Datavetenskap

Tomas Nilsson & Fredrik Sanamon

Fjärradministration av Iptables

Management Server

Examensarbete, C-nivå

2003:04

Fjärradministration av Iptables Management Server

Tomas Nilsson & Fredrik Sanamon

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Tomas Nilsson & Fredrik Sanamon

Godkänd, 2003-01-15

Handledare: Stefan Alfredsson

Examinator: Stefan Alfredsson

Sammanfattning

Detta dokument beskriver ett examensarbete inom datavetenskap på C-nivå vid Karlstads universitet. Arbetet bestod i att vidareutveckla ett system som påbörjats i tidigare examensarbeten på Karlstads universitet utförda vårterminen 2002. Projektet syftar till att skapa en objektorienterad miljö för fjärradministration av brandväggsfunktionen i Linux, Netfilter/Iptables. Dokumentet innefattar en översiktlig beskrivning av brandväggar i allmänhet samt en mera ingående beskrivning av Netfilter/Iptables. Vi inför en objektorienterad syn på beståndsdelarna i ett regelverk för Iptables samt implementerar en managementserver och en prototyp av ett grafiskt gränssnitt som använder denna objektorienterade syn för att skapa och hantera Linux-brandväggen Iptables. Denna konstruktionslösning samt implementation utgör huvuddelen av rapporten.

Remote administration of Iptables

Management server

Abstract

This document describes a bachelor's project within computer science at the university of Karlstad. The assignment consisted of further development of a system developed in prior bachelor's projects at the university of Karlstad, spring term of 2002. The purpose of the project is to create an object oriented environment for remote administration of Linux Netfilter/Iptables. The document contains a summary of firewalls in general and a more detailed description of Netfilter/Iptables. We introduce an object oriented view of the components of an Iptables ruleset. We implement a management server and a prototype of a graphical user interface that uses this objectoriented view to create and manage the Linux firewall Iptables. The construction and implementation consitiute the main part of the document.

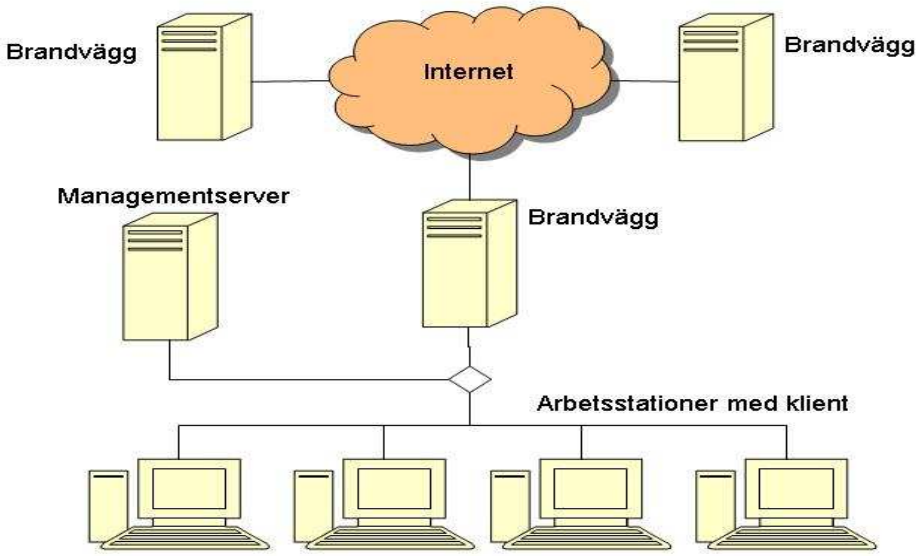
Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Detta examensarbete utfördes hösten 2002 i samarbete med Veriscan Security AB och bygger på tidigare examensarbeten utförda våren 2002. Projektet syftar till att underlätta hantering av den i Linux inbyggda brandväggen Netfilter/Iptables genom att ge möjlighet till fjärrstyrning av brandväggen. Systemet skall bestå av en grafisk klient och en server som handhar lagring av information om brandväggar, brandväggskonfigurationer och övriga noder. Figur 1:1 ger en översiktsbild av systemet.

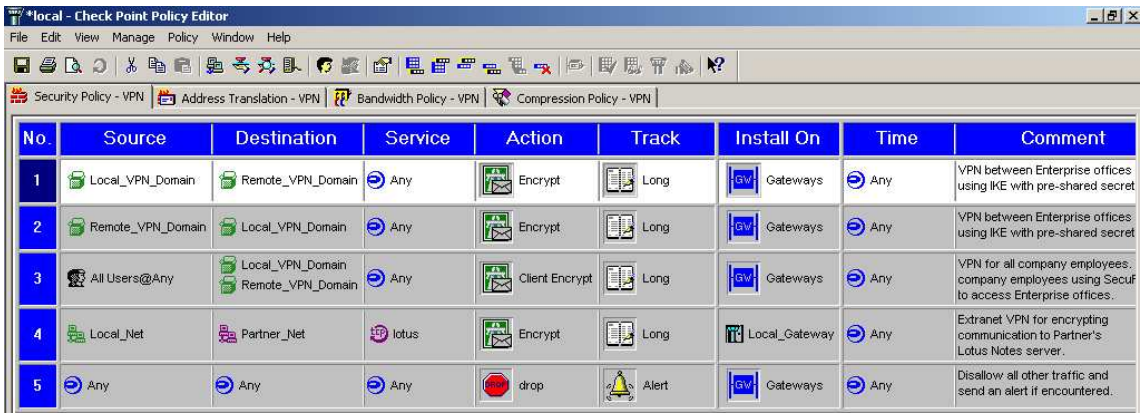


Figur 1:1 Översiktsbild av systemet

Systemet hanterar även kommunikationen mellan klient och berörda brandväggar. Klient och server skall kunna köras på olika plattformar och därför har tidigare delar av projektet implementerats i programspråket Java. Resten av rapporten är strukturerad enligt följande; kapitel 2 innehåller examensarbetets bakgrund, kapitel 3 ger en allmän introduktion till brandväggar. För att skapa mer förståelse för vårt arbete så innehåller kapitel 4 en ingående beskrivning av Linux inbyggda brandvägg Netfilter/Iptables. I kapitel 5 beskrivs vår konstruktionslösning och kapitel 6 innehåller en beskrivning av implementationen. Kapitel 7 innehåller kommentarer om arbetet och i kapitel 8 sammanfattas rapporten.

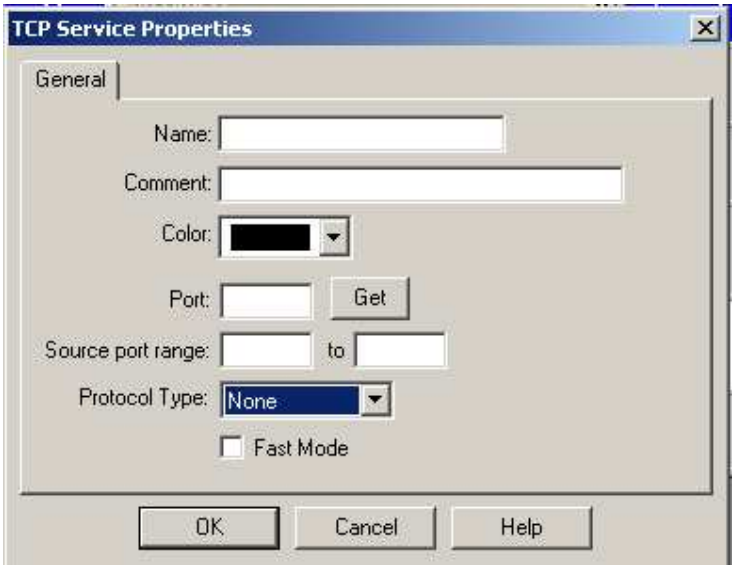
2 Bakgrund

Veriscans önskan var att kunna, via ett grafiskt användargränssnitt, representera nät, tjänster, datorer o.s.v. med hjälp av symboler för att sedan kunna bygga regelverk med hjälp av dessa symboler. De delar av projektet som utfördes under våren 2002 var verifiering av regelverk [1], vilket innebär att ett regelverk skall kunna kontrolleras med avseende på regler som döljer varandra, säker kommunikation [2] mellan brandvägg och server, då brandväggen som skall administreras inte nödvändigtvis finns inom det lokala nätet, samt ett grafiskt gränssnitt [3] för hantering av Iptables. Vår uppgift blev därför att införa en objektorienterad hantering i projektet samt att i mån av tid skapa ett nytt grafiskt gränssnitt med stöd för objekthantering. Det finns ett antal både kommersiella och icke-kommersiella lösningar på brandväggssystem. En av de mest populära kommersiella brandväggarna är Checkpoints Firewall-1 [4] vilken ger användaren stora möjligheter att grafiskt skapa regelverk med hjälp av objekt. Veriscan ville därför skapa möjlighet att arbeta med Netfilter/Iptables enligt samma modell. Firewall-1 ger en mera abstrakt syn på konfigurationen genom att endast presentera symboler för reglernas beståndsdelar i huvudfönstret. Figur 2:1 visar exempel på huvudfönstret i gränssnittet för Checkpoints Firewall-1.



Figur 2:1 Huvudfönster Checkpoint Firewall-1

För att specificera detaljer i en regel krävs att användaren högerklickar på det fält i den regel denne önskar editera vilket ger användaren ett nytt fönster innehållande fältets modifierbara detaljer. Figur 2:2 visar editering av en TCP-baserad service.



Figur 2:2 Exempel på editeringsfönster Checkpoint Firewall-1

2.1 Krav

- Konstruera en administrativ server (Management Server) som kan lagra data om den miljö som skall skyddas av Iptables samt själva regelverket. Datan skall användas av en klient med grafiskt användargränssnitt (GUI) som används för drift av en eller flera brandväggar.
- Utarbeta protokoll för kommunikation mellan GUI-klient och server.
- Utarbeta objektmodell i JAVA som representerar nätverksobjekt ex. servrar, nätverk och protokoll.
- Modifiera befintligt, eller skapa nytt, GUI i mån av tid.

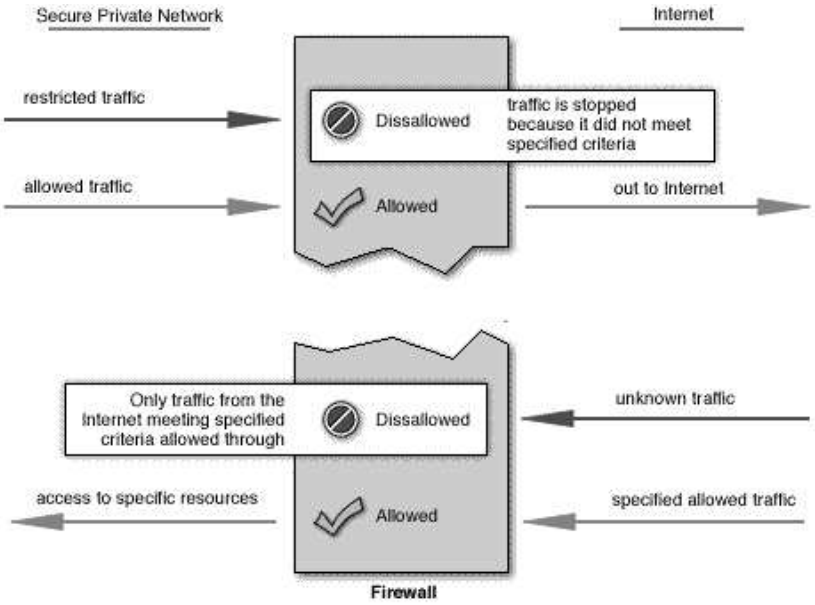
2.2 Mål

- Skapa en server som kan lagra nätverk och protokollrelaterade objekt representerande den miljö och det regelverk som råder för en viss implementation av Iptables.
- Servern skall kunna kommunicera med GUI-klient och brandvägg (Iptables).

3 Brandväggar

I detta kapitel ger vi en översiktlig bild av brandväggar. Internet och dess öppenhet innebär enorma kommunikativa möjligheter. Men för ett företag med värdefull information i datorerna medför öppenheten också risker. Företag som inte vill blotta sig för inkräktare gör därför klokt i att installera en brandvägg.

Vad är en brandvägg? En brandvägg är i datortermer en enhet som skyddar ett lokalt nätverk från utomstående nätverk som t.ex. Internet. Någon som inifrån det lokala nätverket vill nå Internet måste passera genom brandväggsenheten. På samma sätt måste någon som utifrån vill nå det lokala nätverket också passera brandväggen. En brandvägg utgörs vanligen av flera samverkande komponenter, i första hand routerutrustningar och en speciellt anpassad dator. All trafik som passerar genom brandväggen skall kontrolleras och bara den ”godkända” trafiken får passera. Brandväggen skall dessutom ha ett eget skydd mot manipulation. En brandvägg bör kunna konfigureras så att man kan erhålla en logg över nättrafiken mellan de olika näten. En brandvägg har två extrema lägen, helt öppen eller helt stängd. Om den är helt stängd kan ingen trafik komma in men ingen trafik kan heller komma ut och om den är helt öppen kan all trafik passera, ingen större säkerhet med andra ord. Det gäller att öppna brandväggen precis så mycket som är nödvändigt och låta resten vara stängt, vilket visas i figur 3:1 [5].

