FOA-R----SE

ISSN 1104-9154

rapport

Alf Bengtsson, Jonas Hallberg, David Lindahl, Ulf Lindbergh, Mats Persson

Tillämpad nätverkssäkerhet

Avdelningen för Ledningssystemteknik

SE-581 11 LINKÖPING

FÖRSVARETS FORSKNINGSANSTALT
Avdelningen för Ledningssystemteknik
Box 1165
SE-581 11 LINKÖPING

FOA-R----SE

ISSN 1104-9154

Alf Bengtsson, Jonas Hallberg, David Lindahl, Ulf Lindbergh, Mats Persson

Applied Network Security

:

Dokumentets utgivare	Dokumentbeteckning, ISRN	
Försvarets forskningsanstalt Avdelningen för Ledningssystemteknik Box 1165 SE-581 11 LINKÖPING	FOA-R----SE	
	Dokumentets datum	Uppdragsnummer E7023
	Projektnamn (ev förkortat)	

Upphovsman(män)	Uppdragsgivare
Alf Bengtsson, Jonas Hallberg, David Lindahl, Ulf Lindbergh, Mats Persson	Projektansvarig
	Fackansvarig

Dokumentets titel

Tillämpad nätverkssäkerhet

Sammanfattning

Under våren 1999 följde fem personer ur FOAs grupp för IT-säkerhet en doktorandkurs "Applied Network Security". Kursen avslutades med projektarbeten där FOA ingick i två projekt. Det ena rörde Kerberos/Sesame, det andra WWW-säkerhet. Båda dessa områden har, respektive kommer att få, betydelse inom försvarets ledningssystem.

Syftet med denna rapport är dubbelt. Dels kan nämnda projektrapporter vara underlag för utvecklingen av ledningssystem. Dels skall FOA år 2000 hålla utbildning åt FHS. I kursavsnitt om säkerhet i distribuerade system kan projektrapporterna, kompletterade med en mer allmän beskrivning av området, tjäna som kursmaterial. Denna allmänna beskrivning innehåller fyra arkitekturer för distribuerade system - DCE, CORBA, distribuerade filsystem och WWW. Några olika säkerhetsfunktioner i dessa arkitekturer beskrivs. I appendix finns de två projektrapporterna från doktorandkursen, på engelska.

Nyckelord

Övriga bibliografiska uppgifter	Språk Svenska

ISSN 1104-9154	ISBN	
	Omfång	Pris Enl. prislista
	V. 1.3	

Distributör (om annan än ovan)

Issuing organization	Document ref. No., ISRN	
Defence Research Establishment Division of Command and Control Warfare Technology P O Box 1165 SE-581 11 LINKÖPING SWEDEN	FOA-R----SE	
	Date of issue	**Project No.** E7023
	Project name (abbrev. if necessary)	
Author(s) Alf Bengtsson, Jonas Hallberg, David Lindahl, Ulf Lindbergh, Mats Persson	**Initiator or sponsoring organization**	
	Project manager	
	Scientifically and technically responsible	

Document title in translation

Applied Network Security

Abstract

Key words

Further bibliographic information	**Language** Swedish	
ISSN 1104-9154	**ISBN**	
	Pages	**Price** Acc. to pricelist

Distributor (if not issuing organization)

V.1.1

Tillämpad nätverkssäkerhet

1. Inledning

Distribuerade system får allt större betydelse efter hand som man vill koppla ihop alltfler datorer med alltfler funktioner. Inte minst gäller detta försvarets ledningssystem. Visionen, att ha ett enda ledningssystem, leder till ett gigantiskt distribuerat system. Komplexiteten i stora distribuerade system underskattas lätt. Speciellt har det visat sig vara svårt att hantera säkerhetsfrågorna.

Under våren 1999 följde fem personer ur FOAs grupp för IT-säkerhet en doktorandkurs "Applied Network Security". Kursen avslutades med projektarbeten där FOA ingick i två projekt. Det ena rörde Kerberos/Sesame, det andra WWW-säkerhet. Båda dessa områden har, respektive kommer att få, betydelse inom försvarets ledningssystem.

Syftet med denna rapport är dubbelt. Dels kan nämnda projektrapporter vara underlag för utvecklingen av ledningssystem. Dels skall FOA år 2000 hålla utbildning åt FHS. I kursavsnitt om säkerhet i distribuerade system kan projektrapporterna, kompletterade med en mer allmän beskrivning av området, tjäna som kursmaterial.

:

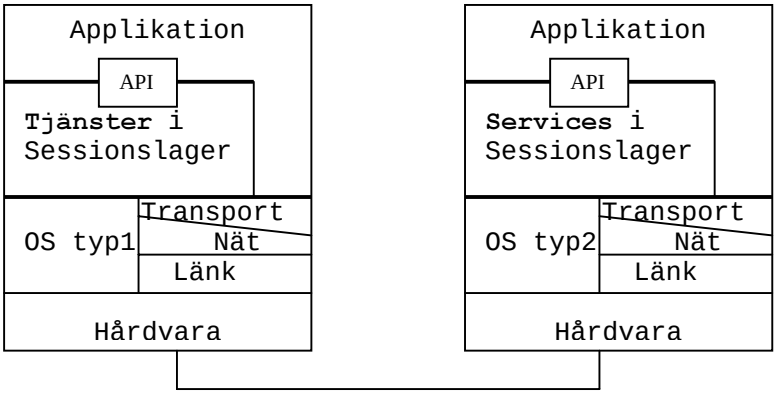
Rapporten är upplagd enligt följande. I kapitel 2 ges en översiktlig beskrivning över några olika arkitekturer hos distribuerade system. I kapitel 3 beskrivs några säkerhetsfunktioner på olika nivåer i systemet. I kapitel 4 kommenteras appendix. I appendix finns de två projektrapporterna.

