Datavetenskap

Tony Bergh

Håkan Bergmark

Fjärradministration av Iptables

Säker kommunikation

Examensarbete, C-nivå

2002:05

Fjärradministration av Iptables

Säker kommunikation

Tony Bergh

Håkan Bergmark

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.

Tony Bergh

Håkan Bergmark

Godkänd, 2002-06-04

Handledare: Hans Hedbom

Examinator: Stefan Lindskog

Sammanfattning

Detta examensarbete syftar till att tillhandahålla en kommunikationsmodul för att kunna fjärradministrera brandväggen i Linux på ett säkert sätt. Iptables är Linux' senaste verktyg för att konfigurera den inbyggda brandväggen, Netfilter. Dokumentet inleder med att beskriva en brandväggs uppgifter och betydelse i ett nätverk. En analys av olika hot mot datakommunikation genomförs också. Detta följs av en genomgång av olika medel man kan använda sig av för att skydda kommunikationen, till exempel autentisering och kryptering. Vi analyserar tre olika säkerhetsprotokoll, jämför dem och bestämmer oss för att använda ett av dem. De tre protokollen är SSL, IPSec och SSH. Vi beskriver hur vi har löst uppgiften med hjälp av RMI som kommunikationsteknik och IPSec. Motivationer till valen av IPSec och RMI ges också. Till sist beskrivs hur projektet gick och vad arbetet har nått för resultat. Rapporten innehåller även flera bilagor vilka bland annat beskriver hur man får stöd för IPSec i Linux och Windows.

Remote Administration of Iptables

Secure Communication

Abstract

The purpose of this bachelor's project is to supply a communications module with which one can remotely administrate the firewall in Linux in a secure manner. Iptables is Linux' latest tool for configuration of the built-in firewall, Netfilter. Therefore, the document begins with a description of a firewalls tasks and importance in a network. An analysis of different kinds of threats that exist to data communication is also implemented. This is followed by a review of different methods one can use to protect the communication, for example authentication and encryption. We analyse three different security protocols, compare them and decide to use one of them. The three protocols are SSL, IPSec and SSH. We describe how we have solved the task with the use of RMI as communications technique and IPSec. Motivations to why IPSec and RMI have been chosen are also given. We finish our report by describing how the project went and the results of the same. The report also contains several appendices that among other things describe how to get IPSec support in Linux and Windows.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Det här arbetet utförs under våren 2002 i samarbete med Veriscan Security AB och ingår i ett större projekt som syftar till att kunna fjärradministrera en brandvägg i Linux-miljö. Projektet är uppdelat i tre mindre delar. De tre delarna är: grafiskt användargränssnitt (GUI), verifiering av regelverk och säker kommunikation. Vår del är säker kommunikation.

1.1 Bakgrund

Veriscan Security AB är ett IT-säkerhetsföretag som fungerar som en oberoende part åt organisationer vad gäller IT-säkerhet. Veriscan Security AB är knutet till Inova Science Park vid Karlstads universitet. Inova Science Park är en del av Swede Park. Karlstads universitet är specialiserade på IT-säkerhet.

Veriscan Security AB har utarbetat ett projekt som syftar till att skapa en miljö där en brandväggsadministratör ska kunna fjärradministrera den inbyggda brandväggen i Linux (Netfilter). Detta betyder att en administratör ska kunna styra brandväggen över ett nätverk, till exempel Internet. Brandväggen styrs med hjälp av Iptables, som är Linux' senaste verktyg för att sätta upp brandväggens regelverk (ingår från och med 2.4-kärnan).

Iptables är kommandostyrt och behöver ett grafiskt gränssnitt för att vara användarvänligt, detta är GUI-gruppens uppgift. Regelverket, som man vill införa i brandväggen, måste kunna verifieras så att funktionaliteten säkerställs och regelverket fungerar som man tänkt sig innan det tas i drift. Detta är verifieringsgruppens uppgift. Eftersom ett nätverk är osäkert krävs att kommunikationen sker på ett säkert sätt. Detta för att säkerställa att obehöriga ej får tillgång till information om brandväggsreglerna och om hur det interna nätverket är uppbyggt. Man måste också kunna skydda sig från att obehöriga kan ändra reglerna i brandväggen via fjärrstyrningen.

1.2 Mål

Syftet med vår del av projektet är att skapa en kommunikationsmodul, som skall användas för att överföra information mellan administratör och brandvägg på ett säkert sätt. Detta innebär

att vi måste utvärdera olika befintliga protokoll och implementationer för säker överföring. De alternativ som vi ska titta närmare på är Secure Sockets Layer (SSL), IP Security (IPSec) och Secure Shell (SSH). Efter val av säkerhetsmetod och val av programspråk ska vi sedan implementera modulen. Modulen ska testas ihop med de andra delarna av projektet för att säkerställa att miljön är funktionsduglig.

1.3 Disposition

Vi har valt att dela upp rapporten på så sätt att vi börjar beskriva allmänt vad en brandvägg är och vilket syfte den har. Sedan kommer vi att beskriva vilka typer av hot som finns mot kommunikationen och sätt att skydda sig mot dessa. Vi kommer också att beskriva de tre säkerhetsprotokollen SSL, IPSec och SSH. Efter detta ska vi gå igenom hur vi har tänkt lösa uppgiften och vilket säkerhetsprotokoll vi väljer att lösa uppgiften med. Vi avslutar med att testa systemet samt att sammanfatta projektet med sådant som har gått bra och sådant som gått mindre bra. Rapporten omfattar också en mängd bilagor som innefattar allt från konfigurering av IPSec till sekvensdiagram.

2 Brandväggar

En brandvägg kan ses som ett slags filter mellan det interna nätverket och omvärlden. Brandväggens uppgift är att skydda nätverket mot attacker utifrån. Brandväggen ska vara den enda kontaktpunkten mellan det inre nätverket och omvärlden. Detta medför att det blir enklare att upprätta säkerhets-policies, eftersom de endast behöver specificeras på ett ställe. Det blir också enklare att logga Internettrafiken och därmed är det lättare att upptäcka misstänkt trafik och intrångsförsök. Typiska uppgifter för en brandvägg är:

- Accesskontroll baserat på sändarens eller mottagarens adresser.
- Accesskontroll baserat på den begärda tjänsten.
- Gömma det interna nätverket för att till exempel hemlighålla nätverkets topologi, adresser med mera.
- Autentisering baserat på trafikens ursprung.
- Loggning av Internetaktiviteter.

Man brukar skilja på två grundläggande typer av brandväggar, nämligen paketfiltrerande brandväggar och proxyservrar. Brandväggar kan även ha olika arkitektur, detta beskriver vi översiktligt i kapitel 2.3.

2.1 Paketfiltrerande brandväggar

Paketfiltrerande brandväggar fungerar på så sätt att de släpper igenom respektive stoppar paket med hjälp av en uppsättning regler. Dessa regler kan till exempel baseras på mottagaradress och/eller sändaradress, typ av protokoll och intern eller extern uppkoppling. Här är det vanligt att det som inte uttryckligen är förbjudet är tillåtet, detta kan skilja sig åt mellan olika tillverkare.

Linux' interna brandvägg, som vi skall använda oss av i projektet, tillhör den här typen av brandvägg.

2.2 Proxyservrar

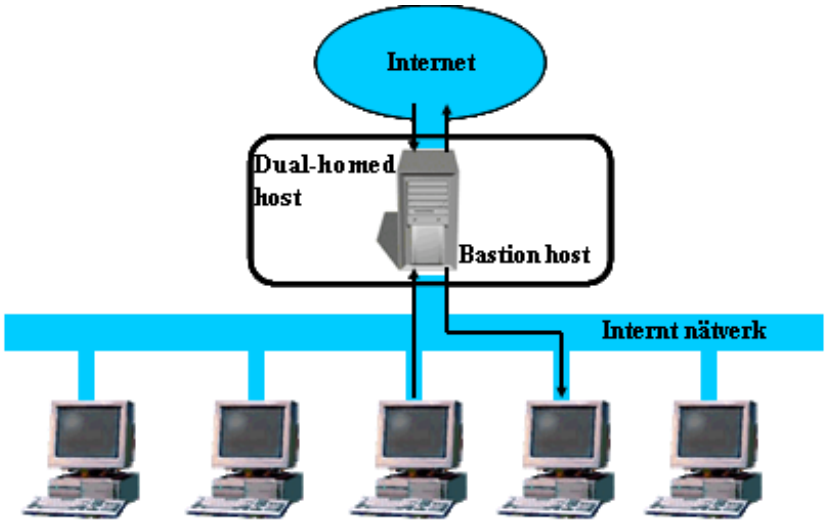
Dessa brandväggar arbetar på en högre nivå än de paketfiltrerande. Det fungerar så att klienten kopplar upp sig mot proxyservern som i sin tur kopplar upp sig mot servern. Detta sker helt transparent. Klienten tror att den pratar med servern när den i själva verket pratar med proxyservern. Detsamma gäller åt andra hållet för servern. Istället för att kontrollera alla kombinationer av olika paket som skall spärras eller släppas igenom behöver man endast bestämma vilka applikationer som får köras genom brandväggen.

2.3 Arkitektur

Detta avsnitt handlar om olika sätt brandväggar kan vara konfigurerade på rent hårdvarumässigt. Beroende på vilket skydd man vill erbjuda sitt nätverk finns det flera olika möjligheter att bygga upp brandväggen. Man brukar dela upp dessa i tre olika typer av brandväggsarkitekturer.

2.3.1 Dual-homed host

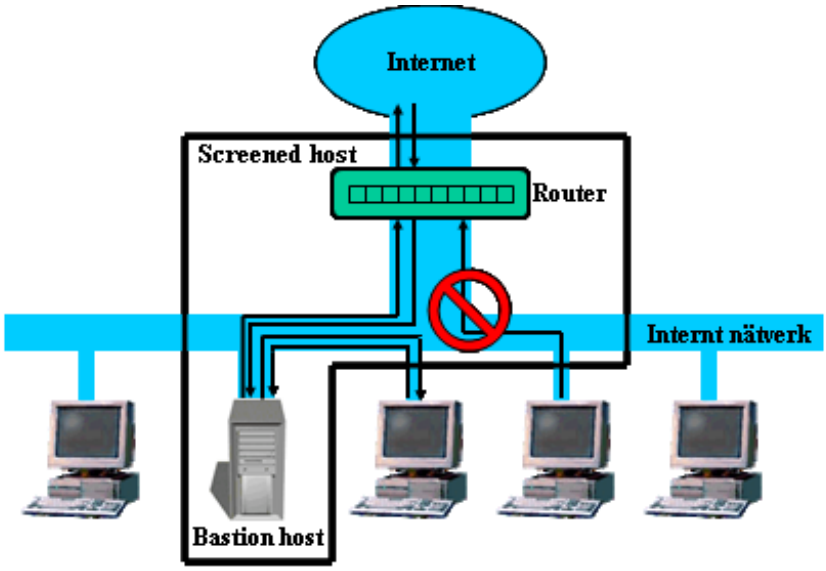
Den här arkitekturen består av en extra noggrant säkrad dator, en så kallad bastion host, vilken har åtminstone två nätverksgränssnitt, se Figur 2.1. Ett av nätverksgränssnitten är anslutet till Internet och de andra är anslutna till interna nätverk. All trafik kommer alltså alltid att passera just den datorn. Därför kan man lätt kontrollera och specificerar vilken typ av trafik som får passera. [3, 11]



Figur 2.1: Dual-homed host.

2.3.2 Screened host

Denna arkitektur består av en paketfiltrerande router, en så kallad screened router, samt en bastion host, se Figur 2.2. Routern filtrerar och dirigerar trafiken till bastion hosten, som i sin tur förmedlar vidare trafiken till rätt klient. Detsamma gäller ut från det interna nätverket. Den här arkitekturen fungerar på samma sätt som en dual-homed host, förutom att man låter den paketfiltrerande routern ta hand om all paketfiltrering. Bastion hosten är alltså den enda dator som klienter från Internet kan koppla upp mot. [3, 11]

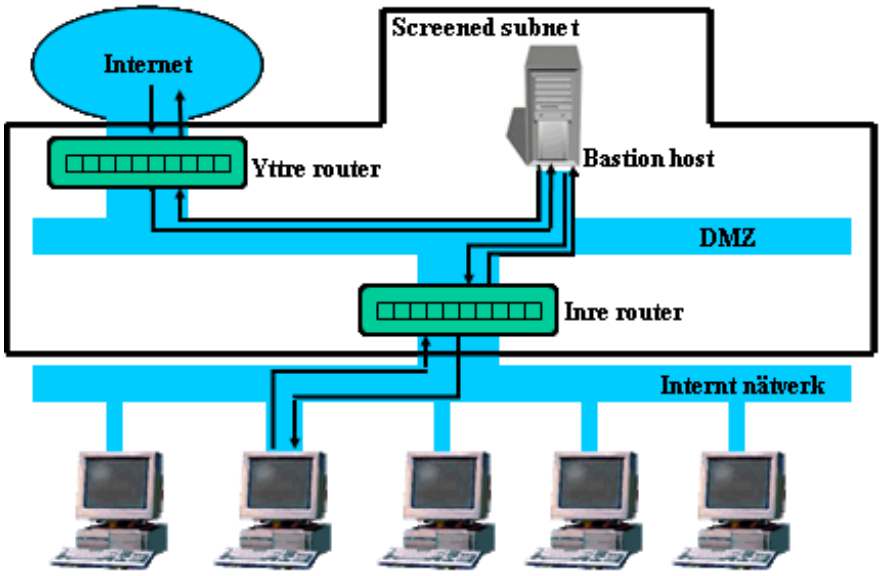


Figur 2.2: Screened host.

2.3.3 Screened subnet

Här har man en yttre router som arbetar mot Internet och en inre router som arbetar mot det inre nätverket, se Figur 2.3. Nätet som de två routrarna innesluter bildar en så kallad demilitarized zone (DMZ), där det endast finns bastion hostar. Dessa erbjuder vissa tillåtna tjänster mot Internet och det interna nätverket. Den yttre routern sköter filtreringen mot DMZ:at medan den inre sköter filtreringen mot det bakomliggande nätet. Paketfiltreringsreglerna är oftast hårdare specificerade för den inre routern än för det yttre. Detta på grund av att vissa tjänster ska kunna nås genom den yttre routern utifrån. Fördelen med den här arkitekturen är att Internet endast känner till DMZ:at. Det inre nätverket känner även det endast till DMZ:at och måste därmed gå via bastion hosten i DMZ:at för att kontakta Internet. [3, 11]

Den här arkitekturen erbjuder den högsta säkerheten av de tre. Detta på grund av att den ytterligare isolerar det interna nätverket från Internet och ger skydd på djupet.



Figur 2.3: Screened subnet.

3 Hot mot datakommunikation

Det finns många hot mot datakommunikation eftersom överföringsmediet i sig är osäkert. Vi kommer i detta kapitel att gå igenom en del av de många faror som finns. De kan delas upp i två större kategorier, passiva och aktiva attacker.

3.1 Passiva attacker

Passiva attacker består av att man försöker avlyssna och övervaka trafiken. På så sätt hoppas man kunna få information om vad som skickas. Det finns två typer av passiva attacker: avlyssning och trafikanalys.

Passiva attacker är väldigt svåra att upptäcka eftersom de inte förändrar data på något sätt. Man hanterar passiva attacker på så sätt att man förhindrar dem istället för att upptäcka dem.

3.1.1 Avlyssning

Avlyssning innebär att obehöriga personer kan läsa känslig eller hemlig information direkt från det skickade meddelandet, till exempel ett e-postmeddelande som skickas i klartext. [17]

3.1.2 Trafikanalys

Trafikanalys är när en förövare kontrollerar trafiken, till exempel hur långa meddelanden som skickas, hur ofta meddelanden skickas eller hur meddelanden skickas. På detta sätt hoppas förövaren kunna lista ut vilken typ av kommunikation som förs över. [17]

3.2 Aktiva attacker

Aktiva attacker innebär att förövaren skickar falska meddelanden eller ändrar data i meddelanden. Attackerna kan delas in i fyra underkategorier, maskering (masquerade), avspelning (replay), manipulation och denial of service (DoS).

Aktiva attacker är svåra att skydda sig helt emot, så man använder sig av taktiken att försöka upptäcka dessa attacker istället för att försöka förebygga dem.

3.2.1 Maskering

Maskering är när någon utger sig för att vara någon annan. Detta kan användas för att skydda det interna nätverket från den yttre världen, så kallad NAT:ning. Detta kan även användas i onda syften, till exempel för att komma över information som man inte har rätt till. Den vanligaste formen av maskering är helt enkelt att ta över någons konto och på så sätt utge sig för att vara någon annan. Ett annat exempel på maskering är IP-spoofing. Där gör förövaren egna IP-datagram där IP-adresserna är förfalskade så att mottagaren tror att paketen kommer från någon man litar på. På så sätt kan förövaren få tillgång till resurser som normalt inte är tillgängliga, till exempel kan förövaren eventuellt komma igenom en paketfiltrerande brandvägg. [17]

Maskering används ofta tillsammans med andra attacker, till exempel IP-spoofing tillsammans med en DoS-attack, se 3.2.4.

3.2.2 Avspelning

Avspelning innebär att förövaren kopierar meddelanden som skickas mellan två parter. Meddelandena skickas om till mottagaren, i syfte att få denne att tro att de kommer från en part som den litar på. Om meddelandena till exempel innehöll användarnamn och lösenord kan förövaren använda dessa för att få tillgång till resurser hos mottagaren. [17]

3.2.3 Manipulation

Manipulation är när förövaren snappar åt sig ett eller flera meddelanden och ändrar innehållet, fördröjer sändningen eller ändrar deras ordning. På så sätt har förövaren gjort en otillåten åtgärd och detta kan ha förödande effekter. [17]

3.2.4 Denial of service

Denial of servise-attacker (DoS-attacker) omfattar allt från att försöka sänka enskilda resurser till att sänka ett helt nätverk. Grundtanken är att det skickas massvis med skräpmeddelanden som syftar till att få resursen att bearbeta dem istället för att utföra vettigt arbete. Detta kan ske på många olika sätt, exempel på dessa är: smurf-attack, Ping of Death, Teardrop och SYN flooding.

En smurf-attack är när en förövare skickar ICMP echo request-meddelanden (PING-meddelanden) till broadcast-adresser. Varje broadcast-adress stödjer ett stort antal olika klienter, alltså kan ett enda PING-meddelande generera ett lika stort antal svar.

Meddelandenas svarsadress är förfalskad (spoofad) till offrets adress. På så sätt kommer offrets nät att fyllas med svarsmeddelanden och försvåra eller förhindra övrig kommunikation. [22]

Ping of Death utnyttjar kända buggar i TCP/IP-implementationer. Man tillverkar PING-meddelanden vars IP-datagram är större än 65536 bytes. IP-specifikationen säger att IP-datagram får vara maximalt 65536 bytes. När ett offer då får IP-datagram som är större än maxstorleken kan systemet krascha, hänga sig eller starta om. Detta är åtgärdat i de nya operativsystemen och det finns patchar till de äldre operativsystemen som fixar problemet. [22]

Teardrop utnyttjar, precis som Ping of Death, buggar i TCP/IP-implementationer. Det handlar här om att man utnyttjar fragmenteringen av IP-datagrammen på Internet. Varje sådant fragment ser ut precis som originalpaketet förutom att offseten anger vilken del av originalpaketet fragmentet är. Det är här man utnyttjar en bugg i implementationerna, man ändrar offseten på fragmenten så att de överlappar varandra. När mottagaren sedan skall sätta ihop fragmenten till originalmeddelandet kan en del system krascha, hänga sig eller starta om. [22]

SYN-flooding är när en förövare sänder TCP SYN-paket (uppkopplingsbegäran) till servern där varje paket har en spoofad IP-adress. Servern reserverar då minne för TCP-uppkopplingen och slutför det andra steget i TCP-handskakningen genom att skicka ett svarsmeddelande till den spoofade IP-adressen. Eftersom den dator som har den spoofade IP-adressen inte har skickat någon uppkopplingsbegäran kommer den heller inte att genomföra det tredje steget i TCP-handskakningen. Eftersom förövaren översvämmar servern med falska uppkopplingsbegäran kommer snart serverns minne att ta slut. Servern kan därmed inte ta emot flera uppkopplingsbegäranden, vilket innebär att servern är satt ur spel. [22]

4 Skydd för datakommunikation

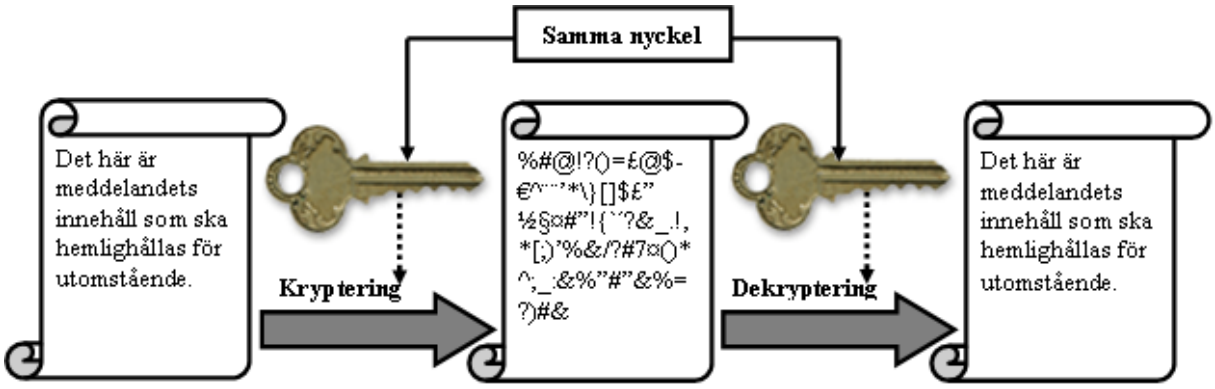
Uppenbarligen behöver man skydda kommunikationen på något sätt. Det finns olika sätt att skydda sig mot olika hot, till exempel vill man kanske kunna lita på att den man kommunicerar med är den som denne utger sig för att vara. Ett annat exempel är att man kanske vill hemlighålla informationen i meddelandet för obehöriga. Dessa två exempel kan åstadkommas genom autentisering respektive kryptering.

4.1 Kryptering

Kryptering är när man på matematiskt vis förvränger information så att den inte kan läsas av obehöriga. Vanligen används kryptering för att förmedla känslig information över ett osäkert medium. Sändaren måste då kryptera meddelandet på ett sådant sätt att mottagaren kan återskapa den ursprungliga informationen. Det förekommer också fall där man inte behöver kunna återskapa informationen utan man bara behöver hemlighålla den. Vi har valt att beskriva tre olika krypteringsmetoder och vi ger även exempel på algoritmer för varje metod.

4.1.1 Symmetriska algoritmer

De symmetriska algoritmerna bygger på att de två parterna (alltså sändare och en eller flera mottagare) har en gemensam hemlighet, en så kallad symmetrisk eller hemlig nyckel. Denna nyckel används vid såväl kryptering som dekryptering, därför är det extremt viktigt att nyckeln hålls hemlig och att endast de två parterna känner till den. Se Figur 4.1. Ett stort problem med symmetriska algoritmer är distribueringen av nycklarna. Ofta löser man distribueringen av nycklarna genom att använda sig av asymmetrisk algoritm, se kapitel 4.2.2, eller genom att låta en betrodd tredje part (key distribution centre, KDC) distribuera nycklarna.



Figur 4.1: Symmetrisk kryptering.

