IP-baserade program

- Det här kapitlet behandlar några klassiska TCP/IP-baserade program. Främsta fokus är HTTP men även lite enklare applikationer som telnet och FTP behandlas.
- Kapitlet är tänkt att kunna läsas fristående men fungerar bäst om du vet hur TCP/IP fungerar.

Så har vi kommit fram till programmen. Själva vitsen med TCP/IP är ju att alla typer av program ska kunna kommunicera med varandra oavsett hur kanalen eller länken ser ut. Vi har tidigare konstaterat att vi förväntar oss att visst stöd och vissa program ska ingå i Internetprotokollen. Ofta har program och protokoll blivit synonymer.

Det finns ett par program som i princip inte dokumenterats. Hit hör traceroute och ping, hur de fungerar behandlas bland annat i kapitlet Felsökning i TCP/IP-miljö. Traceroute används av tusentals användare i hundratals länder. Trots det finns det väldigt lite skrivet om hur programmet kan och borde fungera, och de beskrivningar som finns tillkom efteråt. Programmen och kommandona är så gamla så det kommer från en tid då man i Unix hade den informella principen att ”all dokumentation finns i källkoden”. Ett program blev de facto standard och tillslut ett eget protokoll.

Telnet

Att kunna logga in på en annan dator via en fjärranslutning är en gammal idé, det var bland det första man ville göra med Arpanet. I en grafisk miljö brukar man prata om remote desktop, i textbaserade system om remote login. Telnet hanterar textbaserade system.

Principen kan verka enkel: det som skrivs på den ena datorns tangentbord ska utgöra input på andra sidan. Det som utgör output på en sida ska visas på en textbaserad konsol på andra sidan. Men det är svårt och behandlas i en mängd RFC:er som 854, 859–861 och många fler (och tyvärr behandlar varje RFC bara delar av protokollet). Svårigheterna har med en mängd optioner att göra, att applikationen ligger nära operativsystemet och hårdvaran (en miljö kanske använder knappt 50 tangenter och en annan runt 75)

men även med principen att göra om enstaka tangentnedtryckningar till paket på ett effektivt sätt. Fortfarande är textbaserade system vanliga, de kan göras väldigt inmatningseffektiva, den datamodell som de arbetar med är väl beprövad och de är lätta att nå: i stort sett vilken dator som helst med TCP/IP kan vara en telnet-terminal. I Linux är tangent och konsol fortfarande standard input respektive output. I Windows finns stöd för telnet i konsolläge och i programmet Hyperterminal.

Liksom många andra applikationer bygger telnet och FTP på TCP.

Telnet bygger på TCP, vi vill ju att dataöverföringen ska vara tillförlitlig och visa samma data i bägge ändar. Telnet använder ofta push-funktionen i TCP för att ge en snabbare användarupplevelse.

För att förhandla fram optioner och inställningar som 7- eller 8-bitars ASCII, om CR, LF eller CR+LF ska användas etc så används ett begrepp vid namn NVT, Network Virtual Terminal. Både klient och server tar omvägen över NVT, det gör att man enklare kan anpassa ett telnet-program för varje maskins egenheter.

Till telnet finns en mängd optioner. Både klient- och serversidan kan meddela att de avser att använda en option och den andra ändan får bekräfta eller stoppa. Några exempel på optioner är ECHO (tillåta lokalt eko) och end-of-record. NVT ser till att bägge ändar kan kommunicera överhuvudtaget.

Telnet kan användas på andra portar än nummer 23. Detta är praktiskt vid felsökning med mera.

Telnet ställer inga krav på att det verkligen är ett tangentbord eller en konsol som skickar eller tar emot data. Detta gör att telnet kan användas för att testa förbindelser och protokoll. Det vi skickar från ena sidan kommer fram till den andra. Genom att använda telnet på andra portar än port 23 kan telnet användas vid felsökning. Vi kan till exempel telnetta till en webbserver på port 80 med kommandot ”telnet www.firma.se 80”. Sedan skickar vi en slumpartad textsträng och trycker Enter två gånger. Webbserver kommer då svara med att vårt HTTP-anrop var fel. Men då ser vi att webbservern svarar. Om vi kan syntaxen för HTTP kan vi dessutom skicka korrekta kommandon till servern och se hur den svarar. Samma sak gäller bland annat protokollen SMTP, POP3 och FTP.

