extensies op de standaardspecificaties (zie punt 4.2.3). Zo'n standaardspecificatie is bv. het bron-ip-adres terwijl zo'n gelijkenisfunctie bv. de tcp-extensie is zoals besproken in punt 4.2.3. De gelijkenisfuncties kunnen als module in en uit de kern worden geladen terwijl de standaardspecificaties in de iptables-module zelf zijn gecodeerd en zich dus ook niet bevinden

in de lijst van gelijkenisfuncties. De initialisatiecode zal de tcp-, udp- en icmp-gelijkenisfuncties aan de lijst van gelijkenisfuncties toevoegen. Dus eigenlijk kunnen deze drie gelijkenisfuncties als speciale standaardspecificaties aanzien worden. Deze drie gelijkenisfuncties kunnen niet uit de kern worden verwijderd aangezien ze bij de creatie van de kern erin zijn gecompileerd.
Ook registreert iptables zich bij netfilter zodat het kan communiceren met het gebruikersprogramma dat ook iptables heet. Communicatie met gebruikersprogramma's gebeurt dus ook op een modulaire, uniforme manier voor alle software die netfilter gebruikt.
Iptables zal zich niet aan bepaalde haken vasthaken, dit doen de individuele tabellen. We hebben bv. gezien dat de nav-tabel op andere haken is vastgehaakt dan de filtertabel. Iptables zorgt dus voor de algemene structuur en functionaliteit terwijl de rest van de gewenste functionaliteit door de tabellen zelf moet worden gegeven.
De tabellen die iptables gebruiken, zullen zich dus moeten bekendmaken. De drie tabellen die we ter beschikking hebben (filter, aanpassingen, nav) kunnen net als iptables ook als module worden gebruikt. Wanneer iptables als module wordt gebruikt zullen de tabellen ook als modules moeten worden gebruikt. Indien dit niet zou gebeuren, kan iptables niet worden verwijderd aangezien het de tabellen niet kan elimineren. De kernconfiguratieprogramma's (zie punt 5.4) zullen ervoor zorgen dat deze situatie niet kan gekozen worden.
In meta-taal ziet de initialisatie van iptables er zo uit:

1. `voeg de tcp-gelijkenisfunctie toe aan de (lege) lijst;`
2. `voeg de udp-gelijkenisfunctie toe aan de lijst;`
3. `voeg de icmp-gelijkenisfunctie toe aan de lijst;`
4. `registreer iptables bij netfilter voor communicatie met gebruikersprogramma's.`

7.1.2 Initialisatie van de filtertabel

De drie kettingen van de filtertabel (INPUT, OUTPUT en FORWARD) zullen geïnitialiseerd worden als lege kettingen met als standaardbeleid ACCEPT.

Vervolgens registreert de tabel zich bij iptables en als laatste stap zal de registratie gebeuren op de drie ip-haken (zie figuren 4.2 en 4.3) die door deze tabel worden gebruikt.
In meta-taal ziet deze initialisatie er zo uit:

1. `initialiseer de drie kettingdatastructuren;`
2. `registreer bij iptables, de kettingen worden meegegeven aan iptables;`
3. `registreer bij netfilter op de drie gebruikte ip-haken.`

7.1.3 Initialisatie van de aanpassingentabel

De acties die hier gebeuren, zijn analoog met die van de filtertabel. Alhoewel er doelen bestaan voor de aanpassingentabel (bv. het ToS-doel) worden deze hier niet geregistreerd bij iptables. Dit aangezien het helemaal niet zeker is dat deze doelen ooit gaan gebruikt worden en ze op deze manier niet nutteloos geheugenruimte innemen. Indien de iptables-code een verwijzing naar een doel, dat niet in de kern aanwezig is, ontmoet, zal iptables het doel als module proberen in te laden.

7.1.4 Initialisatie van de nav-tabel

De nav-tabel wordt in iptables geregistreerd en de kettingen geïnitialiseerd. De doelen bnav en dnav worden ook geregistreerd bij iptables.
De nav-code houdt een lijst bij van alle protocollen die ondersteund worden voor nav. Aangezien nav kan gedaan worden voor de protocollen tcp, udp en icmp wordt de nav-informatie voor deze protocollen in de lijst geplaatst. Deze lijst wordt dus niet door iptables gebruikt, enkel door nav. Als laatste stap haakt de nav-code zich vast aan de PREROUTING-, POSTROUTING- en OUTPUT-haken.

7.1.5 Haakprioriteiten van de tabellen

Twee haken worden door meer dan één tabel gebruikt: OUTPUT en PREROUTING.
De prioriteiten worden weergegeven in figuur 7.1. Hoe lager de waarde, hoe hoger de prioriteit. Noteer dat bnav niet is toegelaten op de PREROUTING- en OUTPUT-haken, de filtertabel kan dus altijd weten van waar het pakket oorspronkelijk afkomstig is.
Dit zijn dus de prioriteiten die netfilter gebruikt om de uitvoeringsvolgorde van de functies, die vastgehaakt zijn op een haak, te bepalen (zie punt 6.2.2).

tabel	OUTPUT	PREROUTING
filter	0	/
nav	-100	-100
aanpassingen	-150	-150

Figuur 7.1: Prioriteiten OUTPUT- en PREROUTING-haken

7.1.6 De interactie tussen de tabellen en de iptables-architectuur

Voor de drie tabellen gebeurt dit gelijkaardig. We zullen als voorbeeld de filtertabel gebruiken. Er dient opgemerkt te worden dat nav ingewikkelder is dan filtering en het werk dat in de aanpassingentabel gebeurt. Hieronder zal nav nog eens apart worden behandeld (zie punt 7.1.8).
Wanneer een pakketje één van de drie filterhaken (INPUT, OUTPUT, FORWARD) passeert, zal de netfilter-architectuur dit pakketje doorgeven aan de filtertabelcode om het te inspecteren. De filtertabelcode krijgt dus het pakketje te zien vóór de iptables-code. Alhoewel we dit de filtertabelcode noemen zal deze code de regels van haar kettingen niet kennen. Deze regels en kettingen worden nl. door de iptables-code bijgehouden. Alle acties die de kettingen en regels nodig hebben worden dus door de iptables-code uitgevoerd.

Indien de filtertabelcode het pakketje niet zou doorgeven aan de iptables-code, zou iptables het pakketje nooit zien. Maar dat is de bedoeling natuurlijk niet. Zowel de aanpassingen- als de filtertabel geven het pakketje direct door aan iptables. De nav-tabel doet eerst een paar controles en geeft het pakketje niet altijd door aan iptables (zie punt 7.1.8). Voor de filtertabel ziet dit er in meta-taal zo uit:

1. `netfilter geeft het pakket aan de code specifiek voor de filtertabel;`
2. `deze code roept de iptables-code op;`
3. `de beslissing van iptables wordt teruggegeven aan netfilter.`

Bekijken we even wat er gebeurt in de iptables-code.
Alle gelijkenisfuncties en doelen die in regels van kettingen voorkomen zullen zich in de kern bevinden (indien het gebruikersprogramma een regel toevoegt die gelijkenisfuncties of doelen vereist die niet in de kern zitten, zullen deze direct worden ingeladen als module, dus nog vóór een pakketje met deze regel wordt vergeleken). Hier zal dus nooit een module moeten worden ingeladen. De iptables-code zal één voor één de regels van de ketting overlopen. Per regel zal er eerst gekeken worden of de standaardspecificaties van de regel overeenkomen met het pakketje. Daarna zullen alle voor die regel gebruikte gelijkenisfuncties op het pakketje worden uitgevoerd. Zo'n gelijkenisfunctie is een functie die het pakketje bekijkt en besluit of het overeenkomt met de regel van de gelijkenisfunctie. Indien dit alles overeenkomt met het pakketje zal het doel worden uitgevoerd. Deze doelen worden besproken in punt 4.2.2. De beslissing die iptables zal nemen is afhankelijk van het doel. Indien het doel bv. DROP is zal de beslissing NF_DROP zijn, indien het doel bv. een logging van de pakketten voorstelt zal de beslissing NF_ACCEPT zijn. Iptables zal de beslissing mededelen aan de filtertabelcode die de beslissing op haar beurt zal mededelen aan de netfilter-code. Voor wat er hierna gebeurt met het pakketje verwijzen we naar hoofdstuk 6 dat handelt over deze netfilter-code. Er dient nog opgemerkt te worden dat het pakketje mag veranderd worden. Dit gebeurt echter alleen maar in de aanpassingen- en nav-tabellen.

7.1.7 Het bijhouden van een connectie

Inleiding

Bij een communicatie tussen twee computers is het logisch dat het verkeer van die communicatie bidirectioneel is, een communicatie is altijd een spel van vraag en antwoord.
De code die connecties bijhoudt voor het ip-protocol heeft een ruimere definitie voor 'connectie' dan normaal. Zo bestaat het concept 'connectie' niet in het ip-protocol. Voor de code die connecties bijhoudt bestaat dit concept echter wel voor alle ip-pakketten. Een ip-pakket zal ofwel behoren tot een connectie, ofwel een nieuwe connectie opstarten. Hierbij zal,indien mogelijk, rekening worden gehouden met het protocol dat dit ip-pakket aanmaakte (bv. het tcp- of udp-protocol). Indien er protocolspecifieke code voor het bijhouden van een connectie bestaat voor het protocol dat dit pakket aanmaakte zal, deze worden gebruikt. Zo bestaan er voor de icmp-, udp- en tcp-protocollen extensies voor het bijhouden van connecties. Om te kijken tot welke connectie een pakket behoort, wordt eerst naar het bron- en bestemmings-ip-adres gekeken. Daarna wordt de protocolspecifieke code uitgevoerd (indien aanwezig). Het protocol dat dit pakket aanmaakte is bekend aangezien er in de ip-kop een veld aanwezig is dat dit protocol specificeert (zie figuur 4.1).
Om de werkwijze duidelijker te maken geven we enkele voorbeelden. Stel dat pakket A eerder werd gezien dan pakket B en dat we nu pakket B moeten verwerken. Voor deze voorbeelden is het voldoende te weten dat een udp-kop o.a. bestaat uit het bron- en bestemmingspoortnummer. Dit poortnummer wordt gebruikt om udp-verbindingen van elkaar te onderscheiden.