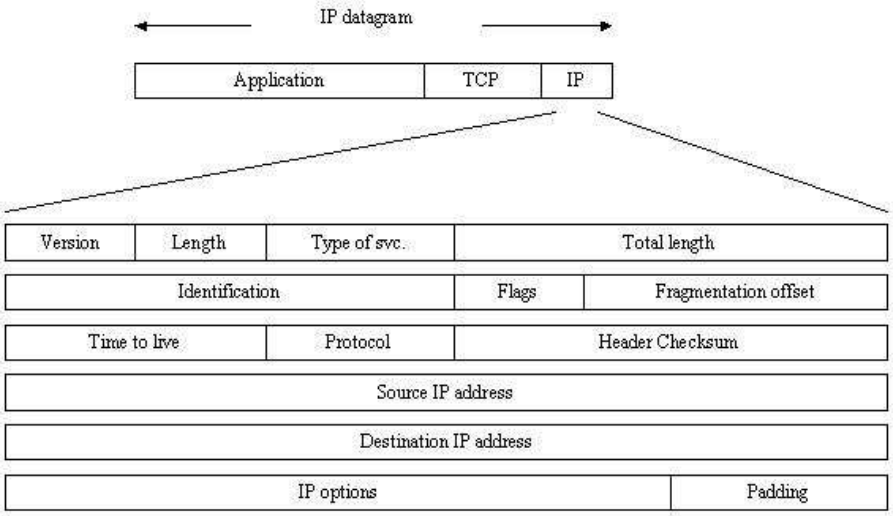


Figur 3:1 Brandväggens funktion

Detta garanterar dock inte optimal nätverkssäkerhet, för att åstadkomma det så kan helt enkelt nätverkskabeln klippas av. Det finns dock andra metoder för att komma åt systemet, t.ex. fysisk åtkomst d.v.s. en obehörig person måste hindras från direktkontakt med maskinerna. Dessa metoder behandlas ej vidare i denna rapport.

3.1 Paketfiltrerande brandväggar

Med paketfiltrering analyseras all nätverkstrafik genom kontroll av IP-paketen. Varje IP-paket undersöks för att se om det matchar någon regel som definierar det tillåtna dataflödet. Om en matchande regel existerar hanteras paketet enligt regeln, annars släpps paketet igenom. Paradigmen som gäller för paketfiltrering är: det som inte uttryckligen är förbjudet är tillåtet. Figur 3:2 visar de olika fälten i ett IP-pakets huvud, vilka kan användas av Netfilter/Iptables för att specificera regler. I de fall IP-paketet innehåller ett TCP-paket kan filtrering även göras på TCP-paketets fält.



Figur 3:2 IP header

Paketfiltrerande brandväggar är i regel snabba och genomskinliga för användare, vilket innebär att en användare på ett nätverk som är uppkopplad mot en användare på ett annat nätverk ska inte uppleva någon prestandaförsämring på grund av brandväggen. Paketfiltreringssystem skickar paket mellan interna och externa datorer på ett selektivt sätt. De tillåter eller stoppar speciella typer av paket beroende av administratörens policy. Routern som används i en paketfiltrerande brandvägg kallas screening router.

En vanlig router tittar bara på destinationsadressen för varje paket och sänder paketet till denna adress där beslut om hur paketet skall behandlas fattas.

En screening router däremot undersöker paketet noggrannare och bestämmer inte bara om den kan skicka paketet till sin destination utan också om paketet skall skickas vidare. Vilket beslut routern tar fattas vid konfigurationen. Detta tillvägagångssätt placerar en större börda på routern. Inte bara behöver routern utföra alla routingbeslut, den är också enda säkerhetspunkten. Skulle routern falla för en attack exponeras hela nätverket.

4 Netfilter/Iptables

Detta kapitel beskriver uppbyggnaden av Netfilter/Iptables samt användningsförfarandet av Iptables. Netfilter/Iptables ingår från och med version 2.4 i Linuxkärnan och kan ses som ett gränssnitt mellan kärnan och användarnivån i protokollstacken där Netfilter arbetar mot kärnan och Iptables utgör användargränssnittet. För att nätverkstrafiken skall kunna påverkas från användarnivån måste paket kunna hämtas från kärnan, undersökas och eventuellt modifieras, samt skickas tillbaks för vidare transport eller stoppas. Netfilter definierar ett antal punkter, kallade hooks, där detta förfarande kan ske i trafikens väg genom kärnan. En hook kan registrera lyssnare som meddelas när ett paket anländer. För Ipv4 är 5 stycken hooks definierade vilka illustreras i figur 4:1. Den första punkten i traverseringen är NF_IP_PRE_ROUTING som passeras när paketets checksumma kontrollerats, varefter paketet undersöks av routingkoden. Om paketet är avsett för maskinen som behandlar paketet passerar det NF_IP_LOCAL_IN, om det skall routas till ett annat interface passerar paketet NF_IP_FORWARD för att slutligen passera NF_IP_POST_ROUTING innan det skickas iväg på det avsedda interfacet. Om ett paket skapats lokalt passerar det NF_IP_LOCAL_OUT samt NF_IP_POST_ROUTING på vägen till interfacet som det skall skickas ut på.

Figur 4:1 Pakettraversering Netfilter

Iptables utgör användarens del av gränssnittet och är kopplat till Netfilters hooks. Iptables administreras från en kommandotolk samt konfigurationsfiler och styrs av regelverk bestående av tabeller, kedjor och regler. Ett regelverk kan specificeras i en fil som sedan kan läsas in av Iptables med hjälp av kommandot:

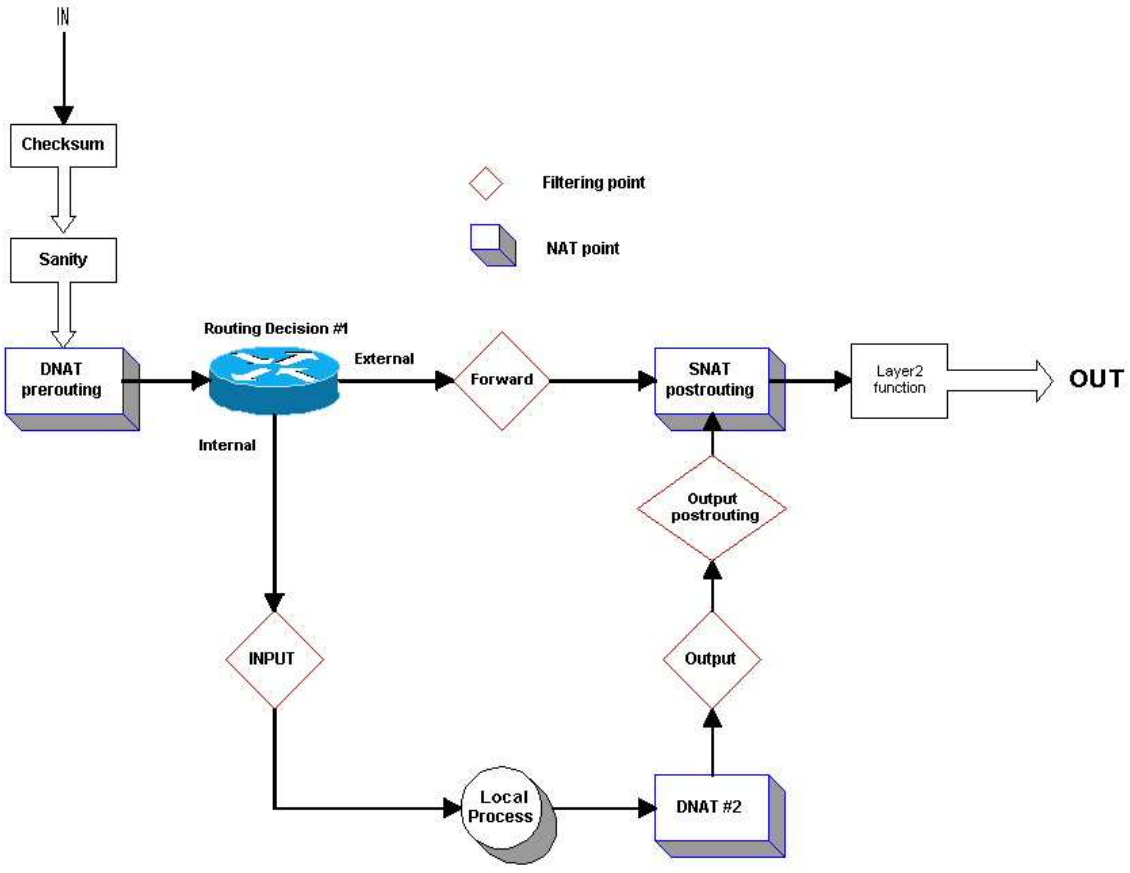
```
iptables-restore < filnamn
```

Man kan även återskapa det regelverk som Iptables använder för tillfället med hjälp av kommandot:

```
iptables-save > filnamn
```

Exempel på de nämnda filerna finns i bilaga B

Totalt finns tre tabeller, filter, NAT och mangle, innehållande kedjor. Varje kedja består av regler som matchas mot nätverkstrafiken. De kedjor som finns fördefinierade i Iptables motsvarar en koppling till Netfilters hooks. För varje hook finns en fördefinierad kedja enligt figur 4:2 [6]. Denna figur visar kedjornas placering i trafikens väg motsvarande de hooks som presenterats i figur 4:1. Kedjornas användning presenteras i delkapitel 4.1 – 4.5.



Figur 4:2 Traverseringsdiagram Linux Netfilter/Iptables

4.1 Filter-tabellen

Filtertabellen används för filtrering av datapaket d.v.s. en användare kan bestämma vad som skall ske med varje paket. Exempel på åtgärder är `DROP` - paketet kastas direkt, `REJECT` - paketet kastas och icmp-paket skickas tillbaka till avsändaren, `LOG` - information om paketet lagras i loggfil, `ACCEPT` - paketet accepteras och släpps igenom. Filtertabellen innehåller tre fördefinierade kedjor, `INPUT`, `OUTPUT` samt `FORWARD`. Inputkedjan behandlar inkommande trafik destinerat till maskinen där brandväggen körs. Outputkedjan behandlar trafiken som genereras av applikationer på maskinen där brandväggen körs. Forwardkedjan behandlar trafik som skall vidarebefordras från ett externt nät till en adress inom det interna nätet, eller trafik från det interna nätet till ett externt. För att lägga till en regel i en kedja används kommandot iptables följt av regeln.
Exempel på addering av en regel i respektive INPUT-kedjan:

```
user@host#> iptables -A INPUT –s 127.0.0.1 –p icmp –j DROP
```

`-A` innebär regeln skall läggas till sist i kedjan (append).
`INPUT` är namnet på kedjan där regeln skall sättas in.
`-s` anger paketets källa, i det här fallet maskinens loop-back adress `127.0.0.1`.
`-p` icmp anger att icmp är protokollet som skall kontrolleras.
`-j` anger vilken åtgärd som skall vidtas, i detta fall `DROP`
Denna regel innebär att maskinen inte kan skicka, eller rättare sagt vidarebefordra, icmp-meddelanden till sig själv.

För enkelhetens skull utelämnas kommandot iptables i de följande exemplen.
Ytterligare exempel på regler:

```
-D OUTPUT 1
```

`-D` innebär att en regel skall tas bort (delete).
`OUTPUT` är kedjan som regeln som påverkas.
`1` är ordningsnumret på regeln.
En regel kan även tas bort genom matchning av regelattributen (`-A` byts mot `–D` i första exemplet).

Fler exempel på insättning av regler:

```
-I FORWARD 4 –p tcp --dport 21 –j ACCEPT
```

`-I` innebär att regeln skall sättas in på placeringen som anges efter kedjans namn (insert).
`--dport` innebär att paketet matchas om en mottagarport är satt till `21`.

Varje fördefinierad kedja har en policy som träder i kraft om ingen regel matchat paketet. Denna policy bör sättas till DROP på INPUT och FORWARD för att öka säkerheten.