Telnet har dålig säkerhet. Kommunikationen sker i klartext vilket gör att användaridentitet och lösenord kan avlyssnas enkelt. Detta gör att användningen av telnet bör begränsas. En följd av detta är också at flera Unix-varianter idag inte tillåter att man loggar in sig med root-behörighet via telnet.

Att all kommunikation sker i klartext, även felhantering och intern förhandling mellan klient och server, är säkerhetsmässigt en nackdel. Men för felsökning är det en fördel. Man kan enkelt koppla in en analysator och se var problemen uppstår. Detta är en egenskap som flera TCP/IP-baserade program delar och ett skäl till

att Internetprotokollen blivit populära. Felhanteringen är dessutom relativt utförlig, vid förhandlingar om optioner inom applikationer sker mycket i klartext och inte bit-orienterat (via flaggor). Om den ena sidan inte förstår skickar de ofta meddelanden med text ”the client sent a message the server can’t understand” eller något liknande.

Felmeddelanden och intern kommunikation i klartext gör felsökning enkel men är en säkerhetsrisk.

Secure Shell – SSH

Säkerheten i telnet är låg och inloggningssekvenser skickas i klartext. Eftersom det finns ett behov av fjärrterminaler har konceptet utvecklats vidare till bland annat SSH, Secure SHell. SSH har även blivit populärt genom en lättanvänd shareware vid namn PuTTY.

SSH ger möjlighet till autentisering via PKI-teknik. Med hjälp av serverns publika nyckel och ett fingeravtryck kan klienten autentisera servern. Den publika nyckeln kan sedan användas för att upprätta krypterad kommunikation mellan klient och server. På samma sätt som det går till i HTTPS förhandlar sedan klient och server fram ett symmetriskt krypto och en sessionsnyckel för att spara CPU-belastning.

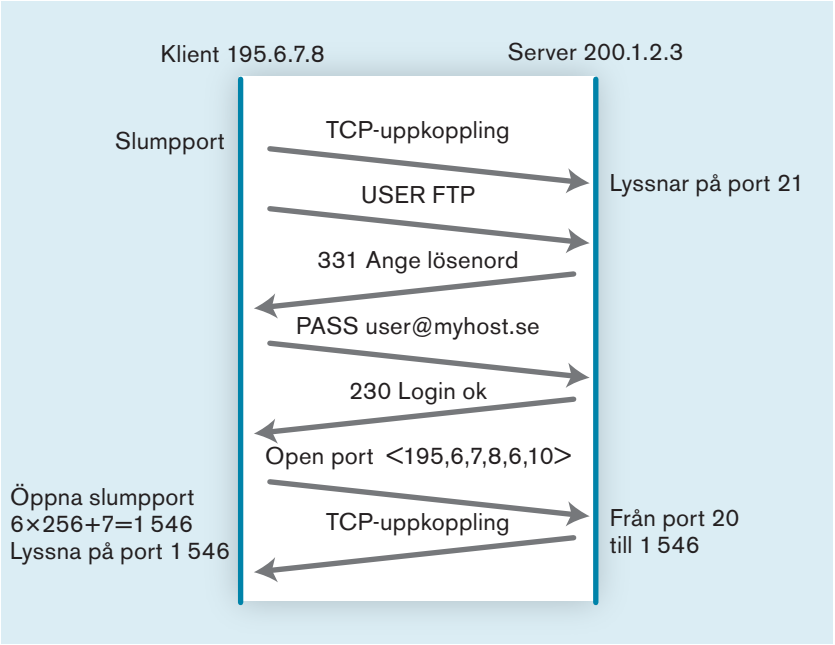
En funktion till som gjort SSH populärt är en funktion för port forwarding. SSH kan konfigureras så att en godtycklig applikation och port kan använda SSH, och SSH kan således skapa en krypterad tunnel mellan två noder.

