9. De senaste 20 åren har webben växt enormt mycket. Det visar hur väl protokollen fungerar, till exempel http. Webbtjänster utgör det mesta av trafiken, där klienter på olika sätt kontaktar webbservrar. Klienter kan se ut på många olika sätt, med många olika webbläsare och dedikerade program. För att överföra data över webben med HTTP finns olika protokoll. Ett exempel är SOAP, som inte används särskilt mycket i dagsläget (Legacy). Med SOAP strukturerar man ett XML dokument som skickas med frågan, och ett strukturerat XML kommer som svar.

Ett annat, mer använt alternativ är REST. REST är ett lösare definierat protokoll där varje implementering kan variera ganska mycket. Med REST används URL, HTTP-metoderna GET, PUT, POST, DELETE och meddelandets ”body” för att göra förfrågningar.

Många webbtjänster som Google, Amazon, eBay, etc. har öppna APIer. Utvecklare kan då ansluta sina tjänster direkt till webbtjänsten och bygga på dess funktionalitet. Det här används ofta för att koppla ihop tjänster, t.ex. koppla eBay med ett bokföringsprogram för att automatisera mycket av arbetet.

En viktig del av överföringar via internet är att de skönts asynkront. Processen på mottagarsidan ska alltså inte sitta och vänta på att meddelandet ankommer, utan snarare att den fortsätter med annat tills meddelandet dyker upp.

Lös koppling är också en önskad egenskap på en webböverföring. Det ska alltså inte behövas samma interface som på andra sidan för att kunna tolka meddelandet. Därför används ofta standardiserade protokoll som XML och JSON.

Alla webbtjänster brukar ha en URI, där URL är den vanligaste formen. Den visar alltså var i ett teoretiskt filsystem resursen ligger. Webbtjänsterna behöver inte alltid vara igång, utan kan väckas när en förfrågning kommer in. Det kan också ligga på en annan dator och bara ”studsa” vidare.

Jämfört med Remote Object References är URI är webbtjänst-versionen betydligt mycket enklare då garbage collection och remote object referencing inte behövs. Coola funktionerna med referencing saknas dock också.

En ”Service description” används för att beskriva var en online-resurs finns och hur man kan använda den. Den innehåller bland annat tjänstens URI. Språk/protokoll finns redan för detta, som t.ex. WSDL där fråga och svar definieras. Det går även att definiera vilka fel som kan uppstå och vilken part i kommunikationen en viss rutt är ägnad åt. (Legacy). Det finns även verktyg som UDDI där hierarkin och strukturen på objekten som skickas kan definieras. Räknas också som legacy-verktyg.

XML kan behöva utökad säkerhet i vissa fall. Ex. om den innehåller känslig data. Kraven som ställs på överföringen skulle kunna vara dessa:

- Möjlighet att kryptera dokumentet i sin helhet eller delvist
- Möjlighet att signera dokumentet i sin helhet eller delvist
- Möjlighet att signera ett redan signerat dokument
- Möjlighet att tillåta olika användare att se olika delar av dokumentet
- Möjlighet att lägga till innehåll i ett redan krypterat dokument och kryptera det nya dokumentet

Att hjälpa användarna att hitta rätt nycklar kan också vara en fördel. Därför kan KeyInfo-elementet användas.

Ett sätt att implementera distribuerade system är cloud computing. Med hjälp av en cloud-infrastruktur kan många applikationer köras, data lagras helt utan att användaren ska behöva göra något lokalt. Ett exempel på en cloud infrastruktur är AWS. Tjänsterna EC2 och S3 uppfyller kraven för processering respektive lagring.

10. Peer-to-peer är ett sätt att låta skalbarheten av en tjänst vara nära till obegränsad. Det kan vara användbart för att t.ex. dela filer där bandbredden helt enkelt inte hade räckt till mellan en tjänsts server och användare. Ett helt peer-to-peer system kan definieras som följande egenskaper.

- Alla användare bidrar till systemet
- Alla noder ska ha samma funktionalitet, även om prestandan kan vara olika
- Systemet ska fungera utan att en centraliserad nod existerar
- Användarna och resursernas källa kan vara anonym
- Algoritmen för spridning av data måste lägga till minimal overhead.

Noderna sätter upp en egen topologi av nätverket runtom de, och utefter den kan kommunikation ske. Paketen hashas på så sätt att mottagaren kan identifiera om paketet faktiskt kommer från rätt nod.

Idéen av att låta oanvänd datorkraft till något större har funnits länge, och har t.ex. används i projekt som seti@home. De brukar dock behöva en något centraliserad nod för att köras effektivt.