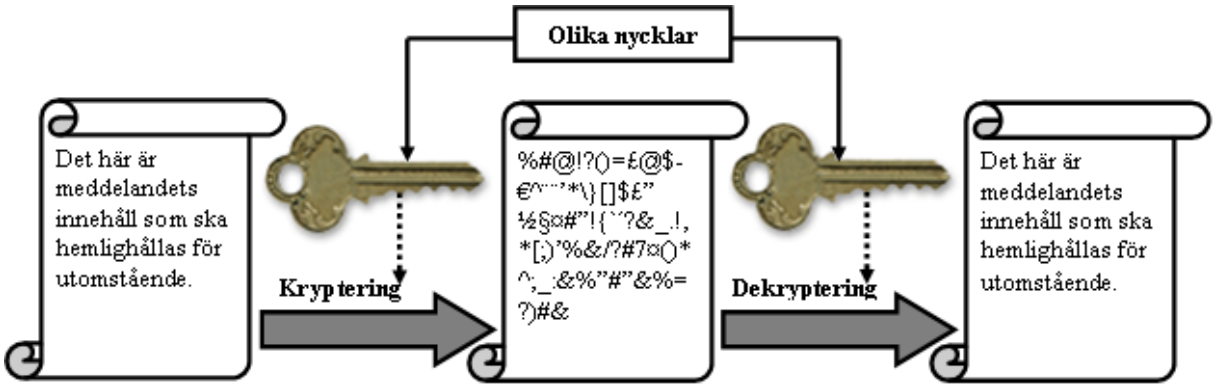
Exempel på symmetriska algoritmer: DES [10], 3DES [10], IDEA [17], BLOWFISH [20], SKIPJACK [13].

4.1.2 Asymmetriska algoritmer

I asymmetriska algoritmer använder man sig av två olika nycklar, en publik nyckel och en privat nyckel. Den publika nyckeln distribueras fritt medan den privata nyckeln hålls hemlig. Anledningen till detta är att man använder sig av mottagarens publika nyckel när man krypterar ett meddelande. Det enda sättet att dekryptera detta meddelande är sedan att använda sig av mottagarens privata nyckel, som endast mottagaren skall känna till. På det här sättet undviker man problemet, som uppstår vid distributionen av hemliga nycklar, i symmetriska algoritmer. Se Figur 4.2.

För att säkerställa att den publika nyckeln verkligen hör till den mottagare man har tänkt sig, så brukar nycklarna certifieras med ett nyckelcertifikat. Detta certifikat länkar samman en publik nyckel med en viss användare.

Ett problem med asymmetriska algoritmer är att de tenderar att vara väldigt beräknings-intensiva. Därför används asymmetriska algoritmer ofta vid distribution av symmetriska nycklar för att sedan kunna använda en symmetrisk algoritm vid kommunikationen.



Figur 4.2: Asymmetrisk kryptering.

Exempel på asymmetriska algoritmer: RSA, DSS. [17]

4.1.3 Envägskryptering

När man inte är intresserad av, eller inte behöver få fram, ursprungsinformationen kan man använda sig av envägskryptering (även kallat hashfunktion). Det typiska kännetecknet för en envägskryptering är att när man skickar in ett värde genereras ett resultat med konstant längd. Genom resultatet skall det inte gå att beräkna ursprungsinformationen. En annan viktig egenskap för dessa funktioner är att två olika invärden inte får ge samma utvärde, det vill säga utvärdet måste vara unikt för ett invärde.

Envägskryptering används bland annat till att lagra lösenord och verifiera ett meddelandes integritet. Vid lagring av envägskrypterade lösenord, envägskrypteras inmatade lösenord och resultatet jämförs med de lagrade värdena för att söka efter matchningar. På så sätt behöver man inte lagra lösenord i klartext. För att kontrollera ett meddelandes integritet kan meddelandet, innan sändning, köras genom en hashfunktion. Resultatet krypteras och sänds tillsammans med meddelandet till mottagaren. Mottagaren dekrypterar det bifogade hashade värdet kör sedan meddelandet genom en likadan hashfunktion. De två resultaten jämförs, är de lika så är meddelandet oförändrat. [1, 17]

Exempel på envägsalgoritmer: MD5, MD4, MD2, RIPE-MD, SHA-1. [17]

4.2 Autentisering

En användare ska kunna lita på att den andra parten i en kommunikation är den han utger sig för att vara, detta sker genom så kallad autentisering. Beroende på vilken säkerhet man vill

uppnå, kan man autentisera med 1-faktor-, 2-faktor- eller 3-faktorautentisering. Detta innebär att den person eller entitet som skall autentiseras måste ha en, två eller tre egenskaper. Dessa egenskaper kan vara: något man vet, något man har eller vart man är osv. Det finns olika former av autentisering, till exempel genom certifikat eller digital signatur.

4.2.1 Certifikat

Ett certifikat är en samling data som är utfärdat och digitalt signerad av en certifikatsutfärdare (certification authority, CA). Certifikatet binder en viss entitet till viss information. Ett certifikat används ofta för att binda en publik nyckel, i ett asymmetriskt nyckelpar, till en viss person eller organisation. Certifikat brukar innehålla information om användaren såsom namn och e-postadress. Ett certifikat brukar även specificera giltighetsperiod för certifikatet samt innehålla bland annat namnet på certifikatsutfärdaren.

För att garantera certifikatets giltighet kan man kontrollera certifikatsutfärdarens riktighet genom att verifiera certifikatsutfärdarens certifikat. Detta skall vara signerat av en överordnad certifikatsutfärdare, som i sin tur är underställd ytterligare en certifikatsutfärdare osv. Certifikatsutfärdarna är alltså uppbyggda i en hierarkiskt ordnad trädstruktur. Det hela bygger på att ”rotcertifikatsutfärdaren” är betrodd av alla parter.

4.2.2 Digital signatur

En digital signatur bekräftar att en sändare är den han utger sig för att vara och därmed den som har skickat meddelandet. Man kan använda både symmetriska och asymmetriska nyckelpar för att skapa digitala signaturer. Om man använder symmetriska nyckelpar måste det finnas en tredje part som man kan lita på. Det är den tredje parten som distribuerar nycklarna i det fallet. Vanligast är dock att man använder sig av asymmetriska nyckelpar, vi ger två exempel på detta nedanför.

Ett exempel på en digital signatur är när den ena parten krypterar meddelandet med sin privata nyckel. Då kan meddelandet dekrypteras med partens publika nyckel. Detta innebär att den som har krypterat meddelandet måste ha haft tillgång till den privata nyckeln. Den här formen av digital signatur erbjuder endast autentisering, den erbjuder inte konfidentialitet.

En vanligare form av digital signatur är att man först räknar ut en kontrollsumma som beror av dokumentet. Kontrollsumman krypteras sedan med den privata nyckeln. Efter detta

krypterar man hela meddelandet, inklusive kontrollsumman, med mottagarens publika nyckel. På detta sätt erbjuder man både autentisering och konfidentialitet. Det är bara mottagaren som kan dekryptera meddelandet, med sin privata nyckel, och sedan kan dekryptera och verifiera kontrollsumman med sändarens publika nyckel. Om kontrollsumman stämmer vet mottagaren att meddelandet kommer från den som säger sig ha skickat det. Mottagaren vet även att meddelandet inte har förändrats på vägen.

5 Säkerhetsprotokollen SSL, IPSec och SSH

I det här kapitlet kommer vi att beskriva tre olika metoder/implementationer som erbjuder säker kommunikation genom kryptering och autentisering. De tre är IP Security (IPSec), Secure Sockets Layer (SSL) samt Secure Shell (SSH).

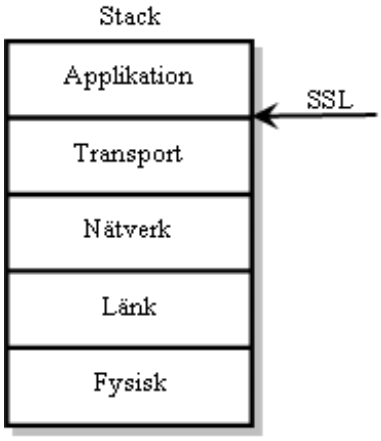
5.1 Secure Sockets Layer

Secure Sockets Layer-protokollet (SSL-protokollet) har utvecklats av Netscape, och då huvudsakligen för att skydda WWW-trafik och brukar ofta användas ihop med webbprotokollet http. IETF har på senare år tagit fram en Internetstandard för ett protokoll, som har haft SSL version 3 som bas, vilket kallats Transport Layer Security (TLS) [9, 14].

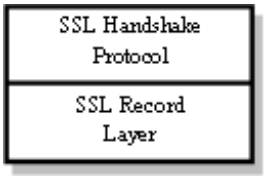
SSL arbetar ovanpå transportlagret, under applikationslagret, se Figur 5.1 och Figur 5.2. SSL körs över TCP-förbindelser vilket medför att det erhåller TCPs egenskaper gällande förbindelseorientering och pålitlighet. Man kan se om en applikation använder sig av SSL genom att se om Internetadressen börjar med *https://*, vilket indikerar en SSL-krypterad session.

SSL erbjuder stark kryptering för att tillhandahålla konfidentialitet, dataintegritet och autentisering mellan två applikationer. SSL stödjer en mängd algoritmer vid kryptering av meddelanden. Meddelanden krypteras med en symmetrisk algoritm som bestäms i förhand.
SSL stödjer bland annat följande krypteringsalgoritmer: IDEA, DES och 3DES. Certifikat används för autentiseringen av parterna i SSL-kommunikationen.

Eftersom SSL används på socket-nivå säkras kommunikationen mellan två applikationer istället för mellan två datorer. På grund av att SSL återfinnes så högt upp i protokollstacken måste varje applikation som vill använda sig av SSL ange detta explicit genom att anropa SSL connect istället för TCP connect. SSL-protokollet utgörs av två större komponenter nämligen SSL Handshake Protocol och SSL Record Layer. [4, 8, 14]



Figur 5.1: SSL's placering i Internetprotokollstacken.



Figur 5.2: SSL's protokoll.

5.1.1 SSL Handshake Protocol

För att kunna åstadkomma en SSL-förbindelse måste de kommunicerande applikationerna utbyta information för att autentisera sig för varandra och komma överens om metoder för nyckelutbyte och krypteringsalgoritmer. Allt detta sker med hjälp av handskakningsprotokollet vilket därmed är den mest komplexa delen i SSL-protokollet. Här följer en fullständig handskakningsprocess och alla dess steg.

1. Klienten initierar körningen med protokollet genom att skicka ett client_hello-meddelande till servern. Detta meddelande innehåller bland annat det senaste versionsnumret av SSL som klienten stödjer, ett slumptal som är till för att förhindra avspelningsattacker samt om man ska använda sig av en tidigare förbindelse eller skapa en ny. Meddelandet innehåller också information om vilka krypterings- och komprimeringsalgoritmer som klienten kan hantera.
2. Servern svarar med ett server_hello-meddelande som har samma utseende som klientens meddelande. Servern anger det högsta versionsnumret av SSL som båda har stöd för. Den väljer också ut en krypterings- och komprimeringsalgoritm från klientens urval som den själv tillåter.

3. Servern skickar sedan sitt certifikat med ett certificate-meddelande för att på så sätt autentisera sig för klienten. Sedan skickas alternativt ett server_key_exchange-meddelande beroende på vilken för nyckelutbytesalgoritm man har tänkt använda sig av. Efter detta kan ytterligare ett valbart meddelande skickas, nämligen certificate_request-meddelandet. Detta skickas om servern i sin tur också kräver att klienten ska autentisera sig. Till sist skickas ett server_hello_done-meddelande för att signalera att server hello-fasen har avslutats.
4. Klienten verifierar att serverns certifikat är korrekt och att alla val är korrekta angående kryptering och versionsnummer etcetera. Om allt är korrekt så skickas nya meddelanden tillbaka till servern. Om till exempel servern har begärt klientens certifikat så skickas det med ett certificate-meddelande. Nästa meddelande som skickas är client_key_exchange-meddelandet. Detta meddelande innehåller en nyckel som klienten skapat slumpmässigt. Denna krypteras och skickas till servern. Med hjälp av denna nyckel, och tillsammans med slumptalet som utbyttes i hello-fasen, kan sedan klienten och servern beräkna en huvudnyckel som kommer att användas för att skapa resterande nycklar som krävs i kommunikationen. Om klienten har skickat sitt certifikat kommer även ett certificate_verify-meddelande att skickas för att verifiera certifikatet.
5. Nästa meddelande, change_cipher_spec, tillhör egentligen inte handskakningsprotokollet utan Change Cipher Spec-protokollet men måste inkluderas för att kunna slutföra handskakningsprocessen. Detta meddelande talar om för servern att från och med nu så kommer all kommunikation att använda sig av de överenskomna algoritmerna. Till slut skickas finished-meddelande som markerar att handskakningen är slutförd från klientens sida. Detta meddelande autentiserar också alla tidigare skickade meddelanden exklusive change_cipher_spec-meddelandet.
6. Servern avslutar handskakningsprocessen med att även den skicka ett change_chiper_spec-meddelande samt ett finished-meddelande. [21]

5.1.2 SSL Record Layer

Efter att handskakningen är slutförd kan data börja skickas över SSL-uppkopplingen på ett säkert sätt. SSL Record Layer tillhandahåller två tjänster för SSL-kommunikationen nämligen konfidentialitet genom kryptering och integritet genom MAC. Det fungerar på så sätt att meddelanden som ska skickas delas upp i mindre block med storleken 2^{14} eller mindre, ett

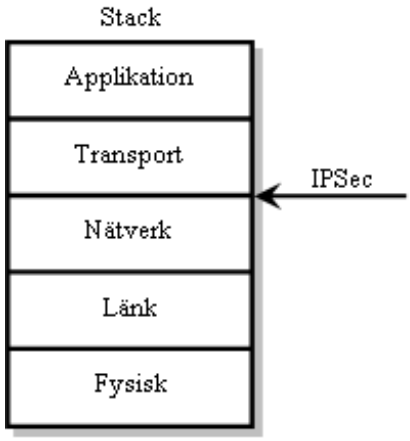
MAC-värde beräknas och adderas till blocket som sedan krypteras med överenskommen algoritm. Det krypterade datablocket erhåller sedan en header och kallas nu en record. Recorden sänds sedan som ett TCP-segment till mottagaren som i sin tur kan dekryptera och kontrollera integriteten för recorden, för att sedan sammanfoga blocken till ursprungsmeddelandet och leverera upp det till den avsedda applikationen.

5.2 IP Security

IP Security (IPSec) opererar, som kan uttydas av namnet, i samband med IP-lagret mellan transportlagret och nätverkslagret i protokollstacken, se Figur 5.3. Detta innebär att all kommunikation kommer att gå över IPSec om den andra parten även stödjer detta. Detta medför att IPSec är helt transparent för olika applikationer vare sig de är säkerhetsmedvetna eller ej. Eftersom IPSec körs direkt över IP som är förbindelselöst och opålitligt angående sändningen av IP-datagram, måste man lita på att överliggande protokoll sköter omsändning av datagram. IPSec stöds av både IPv4 och IPv6. IPv4 kräver dock en implementation av IPSec i protokollstacken, en så kallad bump in the stack. I IPv6 ingår IPSec i arkitekturen och behöver endast anges som ett extra val i IP-headern.

IPSec är inte ett protokoll i vanlig bemärkelse, utan är ett gemensamt namn för protokoll som erbjuder säkerhet på IP-nivån, av vilka de två viktigaste är Authentication Header-protokollet (AH) och Encapsulation Security Payload-protokollet (ESP). AH-protokollet erbjuder autentisering av sändaren, för att förhindra IP-spoofing, samt integritet, men inte konfidentialitet. ESP-protokollet däremot erbjuder både konfidentialitet och integritet samt valmöjlighet angående autentisering. Vi kommer att gå igenom AH och ESP lite mer i detalj senare i texten och då se fördelarna respektive nackdelarna med respektive protokoll. Före säker sändning kan ske mellan två parter måste de två först utföra en handskakning och sätta upp en logisk koppling mellan datorerna på nätverkslagret. Den logiska kopplingen kallas för en Security Association, en så kallad SA. En SA är enkelriktad vilket innebär att om data ska skickas i båda riktningarna måste två SA:n etableras. I SA:n anger man bland annat vilket säkerhetsprotokoll som ska användas, AH eller ESP. I en SA anges också mottagarens IP-adress samt en bitsträng, kallad Security Parameter Index (SPI), som fungerar som en unik identifierare för SA:n. Om man vill använda sig av IP-datagram som både använder sig av AH och ESP, nästlat, måste man ange två SA:n i vardera riktningarna. Vi kommer senare att ge flera exempel på olika strukturer och nästlingar av datagram. För att denna logiska

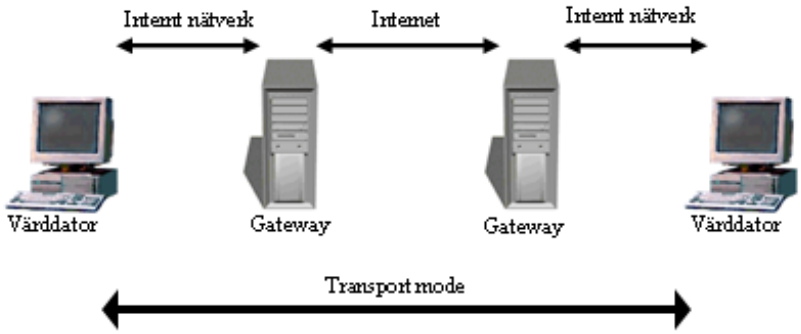
koppling ska kunna etableras på ett automatiskt och enkelt sätt, har man specificerat flera protokoll som skall lösa detta. Internet Key Exchange (IKE) är ett ramverk för att förhandla om och tillhandahålla autentiserat nyckelmaterial för att på så sätt kunna sätta upp SA:n på ett säkert sätt. För detta använder sig IKE av de två protokollen Internet Security Association and Key Management-protokollet (ISAKMP) och Oakley Key Determination-protokollet (Oakley). ISAKMP tillhandahåller ett ramverk för nyckelhantering på Internet. Det tillhandahåller protokoll för förhandling av säkerhetsattribut men specificerar inget speciellt nyckelutbytesprotokoll. Oakley är ett nyckelutbytesprotokoll baserat på Diffie-Hellman-algoritmen, men erbjuder högre säkerhet. Oakley är ett allmänt protokoll i den mening att det inte föreskriver några specifika format utan kan användas ihop med ett godtyckligt meddelandeprotokoll. Det finns två olika sätt IP-datagram kan skickas över IPSec, med så kallad trasport mode och tunnel mode. Vilket som ska användas bestäms med hjälp av vilken typ av arkitektur som kommunikationen går över samt vilken säkerhetsnivå man vill erbjuda. [6, 12, 14, 21]



Figur 5.3: IPSec's placering i Internetprotokollstacken.

5.2.1 Transport Mode

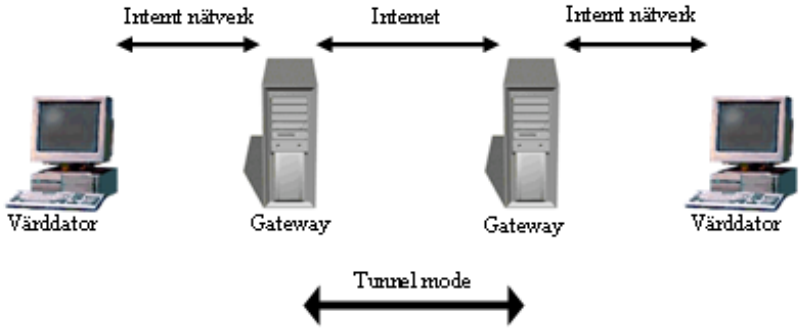
I transport mode skyddas i huvudsak endast det övre lagrets protokoll såsom TCP- eller UDP-segment. Det fungerar på så sätt att IPSec-headern (AH eller ESP) placeras efter IP-headern och före segmentet från det övre lagret. Därmed skyddas det övre lagrets segment medan IP-headern endast ges ett begränsat skydd beroende på om AH eller ESP används. Transport mode används vanligtvis mellan end-to-end-kommunikation mellan två datorer, till exempel mellan en klient och en server. Se Figur 5.4.



Figur 5.4: Transport mode.

5.2.2 Tunnel Mode

Om man istället använder sig av tunnel mode så erbjuds skydd för hela IP-datagrammet. Detta åstadkommes genom att man, efter att ha tillfört IPSec-headern, lägger hela datagrammet som en payload i ett nytt datagram. Det nya datagrammet kan erhålla helt nya ursprungs- och destinationsadresser. När datagrammen sedan skickas iväg över det osäkra mediet så syns inte den inre IP-headern vilket kan förebygga viss trafikanalys, man säger att det inre datagrammet färdas genom en tunnel från en punkt till en annan. Denna metod har som fördel att man kan kommunicera på ett säkert sätt mellan två nätverk genom att endast implementera IPSec i respektive näts gateway. Denna gateway behandlar inkommande och utgående datagram på ett korrekt sätt och förmedlar dem till avsedd mottagare. Se Figur 5.5.



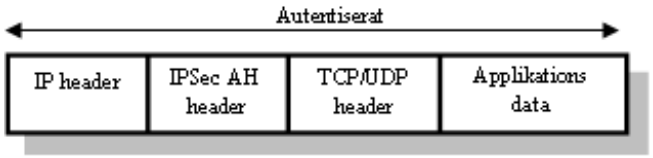
Figur 5.5: Tunnel mode.

5.2.3 Authentication Header

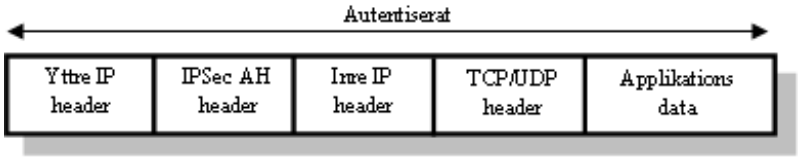
Som nämnts tidigare så erbjuder Authentication Header (AH) stöd för både dataintegritet och autentisering av IP-datagram men inte konfidentialitet. Mottagaren vet hur den ska behandla datagrammet genom att numret 51 anges i original-IP-headerns fält. Här anges annars normalt vilket protokoll som används av det överliggande lagret i det efterföljande segmentet, detta anges nu istället i AH-headern. Man kan ju då fråga sig varför man ska använda sig av AH

när, som nämnts tidigare, ESP kan erbjuda autentisering, integritet och konfidentialitet. Svaret är att man inte alltid behöver erbjuda konfidentialitet utan kan nöja sig med autentisering och integritet. Sanningen är att AH faktiskt erbjuder bättre autentisering och integritet för IP-datagram, eftersom ett MAC-värde beräknas för hela datagrammet, inklusive original-IP-headern, vilket inte sker för ESP. Dessutom är kryptering och dekryptering ett extra steg som kan vara tids- och beräkningskrävande, vilket kan dra ner hastigheten för kommunikationen. AH erbjuder också skydd mot avspelningsattacker genom att tillhandahålla ett sekvens-nummer för varje datagram. Anländande datagram kontrolleras sedan om de redan har mottagits, om så inte är fallet kontrolleras MAC-värdet. Om autentiseringen lyckas, markeras det specifika datagrammet som mottaget. Om däremot autentiseringen misslyckas eller datagrammet redan har markerats som mottaget, kastas det och händelsen kan sedan loggas. När det gäller MAC-algoritmer stöds både HMAC-MD5 och HMAC-SHA-1, båda använder de 96 första bitarna från de genererade nycklarna som MAC-värde. [12, 21]

Som avslutning ska vi beskriva och visa skillnaderna mellan transport mode och tunnel mode för AH. AH-headern läggs mellan IP-headern och IP-payloaden, MAC-värdet beräknas över hela IP-datagrammet, exklusive vissa fält i IP-headern som ändras under transporten över nätet. I tunnel mode läggs istället AH före IP-headern vilket sedan läggs i ett nytt IP-datagram med en ny header. Här autentiseras hela det inre datagrammet inklusive alla fält i det inre datagrammets IP-header. Autentiseringen täcker också det yttre datagrammets header men då endast på samma sätt som vid transport mode. Fördelen är som tidigare nämnts att den inre och yttre IP-headern kan innehålla olika sändar- och mottagaradresser, och på så sätt ses endast de adresserna i den yttre headern av routrarna i nätet. Se Figur 5.6 och Figur 5.7.