File Transfer Protocol – FTP

Program för att överföra filer tillhör också de äldsta tillämpningarna. De äldsta RFC:erna är från tiden innan TCP/IP. Klassisk FTP fungerar fortfarande på ett sätt som inte passar när brandväggar ska konfigureras, se nedan. Och precis som för Telnet är säkerheten svag, såväl kontrollinformation som inloggningar skickas i klartext.

FTP använder två portnummer. På port 21 skickas kontrollinformation för inloggning med mera. När sedan data ska överföras (vilket även omfattar en listning av vilka filer som finns att hämta) så startas dotterprocesser på servern och de använder port 20.

Det som är lite udda i klassisk FTP är att dataöverföringen dels startar från servern sessionsmässigt (ett problem då brandväggar brukar acceptera inkommande SYN+ACK-paket men inte SYN-paket), dels att det då är servern som har ett känt portnummer. Med FTP i passivt läge, även kallat PASV, så löser man detta problem och PASV är numer oftast förvalt flera klienter. Ett klassiskt



Klassisk FTP använder två portnummer och dataöverföring sker via en session som startas från servern.

undantag har varit FTP i MS Windows kommandoläge. FTP och speciellt passivt läge behandlas även i kapitlet om adressöversättning.

FTP använder en delmängd av protokollet telnet för att skicka kommandon och textsträngar. Även NVM används på ett sätt liknande telnet, men jämfört med telnet används bara ett fåtal optioner. På samma sätt som med telnet påverkar detta säkerheten negativt men det underlättar för felhantering.

Det finns varianter på filöverföring med högre säkerhet, där till exempel Kerberos används för inloggning eller SSL (Secure Socket Layer) används för att skydda såväl inloggning som överföring. En annan konsekvens av FTP:s svaga säkerhet är att FTP ofta används för anonym FTP. En användare loggar in med användarid ”anonymous” eller ”ftp” och anger sedan sin e-postadress som lösenord (av artighetsskäl så att servern kan logga hur servern används). Till stor del har webbservrar tagit över denna funktion men en del leverantörer väljer fortfarande att visa olika information på webb- respektive FTP-servrar.

En förenklad variant av FTP har utvecklats i form av Trivial FTP, TFTP. TFTP blev snabbt en succé och används flitigt då till exempel konfigurationsfiler ska överföras eller då system ska boota upp. TFTP utmärks av:

TFTP använder UDP och är ett väldigt enkelt protokoll för att överföra (huvudsakligen små) filer.

- ingen inloggning eller autentisering
- ingen möjlighet till kryptering
- UDP används som nivå 4-protokoll. Detta sparar både minne

och kod. För att kontrollera att data kom fram används ett enkelt stop-and-wait protokoll där servern väntar på kvittenser innan nästa paket skickas. Både klient och servern har timers så att de kan sända om vid behov.

- Protokollet kan användas för att ta emot och sända filer (kallas READ respektive WRITE inom protokollet).

HyperText Transfer Protocol – HTTP

HTTP har utvecklats från att vara ett ganska enkelt protokoll för att överföra webbsidor (HTML och de objekt som hör till dem) till en generell och ganska komplex metod med flera varianter. HTTP används inte bara för att skicka webbsidor utan även i modern systemutveckling via regelverk som Web Services och SOAP. I detta kapitel går vi igenom dem översiktligt.

För att göra överföringen tillförlitlig använder HTTP protokollet TCP. För att avgöra vad som ska överföras används URL:er. Uniform Resource Locators är en specifik form av den mer generella formen URI, Uniform Resource Identifiers. Populärt kallar vi dem ofta ”länkar” och de har formen ”protokoll://domännamn [:portnummer] /sökväg [:parametrar] [?fråga]. Variabler inom hakparentes är optioner så en enklare form för endast HTTP blir endast ”http://domännamn/sökväg”. För att översätta domännamn används DNS. Sökvägen anger vilket dokument eller objekt som vi vill hämta. Normalt är det klienten (webbläsaren) som hämtar objekt från webbservern men metoden i sig är tvåvägs och servern kan hämta objekt från klienten.