4.2 NAT-tabellen

Ett lokalt nätverk använder ofta IP-adresser som reserverats för interna nät vilket medför att en maskin utanför det lokala nätverket inte kan kommunicera med en dator som har en lokal-reserverad IP-adress (RFC 1918 [7]). För att denna kommunikation skall möjliggöras krävs att den maskin som kopplar samman det interna nätverket med externa nät översätter avsändaradressen till sin egen externa IP-adress för utgående trafik. Det samma gäller även för inkommande trafik med skillnaden att avsändaradressen ersätts med den sammankopplande maskinens interna IP-adress. Detta förfarande av adressöversättning kallas NAT (Network Adress Translation). NAT-tabellen innehåller också den tre fördefinierade kedjor PREROUTING, POSTROUTING och OUTPUT. Preroutingkedjan används för destinations-NAT, paket maskas mot destinationsport och destinationsadressen ändras till den berörda lokala adressen när paketet kommer in för vidarebefordran av brandväggen. Postroutingkedjan används för käll-NAT d.v.s. paketets källadress ändras när det är på väg ut genom brandväggen, för att den avsedda mottagaren skall kunna skicka ett svar till brandväggen som sedan vidarebefordrar det till den egentliga avsändaren. Outputkedjan används generellt sett inte men kan ändra paket som genererats och skall skickas ut av brandväggen.

Exempel på regel i PREROUTING-kedjan:

```
-A PREROUTING –i eth0 –p tcp –dport 21 –j DNAT --to 192.168.11.15:21
```

`-i` anger vilket interface som trafiken skall komma in på, här eth0

`DNAT` anger destinations-NAT

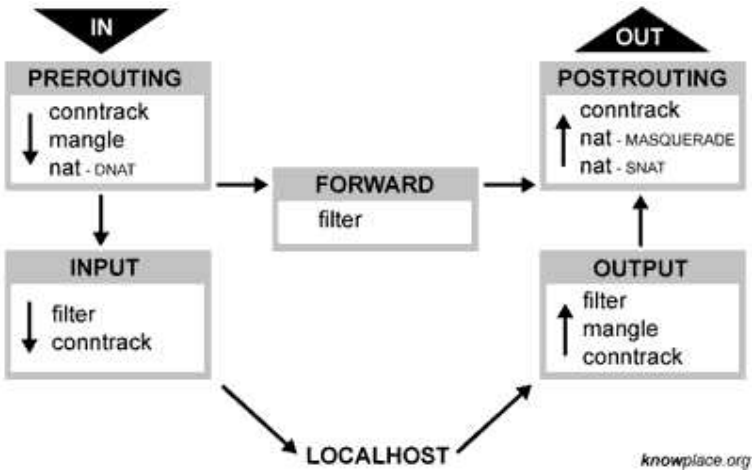
`--to` anger till vilken adress samt port trafiken skall vidarebefordras enligt adress:port

Denna regel medger trafik till kontrollkanalen för en lokal FTP-server. När ett paket matchats mot en regel i NAT-tabellen skickas det vidare till FORWARD-kedjan i filter-tabellen vilket medför att denna kedja måste ha en regel som tillåter trafiken att passera. NAT-tabellen skiljer sig från filter-tabellen genom att endast det första paketet i en ström traverserar tabellen. Om detta paket matchar en regel kommer den specificerade åtgärden att utföras på alla efterföljande paket i strömmen.

4.3 Mangle-tabellen

Mangle-tabellen används för att ändra i ett paket eller medföljande data. Exempel på ändringar är TOS (Type Of Service), TTL (Time To Live) och MARK som påverkar flaggan nfmark vilken t.ex. kan användas för att avgöra vilken process paketet skall köas till eller för att enkelt känna igen paketet i andra tabeller. De fördefinierade kedjor som finns i denna tabell är PREROUTING och OUTPUT.

Figur 4:3 [8] visar sambandet mellan kedjor och tabeller i Iptables.



Figur 4:3 Samband mellan kedjor och tabeller i Iptables

4.4 Egendefinierade kedjor

Det går även att definiera kedjor i en tabell själv genom `–t` <tabell som kedjan skall tillhöra> `–N` <nytt kedjenamn>. Flaggan `–j` som tidigare nämndes innebär egentligen ett ”hopp” till ett specifikt mål. Detta innebär att en egendefinierad kedja kan anropas, eller hoppas till, med hjälp av denna. Ett enkelt exempel på skapande och användande av en egendefinierad kedja är:

```
-t INPUT -N log-and-drop
-A  log-and-drop –j LOG
-A log-and-drop –j DROP
```

Om t.ex. en uppkopplingsbegäran mot Telnet på brandväggsmaskinen skall loggas och paketet kastas kan detta göras med:

```
-A INPUT –t tcp –dport  23 –j log-and-drop
```

Alla inkommande paket jämförs nedåt i tabellen tills en regel matchar paketet. Detta medför att reglernas inbördes ordning har betydelse. I detta fall måste loggregeln placeras högre upp i tabellen än dropregeln annars kastas paketet innan det når loggregeln. De egendefinierade kedjorna har ingen egen policy, om ingen regel matchat paketet när det når slutet av kedjan fortsätter matchningen nedåt i den anropande kedjan från regeln efter den regel som genererade hoppet.

4.5 Connection tracking

Netfilter/Iptables har också möjlighet att ta hänsyn till trafikens tillstånd med hjälp av så kallade conn_track moduler. Dessa moduler kan kontrollera om ett paket är det första paketet i en uppkoppling, hör till en redan existerande uppkoppling eller är del av en uppkoppling som är relaterad till en befintlig uppkoppling. Denna typ av kontroll kallas stateful inspection. Brandväggar som använder sig av connection tracking anses vara säkrare än tillståndslösa brandväggar då antalet konstanta öppningar genom brandväggen kan begränsas. Denna kontroll utförs på olika sätt beroende på vilken typ av transportlagerprotokoll som datan i IP-paketet är av. Sammanfattningsvis kan sägas att Iptables håller reda på paketens sändar/mottagar-adress samt eventuella portar och därigenom avgör paketens status.

Exempel på en regel som använder connection tracking:

```
user@host#> iptables –A INPUT –p tcp –m state --state
ESTABLISHED,RELATED –j ACCEPT
```

`-m` specificerar att en modul skall laddas i kärnan.

`state` specificerar vilken modul som skall laddas i kärnan.

`--state` anger vilka tillstånd som skall matchas

`ESTABLISHED,RELATED` lista av tillstånd separerade av kommatecken utan blanksteg.

Ovanstående regel innebär att redan etablerad tcp-trafik samt nya tcp-uppkopplingar som genererats av denna trafik skall tillåtas.

De tillstånd som kan specificeras är:

- NEW, det första paketet i en ny uppkoppling.
- ESTABLISHED, paket i en redan upprättad uppkoppling, även svaret på en uppkoppling som initierats från brandväggens sida.
- RELATED, uppkoppling som initierats av en annan uppkoppling som har tillståndet ESTABLISHED. Ett exempel är datakanalen för FTP som initieras av kontrollkanalen.
- INVALID, paket som inte kan identifieras, eller som inte har något tillstånd.

5 Konstruktionslösning

För att kunna representera strukturen på nätet som Iptables skall verka på skall information om maskiner, tjänster samt brandväggar med tillhörande interface kunna representeras och lagras i systemet. Detta kapitel beskriver vår konstruktionslösning för att åstadkomma detta. De objekt som utgör denna representation samt beroenden som skapas mellan dessa objekt lagras i någon form av databas. I ett nätverkssystem identifieras komponenter av sifferkombinationer (t.ex. IP-adresser och portnummer) vilket försvårar administrationen av brandväggen då dess regler byggs av dessa sifferkombinationer. För att göra administrationen mer lätthanterlig är det önskvärt att kunna namnge systemkomponenter såsom enskilda datorer, nät och tjänster mm. Dessa alias möjliggör att man kan formulera regler som t.ex. trafik till tjänsten FTP på maskinen Alpha01 från Beta25 skall tillåtas enligt:

```
Alpha01 Beta25 FTP accepteras
```

istället för

```
-A FORWARD –p tcp –s 193.172.10.25 -d 213.36.194.21 –dport 21
–j ACCEPT
-A FORWARD –p tcp –s 193.172.10.25 -d 213.36.194.21 –dport 20
–j ACCEPT
```

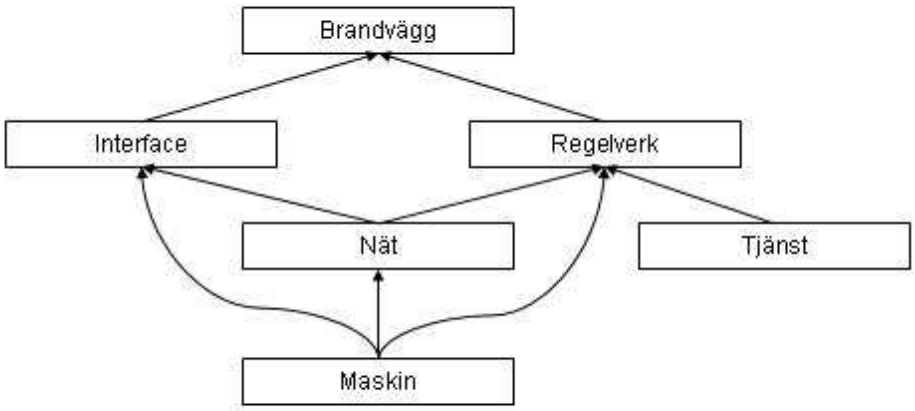
För att möjliggöra denna typ av formulering måste följande alias kunna skapas:

Komponent	**Attribut**
Maskin	Namn, IP-adress
Nät	Namn, IP-adress, nätmask
Tjänst	Namn, protokoll samt ev. port beroende på protokoll
Brandvägg	Namn, interface, regelverk
Interface	Namn, IP-adress, maskiner och nät kopplade till detta interface

Tabell 5:1 Representation av komponenter i systemet

Man skall även kunna skapa namngivna grupper av maskiner, nät samt tjänster vilka har de

gemensamma attributen namn och medlemmar.



Figur 5:1 Representation av brandvägg

Figur 5:1 beskriver hur komponenterna i tabell 5:1 samverkar. En brandvägg har interface och ett regelverk. Varje interface har information om vilka maskiner och nät som är kopplade till interfacet. Reglerna i ett regelverk byggs av nät, maskiner och tjänster. Ett nät kan utgöras av ett antal maskiner. Av de nämnda komponenterna kan brandvägg, nät, tjänst och maskin skapas fristående, d.v.s. utan beroenden av andra komponenter. I denna version kan ett regelverk skapas fristående från brandvägg men detta kan eventuellt uteslutas i kommande version, då fokus kan komma att flyttas till brandvägg och regelverket abstraheras till en beståndsdel av brandväggen. Ett interface har en koppling till regelverket genom att en regel kan specificera ett interface, men då endast via namnet och inte objektet i sig. Ett interface är alltid beroende av brandväggen och kan därför inte skapas fristående.

5.1 Server

Serverns uppgift är att sköta lagring av objekten, loggning av användaraktiviteter samt kommunikation mot berörda brandväggar. När objekten skall lagras måste även deras aktuella tillstånd sparas för att kunna återskapa deras inbördes förhållande. Exempel på ett sådant aktuellt tillstånd är aliasgrupperna samt vilka objekt som hör till vilken grupp. För att åstadkomma tillståndsbeständig lagring så har Java har en inbyggd teknik kallad serialisering som vi använder. Lagringen sker genom att servern sparar ner objekten, genom serialisering, till en databas för att lätt kunna återskapa dessa samt underlätta användning av samma objekt vid ett annat tillfälle. Lagringshanteringen sköts av en separat klass för att ge större modularitet.

De användaraktiviteter som skall loggas är:

- lyckade och misslyckade uppkopplingsförsök mot servern
- uppkoppling mot brandvägg samt begärd operation
- manipulation av objekt som får konsekvenser för andra definierade objekt

Kommunikation mot brandvägg sker genom kommunikationsmodulen som utvecklades i tidigare del av projektet [2]. När en användare begär en operation på en brandvägg kontrolleras behörighet för begärd brandvägg, uppkoppling görs, operation utförs och anslutningen kopplas ned igen. Detta för att ge användaren möjlighet att manipulera flera brandväggar samt för att kunna specificera vilka användare som har åtkomst till vilka brandväggar. För att kunna ge olika rättigheter till olika användare måste användarinformation sparas i managementservern, när en användare loggar in, för att kunna kontrolleras mot brandväggsservern vid begärd uppkoppling mot brandvägg. Servern måste också kunna överföra informationen från ett objektbaserat regelverk till ett strängbaserat regelverk som förstås av brandväggen. Innan regelverket laddas i brandväggen så bör det verifieras. Denna verifiering utförs av den verifikationsmodul som utvecklades i tidigare del av projektet [1]. Denna verifikationsmodul kräver också strängbaserade regler. Varje objektbaserad regel kan generera ett flertal strängbaserade regler då t.ex. en objektbaserad regel innehållande en grupp av alias genererar en separat strängbaserad regel för varje enskild medlem i gruppen. Detta medför att om ett fel hittas vid verifiering måste det ursprungliga objektbaserade regelverket översättas ännu en gång för att kunna avgöra vilken/vilka regler som genererat felet.