Figur 5.6: IP-paket skyddat med AH i transport mode.

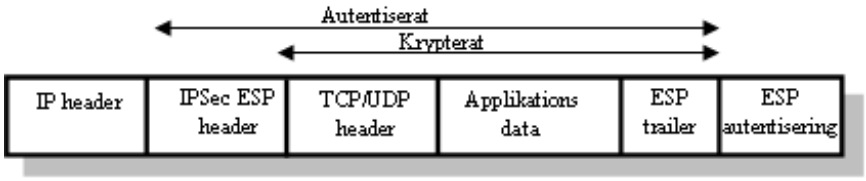


Figur 5.7: IP-paket skyddat med AH i tunnel mode.

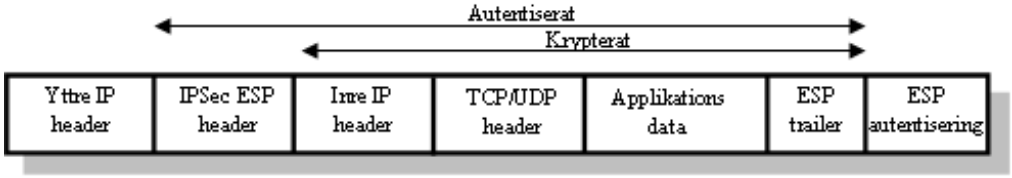
5.2.4 Encapsulating Security Payload

Encapsulating Security Payload-protokollet (ESP) används för att hemlighålla innehållet i IP-payloaden och på så sätt skydda datan från att läsas av obehöriga. Det stödjer konfidentialitet och kan även erbjuda autentisering om man så önskar. ESP har tilldelats numret 50 av Internet Assigned Numbers Authority (IANA), och anges på samma sätt som för AH för att meddela mottagaren att ett ESP-skyddat innehåll följer. ESP-protokollet stödjer en mängd krypteringsalgoritmer såsom 3DES, RC5, IDEA med flera. Dessutom stöds även samma MAC-algoritmer som för AH så att autentisering kan ske om man så önskar. På samma sätt som vid AH så skyddas man mot avspelningsattacker genom att hålla reda på vilka datagram som redan har mottagits. ESP-protokollet består av en ESP-header och en ESP-trailer samt alternativt ett ESP authentication-fält om autentisering ska stödjas. IP-datagrammets innehåll placeras mellan ESP-headern och ESP-trailern vilken alternativt följs av ESP authentication. Innehållet efter ESP-headern, inklusive ESP-trailern, krypteras sedan med vald krypteringsalgoritm. Alternativt räknas också ett MAC-värde ut, vilket täcker från ESP-headern till och med ESP-trailern. Detta innebär att MAC-värdet inte täcker lika stor del av IP-datagrammet som vid användandet av AH-protokollet, vilket också även täcker större delen av IP-headern. Detta betyder att ESP-protokollet inte erbjuder samma starka autentisering av IP-datagram som AH-protokollet. Därför brukar man ibland använda sig av olika kombinationer av SA:n för att erhålla både ESP- och AH-protokollets fördelar genom att nästla ESP- och AH-skyddade IP-datagram. Vi kommer att visa exempel på detta i nästa kapitel. Men först tänkte vi avsluta kapitlet om ESP-protokollet på samma sätt som föregående avsnitt genom att ta en titt på hur ESP används i transport- respektive tunnel mode. [12, 21]

När ESP används i samband med transport mode så placeras ESP-headern mellan IP-headern och IP-payloaden. Efter payloaden placeras ESP-trailern som alternativt följs av ESP authentification om autentisering skall stödjas. Detta innebär att payloaden samt ESP-trailern krypteras. Om autentisering har valts så kommer den att täcka ESP-headern, payloaden samt ESP-trailern. Däremot kommer inte IP-headern och ESP authentication-fältet att täckas av autentiseringen. Vid tunnel mode så placeras ESP-headern före IP-headern och allt detta placeras i ett nytt yttre datagram med en ny IP-header. Detta medför att även den inre IP-headern kommer att omfattas av krypteringen och autentiseringen. Se Figur 5.8 och Figur 5.9.



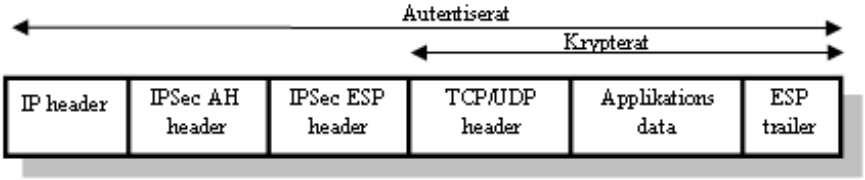
Figur 5.8: IP-paket skyddat med ESP i transport mode.



Figur 5.9: IP-paket skyddat med ESP i tunnel mode.

5.2.5 Kombinera olika SA

Eftersom en SA endast kan använda antingen AH- eller ESP-protokollet, måste man specificera flera SA:n om man vill använda sig av både AH och ESP för en kommunikation. Det kan man göra om man till exempel måste hemlighålla innehållet i IP-datagrammet men tycker att ESP-protokollet inte erbjuder tillräcklig autentisering. Lösningen är då att skapa två SA:n i båda riktningarna där den inre SA:n implementerar ESP och den yttre AH. Denna konstruktion kommer att få till följd att man erhåller samma konfidentialitet som med ESP och dessutom får AH's starkare autentisering och integritet. Figur 5.10 visar hur IP-paketen ser ut när man kombinerar AH och ESP.

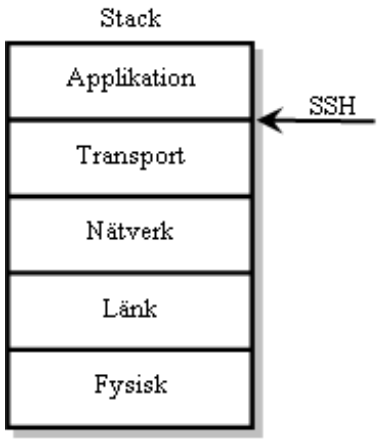


Figur 5.10: IP-paket skyddat av både AH och ESP i transport mode.

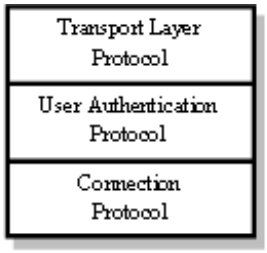
5.3 Secure Shell

Secure Shell (SSH) är ett protokoll för säker fjärrinloggning och filöverföring. SSH stödjer stark kryptering samt metoder för att autentisera användare. Det finns två versioner av SSH, version 1 och version 2, dessa är helt olika och därmed inte kompatibla. SSH version 1 utvecklades av en finländare vid namn Tatu Ylönen för att användas på UNIX-system. Det

utvecklades för att ersätta de helt osäkrade protokollen, rsh, rlogin, rcp, rdist och telnet, för fjärrkommunikation. SSH har sedan dess utvecklats och används nu flitigt för att göra kommunikationen säkrare. SSH stöds numera av flera olika operativsystem såsom Linux, Windows och Macintosh med flera. SSH arbetar över transportlagret och då normalt över TCP/IP, men stödjer även andra protokoll. Se Figur 5.11. SSH-protokollet (version 2) består av tre större komponenter, *Transport Layer protocol*, *User Authentication protocol* och *Connection protocol,* se Figur 5.12. [7]



Figur 5.11: SSH's placering i Internetprotokollstacken.



Figur 5.12: SSH's protokoll.

5.3.1 Transport Layer Protocol

Transport Layer-protokollet erbjuder autentisering av servern, konfidentialitet och integritet. När man här talar om autentisering avses autentisering endast av värddatorer, inte av användare. Detta protokoll erbjuder också komprimering om man så önskar, vilket medför att mängden data som behöver skickas kan minskas avsevärt och därmed snabba på kommunikationen. Protokollet förutsätter att det körs över en pålitlig kommunikation som tar hand om omsändningar vid paketförluster, annars kommer förbindelsen att termineras. Transport Layer körs därför vanligen över en TCP/IP-uppkoppling, men kan också användas tillsammans med andra pålitliga dataströmmar. En uppkoppling med det här protokollet går till på följande sätt:

1. Klienten kontaktar servern för att etablera en SSH-uppkoppling. De två parterna meddelar varandra om bland annat vilka nyckelutbytes-, krypterings-, autentiserings- och komprimeringsalgoritmer som de stödjer och helst vill använda sig av. De kommer överens om algoritmval för kommunikationen och påbörjar nyckelutbytet.
2. Under nyckelutbytet förhandlas en sessionsnyckel fram och servern autentiseras för klienten med hjälp av certifikat. Den vanligast använda algoritmen för nyckelutbytet är Diffie-Hellman.
3. Sedan används sessionsnyckeln, tillsammans med överenskommen krypteringsalgoritm, för att kryptera resterande kommunikation mellan de två parterna.

Protokollet stödjer ett flertal algoritmer för symmetrisk kryptering såsom 3-DES, AES, BLOWFISH, IDEA med flera, vilket medför att man har stor valfrihet angående krypteringsalgoritm. Också när det gäller algoritmer för att säkerställa integriteten av data så finns det möjlighet att välja olika typer av MAC-algoritmer, eftersom både varianter av HMAC-MD5 och HMAC-SHA-1 stöds. [7]

5.3.2 User Authentication Protocol

Det här protokollets uppgift är att tillhandahålla autentisering av klientanvändaren för servern. Protokollet arbetar över transport layer-protokollet. Protokollet förutsätter därför att det underliggande protokollet tillhandahåller kryptering och integritet samt att servern har blivit autentiserad och en identifierare som är unik för sessionen har erhållits. Det är servern som begär att klientanvändaren ska autentisera sig för den. Servern skickar ett meddelande som innehåller vilka typer av autentiseringsmetoder som den tillåter. Klienten får sedan fritt, inom denna mängd, välja autentiseringsmetod. Man kan dela upp dessa metoder i tre olika grupper: autentisering genom lösenord, autentisering med hjälp av publik nyckelsignering samt autentisering med certifikat. Dessutom har servern ett val att endast autentisera klientdatorn och strunta i användarautentisering, men detta är oftast ej att föredra. Man kan konfigurera servern till att vara mer säker genom att, när man till exempel använder sig av lösenordsbaserad autentisering, endast tillåta ett maximalt antal misslyckade lösenord och därefter stänga ner förbindelsen samt en timeout för hur länge man får försöka. Eftersom lösenorden skickas i klartext är det lätt att förstå att den underliggande arkitekturen måste tillhandahålla kryptering om denna metod används. [7]

5.3.3 Connection Protocol

Efter en lyckad autentisering öppnas flera kanaler genom att multiplexera kommunikationen mellan de två systemen. Varje kanal hanterar kommunikationen för olika tjänster som till exempel terminalsessioner, förmedlande av X11-information och andra som önskar använda SSH-förbindelsen. Nya kanaler kan skapas av både servrar och klienter, där man tilldelar varje kanal olika nummer på båda sidorna. När ena parten försöker öppna en ny kanal, skickas den sidans nummer med anropet. Denna information lagras sedan på den motsatta sidan och används för att styra en särskild tjänsts kommunikation till denna kanal. Detta gör på så sätt att olika typer av sessioner inte påverkar varandra och att kanaler kan stängas utan att avbryta den primära SSH-kommunikationen mellan de två systemen. Användandet av kanaler gör SSH flexibelt när det gäller att hantera olika typer av fjärrkommunikationer såsom modem- och LAN-förbindelser. Klienten och servern förhandlar alltid om hur en kanal ska se ut. [7]

6 Analys och konstruktionslösning

I det här kapitlet beskriver vi varför vi väljer olika lösningar och tekniker och hur vi sedan implementerar dem. För att underlätta läsningen har vi flyttat ut de mer tekniska delarna till bilagor som vi hänvisar till där det är lämpligt.

6.1 Val av säkerhetslösning

Som nämndes i inledningen har vi fått i uppgift att tillhandahålla en säker kommunikation mellan brandväggsadministratören på klientsidan och servern på brandväggen. Det enda krav som ställs på oss är att kommunikationen ska vara säker, det vill säga krypterad och autentiserad. Vid kick off-mötet framkom att vi själva skulle få avgöra vilken säkerhetsmetod som skulle implementeras, men vi fick ett antal förslag såsom SSL, IPSec och SSH som vi skulle titta närmare på.

Efter granskning av de olika metoderna, för att tillhandahålla säker kommunikation, fick vi fram följande för- och nackdelar:

6.1.1 SSL

En klar fördel med SSL är att autentiseringen sker ända upp till och med applikatiosnivå, detta innebär att man kan vara säker på att det dels är rätt part man kommunicerar med samt att det är rätt applikation. Detta gör att risken att utsättas för trojaner minskas.

En nackdel med SSL är att det inte är en Internetstandard. Dock har IETF tagit fram TLS som ska fungera som en Internetstandard för SSL. Det nuvarande TLS-standardförslaget är mycket likt SSLv3, med endast ett par undantag. SSL är inte transparent för applikationen som därför måste vara medveten om att SSL ska användas och måste göra ett explicit SSL-anrop. Detta medför att applikationen blir beroende av SSL. SSL körs över TCP-protokollet och kan därmed inte skydda applikationer som körs över andra protokoll, som till exempel UDP. HTTP körs över SSL och för att kunna överföra data på ett ordnat sätt måste man använda sig av till exempel XML.

6.1.2 IPSec

En fördel med IPSec är att det är helt transparent för användarna vilket medför att dessa inte kan påverka säkerheten på ett negativt sätt. Andra viktiga fördelar med att IPSec är transparent är att det är enkelt att implementera och därmed ökar läsbarheten av källkoden och eventuellt underhåll av källkoden blir lättare att utföra. IPSec är heller inte bundet till någon specifik krypterings- eller autentiseringsalgoritm, så om det visar sig att någon algoritm har brister är det förhållandevis enkelt att byta till någon annan. Ändringarna kommer inte att påverka de berörda applikationerna på något sätt, förändringarna sker på en lägre nivå än applikationsnivån. IPSec har också den positiva sidoeffekten att all trafik mellan de två IPSec-säkrade systemen kommer att skyddas, säkerhetsmedvetna såväl som icke säkerhetsmedvetna applikationer. Dessutom är det en Internetstandard, framtagen av IETF, vilket betyder att tekniken kommer att stödjas allt mer i framtiden, särskilt eftersom det ingår i nästa generations IP-standard, IPv6.

En nackdel med IPSec är att applikationer och användare inte autentiseras, det är endast systemen som är autentiserade. Dataintegritet erhålles endast mellan systemen och inte upp till applikationsnivå.

6.1.3 SSH

SSH är open source, detta är en klar fördel eftersom flera ögon då kontrollerar koden och den utvecklas därmed i snabb takt. SSH är utvecklat i Finland, vilket är bra eftersom Finland inte har några lagar som begränsar exporten av varor med stark kryptering. SSH erbjuder en mycket stark kryptering och en mängd möjligheter när det gäller val av algoritm. SSH komprimerar även datan på ett bra sätt, därmed är det en mindre mängd data som behöver transporteras. SSH erbjuder autentisering av både system och användare men dock ej av applikation.

Eftersom användaren är inblandad i användandet av SSH kan denne påverka säkerheten på ett negativt sätt. SSH är ingen riktig standard framtagen av IETF utan har blivit en *de facto* standard.

6.1.4 Slutsats

Fördelarna med IPSec väger över dess nackdelar. Den största fördelen är transparensen. Den väger i sig tyngre än fördelarna med SSH respektive SSL. Vi väljer därför att använda oss av IPSec som säkerhetslösning för vårt projekt. I samband med detta gör vi antagandet att datorerna i sig är säkrade, det vill säga att informationen inte manipuleras på vägen mellan IP-lagret och applikationslagret.

6.2 Konfiguration av IPSec

Vi konfigurerar IPSec-uppkopplingen på så sätt att ett IP-datagram som skyddas av ESP i sin tur skyddas av AH, vilket medför att innehållet i IP-datagram krypteras samtidigt som hela IP-datagrammet autentiseras. Då vet vi att innehållet förblir hemligt samt att det ej har manipulerats. Dessutom kan vi vara säkra på att paketet kommer från den administratörsdator det utger sig för att komma från. Vi skickar paketen med så kallat transport mode, då kommunikationen skall ske mellan två värddatorer. Tunnel mode används vanligen när trafiken ska säkras mellan nätverk eller mellan en värddator och ett nätverk, alltså normalt sett inte mellan enskilda datorer.

Vi väljer att kryptera paketen med krypteringsalgoritmen 3DES och autentiseringen sker med SHA-1 som autentiseringsalgoritm. För autentisering vid nyckelutbytet använder vi oss av preshared secret. Preshared secret är hemlig autentiseringsdata som man har kommit överens om innan autentiseringen sker.

Windows 2000 och Windows XP stödjer IPSec från början. Där kan man på ett enkelt sätt konfigurera olika IPSec policies med hjälp av ett grafiskt gränssnitt och wizard-funktioner.

Linux däremot har inget inbyggt stöd för IPSec i standarddistributionerna och kräver därmed att en tredjepartsimplementation av IPSec, till exempel FreeS/WAN, kompileras in i kärnan, se bilaga B.

Mer information om hur man konfigurerar IPSec i Windows-miljö återfinnes i bilaga A, konfiguration i Linux-miljö beskrivs i bilaga C.

6.3 Val av programspråk

Vi väljer att implementera vår del av projektet i programspråket Java. Valet motiveras med följande:

- Java är plattformsoberoende. Att klienten är plattformsoberoende är en viktig egenskap.
- Java är objektorienterat. Detta borgar för att koden får en bra struktur och ett den blir effektiv. Koden blir även mer lättläst och det underlättar vid eventuellt underhåll.
- Java är ett enkelt och lättförståeligt språk. Detta är en personlig åsikt. Java liknar C++ syntaktiskt men är mer användarvänligt att koda i. Det finns en massa fördefinierade klasser och metoder att använda sig av, detta gör att det går snabbt att koda i Java. Eftersom vi är bekanta med C++ sen tidigare så tycker vi att Java är ett enkelt och lättförståeligt språk.

6.4 Val av kommunikationsteknik

Vi vill ha kommunikationen så transparent som möjligt och vill inte hålla på med socket-programmering. Därför kom vi snabbt att tänka på CORBA. CORBA är en teknik för att kunna utföra funktionsanrop mot en annan dator, oavsett programspråk, på samma sätt som man utför ett lokalt anrop. Eftersom vi använder oss av Java i både server och klient så är RMI ett självklart val.

RMI är en teknik för att göra metodanrop från ett Java-objekt till ett Java-objekt på en annan dator. Anropen fungerar precis som ett vanligt lokalt metodanrop och gör därmed kommunikationen transparent. Det är enklare att använda RMI än CORBA eftersom man i CORBA måste definiera fjärrobjektet i IDL för att kunna kompilera fram stubbar och skelett för det programspråk man använder. Detta behövs inte i RMI eftersom kommunikationen endast sker mellan Java-objekt.

6.5 Autentisering

Eftersom IPSec som bekant endast erbjuder autentisering av system och inte av användare så måste vi själva tillhandahålla en metod för autentisering av dessa. Vi måste även kunna säkerställa att det endast är den som har loggat in som använder sig av serverns tjänster.

Servern måste kunna hantera att en uppkopplad klient försvinner på ett okontrollerat sätt, det vill säga utan att logga ut sig från servern.

För att en användare ska kunna logga in sig på servern tillhandahåller vi en databas med användare och tillhörande lösenord. Databasen är uppbyggd med poster som ser ut på följande sätt:

[användar-ID]:[lösenord hashat med SHA-1]:[fullständigt användarnamn]

Användaren anger användar-ID, som ska vara unikt i databasen, och lösenord vid inloggning. Dessa jämförs med databasen. Vid lyckad inloggning genereras ett stort slumptal som används som sessions-ID. Detta skall anges vid varje metodanrop mot servern. Detta är till för att skydda mot att ej inloggade användare kan utnyttja metoderna. Detta sessions-ID byts ut efter en viss tidsrymd om kommunikationen varar en längre tid. Detta för att minska risken att någon lyckas lista ut sessions-ID:t som används för ju längre tid samma sessions-ID används desto större är risken. Vid felaktig inloggning eller vid användande av felaktigt sessions-ID aktiveras en fördröjning i syfte att skydda mot så kallade ordboksattacker. Vi tillhandahåller även en loggningsfunktion på serversidan. Den loggar två händelser, lyckat inloggning samt inloggningsförsök med korrekt användarnamn och felaktigt lösenord. Varje lyckad inloggning loggas med en rad i loggfilen som ser ut som följer:

[datum och klockslag]:[användarnamn]:”successful login”

Varje inloggningsförsök där rätt användarnamn och fel lösenord har angivits ser ut som följer:

[datum och klockslag]:[användarnamn]:”unsuccessful login”

Vi väljer att endast en användare åt gången ska kunna vara inloggad på servern, detta för att inkonsistens inte ska uppstå i brandväggsreglerna. Detta löser vi med att en boolesk variabel anger om det finns någon inloggad på servern eller inte. Om någon försöker logga in på servern när någon redan är inloggad returneras ett meddelande som säger att servern är upptagen. För att det här ska fungera korrekt krävs det att en användare loggar ut från servern efter avslutat arbete.

Det vi beskriver i stycket ovan är ett idealfall. Vi inser att en användare inte alltid kan logga ut på ett korrekt sätt eftersom kommunikationen kan avbrytas på ett okontrollerat sätt. Detta skulle kunna ske på många sätt, till exempel genom att klientdatorn stängs ner vid strömavbrott, klientdatorn kraschar, nätet går ner etcetera. Därför, för att servern skall kunna fortsätta användas, måste servern kunna återhämta sig efter en sådan händelse. Detta löser vi genom att klienten skickar ut meddelanden till servern med jämna mellanrum för att berätta att den fortfarande lever. Om servern inte erhåller något sådant meddelande inom en viss tidsperiod kommer den att antaga att klienten har försvunnit och därför själv logga ut den inloggade användaren. Därmed gör servern sig tillgänglig för nya inloggningar.