- Het bron- en bestemmings-ip-adres van pakketten A en B is hetzelfde en het protocol dat deze aanmaakte is udp. De protocolspecifieke code voor het udp-protocol wordt aangeroepen. Deze code zal kijken of de udp-poorten van pakket B overeenkomen met de udp-poorten van pakket A. Indien alles overeenkomt behoort pakket B tot dezelfde connectie

als pakket A. Beide pakketten werden in dezelfde netwerkrichting verzonden.

- Een variatie op het eerste voorbeeld is wanneer bij pakket B bron en bestemming telkens zijn omgewisseld. Het bron-ip-adres van pakket B is dan het bestemmings-ip-adres van pakket A, enz. Dan zal pakket B tot dezelfde connectie behoren als pakket A, maar dan in de omgekeerde netwerkrichting.

- Een andere variatie is dat er geen protocolspecifieke code aanwezig is voor het protocol. Dan zal enkel met de ip-adressen rekening worden gehouden.

- Het is natuurlijk ook mogelijk dat pakket B het eerste pakket is dat tussen twee adressen wordt verzonden. Dan zal dit pakket aanschouwd worden als een pakket dat een nieuwe connectie opzet.

Werkwijze

Een firewall die over code beschikt om connecties bij te houden, zal voor volledige connecties kunnen beslissen of ze toegelaten zijn.
Zonder deze code kan een firewall niet weten of een pakket, dat zich voordoet als een antwoord op een eerder verzonden pakket in de omgekeerde netwerkrichting, wel degelijk deel uitmaakt van een connectie. Indien we de connecties wel kunnen volgen, zullen deze potentieel gevaarlijke pakketten de firewall niet kunnen passeren (indien de firewall goed geconfigureerd is). Ook voor nav moeten we connecties kunnen bijhouden.
Deze code is ingehaakt op de PREROUTING-, OUTPUT-, INPUT- en POSTROUTING-haken, met de prioriteiten zoals gegeven in figuur 7.2. Het gebruikersprogramma iptables heeft geen invloed op de werking van deze code. Door de prioriteiten zien we dat de code op de haken PREROUTING en OUTPUT als eerste zal worden uitgevoerd door de netfilter-architectuur, terwijl ze als laatste wordt uitgevoerd bij de andere twee haken.
De connectiegegevens worden bijgehouden op de PREROUTING- en OUTPUT-haken aangezien de pakketten daar in de netwerkcode terecht komen.

haak	prioriteit
PREROUTING	-200
OUTPUT	-200
INPUT	maximum
POSTROUTING	maximum

Figuur 7.2: Prioriteiten voor *connection tracking*

De pakketten afkomstig van de lokale computer passeren nl. eerst de OUTPUT-haak terwijl de pakketten afkomstig van elders eerst de PREROUTING-haak passeren (zie figuur 4.3). Op de twee andere haken zal de code enkele zaken in orde brengen. Laten we beginnen met de PREROUTING- en OUTPUT-haken.

De PREROUTING- en OUTPUT-haken

Indien het ip-pakket gefragmenteerd toekomt zal dit pakket gedefragmenteerd worden. Zolang het volledige ip-pakket niet is toegekomen zullen de fragmenten in een lijst worden gestopt en zal de beslissing NF_STOLEN (zie punt 6.2.2) aan netfilter worden doorgegeven. Zoals eerder gezegd betekent dit dat netfilter zich niets van het pakket aantrekt, wat ons verzekert dat andere code aan het fragment niets kan veranderen. Wanneer we het laatste fragment van het ip-pakket ontvangen, wordt het volledige ip-pakket geconstrueerd. De verschillende fragmenten zullen niet meer bestaan, enkel het volledige ip-pakket. Vervolgens wordt gekeken of het pakket deel uitmaakt van een bestaande connectie (door in de lijst met connecties te kijken). Indien het bij een connectie hoort, zal de staat van die connectie worden bijgewerkt, de mogelijkheden zijn ESTABLISHED, RELATED, NEW en INVALID (zie ook punt 4.2.3 i.v.m. state). De betekenis van deze drie toestanden wordt hier, ter herinnering, herhaald:

ESTABLISHED: het pakket hoort bij een connectie waarvan bidirectioneel verkeer is gezien;
NEW: het pakket start een nieuwe connectie, ofwel maakt het deel uit van

een connectie waarvan tot hiertoe enkel verkeer in één richting is gezien;
RELATED: het pakket start een nieuwe connectie, maar is geassocieerd met een reeds bekende connectie. Dit is bv. nodig bij het ftp-protocol (dit is een protocol dat het tcp-protocol gebruikt voor de overdracht van data tussen twee computers). Dit protocol gebruikt nl. twee connecties: een controleconnectie en een dataconnectie. De controleconnectie wordt eerst opgezet. De dataconnectie wordt pas opgezet wanneer een bestand moet worden verzonden. Het is duidelijk dat deze dataconnectie geassocieerd is met de controleconnectie;
INVALID: heeft geen van de vorige drie toestanden.

Wanneer het pakket een nieuwe connectie wil opstarten, zal een connectie met als toestand NEW worden toegevoegd aan de lijst van connecties.
De code zorgt ervoor dat de informatie over de connectie waartoe het pakket behoort, beschikbaar is. Andere functies die zich ingehaakt hebben op de ip-haken zullen dus tot deze informatie toegang hebben. De nav-code (zie punt 7.1.8) en het state-doel (zie punt 4.2.3) maken hiervan gebruik. Hierna wordt de controle teruggegeven aan netfilter via een NF_ACCEPT (zie punt 6.2.2).
Deze connectie is echter nog niet zeker. Neem bv. een pakket bestemd voor de lokale computer. De pakketfilter kan beslissen dat dit pakket niet naar de lokale computer mag verzonden worden (dit gebeurt in de INPUT-ketting behorende bij de INPUT-haak, zie hoofdstuk 4). Indien deze pakketfilter het pakket niet doorlaat, zal de netfilter-code ervoor zorgen dat de zonet aangemaakte connectie behorende bij dit pakket zal vernietigd worden. Dit is mogelijk aangezien de informatie over de connectie beschikbaar is via het pakket (zie vorige paragraaf). Nu wordt duidelijk waarom de code ook moet ingehaakt zijn op de POSTROUTING- en INPUT-haken, daar moet de connectie nl. worden bevestigd. Deze bevestiging komt erop neer dat een bepaalde variabele behorende bij de connectie een bepaalde waarde krijgt. Dankzij deze waarde kunnen we onderscheid maken tussen echte, bevestigde connecties en de andere. Het is duidelijk dat, zolang een connectie niet bevestigd is, deze niet zal aanzien worden als een geldige connectie. Dit is ook de reden

waarom deze bevestigingscode altijd als laatste moet uitgevoerd worden door netfilter.

De INPUT-haak

Hier wordt de connectie bevestigd. Daarnaast wordt een wekker geassocieerd met deze connectie. Indien er tijdens een bepaalde periode geen verkeer is behorende bij deze connectie, zal deze wekker zorgen dat de connectie wordt verwijderd.

De POSTFORWARDING-haak

Hier gebeurt hetzelfde als in de code voor de INPUT-haak, maar daarnaast moet het pakket ook terug gefragmenteerd worden indien de grootte van het pakket de mtu (*maximum transfer unit*) van het bestemmingsnetwerk overstijgt. Eenmaal dit is gebeurd, wordt de controle teruggegeven aan de netfilter-code, die ervoor zal zorgen dat het pakket verzonden wordt.

7.1.8 De nav-werkwijze

Voor een inleiding, zie punt 4.3.
Op alle drie de haken waarop nav is vastgehaakt (PREROUTING, POSTROUTING en OUTPUT), is de werkwijze dezelfde. Er wordt gekeken naar de informatie van de connectie waartoe het pakket behoort (de code die deze connecties bijhoudt, werd eerder door netfilter uitgevoerd). Merk op dat eveneens dankzij de code die de connecties bijhoudt het ip-pakket volledig is, dus niet uit fragmenten is opgebouwd. De nav-manipulaties die voor dit pakket moeten gebeuren, zijn al bekend voor de nav-code in twee gevallen:

- Het pakket behoort tot een connectie die in de staat ESTABLISHED verkeert. De connectie heeft dus bidirectioneel verkeer gezien, en hierdoor zullen zowel de bnav- als dnav-manipulaties voor deze connectie reeds bekend zijn.

- Het pakket behoort tot een connectie die in de staat NEW of RELATED verkeert en het type nav (dnav of bnav) dat op deze haak voor het

pakket moet gebeuren, is al eens gebruikt voor de verbinding waartoe het pakket behoort. Merk op dat we aan de hand van de haak weten of we met bnav of dnav te maken zullen hebben (bnav en dnav komen nooit op dezelfde haak voor).