5.2 Klient

Vår klient/server-lösning bygger på den kommunikationsmodul som utvecklats i [2]. Vår tanke var att kunna använda denna modul i sin helhet, men då denna enbart hanterar strängar blev vi därför tvungna att anpassa modulen för vår objekthantering. På grund av detta har vi valt att endast använda de metoder som handhar uppkopplingsfasen, vilken inbegriper inloggning och sessionsid, signalering om att uppkopplingen skall upprätthållas vid inaktivitet samt nedkoppling. All dataöverföring sker genom nyutvecklade metoder.

De metoder som krävs för klientens funktionalitet är:

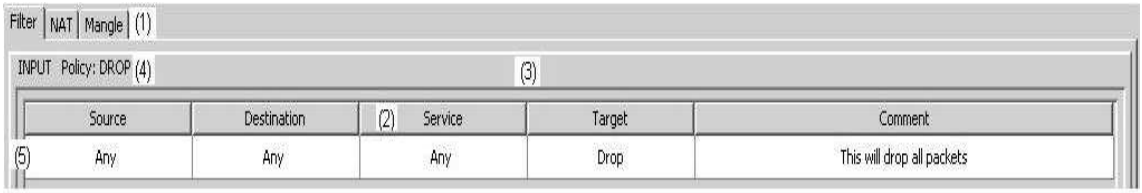
- hämtning och lagring av alias-, brandväggs- och regelverksobjekt
- överföring av regelverk för verifiering
- uppdatering av brandvägg
- hämtning och uppdatering av route- och ARP-tabell för en specificerad brandvägg
- hämtning av logg från en specificerad brandvägg
- hämtning av managementserverns loggfiler

Det vore önskvärt att klienten skötte all hantering av alias-, brandväggs- och regelverksobjekt för att kunna ha ett fristående grafiskt användargränssnitt, men på grund av tidsbrist samt bristande kunskaper i Javas awt- och swing-klasser, som används för att skapa grafiska gränssnitt, har klienten knutits hårt mot det grafiska användargränssnittet.

5.3 Grafiskt användargränssnitt

Som tidigare nämnts var ett av Veriscans önskemål att managementserverns grafiska gränssnitt skulle byggas upp på liknande sätt som gränssnittet i Firewall-1 då detta anses vara ett av marknadens mest användarvänliga. Att grafiskt representera Iptables regelverksstruktur enligt denna modell medför problem. Iptables regelverk har som bekant ett fast antal tabeller vilka därför kan representeras med var sin menyflik, figur 5:2(1), i gränssnittet. Problemet uppstår när en tabells tillhörande kedjor skall visas grafiskt. Då varje kedja har ett dynamiskt antal regler men varje regel har ett fast antal attribut blir den mest naturliga representationen av en kedja en grafisk tabell där kolumnerna motsvarar reglernas attribut, figur 5:2(2). Varje regelverkstabell kan ha egendefinierade kedjor vilket medför att antalet grafiska tabeller på varje menyflik måste kunna förändras under exekvering.

Vi har därför valt att placera varje kedjas grafiska tabell i en egen ram, figur 5:2(3). Dessa ramar placeras i sin tur under varandra i respektive flik. Detta medför att kedjans namn och eventuell policy visas i kedjeramens huvud, figur 5:2(4). Användaren kan eventuellt ges möjlighet att bestämma vilka ramar som skall visas för tillfället via en meny i huvudfönstret. Varje regel i en kedja motsvaras av en rad i den grafiska kedjetabellen, figur 5:2(5).



Figur 5:2 Grafisk representation av kedja

För att öka läsbarheten visas endast en delmängd av regelattributen i tabellen, åtkomst till de övriga attributen ges via högerklick på en regel vilket genererar en popupmeny som har valet ”visa regeldetaljer”. Regeldetaljerna visas i ett separat fönster. Den delmängd av regelattributen som visas i kedjetabellen beror på kedjan som representeras. När en ny regel läggs till i en kedja kommer en defaultregel, vars innehåll beror på kedjan den tillhör, att skapas på angiven plats, som användaren sedan får ändra. Även dessa operationer utförs via tidigare nämnda popupmeny vilken även ger möjlighet att skapa nya kedjor. En regel skall kunna inaktiveras utan att tas bort från regelverket, d.v.s. den inkluderas inte vid översättningen till strängbaserat regelverk. Det vore även önskvärt att kunna visa regelverkets struktur med en trädrepresentation men detta kommer dock, p.g.a. tidsbrist, inte att inkluderas i denna version av gränssnittet.

Aliasobjekt skapas och editeras via rullgardinsmenyer i huvudfönstret där användaren kan välja att skapa och/eller editera tjänst, nät (IP), brandvägg samt regelverk. Beroende på användarens val öppnas separata fönster innehållande objekttypens attribut där önskade modifieringar anges. Oberoende av om användaren väljer att skapa nytt eller editera ett existerande objekt öppnas samma fönster men med skillnaden att vid editering kan inte namnet ändras. Vid skapande av nya aliasobjekt måste varje alias få ett unikt namn för objekttypen, d.v.s. två tjänstealias får inte ha samma namn men däremot får ett tjänstealias och ett nätalias ha samma namn. Denna kontroll skall göras vid skapandet av objekten. Även kontroll av övriga attribut skall ske vid skapandet, t.ex. IP-adressformat och portintervall.

Vid editering av ett objekt som påverkar ett annat objekt bör användaren meddelas om detta, t.ex. då en grupp påverkas när en gruppmedlems egenskaper ändras. Användaren bör även meddelas då ändringar görs i ett regelverk som finns laddat i en brandvägg.

Användaren skall ges möjlighet att se loggar både från managementservern och från de brandväggar som finns i systemet. När en användare begär att se logg från managementservern via en meny i huvudfönstret visas denna i ett separat fönster. Även brandväggslogg visas i ett separat fönster och uppdateras kontinuerligt, av en ny tråd, under tiden fönstret är öppet.

Då flera brandväggar kan finnas i systemet samtidigt måste användaren ange namn eller IP-adress till brandväggen som efterfrågas för tillfället, detta gäller samtliga operationer som skall utföras mot en brandvägg.

Användaren skall också ges möjlighet att se och ändra ARP- och route-tabeller på brandväggar i systemet. Åtkomst till respektive tabell sker via rullgardinsmenyerna i huvudfönstret. Även dessa operationer utförs av separata trådar. Tabellerna visas i nya fönster.

Innan en användare loggat in på systemet ges denne ingen åtkomst till någon del av systemet utom själva inloggningsfönstret. Inloggningsfönstret har tre rutor: användarnamn, lösenord och adress till managementservern. För att undvika att användaren måste skriva in adressen varje gång klienten startas bör den senast angivna adressen sparas ned till en fil vid lyckad inloggning, för att läsas upp och placeras i adressrutan vid efterföljande uppstart. Ett nytt användarkonto skall inte kunna skapas från det grafiska gränssnittet, utan måste skapas på både managementserver samt berörda brandväggar.

Användaren skall när som helst kunna verifiera det aktuella regelverket. Denna verifiering utförs med hjälp av den modul som utvecklats [1]. Resultatet av denna verifiering meddelas användaren i ett popup fönster. Om fel hittas i verifieringen bör de regler som genererat felen markeras i gränssnittet.

Eventuellt skall användaren ges möjlighet att skriva ut det aktuella regelverket.

En knappmeny skall finnas i huvudfönstret för de vanligaste funktionerna:

- Ny brandvägg/regelverk/aliasobjekt
- Öppna brandvägg/regelverk
- Skriv ut regelverk
- Spara
- Verifiera regelverk
- Installera regelverk

Eftersom klienten skall vara plattformsoberoende så skall look-and-feel hämtas från det operativsystem som klienten körs på.

5.4 Kommunikation

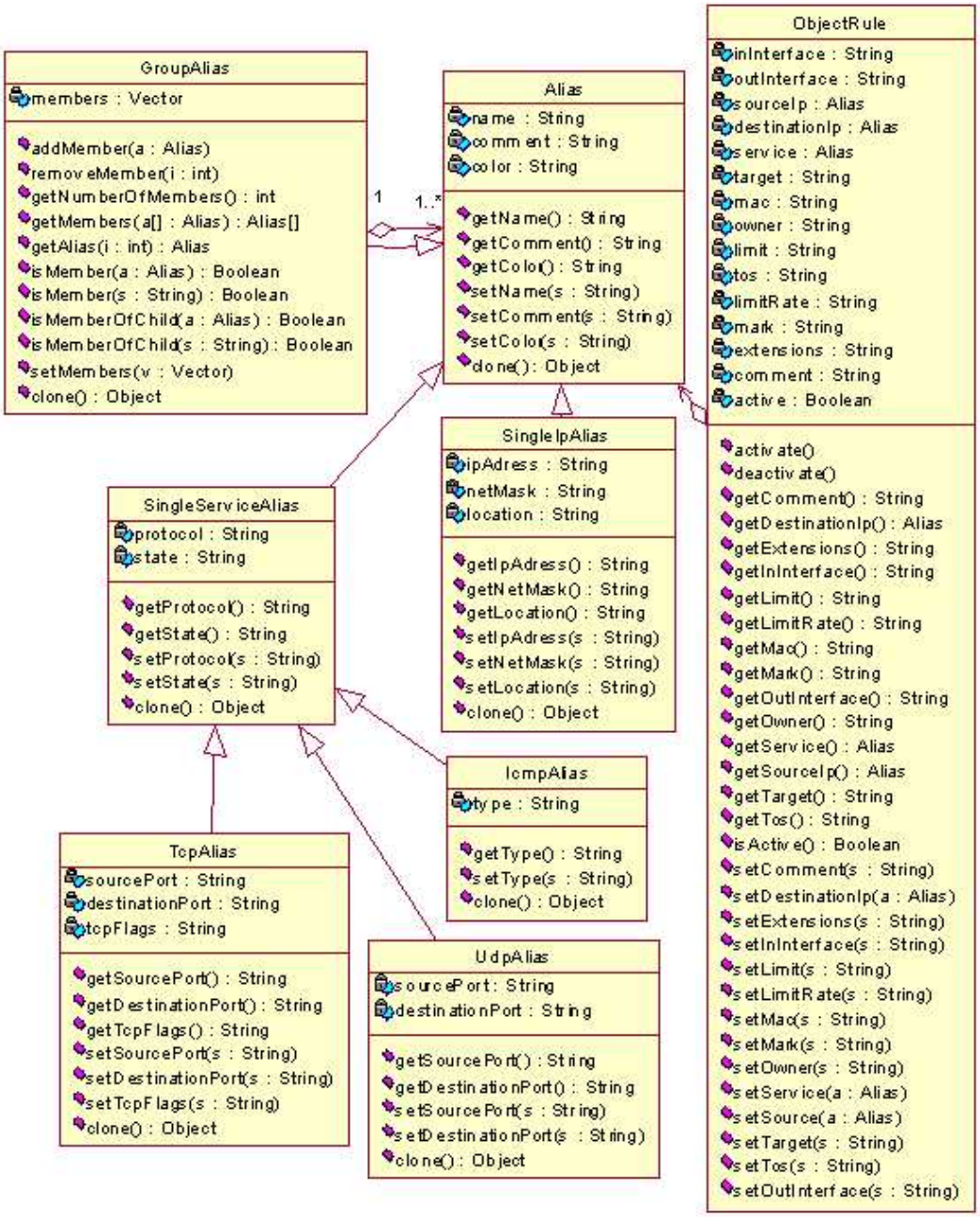
Kommunikationen mellan managementserver och brandvägg sker genom tidigare nämnda kommunikationsmodul. Denna kommunikationsmodul hanterar i det ursprungliga utförandet dock endast strängar. Då kommunikationen mellan managementservern och dess klient kräver överföring av objekt blev vi därför tvungna att modifiera modulen. Kommunikationen sker genom den i Java inbyggda tekniken RMI (Remote Method Invokation) [9]. Denna teknik innebär att ett Javaprogram som körs på en maskin kan anropa objektmetoder i ett program som körs på en annan maskin. Våra modifieringar består därför huvudsakligen av ändrade argument, då den del av kommunikationen som hanterar själva uppkopplingen inklusive inloggning kunde användas i befintligt skick.