6.6 Arbetsuppgifter för vår modul

Alltefter projektets gång har vi också blivit tilldelade fler arbetsuppgifter. Det vi nu ska ta hand om är:

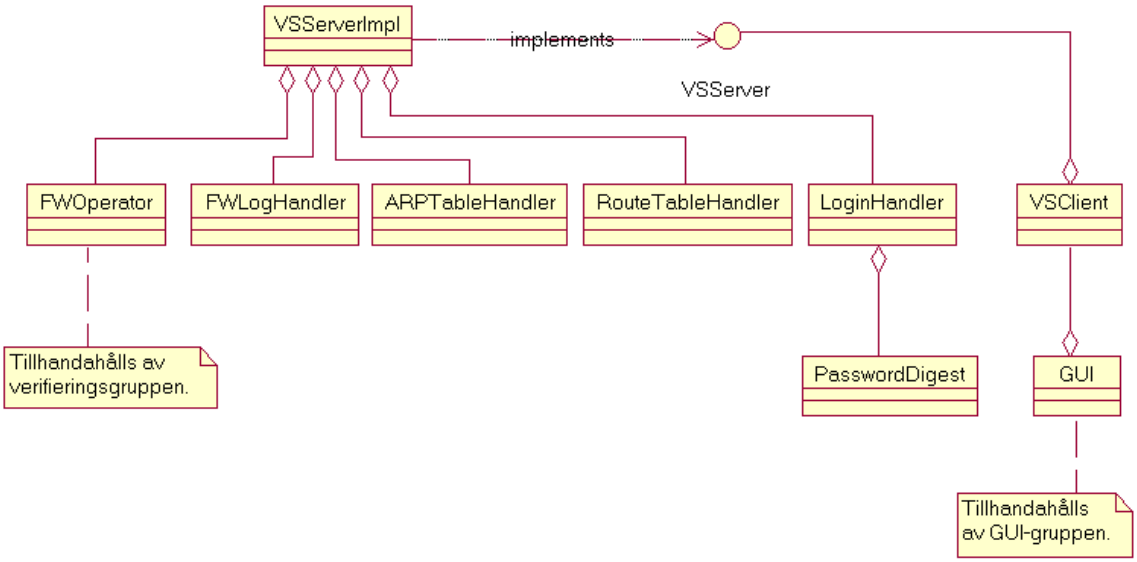
1. Hämta de nuvarande brandväggsreglerna från brandväggen.
2. Skicka över nya regler som ska installeras i brandväggen.
3. Hämta ARP-tabellen från brandväggsdatorn.
4. Hämta route-tabellen från brandväggsdatorn.
5. Tillhandahålla operationer på ARP- respektive route-tabellerna. Exempel på operationer är lägg till post eller ta bort post.
6. Vi ska kunna skicka över de meddelanden som Iptables lägger i systemloggen, så att en administratör ska kunna se vad som händer i brandväggen.

6.7 Hur vi löser arbetsuppgifterna

1. Vi använder oss av den modul som verifieringsgruppen har utvecklat för hämtning av regler från brandväggen.
2. Vi använder oss även här av den modul som verifieringsgruppen har utvecklat. Den används för att installera de nya reglerna i brandväggen.
3. Vi använder oss av Linux' systemkommando *arp* för att hämta ARP-tabellen. Det är kommandot för att skriva ut ARP-tabellen på skärmen som används.
4. Här används Linux' systemkommando *route* för att hämta route-tabellen. Det är kommandot för att skriva ut route-tabellen på skärmen som används.

5. Linux' systemkommandon *route* och *arp* som används för att operera på tabellerna. Argument till kommandona läggs till beroende på vilken operation som skall utföras.
6. Iptables skriver sina loggar till */var/logs/messages* med taggen ”iptables:”. För att hämta meddelandena som har genererats av brandväggen använder vi oss av strängmatchningsverktyget *grep* för att söka efter och hämta ut alla rader som innehåller strängen ”iptables:”. För att undvika att skicka onödig information anger klientprogrammet hur många av de senaste loggraderna det vill ha. Den här lösningen fungerar i alla fall utom ett, nämligen då servermaskinen har namnet ”iptables” eftersom varje rad i loggfilen även innehåller namnet på datorn följt av ett kolon. Då kommer varje rad i loggen att returneras, även de rader som inte har med brandväggen att göra.

6.8 Modellering



Figur 6.1: Översiktligt klassdiagram.

Figur 6.1 visar klassdiagrammet översiktligt. En kort beskrivning av de ingående klasserna, som vi har gjort, följer.

VSClient: Klientsidan i kommunikationen, sköter uppkopplingen mot servern och erbjuder metoder för att anropa metoder hos servern.
VSServer: Ett *interface* som deklarerar vilka metoder serverobjektet måste implementera och som klienten kan använda sig av.

VSServerImpl: Detta är implementationen av servern. Innehåller implementationer av alla metoder som klienten ska kunna anropa samt resterande delar för att servern ska fungera korrekt.
LoginHandler: Sköter inloggningen av användare. Ett objekt av LoginHandler finns alltid på servern och innehåller information om inloggningsstatus.
PasswordDigest: Har hand om hashning av lösenord.
FWLogHandler: Har hand om hämtandet av brandväggsloggen.
ARPTableHandler: Tillhandahåller metoder för operationer på ARP-tabellen.
RouteTableHandler: Tillhandahåller metoder för operationer på route-tabellen.

VSLoginException: Kastas när inloggningen misslyckades på grund av felaktigt användarnamn och/eller lösenord.
VSOccupiedException: Kastas vis inloggningsförsök när det redan är någon inloggad på servern.
VSIdNotValidException: Kastas när sessions-ID:t inte stämmer överens med ID:t för den aktuella sessionen.

API:n till vårt projekt återfinnes i bilaga E. Ett mer utförligt klassdiagram, med alla fält och metoder i varje klass, återfinnes i bilaga D.1, användarfall i form av sekvensdiagram återfinnes i bilaga D.2 och ett par aktivitetsdiagram återfinnes i bilaga D.3.

6.9 Att få igång systemet

Filerna ska placeras enligt Tabell 6.1.

Server	VSServer.java, VSServerImpl.java, LoginHandler.java, PasswordDigest.java, FWLoghandler.java, ARPTableHandler.java, RouteTableHandler.java, VSLoginException.java, VSIdNotValidException.java, VSOccupiedException.java, (FWOperator.java), VSServerPolicy, ”.usrdb”.
Klient	VSClient.java, VSServer.java, VSLoginException.java, VSIdNotValidException.java, VSOccupiedException.java, VSClientPolicy

Tabell 6.1: Filplacering.

Steg för att få igång servern:

1. Kompilera serversidan
   javac *.java
2. Tillverka stubbe och skellett
   rmic VSServerImpl
3. Starta upp servern
   java –Djava.security.policy=VSServerPolicy VSServerImpl <IP-adress>

Steg för att få igång klienten:

1. Flytta stubben som tillverkades på serversidan till klienten och lägg den på samma ställe som de andra klientfilerna.
2. Kompilera klientsidan
   javac *.java
3. Sen är det bara att skapa VSClient-objekt och använda sig av där man vill implementera klienten, till exempel från ett grafiskt användargränssnitt. Viktigt att tänka på är att man på klientsidan, precis som på serversidan, måste ange vilken policy-fil som *security managern* skall använda sig av:
   java –Djava.security.policy=VSClientPolicy <class-fil>

Policyfilen berättar för *security managern* vilka rättigheter den virtuella maskinen ska ha på respektive sida.

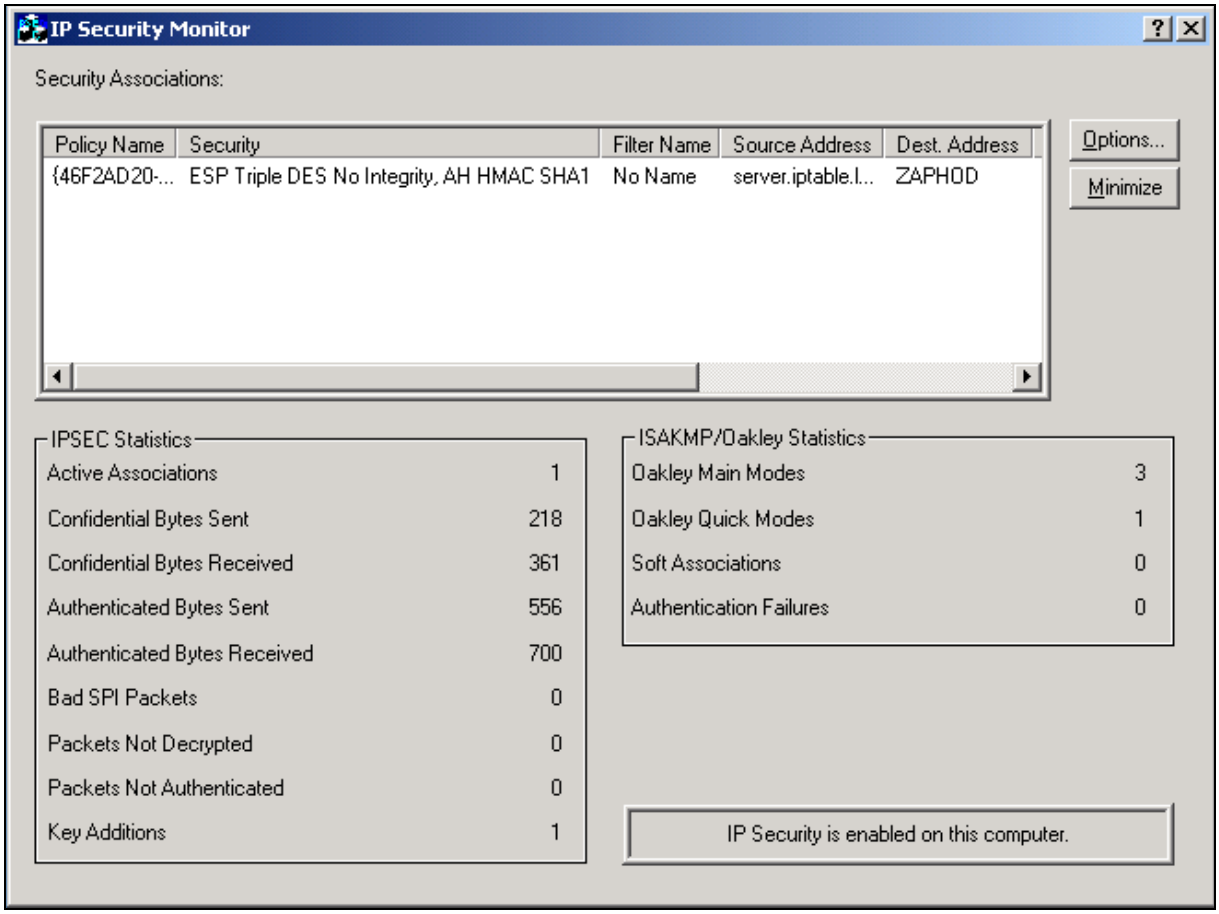
För att man ska kunna få en uppkoppling och kunna kommunicera överhuvudtaget så måste följande gälla för brandväggen:

- UDP-paket på port 500 måste accepteras. Detta för att IKE ska kunna fungera, det vill säga att förhandling av SA:n ska fungera.
- IP-paket med protokollnumren 50 och 51 måste accepteras, 50 är ESP-paket och 51 är AH-paket.
- TCP-paket på port 1099 måste accepteras. Detta för att RMI-uppkopplingen ska fungera.

7 Test av implementationen

I det här kapitlet kommer vi att ta upp olika testfall som vi har genomfört. Att genomföra testfall är ett måste för att kunna verifiera att programmet fungerar som det är tänkt.

Vi har använt oss av det i Windows 2000 medföljande programmet IP Security Monitor (ipsecmon.exe) för att kontrollera att IPSec-uppkopplingar etableras mellan datorerna på ett korrekt sätt. I Figur 7.1 kan man se att en IPSec-förbindelse har upprättats mellan två datorer. Man kan se att IP-paketen skyddas med ESP med 3DES-kryptering och utan ESP's autentisering, samt att IP-paketen även skyddas med AH där SHA1 används som autentiseringsalgoritm. Här kan man även se hur mycket data som har skickats över IPSec-kopplingen plus annan användbar information.



Figur 7.1: IP Security Monitor.

För att verifiera att IP-datagrammen byggs upp på ett korrekt sätt har vi använt oss av programmet Sniffer Pro. Med detta program kan man ”spela in” trafiken som förekommer på ett nätverk, trafiken kan sedan analyseras genom att gå in i varje paket som har skickats och se precis hur det var uppbyggt. Det är här vi har sett att paketen var uppbyggda på det sätt vi ville, det vill säga IP-paketen är skyddade av ESP som i sin tur är skyddade av AH.

Uppkoppling mot servern med RMI fungerar både när servern är i en Linux-miljö och i en Windows-miljö. Dock är det endast meningsfullt att ha servern i en Linux-miljö eftersom vi använder oss av Linux’ systemkommandon för att utföra vissa uppgifter. Klienten har också testats i både Windows och Linux. Den fungerar också tillfredsställande i båda operativsystemen. Detta däremot är ett viktigt resultat, att klienten är plattformsoberoende är bra och en av orsakerna till att vi valde programspråket Java att utveckla vårt system i.

Servern känner av och reagerar på ett korrekt sätt när en uppkopplad klient avbryter kommunikationen på ett okontrollerat sätt. Detta innebär att ”keep alive”-meddelandena från klienten fungerar som de ska. Efter en viss tidsperiod, inom vilken servern inte har fått några ”keep alive”-meddelanden från klienten, loggar servern ut den inloggade användaren.

Inloggning med felaktigt användarnamn och/eller lösenord har testats, det fungerade som vi hade tänkt. Då vet vi att kontrollen mot användardatabasen fungerar korrekt. Detta kan vi våga påstå eftersom det fungerar att logga in en användare med korrekt användarnamn och korrekt lösenord.

Vi har testat att en användare som inte är inloggad inte kan utnyttja systemet. Samt att en användare som inte är inloggad inte kan utnyttja systemet när en annan användare är inloggad, det vill säga endast en användare åt gången kan använda systemet. Av detta kan vi dra slutsatsen att kontrollen av sessions-ID fungerar som den ska, eftersom det är detta sessions-ID som är felaktigt i båda fallen. Naturligtvis kan en användare med för tillfället korrekt sessions-ID använda sig av systemet.

Ett korrekt felmeddelande genereras när felaktiga argument skickas till de operationerna på ARP- och route-tabellerna vi tillhandahåller. Felmeddelandet som genereras är helt enkelt det felmeddelande som skrivs ut på skärmen när man gör felaktiga anrop till kommandona *arp* och *route*.

8 Resultat

Resultatet av arbetet har blivit en kommunikationsmodul som är skriven i Java och använder sig av RMI som kommunikationsteknik. Själva kommunikationen mellan server och klient har säkrats med hjälp av IPSec. Autentiseringen och loggning av användare i vår kommunikationsmodul hjälper också till att höja säkerheten ytterligare. Med vår modul kan man utföra enklare operationer såsom hämtande och modifierande av arp- och route-tabeller på en Linux-dator. Man kan dessutom hämta händelser som har loggats av Iptables i systemloggen. Med vår modul kan man också hämta reglerna i Linux' brandvägg (Netfilter) samt lägga in nya regler i Netfilter vilket var den ursprungliga huvuduppgiften. Vi har även lyckats lösa problemet med klienter som kopplar ner utan att logga ut från servern. Detta genom att servern själv loggar ut en användare om denne inte kontaktar servern inom en viss tidsperiod. Vår modul har också testats tillsammans med de övriga modulerna i projektet med ett tillfredställande resultat. Detta innebär att Veriscan har erhållit det de efterfrågade, nämligen ett program för att kunna fjärradministrera Iptables på ett enkelt och säkert sätt. Vi anser därför att vår del samt hela projektet varit lyckat och bra utfört. Veriscan har fått en bra grund till ett program som man förhoppningsvis kan utveckla vidare.

9 Slutsatser

För att kunna få stöd för IPSec i Linux krävs en hel del kunskaper om operativsystemet. Detta fick vi bittert erfara genom att många kompileringar gick åt skogen. Manualen för FreeS/WAN kan vara lite rörig och svår att förstå för en användare som inte känner till Linux så bra. Man kan ju alltid hoppas på att USAs lättande på restriktionerna gällande export av kryptering kommer att medföra att även Linux distributioner kommer att inkludera stöd för IPSec *by default*. Också på grund av att vi först använde oss av en dator med dålig prestanda så tog kompileringen av kärnan väldigt lång tid och det resulterade i att den övriga utvecklingen blev lidande. Det är med andra ord en klar fördel att använda sig av en snabb dator när man experimenterar med Linux.

Eftersom vi gjorde ett antagande där vi säger att transporten mellan IP-lagret upp till applikationslagret är helt säker så är detta en klar säkerhetsrisk i kommunikationen. Det bästa vore att även erbjuda integritet, autentisering och konfidentialitet ända fram till applikationen. Men eftersom tiden var knapp fick vi här göra en avvägning på detta för att hinna klart med systemet.

Vi använder oss också av preshared secret vid utbytet av nycklar vilket ej är att föredra eftersom de ofta lagras i klarttext i både Linux och Windows. Filerna är dock skyddade, normalt kommer endast root-användare eller administratörer åt filerna. Att föredra hade istället varit att använda sig av certifikat. Av samma anledning som ovan fick vi här göra en avvägning av vad vi skulle hinna med att utföra. Eftersom man ytterligare måste konfigurera IPSec på Linux för att få stöd för detta samt att man måste generera certifikat som sedan ska distribueras.

Referenser

[1] Charles P. Pfleeger. *Security in Computing.* Prentice Hall, Second edition, 1997.

[2] ComputerUser.com
http://www.computeruser.com/resources/dictionary/
(2002-05-19)

[3] D. Brent Chapman, Elizabeth D. Zwicky. *Building Internet Firewalls.* O'Reilly & Associates, 1995.

[4] Dieter Gollman. *Computer Security.* John Wiley & Sons, 2001.

[5] HDC AB
http://www.quicknet.se/hdc/
(2002-05-27)

[6] IETF – IPSec
http://www.ietf.org/html.charters/ipsec-charter.html
(2002-05-21)

[7] IETF – SSH
http://www.ietf.org/html.charters/secsh-charter.html
(2002-05-21)

[8] IETF – SSL
http://www.ietf.org/html.charters/tls-charter.html
(2002-05-21)

[9] IETF - TLS
http://www.ietf.org/html.charters/tls-charter.html
(2002-05-21)

[10] James F. Kurose, Keith W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley, 2001.

[11] Magnus Ewert. *Datakommunikation: Nu och i framtiden*. Studentlitteratur, 2001.

[12] Naganand Doraswamy, Dan Harkins. *IPSec: The New Security Standard for the Internet, Intranets and Virtual Private Networks*. Prentice Hall, 1999.

[13] NIST – SKIPJACK
http://csrc.nist.gov/encryption/skipjack/skipjack-kea.htm
(2002-05-21)

[14] Rolf Oppliger. *Security Technologies: for the World Wide Web.* Artech House, 1999.

[15] Skolverket
http://www.skolverket.se/skolnet/smultron/ordlista.htm
(2002-05-10)

[16] STF
http://www.stf.se/stf/it/ordlista/
(2002-05-27)

[17] Stig Jensen, Arne Gjelstrup, Valentino Berti. *Datakommunikation.* Liber, 2000.

[18] Svenska datatermgruppen
http://www.nada.kth.se/dataterm/rek.html
(2002-05-10)

[19] The UNIX Acronym List
http://www.roesler-ac.de/wolfram/acro/all.htm
(2002-05-10)

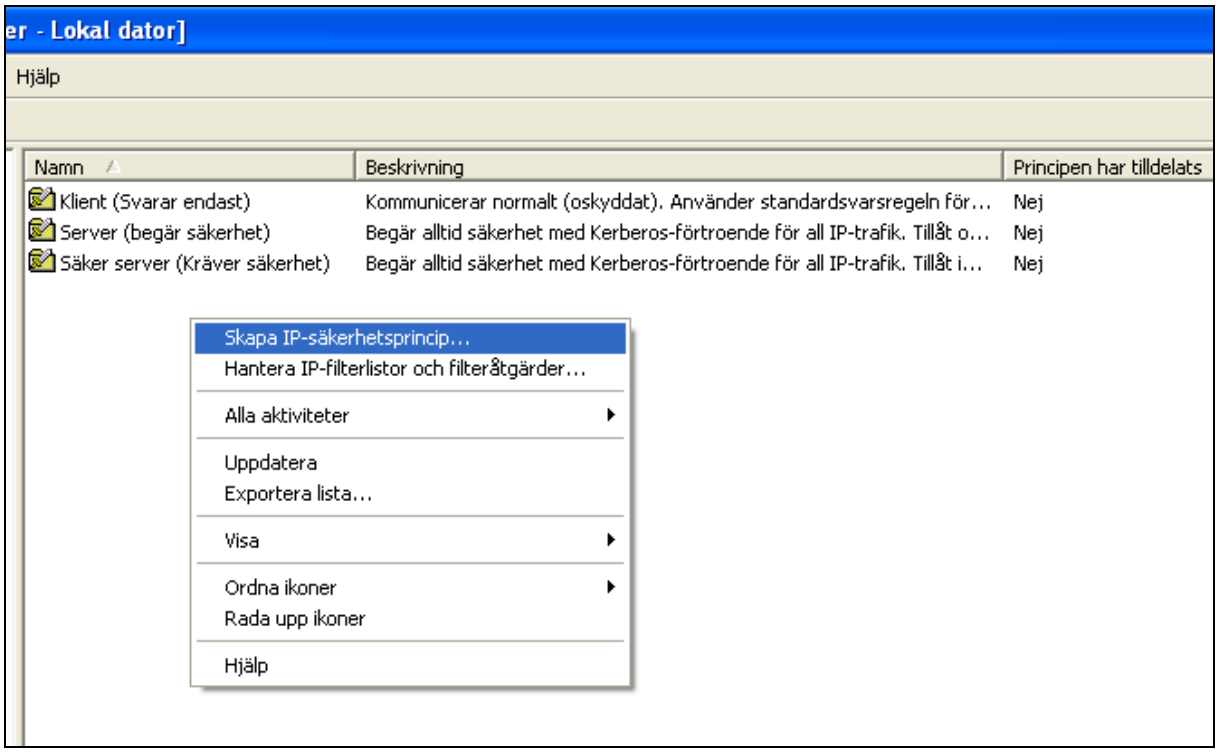
[20] Tropical Software - BLOWFISH
http://www.tropsoft.com/strongenc/blowfish.htm
(2002-05-21)

[21] William Stallings. *Network Security Essentials: Applications and Standards*. Prentice Hall, 2000.

[22] ZDNet: Denial of Service Attacks
http://www.zdnet.com/devhead/stories/articles/0,4413,2172746,00.html
(2002-02-22)

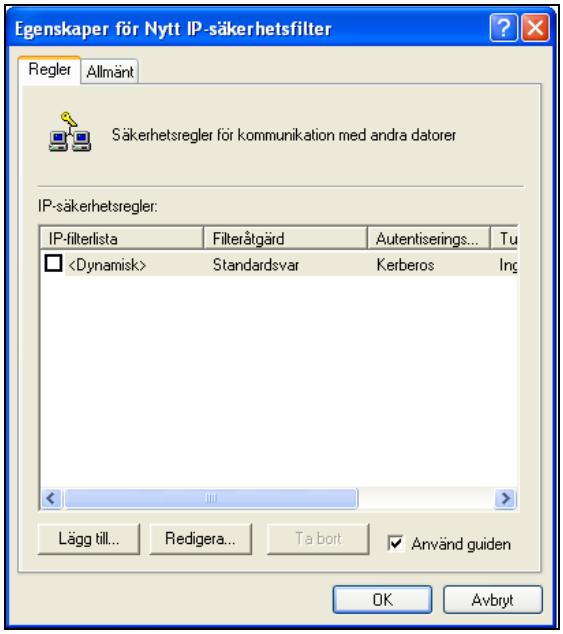
A Konfiguration av IPSec i Windows

Här kommer vi att beskriva hur vi har gått till väga för att konfigurera IPSec i en Windows-miljö. Det finns stöd för IPSec i både Windows 2000 och Windows XP, skärmdumparna som följer är från en XP-miljö men det skiljer inte mycket mellan XP och 2000. Vi har även valt att gå igenom inställningarna med wizard-funktionerna urkopplade, där det är möjligt, för att på så sätt vara säker på att det ska gå att följa instruktionerna på ett bra sätt, vilken variant man än väljer. Under konfigurationens gång får man själv döpa de regler, listor med mera till det man tycker passar och även ge beskrivningar till de samma, detta antas man göra när möjlighet finns. Man når konfigureringen genom att gå till administrationsverktyg->lokal säkerhetspolicy. Där går man till säkerhetsprinciper (eller motsvarande), högerklickar och väljer att skapa en ny säkerhetsprincip. Se Figur A.1.