In de andere gevallen zal iptables worden geconsulteerd om de manipulaties die moeten gedaan worden voor dit pakket op deze haak, te verkrijgen. De nav-tabel zal dus geconsulteerd worden.
De nav-manipulaties worden door de nav-code bijgehouden in een lijst, per connectie (zodat iptables dus niet telkens moet worden geraadpleegd). Deze lijst is altijd beschikbaar voor de nav-code.
Als laatste stap zullen de manipulaties uit de lijst worden uitgevoerd.

7.2 Het gebruikersprogramma iptables

Wanneer een gebruiker iets aan de iptables-kettingen wil veranderen, zal hij/zij gebruik maken van het programma iptables (dit noemt men een gebruikersprogramma, aangezien het zich buiten de kern bevindt).
Dit programma zal aan de kern (meerbepaald de iptables-code) de huidige inhoud van een tabel (of van een bepaalde ketting) vragen. Op deze huidige toestand zal het programma de door de gebruiker gespecificeerde veranderingen aanbrengen en deze nieuwe inhoud zal teruggegeven worden aan de kern. Het gebruikersprogramma weet perfect op welke wijze de tabellen, kettingen en regels worden opgeslagen in de kern. Moest de kern deze veranderingen rechtstreeks afhandelen zou het gebruikersprogramma niet moeten weten hoe de kern de informatie opslaat, maar zou de kern wel een stuk groter zijn (door de extra code). Ook zouden de mogelijkheden moeilijker uitbreidbaar zijn. Stel bv. dat we geen regel tussen twee regels kunnen invoegen. Indien we dit toch wensen, zou de kern herschreven dienen te worden, terwijl nu enkel het gebruikersprogramma moet worden aangepast.
Het mag natuurlijk niet kunnen dat een gebruikersprogramma de kern doet crashen. Daarom moet de kern de door het gebruikersprogramma geleverde nieuwe inhoud controleren zodat er geen inconsistenties zijn. Wanneer de

nieuwe inhoud goedgekeurd is door de kern zullen de gelijkenisfunctie- en doelmodules die gebruikt zullen worden en nog niet aanwezig zijn, in de kern worden geladen. De oude informatie zal worden vernietigd en vanaf dan zal de iptables-kerncode met de nieuwe informatie werken.

7.3 Uitbreidbaarheid

Iptables is makkelijk uitbreidbaar. Zo kunnen nieuwe tabellen, doelen en gelijkenisfuncties worden ontworpen (ook als module). Indien dit gebeurt, moet natuurlijk ervoor gezorgd worden dat het gebruikersprogramma ermee overweg kan. Aangezien ook het gebruikersprogramma makkelijk uitbreidbaar is, geeft dit geen grote problemen.

Hoofdstuk 8

Brepf: brug ethernet-protocolfilter

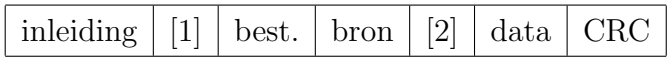
In tegenstelling tot de vorige hoofdstukken beschrijven de volgende hoofdstukken de code die voor deze thesis werd geschreven. Alle voorgaande hoofdstukken beschreven bestaande infrastructuren.
In dit hoofdstuk zullen we de brepf-code die voor deze thesis werd geschreven, gedetailleerd bespreken. De volledige brepf-code werd dus voor deze thesis ontworpen.
De meeste technieken die gebruikt worden in de brepf-code, wendt men ook aan in de iptables-code. De intensieve studie van deze iptables-code heeft het schrijven van deze software dan ook sterk geholpen. We bespreken nu pas deze technieken om het vorige hoofdstuk zo eenvoudig mogelijk te houden. Het is ook beter om de zelf geschreven code uitgebreider te behandelen. Ook is de structuur van deze code eenvoudiger dan die van iptables, wat een meer uitgebreide behandeling beter toelaat. Eerst bekijken we het ethernet-frame van dichterbij.

8.1 De ethernet-brug en het ethernet-frame

Een brug dient, zoals gezegd, voor het op transparante wijze routeren van pakketten. Een brug heeft genoeg aan het bron- en bestemmingsadres van de datalinklaag (zie figuur 1.1) om deze routering te doen. De brug houdt

dynamisch een tabel bij van de adressen met de plaats in het netwerk (d.i. via welke interface het adres te bereiken is). De brugcode die in Linux aanwezig is, ondersteunt ethernet, wat een datalinklaagprotocol is. In figuur 8.1 wordt de structuur van een ethernet-frame gegeven. Alle data die over een ethernet-netwerk worden gestuurd, komen toe verpakt in ethernet-frames.

Bespreking van het ethernet-frame:



Figuur 8.1: Ethernet-kop

- **Inleiding**: 7 bytes met waarde 10101010.
- **[1]**: Duidt de start van het frame aan. 1 byte, nl. 10101011.
- **Bron- en bestemmingsadres**: elk 6 bytes en uniek in de wereld.
- **[2]**: Typeveld. Betekent ofwel welk protocol dit pakket creëerde, ofwel de lengte van het dataveld. Indien de waarde van dit veld tussen 46 en 1500 ligt stelt het de lengte voor (zie de commentaar over het dataveld). Dit is een verouderde vorm van ethernet-communicatie. Indien de waarde groter of gelijk is aan 2048 stelt dit veld een protocol voor.
- **Data**: De eigenlijke data, de lengte variëert van 46 tot 1500 bytes. Deze waarden worden opgelegd door het CSMA/CD-protocol (*carrier sense multiple access/collision detection*). Dit protocol regelt de toegang tot de informatiedrager (bv. optische kabel) bij ethernet-netwerken.
- **CRC**: *Cyclic redundancy check*. Een methode om fouten in het frame te kunnen detecteren, de lengte is 4 bytes.

Koppen en data van hogerliggende OSI-lagen zullen zich in het dataveld bevinden (indien nodig gefragmenteerd over meerdere frames).

8.2 De functie en mogelijkheden van brepf

8.2.1 Functie

Zoals eerder gezegd laat de brug alle pakketten door zonder firewalling. Indien de iptables-code de ip-pakketten die de brug passeren kan filteren, zullen alle pakketten die door een ander protocol zijn gemaakt (bv. appletalk-pakketten) nog altijd zonder firewalling worden gerouteerd.
Met de brug ethernet-protocolfilter kunnen we pakketfiltering toepassen op het protocoltypeveld van het ethernet-frame. Daarnaast kunnen nog statistieken geconsulteerd worden.
Net zoals de tabellen van iptables werkt de brepf-code met kettingen en regels. Er zijn 5 kettingen die elk op een andere haak zijn ingehaakt. Deze haken zijn uiteraard niet de ip-haken, maar wel haken gedefinieerd voor de brugcode van Linux. De namen van de kettingen zijn: PRE_ROUTING, LOCAL_IN, FORWARD, LOCAL_OUT en POST_ROUTING. De manier waarop pakketten de verschillende haken doorlopen is identiek met figuur 4.3.

8.2.2 Verschillende commando's van brepf

brepf -A haak -p protocol [-i in_interface] [-o uit_interface] -t doel
Een regel toevoegen aan de ketting met naam haak. We zien dat een regel drie zaken kan specificeren:

- Het protocol. Deze specificatie is verplicht;
- De interface waarop het pakket is toegekomen;
- De interface waarop het pakket de computer zal verlaten.

Het doel is ACCEPT of DROP. Deze hebben dezelfde betekenis als hun equivalenten in iptables.

brepf -D haak -p protocol [-in_interface] [-o uit_interface] -t doel
Een regel verwijderen.

brepf -P haak standaardbeleid
Het standaarddoel (beleid) van de ketting veranderen, dit is ACCEPT of DROP. Indien geen enkele regel gevonden wordt die correspondeert met het pakket zal het beleid worden gevolgd.

brepf -F [haak]
De volledige tabel leegmaken (*flush*) of de gespecificeerde ketting leegmaken.

brepf -V
Het versienummer van het gebruikersprogramma brepf weergeven.

brepf -h
Een helptekst weergeven.

brepf -L [haak]
De volledige tabel weergeven, of enkel de informatie van de gespecificeerde ketting weergeven.

brepf -l
De databank weergeven. De kern kan nl. een databank onderhouden met de verschillende protocollen die de brughaken passeren. Dit kan handig zijn voor monitoring en om te weten te komen welke regels voor de ethernet-protocolfilter moeten opgesteld worden (zie appendix A).

brepf -d [y/n]
y: De databank opstarten en bijhouden.
n: De databank niet gebruiken (ze wordt uit het geheugen verwijderd).

brepf -c [y/n]
y: per regel wordt een teller (*counter*) bijgehouden. Wanneer een pakket overeenstemt met de regel zal de teller worden verhoogd. Wanneer het commando -L wordt uitgevoerd zal deze teller naast de regel worden getoond.
n: geen tellers bijhouden.

Daarnaast zijn voor de commando's alternatieve schrijfwijzes mogelijk, zie figuur 8.2.

commando	alternatief
-A	--append
-D	--delete
-P	--policy
-F	--flush
-V	--version
-h	--help
-L	--list
-l	--listdb
-d	--db
-c	--count
-p	--protocol
-t	--target
-i	--in-interface
-o	--out-interface

Figuur 8.2: Alternatieve schrijfwijze

8.3 De reis van een pakket doorheen de netwerkstack van Linux

De informatie in dit punt is grotendeels afkomstig uit [13]. We behandelen het geval van een pakket dat op een computer met een i386-architectuur (Intel) toekomt.

Wanneer een ethernet-kaart een ethernet-frame ontvangt dat voor deze kaart bestemd is, zal het een onderbrekingsaanvraag genereren. Dit frame is bestemd voor de netwerkkaart indien het bestemmingsadres (een MAC-adres) dit van de kaart is, indien het een datalinklaagomroepframe is of indien de kaart alle frames aan de computer doorgeeft; de kaart bevindt zich dan in de *promiscuous mode* (wat het geval is bij een brug).