6 Implementation

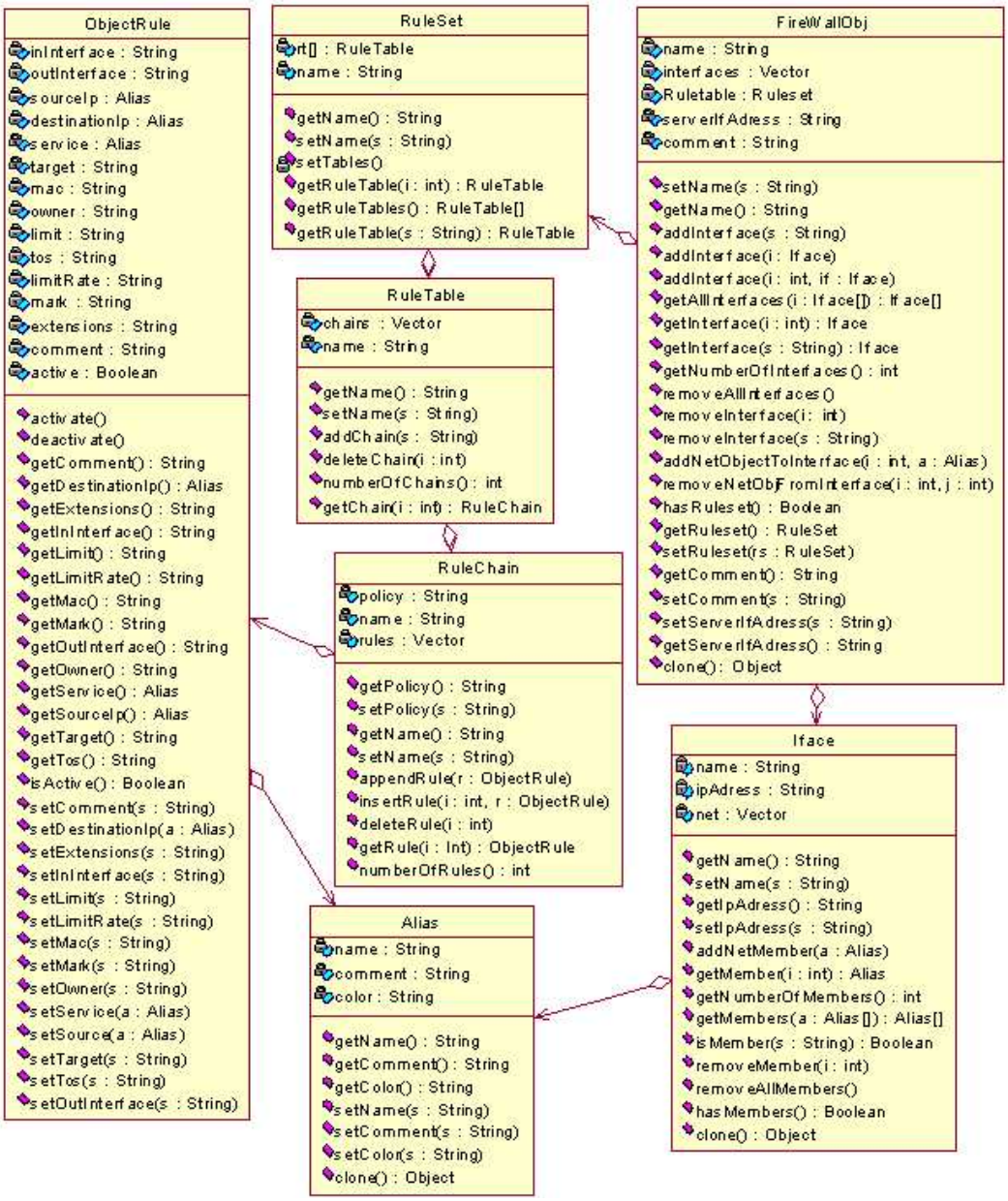
Mängden klasser som vi byggt systemet av kan delas in i fyra delmängder vilka presenteras i följande underkapitel; de klasser som utgör de komponenter som används för att representera nätverket beskrivs i kapitel 6.1, Kapitel 6.2 beskriver vår implementation av managementservern, kap. 6.3 tar upp vår klient samt kap. 6.4 beskriver det grafiska användargränssnittet.
Ett regelverk kan delas ned i tabeller, kedjor och regler. En regel kan i sin tur delas in i ett antal beståndsdelar. För att åstadkomma den representation av Iptables regelverk som tidigare nämnts har vi skapat en klass för varje objekt som ingår i ett sådant regelverk.

6.1 Representation av komponenter i systemet

Detta underkapitel presenterar de klasser som utgör byggstenarna för att representera ett nät. Vi inleder med att presentera klassdiagram för att underlätta förståelsen av de följande implementationsbeskrivningarna. Vi har av utrymmesskäl delat upp klassdiagrammet i två delar. Klassdiagrammet i figur 6:1 visar Aliasklasser samt regelklassen. Figur 6:2 visar representationen av brandvägg och regelverk.



Figur 6:1 Klassdiagram, Alias och regel



Figur 6:2 Klassdiagram brandvägg och regelverk

6.1.1 Aliasklasser

För att enkelt kunna hantera de objekt som representerar de nätkomponenter som skapas i systemet har vi implementerat en klass, Alias, som innehåller de attribut som är gemensamma för dessa objekt. De gemensamma attributen är namn, färg och kommentar.

För att kunna representera maskiner och nät har klassen SingleIpAlias skapats, vilken ärver av Alias. Denna klass innehåller attribut för att lagra IP-adress, nätmask samt en beskrivning av maskinens placering i nätverket. Denna placering är endast till för att förenkla och förtydliga för användaren av systemet och används inte av systemet självt. Samtliga dessa attribut är definierade som strängar för att ge en stor flexibilitet, då intervall av IP-adresser samt en lista av IP-adresser kan lagras i attributet ipAdress. Detta ger även möjlighet att använda DNS-namn om så önskas. Även nätmask kan anges på skilda sätt, antingen på formen 255.255.255.0, eller IP-adress / 24. Nätmasken anger antal möjliga IP-adresser som går att använda i det berörda nätet. 24 betyder inte 24 adresser, utan att 24 bitar i adressen skall kontrolleras.

För att representera tjänster krävs flera klasser eftersom de olika protokollen medger olika typer av kontroller. Den grundläggande klassen SingleServiceAlias, vilken också ärver av Alias, innehåller attribut för att lagra protokoll samt de tillstånd som är definierade i Iptables. Även i denna klass är attributen strängar, då Iptables medger att protokoll kan anges med antingen namn eller sifferbeteckning och tillstånden kan anges som en lista. För att representera tjänster som använder UDP-protokollet har vi skapat klassen UdpAlias, vilken ärver av SingleServiceAlias, innehållande attributen sourcePort och destinationPort. Iptables medger att flera portar specificeras i samma regel som en lista separerad av kommatecken varför även dessa attribut är strängar. TCP-protokollet representeras på liknande sätt med skillnaden att klassen TcpAlias har ett strängattribut för att representera TCP-protokollets flaggor. IcmpAlias som representerar ICMP-protokollet ärver även den av SingleServiceAlias och har endast ett attribut, strängen type, för att representera typ av ICMP-paket.

Både nätkomponenter och tjänster skall kunna grupperas var för sig, d.v.s. flera alias skall kunna samlas under ett gemensamt namn men denna grupp får endast bestå av antingen tjänster eller nätkomponenter, inte båda delarna i samma grupp. För att åstadkomma detta har vi skapat klassen GroupAlias, vilken ärver av Alias, innehållande endast ett attribut bestående av en Vector som lagrar nämnda grupper. Den Vectortyp som finns inbyggd i Java kan lagra vilket Javaobjekt som helst, vilket medger att gruppen kan tillåtas innehålla grupper.

Eftersom samtliga nämnda klasser ärver av Alias medger Javas polymorfism att klassen GroupAlias lagrar både tjänsteobjekt och nätobjekt. Detta kräver dock att programmet har kontroll över att typerna inte blandas i samma grupp. Samtliga klasser innehåller set- och get-metoder för sina attribut.

6.1.2 Regelverksrepresentation

En regel representeras av klassen ObjectRule. Denna klass innehåller attribut för att representera en regel på Iptables form samt en boolesk kontrollvariabel som används för att ge användaren möjlighet att inaktivera en regel utan att den tas bort. De attribut som motsvarar nätadresser samt tjänster representeras med hjälp av de aliasobjekt som nämnts i föregående underkapitel, övriga attribut representeras av strängar.

En kedja representeras av klassen RuleChain. Denna klass har två strängattribut för att representera policy för de kedjor som finns fördefinierade i Iptables samt kedjans namn. För att kunna lagra regler har den även ett vektorattribut.

En Iptablestabell motsvaras av klassen RuleTable som innehåller ett strängattribut som lagrar tabellens namn samt ett vektorattribut för att lagra kedjor.

Det kompletta regelverket representeras av klassen RuleSet. Denna klass har ett strängattribut för att lagra namnet på regelverket samt en array, av typen RuleTable, med tre platser för att lagra de tre tabeller som finns i Iptables. När konstruktorn i denna klass anropas skapas ett objekt innehållande tre tomma tabeller. Orsaken till detta är att Iptables i dagsläget inte stöder användardefinierade tabeller. Detta medför även att klassen inte innehåller några metoder för att skapa nya tabeller. Dessa funktioner kan emellertid enkelt läggas till om behov uppstår.

6.1.3 Brandväggsrepresentation

Klassen FireWallObj har skapats för att representera brandväggar i systemet. Denna klass har strängattribut för att lagra brandväggsnamn, den IP-adress (eller DNS-namn) till det interface som har kontakt med managementservern samt kommentar. Klassen har ett attribut för att representera regelverket som ett brandväggsobjekt använder. Den innehåller även ett vektorattribut för att lagra information om brandväggens interface. För att hantera dessa interface har klassen Iface skapats. Denna klass lagrar namn och adress med hjälp av strängar. Den innehåller även en vektor för att lagra de nätkomponenter som är kopplade till ett interface.

Detta för att ge möjlighet att representera nätets topologi i användargränssnittet samt för att ge användaren hjälp med s.k. anti-spoofingregler, vilket innebär att brandväggen kan kontrollera att ett pakets avsändare verkligen finns i nätet som är kopplat till det interface som paketet kommer in på.

6.2 Server

Managementservern bygger på VSServerklassen som implementerades i [2]. Vi har valt att endast använda delar av denna server i vår implementation i stället för att inkludera klassen som ett attribut i vår klass. Som tidigare nämnts har vi byggt metoder för att sköta överföring av de objekt som hanteras av systemet. Servern handhar även omvandling av objektbaserade regler till strängbaserade regler samt verifieringen av dessa. Vår server sköter också uppkoppling och inloggning mot VSServerobjekt som finns på de brandväggar som ingår i systemet. Detta medför att servern har ett attribut bestående av VSClient för att möjliggöra denna kommunikation. Servern har också ett attribut bestående av klassen DBHandler som sköter den fysiska lagringen av de berörda objekten samt ett attribut bestående av klassen LoginHandler, utvecklad i [2], som handhar kontroll och inloggning av användare från vår klient. När en användare loggas in på servern lagras användarinformationen för att automatiskt kunna logga in mot ett VSServerobjekt när användaren begär en operation på en specificerad brandvägg. Managementservern sköter även om loggning av de händelser som specificerades i konstruktionen. För att möjliggöra kommunikationen via RMI ärver serverklassen av Javaklassen UnicastRemoteObject [9]. För att kunna använda RMI krävs även att ett Javagränssnitt implementeras som klienten kan använda.

6.3 Klient

Klienten har i denna version av programmet endast implementerats för att möjliggöra kommunikationen mot servern och har ingen ytterligare intelligens. Klassen inkluderar ett attribut bestående av serverns interface samt attribut från klassen VSClient från [2] som krävs för att skapa och upprätthålla förbindelsen mot serverobjektet.

6.4 Grafiskt användargränssnitt

Det grafiska gränssnittet är i skrivande stund fortfarande under utveckling.