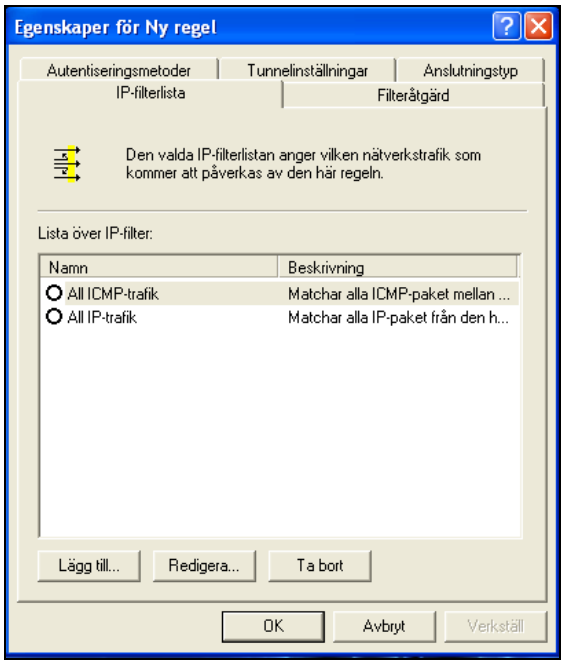
Figur A.1: Skapa IP-säkerhetsprincip.

En wizard startar, följ instruktionerna i den. Det spelar ingen roll vad man väljer eftersom vi inte kommer att använda oss av den säkerhetsregeln i alla fall. När wizarden är färdig klickar man bort den nyligen gjorda säkerhetsregeln, klickar bort ”Använd guiden” eller motsvarande och väljer att lägga till en ny säkerhetsregel, se Figur A.2.



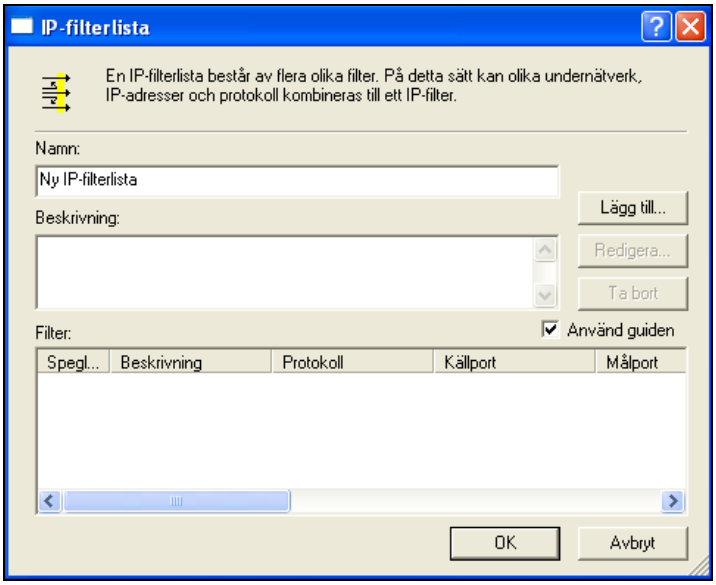
Figur A.2: Egenskaper för IP-säkerhetsprincip.

Nu ska man ställa in egenskaperna för den nya regeln. Under fliken ”IP-filterlista” väljer man att lägga till en ny IP-filterlista, se Figur A.3.



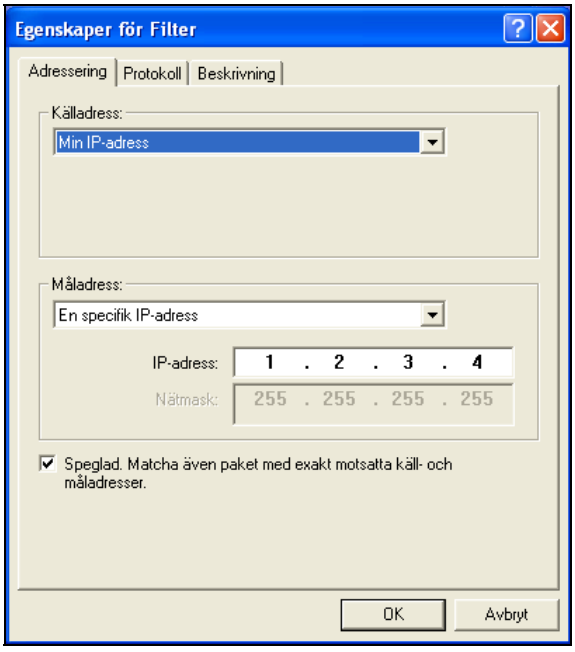
Figur A.3: Egenskaper för IP-säkerhetsregel.

Man klickar ur ”Använd guide” och väljer att lägga till ett nytt IP-filter, se Figur A.4.



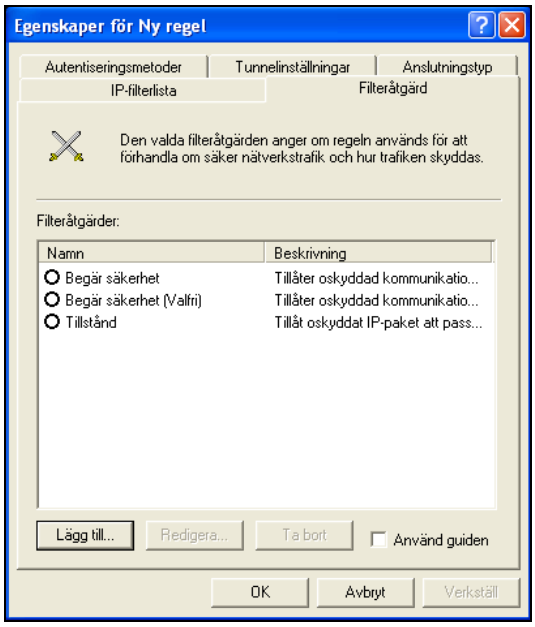
Figur A.4: Ny IP-filterlista.

Under fliken ”Adressering”, se Figur A.5, väljer man ”Min IP-adress” som ”Källadress”. Som ”Måladress” väljer man ”En specifik IP-adress”, sen fyller man i den IP-adress man syftar på. Viktigt här är att se till att den är markerad som speglad. Då säkerställer man att det görs associationer åt båda håll i kommunikationen. Under fliken ”Protokoll” låter man kommunikationen gälla mellan alla protokoll. Tryck ”OK”. Lägg till flera IP-filter till IP-filterlistan efter behov. Tryck ”OK” igen för att godkänna IP-filterlistan. Se till att den nya IP-filterlistan är markerad.



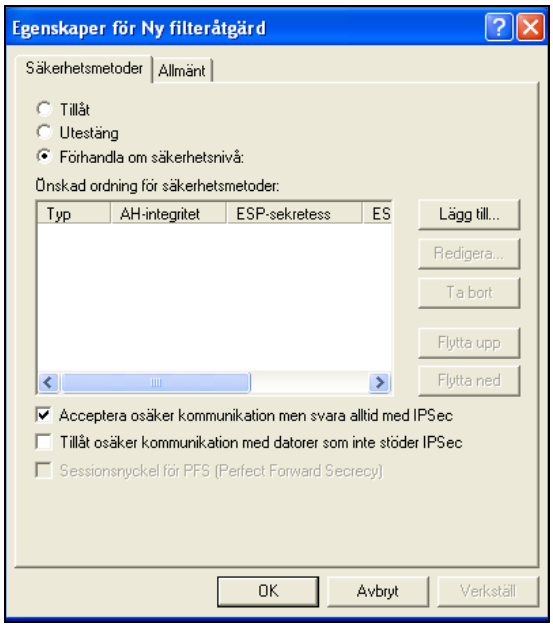
Figur A.5: Egenskaper för IP-filter.

Gå till fliken ”Filteråtgärd” och klicka bort ”Använd guiden”. Välj att lägga till en ny filteråtgärd. Se Figur A.6.



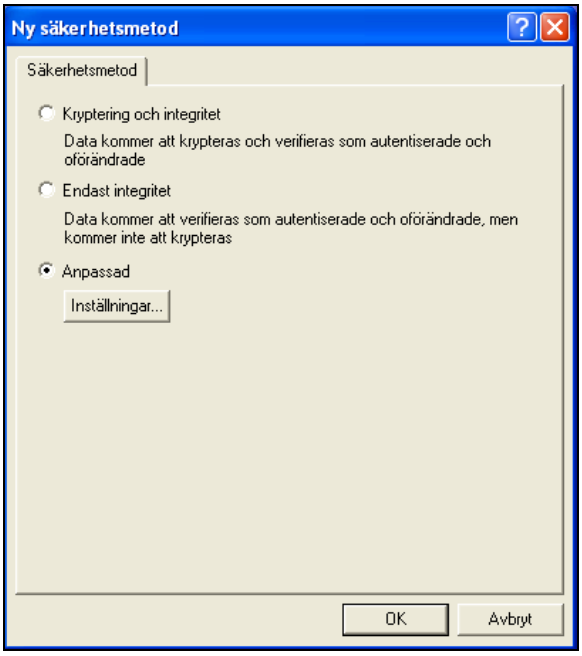
Figur A.6: Filteråtgärder.

Markera ”Förhandla om säkerhetsnivå”, klicka bort ”Acceptera osäker komm…”, markera ”Tillåt osäker komm…”. Läs varningsmeddelandet och klicka på ”Ja”. Anledningen till att vi måste tillåta osäker kommunikation är att datorn måste kunna kommunicera utan IPSec. Kunde den inte det så skulle kommunikation med datorer som inte är med i IP-filterlistan vara omöjlig. Kryssa i ”Sessionsnyckel för PFS…”. Välj sedan att lägga till en säkerhetsmetod. Se Figur A.7.



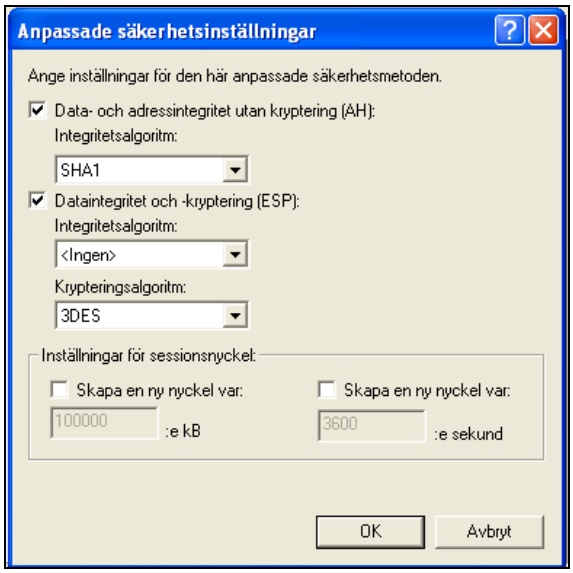
Figur A.7: Ny filteråtgärd.

Markera ”Anpassad” och klicka på ”Inställningar…”, se Figur A.8.



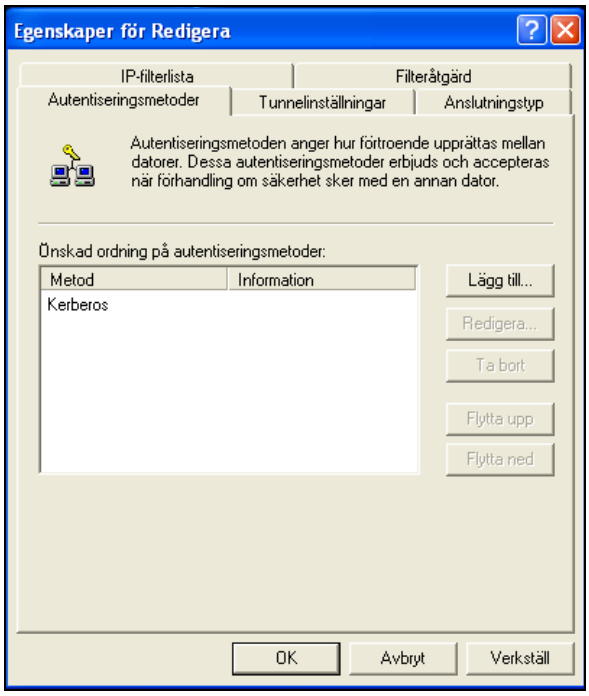
Figur A.8: Ny säkerhetsmetod.

Här ställer man in hur paketen ska vara skyddade. Man markerar att man vill använda sig av AH och som integritetsalgoritm använder man sig av SHA1. Man markerar även att man vill använda sig av ESP. Där ska man inte ha någon integritet, detta sköter ju AH om, därför väljer man <ingen> som integritetsalgoritm. Som krypteringsalgoritm väljer man 3DES. De andra inställningarna låter man vara. Klicka på ”OK”. Se Figur A.9. Klicka på ”OK” igen. Markera att den nya filteråtgärden ska användas. Tryck på fliken ”Autentiseringsmetoder”.



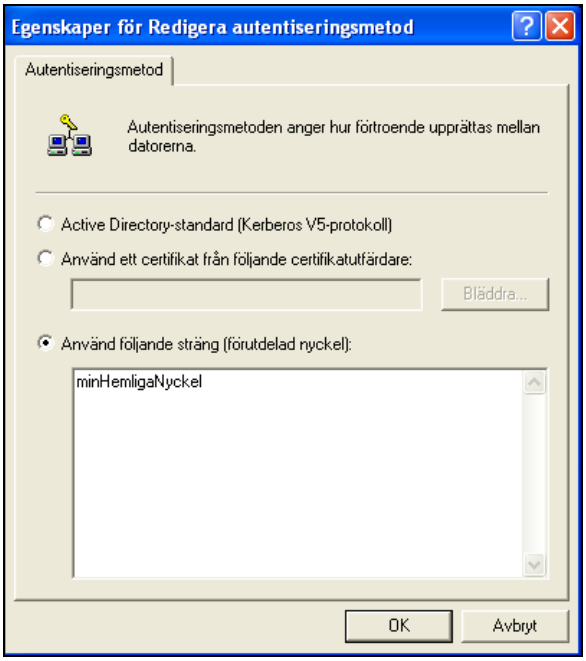
Figur A.9: Anpassade säkerhetsinställningar.

Välj att redigera den befintliga autentiseringsmetoden eller välj att lägga till en ny om det inte finns någon, se Figur A.10.



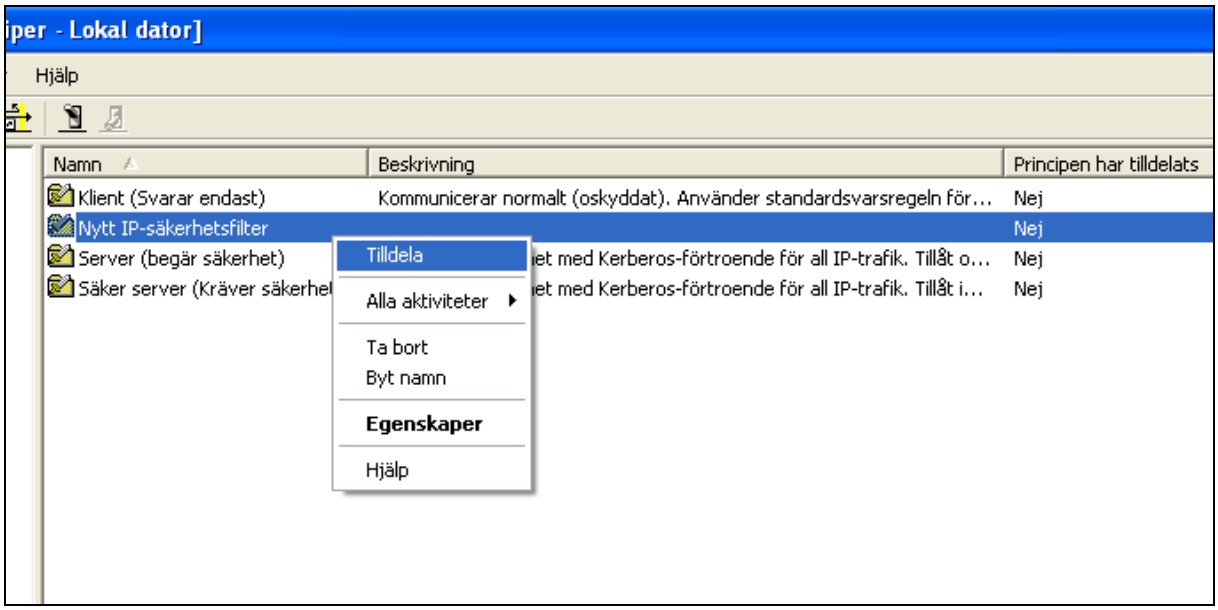
Figur A.10: Autentiseringsmetoder.

Markera ”Använd följande sträng (förutdelad nyckel)” och klistra in alternativt skriv in den preshared secret som skall användas. Tryck ”OK”. Se Figur A.11.



Figur A.11: Redigera autentiseringsmetod.

Säkerställ att ingen tunnel är definierad under fliken ”Tunnelinställningar”. Under fliken ”Anslutningstyp” anger man att alla nätverksanslutningar skall gälla. Tryck ”OK”. Tryck på ”Stäng”. Högerklicka på den nya säkerhetsprincipen och tryck på ”Tilldela”, se Figur A.12. Nu är säkerhetsprincipen aktiverad och IPSec-kommunikation skall fungera med de datorer man har angivit i IP-filterlistan.



Figur A.12: Tilldelning av IP-säkerhetsprincip.

B Få stöd för IPSec i Linux med FreeS/WAN

Först och främst ska man logga in i Linux-systemet som root. Den här dokumentationen kommer att beskriva hur man får IPSec stöd i Red Hat-distributioner, men bör även fungera för andra Linux-distributioner. Installationen som beskrivs är när man installerar direkt från FreeS/WAN-källfiler.

För att få stöd för IPSec i Linux krävs att man installerar tredjepartsprogrammet FreeS/WAN. FreeS/WAN finns att ladda ner på följande URL: ftp.xs4all.nl.

Enklaste sättet att ladda ner FreeS/WAN-filerna är att utföra följande kommandon:
cd /usr/src/
ncftpget ftp://ftp.xs4all.nl/pub/crypto/freeswan/freeswan-*

Packa upp *.tar*-filen med följande kommando:
tar -xzf freeswan*.gz

Detta kommer att generera ett directory som heter:
/usr/src/freeswan<version>
Detta innehåller alla FreeS/WAN-filer.

Bästa sättet är att först konfigurera, kompilera, installera och testa en linux-kärna utan FreeS/WAN för att på så sätt underlätta eventuell felsökning.

Följande verktyg och bibliotek måste inkluderas i konfigurationen för att få FreeS/WAN att fungera korrekt:

- Verktyg
  - En GNU C-kompilator (till exempel *gcc* eller *egcs*).
  - assembler och linker för den specifika arkitekturen (bin86-paketet för PC)
  - Utvecklingsverktygen *patch(1)* och *make(1)*.
- Bibliotek, både headers och objektmoduler
  - Standardkompilatorbibliotek som *glibc*.

- *GMP-biblioteket*, används av Pluto[1] för publik nyckelberäkningar.
- *ncurses*-biblioteket för att kunna använda kommandot *menuconfig*.

Vi konfigurerade och installerade kärnan med hjälp av att ge följande kommandon i */usr/src/linux/*:

(Eftersom directoryt */usr/src/linux/* inte existerar på grund av att directoryt innehåller kärnans versionsnummer i sitt namn, till exempel *linux-2.4.7-10*, så måste den skapas. Detta görs genom att göra en så kallad *soft link* som heter *linux* och pekar på *linux-2.4.7-10*. Detta görs på följande sätt:
ln –s /usr/src/linux-2.4.7-10 /usr/src/linux

make menuconfig
   För att bestämma vad som skall ingå i kärnan och vad som skall kunna köras som moduler i kärnan.
make dep
   För att hitta beroenden mellan olika filer.
make bzImage
   Bygger en laddbar kärnbild (kernel image) komprimerad med *bzip(1)*.
make install
   Installerar kärnan.
make modules
   Bygger modulerna som skall kunna köras i kärnan.
make modules_install
   Installerar modulerna.
lilo
   Uppdaterar lilo (boot loader).

Före *lilo*-kommandot måste man gå in i */etc/lilo.conf* och lägga till den nya kärnan så att den går att boota.

[1] Pluto är en implementation av IKE i Linux. Den körs som en så kallad daemon på den IPSec-säkrade Linux-maskinen.

Nu kan man testa att boota om på den nya kärnan för att säkerställa att den fungerar som den ska. Om allt verkar vara i sin ordning kan man nu gå vidare med att kompilera in FreeS/WAN i kärnan.

cd /usr/src/freeswan<version>
make menugo

Viktigt att tänka på är att man alltid ska spara kärnkonfigurationen även om man inte har gjort några förändringar.

Om den sista utskriften på skärmen är *make* som talar om att den anropar FreeS/WAN:s *errcheck*-skript så har allt gått väl. Om inte detta meddelande är det sista har något fel inträffat. Då kan man konsultera lämplig *out.**-fil för detaljer.

Sedan installerar man den nya kärnan genom att man, på samma ställe som förut, kör följande kommando:
make kinstall

Kontrollera */etc/lilo.conf* för att se att allt är i sin ordning. Kör sedan *lilo*-kommandot för att systemet skall bli varse ändringarna.

För att testa att installationen gick bra kan man göra följande:
Starta om och boota upp den nya kärnan. Se så att IPSec startar och att det i övrigt inte blir några fel. Kör sedan följande kommandon:

- *ipsec --version*, för att se om man kan använda ipsec-kommandon.
- *ipsec whack --status*, för att kontrollera statusen på Pluto.

Innan IPSec kan tas i bruk måste IP Forwarding slås på, om IP-paketen ska routas vidare det vill säga om maskinen skall användas som en gateway. Detta görs genom att gå in i standardbootscriptet */etc/sysconfig/network* och där ange följande *net.ipv4.ip_forward=1*. För att kunna göra detta måsta IP forwarding angetts vid konfigurationen av kärnan, om detta inte stöds kommer ovanstående inte att fungera.

Användbara verktyg för felsökning av FreeS/WAN är *ipsec look* som ger en kortfattad rapport om statusen samt *ipsec barf* vilken visar all IPSec-genererad information.

Viktigt att observera är också att om brandväggar ingår i kommunikationen så måste dessa ställas in så att de släpper igenom UDP port 500 eftersom denna används av IKE vid nyckelförhandlingen. Dessutom behöver man släppa igenom datagram med nummer 50 respektive 51 för att ESP och AH ska kunna användas.

Egna observationer som vi har gjort är att IPSec tycks fungera bättre om man kompilerar in stöd för det direkt in i kärnan och alltså inte som moduler. Om detta är en tillfällighet beroende av flera faktorer eller om det verkligen ligger till på detta sätt vet vi inte.