HTTP har flera egenskaper gemensamma med telnet och FTP. TCP används. Inloggning, kommunikation och felhantering sker i klartext. Detta gör felsökning enkel men det medför också säkerhetsproblem.

I princip kan vi alltså få olika felmeddelanden beroende på om det är sökvägen som inte finns eller om det är namnuppslagningen som inte lyckas. Eller om namnuppslagningen lyckas men aktuell IP-adress inte svarar på port 80.

HTTP har ett par egenskaper som är värda att notera:

HTTP är tillståndslöst. Detta gör det enkelt att bygga servers, de behöver inte hålla reda på vad klienten gjort förr. Klienten frågar efter ett objekt (anger en sökväg) och får ett svar. I princip är överföringen sedan klar och sessionen stängs. Detta är HTTP:s grundfunktion, men ibland vill vi ha kontroll på vad som skett tidigare, det vill säga servern svarar olika beroende på vad klienten har gjort tidigare. Flera tillägg har utvecklats för detta. Ett sätt är att använda cookies (se sidan 79) ett annat sätt är att skicka status via själva URL:en. Följaktligen blir då URL:erna bärare av information och i princip för komplexa för människor. Följande är kanske inte en länk man tipsar sin kompis om: http://www.bolag.

se/jtrac/flow?_flowExeKey=
cC5728581-2A39-B914-6D50-8A7981D8CA5F-k71AAD132-2501ABE5-D231-94460E3C4B6C&_eventId=back.

HTTP är tvåvägs. (En egenskap som delas med de flesta program.) Normalt efterfrågar klienten objekt eller egenskaper hos servern men motsatsen finns också. Dessutom finns kommandona PUT och POST för att klienten ska kunna överföra information till servern.

HTTP är byggt för caching och mellanhänder (proxies).

HTTP version 1.1 använder persistenta förbindelser. När klienten hämtat ett objekt (varav själva HTML-texten kan vara ett) så behålls TCP-förbindelsen öppen tills ena parten begär att den ska stängas. Detta ger flera fördelar för komplexa (moderna) webbsidor. En modern webbsida består av många objekt och flera av dessa kan överföras via samma session. HTTP version 1.0 öppnade en session per objekt. HTTP version 1.0 och dess teknik ökar handskakningen via TCP och dessutom tar det tid för TCP med slow start att uppnå bästa prestanda. HTTP version 1.1 ger en klar förbättring speciellt över länkar med lång fördröjning.

Testa själv

För att testa HTTP kan du enkelt använda telnet och fråga efter en webbsida.

```
C:> telnet www.twoviews.se 80
GET /index.html HTTP/1.1
```

Slå sedan Enter två ganger och din terminal ska få en mängd HTML-kod. En webbläsare ser sedan vilka objekt som ingår och fortsätter fråga efter dem.

HTTP har utvecklats från att i grunden vara ett enkelt protokoll till en komplex samling tekniker och optioner. En webbsida kan bestå av 30–40 objekt och behöva hundratals paket. Det vi uppfattar som en webbsida kan egentligen bestå av flera webbsidor som hämtas på olika servrar. För att förbättra prestanda används dessutom mellanlagring på klienten men även på Internet.