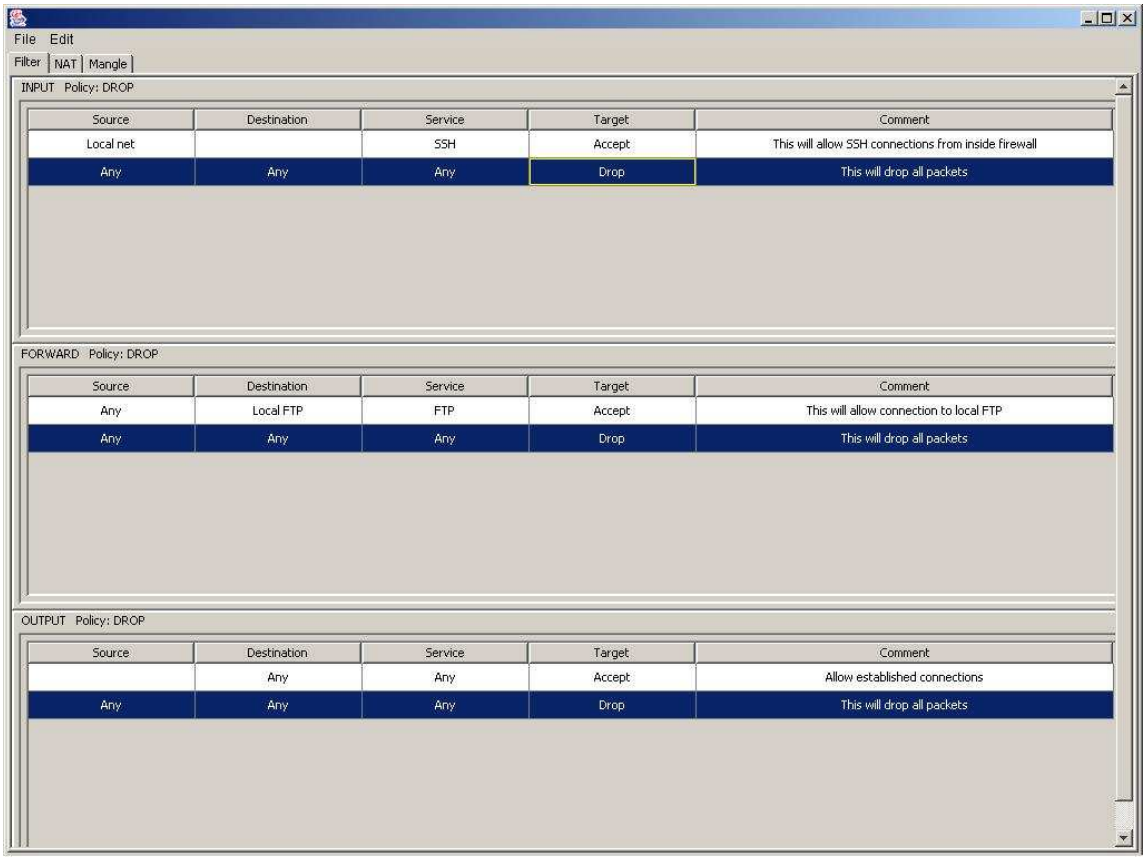
När användaren startar klientprogrammet visas först ett inloggningsfönster, figur 6:3, där användaren får ange adressen till servern samt användarnamn och lösenord. Adressen kan vara antingen en IP-adress eller ett DNS-namn.

Figur 6:3 Inloggningsfönster, grafiskt gränssnitt

Om inloggningen misslyckas visas ett dialogfönster, figur 6:4, innehållande ett meddelande med orsaken till misslyckandet.

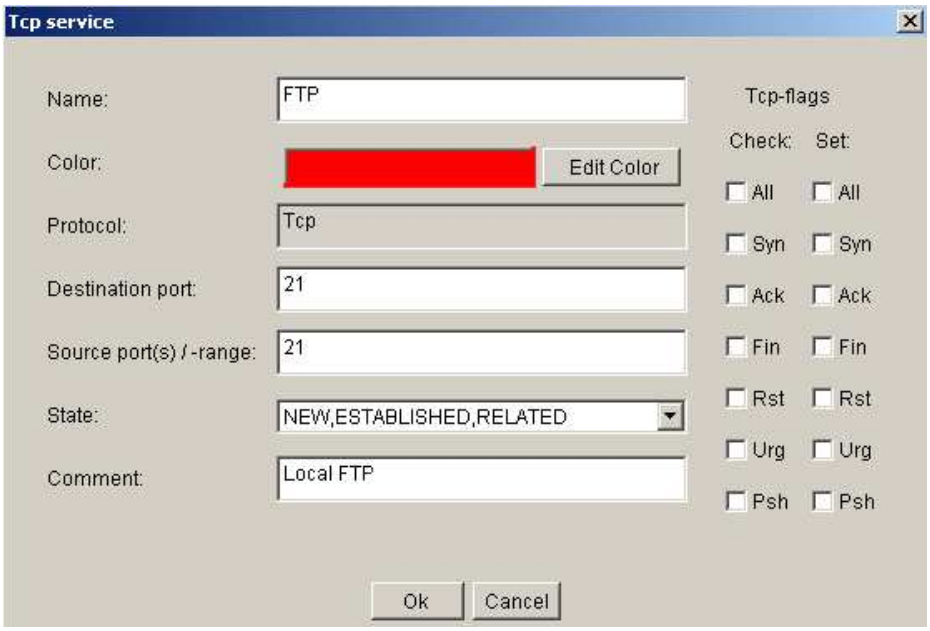
Figur 6:4 Feldialog vid inloggning

När inloggningen lyckas visas huvudfönstret, figur 6:5, innehållande en meny med alternativ för att spara, öppna, skapa nya objekt o.s.v. Huvudfönstret har tre flikar, Filter, NAT och Mangle. Varje flik innehåller den grafiska representationen för motsvarande Iptablestabell. Användaren ges här möjlighet att välja mellan två olika vyer, antingen en hel Iptablestabell i samma grafiska tabell eller varje kedja i en egen grafisk tabell, där den senare återger Iptables interna struktur. För att editera en cell i tabellen markerar användaren önskad cell och högerklickar sedan, vilket ger en popup-meny innehållande de aktuella valmöjligheterna. Beroende av användarens val öppnas ett nytt fönster i vilket editeringen sedan utförs. Åtkomst till logg, ARP- och routingtabell på en specificerad brandvägg ges genom tidigare nämnda menyer, dessa visas i ett eget fönster.



Figur 6:5 Huvudfönster grafiskt gränssnitt

Ett exempel på ett fönster för att skapa ett nytt servicealias kan ses i figur 6:6.

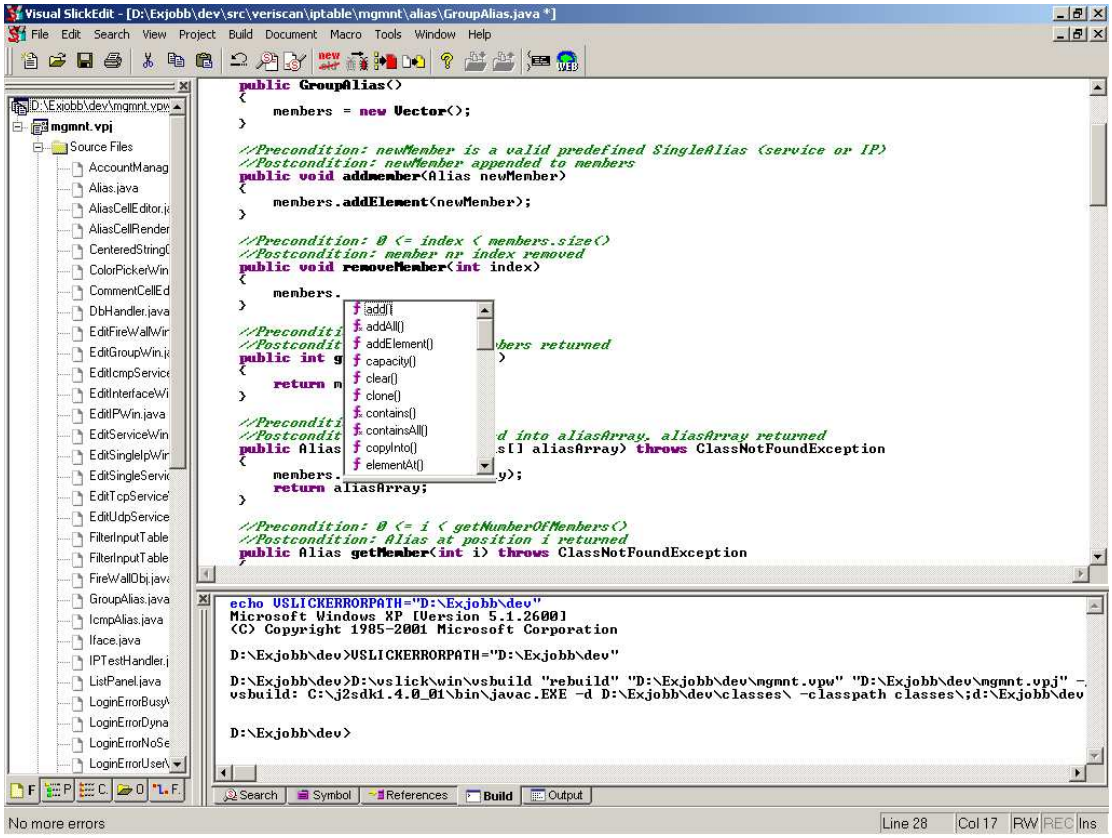


Figur 6:6 Skapa en TCP-service

Denna version av gränssnittet är dock endast en prototyp som implementeras för att säkerställa funktionen hos de övriga delarna av systemet. På grund av detta har endast den tabellvy som visar Iptables struktur implementerats i detta skede. Utvecklingen av gränssnittet kommer att fortgå under våren 2003.

7 Kommentarer

Vid projektets start var vi praktiskt taget helt obekanta med språket Java, vilket ledde till att en del tid gick åt till att sätta sig in i språket innan vi kunde börja utveckla något användbart. Till vår hjälp hade vi utvecklingsverktyget Visual SlickEdit [10], en utvecklingsmiljö som stöder ett antal programmeringsspråk, vi har dock endast använt den del av miljön som berör Java. När ett Javapaket eller en Javaklass inkluderas i en av användaren skapad klass känner miljön till de tillgängliga metoderna och variablerna och ger användaren möjlighet att välja dessa från en lista då de är tillämpbara, vilket visas i figur 7:1. Verktyget har stöd för både kompilering av enstaka klasser samt hela projekt och ger även programmeraren hjälp med indentering och syntax. Vi anser detta vara ett utomordentligt verktyg. När det var dags att utveckla den grafiska delen av projektet provade vi även NetBeans [11], en miljö för enbart Java som innehåller ett grafiskt utvecklingsverktyg för att skapa fönster med tillhörande komponenter. Denna miljö är dock oerhört resurskrävande, vilket leder till långa väntetider vid ett flertal operationer. Vi har därför valt att använda Visual SlickEdit även för den grafiska delen.



Figur 7:1 Visual SlickEdit

När vi skapade prototypen för det grafiska gränssnittet har vi utgått från en användare som har kännedom om Iptables struktur med tabeller och kedjor. Det vore önskvärt att kunna abstrahera bort den interna strukturen hos Iptables helt, men vi har inte lyckats hitta något bra sätt att göra detta på ännu. Man kan tänka sig att representera en Iptablestabell med en enda grafisk tabell, men eftersom reglernas inbördes ordning är viktig är det svårt att göra detta på ett meningsfullt sätt. Om systemet skall placera en regel i rätt kedja automatiskt baserat på regelns innehåll kommer den att bli tvungen att flyttas i den grafiska representationen också för att visas i sin rätta omgivning. Detta medför att reglerna kan upplevas ”hoppa omkring” i tabellen, vilket inte är ett önskvärt beteende. Om regeln inte flyttas i den grafiska representationen finns inget sätt att visa användaren den faktiska innebörden av en sekvens av regler och systemet måste mer eller mindre ”gissa” ordningen på reglerna. För att göra systemet riktigt användbart måste en användare utan detaljkännedom om Iptables kunna använda det, vilket medför att denna ytterligare abstraktion är nödvändig. Detta problem återstår fortfarande att lösa. Systemet bör även kunna förse användaren av systemet med en specificerad brandväggsmaskins interfacenamn vid skapandet av ett nytt brandväggsobjekt. Denna funktion är i den här versionen av systemet inte implementerad.

Vi har heller inte implementerat den nämnda knappmenyn samt önskade snabbkommandon i prototypen av gränssnittet.

Kommunikationsmodulen som vi använder oss av använder sig av IPSEC [12] vilket innebär att de maskiner som ingår i systemet måste konfigureras för detta. Hur denna konfiguration går till beskrivs i [2]. För att RMI skall fungera på ett säkert sätt krävs att man använder sig av en policyfil som anges vid uppstarten av programmet. Innehållet i denna fil specificerar hur Javas SecurityManager [9] skall bete sig. Då denna kan påverka flera program som körs på en maskin har vi i detta läge avstått från att inkludera en sådan. Vi har använt oss av en policy som medger ”allt” för att testa systemet.