Om problem uppstår eller mer ingående information eftertraktas hänvisar vi till FreeS/WAN's hemsida och dess dokumentation:
http://www.freeswan.org

C Konfigurering av IPSec i Linux

Konfigureringen av IPSec i Linux, när man har tredjepartsprogrammet FreeS/WAN installerat, sköts med hjälp av två filer */etc/ipsec.secrets* och */etc/ipsec.conf*. Vi kommer att kortfattat beskriva innehållet i dessa två filer den efterföljande texten och ge ett utdrag av de båda.

C.1 ipsec.secret

I denna fil lagras de data som används vid autentiseringen av systemen som IKE använder sig utav vid nyckelutbytet. Denna fil brukar därför endast hållas läsbar för root, så att inte hemlig autentiserings-information ska komma på villovägar. Autentiseringen kan antingen ske med hjälp av så kallade Preshared secrets eller också på det mer eleganta och säkrare sättet med RSA. Ett exempel på innehållet i *ipsec.secrets* filen ges nedan. Den första okommenterade raden (# används som kommentartecken) specificerar att de två interfacen 192.168.213.21 och 192.168.213.10 ska autentisera sig med en Preshared secret som består av strängen ”minHemligaNyckel”. Den så kallade Preshared secret key’n kan genereras av verktyget *ipsec ranbits* för att på så sätt erhålla en lång slumpmässig sträng för att försvåra att nyckeln knäcks. Den andra delen i filen, som avgränsas med ett **:**, innehåller RSA specifik information som kan användas för autentisering om man så önskar denna metod för detta. Ett RSA nyckelpar kan skapas med verktyget *ipsec rsasigkey* på ett enkelt sätt. Att observera när man arbetar med filen är att om man väljer att bryta mitt i en rad för att fortsätta på en ny rad så måste denna rad indenteras för att det ska fungera. Konsultera *ipsec.secrets* mansidorna på FreeS/WAN hemsida för mer ingående information.

Exempel på en ipsec.secret-fil:

```
   # This file holds shared secrets or RSA private keys for inter-Pluto
   # authentication.  See ipsec_pluto(8) manpage, and HTML documentation.

   192.168.213.21 192.168.213.10 : PSK "minHemligaNyckel"

   # RSA private key for this host, authenticating it to any other host
   # which knows the public part.  Suitable public keys, for ipsec.conf,
DNS,
   # or configuration of other implementations, can be extracted
conveniently
   # with "ipsec showhostkey".
   : RSA    {
            # RSA 2192 bits   Mannerheim   Tue Apr 16 13:41:08 2002
            # for signatures only, UNSAFE FOR ENCRYPTION
            #
pubkey=0sAQO3blyJDehZ7pg68Dnl7GcuWoDhG4WsgxopsqJ90eKzEPQLojseE890...
            # (0x4200 = auth-only host-level, 4 = IPSec, 1 = RSA)
```

C.2 ipsec.conf

Denna fil innehåller information om hur de olika IPSec systemen skall kommunicera med varandra. Här specificerar man bland annat hur systemen ska autentisera sig för varandra, om trafiken skall vara krypterad eller endast autentiserad det vill säga om ESP eller AH skall användas. Vilket mode som ska användas transport eller tunnel, och mellan vilka system trafiken gäller. I den första sektionen i filen anges allmänna inställningar om hur systemet är konfigurerat, till exempel vilket interface som ska IPSec-säkras och hur Pluto ska bete sig. Sedan följer en sektion där man anger default inställningarna som ska gälla för alla kommunikationer om inget annat anges. Sist i filen är sektionerna där man specificerar de egendefinierade uppkopplingarna som skall gälla, här kan man också ange andra inställningar än dem som angetts i default sektionen. Sist i detta kapitel visas ett utdrag ur vår *ipsec.conf* fil, som specificerar en uppkoppling mellan två värddatorer användande transport mode och både ESP och AH. Här följer några förklaringar på de olika parametrarna som används i sektionerna i filen:

config setup

- **interfaces:** anger på vilket interface/interfaces som IPSec skall användas på. Om man bara har ett interface går det lika bra att skriva *interfaces=%defaultroute* istället för att specificera ett visst interface som till exempel *interfaces=”ipsec0=eth0”*.
- **klipsdebug:** anger hur mycket debug-information för KLIPS som ska genereras.
- **plutodebug:** anger hur mycket debug-information för Pluto som ska genereras.
- **plutoload:** här anges vilka uppkopplingar som automatiskt ska laddas när Pluto startar. Ofta brukar man ange *%search* eftersom då kommer alla uppkopplingar med parametern *auto=add* att laddas automatiskt. När parametern har blivit satt till föregående så väntar Pluto på att de andra systemet ska ta kontakt istället för att göra det själv.
- **plutostart:** här anges vilka uppkopplingar som automatiskt ska förhandlas fram när Pluto startar. På samma sätt som ovan, brukar man ange *%search*, för att på så sätt starta alla uppkopplingar med parametern *auto=start* automatiskt. När man har parametern *auto=start* satt, så kommer Pluto att själv starta förhandlingen om IPSec-uppkopplingen, vilket är att föredra i de flesta fall.

conn %default

- **keyingtries:** här anges hur mycket förhandling om uppkopplingar som ska utföras. Ett lågt nummer här anger att man ger upp snabbt om förhandlingar misslyckas, och ett högt nummer anger att man är mer envis. Att observera är dock att 0 innebär att man aldrig ger upp.

conn *namn på uppkopplingen*

- **type:** här anger man vilket mode som ska användas *transport* för transport mode och *tunnel* för tunnel mode.
- **right:** den ena partens yttre interface i kommunikationen.
- **left:** den andra partens yttre interface i kommunikationen. Det har ingen betydelse vilken man anger som *right* respektive *left* bara man använder dem konsekvent. På så sätt är det enkelt att överföra filen till det andra (Linux)systemet, utan att behöva ändra i den (förutom möjligtvis conn setup parametrarna).
- **auth:** hur IP-datagrammen ska autentiseras, det vill säga *ah* för AH eller *esp* för autentisering med ESP, anges här. Vilka algoritmer som ska gälla för AH respektive ESP kan anges explicit med verktyget *ipsec spi*.

- **pfs:** om man ska använda sig av *Perfect Forward Secrecy* vid nyckelhanteringen eller ej, bestäms med hjälp av denna parameter.
- **authby:** här anges vilken metod systemen skall använda sig av vid autentiseringen vid nyckelförhandlingen. Kan antingen var *secret* för autentisering med hjälp av shared secret, eller också *rsasig* för autentisering med RSA-signaturer.
- **auto:** som nämdes ovan i *conn setup* sektionen anger denna parameter om Pluto ska ska ladda eller starta uppkopplingen automatiskt.

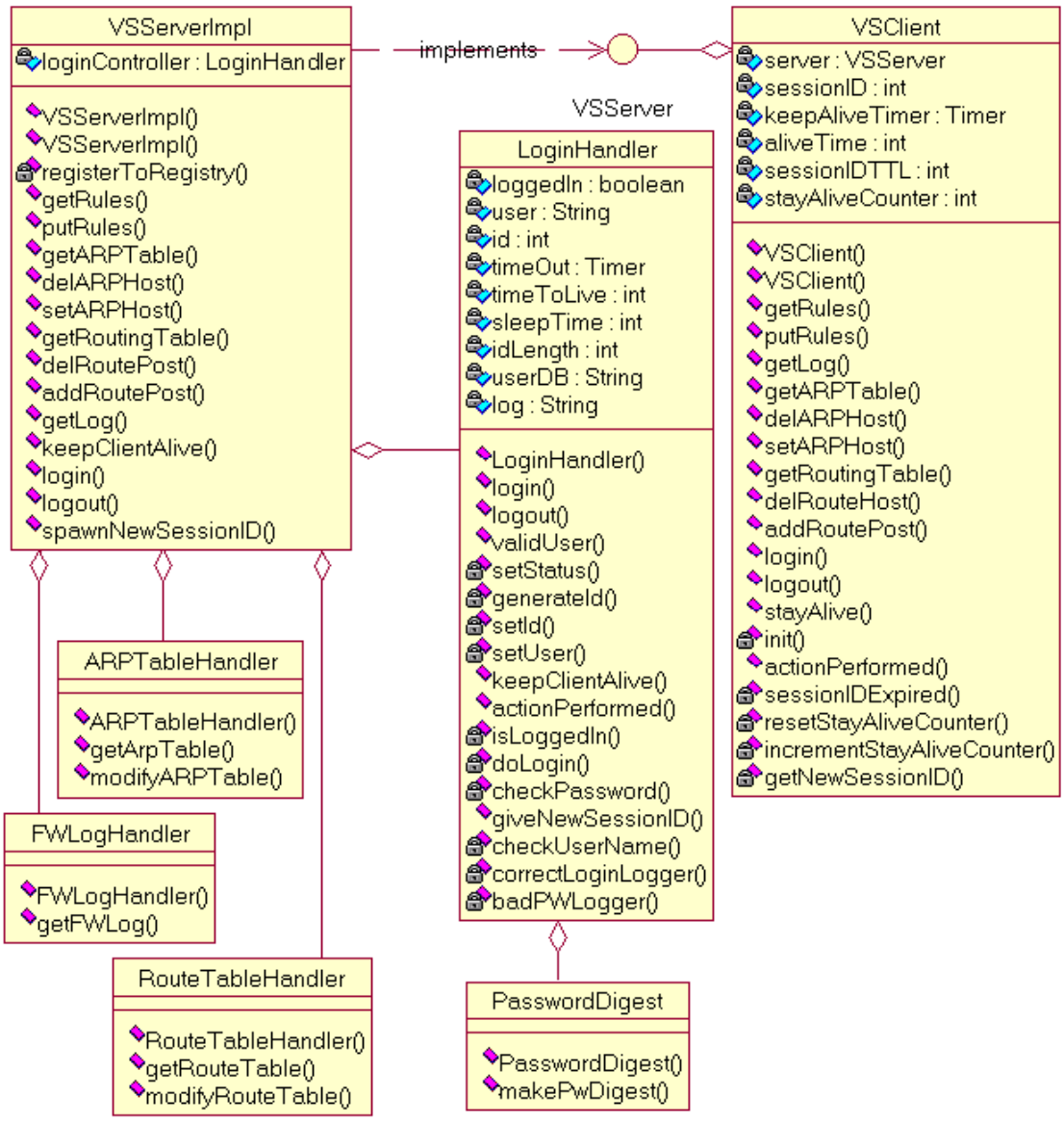
Det finns många fler parametrar än de som har beskrivits här ovan och den mer intresserade hänvisas till *ipsec.conf(5)* mansidorna på FreeS/WANs hemsida för fullständig information. Att tänka på när man skriver en *ipsec.conf* fil är att den första okommenterade raden i en sektion inte får vara indragen medan alla okommenterade efterföljande rader måste vara indragna. Blankrader får inte förekomma i en sektion, eftersom de används just för att avdela sektioner. Om man vill ha en tomrad i en sektion så får man använda kommentartecknet # på en ensam rad istället.

Exempel på en ipsec.conf-fil:

```
# /etc/ipsec.conf - FreeS/WAN IPsec configuration file

# basic configuration
config setup
   # THIS SETTING MUST BE CORRECT or almost nothing will work;
   interfaces="ipsec0=eth0"
   # Debug-logging controls:  "none" for (almost) none, "all" for lots.
   klipsdebug=none
   plutodebug=none
      # Use auto= parameters in conn descriptions to control startup
   actions.
   plutoload=%search
   plutostart=%search



# defaults for subsequent connection descriptions
conn %default
   keyingtries=0


# here follows the specifications for our connection
# our connection with the w2k-computer with encryption
conn our_conn_esp
   type=transport
   right=192.168.213.10
   left=192.168.213.21
   auth=esp
   pfs=yes
```

D Design

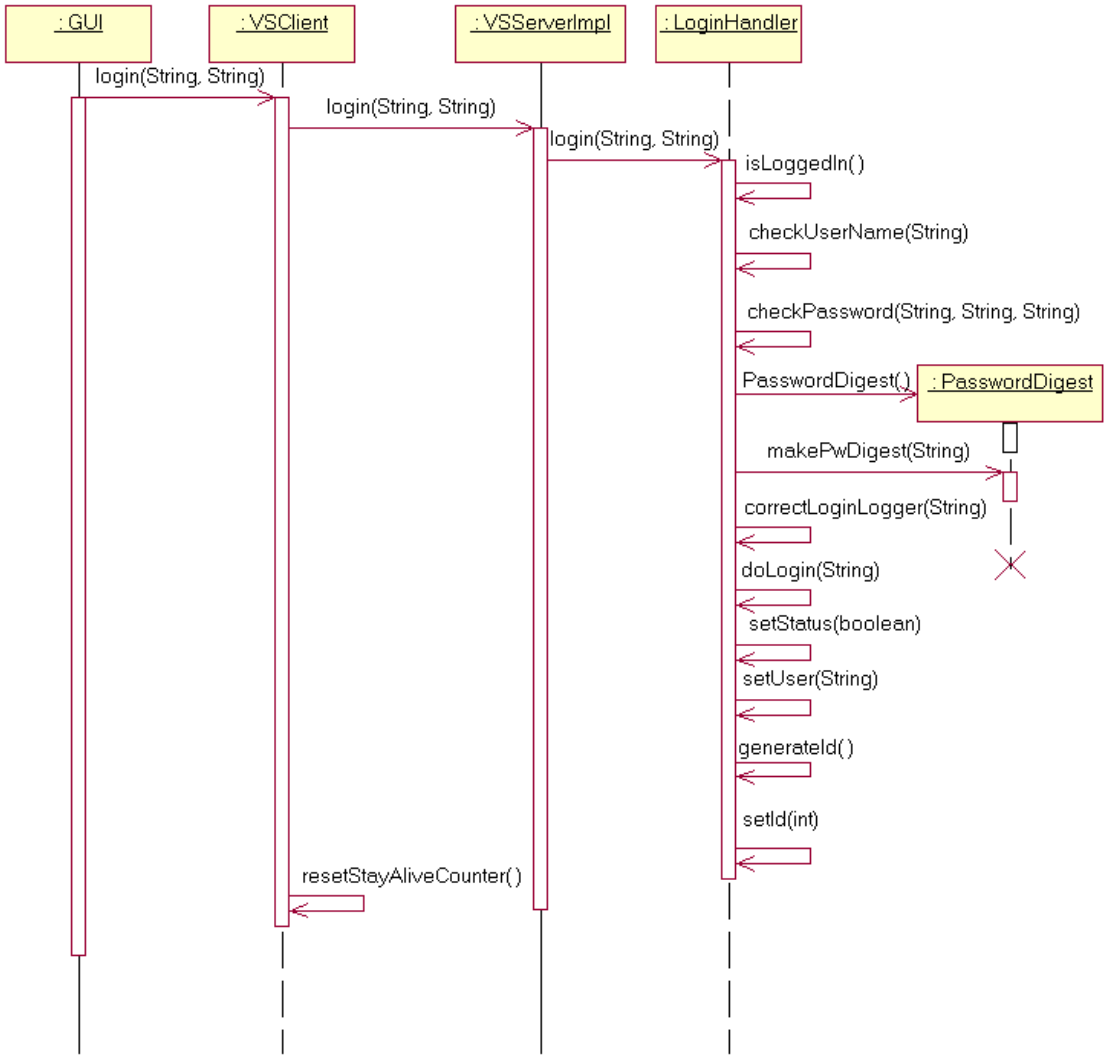
D.1 Klassdiagram



Figur D.13: Klassdiagram.

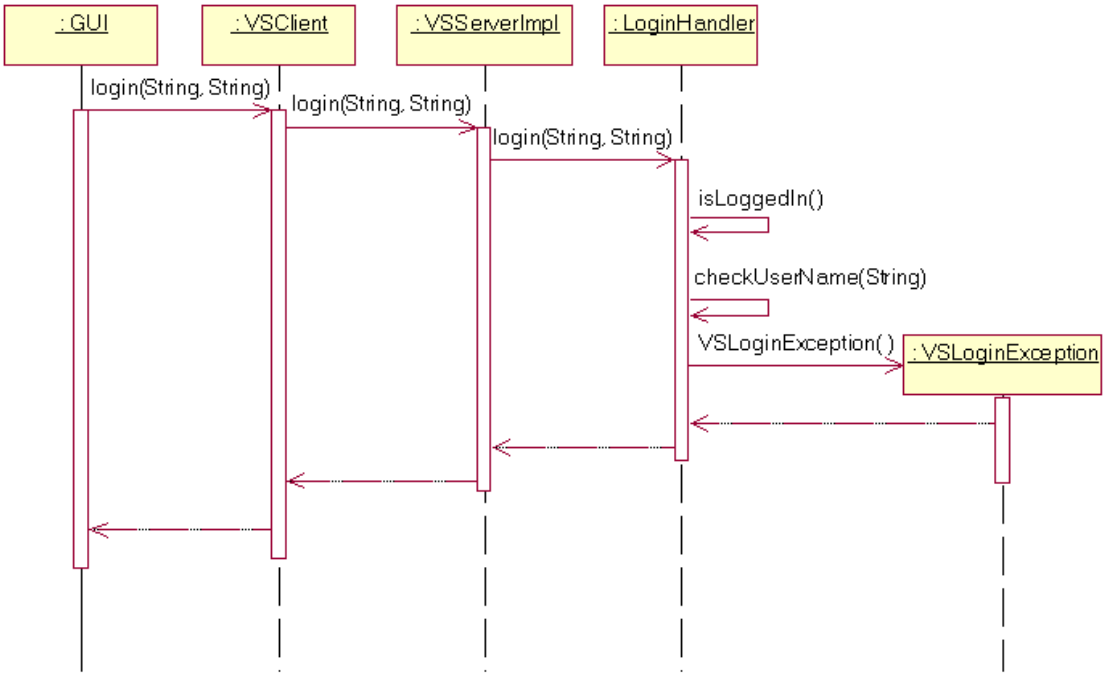
D.2 Sekvensdiagram

En lyckad inloggning



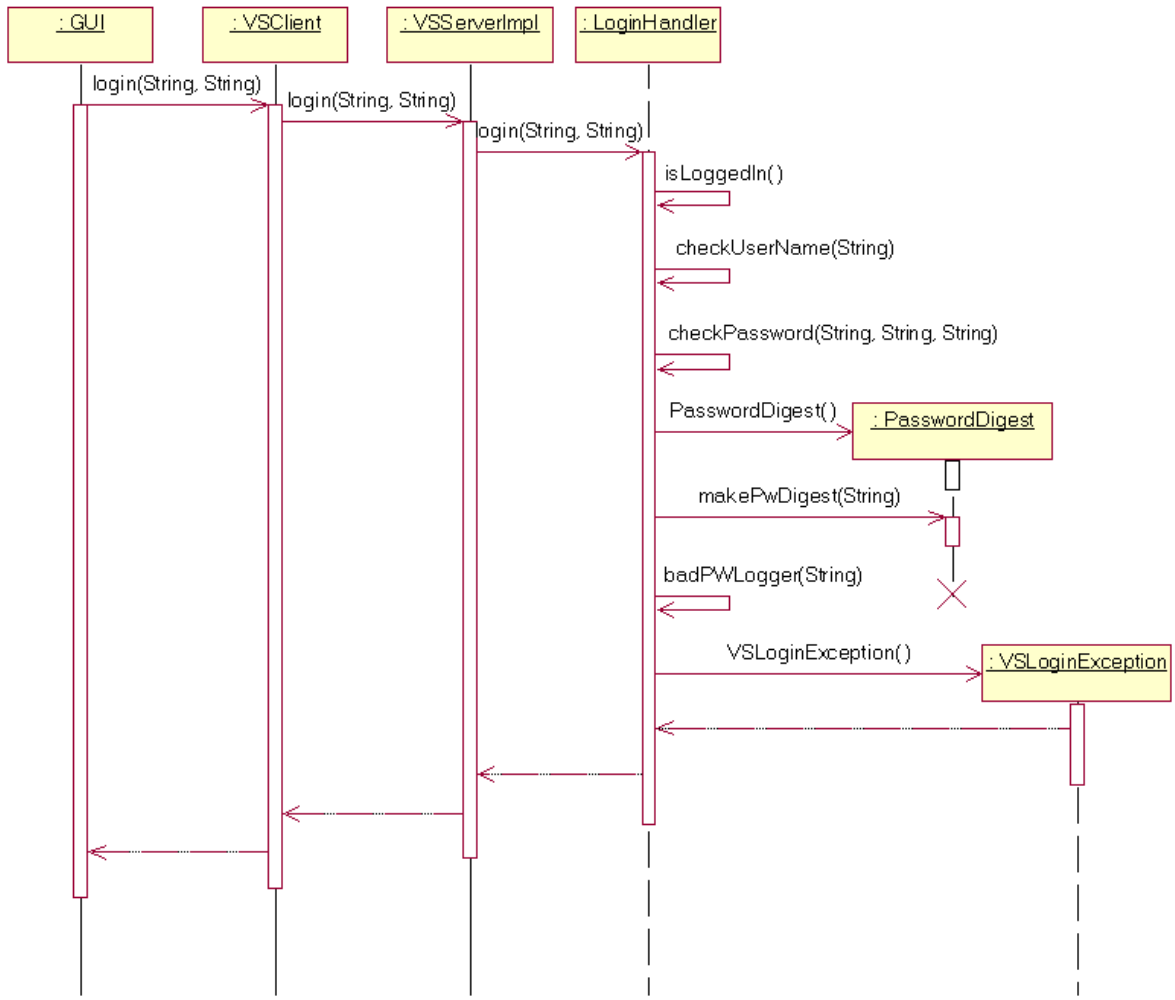
Figur D.14: Lyckad inloggning.

Ett misslyckat inloggningsförsök – felaktigt användarnamn



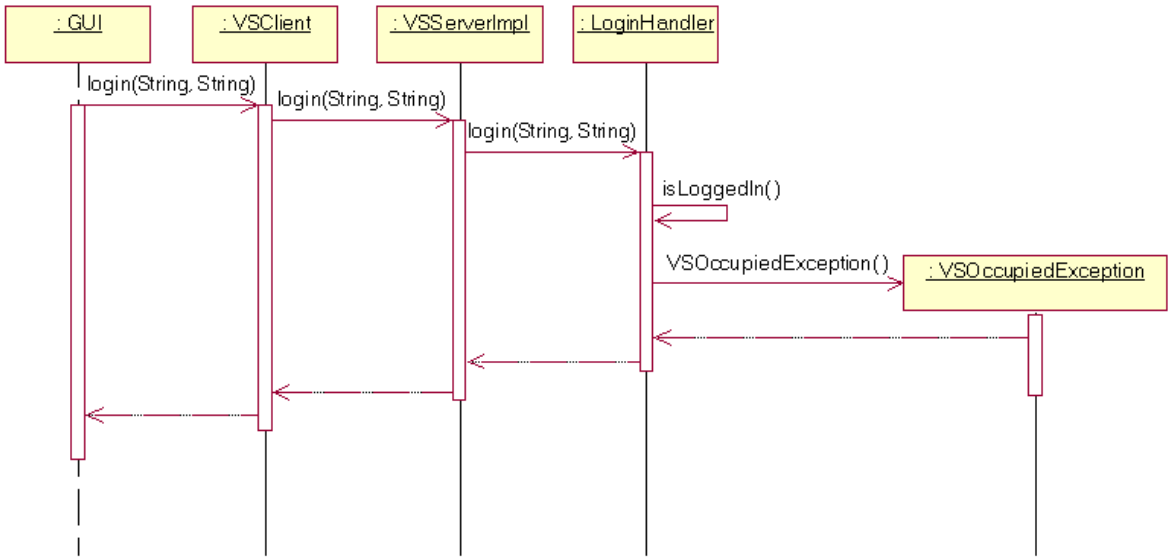
Figur D.15: Misslyckad inloggning med fel användarnamn.