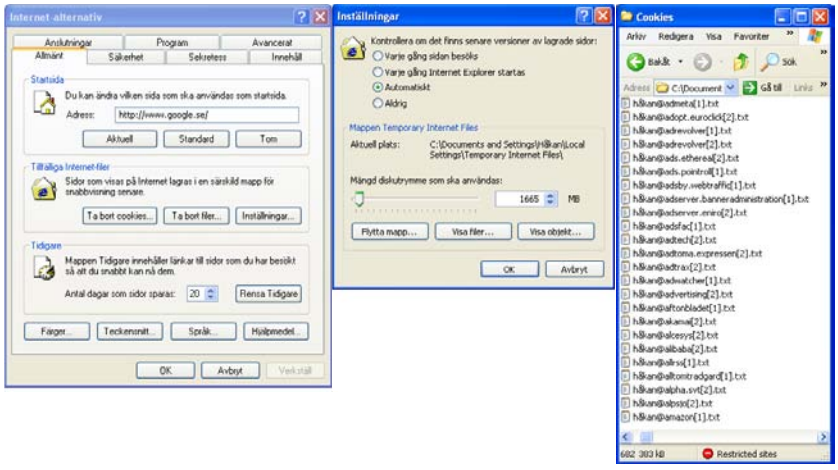
Det är också värt att notera att många webbplatser lagras på många ställen. Om vi surfar till exempelvis microsoft.com, www.fbi.gov eller www.whitehouse.gov så hamnar vi oftast på en av många kopior av webbplatsen. Detta gör att prestanda förbättras eftersom kopian är lokaliserad nära oss, men det gör både namnuppslagning och HTTP mer komplicerad. Vidare kan klient och

server förhandla om språkstöd etc så klienten hämtar olika objekt beroende på våra lokala inställningar.

Webben var från början tillståndslös, en sak som kan lösas med cookies. En cookie är i huvudsakligen en liten sträng som läggs på en bestämd plats på din dator. Varje webbplats skapar en egen sträng. En cookie är mer tillförlitlig än en IP-adress eftersom bland annat adressöversättning gör att fler användare kan uppträda som en IP-adress. Den här lilla strängen gör att en server kan komma ihåg att när du går till kassan så har du tidigare klickat på köp av två böcker. All denna logik bygger på vad servern gör och lagrar om din unika cookie.

På ett liknande sätt kan servern lagra vilka sökningar du brukar göra och sortera svaren efter vad du valt. Givetvis kan servern på detta sätt även styra vilken reklam den vill visa. Cookies är alltså inte en speciellt avancerad funktion och röjer inte så många hemligheter i sig. Men genom att granska vilka cookies en användare har på datorn kan man se vilka webbplatser han besökt. Webbläsare har historiskt också haft problem så att servrar har kunnat läsa varandras cookies, vilket inte är meningen och en oetisk användning. Cookies fick därför snabbt oförtjänt dåligt rykte. I alla moderna webbläsare kan man blockera cookies, men vissa webbplatser fungerar inte utan cookies så de får då användas tillfälligt.

Cookies används även för att räkna antal unika besökare på en webbplats. Flera webbplatser lever på annonser och man behöver skaffa sig en uppfattning om antal unika besökare.



Exempel på inställningar för mellanlagring (cache) och cookies. Längst till höger exempel på cookiefiler.

Säkerhet på applikationsnivå

De program vi gått igenom i det här kapitlet har alla i någon mening säkerhetsproblem. De flesta program skickar även intern kommunikation och inloggningssekvenser i klartext. Detta gör debuggning och felsökning enkel men det gör trafiken känslig för avlyssning.

De flesta program finns i förbättrade varianter med bättre säkerhet inklusive autentisering och kryptering. För HTTP finns en förbättring kallad HTTPS, som bygger på SSL, Secure Socket Layer.

Många äldre protokoll, speciellt de som heter något med "simple", har dålig säkerhet och skickar till exempel lösenord i klartext. Detta måste man vara medveten om. Man bör därför ha andra lösenord i dessa system (så att inte de röjer lösenord till system som i sig är säkra) och inte hantera känsligt data inom dessa protokoll.

Referenser

RFC 959	FTP
RFC 1630 och 1738	Behandlar URL
RFC 2854	HTML
RFC 1945	HTTP 1.0
RFC 2616	Behandlar HTTP version 1.1
www consortium	www.w3c.org
HTTP Essential	Stephen Thomas, 2001 (Enklare att läsa än RFC 2616)
TCP/IP Handboken	Gunnar Gunnarsson, 1999 Trots namnet handlar boken mest om applikationer.