8 Sammanfattning

Vi har i detta examensarbete implementerat en managementserver för Linux Netfilter/Iptables samt en prototyp för ett grafiskt gränssnitt för denna server på uppgift av Veriscan Security. Detta projekt är en vidareutveckling av tidigare examensarbeten utförda för Veriscan Security. De tidigare examensarbetena bestod av en kommunikationsmodul samt en verifieringsmodul för Iptables vilka vi, efter smärre anpassningar, inkluderat i vårt system. Vi har i detta arbete infört en objektorienterad hantering av regler för Iptables där regler byggs av användardefinierade komponenter istället för de traditionella textbaserade reglerna, vilket medfört att vi utarbetat en objektmodell för att representera brandväggar, maskiner, nät och tjänster. Denna rapport ger en översiktlig beskrivning av brandväggar samt en något mera ingående beskrivning av Netfilter/Iptables vilken behövs för att få en djupare förståelse om de problemställningar vi ställts inför. Denna beskrivning inleds med en beskrivning av Netfilters funktion samt hur denna kopplas samman med Iptables för att sedan övergå till att beskriva Iptables tabell- och kedjehantering samt en kortfattad beskrivning av connection tracking. Vi presenterar sedan vår konstruktionslösning av de delar som ingår i projektet.
Dessa delar är:

- en server som handhar lagring av information om brandväggar, maskiner, nät och tjänster, uppkoppling mot brandvägg för installering av regelverk samt hämtande av logg, ARP- och route-tabell
- en klient som innehåller logik för uppkoppling mot servern med autentisering av användare och sessionsid
- ett grafiskt användargränssnitt som ger en användare möjlighet att administrera en Linux Netfilter/Iptables brandvägg.

Konstruktionslösningen efterföljs av en beskrivning av hur vi implementerat de olika delarna i projektet. Rapporten avslutas av en kort redogörelse av projektets utförande, diverse problem som uppstått, vilka utvecklingsverktyg som använts samt vad som i skrivande stund återstår att göra. Vi har även inkluderat bilagor bestående av Iptables man-sida samt exempel på filer som kan användas för att administrera Iptables.

Referenser

[1] Joel Ivarsson & Leo Wentzel, Fjärradministration av Iptables Verifiering, examensarbete 2002:17, Datavetenskap Karlstads universitet

[2] Tony Bergh & Håkan Bergmark, Fjärradministration av Iptables Säker kommunikation, examensarbete 2002:05, Datavetenskap Karlstads universitet

[3] Olof Larsson & Joacim Söderholm, Fjärradministration av Iptables Grafiskt gränssnitt, examensarbete 2002:19, Datavetenskap Karlstads universitet

[4] Hemsida för Checkpoints Firewall-1, 2002-11, http://www.checkpoint.com/

[5] Vicomsoft, säkerhetsinriktat företag med information och kommersiella säkerhetslösningar, 2002-11, http://www.vicomsoft.com/

[6] Matthew Wronkowski, /projects/linux netfilter tutorial, information om Linux Netfilter, 2002-11, http://www.csh.rit.edu/~mattw/

[7] Network Working Group, Request For Comments 1918 om IP-adresser för lokala nät, 2002-11, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1918.html

[8] Shane Chen, Information om Netfilter/Iptables, 2002-11, http://www.knowplace.org/netfilter/syntax.html#traversal

[9] Sun Microsystems, Dokumentation och API för Java RMI, 2002-11, http://java.sun.com/products/jdk/rmi/

[10] SlickEdit Inc., utvecklare av Visual SlickEdit, 2002-11, http://www.slickedit.com/

[11] netBeans, Hemsida för NetBeans IDE, 2002-11, http://www.netbeans.org/ide/index.html

[12] IETF Secretariat, Information om IPSec, 2002-11, http://www.ietf.org/html.charters/ipsec-charter.html

A Iptables manpage

A.1 NAME

Iptables - IP packet filter administration

A.2 SYNOPSIS

iptables -[ADC] chain rule-specification [options]

iptables -[RI] chain rulenum rule-specification [options]

iptables -D chain rulenum [options]

iptables -[LFZ] [chain] [options]

iptables -[NX] chain

iptables -P chain target [options]

iptables -E old-chain-name new-chain-name

A.3 DESCRIPTION

Iptables is used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. There are several different tables which may be defined, and each table contains a number of built-in chains, and may contain user-defined chains.

Each chain is a list of rules which can match a set of packets: each rule specifies what to do with a packet which matches. This is called a `target', which may be a jump to a user-defined chain in the same table.

A.4 TARGETS

A firewall rule specifies criteria for a packet, and a target. If the packet does not match, the next rule in the chain is the examined; if it does match, then the next rule is specified by the value of the target, which can be the name of a user-defined chain, or one of the special values *ACCEPT*, *DROP*, *QUEUE*, or *RETURN*.

ACCEPT means to let the packet through. *DROP* means to drop the packet on the floor. *QUEUE* means to pass the packet to userspace. *RETURN* means stop traversing this chain, and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached, or a rule in a built-in chain with target *RETURN* is matched, the target specified by the chain policy determines the fate of the packet.

A.5 TABLES

There are current three tables (which tables are present at any time depends on the kernel configuration options and which modules are present).

-t, --table

This option specifies the packet matching table which the command should operate on. If the kernel is configured with automatic module loading, an attempt will be made to load the appropriate module for that table if it is not already there.

The tables are as follows: **filter** This is the default table, and contains the built-in chains INPUT (for packets coming into the box itself), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets). **nat** This table is consulted when a packet which is creates a new connection is encountered. It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT (for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out). **mangle** This table is used for specialized packet alteration. It has two built-in chains: PREROUTING (for altering incoming packets before routing) and OUTPUT (for altering locally-generated packets before routing).

A.6 OPTIONS

The options that are recognized by **iptables** can be divided into several different groups.

A.6.1 COMMANDS

These options specify the specific action to perform; only one of them can be specified on the command line, unless otherwise specified below. For all the long versions of the command and option names, you only need to use enough letters to ensure that **iptables** can differentiate it from all other options.

-A, --append

Append one or more rules to the end of the selected chain. When the source and/or destination names resolve to more than one address, a rule will be added for each possible address combination.

-D, --delete

Delete one or more rules from the selected chain. There are two versions of this command: the rule can be specified as a number in the chain (starting at 1 for the first rule) or a rule to match.

-R, --replace

Replace a rule in the selected chain. If the source and/or destination names resolve to multiple addresses, the command will fail. Rules are numbered starting at 1.

-I, --insert

Insert one or more rules in the selected chain as the given rule number. So, if the rule number is 1, the rule or rules are inserted at the head of the chain. This is also the default if no rule number is specified.

-L, --list

List all rules in the selected chain. If no chain is selected, all chains are listed. It is legal to specify the **-Z** (zero) option as well, in which case the chain(s) will be atomically listed and zeroed. The exact output is effected by the other arguments given.

-F, --flush

Flush the selected chain. This is equivalent to deleting all the rules one by one.

-Z, --zero

Zero the packet and byte counters in all chains. It is legal to specify the **-L, --list** (list) option as well, to see the counters immediately before they are cleared; see above.

-N, --new-chain

Create a new user-defined chain of the given name. There must be no target of that name already.

-X, --delete-chain

Delete the specified user-defined chain. There must be no references to the chain (if there are you must delete or replace the referring rules before the chain can be deleted). If no argument is given, it will attempt to delete every non-builtin chain.

-P, --policy

Set the policy for the chain to the given target. See the section

-E, --rename-chain

Rename the user specified chain to the user supplied name; this is cosmetic, and has no effect on the structure of the table. **TARGETS** for the legal targets. Only non-userdefined chains can have policies, and neither built-in nor user-defined chains can be policy targets.

-h

Help. Give a (currently very brief) description of the command syntax.

A.6.2 PARAMETERS

The following parameters make up a rule specification (as used in the add, delete, replace, append and check commands).

-p, --protocol [!] *protocol*

The protocol of the rule or of the packet to check. The specified protocol can be one of *tcp*, *udp*, *icmp*, or *all*, or it can be a numeric value, representing one of these protocols or a different one. Also a protocol name from /etc/protocols is allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to *all*. Protocol *all* will match with all protocols and is taken as default when this option is omitted. *All* may not be used in in combination with the check command.

-s, --source [!] *address*[/*mask*]

Source specification. *Address* can be either a hostname, a network name, or a plain IP address. The *mask* can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of *24* is equivalent to *255.255.255.0*. A "!" argument before the address specification inverts the sense of the address. The flag **--src** is a convenient alias for this option.

-d, --destination [!] *address*[/*mask*]

Destination specification. See the description of the **-s** (source) flag for a detailed description of the syntax. The flag **--dst** is an alias for this option.

-j, --jump *target*

This specifies the target of the rule; ie. what to do if the packet matches it. The target can be a user-defined chain (not the one this rule is in), one of the special builtin targets which decide the fate of the packet immediately, or an extension (see **EXTENSIONS** below). If this option is omitted in a rule, then matching the rule will have no effect on the packet's fate, but the counters on the rule will be incremented.

-i, --in-interface [!] [*name*]

Optional name of an interface via which a packet is received (for packets entering the **INPUT**, **FORWARD** and **PREROUTING** chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, the string "+" is assumed, which will match with any interface name.

-o, --out-interface [!] [*name*]

Optional name of an interface via which a packet is going to be sent (for packets entering the **FORWARD**, **OUTPUT** and **POSTROUTING** chains). When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, the string "+" is assumed, which will match with any interface name.

[!] -f, --fragment

This means that the rule only refers to second and further fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!" argument precedes the "-f" flag, the sense is inverted.

A.6.3 OTHER OPTIONS

The following additional options can be specified:

-v, --verbose

Verbose output. This option makes the list command show the interface address, the rule options (if any), and the TOS masks. The packet and byte counters are also listed, with the suffix 'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipliers respectively (but see the **-x** flag to change this). For appending, insertion, deletion and replacement, this causes detailed information on the rule or rules to be printed.

-n, --numeric

Numeric output. IP addresses and port numbers will be printed in numeric format. By default, the program will try to display them as host names, network names, or services (whenever applicable).

-x, --exact

Expand numbers. Display the exact value of the packet and byte counters, instead of only the rounded number in K's (multiples of 1000) M's (multiples of 1000K) or G's (multiples of 1000M). This option is only relevant for the **-L** command.

--line-numbers

When listing rules, add line numbers to the beginning of each rule, corresponding to that rule's position in the chain.

A.7 MATCH EXTENSIONS

iptables can use extended packet matching modules. The following are included in the base package, and most of these can be preceded by a **!** to invert the sense of the match.

A.7.1 tcp

These extensions are loaded if `--protocol tcp' is specified, and no other match is specified. It provides the following options:

--source-port [!] [*port[:port]*]

Source port or port range specification. This can either be a service name or a port number. An inclusive range can also be specified, using the format *port*:*port*. If the first port is omitted, "0" is assumed; if the last is omitted, "65535" is assumed. If the second port greater then the first they will be swapped. The flag **--sport** is an alias for this option.

--destination-port [!] [*port[:port]*]

Destination port or port range specification. The flag **--dport** is an alias for this option.

--tcp-flags [!] *mask comp*

Match when the TCP flags are as specified. The first argument is the flags which we should examine, written as a comma-separated list, and the second argument is a comma-separated list of flags which must be set. Flags are:

SYN ACK FIN RST URG PSH ALL NONE. Hence the command
iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST SYN
will only match packets with the SYN flag set, and the ACK, FIN and RST flags unset.

[!] --syn

Only match TCP packets with the SYN bit set and the ACK and FIN bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. It is equivalent to **--tcp-flags SYN,RST,ACK SYN**. If the "!" flag precedes the "--syn", the sense of the option is inverted.

--tcp-option [!] *number*

Match if TCP option set.

A.7.2 udp

These extensions are loaded if `--protocol udp' is specified, and no other match is specified. It provides the following options:

--source-port [!] [*port[:port]*]

Source port or port range specification. See the description of the **--source-port** option of the TCP extension for details.

--destination-port [!] [*port[:port]*]

Destination port or port range specification. See the description of the **--destination-port** option of the TCP extension for details.

A.7.3 icmp

This extension is loaded if `--protocol icmp' is specified, and no other match is specified. It provides the following option:

--icmp-type [!] *typename*

This allows specification of the ICMP type, which can be a numeric ICMP type, or one of the ICMP type names shown by the command iptables -p icmp -h

A.7.4 mac

--mac-source [!] *address*

Match source MAC address. It must be of the form XX:XX:XX:XX:XX:XX. Note that this only makes sense for packets entering the **PREROUTING**, **FORWARD** or **INPUT** chains for packets coming from an ethernet device.

A.7.5 limit

This module matches at a limited rate using a token bucket filter: it can be used in combination with the LOG target to give limited logging. A rule using this extension will match until this limit is reached (unless the `!' flag is used).

--limit *rate*

Maximum average matching rate: specified as a number, with an optional `/second', `/minute', `/hour', or `/day' suffix; the default is 3/hour.

--limit-burst *number*

The maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is 5.

A.7.6 multiport

This module matches a set of source or destination ports. Up to 15 ports can be specified. It can only be used in conjunction with **-p tcp** or **-p udp**.

--source-port [*port[,port]*]

Match if the source port is one of the given ports.

--destination-port [*port[,port]*]

Match if the destination port is one of the given ports.

--port [*port[,port]*]

Match if the both the source and destination ports are equal to each other and to one of the given ports.

A.7.7 mark

This module matches the netfilter mark field associated with a packet (which can be set using the **MARK** target below).