Ett misslyckat inloggningsförsök – felaktigt lösenord



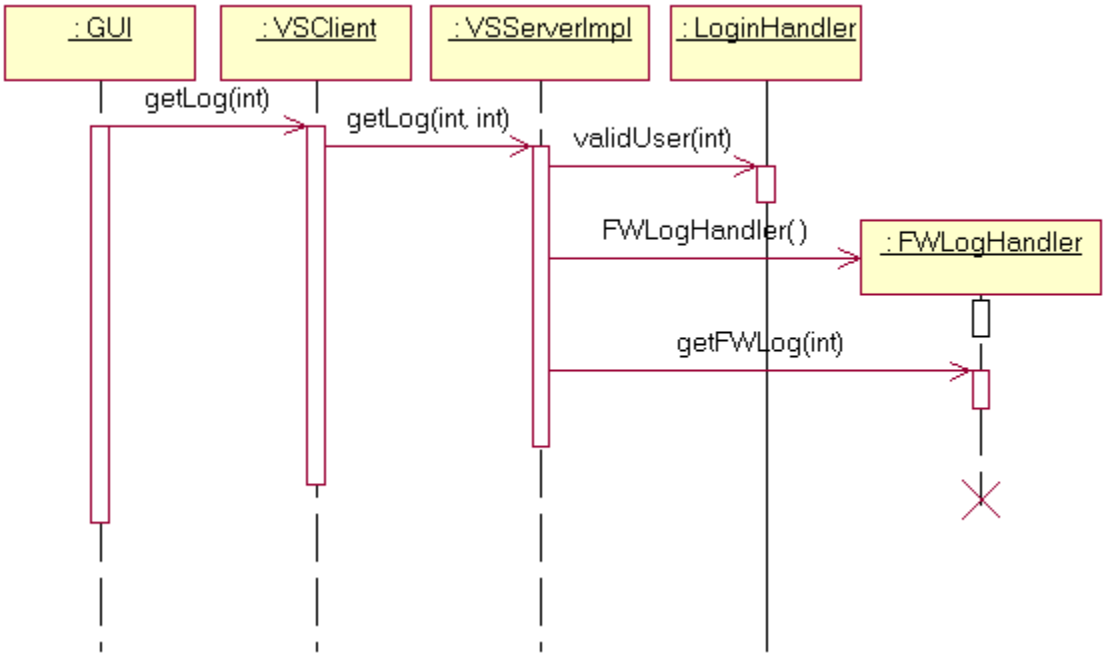
Figur D.16: Misslyckad inloggning med fel lösenord.

Ett misslyckat inloggningsförsök – servern upptagen



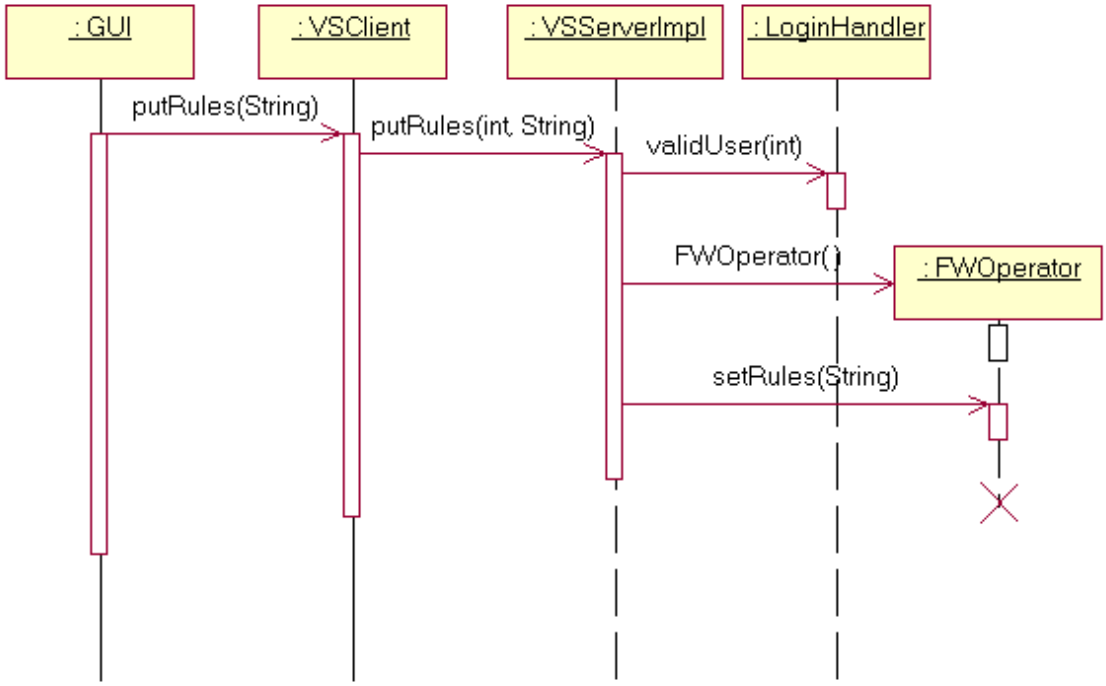
Figur D.17: Misslyckad inloggning när servern är upptagen.

Ett lyckat hämtande av brandväggens regelverk



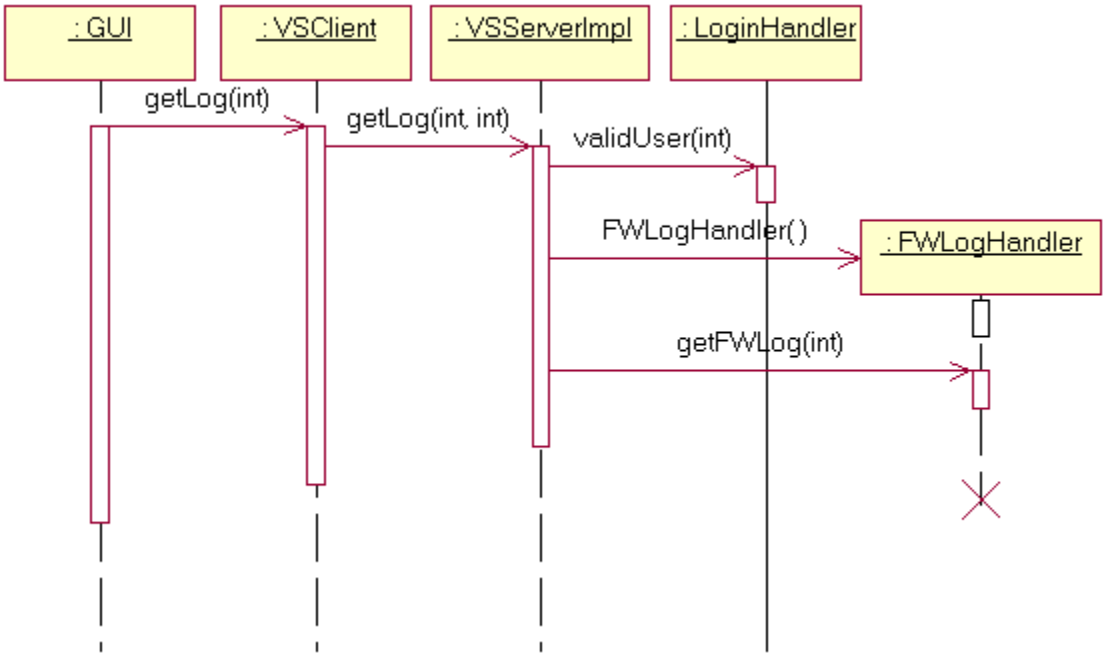
Figur D.18: Hämtande av brandväggens regelverk.

Insättning av nytt regelverk i brandväggen



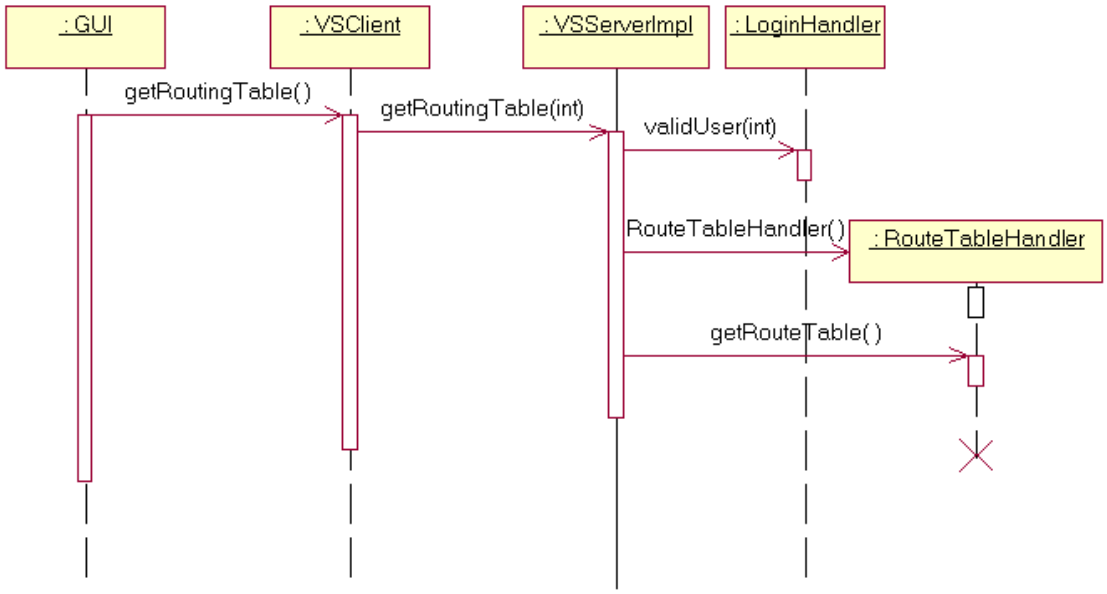
Figur D.19: Insättning av regelverk i brandväggen.

Ett lyckat hämtande av brandväggsloggen



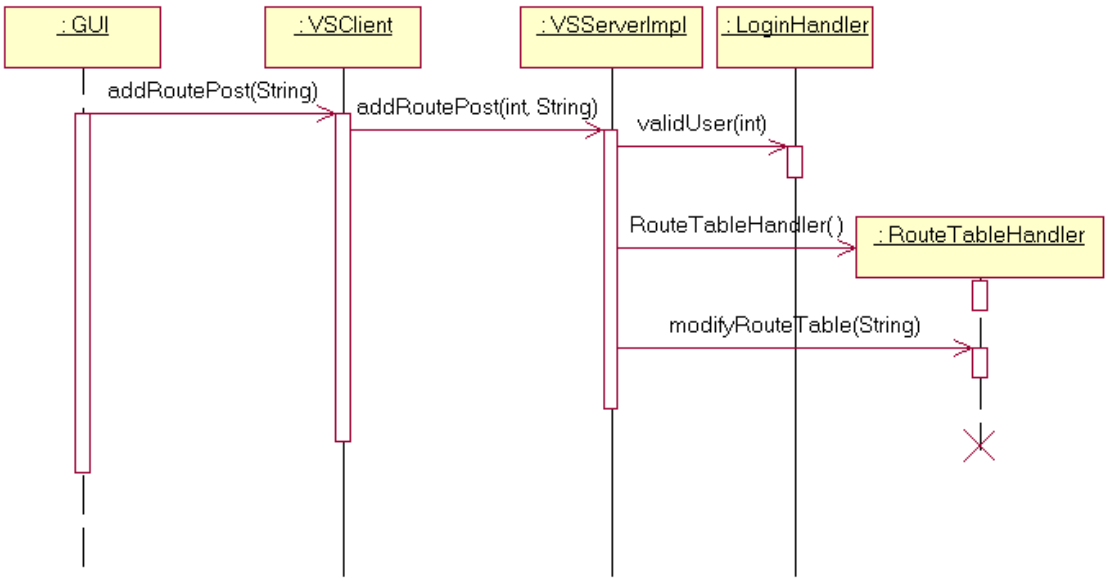
Figur D.20: Hämtande av brandväggslogg.

Ett lyckat hämtande av route-tabellen



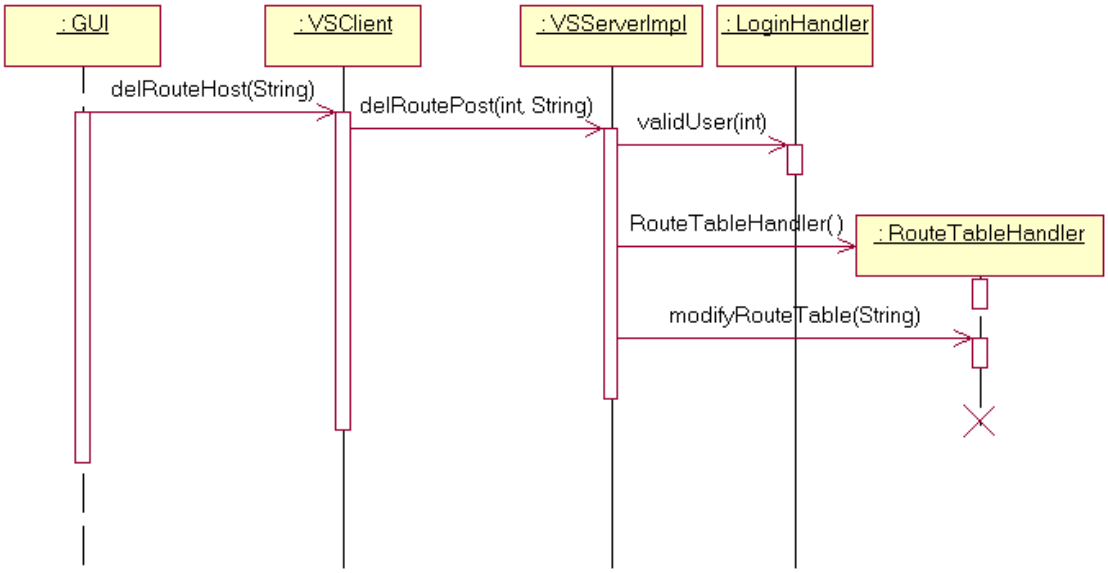
Figur D.21: Hämtande av route-tabellen.

Insättning av ny route-tabellpost



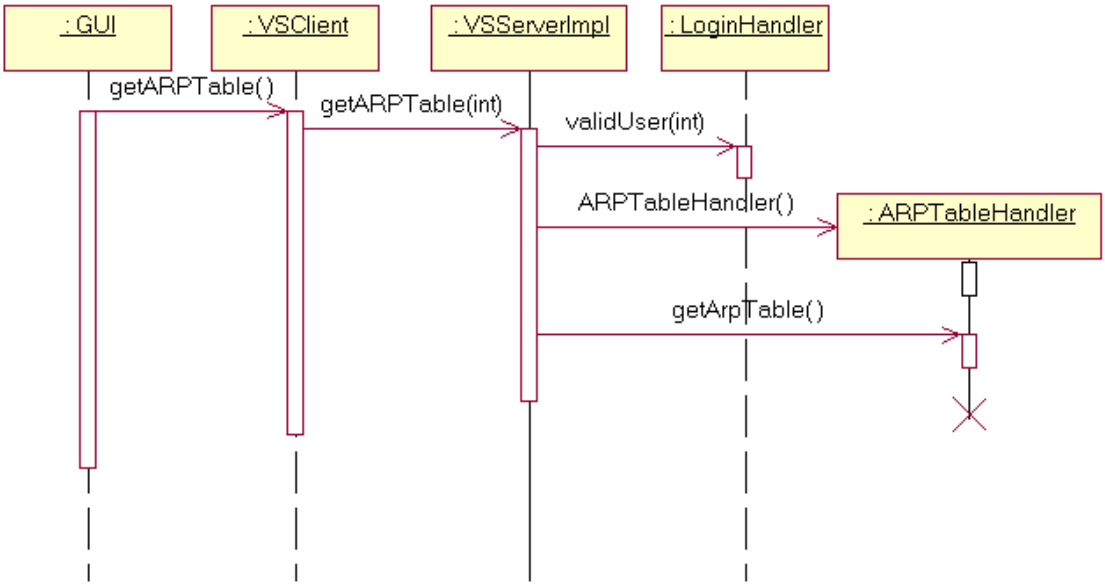
Figur D.22: Insättning av ny route-tabellpost.

Ett lyckat borttagande av en route-tabellpost



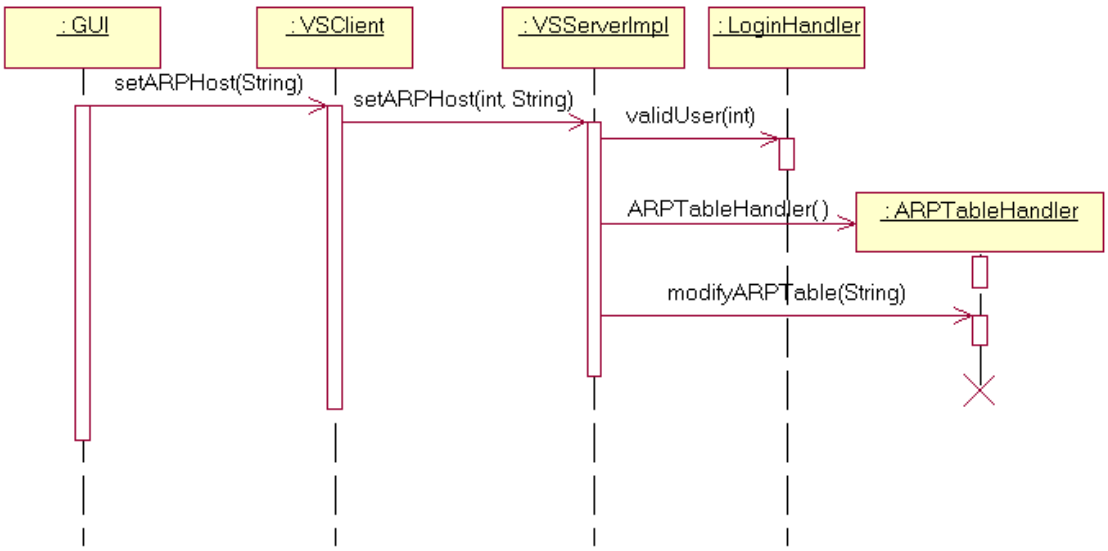
Figur D.23: Borttagande av route-tabellpost.

Ett lyckat hämtande av ARP-tabellen



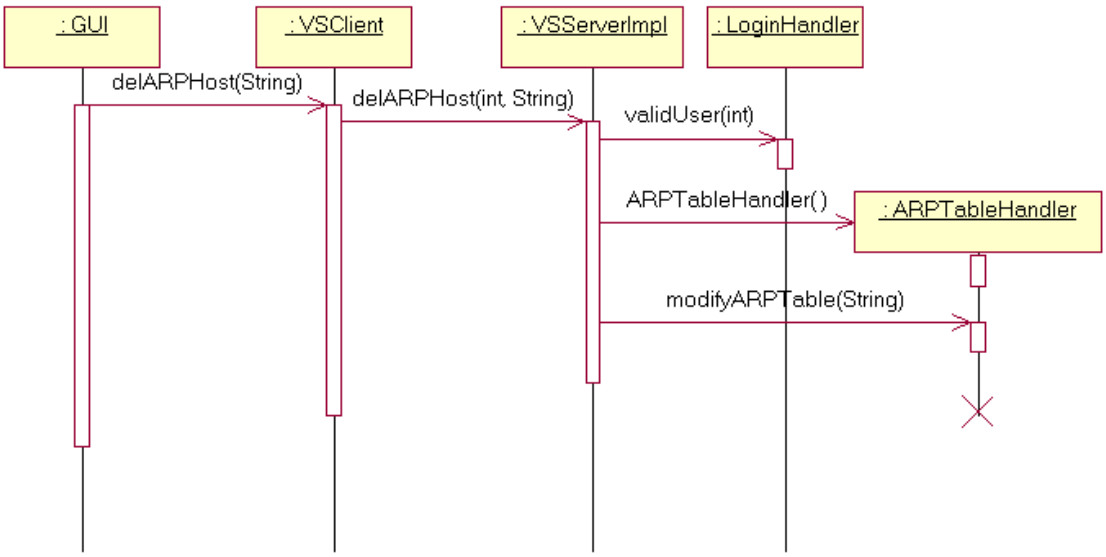
Figur D.24: Hämtande av ARP-tabellen.

Insättning av ny ARP-tabell-host



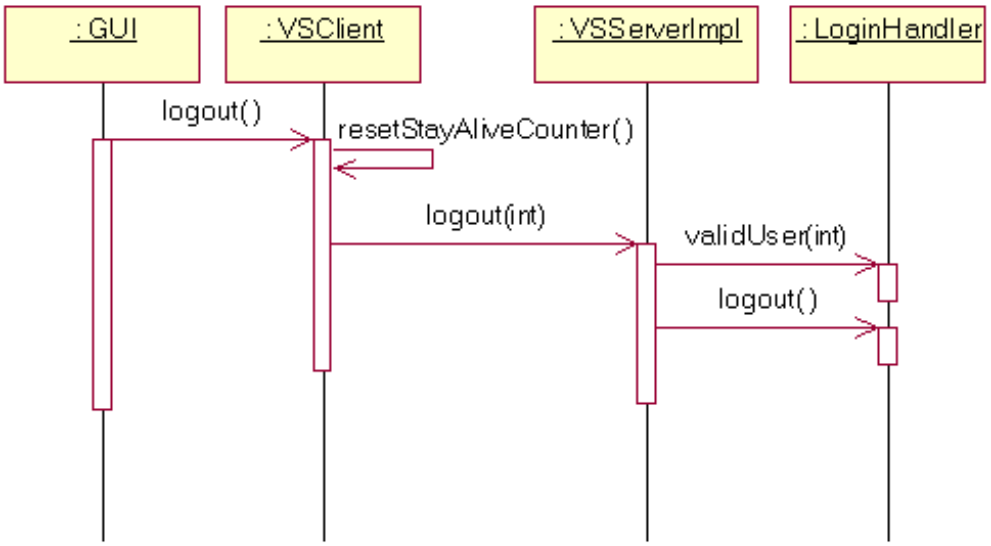
Figur D.25: Insättning av ny ARP-tabell-host.

Ett lyckat borttagande av en ARP-tabell-host



Figur D.26: Borttagande av ARP-tabell-host.

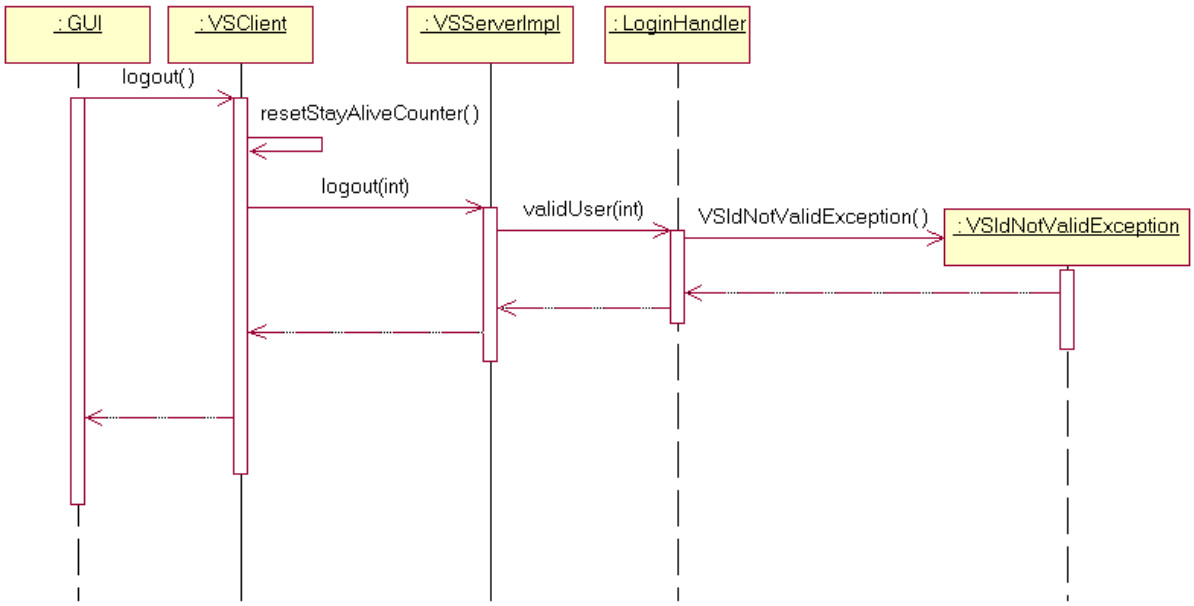
Lyckad utloggning



Figur D.27: Lyckad utloggning.