2. Distribuerade system - olika arkitekturer

En första fråga att ställa sig är vad man menar med ett distribuerat system. Man menar att systemet är distribuerat geografiskt, d v s systemets delar är utspridda på olika platser. Hopkopplingen av delarna sker via ett nät, eller med en svengelsk terminologi ett nätverk (network). Men hopkopplingen kan vara olika omfattande, vilket gör att det totala systemet kan få olika arkitekturer.

En ytterlighet är när systemets delar är så hårt hopkopplade att systemet upplevs som en enda dator. Ett exempel är parallelldatorsystem, där man för att uppnå högsta möjliga prestanda sprider beräkningsdelar för parallell beräkning på de processorer som för tillfället råkar vara lediga. För att realisera detta krävs ett speciellt operativsystem, och ofta också specialskrivna applikationer.

Den andra ytterligheten är ett väldigt löst hopkopplat system, där kanske kommunikationen enbart sker via e-post. De olika delarna utgörs då av självständiga datorer av olika typ och med olika operativsystem. Det behövs ingen programvara utöver standard operativsystem och kommunikationsprogram.



Figur 1 Mellanskiktsprogram i OSI-modellen

Det vi här menar med distribuerat system är någon slags mellannivå. Mellan operativsystemet, av standardtyp, och applikationen finns mellanskiktsprogram (middleware, services, ….) som håller ihop systemet och som gör att användaren inte behöver hålla reda på var olika delar finns. Funktioner som hanteras av mellanskiktet kan vara - adressering av filer och andra objekt, översättning mellan internt format och standardformat, mm. Inte minst kan i mellanskiktet finnas olika säkerhetsfunktioner - nyckelhantering, inloggning, grupptillhörighet mm. Om man ritar in mellanskiktet i OSI-modellen, [1] sid 60-63, hamnar det i sessionslagret ungefär enligt figur 1.

En idag vanlig arkitektur på denna mellannivå är client/server arkitekturen. På klienten körs huvuddelen av applikationen. Då och då skickar klienten olika typer av begäran (requests), t ex hämta/lagra data, till en eller flera servrar. Servern skickar tillbaka ett svar eller en kvittens varefter klienten fortsätter sin avbrutna applikation. Ibland menar man också med client/server att det skall vara tillståndslöst, d v s att varje fråga/svar skall vara oberoende av andra frågor/svar. I så fall tillåter man t ex inte att man vid första fråga/svar utför speciella kontroller som man sedan litar på i fortsättningen.

Beroende på arkitekturen hos det distribuerade systemet finns olika många tjänster, d v s olika mycket funktionalitet, i mellanskiktet. Ett exempel på en löst kopplad arkitektur är WWW. I det enklaste fallet finns i stort sett bara http-protokollet, för att överföra data till/från en WWW-server, i mellanskiktet. För att erhålla bättre säkerhet kan flera services läggas i mellanskiktet, jfr TLS i avsnitt 3.2. Ett exempel på en relativt hårt kopplad arkitektur är DCE, Distributed Computing Environment, se 2.1. Här finns mängder med funktioner i mellanskiktet.

Figur 1 illustrerar också att mellanskiktet logiskt sett ligger mellan applikationerna och operativsystemet. Det finns alltså ett API, Application Program Interface, som talar om hur applikationerna anropar de olika funktionerna i mellanskiktet. Ju fler funktioner som finns desto mer komplicerat blir naturligtvis API. När en applikation skall anpassas till en distribuerad miljö måste den kompletteras med API-anrop. Om det är en komplicerad arkitektur, och särskilt om applikationen är skriven i ett ovanligt språk, kan detta innebära ett mycket stort arbete. Eftersom det finns ett antal alternativa arkitekturer, blir följden att de flesta applikationer inte har anpassats till alla arkitekturer.

Ju större och mer distribuerat ett system är desto större blir komplexiteten och därmed säkerhetsproblemen. Svåra problem är t ex stöd för auktorisering (vem eller vilka får göra vad och när) samt administration och distribution av kryptonycklar. Om sådana funktioner skall kunna hanteras centraliserat, vilket har många fördelar, krävs ett ganska hårt kopplat system. Å andra sidan kan löst kopplade system bli mer flexibla och robusta.

Nedan beskrivs kort några olika arkitekturer.

2.1 DCE

Distributed Computing Environment [2] är en öppen standard, framtagen inom OSF, Open Software Foundation. Som sådan är den först implementerad i Unixmiljö, följd av andra implementationer. Försvarsmaktens handbok IT, FM HIT [4], anger att distribuerade system (t ex ledningssystem) inom försvarsmakten skall baseras på DCE.

DCE är en mycket omfattande arkitektur. En del funktioner är väldigt operativsystemnära och borde kanske inte betraktas som mellanskiktsfunktioner. Ett exempel är trådning, vilket innebär att en applikation kan hantera flera uppgifter parallellt utan att begära nya processer från operativsystemet. Ett annat exempel, som DCE självt utnyttjar flitigt, är Remote Procedure Call. RPC innebär att man via API kan styra funktionsanrop till vilken dator som helst i systemet. RPC hanterar alla besvärliga problem som synkronisering, datakonvertering m m.