--mark *value[/mask]*

Matches packets with the given unsigned mark value (if a mask is specified, this is logically ANDed with the mark before the comparison).

A.7.8 owner

This module attempts to match various characteristics of the packet creator, for locally-generated packets. It is only valid in the OUTPUT chain, and even this some packets (such as ICMP ping responses) may have no owner, and hence never match.

--uid-owner *userid*

Matches if the packet was created by a process with the given effective user id.

--gid-owner *groupid*

Matches if the packet was created by a process with the given effective group id.

--pid-owner *processid*

Matches if the packet was created by a process with the given process id.

--sid-owner *sessionid*

Matches if the packet was created by a process in the given session group.

A.7.9 state

This module, when combined with connection tracking, allows access to the connection tracking state for this packet.

--state *state*

Where state is a comma separated list of the connection states to match. Possible states are **INVALID** meaning that the packet is associated with no known connection, **ESTABLISHED** meaning that the packet is associated with a connection which has seen packets in both directions, **NEW** meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions, and **RELATED** meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error.

A.7.10 unclean

This module takes no options, but attempts to match packets which seem malformed or unusual. This is regarded as experimental.

A.7.11 tos

This module matches the 8 bits of Type of Service field in the IP header (ie. including the precedence bits).

--tos *tos*

The argument is either a standard name, (use iptables -m tos -h to see the list), or a numeric value to match.

A.8 TARGET EXTENSIONS

iptables can use extended target modules: the following are included in the standard distribution.

A.8.1 LOG

Turn on kernel logging of matching packets. When this option is set for a rule, the Linux kernel will print some information on all matching packets (like most IP header fields) via *printk*().

--log-level *level*

Level of logging (numeric or see *syslog.conf*(5)).

--log-prefix *prefix*

Prefix log messages with the specified prefix; up to 14 letters long, and useful for distinguishing messages in the logs.

--log-tcp-sequence

Log TCP sequence numbers. This is a security risk if the log is readable by users.

--log-tcp-options

Log options from the TCP packet header.

--log-ip-options

Log options from the IP packet header.

A.8.2 MARK

This is used to set the netfilter mark value associated with the packet. It is only valid in the **mangle** table.

--set-mark *mark*

A.8.3 REJECT

This is used to send back an error packet in response to the matched packet: otherwise it is equivalent to **DROP**. This target is only valid in the **INPUT**, **FORWARD** and **OUTPUT** chains, and user-defined chains which are only called from those chains. Several options control the nature of the error packet returned:

--reject-with *type*

The type given can be **icmp-net-unreachable**, **icmp-host-unreachable**, **icmp-port-unreachable**, **icmp-proto-unreachable**, **icmp-net-prohibited**or **icmp-host-prohibited**, which return the appropriate ICMP error message (port-unreachable is the default). The option **echo-reply** is also allowed; it can only be used for rules which specify an ICMP ping packet, and generates a ping reply. Finally, the option **tcp-reset** can be used on rules in (or called from) the **INPUT** chain which only match the TCP protocol: this causes a TCP RST packet to be sent back.

A.8.4 TOS

This is used to set the 8-bit Type of Service field in the IP header. It is only valid in the **mangle** table.

--set-tos *tos*

You can use a numeric TOS values, or use iptables

-j TOS –h

to see the list of valid TOS names.

A.8.5 MIRROR

This is an experimental demonstration target which inverts the source and destination fields in the IP header and retransmits the packet. It is only valid in the **INPUT**, **FORWARD** and **OUTPUT** chains, and user-defined chains which are only called from those chains.

A.8.6 SNAT

This target is only valid in the **nat** table, in the **POSTROUTING** chain. It specifies that the source address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined. It takes one option:

--to-source <*ipaddr*>*[-*<*ipaddr*>*][:port-port]*

which can specify a single new source IP address, an inclusive range of IP addresses, and optionally, a port range (which is only valid if the rule also specifies **-p tcp** or **-p udp**). If no port range is specified, then source ports below 512 will be mapped to other ports below 512: those between 1024 will be mapped to ports below 1024, and other ports will be mapped to 1024 or above. Where possible, no port alteration will occur.

A.8.7 DNAT

This target is only valid in the **nat** table, in the **PREROUTING** and **OUTPUT** chains, and user-defined chains which are only called from those chains. It specifies that the destination address of the packet should be modified (and all future packets in this connection will also be mangled), and rules should cease being examined. It takes one option:

--to-destination <*ipaddr*>*[-*<*ipaddr*>*][:port-port]*

which can specify a single new destination IP address, an inclusive range of IP addresses, and optionally, a port range (which is only valid if the rule also specifies **-p tcp** or **-p udp**). If no port range is specified, then the destination port will never be modified.

A.8.8 MASQUERADE

This target is only valid in the **nat** table, in the **POSTROUTING** chain. It should only be used with dynamically assigned IP (dialup) connections: if you have a static IP address, you should use the SNAT target. Masquerading is equivalent to specifying a mapping to the IP address of the interface the packet is going out, but also has the effect that connections are *forgotten* when the interface goes down. This is the correct behavior when the next dialup is unlikely to have the same interface address (and hence any established connections are lost anyway).

It takes one option:

--to-ports *<port>[-<port>]*

This specifies a range of source ports to use, overriding the default **SNAT** source port-selection heuristics (see above). This is only valid with if the rule also specifies **-p tcp** or **-p udp**).

A.8.9 REDIRECT

This target is only valid in the **nat** table, in the **PREROUTING** and **OUTPUT** chains, and user-defined chains which are only called from those chains. It alters the destination IP address to send the packet to the machine itself (locally-generated packets are mapped to the 127.0.0.1 address). It takes one option:

--to-ports *<port>[-<port>]*

This specifies a destination port or range or ports to use: without this, the destination port is never altered. This is only valid with if the rule also specifies **-p tcp** or **-p udp**).

A.9 DIAGNOSTICS

Various error messages are printed to standard error. The exit code is 0 for correct functioning. Errors which appear to be caused by invalid or abused command line parameters cause an exit code of 2, and other errors cause an exit code of 1.

A.10 BUGS

Check is not implemented (yet).

A.11 COMPATIBILITY WITH IPCHAINS

This **iptables** is very similar to ipchains by Rusty Russell. The main difference is that the chains **INPUT** and **OUTPUT** are only traversed for packets coming into the local host and originating from the local host respectively. Hence every packet only passes through one of the three chains; previously a forwarded packet would pass through all three.

The other main difference is that **-i** refers to the input interface; **-o** refers to the output interface, and both are available for packets entering the **FORWARD** chain.

iptables is a pure packet filter when using the default `filter' table, with optional extension modules. This should simplify much of the previous confusion over the combination of IP masquerading and packet filtering seen previously. So the following options are handled differently:

-j MASQ

-M -S

-M -L

There are several other changes in iptables.

A.12 SEE ALSO

The iptables-HOWTO, which details more iptables usage, and the netfilter-hacking-HOWTO which details the internals.

A.13 AUTHORS

Rusty Russell wrote iptables, in early consultation with Michael Neuling.

Marc Boucher made Rusty abandon ipnatctl by lobbying for a generic packet selection framework in iptables, then wrote the mangle table, the owner match, the mark stuff, and ran around doing cool stuff everywhere.

James Morris wrote the TOS target, and tos match.

Jozsef Kadlecsik wrote the REJECT target.

The Netfilter Core Team is: Marc Boucher, Rusty Russell.

B Styrning av Iptables via konfigurationsfiler

Exempel på konfigurationsfil som kan användas tillsammans med kommandot iptables-restore (B.1) samt filen som genereras av iptables-save (B.2) om B.1 använts för konfiguration.

B.1 iptables-restore

```
#Clearing any existing rules, setting default policy to DROP
-P INPUT DROP
-F INPUT
-P OUTPUT DROP
-F OUTPUT
-P FORWARD DROP
-F FORWARD
-F -t nat

# Delete all User-specified chains
-X

# Reset all IPTABLES counters
-Z

#Configuring specific CHAINS for later use in the ruleset
#  Creating a DROP chain
-N drop-and-log-it
-A drop-and-log-it -j LOG --log-level info
-A drop-and-log-it -j DROP

# INPUT chain

# loopback interfaces are valid.
-A INPUT -i lo -s 0/0 -d 0/0 -j ACCEPT

# local interface, local machines, going anywhere is valid
-A INPUT -i eth1 -s 192.168.0.0/24 -d 0/0 -j ACCEPT

#Remote interface, claiming to be local machines, IP spoofing, #deny
-A INPUT -i eth0 -s 192.168.0.0/24 -d 0/0 -j drop-and-log-it

# Allow any related traffic coming back to the MASQ server in
-A INPUT -i eth0 -s 0/0 -d 213.112.82.135 -m state –state \
ESTABLISHED,RELATED -j ACCEPT

# Catch all rule, all other incoming is denied and logged.
-A INPUT -s 0/0 -d 0/0 -j drop-and-log-it

# OUTPUT chain

# loopback interface is valid.
-A OUTPUT -o lo -s 0/0 -d 0/0 -j ACCEPT

# local interfaces, any source going to local net is valid
-A OUTPUT -o eth1 -s 213.112.82.135 -d 192.168.0.0/24 –j ACCEPT

# local interface, any source going to local net is valid
-A OUTPUT -o eth1 -s 192.168.0.1/24 -d 192.168.0.0/24 –j m ACCEPT

# outgoing to local net on remote interface, stuffed routing, #deny
```

```
-A OUTPUT -o eth0 -s 0/0 -d 192.168.0.0/24 -j drop-and-log-it

# anything else outgoing on remote interface is valid
-A OUTPUT -o eth0 -s 213.112.82.135 -d 0/0 -j ACCEPT

# Catch all rule, all other outgoing is denied and logged.
-A OUTPUT -s 0/0 -d 0/0 -j drop-and-log-it

# FORWARD chain

# Allow all connections OUT and only existing/related IN
-A FORWARD -i eth0 -o eth1 -m state –state ESTABLISHED,RELATED -j ACCEPT
-A FORWARD -i eth1 -o eth0 -j ACCEPT

# Allow and redirect connection for FTP
-A FORWARD -i eth0 -o eth1 -p tcp --dport 21 -m state –state \
NEW,ESTABLISHED,RELATED -j ACCEPT
-A PREROUTING -t nat -p tcp -d 213.112.82.135 --dport 21 –j DNAT –to \
192.168.15.25:21

# Catch all rule, all other forwarding is denied and logged.
-A FORWARD -j drop-and-log-it

# NAT: Enabling SNAT (MASQUERADE) functionality on eth0
-t nat -A POSTROUTING -o eth0 -j SNAT --to 213.112.82.135
```

B.2 iptables-save

```
# Generated by iptables-save v1.2.6a on Wed Dec  4 14:11:52 2002
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
:drop-and-log-it - [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -s 192.168.0.0/255.255.255.0 -i eth1 -j ACCEPT
-A INPUT -s 192.168.0.0/255.255.255.0 -i eth0 -j drop-and-log-it
-A INPUT -d 213.112.82.135 -i eth0 -m state --state RELATED,ESTABLISHED \
–j  ACCEPT
-A INPUT -j drop-and-log-it
-A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth1 -o eth0 -j ACCEPT
-A FORWARD -i eth0 -o eth1 -p udp -m udp --dport 9110 -m state –state \
NEW,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -o eth1 -p tcp -m tcp --dport 21 -m state –state \
NEW,RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -j drop-and-log-it
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -s 213.112.82.135 -d 192.168.0.0/255.255.255.0 -o eth1 -j ACCEPT
-A OUTPUT -s 192.168.0.0/255.255.255.0 -d 192.168.0.0/255.255.255.0 -o \
eth1 -j ACCEPT
-A OUTPUT -d 192.168.0.0/255.255.255.0 -o eth0 -j drop-and-log-it
-A OUTPUT -s 213.112.82.135 -o eth0 -j ACCEPT
-A OUTPUT -j drop-and-log-it
-A drop-and-log-it -j LOG --log-level 6
-A drop-and-log-it -j DROP
COMMIT
# Completed on Wed Dec  4 14:11:52 2002
# Generated by iptables-save v1.2.6a on Wed Dec  4 14:11:52 2002
*nat
:PREROUTING ACCEPT [705162:67684821]
:POSTROUTING ACCEPT [70348:8332912]
:OUTPUT ACCEPT [178746:13917648]
-A PREROUTING -d 213.112.82.135 -p udp -m udp --dport 9110 -j DNAT \
--to-destination 192.168.0.2:9110
-A PREROUTING -d 213.112.82.135 -p tcp -m tcp --dport 21 -j DNAT \
--to-destination 192.168.0.2:21
-A POSTROUTING -o eth0 -j SNAT --to-source 213.112.82.135
COMMIT
# Completed on Wed Dec  4 14:11:52 2002
```