Napster (musiktjänst) var en av de stora peer-to-peer applikationerna i tidiga 2000-talet. De hade ett centraliserat index av vilka låtar som fanns att spela, men filerna låg hos, och delades mellan användarna. När användare kopplar sig till nätverket identifierar napster vilka låtar den har på sin dator och gör de tillgängliga till andra användare i topologin.

Middleware/overlay kan användas för att förenkla kommunikationen mellan noder. För identifiering används GUIDs. Nodens GUID genereras som en hash av nodens egenskaper. För att skicka, publicera och avregistrera filer inom peer-to-peer kan då funktioner med GUIDs användas, istället för att behöva mecka med IP+port och liknande. Funktionen kan se ut som följande: publish(guid), unpublish(guid), sendToObj(msg, guid, […]). Det ger en mer abstraherad kodning och använder sig av GUIDs för att beräkna antal hopp och liknande nätverkslogik.

11. Säkerhet i distribuerade system kräver ganska avancerade lösningar på grund av alla olika kanaler som en attack skulle kunna använda sig av. Exempelvis behöver processerna kapsla in data som hanteras av den, så att inte minnet kan läsas av eller andra hårdvaruattacker. Processerna behöver också kryptera och/eller signera data som skickas över nätverket för att se till att inget sker med den mellan noderna.

Som access till olika rum och byggnader för ett företag kan filer, program och andra resurser låsas till vissa användare. Kryptografi spelar en viktig roll i säkerhet kring data, genom att använda det kan man skapa signaturer och kodning som bara användare med nyckel kommer att ha tillgång till. Kryptering av data har utvecklats under lång tid, med sina rötter hos 60 - 90 talets matematiker. Efter att länge ha använts för bara militära och politiska ändamål började det bli mer lättåtkomligt under 90-talet.

Att sniffa paket på lokala nätverket är en vanlig attack, medans de mer avancerade kan vara program och filer som kopierar och installerar filer på användarens dator. För att förebygga attacker problemområdena kategoriseras:

- Läckage: Obehörig mottagare får tag på data
- Manipulering: Obehöriga redigerar data
- Vandalism: Användare förstör utan att vinna något på det.
- Avlyssning: Obehöriga får kopia av meddelanden
- Personifiering: Skicka/ta emot meddelanden som en användare utan att ha behörighet från användaren
- Manipulering av meddelanden: Attacker som man-in-the-middle där obehöriga stoppar meddelandet på vägen, byter ut det med något annat och skickar vidare som om det vore originalet
- Återuppspelning: Meddelanden kopieras på vägen och sparas till senare
- Denial of service/DDOS: Skicka så pass många meddelanden att kanalen blockeras för andra användare.

För att se till att dessa attacker inte sker finns det många olika säkerhetsåtgärder. JVM är ett bra exempel, där många steg av verifiering sker innan kod kan köras. När JVM tar emot ”remote” klasser lagras de alltid separat, och kan då inte ersätta den lokala kopian. Bytecodes verifieras alltid för att se till att de är korrekta och att de har tillträde till den instruktionen de ska utföra.

Med läckage av information skulle en obehörig person kunna få värdefull information, som t.ex. om många köper och säljer en viss aktie skulle det kunna förvarna för något. Därför används kryptografi för att se till att bara sändaren och mottagaren kan läsa informationen. Allmänt behöver transaktioner på nätet säkras på ett liknande sätt. Epost kan krypteras, och sedan dekrypteras vid inläsning. För internetköp och banktransaktioner krypteras det mesta samt att man ofta bara skickar över publika nycklar så kortnummer inte kapas på vägen.

En annan viktig aspekt av online-transaktioner är så kallad *Non-repudiation.* Det innebär att användaren som skapat transaktionen kan verifieras ha skickat transaktionen, och kan alltså inte senare backa ut genom att förneka att han ens varit med.

Att designa ett säkert distribuerat system är verkligen inte lätt. Utvecklaren måste hitta lösningar mot alla möjliga attacker och kryphål. För att kunna åstadkomma detta behöver utvecklaren först definiera en lista med alla möjliga kryphål och attacker. Det går sedan att testa mot dessa fall och se till så systemet klarar attackerna. Loggning är även viktigt, så att nya attacker eller attacker som utvecklaren kanske missat registreras för att sedan åtgärdas. Säkerhetsåtgärderna behöver alltid vägas mot deras kostnad varav det finns två olika typer: kostnad i resurser (nätverk och processorkraft) samt åtgärder som låser ut giltiga användare, eller gör deras arbete svårare.

Ett par scenarion kan tas för givet vid designfasen av ett säkert distribuerat system:

- Interface är synliga, meddelanden kan skickas av alla till alla interface
- Nätverk är osäkra, ett meddelandes källa kan falsifieras, host-adresser kan “spoofas”
- Varje nyckel måste ha en begränsat giltig tid, så risken för att den används av någon obehörig i framtiden minskar
- Algoritmer och programkod är tillgänglig till angriparna
- Angriparna har stora resurser
- Minimera mängden system och användare som systemet litar på

Praktiskt sett innebär kryptografi användningen av olika algoritmer som kodar och avkodar meddelanden med hjälp av nycklar.