Tjänster (services) som mer naturligt finns i ett mellanskikt är tidstjänst, katalogtjänst, distribuerat filsystem och säkerhetstjänst. Tidstjänst innebär att klockorna kan synkroniseras i hela systemet. Katalogtjänst innebär en katalog med adresser för systemets olika resurser. Det distribuerade filsystemet (DFS) innebär att alla filer i hela systemet kan vara åtkomliga från vil-

ken dator som helst som om de vore lagrade lokalt på datorn. Säkerhetstjänsten tillhandahåller autenticering, nyckelhantering och viss kryptering. Den är baserad på Kerberos, se 4.1.1.

2.2 CORBA

Common Object Request Broker Architecture [3] är en uppsättning specifikationer där man gör en objektorienterad beskrivning av distribuerade system. Funktionaliteten hos CORBA påminner om den i DCE, men operativsystemnära funktioner, t ex trådning, finns inte med. I gengäld finns flera andra tjänster med i specifikationerna, som därför är omfattande. Detta i sin tur har som följd att enbart en del av specifikationerna har hunnit implementeras, vilket i sin tur kan leda till problem om man skall integrera delsystem som alla säger sig följa CORBA.

Idén med CORBA, vilket också framgår av namnet, är att applikationerna inte skall behöva bry sig om var i systemet olika objekt råkar finnas. Man skall inte anropa objekten direkt utan via CORBA, som fungerar som en objektmäklare (broker). Den objektorienterade ansatsen innebär att det inte finns något API med funktioner, som applikationerna skall anropa direkt. I stället anropas en metod hos ett objekt. Detta tas om hand av CORBA, som lokaliserar objektet och administrerar metodanropet. Detta avlastar applikationerna, som blir enklare att utveckla. Men mycket av arbetet är ju flyttat till CORBA, och om CORBA är ofullständigt eller felaktigt implementerat så får man tillbaka problemen. Det som definitivt måste vara implementerat är själva lokaliseringstjänsten, som kan sägas motsvara katalogtjänsten i DCE.

2.3 Distribuerade filsystem

Bland de första distribuerade arkitekturerna finns distribuerade filsystem. Dessa kan i viss mening sägas vara hårt kopplade. Ett tydligt behov i distribuerade system är att kunna läsa och skriva filer på olika platser i systemet. Det är då naturligt att sikta på att man som användare bara skall se ett enda, hierarkiskt, filsystem där filåtkomst sker på ett enda sätt, oberoende av om filen finns lokalt på den egna datorn eller om den finns någon annanstans. Det är också så flera distribuerade filsystem är uppbyggda.

De första distribuerade filsystemen knöts till ett visst operativsystem. Exempel är NFS [16] för Unix och SMB-protokollet [5] för fildelning (shares) under Windows. Därför finns inte mycket implementerat i vad som ovan kallades mellanskiktet. Grundläggande funktioner för

läsning och låsning av filer etc är en del av operativsystemet. Andra funktioner, t ex strukturering av filträdet och åtkomstskydd av filer, görs i stor utsträckning manuellt.

Så småningom har funktioner tillkommit i mellanskiktet. Exempel är DFS i DCE, SAMBA i Unix för hantering av Windows-filer och motsvarande för NFS i Windows. Likaså finns tillägg, t ex för hantering av åtkomstskydd, i mellanskiktet.

2.4 WWW

Distribuerade filsystem används mest i lokala nät. Visserligen kan man, t ex via Internet, lätt koppla ihop datorer även på långt avstånd, men den manuella administrationen medför att antalet datorer måste bli begränsat. För att få åtkomst till data på alla jordens datorer kan man inte använda ett distribuerat filsystem. För detta har sedan länge utvecklats speciella applikationer, t ex ftp, File Transfer Protocol. Användning av ftp innebär en hel del manuella handgrepp. Man måste leta reda på den server där den önskade filen finns, man måste leta i serverns filsystem, man måste bestämma vad som skall hända med filen när den överförts mm.

Den explosionsartade utvecklingen av Internet, till ett verktyg för gemene man, kommer sig av att man i slutet av 80-talet insåg att det fanns ett behov av att kunna överföra och presentera innehållet i filer utan manuella handgrepp, bara med ett musklick. WWW, World Wide Web, utvecklades i början av 90-talet och blev allmänt i mitten av 90-talet. Succén för WWW kan sägas bygga på två standarder. HTTP, HyperText Transmission Protocol [6], för hantering av själva filöverföringen, det kan sägas vara en förenklad ftp. HTTP ligger på nivån över TCP/IP (Internets transportprotokoll) och finns alltså på mellanskiktet i figur 1. HTTP kan användas av vilken applikation som helst för filöverföring. Den vanligaste applikationen är bläddraren (Netscape Navigator, Internet Explorer, Opera, ….) som presenterar data i fönster på skärmen. Bläddraren följer den andra viktiga WWW-standarden HTML, HyperText Markup Language [7]. HTML beskriver framför allt datas layout, men även t ex om data skall processas på något sätt. HTML är implementerat direkt i bläddraren och ligger alltså inte i mellanskiktet.

WWW är i sitt grundutförande ett löst kopplat distribuerat system. Mycket av succén är en följd av att det är löst kopplat, d v s att man bara standardiserat ett litet antal funktioner, men dessa finns implementerade överallt. När väl denna grund fått allmän acceptans diskuterar

man sig stegvis fram till förbättringar och utbyggnader, jfr evolutionär systemutveckling. Läget nu är att HTML håller på att utvecklas. Man har insett att det finns behov att standardisera mycket annat än presentationsformat hos WWW-dokument. Målet är en standard, XML, Extended Markup Language [8], för allmän dokumentbeskrivning. XML kommer troligen att ligga som en fristående modul i mellanskiktet. När det gäller HTTP är utvecklingen lite annorlunda. En del föreslagna utvidgningar har inte blivit allmänt införda. En angelägen utvidgning gäller säkerheten, som är bristfällig i HTTP. De facto standard är en modul i mellanskiktet, TLS, fristående från HTTP, se avsnitt 3.2. TLS + HTTP tillsammans utgör HTTPS-protokollet. Det anropas från en WWW-bläddrare genom att WWW-adressen (URL) inleds med https:.