Hierdoor zal het genereren van onderbrekingen tijdelijk uitgeschakeld worden, nl. zolang de onderbrekingsafhandelaar actief is. Het pakket zal via DMA (*direct memory access*), PIO (*programmed input/output*) of via een

andere methode in het hoofdgeheugen worden geladen. De volledige informatie van het pakket (ook de interface waar het is binnengekomen bv.) wordt in een wachtrij gestopt. Bij deze wachtrij hoort een software-onderbreking (SO, zie punt 8.4.5), de netwerkontvangst-SO. Deze SO wordt gemarkeerd voor uitvoering. Hiermee is de onderbrekingsafhandelaar klaar en zullen de onderbrekingen weer toegelaten worden. Indien de wachtrij vol zat, wordt het pakketje genegeerd en zal de SO niet worden gemarkeerd voor uitvoering. Deze software-onderbrekingen zijn functies die enkel uitgevoerd worden wanneer ze gemarkeerd zijn voor uitvoering en wanneer de kern beslist dat ze mogen werken. De SO-afhandelaar zal alle gemarkeerde SO's één voor één uitvoeren. De SO-afhandelaar wordt opgeroepen op drie plaatsen in de kern:

- In de generische onderbrekingsafhandelaar, deze wordt uitgevoerd telkens een onderbrekingsaanvraag aankomt. Van hieruit wordt dan de onderbrekingsafhandelaar voor de onderbreking uitgevoerd en daarna worden de voor uitvoering gemarkeerde software-onderbrekingen uitgevoerd.

- Wanneer er teruggekeerd wordt van een systeemaanroep. Een systeemaanroep wordt veroorzaakt door een gebruikersprogramma dat een kernfunctionaliteit nodig heeft en dus een systeemaanroep doet. Hierdoor krijgt de kern de controle over de computer en zal de kernfunctie behorende bij de systeemaanroep worden uitgevoerd. Hierna zullen alle software-onderbrekingen worden afgehandeld, vooraleer terug te keren naar het gebruikersprogramma.

- In de proceswerkverdeler van de kern.

Wanneer de SO-afhandelaar wordt uitgevoerd kijkt hij of er al een andere SO-afhandelaar bezig is op de processor. Indien zo zal de SO-afhandelaar direct eindigen aangezien anders een oneindige lus kan ontstaan: de SO-afhandelaar wordt onderbroken, de generische onderbrekinsafhandelaar roept de SO-afhandelaar op, deze wordt terug onderbroken, enzovoort.
Wanneer de netwerkontvangst-SO wordt uitgevoerd zal het de pakketten uit de wachtlijn halen en deze pakketten verwerken. Deze verwerking kan bv. de

iptables-code omvatten, of bv. de routingcode van de brug. Wanneer deze code wordt uitgevoerd zitten we dus in een software-onderbreking. Ook de brug ethernet-protocolfilter zal dus in deze SO uitgevoerd worden. Bij de implementatie ervan moet hiermee rekening worden gehouden.

8.4 Kernslotvergrendeling in Linux

De hier verstrekte informatie komt grotendeels uit [14]. Voor het correct functioneren van de ethernet-protocolfilter speelt slotvergrendeling een belangrijke rol, daarom wordt deze vergrendeling en hetgeen ermee verband houdt hier van dichterbij bekeken.
Conflicten tussen gelijktijdige processen treden op wanneer de processen wedijveren om het gebruik van dezelfde bron. Het gebruik van slotvergrendeling om toegang tot deze bronnen te krijgen, kan aanzien worden als het gebruiken van een scheidsrechter die ervoor zorgt dat de bronnen op een eerlijke en veilige manier toegankelijk zijn voor de processen.

8.4.1 De noodzaak van slotvergrendeling

Om de noodzaak van sloten te beklemtonen, geven we een eenvoudig voorbeeld (zie [9] voor een meer volledige bespreking).
Stel dat 2 instanties van een programma 'tegelijkertijd' uitgevoerd worden en dezelfde variabele `teller` gebruiken. Wat van hen verwacht wordt te doen zal er zo uitzien:

```
Instantie 1                     Instantie 2
-----------                     -----------
lees teller (5)
tel 1 bij teller op (6)
schrijf waarde in teller (6)
                                lees teller (6)
                                tel 1 bij teller op (7)
                                schrijf waarde in teller (7)
```

Wat echter verkeerd kan lopen, is dit:

```
Instantie 1                    Instantie 2
-----------                     -----------
lees teller (5)
                               lees teller (5)
tel 1 bij teller op (6)
                               tel 1 bij teller op (6)
schrijf waarde in teller (6)
                               schrijf waarde in teller (6)
```

In het vakjargon noemt men deze situaties, waar wat gebeurt afhankelijk is van het relatieve tijdstip waarop taken worden uitgevoerd, race-condities. De code die dit gelijktijdigheidsprobleem bevat wordt de kritieke sectie genoemd. De oplossing is het vinden van deze kritieke secties en sloten te gebruiken zodat enkel één instantie de kritieke sectie kan betreden op een bepaald moment.

8.4.2 De smp-situatie

Een smp-machine is een machine met meerdere processoren die onafhankelijk van elkaar werken, maar hetzelfde geheugen en besturingssysteem gebruiken. Eén kopie van het besturingssysteem heeft dus de controle over alle processoren. Op elk moment kan elk van de processoren in een smp-systeem in één van volgende toestanden verkeren (in Linux):

- Niet geassocieerd aan een proces, een hardware-onderbreking afhandelend;
- Niet geassocieerd aan een proces; een SO, *tasklet* of *bottom half* afhandelend;
- Geassocieerd aan een proces, maar in kernmodus;
- Geassocieerd aan een proces, een gebruikersprogramma.

Er bestaat een strikte ordening tussen deze: behalve de laatste categorie kan een categorie enkel preëmptief onderbroken worden door de categorieën

erboven. Bv. wanneer een SO op een processor wordt uitgevoerd, zal een andere SO deze niet kunnen onderbreken, terwijl een hardware-onderbreking dit wel zal kunnen. Deze ordening geldt natuurlijk niet tussen processoren onderling aangezien deze elkaar niet kunnen onderbreken.

8.4.3 Twee soorten sloten: spinsloten en semaforen

Een spinslot werkt als volgt: wanneer een taak het spinslot niet kan vergrendelen, blijft de taak proberen tot het lukt. De taak zal niet vrijwillig de controle over de computer afgeven. Dit maakt spinsloten zeer klein en snel.
Met een semafoor kunnen meerdere taken tegelijk het semafoorslot vergrendelen (dit aantal is begrensd en wordt gespecificeerd bij de creatie van de semafoor). Wanneer een taak het slot niet kan vergrendelen (omdat het maximale aantal vergrendelingen is bereikt), zal de taak in een wachtlijst voor deze semafoor worden gezet, totdat de taak toegang krijgt en geactiveerd wordt. De processor zal in de tussentijd iets anders doen terwijl de taak wacht.
Het moge duidelijk zijn dat werken met semaforen de code veel trager maakt dan werken met spinsloten. Aangezien de ethernet-protocolfilter wordt uitgevoerd op elk pakketje dat op de interfaces (behorende tot een brug) van een computer binnenkomt, is het niet gewenst semaforen te gebruiken aangezien de kans op verloren pakketten te groot zou zijn.
Bij een kern voor een uniprocessormachine zijn spinsloten niet nodig aangezien niets terzelfdertijd kan uitgevoerd worden. Wachten met een spinslot zou totaal nutteloos zijn omdat geen andere code wordt uitgevoerd op dat moment en het dus zeker is dat het slot niet ontgrendeld zal worden terwijl we wachten. Semaforen blijven daarentegen wel nuttig bij uniprocessoren aangezien de taak in een wachtlijn zal worden gestoken en andere code zal worden uitgevoerd.
Zowel spinsloten als semaforen hebben lees/schrijf-varianten. Deze delen de gebruikers in twee klasses in: de lezers en de schrijvers. Velen kunnen het leesslot tegelijkertijd vergrendelen, daarentegen kan niemand het slot vergrendelen terwijl een schrijver het heeft vergrendeld (er mogen dus geen an-

dere lezers of schrijvers zijn). Voor de sloten van de ethernet-protocolfilter worden lees/schrijf-spinsloten gebruikt.

8.4.4 Hardware-onderbrekingen (hard IRQ's)

Klokken, netwerkkaarten en toetsenborden zijn voorbeelden van hardware-apparaten die onderbrekingen produceren op gelijk welk ogenblik. Wanneer dit gebeurt zal de kern een onderbrekingsafhandelaar uitvoeren. Wanneer andere onderbrekingen tijdens deze uitvoering aankomen, zullen ze in een wachtlijn worden geplaatst of worden genegeerd. Omdat hij de onderbrekingen uitschakelt moet deze afhandelaar snel zijn: veelal doet hij niets meer dan de onderbreking beantwoorden en een SO voor uitvoering markeren.

