ARKIV

Medlemsblad for

Dansk UNIX-system Bruger Gruppe

DKUUG-Nyt

Mikro Data 89

Indhold

Redaktionelt

Dette er et særnummer af DKUUG-Nyt, udgivet i anledning af Mikro-Data '89. DKUUG-Nyt udkommer 10 gange om året og udsendes til alle medlemmer af DKUUG.

DKUUG-Nyts redaktion består af Søren O. Jensen og René Seindal (ansvarshavende).

Vi er naturligvis altid interesserede i indlæg fra folk. Det behøver ikke være lange artikler, men kan også være annonceringer, opfølgninger af tidligere artikler, eller andet. Hvis I blot har ønsker eller gode ideer til artikler, er I også meget velkomne til at kontakte os. Bidrag til bladet bør indleveres på maskinlæsbar form.

Indlæg, foreslag, ønsker, etc. kan sendes med elektronisk post til redaktionen på adressen:

`dkuugnyt@dkuug.dk`

eller, hvis man foretrækker almindelig sneglepost, til:

René Seindal
Datalogisk Institut
Universitetsparken 1–3
2100 København Ø
Telefon: 31–39 64 66, lokal 222.

DKUUG og DKUUG-Nyt

Af *Søren Oskar Jensen*

Dansk UNIX-system Bruger Gruppe (DKUUG) er som navnet indikerer en forening for brugere af Unix-operativsystemet. Foreningen blev stiftet i 1983 som et forum for firmaer, institutioner og privatpersoner med interesse for Unix. I dag har foreningen ca. 300 medlemmer, hvoraf størstedelen er firmaer og institutioner.

DKUUG er medlem af foreningen EUUG—European UNIX systems User Group, der fungerer som paraplyorganisation for de europæiske Unix-brugergrupper.

DKUUG tilbyder en række aktiviteter til sine medlemmer, disse aktiviteter omfatter: DKUUG-Nyt, medlemsmøder, DKnet, båndsevice og en markedsoversigt.

DKUUG-Nyt

Du sidder for øjeblikket og læser i et særnummer af DKUUG-Nyt—det eneste danske blad om Unix. Dette nummer er ment som en præsentation af bladet og indeholder derfor en række af de artikler, der har været bragt i bladet indenfor det sidste år.

DKUUG-Nyt udkommer 10 gange om året og udsendes til alle medlemmer af foreningen. Bladet bruges dels til orientering om foreningens aktiviteter, dvs. annoncering af møder, reportager fra arrangementer etc., dels til generel information om Unix.

Medlemsmøder

DKUUG afholder omkring 8 medlemsmøder om året, disse møder er enten "gå-hjem-møder" eller heldagsmøder. De enkelte møder er helliget et Unix-tema, der så behandles af danske og udenlandske talere. I 1989 har temaerne for disse møder bl.a. været: grafik, CASE-værktøjer, Unix i kontormiljøet, netværk og distribuerede systemer.

Derudover kan man som medlem af DKUUG deltage i de 2 årlige EUUG-konferencer.

DKnet

I 1983 oprettede DKUUG et netværk til udveksling af elektronisk post og nyheder mellem Unix-systemer, kaldet DKnet. Dknet har i dag ca. 80 tilslutninger i Danmark. Ligesom EUUG fungerer som paraplyorganisation for de nationale brugergrupper, så har EUUG også en netværks-organisation, kaldet EUnet, der samler de europæiske Unix-net.

Via DKnet kan man udveksle informationer med over 100.000 andre Unix-systemer over hele verden. Endvidere er det også muligt at kommunikere med personer på andre netværk via DKnet.

Man kan mod betaling blive koblet på DKnet, hvis man er medlem af DKUUG.

Båndservice

DKUUG har en båndservice, som giver medlemmer mulighed for, til kostpris, at få magnetbånd med alt det nyeste gratis-programmel. Der er bånd med X vindues systemet, med tekstbehandlingssystemet TEX, md GNU programmel, og meget mere. Alt programmel distribueres i kildetekstform, og er frit for licenser.

Priserne for DKUUG bånd ligger normal omkring kr. 200–300, inklusiv forsendelse.

Markedsoversigt

DKUUG udarbejder årligt en markedsoversigt, som omfatter størstedelen af de Unix produnkter, der er tilgængelig via danske leverandørere.

Medlemskab af DKUUG

For at få adgang til DKUUG forskellige services, skal man naturligvis være medlem af foreningen. Indmeldingsblanket og betingelserne for indmeldelse fremgår af en separat DKUUG folder, som forefinde på DKUUGs stand på Mikro Data 89.