3. Säkerhetsfunktioner på olika nivåer

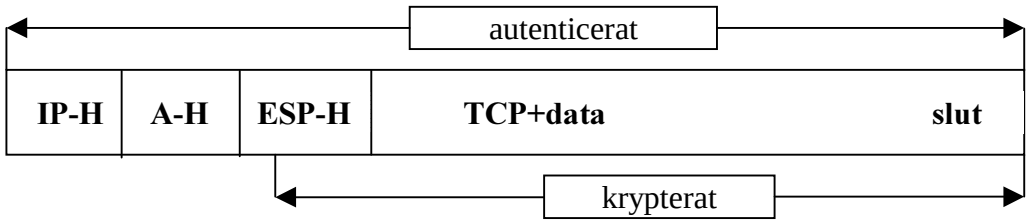
I föregående avsnitt beskrevs hur olika funktioner kan ligga på olika nivåer i OSI-modellen, beroende på arkitekturen hos det distribuerade systemet. Säkerhetsfunktioner kan alltså finnas på olika nivåer, med olika för- och nackdelar. Nedan exemplifieras säkerhetsfunktioner på olika nivåer. Gemensamt för dem alla är att metoderna och algoritmerna för autenticering och kryptering är väl genomtänkta och standardiserade. Däremot finns det oklarheter när det gäller andra viktiga aspekter, t ex nyckel- och certifikathantering.

3.1 IPSEC

På nätnivån bland Internetprotokollen finns IP-protokollet. Detta hanterar datapaketen, ett efter ett, helt fristående från varandra. På IP-nivån bryr man sig inte om ifall paket hör ihop, t ex hör till samma TCP-session. Man tittar bara i IP-headern, där det står att paketet kommer från den ursprunglige avsändaren A, och att det skall slussas till den slutlige mottagaren B. Sett från IP-nivån består ett IP-paket av två delar, IP-header av fix längd följt av data av variabel längd. Dessa data består i sin tur av paket, t ex TCP-header följd av TCP-data. Resultatet blir det välkända "paket inuti paket inuti paket ….".

IPSEC [9] innebär att man mellan IP-nivå och TCP-nivå lägger en eller två ytterligare headers, A-H authentication header och ESP-H Encapsulating Security Payload header. A-H talar om på vilket sätt paketet skall autenticeras, d v s enligt vilken metod man verifierar att

det är äkta. ESP-H talar om på vilket sätt paketet är krypterat. Detta görs i båda fallen genom ett heltal (SPI, security parameters index), som anger en kombination av algoritm, nyckel, m m. De två kommunicerande parterna, A och B, måste alltså först på något sätt komma överens om vad olika värden på SPI skall innebära. Hur detta skall göras ingår inte i standarden IPSEC. Sedan utförs de kryptologiska operationerna för autenticering och/eller kryptering på IP-nivå hos avsändaren A. Paketen slussas oförändrade genom nätet och de omvända operationerna görs på IP-nivå hos mottagaren B. Ett paket kan åskådliggöras enligt figur 2.



Figur 2 IP-paket med IPSEC-headers

IP finns i två versioner. Den äldre version 4, IPv4, har varit i drift många år på Internet. Den håller så sakteliga på att ersättas av version 6, IPv6, med bl a längre header. IPSEC kan användas i båda versionerna. Dock är IPSEC ett frivilligt tillägg i IPv4 medan IPv6 måste kunna hantera minst en autenticeringsalgoritm (MD5) och minst en krypteringsalgoritm (DES-CBC).

IPSEC möjliggör att alla data, från alla applikationer, som sänds från A till B kan autenticeras och/eller krypteras utan någon åtgärd från applikationen, vilket är en stor fördel. Nackdelen är att allt måste vara väl definierat och standardiserat. Enligt ovan måste, för att kunna bilda SPI, nyckeldistributionen vara implementerad på ett säkert och standardiserat sätt. Detta är en av de besvärligaste områdena inom kryptofältet. Den vanligaste användningen av IPSEC har blivit vid uppbyggnad av VPN, Virtual Private Network. Genom att kryptera all trafik mellan egna noder, som ingår i ett publikt större nät, får man ett avgränsat privat nät. Det finns också andra sätt än IPSEC att realisera VPN.

3.2 TLS, Transport Layer Security

En nivå högre upp i OSI-modellen finns transportnivån. Man vill där autenticera och/eller kryptera en hel TCP-session. Internetstandarden heter nu TLS [10]. Den lanserades ursprungligen under namnet SSL, Secure Socket Layer.

TLS är uppdelat i två delar. En handskakningsdel där de två applikationerna, A och B, skall autenticera varann och komma överens om en kryptonyckel, som skall användas under resten av sessionen. Observera att A och B nu inte är sändande/mottagande dator, utan två applikationer. Även om det är samma två datorer inblandade öppnar alltså olika applikationer olika TLS-sessioner, med bl a olika krypteringar. Den andra delen av TLS-protokollet är själva datahanteringen. Data bl a komprimeras och krypteras innan de skickas till TCP-lagret.