8.4.5 Software-onderbrekingscontext: *bottom halves*, *tasklets* en software-onderbrekingen

Veel van het echte onderbrekingsafhandelingswerk gebeurt hier (aangezien onderbrekingen hier niet zijn uitgeschakeld, in tegenstelling tot bij de onderbrekingsafhandelaars).
Hoeveel processoren er ook mogen zijn, twee *bottom halves* zullen nooit tegelijkertijd worden uitgevoerd. Software-onderbrekingen zijn de volledige smp-versies van *bottom halves* want ze kunnen op zoveel processoren als nodig tegelijkertijd worden uitgevoerd. Dit betekent dat moet opgelet worden voor race-condities voor gemeenschappelijk gebruikte data. *Tasklets* zijn zoals SO's maar garanderen dat een bepaald *tasklet* maar op één processor tegelijkertijd zal uitgevoerd worden. Verschillende *tasklets* kunnen echter wel tegelijkertijd worden uitgevoerd (in tegenstelling tot bij *bottom halves*). Zoals eerder gezegd maakt de ethernet-protocolfilter deel uit van een SO.

Slotvergrendeling tussen gebruikerscontext en SO's/*tasklets*/*bottom halves*

We zitten in gebruikerscontext wanneer een gebruikersprogramma een systeemaanroep heeft gedaan. Voor de ethernet-protocolfilter zitten we in ge-

bruikerscontext wanneer het gebruikersprogramma informatie vraagt of geeft aan de kern (bv. de databankgegevens). Aangezien deze gebruikerscontext kan worden onderbroken door een SO en de netwerkontvangst-SO deze databank ook gebruikt, ontstaat een race-conditie. De manier om dit op te lossen, is door in gebruikerscontext eerst de software-onderbrekingen uit te schakelen en daarna het slot te vergrendelen. Deze methode wordt dan ook toegepast in de code van de ethernet-protocolfilter. Op deze manier kan volgende situatie vermeden worden:

```
Gebruikerscontext       Software-onderbreking (SO)
-----------------       --------------------------
neem db-schrijfslot
--Gebruikerscontext wordt onderbroken voor de SO--
                        probeer (neem  db-leesslot)
                        ...
```

Dit is een probleemgeval dat 'patstelling' wordt genoemd. Aangezien het slot een spinslot betreft, zal de software-onderbreking constant blijven proberen om het databankslot te krijgen. De kern zal hangen aangezien een software-onderbreking niet door een gebruikersprogramma kan worden onderbroken: de gebruiker heeft de controle over de computer verloren.

Slotvergrendeling tussen SO's

Dezelfde SO kan tegelijkertijd op verschillende processoren worden uitgevoerd. Deze gebruiken dan gemeenschappelijke data (bv. de databank in brepf) en dus moeten ze gebruik maken van een spinslot om niet tezelfdertijd in de kritische sectie te komen. Deze werkwijze is natuurlijk wel zeer ongunstig voor smp's aangezien de ene processor niets kan doen terwijl hij wacht op het vrijgeven van het slot door een andere processor. Hiervoor bestaat echter een oplossing, die ook gebruikt wordt in brepf (voor de databank en de tellers).
Nemen we het voorbeeld van de databank. Elke processor zal zijn eigen databank bijhouden van de soort pakketjes die deze processor moet verwerken. Een schrijfslot is niet meer nodig, in de plaats krijgen we een leesslot. Wan-

neer we aan het gebruikersprogramma de volledige databank willen geven, zullen we dan een schrijfslot nemen (in gebruikerscontext) zodat niets meer kan bijgevoegd worden en zullen de databanken van alle processoren tot één databank worden teruggebracht, die dan wordt gegeven aan het gebruikersprogramma.

8.5 De werking van het gebruikersprogramma brepf

8.5.1 Communicatie tussen het gebruikersprogramma en de kern

Voordat het programma data kan veranderen of op het scherm weergeven, zal het deze data moeten verkrijgen van de kern. Daarvoor dient gecommuniceerd te worden met de kern. Dit gebeurt via een fitting, alhoewel geen connectie tot stand wordt gebracht zoals in de klassieke netwerkfittingmanier. De fitting (*socket*) wordt als volgt aangemaakt:

```
sockfd = socket(AF_INET, SOCK_RAW, PF_INET);
```

Eenmaal de fitting is gedefinieerd, gebeurt de communicatie via de systeemaanroep `getsockopt()`. Laten we deze systeemaanroep van dichterbij bekijken.

Het prototype ziet er als volgt uit:

```
int getsockopt(int fitting, int niveau, int optie_naam,
     void *optie_waarde, socklen_t *optie_lengte);
```

Bespreking van de argumenten:

- **fitting**: De fitting die we zonet gedefinieerd hebben.
- **niveau**: Getsockopt wordt veelvuldig gebruikt voor communicatie tussen de kern en gebruikersprogramma's. Het niveau geeft aan met welk deel van de kerncode we wensen te praten, bij ons zal dit niveau IPPROTO_IP zijn. De netfilter-code (zie hoofdstuk 6) zit op het ip-niveau ingeschakeld, dus gebruiken we dit niveau. De brepf-code werkt

eigenlijk niet op het ip-niveau, maar aangezien het logisch is via netfilter te communiceren met de kern moeten we toch dit IPPROTO_IP-niveau gebruiken.

- **optie_naam**: Per niveau moeten natuurlijk verschillende boodschappen kunnen verzonden worden: deze parameter dient hiervoor. Iptables gebruikt bv. andere optienummers dan deze die gebruikt worden door de brepf-code. Aangezien de brepf- en iptables-code deze opties moeten registreren bij de netfilter-code (in hun initialisatiecode), zal de netfilter-code weten welke code moet opgeroepen worden.

- **optie_waarde**: Deze parameter dient vooral om het adres mee te delen waar de kern de gevraagde informatie moet opslaan. Bijkomende opties kunnen via deze wijzer ook meegegeven worden, maar dit gebeurt toch beter via de `optie_naam`-parameter.

- **optie_lengte**: Hiermee delen we de kern mee hoeveel geheugen het programma heeft gealloceerd voor de te ontvangen data. De kern kan deze waarde veranderen indien niet zoveel ruimte nodig was.

De aanroep van `getsockopt()` is een systeemaanroep en daardoor komt de uitvoerende processor in gebruikerscontext terecht (zie punt 8.4.5).
Naast `getsockopt()` bestaat ook nog de systeemaanroep `setsockopt()`. Deze aanroep wordt gebruikt indien het programma data wil geven aan de kern. Dit gebeurt wanneer de gebruiker een verandering in de ketting wil aanbrengen of wanneer hij/zij de tellers of de databank wil aan- of uitschakelen. Deze aanroep gebeurt analoog.

8.5.2 Het gebruikersprogramma vraagt data aan de kern

We onderscheiden twee gevallen. Als eerste geval kunnen we de tabeldata opvragen. Dit gebeurt in twee stappen: eerst wordt de te alloceren geheugengrootte aan de kern gevraagd. Daarna wordt het geheugen gealloceerd en wordt de eigenlijke data aan de kern gevraagd. Indien geen tellers worden

gebruikt bestaat deze data uit de tabel met haar kettingen. Indien de tellers zijn aangeschakeld zal nog ruimte moeten gealloceerd worden voor deze tellers (één teller per regel).
Het tweede geval betreft het commando om de databank op het scherm weer te geven. Dit gebeurt ook in twee stappen. In oudere netwerken kan het zijn dat het protocolveld van het ethernet-frame (zie figuur 8.1) wordt gebruikt om de lengte van het dataveld aan te duiden. In plaats van deze lengte te beschouwen als een protocol, zal de boodschap 'NO PROTO, OLD 802.3 STYLE LENGTH FIELD' op het scherm verschijnen in het protocolveld.

8.5.3 Het gebruikersprogramma maakt de veranderingen

Het programma krijgt de data van de kern zoals deze daar zijn opgeslagen. Het moet dus exact weten hoe de kern deze data opslaat. De structuur van de data die het programma ontvangt, wordt weergegeven in figuur 8.3. De data die het teruggeeft aan de kern moet dezelfde structuur hebben.

Indien de tellers niet gebruikt worden, zal de laatste deelstructuur natuurlijk niet aanwezig zijn. Het is de taak van het gebruikersprogramma om ervoor te zorgen dat deze structuur consistent blijft. Elke inconsistentie zal door de kern worden opgemerkt en deze zal de data negeren. Tussen de kerndatastructuur en deze van het gebruikersprogramma zijn twee verschillen. In de kern bevinden de tellers zich niet noodzakelijk direct na de tabeldata en zal de structuur van de tellers er bij smp-machines ook anders uitzien (zie verder). Daarnaast zal de eerste deelstructuur zich niet vóór de tabelgegevens bevinden (dit zijn natuurlijk details). De structuur van één regel wordt gegeven in figuur 8.4. Een teller bestaat uit een 64-bit natuurlijk getal.

Wanneer we met tellers werken, moeten we ervoor zorgen dat de tellers blijven kloppen wanneer we de tabel veranderen. Het volgende probleem doet zich voor. Stel dat we bv. een regel willen toevoegen; hiervoor moeten we eerst de tabelgegevens samen met de tellers opvragen. Nadat deze zijn opgevraagd, kunnen de tellers in de kern nog verhoogd worden doordat nieuwe pakketjes overeenkomen met een regel. Het gebruikersprogramma heeft hier

geen weet van. Daarom zal de kern, nadat hij de gegevens heeft verzonden, zijn tellers op nul zetten. Wanneer het programma de nieuwe tabel teruggeeft, geeft het ook de oude tellers mee terug. De juiste tellers worden dan bekomen door de som van de oude tellers en de waarde van de tellers die daarnet op nul waren gezet. Indien we op deze manier werken, zou de kern moeten weten welke veranderingen aan de tabelgegevens zijn gebeurd, omdat hij moet weten welke tellers bij elkaar moeten worden opgeteld. Aangezien we willen dat enkel het gebruikersprogramma zich bezighoudt met de veranderingen, zullen we in drie stappen moeten werken indien de tellers gebruikt worden.