Med delad-nyckel-kryptografi kan en överföring av meddelanden se ut som följande:

1. A använder den delade nyckeln och krypterar meddelandet.
2. B avkrypterar meddelandet med hjälp av deras delade nycklar.

Problemen som uppstår vid den här enkla kryperingsmetoden är:

- Hur kan A skicka B deras delade nykel?
- Hur vet B att meddelandet är det som skickats av A, och inte ett meddelande som förändrats på vägen?

Autentisering kan användas för att bekräfta att A faktiskt är A. Det finns lite olika metoder för autentisering, exempelvis att använda en centraliserad autentiserings-server. En likande lösning skulle kunna uppnås genom att använda public/private-nycklar. Låt A kryptera ett meddelande med B:s publika nyckel. A kan då kryptera den delade nyckeln och skicka den till B så att bara B kan avkryptera den med sin privata nyckel.

Det här löser problemet med delade nyckeln, men A kan fortfarande vara någon obehörig. För det problemet kan vi istället använda digitala signaturer. A krypterar ett meddelande med Apriv, vilket innebär att bara A kan ha skickat det då bara A:s privata nyckel kommer kunna dekryptera det.

Certifikat är ett sätt att se till så t.ex. publika nycklar och signaturer inte modifieras. Man kan då anlita en CA (Certificate authority) och använda de för att validera sin signatur.

För att säkra hårdvarusystem används access control, credentials (lösenord). Brandväggar kan användas för att se till att så lite som möjligt av nätverket är tillgängligt utåt.

Vanliga delade-nyckar-krypteringsalgoritmer är TEA, DES, IDEA, RC4, AES. För public/private algoritmer kan istället RSA m.fl. användas. Oftast används dessa tillsammans med varandra, man talar då om hybrida krypteringsalgoritmer. Alltså asymmetrisk för autentisering och signatur, för att överföra den delade nyckeln, sedan används symmetriska algoritmen.

Inom signering brukar s.k. digest användas, där protokollen MD5 och SHA används. Man skapar då en hash som bara blir likadan om rätt signatur används vid hashning.

REST är de facto standarden för att representera resurser tillgängliga över client-server kommunikation. Det är inte en standard, utan snarare ett tankesätt vid design av system. Det använder sig av färdiga standarden HTTP. Resurser mappas som en URI och vad som behöver följa med i förfrågan till server. Ex. GET /news, som hämtar resursen på /news via HTTP metoden GET.

REST är tänkt att vara stateless, vilket innebär att alla transaktioner måste innehålla all relevant data. Om användaren t.ex. försöker beställa något kan det inte antas att kundvagnen redan finns hos servern, utan den ska skickas med. För att spara information över flera transaktioner används Sessions, där användaren då lagrar någon form av identifikator på sin lokala enhet i en cookie.

Fördelen med REST och dess krav är skalbarhet samt prestanda. Genom att inte känna till något om användaren innan transaktionen kan information som skickas över ofta lagras i caches, och på så sätt spara bandbredd och processorkraft hos servern. Exempelvis används CDN-tjänster för detta i praktiken.

Reflektion

En snabb tanke som dyker upp är väl hur mycket enklare de moderna protokollen är jämfört med vad som presenteras i boken. CORBA, SOAP och de andra protokoll som boken lyfter har sjukt många steg för allt och massvis med data som verkar skickas i onödan. Fast det kanske man kan säga om REST och allt annat som används idag också.

Folding@home verkar vara ett perfekt exempel på användandet av användarnas datorer som fortfarande är sjukt aktivt än idag. Det verkar dock som att säkerhetsriskerna och lagliga frågan med peer-to-peer har lett till att de allra flesta webbtjänster utförs med client-serverkommunikation. BitTorrent är väl det som man kommer att tänka på när Peer-to-peer nämns idag.

Kapitlet om säkerhet är mycket intressant, och den hybrida lösningen med autentisering och överföring av nycklar med asynk, för att sedan köra överföringen av stora filer med synk-algoritmerna känns logisk. Det ska bli kul att prova kryptering då det är något som fortfarande inte förekommit i utbildningen, ex. 2:ans projekt använde sig av SSO som integrerades av företaget efter vi lämnat projektet.

REST är mycket bekant och känns som standard åtminstone för ett par år framöver. Som vanligt inom data lär ett nytt protokoll eller designsätt dyka upp, men det kommer nog bli svårt att slå friheten och enkelheten av ett väl dokumenterat RESTful API.