Handskakningen består av upp till 12 stycken meddelanden som skickas mellan de två applikationerna. De första meddelandena innehåller listor över vilka krypterings- och autenticeringsmetoder och vilka typer av certifikat mm, som respektive applikation har implementerat. Om endera applikation kräver autenticering, vilket alltså är valbart, skickar den andra applikationen över listor med certifikat. Några meddelanden går åt för att bygga upp den krypteringsnyckel, känd bara av de två applikationerna, som skall användas under resten av TLS-sessionen. De sista meddelandena är autenticering av de tidigare meddelandena, för att verifiera att dessa var äkta. Efter denna handskakning krypteras resten av TLS-sessionen. En och samma TLS-session kan innehålla flera TCP-sessioner. Det finns därför specificerat hur man förlänger en TLS-session utan förnyad handskakning.

Som nämnts är det ett problem att delar av nyckelhantering, certifikathantering etc, inte är fullständigt standardiserat. En vanlig användning av TLS är därför att bara använda kryptering, utan autenticering, t ex för kreditkortsdata eller andra ekonomiska transaktioner på Internet.

3.3 S/MIME och PGP

Den högsta OSI-nivån är applikationsnivån. Även här kan man lägga säkerhetsfunktioner. Två exempel är S/MIME och PGP. I båda fallen kan man beskriva det som så att det finns funktioner, analogt med mellanskiktsnivån, som applikationen anropar med data - meddelande, dokument, fil eller annan typ av data. Funktionerna transformerar data, t ex krypterar eller

signerar, och formaterar resultatet enligt specificerad standard. Resultatet skickas tillbaka till den anropande applikationen som själv skickar det till transportlagret i OSI-modellen.

S/MIME, Secure/Multipurpose Internet Mail Extensions [11], är en utvidgning av MIME. Denna är framtagen som standard för formattering av e-post, men kan lika väl användas som standard för andra dokument. I MIME standardiseras t ex hur bild eller ljud skall läggas in i ett dokument. Tilläggen i S/MIME beskriver krypterade data, digitala signaturer, certifikat mm.

PGP, Pretty Good Privacy [12], har blivit de facto standard för e-post på Internet, men kan också användas för andra datafiler. Det speciella med PGP är att man inte använder hierarkier av certifikat, d v s man förutsätter ingen betrodd tredje part. I stället signerar användare var-andras öppna nycklar efterhand som det behövs och man knyter ihop sändare och mottagare med en kedja av signaturer (om man har tur). Man bygger en "web of trust". Det ursprungliga PGP har en egen standardisering av hur meddelanden skall se ut, av kryptering och komprimering etc. I senare versioner har man närmat sig MIME-standard. Man håller dock fast vid "web of trust" i stället för hierarkier av certifikat.

3.4 Servrar (betjänter)

De säkerhetsfunktioner som beskrivits ovan, i 3.1-3.3, har karakteriserats av vilken nivå i OSI-modellen som funktionen finns på, t ex på vilken nivå krypteringen sker. Men OSI-modellen ger bara en struktur över hur det ser ut inuti en enskild dator, inte över hela det distribuerade systemet. Ett annat sätt att beskriva strukturen, i princip frikopplat från OSI-modellen, är att beskriva var i det totala, distribuerade, systemet som en säkerhetsfunktion utförs. Sker den utspritt, eller är den koncentrerad till en eller ett fåtal ställen i systemet? I det senare fallet brukar man tala om säkerhetsservrar.

En fördel med servrar är att man då koncentrerar säkerhetskritiska funktioner och data till ett ställe. Nackdelar är att man inför en sårbar punkt, att kommunikationen med servern är säkerhetskritisk och den kan bli komplicerad. Dessutom måste tillämpningarna anpassas till att använda servern. För att slippa göra om samma sak flera gånger, när olika tillämpningar anpassas till olika servrar, vill man naturligtvis standardisera anpassningen. Detta är dock lättare

sagt än gjort. Den standard som finns har namnet GSS-API, Generic Security Service Application Program Interface [13].

Funktioner som man vill koncentrera till servrar är t ex
- autenticering, i betydelsen identitetsverifiering (Single Sign On)
- hantering av roller - en individ skall agera i olika roller med olika behörigheter
- nyckelhantering av öppna och/eller hemliga kryptonycklar
- auktorisering - vem, eller vilken roll, får göra vad och när får man göra det
- systemadministration, databaser, filsystem etc

I arkitekturen DCE (avsnitt 2.1) ingår servrar. Autenticering och nyckelhantering baseras på Kerberos, se 4.1 och A.1. Där beskrivs också SESAME, som liknar Kerberos men som också omfattar rollhantering.

4. Kommentarer till appendix

Doktorandkursen "Applied Network Security" följdes 1999 av fem personer från FOA. Kursen avslutades med projektarbeten. FOA deltog i två projekt - en jämförelse mellan säkerhetsarkitekturerna Kerberos och SESAME samt en prototyp för säker anbudshantering via WWW. Båda dessa projekt har försvarsintresse. Kerberos ingår i DCE, som i dagsläget är påbjudet som distribuerat system inom försvarsmakten [4], och i framtiden kommer man att vilja basera många system på WWW. De två projektrapporterna, på engelska, finns som appendix. Här, i avsnitt 4, ges en kort bakgrund och sammanfattning.