- **Stap 1**: De tabel en de tellers worden aan het gebruikersprogramma gegeven. De kern blijft met deze tabel werken en zet de tellers niet op nul.

- **Stap 2**: De kern krijgt de nieuwe data binnen. Hij zal de nieuwe tabel gebruiken en de tellers ervan worden op nul gezet. De oude tellers (van de oude tabel) worden aan het programma gegeven en daarna verwijderd uit het geheugen.

- **Stap 3**: Het programma zal een nieuwe tellerstructuur maken voor de nieuwe tabel waarbij de tellers die het heeft teruggekregen van de kern worden geïntegreerd. Deze nieuwe tellerstructuur wordt teruggegeven aan de kern. Nu kan de kern deze tellers optellen bij de tellers die hijzelf had aangemaakt (en op nul had gezet) zonder over extra informatie te moeten beschikken.

Op deze manier moet de kern zich niets aantrekken van hetgeen veranderd is in de tabel en blijven de tellers consistent. Wanneer geen tellers gebruikt worden, zullen de veranderingen in twee stappen kunnen gebeuren (de tabel van de kern ontvangen en de gewijzigde tabel naar de kern terugsturen).
Indien de maximale waarde van een teller wordt bereikt, zal een overloop optreden waardoor de teller op nul wordt gezet. Behalve de inconsistentie brengt dit geen verdere problemen met zich mee.

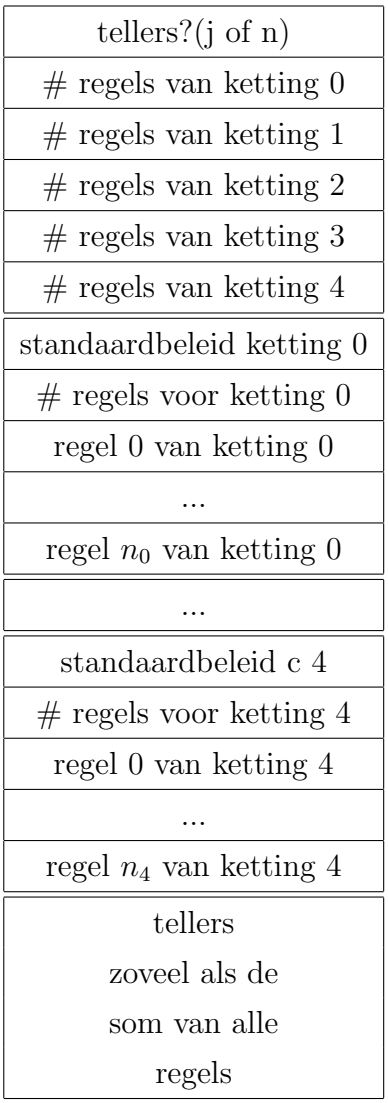
8.6 De werking van de brepf-kerncode

Tijdens de initialisatie haakt de code zich vast aan de vijf brughaken en registreert zich bij netfilter voor de communicatie met gebruikersprogramma's. De tabel wordt geïnitialiseerd met lege kettingen die allen het standaardbeleid ACCEPT hebben. De twee gebruikte lees/schrijf-spinsloten worden ook geïnitialiseerd. De databank en tellers worden standaard uitgeschakeld.
De kerncode bestaat uit twee delen, enerzijds de code die uitgevoerd wordt in een SO en anderzijds de code die in gebruikerscontext wordt uitgevoerd. Zoals gezegd kan een SO de gebruikerscontext preëmptief onderbreken, daarom moeten we de kritieke secties van de gebruikerscontext beschermen met een lees/schrijf-spinslot nadat eerst de software-onderbrekingen zijn uitgeschakeld.
Het gebruikerscontextgedeelte bestaat uit de functies die de `getsockopt()` en `setsockopt()` systeemoproepen afhandelen, samen met de functies die door deze twee functies worden opgeroepen.
Er wordt gebruik gemaakt van twee sloten: één om de tabel te beschermen en één om de databank te beschermen. In gebruikerscontext wordt voor de databank altijd een schrijfslot genomen, ook indien we enkel willen lezen. Volgende situatie zou zich anders bv. kunnen voordoen: (met een leesslot i.p.v. een schrijfslot)

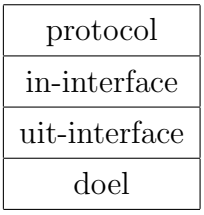
```
processor 0(SO)                          processor 1(gebr.context)
--------------                           -------------------
leesslot(databank)                       leesslot(databank)
we zien dat de databank vol is
alloceer meer geheugen voor db
kopieer oude db naar nieuwe db           lees plaats van oude db
geef geheugen oude db vrij
                                         kopieer db naar programma
verwijs naar nieuwe geheugen
```

Merk op dat, zoals gezegd, bij een uniprocessor deze sloten niet bestaan en het dus ook niet uitmaakt of we een lees- of schrijfslot gebruiken. We zien

dat de gebruikerscontext geheugen kopiëert dat niet meer gealloceerd is. Wat zich op die geheugenplaatsen bevindt, is onvoorspelbaar.
Voor het tabelslot dienen we in gebruikerscontext ook altijd een schrijfslot aan te vragen. Wanneer we de tabelgegevens wensen, moeten we een schrijfslot nemen omdat de tellers ook worden meegekopiëerd en deze anders kunnen veranderd worden (door de SO), terwijl we aan het kopiëren zijn. Indien de tellers niet gebruikt worden, hebben we eigenlijk genoeg aan een leesslot. Iindien we deze uitzondering willen maken, moeten we echter met drie sloten werken, wat niet de moeite loont. Dit zou nl. de SO-code trager maken (aangezien het twee leessloten moet nemen om de tellers te kunnen verhogen). Enkel de situatie met de gebruikerscontext zoals daarnet besproken zou sneller werken. Deze situatie komt relatief gezien echter bijna nooit voor aangezien de gebruiker niet constant informatie aan de kern vraagt.
In de SO's zijn enkel leessloten nodig. Aangezien gebruikerscontext haast nooit wordt uitgevoerd (enkel indien de gebruiker het gebruikersprogramma gebruikt) geeft dit een efficiënte slotvergrendelingstechniek.
De reden waarom we in de software-onderbrekingen enkel met leessloten moeten werken, is omdat we de data die wordt veranderd, opsplitsen per processor (in het jargon per-processor data genoemd). Dit gebeurt voor de databank en de tellers, aangezien deze kunnen worden veranderd door de SO-code. De eigenlijke tabel is wel gemeenschappelijk want deze wordt nooit veranderd in de SO-code. Aangezien we nu zeker zijn dat een andere processor geen toegang heeft tot de data die we willen veranderen, hebben we genoeg aan een leesslot. Een leesslot is zeker nodig, aangezien een andere processor bezig kan zijn met het uitvoeren van de gebruikerscontextcode van brepf.



Figuur 8.3: Datastructuur voor het gebruikersprogramma



Figuur 8.4: Regelstructuur

Hoofdstuk 9

Inkadering in het grotere geheel

Naast de protocolonafhankelijke firewall-software brepf is er meestal ook nood aan een protocolspecifieke firewall. Voor Linux bestaat hiervoor, zoals gezegd, iptables. Om deze software te kunnen gebruiken op een Linux-brug moeten we een methode verzinnen om de ip-haken te koppelen aan de haken die gedefinieerd zijn voor de brug. De code die hiervoor zorgt is geschreven door Lennert Buytenhek, die tevens ook de auteur is van de brugcode en de haken voor de brug. Deze code maakt geen deel uit van de standaardkern van Linux, maar ze kan via een kernaanpassing gebruikt worden.

9.1 Koppeling van de ip-haken aan de brughaken

9.1.1 Een eerste blik op het probleem

Aangezien de pakketten de brughaken op dezelfde manier doorlopen als de ip-pakketten de ip-haken doorlopen, is er voor elke brughaak een equivalente ip-haak.
Met de kennis die we opgebouwd hebben over netfilter kunnen we een makkelijke techniek bedenken voor de koppeling. In meta-taal ziet onze methode er als volgt uit:

1. ethernet-frame passeert 1 van de 5 brughaken
2. de netfilter-architectuur ontvangt dit pakket en geeft het door aan onze functie (die op alle hakken is vastgehaakt);
3. onze functie ziet dat het frame door het ip-protocol is aangemaakt en emuleert de equivalente ip-haak;
4. de netfilter-architectuur wordt terug opgeroepen. Nu zal ze de functies die vastgehaakt zijn op de ip-haak oproepen;
5. netfilter handelt hun beslissing af en geeft de beslissing terug aan onze functie;
6. onze functie geeft de beslissing NF_STOLEN terug aan netfilter;
7. netfilter ontvangt NF_STOLEN en zal niets meer aanvangen met het pakket.