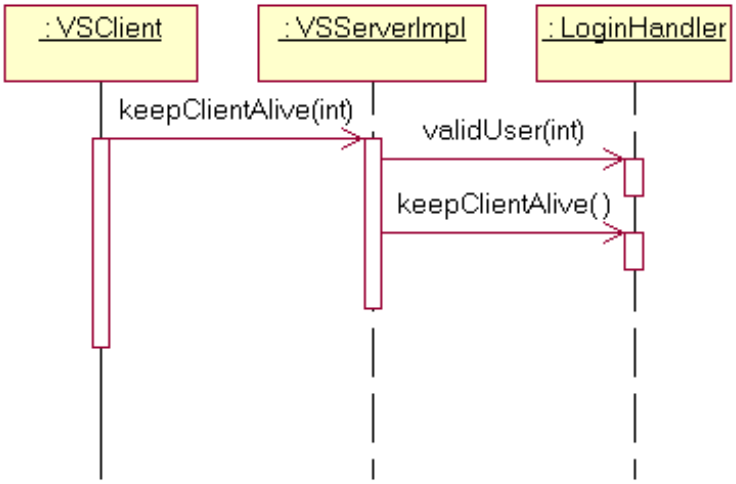
Misslyckad utloggning

Det fungerar på motsvarande sätt med andra anrop när sessions-ID inte är korrekt. De enda skillnaderna är metodanropen och att resetStayAliveCounter()-anropet inte sker.



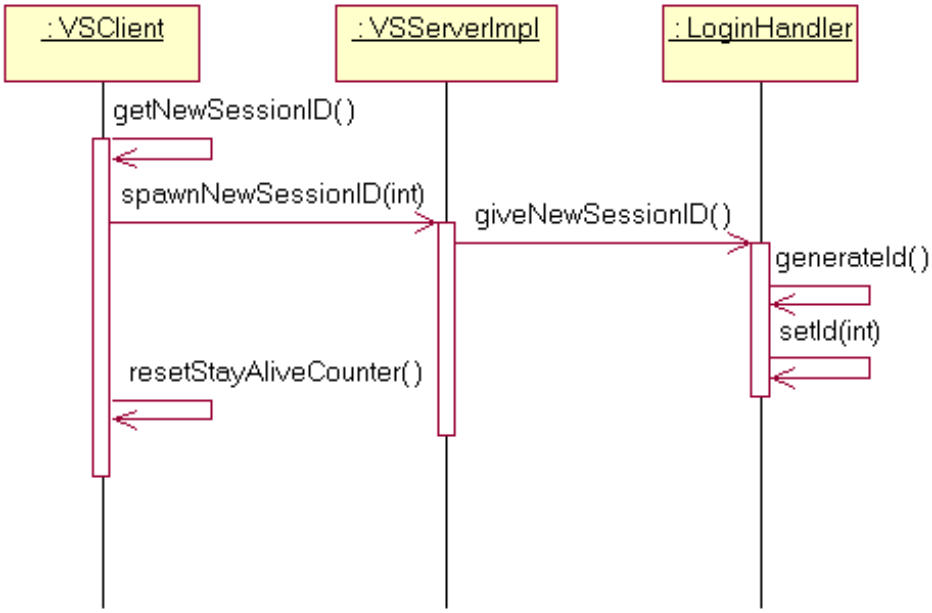
Figur D.28: Misslyckad utloggning.

Ett keep alive-meddelande från klienten



Figur D.29: Keep alive-meddelande från klienten.

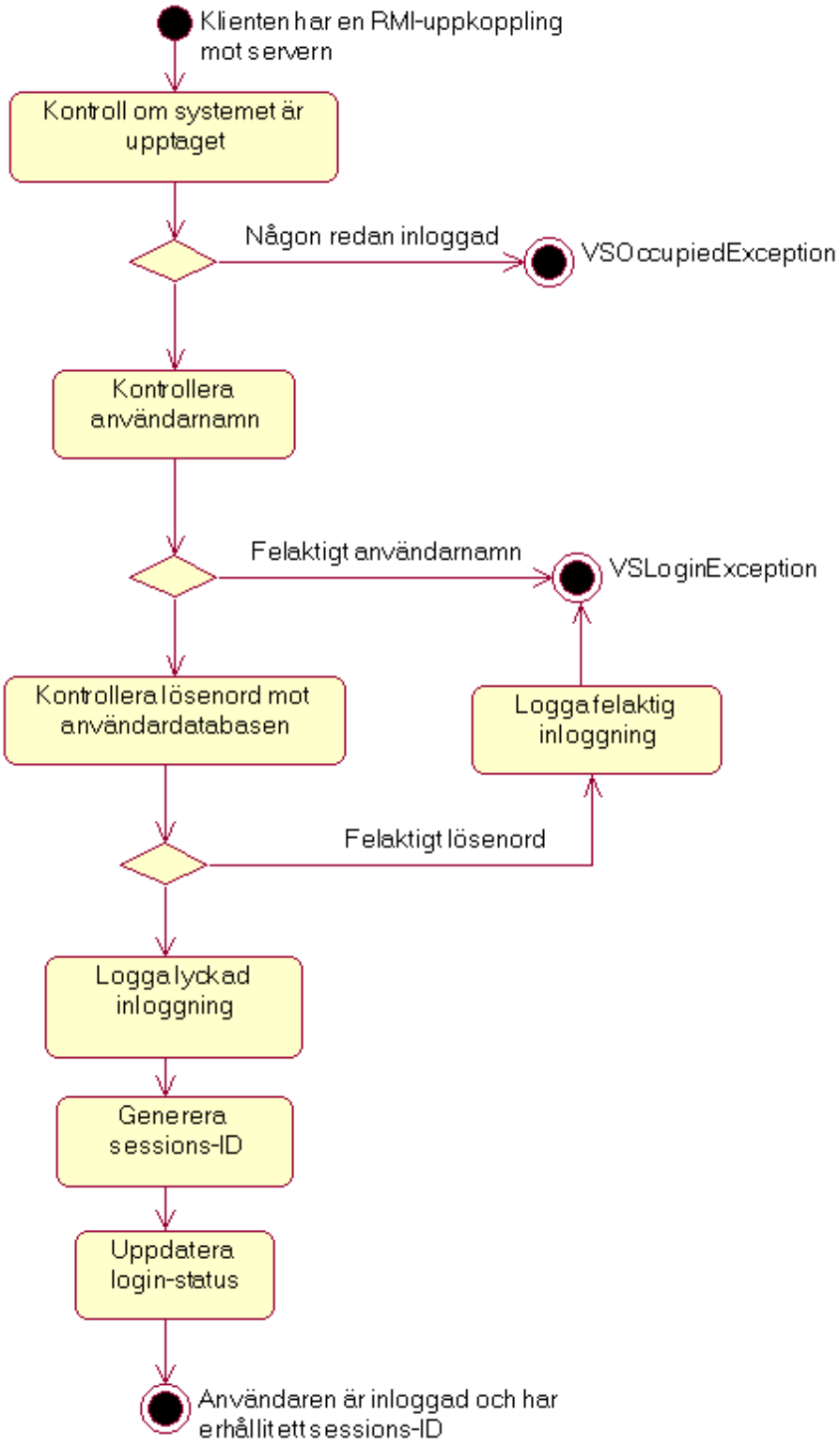
Ett byte av sessions-ID



Figur D.30: Byte av sessions-ID.

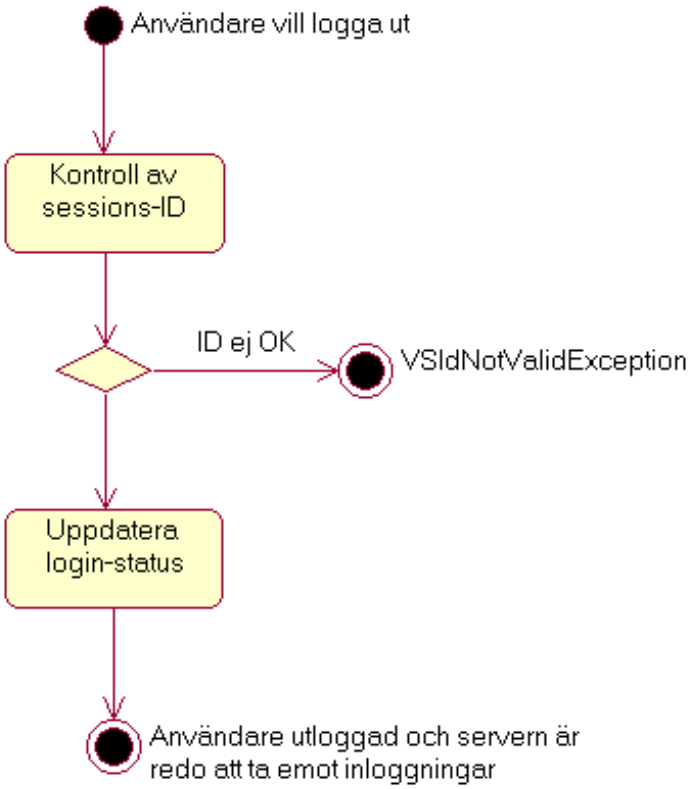
D.3 Aktivitetsdiagram

Inloggning



Figur D.31: Aktivitetsdiagram - inloggning.

Utloggning



Figur D.32: Aktivitetsdiagram - utloggning.

E API

Constructor Detail

VSClient

```
public VSClient(java.net.InetAddress ip)
         throws java.rmi.NotBoundException,
                java.net.MalformedURLException,
                java.rmi.RemoteException
```

Constructs VSClient object with an IP-address as parameter.

Pre: True.

Post: An object of class VSClient has been initialized.

Parameters:

`ip` - The IP-address to the server.

Throws:

`java.rmi.NotBoundException` - A NotBoundException is thrown if an attempt is made to lookup or unbind in the registry a name that has no associated binding.

`java.net.MalformedURLException` - Thrown to indicate that a malformed URL has occurred. Either no legal protocol could be found in a specification string or the string could not be parsed.

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

VSClient

```
public VSClient(java.lang.String address)
         throws java.rmi.NotBoundException,
                java.net.MalformedURLException,
                java.rmi.RemoteException
```

Constructs VSClient object with an URL as parameter.

Pre: True.

Post: An object of class VSClient has been initialized.

Parameters:

`address` - The URL to the server.

Throws:

`java.rmi.NotBoundException` - A NotBoundException is thrown if an attempt is made to lookup or unbind in the registry a name that has no associated binding.

`java.net.MalformedURLException` - If address cannot be used to construct a valid URL.

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

Method Detail

getRules

```
public java.lang.String getRules()
                         throws java.rmi.RemoteException,
                                VSIdNotValidException,
                                java.io.IOException,
                                java.lang.InterruptedException
```

Gets the firewall rules from the server

Pre: True.

Post: The firewall rules has been delivered if the session ID was valid else a VSIdNotValidException has been thrown.

Returns:

A String object representing the rule chains.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

putRules

```
public boolean putRules(java.lang.String rules)
                 throws java.rmi.RemoteException,
                        VSIdNotValidException,
                        java.io.IOException,
                        java.lang.InterruptedException
```

Sends the firewall rules to the server.

Pre: True.

Post: The rules has been installed if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`rules` - The rules that are to be sent to and installed on the server.

Returns:

A boolean value representing the success status of the installation of the rules.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

getLog

```
public java.lang.String getLog(int rows)
                        throws java.rmi.RemoteException,
                               java.io.IOException,
                               VSIdNotValidException,
                               java.lang.InterruptedException
```

Gets the updated log generated by the firewall.

Pre: True.

Post: The log has been returned if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`rows` - The number of rows from the log that already has been received.

Returns:

A String representing the firewall log that has not yet been received.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

getARPTable

```
public java.lang.String getARPTable()
                              throws java.rmi.RemoteException,
                                     java.io.IOException,
                                     VSIdNotValidException,
                                     java.lang.InterruptedException
```

Gets the arp table from the server.

Pre: True.

Post: The arp table has been returned if the session ID was valid else a VSIdNotValidException has been thrown.

Returns:

A String representing the arp table.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a

remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

delARPHost

```
public java.lang.String delARPHost(java.lang.String flags)
                            throws java.rmi.RemoteException,
                                   java.io.IOException,
                                   VSIdNotValidException,
                                   java.lang.InterruptedException
```

Deletes a host in the servers arp table.

Pre: True.

Post: A host in the servers arp table has been deleted if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`flags` - The arguments for deleting the host.

Returns:

A String representing the possible error.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

setARPHost

```
public java.lang.String setARPHost(java.lang.String flags)
                            throws java.rmi.RemoteException,
                                   java.io.IOException,
                                   VSIdNotValidException,
                                   java.lang.InterruptedException
```

Sets an new host in the servers arp table.

Pre: True.

Post: A host in the servers arp table has been set if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`flags` - The arguments for setting the host.

Returns:

A String representing the possible error.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

getRoutingTable

```
public java.lang.String getRoutingTable()
                                throws java.rmi.RemoteException,
                                       java.io.IOException,
                                       VSIdNotValidException,
                                       java.lang.InterruptedException
```

Gets the routing table from the server.

Pre: True.

Post: The route table has been returned if the session ID was valid else a VSIdNotValidException has been thrown.

Returns:

A String representing the routing table.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

delRoutePost

```
public java.lang.String delRoutePost(java.lang.String flags)
                              throws java.rmi.RemoteException,
                                     java.io.IOException,
                                     VSIdNotValidException,
                                     java.lang.InterruptedException
```

Deletes a post in the servers route table.

Pre: True.

Post: A post in the servers route table has been deleted if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`flags` - The arguments for deleting a post.

Returns:

A String representing possible error.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

addRoutePost

```
public java.lang.String addRoutePost(java.lang.String flags)
                               throws java.rmi.RemoteException,
                                      java.io.IOException,
                                      VSIdNotValidException,
                                      java.lang.InterruptedException
```

Adds a new post in the servers route table.

Pre: True.

Post: A post in the servers route table has been added if the session ID was valid else a VSIdNotValidException has been thrown.

Parameters:

`flags` - The arguments for adding a post.

Returns:

A String representing possible error.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

login

```
public void login(java.lang.String user,
                  java.lang.String pw)
           throws java.rmi.RemoteException,
                  VSLoginException,
                  VSOccupiedException,
                  java.io.IOException,
                  java.security.NoSuchAlgorithmException
```

Logs in and authenticates the user to the server.

Pre: True.

Post: The user is logged in and has got a session ID if no one was already logged in and the username and the password was valid. If the server was already occupied a VSOccupiedException has been thrown. If the username or the password was not valid a VSLoginException has been thrown.

Parameters:

`user` - The user name.

`pw` - The password.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`VSLoginException` - Thrown to indicate that the login was incorrect.

`VSOccupiedException` - Thrown to indicate that the server is occupied i.e. a user is already logged on.

`java.io.IOException` - Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

`java.security.NoSuchAlgorithmException`

---

logout

```
public void logout()
```

```
        throws java.rmi.RemoteException,
               VSIdNotValidException,
               java.lang.InterruptedException
```

Logs out the user from the server.

Pre: True.

Post: The user is logged out if the session ID was valid else a VSIdNotValidException has been thrown.

Throws:

`java.rmi.RemoteException` - A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.

`VSIdNotValidException` - Thrown to indicate that the session ID was incorrect.

`java.lang.InterruptedException` - Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

---

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent actionEvent)
```

Catches an event and tells the server that the client is alive.

Pre: True.

Post: The event has been handled.

Specified by:

`actionPerformed` in interface `java.awt.event.ActionListener`

Parameters:

`actionEvent` - The event that has occured.

F Använd mjukvara och hårdvara i projektet

Vi har använt oss av följande system:

Processor	Intel Pentium 3 633MHz
Primärminne	128 Mb
Operativsystem	Microsoft Windows 2000 Server

Tabell F.1: System 1.

Processor	Intel Pentium 4 1,6 GHz
Primärminne	128 Mb
Operativsystem	Microsoft Windows XP Professional Edition / Red Hat 7.2

Tabell F.2: System 2.

Processor	Intel Pentium 4 1,6 GHz
Primärminne	128 Mb
Operativsystem	Microsoft Windows XP Professional Edition

Tabell F.3: System 3.

Utvecklingsverktyg: Forte for Java

Övrigt: Sniffer Pro
IP Security Monitor
FreeS/WAN
Tcpdump

G Ordlista

Adress Resolution Protocol: Protokoll som används för att mappa en känd IP-adress med en maskinvaruadress, till exempel en MAC-adress. [17]

ARP: Se *Address Resolution Protocol*. [19]

Autentisering: Kontroll av uppgiven identitet, till exempel vid inloggning, vid kommunikation mellan två system eller vid utväxling av meddelande mellan användare. [18]

Brandvägg: Hinder mot oönskad kommunikation mellan olika datornät, främst mot intrång. [18]

CA: Se *Certification authority*.

Certification authority: På svenska 'certifikatutfärdare'. Ett certifikat krävs vid krypterad överföring av till exempel information på Internet. I Sverige är till exempel Posten och Telia certifikatutfärdare (år 2000). [5]

Client: Se *Klient*.

Common Object Request Broker Architecture: En standard för distribuerade objekt från Object Management Group (OMG). CORBA specificerar ett ramverk för att distribuerade applikationer skall kunna arbeta tillsammans över nätverk. Tanken med CORBA är att det skall vara lätt att integrera applikationer som kör på olika plattformar. [5]

CORBA: Se *Common Object Request Broker Architecture*.

Databas: Samling av strukturerad datorlagrad information. Det finns speciella program som håller reda på informationen i en databas. De kallas databashanteringssystem (DBMS). Ibland syftar ordet på den dator där informationen finns, till exempel en BBS. [15]

eXtensible Markup Language: Bantad version av SGML. [16]

Firewall: Se *Brandvägg*.

Gateway: Utrustning som förbinder olika sorters nät eller konverterar mellan olika kommunikationsprotokoll. [15]

Gränssnitt: Kontaktyta mellan olika funktioner eller delar i ett system [18]

Handshaking: Se *Handskakning*.

Handskakning: [inom datakommunikation:] Funktion som innebär att sändande och mottagande utrustning underrättar varandra om att de är redo att sända respektive ta emot. [18]

HTML: Se *Hypertext Markup Language*.

HTTP: Se *Hypertext Transfer Protocol*.

Hypertext Markup Language: Sidbeskrivningsspråket för dokument i World Wide Web. Innehåller direktiv för hantering av typsnitt, bilder och länkar till andra dokument. Innehåller möjlighet att styra vidare till annan WWW-sida via URL. [16]

Hypertext Transfer Protocol: Det protokoll som ligger till grund för överföring av dokument i World Wide Web. [15]

ICMP: Se *Internet Control Message Protocol*.

IDL: Se *Interface Definition Language*.

IETF: Se *Internet Engineering Task Force*.

Interface: Se *Gränssnitt*.

Interface Definition Language: Metaspråk som används för att beskriva ett objekts tjänster på ett språkoberoende sätt. [16]

Internet: Det stora internationella nätet av många sammankopplade datornät, med TCP/IP som kommunikationsprotokoll. Stavas med versalt i. [15]

Internet Control Message Protocol: Den del av IP-protokollet som hanterar fel- och kontrollmeddelanden. Används bland annat av routrar för att återrapportera fel till ursprunglig källa. [16]

Internet Engineering Task Force: Den internationella grupp som samordnar teknik-utvecklingen i Internet. [15]

Internet Protocol: Kommunikationsprotokoll som beskriver adressering och vägval (engelska routing) för datapaket i Internet. [15]

IP: Se *Internet Protocol*.

IP-adress: Logisk adress som tilldelas alla datorer på Internet. Varje Internetansluten dator måste ha en unik IP-adress. [15]

IPv4: Det IP-protokoll som för närvarande används på Internet.

IPv6: Kommer att innehålla funktioner för betalning av nätanvändning, bättre säkerhet samt möjliggöra storlekstillväxt av Internet. [15]

Java: Programmeringsspråk, gjort för att användas på Internet. [15]

Klient: Program som utnyttjar de tjänster som erbjuds av en server. Exempelvis Netscape Navigator som används för att "beställa" webbsidor från webbservrar. [15]

MAC: Se *Message Authentication Code*.

MAC-adress: Den fysiska adressen till en enhet som är kopplad till ett nätverk, uttryckt som ett 48-bitars hexadecimalt nummer. [2]

Message Authentication Code: Resultatet av att använda en funktion som tar ett invärde med variabel längd samt en nyckel. Resultatet är ett utvärde som alltid har fast längd oberoende av längden på invärdet.

Proxy: Mellanlagringsdator för Internetsidor. Om flera personer, till exempel på ett företag, ofta tittar på samma Internetsidor är det onödigt att hämta samma sidor flera gånger. Istället låter man en Proxy-server hämta sidorna första gången en användare ber om dem. Om sedan en annan användare ber att få se samma sidor kontrollerar Proxy-servern om sidorna ändrats sedan förra gången. Om de inte gjort det skickas de sidor som lagrat på Proxy-servern och man slipper onödig trafik på Internet. [16]

Remote Method Invocation: Kommunikation mellan två Java processer. [5]

RMI: Se *Remote Method Invocation*.

Router: Dator som väljer väg för och vidarebefordrar data i ett datornät. [18]

Server, serverdator: Dator med ett eller flera serverprogram. [18]

Server, serverprogram: Datorprogram som tillhandahåller gemensamma servicefunktioner i ett datornät, till exempel datalagring och e-postkommunikation. [18]

SGML: Se *Standard Generalised Markup Language*.

Sockets: Programmeringsgränssnitt mot TCP/IP-protokollfamiljen. [5]

Standard Generalised Markup Language: Standard för att beskriva dokuments innehåll och layout, HTML utgör en delmängd av SGML. [16]

Tagg: Tecken eller teckenkombination som används för märkning eller klassificering av data. [18]

TCP: Se *Transmission Control Protocol*.

Transmission Control Protocol: Protokoll som delar upp strömmen av data i paket och garanterar felfri överföring. [15]

Uniform Resource Locator: En standardiserad och unik adress för dokument i Internet. Byggs i huvudsak upp enligt protokoll://dator.adress/katalog/fil.typ. [16]

URL: Se *Uniform Resource Locator*.

WWW: World Wide Web (WWW, W3 eller bara "Web"). Hypertextbaserat system för länkning av dokument över Internet. Även World Weather Watch. [16]

X11: X11 eller X windows är ett fönstersystem för UNIX-datorer som använder TCP/IP som överföringsprotokoll. Jämförbart med, men ganska olikt Microsoft Windows. [15]

XML: Se *eXtensible Markup Language*.