4.1 Kerberos och SESAME

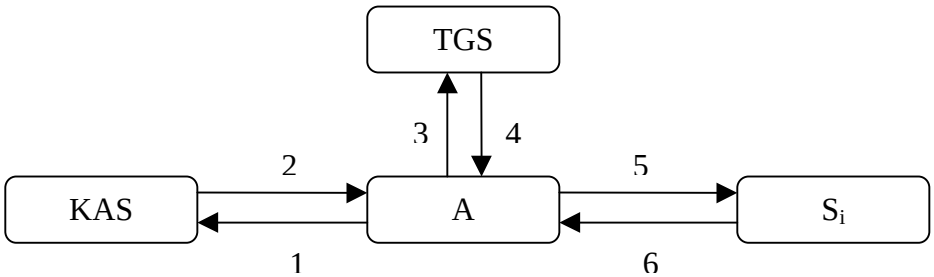
4.1.1 Kerberos

Kerberos [14] är en säkerhetsarkitektur för distribuerade system som utvecklades på MIT på 1980-talet. Man hade framför allt två mål med Kerberos.

1. Användaren skulle bara behöva autenticera, i betydelsen bevisa sin identitet, en gång per session. Autenticeringen bygger på lösenord och det är helt förkastligt att avkrävas flera olika lösenord efter hand som man anropar olika applikationer. Eftersom lösenordet blir säkerhetskritiskt får det inte sändas över nätet ens i krypterad form.

2. Efter det att en användare autenticerat sig, enligt 1, skall inte det kritiska lösenordet användas. Autenticeringen mellan användare-applikation skall i stället ske via nycklar med kort varaktighet. Dessa nycklar överförs till applikationen i ett krypterat meddelande som man kallar "ticket", d v s biljett.

Kerberos kan logiskt beskrivas med följande figur 3.



Figur 3 Logisk uppbyggnad av Kerberos

De ömsesidiga autenticeringarna, mellan användaren A och någon annan ruta i figur 3, sker via symmetriska nycklar, d v s båda parter måste ha en gemensam hemlighet. Kerberos Authentication Server (KAS) och Ticket Granting Server (TGS) är logiskt skilda enheter men är i praktiken ofta implementerade på samma ställe. Användaren A skall använda olika applikationer S_i. De meddelanden 1-6 som skickas är symmetriskt krypterade och uppbyggda på ett ganska komplicerat sätt, för detaljer se []. I huvudsak händer följande i de olika stegen.

1. A och KAS har den gemensamma nyckeln K_{KA}, som bildas som envägsfunktion ur användarens lösenord (jfr lösenord i Unix). När A vill börja använda systemet (inloggning, autenticering) skickar han meddelande 1, som i princip lyder "jag är A, klockan är "tidpunkt krypterad med K_{KA}"".

2. KAS dekrypterar tidpunkten i 1 och om den verkar stämma verkar det troligt att det verkligen är A och meddelandet 2 bildas. KAS och TGS har den gemensamma nyckeln K_{KT}. KAS bildar en nyckel K_{AT}, som A och TGS skall använda under sessionen (alltså en korttidsnyckel). Meddelande 2 består av två delar. En del är krypterad med K_{KA} och innehåller bl a nyckeln K_{AT}. Det är detta som är själva autenticeringen. Om inte A känner K_{KA}, d v s "falsk A", får inte A kännedom om K_{AT} och stegen 3-6 nedan kan inte genomlöpas. Den

andra delen av 2 kallas "ticket granting ticket". Denna är krypterad med K_{KT} och lyder i princip "TGS och A skall mellan tid1 och tid2 använda nyckeln K_{AT} ".

3. Stegen 3-6 skall genomlöpas en gång för varje applikation S_i som A vill använda. Meddelande 3 består av två delar. Dels biljetten "ticket granting ticket" från 2. Ur denna kan TGS (och ingen annan) få fram K_{AT}. Dels en del som i princip lyder "jag vill använda applikation S_i "tidpunkt krypterad med K_{AT} "". Denna del motsvarar i princip meddelandet 1 med KAS ersatt av TGS.

4. Likaså motsvarar detta meddelande 2. Det består dels av delen "ticket for S_i ", som är krypterad med en nyckel K_{TSi} (gemensam för TGS och S_i), och som innehåller en sessionsnyckel K_{ASi} att användas mellan A och S_i under en viss tid. Den andra delen av 4 är just K_{ASi} krypterad med K_{AT}. I detta steg skulle också en centraliserad behörighetskontroll kunna utföras. TGS kan bedöma om A är behörig att använda S_i. Detta ingår dock inte i Kerberos. Behörighetskontrollen sker lokalt i varje applikation S_i.

5. A dekrypterar K_{ASi}. Meddelandet 5 består av två delar. En del, krypterad med K_{Asi}, som i princip lyder "jag är A, klockan är yy:mm:dd:hh:mm". Den andra delen är "ticket for S_i " enligt ovan (som alltså är krypterad med K_{TSi}).

6. S_i dekrypterar "ticket for S_i " och använder K_{ASi} att kryptera en kvittens "jag är S_i, klockan är yy:mm:dd:hh:mm".

Härmed har A och S_i ömsesidigt autenticerat varandra. De kan använda K_{ASi} under resten av sessionen till att kryptera trafik sig emellan.

Den största bristen i Kerberos är att allt bygger på symmetriska nycklar. Det är ett klassiskt, svårt problem att hemlighålla, ändra och allmänt hantera många parvisa hemligheter i ett stort system. KAS måste ju dela nycklar med alla användare A och TGS måste dela med alla applikationer S_i. I ett verkligt system vill man ofta ha mer än en KAS/TGS, vilket ytterligare komplicerar nyckelhanteringen. En annan följd av symmetriskt system är att alla hemligheter skapas på ett ställe, KAS/TGS. Om detta röjs eller blockeras rasar hela systemet. Det finns ett antal förslag till modifiering av Kerberos till autenticering med öppna nycklar.