De werkelijke C-code van ons algoritme ziet er als volgt uit:

```
static unsigned int br_hook (unsigned int hook,  struct sk_buff
 **pskb, const struct net_device *in, const struct net_device
 *out, int (*okfn)(struct sk_buff *)) {
  if ((*pskb)->mac.ethernet->h_proto == htons(ETH_P_IP)) {
    NF_HOOK(PF_INET, hook, *pskb, (struct net_device *)in,
    (struct net_device *)out, okfn);
    return NF_STOLEN;
  }
  if ((*pskb)->mac.ethernet->h_proto == htons(ETH_P_IPV6)) {
    NF_HOOK(PF_INET6, hook, *pskb, (struct net_device *)in,
    (struct net_device *)out, okfn);
    return NF_STOLEN;
  }
  return NF_ACCEPT;
}
```

Deze functie wordt aangeroepen door de netfilter-architectuur wanneer een pakket een brughaak passeert. Bij de initialisatie van de passthrough-software zal deze functie dus worden vastgehaakt aan de brughaken. Zoals reeds eerder gezegd (zie hoofdstuk 6) gebeurt dit door de functie bij de netfilter-architectuur te registreren op deze haken.
`NF_HOOK()`, NF_STOLEN en NF_ACCEPT werden ook besproken in hoofdstuk 6. Uit de C-code is af te leiden dat ethernet-pakketten met als typeveld ip versie 4 (ETH_P_IP) of ip versie 6 (ETH_P_IPV6) verder zullen afgehandeld worden door de ip-code. Dit aangezien als protocolfamilie PF_INET of PF_INET6 wordt gebruikt bij de aanroep van `NF_HOOK()`. Deze twee waarden kunnen gelezen worden als protocolfamilie internet versie 4 resp. versie 6. De waarde van deze parameter zal, voor de aanroepen van `NF_HOOK()` in de brugcode, gelijk zijn aan PF_BRIDGE. We zien dus dat we op deze manier de netfilter-code doen geloven dat het pakket een ip-haak heeft gepasseerd. Hierdoor zal netfilter de software vastgehaakt aan deze ip-haken uitvoeren. Ook de ip-filter zal dus dit pakketje ontvangen.
In beide gevallen (ip versie 4 of ip versie 6) wordt als beslissing NF_STOLEN teruggegeven aan netfilter, waardoor netfilter dit pakket zal negeren (maar niet verwijderen).
Wat dit misschien wat ingewikkeld maakt is dat netfilter twee maal wordt opgeroepen (via `NF_HOOK()`). De eerste keer gebeurt dit in de brugcode, de tweede keer in onze functie (waarvan we de code daarnet getoond hebben). De tweede maal dat netfilter wordt opgeroepen zal de code de functies die op de ip-haak vastgehaakt zijn, uitvoeren. Netfilter zal hier ook de beslissing van deze functies afhandelen. Aangezien deze afhandeling hier al is gebeurd, moet gezorgd worden dat netfilter deze afhandeling geen tweede maal doet wanneer teruggekeerd wordt naar de eerste oproep van netfilter. Hiervoor wordt gezorgd door de beslissing NF_STOLEN.
Alle pakketten die niet van het ip-protocol afkomstig zijn worden door onze code genegeerd door als beslissing NF_ACCEPT terug te geven aan de netfilter-code.
Het is duidelijk dat deze software, genaamd *passthrough*, zeer compact is.
De netfilterprioriteit van de passthrough-functie is 0, terwijl de prioriteit van

de brepf-functie -200 is. Netfilter zal dus eerst de brepf-code uitvoeren, wat de logische volgorde is.

9.1.2 Complicaties

Voor ethernet-frames die door de ethernet-brug rechtstreeks worden doorgezonden werkt de vorige methode perfect. Indien het bestemmings-MAC-adres van het ethernet-frame echter dit van de interface is waarop het frame binnenkwam, krijgen we complicaties. Deze complicaties worden veroorzaakt door de manier waarop deze ethernet-frames worden doorgegeven aan de netwerkcode. De ethernet-code weet nl. niet hoe het lokale frames moet verwerken, daarom zal het deze frames injecteren in de algemene netwerkcode. Dit injecteren houdt in dat de code dit frame in de juiste wachtlijn zal plaatsen. Laten we een ethernet-frame bekijken dat als typeveld het ip-protocol heeft (versie 4). Dit frame komt de computer binnen en wordt geleverd aan de brugcode. Het pakket passeert eerst de PRE_ROUTING-brughaak en daarna de LOCAL_IN-brughaak. Daarna wordt het pakket geïnjecteerd in de netwerkcode. Deze netwerkcode ziet dat ze code bezit voor het afhandelen van ip-pakketten en geeft dit frame door aan deze code. Het frame (dat we nu ip-pakket kunnen noemen) passeert de PRE_ROUTING-ip-haak en dan de ip-routeringscode. Aan de hand van het bestemmings-ip-adres zal het pakket verder gezonden worden, ofwel aan de lokale computer gegeven worden. De volgende haak dat het pakket passeert is dus ofwel de FORWARD-haak, ofwel de LOCAL_IN-haak.
In beide gevallen klopt iets niet aan de manier waarop we werken: ofwel wordt zowel de INPUT-ketting als de FORWARD-ketting opgeroepen, ofwel wordt de INPUT-ketting van de filtertabel van iptables twee maal opgeroepen (wat de tellers dus 2 maal verhoogt). De oplossing voor dit probleem is om in de functie `br_hook()` van de passthrough-software (zie vorig punt) enkel de frames naar de ip-code te sturen indien ze de PRE_ROUTING-, POST_ROUTING- of FORWARD-haak passeren. De functie ziet er dan als volgt uit:

```
static unsigned int br_hook (unsigned int hook,  struct sk_buff
 **pskb, const struct net_device *in, const struct net_device
 *out, int (*okfn)(struct sk_buff *)) {
  if (hook == NF_BR_LOCAL_IN || hook == NF_BR_LOCAL_OUT)
    return NF_ACCEPT;
  if ((*pskb)->mac.ethernet->h_proto == htons(ETH_P_IP)) {
    NF_HOOK(PF_INET, hook, *pskb, (struct net_device *)in,
    (struct net_device *)out, okfn);
    return NF_STOLEN;
  }
  if ((*pskb)->mac.ethernet->h_proto == htons(ETH_P_IPV6)) {
    NF_HOOK(PF_INET6, hook, *pskb, (struct net_device *)in,
    (struct net_device *)out, okfn);
    return NF_STOLEN;
  }
  return NF_ACCEPT;
}
```

Merk op dat we de ip-software vastgehaakt op de PRE_ROUTING-, FORWARD- en POST_ROUTING-brughaken wel wensen uit te voeren aangezien deze haken worden gepasseerd wanneer de brug een frame in een netwerk verstuurt. Om dit te doen heeft de brugcode genoeg informatie en zal ze het frame niet moeten injecteren in de netwerkcode. Indien we deze brughaken niet zouden verbinden met de ip-haken, zouden we dus geen ip-filtering doen. Hiermee is echter nog niet alles opgelost. De aandachtige lezer zal gemerkt hebben dat de PRE_ROUTING-ip-haak twee maal worden gepasseerd wanneer het bestemmings-MAC-adres dit van de ethernet-kaart is. Voor de code die connecties bijhoudt is dit geen probleem, deze code is slim genoeg om niet twee maal hetzelfde te doen op een pakket. De nav-code is echter niet zo slim en zal dus niet werken. Voor de POST_ROUTING-ip-haak gelden gelijkaardige problemen.
Voor het elis-netwerk is dit geen probleem aangezien nav niet wordt gebruikt. Momenteel wordt nog overlegd met Lennert Buytenhek hoe dit probleem kan opgelost worden. Waarschijnlijk zal dit eenvoudig gebeuren door extra code

in de netfilter-code te voorzien zodat frames (of pakketten, zoals u wilt) die een tweede maal op dezelfde haak passeren, en indien de reden hiervoor de brugcode is, geen tweede maal worden gegeven aan de software vastgehaakt op die haak.

Hoofdstuk 10

Resultaten en besluit

10.1 Samenvatting

Voor deze thesis werd een grondige studie verricht over de werking van pakketfiltering in Linux. Daarna werd een ethernet-pakketfilter ontworpen. Deze software werd smp-vriendelijk gemaakt (zie punt 8.6). Als laatste stap werd samengewerkt met Lennert Buytenhek om ip-pakketfiltering op een brug mogelijk te maken (zie hoofdstuk 9).
Door deze handelingen werd de mogelijkheid gecreëerd om een transparante firewall op een brug te gebruiken, die ip-connecties kan bijhouden en waarop ethernet-pakketfiltering en ip-pakketfiltering kan gebeuren.

10.2 Uitgevoerde testen

Zowel de brepf- als de passthrough-code zijn stabiel en werken ook op een smp. Beide werden grondig getest. In een eerste fase in een klein netwerk met aan de ene kant van de firewall slechts 1 computer en aan de andere kant het elis-netwerk. In de laatste fase werd de firewall uitgerust met de firewall-regels van de elis-firewall, waarna onze firewall dienst deed als de elis-firewall. De elis-firewall (die draait op de computer linus) werd dus vervangen door onze firewall (die draait op de computer bach). Hiervoor nam bach dus de plaats in het netwerk in, waarop linus zich normaal gezien bevindt. De brepf-software werd hierin natuurlijk ook betrokken: de tellers en databank werden

geactiveerd en er werden brepf-regels gedefinieerd om betere bescherming te bieden (zie bijlage B). Deze fase kan aanzien worden als een ultieme test: indien de computer crasht, crasht het complete elis-netwerk. De stroom van pakketten die de computer dient te verwerken, is in deze configuratie aanzienlijk. Indien er iets mis is met de code, is de kans dat we dit met deze configuratie te weten komen, groot. Onze firewall heeft al een aantal dagen op deze manier in het elis-netwerk gefunctioneerd.
De inhoud van de brepf-databank voor het elis-netwerk wordt in bijlage A gegeven.
De performantie van de firewall werd ook gemeten. We hebben twee netwerkconfiguraties gemeten:

- Twee computers, met elkaar verbonden via onze firewall op een brug. In deze configuratie draaide de brepf-software samen met de passthrough-software op de firewall-computer. De tellers en de databank van de brepf-software waren aangeschakeld, en er bevonden zich 3 regels in de FORWARD-ketting (zie bijlage B). Deze regels zorgden ervoor dat enkel ethernet-pakketten met typeveld ip versie 4, ip versie 6 en ARP (een protocol gebruikt in ethernet-netwerken) werden doorgelaten. Het beleid van de FORWARD-ketting werd op DROP gezet. De ip-firewall-regels waren deze die gebruikt worden door de elis-firewall.