Ett annat problem, som inte är unikt för Kerberos, är att samtliga applikationer måste anpassas. Detta kan bli mer komplext än man i förstone inser. Den tidigare nämnda standarden, GSS-API, innehåller ca 40 funktionsanrop.

4.1.2 SESAME

SESAME [15] är ett europeiskt (ICL, Bull och Siemens) initiativ till "ett förbättrat Kerberos". Dels kunde inte, p g a amerikanska exportrestriktioner, Kerberos fritt användas utanför USA. Dels ville man införa förbättringar, t ex användning av öppen-nyckel algoritmer. Eftersom Kerberos blivit de facto standard i USA skall SESAME också hantera symmetriska nycklar kompatibelt med Kerberos.

Själva grundstrukturen är samma som i Kerberos, det finns Security Servers som utfärdar biljetter, tickets, som "öppnar dörren" till följande steg i en kedja à la 1-6 i Kerberos. Förutom KAS och TGS (som i SESAME har andra benämningar) finns en tredje typ av server, PAS, Privilege Attribute Server. Detta är den centraliserade behörighetskontroll som nämndes i steg 4 i 4.1.1. Men ambitionen är ännu högre än att kontrollera om en användare är behörig att anropa en viss applikation. Man inför även rollhantering. En användare skall under en session kunna växla mellan olika roller, var och en med olika behörigheter. PAS ger ifrån sig certifikat (med viss analogi till biljett) som kallas PAC, Privilege Attribute Certificate.

Kravet på kompatibilitet med Kerberos, plus utvidgning med öppen-nyckel metoder, plus rollbaserad behörighetskontroll, har resulterat i ett system, som är ytterligare en storleksordning mer komplext än Kerberos. GSS-API blir därför ännu mer omfattande och det är inte så många applikationer som anpassats till SESAME. Det är tveksamt om SESAME kan nå upp till positionen som en allmänt använd arkitektur.

4.1.3 Jämförelse Kerberos/SESAME

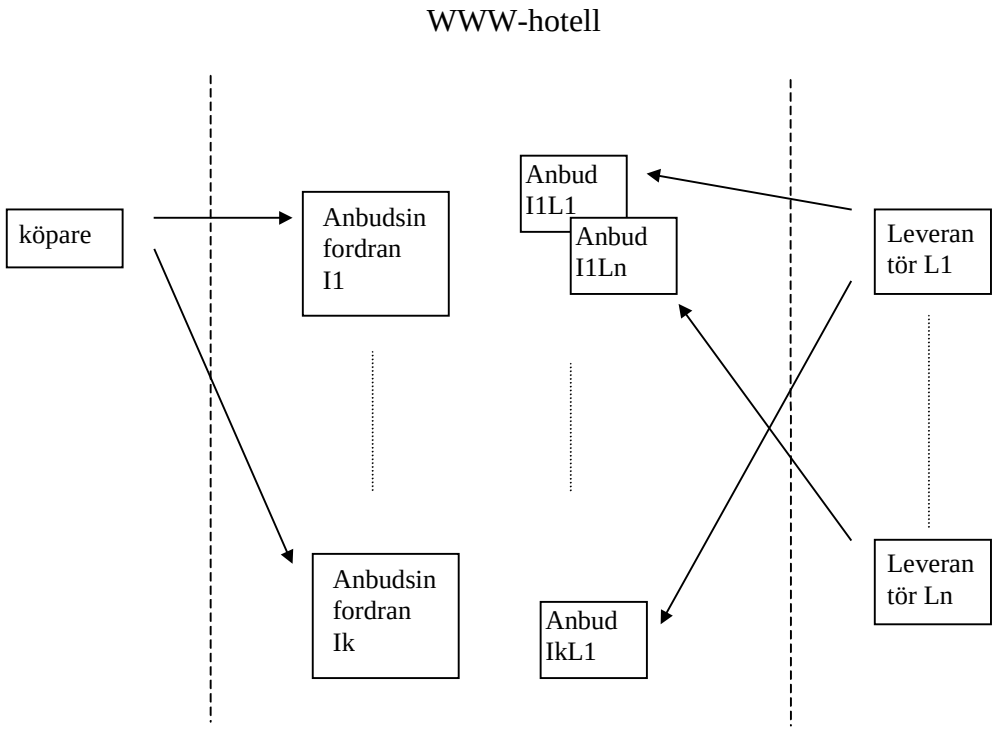
I doktorandkursen "Applied Network Security" ingick som projektarbete en jämförelse av Kerberos och SESAME. Först och främst skulle gratisversioner av de två systemen installeras. Sedan skulle kompatibiliteten dem emellan undersökas. T ex skulle vi undersöka om man efter användarautenticeringen (d v s steg 2 i 4.1.1) i det ena systemet kunde övergå till det andra systemet.

I A.1 finns den engelska projektrapporten. Det gick att installera systemen, dock med ett flertal problem vad gäller SESAME. Det gick tyvärr inte att få dem installerade tillräckligt väl för att kompatibiliteten skulle kunna testas.

4.2 WWW-säkerhet, fallstudie.

Den totala tillgängligheten av WWW, se 2.4, gör att det finns, och än mer kommer att finnas, ett stort intresse av att använda WWW för olika distribuerade system. Frågan blir då om säkerhetskraven kan lösas. Säkerheten i det grundläggande protokollet, HTTP, har många brister. Den de facto standard som finns för bättre säkerhet är TLS, se 3.2.

I kursen "Applied Network Security" ingick som fallstudie att utnyttja WWW för anbudsinfordran. En köpare skall lägga ut anbudsinfordran på en allmänt tillgänglig plats, t ex ett WWW-hotell. Ett stort antal leverantörer skall obehindrat kunna läsa anbudsinfordran och själva kunna besvara dem med anbud. Figur 4 är en skiss (pilar som anger läsrättigheter är ej utritade) av fallet. Av olika anledningar gjordes implementationen enligt appendix A.2 inte på ett WWW-hotell utan på en server hos köparen.



Figur 4 Anbud via WWW

Ett flertal krav kan formuleras:

1. Det är enbart köparen som får lägga ut en anbudsinfordran. Alternativt att infordran innehåller en digital signatur. Annars kan t ex en leverantör ändra i texten i anbudsinfordran.

2. Vilken leverantör som helst skall kunna lägga upp anbud. Anbud får tas bort, eller ändras, enbart av författande leverantör.

3. Anbud måste otvetydigt knytas till rätt leverantör. Annars kan leverantörer lägga upp falska anbud i varandras namn.

4. Anbudsinfordran och anbud skall inte kunna förnekas av respektive författare.

5. Anbud skall bara kunna läsas av köpare samt av den leverantör som författat anbudet.

Det finns fler liknande krav man kan ställa. Dessutom kan man kräva tidsstämpling, loggning m m.

Kraven 1-5 är klassiska krav på sekretess, autenticering, behörighetskontroll och signering. Kraven 1, 2 och 5 kräver autenticering följd av behörighetskontroll. Kraven 1, 3 och 4 kräver digitala signaturer. Kravet 5 är krav på sekretess. Dessa krav löses inte i standard WWW. Digitala signaturer finns inte, inte heller sekretess (kryptering). Autenticering finns bara i en mycket rudimentär form. Man kan avkräva lösenord, som sedan skickas i klartext över nätet. Åtkomstkontroll finns bara baserat på sändande dators IP-adress, alternativt lösenord. Både lösenordshantering och åtkomstkontroll är ytterligt tungadministrerade.

TLS löser en del krav under själva överföringen via Internet. Man får kryptering, vilket delvis tillgodoser krav 5. Man får stark autenticering, via certifikat, vilket gör att WWW-hotellet i figur 4 ovan kan vara säker på vilken individ som fanns i andra änden under överföringen.

Behörighetskontrollen löses inte av TLS. Men den förbättras genom att vissa webservrar (i vårt fall Apache) kan utnyttja TLS-autenticeringen i sin behörighetskontroll. Men det är tungadministrerat, i vart fall i Apache.

Digitala signaturer måste skapas, respektive verifieras, av fristående program. Om man använder nyare version av Netscapes bläddrare, Navigator, finns möjlighet att använda samma databas med nycklar och certifikat som TLS använder vid överföringen. Man behöver alltså inte administrera två databaser, vilket underlättar. Men databasen är inte standardiserad och tillgänglig för andra applikationer. Digital signering kan göras automatiskt av Navigator när man fyllt i ett formulär. Vid verifiering av en signatur måste man dock först spara undan dokumentet och sedan verifiera manuellt.

I appendix A.2 beskrivs hur vi löste fallstudien via förbättrad WWW-säkerhet. Slutsatsen är att man får påtagligt bättre säkerhet, men brister kvarstår. De största bristerna är tungarbetad behörighetskontroll och signaturverifiering, samt att inte databasen för certifikat och nycklar är standardiserad och allmänt tillgänglig. En allvarlig brist är att den privata nyckeln, som är grunden till autenticeringen, är bristfälligt skyddad. I vår lösning är nyckeln krypterad av Netscape Navigator. Man måste ange ett lösenord till Navigator för att kunna dekryptera den privata nyckeln, vilket är ett dåligt skydd. Den privata nyckeln bör lagras på en plats med bättre skydd, t ex i ett aktivt kort.

Referenser

1. Säkerhetsarkitekturer, Dataföreningen i Sverige, SIG Security, 1998

2. D. Hartman, "Unclogging Distributed Computing, IEEE Spectrum, 29(5), sid 36-39, maj -92

3. The Object Management Group, http://www.omg.org/

4. FM HIT 97, Försvarsmaktens handbok för informationsteknik, Försvarets bok- och blankettförråd, -97

5. What is SMB?, http://anu.samba.org/cifs/docs/what-is-smb.html

6. HTTP-charter, http://www.ietf.org/html.charters/http-charter.html

7. HTML 2.0 Materials, http://www.w3.org/MarkUp/html-spec/

8. Extensible Markup Language (XML), http://www.w3.org/XML/

9. S. Kent, R. Atkinson, Internet Drafts, RFC 1825-1827, augusti -95

10. "The TLS Protocol Version 1.0", RFC 2246, januari -99

11. Dussé et al, "S/MIME version 2 Message Specification", RFC 2311, mars -98

12. The International PGP Home, http://www.pgpi.com/

13. J. Linn, "Generic Security Service Application Program Interface Version 2", RFC 2078, januari -97

14. B.C. Neuman och T. Ts´o, "Kerberos: An Authentication Service for Computer Networks", IEEE Comminications Magazine, vol 32, sid 33-38, september -94

15. P. Kaijser, T. Parker och D. Pinkas, "SESAME: The Solution to Security for Open Distributed Systems", Computer Communications, 17(7), sid 501-518, juli -94

16. White Paper: The NFS Distributed File Service, http://www.sun.com/software/white-papers/wp-nfs/nfs_12.html

Appendices

A.1 Kerberos and SESAME

A.2 A Secure Electronic Tender System