- In de tweede configuratie gebruikten we ook drie computers, maar op de brug werd geen firewall geïnstalleerd. We gebruikten de originele brugcode die zich in de standaardkern bevindt. Er waren dus geen haken gedefinieerd in de brugcode.

We hebben gebruik gemaakt van het programma tcpblast. Hiermee kunnen we de bandbreedte van een netwerk meten. Dit programma stuurt een aantal tcp-pakketten met een bepaalde grootte naar een andere computer in het netwerk. Door te meten hoe lang het duurt totdat alles is verzonden vinden we een maat voor de bandbreedte.
Bij elke configuratie werden door tcpblast 9999 pakketten met grootte variërend van 1 KB tot 6 KB verstuurd. De resultaten worden in tabel 10.1

weergegeven. We zien dat het verschil minder dan 1 procent bedraagt. De firewall heeft dus geen grote invloed op de bandbreedte.

pakketgrootte	zonder firewall (KB/s)	met firewall (KB/s)
1 KB	11414.38	11362.50
2 KB	11466.74	11463.32
3 KB	11457.98	11456.74
4 KB	11453.61	11446.89
5 KB	11453.61	11450.98
6 KB	11453.61	11447.98
gemiddelde	11449.99	11438.07

Figuur 10.1: Bandbreedtevergelijking

10.3 Bespreking gebruikte brepf-regels voor het elis-netwerk

Uit bijlage B blijkt dat we maar drie protocollen doorlaten: ip versie 4, ip versie 6 en ARP. Ip versie 6 wordt eigenlijk nog niet gebruikt in het elis-netwerk, maar in de toekomst is het de bedoeling dat netwerken ip versie 6 zullen gebruiken. Het ARP-protocol wordt gebruikt om ip-adressen om te zetten in MAC-adressen (de ethernet-adressen), m.a.w. om netwerklaagadressen om te zetten in datalinklaagadressen. Communicatie binnen het elis-netwerk gebeurt nl. met de netwerklaagadressen (zie punt 1.1). De ip-adressen zijn maar belangrijk voor communicatie met computers buiten het elis-netwerk gelegen. Netwerkprogramma's werken echter altijd met ip-adressen, zodat een conversie moet gemaakt worden naar de MAC-adressen m.b.v. het ARP-protocol. Behalve deze drie protocollen laat de firewall niets door.
In bijlage A komen we nog drie andere waarden tegen voor het protocol:

- NO PROTO, OLD 802.3 STYLE LENGTH FIELD: in punt 8.1 werd gezegd dat het type-veld (wat we hier als protocolwaarde gebruiken)

ook de lengte van het dataveld van het frame kan voorstellen. Wanneer dit zo is worden deze lengtes aanzien als hetzelfde 'protocol' en wordt met deze zin aan de gebruiker gemeld dat het om een lengteveld gaat.

- 9000 en 9dc1: de bron van deze ethernet-frames is niet bekend en het is ook niet geweten welke protocollen deze getallen voorstellen. Om de bron van deze frames te weten te komen, moet er een logfaciliteit aanwezig zijn in de brepf-code. Deze faciliteit is momenteel niet aanwezig, maar zal in de toekomst worden geïmplementeerd.

Er werd besloten deze twee onbekende protocollen niet door te laten. Aangezien geen klachten hieromtrent werden ontvangen, kunnen we stellen dat het geen kwaad kan deze protocollen niet door te laten.

10.4 Beperking

Uit de praktijk blijkt dat het heel handig zou zijn indien de mogelijkheid werd voorzien om te loggen. Men zou dan kunnen zorgen dat, telkens aan een regel uit een ketting wordt voldaan, een log wordt gemaakt van de relevante informatie over dat frame (natuurlijk enkel indien de gebruiker dit wenst voor deze regel). Deze log wordt geschreven in een bestand dat hiervoor dient: `/var/log/messages`. In deze log wordt dan, naast de informatie die ook te zien is in de databank, ook het bron- en bestemmings-MAC-adres van het frame gezet. Door in de logs te kijken kan men dan achterhalen welke computers uit het netwerk bepaalde protocollen versturen.

Bijlage A

brepf-databank voor het elis-netwerk

Hier vindt u de volledige brepf-databank zoals deze eruit zag wanneer deze firewall dienst deed als de echte elis-firewall.

```
number of entries: 34
1:  hook: PRE_ROUTING  in-if: eth1 out-if:
    protocol: ARP
2:  hook: LOCAL_IN     in-if: eth1 out-if:
    protocol: ARP
3:  hook: FORWARD      in-if: eth1 out-if: eth0
    protocol: ARP
4:  hook: POST_ROUTING in-if:      out-if: eth0
    protocol: ARP
5:  hook: PRE_ROUTING  in-if: eth1 out-if:
    protocol: IPV4
6:  hook: FORWARD      in-if: eth1 out-if: eth0
    protocol: IPV4
7:  hook: POST_ROUTING in-if:      out-if: eth0
    protocol: IPV4
8:  hook: LOCAL_IN     in-if: eth1 out-if:
    protocol: 9dc1
```

```
9:  hook: FORWARD      in-if: eth1 out-if: eth0
    protocol: 9dc1
10: hook: LOCAL_IN     in-if: eth1 out-if:
    protocol: IPV4
11: hook: PRE_ROUTING  in-if: eth1 out-if:
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
12: hook: LOCAL_IN     in-if: eth1 out-if:
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
13: hook: FORWARD      in-if: eth1 out-if: eth0
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
14: hook: POST_ROUTING in-if:      out-if: eth0
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
15: hook: PRE_ROUTING  in-if: eth0 out-if:
    protocol: IPV4
16: hook: FORWARD      in-if: eth0 out-if: eth1
    protocol: IPV4
17: hook: POST_ROUTING in-if:      out-if: eth1
    protocol: IPV4
18: hook: PRE_ROUTING  in-if: eth0 out-if:
    protocol: ARP
19: hook: FORWARD      in-if: eth0 out-if: eth1
    protocol: ARP
20: hook: POST_ROUTING in-if:      out-if: eth1
    protocol: ARP
21: hook: PRE_ROUTING  in-if: eth0 out-if:
    protocol: 9000
22: hook: LOCAL_IN     in-if: eth0 out-if:
    protocol: ARP
23: hook: LOCAL_IN     in-if: eth0 out-if:
    protocol: 9dc1
24: hook: FORWARD      in-if: eth0 out-if: eth1
    protocol: 9dc1
25: hook: LOCAL_IN     in-if: eth0 out-if:
```

```
    protocol: IPV4
26: hook: LOCAL_OUT    in-if: br0  out-if: eth0
    protocol: IPV4
27: hook: LOCAL_OUT    in-if: br0  out-if: eth1
    protocol: IPV4
28: hook: LOCAL_OUT    in-if: br0  out-if: eth1
    protocol: ARP
29: hook: LOCAL_OUT    in-if: br0  out-if: eth0
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
30: hook: LOCAL_OUT    in-if: br0  out-if: eth1
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
31: hook: POST_ROUTING in-if:      out-if: eth1
    protocol: NO PROTO, OLD 802.3 STYLE LENGTH FIELD
32: hook: LOCAL_OUT    in-if: br0  out-if: eth0
    protocol: ARP
```

Bijlage B

brepf-regels voor de FORWARD-ketting

De situatie in de FORWARD-ketting wanneer de nieuwe firewall als elis-firewall dienst deed.

```
Bridge hook: FORWARD
Policy: DROP
nr. of entries: 3
eth proto: ARP, in-if: , out-if: , target: ACCEPT, count = 51344
eth proto: IPV4, in-if: , out-if: , target: ACCEPT, count = 21246
eth proto: IPV6, in-if: , out-if: , target: ACCEPT, count = 0
```

Bibliografie

[1] S. DEWINTER, Firewalls, Universiteit Gent, Faculteit Toegepaste Wetenschappen, Vakgroep Electronica en informatiesystemen, (2000)

[2] C. BRENTON, Mastering Network Security, SYBEX Inc., (1999)

[3] W. CHESWICK, S. BELLOVINN, Firewalls and Internet Security, Addison-Wesley, (1994)

[4] D. CHAPMAN, E. ZWICKY, Building Internet Firewalls, O'Reilly, (1995)

[5] The netfilter project homepage, http://antarctica.penguincomputing.com/~netfilter/

[6] R. CARD, E. DUMAS, F. MEVEL, the Linux kernel book, Wiley, (1998)

[7] S. MAXWELL, Linux Core Kernel Commentary, CoriolisOpen Press, (1999)

[8] M.BECK, H. BOHME, M. DZIADZKA, U. KUNITZ, R. MAGNUS, D. VERWORNER, Linux kernel internals (second edition), Addison-Wesley, (1998)

[9] W. STALLINGS, Operating Systems: achtergronden, werking en ontwerp, Academic Service, (1999)

[10] De Linux bridge homepage, http://bridge.sourceforge.net/

[11] Paul Russel, Packet filtering howto,
http://antarctica.penguincomputing.com/ netfilter/

[12] primaire site voor de linuxkern broncode, http://www.kernel.org/

[13] Harald Welte, The journey of a packet through the linux 2.4 network stack, //http://www.gnumonks.org/ftp/pub/doc/packet-journey-2.4.html

[14] Paul Russel, Kernel locking guide