X Window System

Af *Steen Lindén*
anubis@diku.dk

Efterhånden som arbejdsstationer er blevet mere og mere almindelige i Unix-verdenen, og skærmteknologien har udviklet sig i retning af bitmap orienterede højopløsningsskærme, er det blevet mere og mere almindeligt at anvende vinduessystemer. SunView på Sun arbejdsstationer og X Window System på bla. Hp, IBM og Digital[1] arbejdsstationer er de to kendteste. Denne artikel omhandler designet af X vinduessystemet.

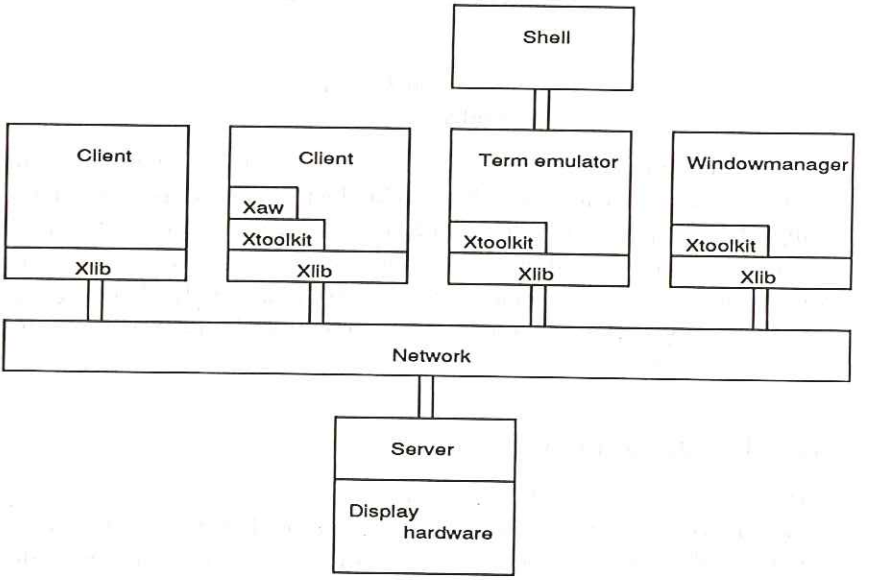
Hvad er X Window System

Fra brugersynspunkt opdeler X Window System arbejdsstationens fysiske skærm i et hieraki af vinduer, hvori der kan tegnes og skrives. Dette vindueshieraki manipuleres ved hjælp af en mus eller en anden anordning, der gør det muligt at bevæge en markør på skærmen. Med denne markør kan man gøre vinduerne større eller mindre, flytte dem rundt på skærmen og dirigere input fra tastaturet til det vindue man ønsker. Det er tilladt for vinduerne at overlappe hinanden, så nogle vinduer er helt eller delvist tildækkede af andre. Effekten ved alt dette er, at f.eks. et antal shells, et ur og et CAD-program kan anvende arbejdsstationens skærm simultant.

X Window System er udviklet på Massachusetts Institute of Technology (MIT) med støtte fra forskellige firmaer, bla. de ovennævnte, der tilsammen udgør X Consordium, som er ansvarlig for den videre udvikling af X standarden.

MIT distribuerer sin version af vinduessystemet *gratis* med kildetekster og dokumentation, og sammen med den designfilosofi, der ligger til grund for X, har det bevirket, at X er blevet meget populær overalt i Unix-verdenen. Bla. GNU projektet har adopteret X som det vinduessystem, der supporteres af GNU Emacs.

[1] Digital lancerer X under navnet DECwindows.



Figur 1: Server/klient princippet i X

Det grundlæggende design

X er opbygget efter en netværksbaseret server/klient model. Serveren distribuerer input fra musen og tastaturet til klienterne og modtager output til skærmen fra disse. Klienterne er brugerapplikationer, der har forbundet sig til serveren og kan f.eks. være en terminalemulator, der gør det muligt at køre programmer, som er designet til konventionelle terminaler eller et grafisk repræsentation af et analogt ur. Server/klient princippet er illustreret i figur 1

Denne opdeling i klienter og en server, som kommunikerer via X protokollen, bevirker at X er netværkstransparent. Det vil sige, at man foruden at køre klienter på sin arbejdsstation samtidig kan køre klienter på andre datamater i et netværk, også på datamater, der har

anderledes maskinarkitektur og operativsystem i forhold til arbejdsstationen. Det eneste krav er, at serveren og klienten benytter samme netværksprotokol. Dette synes måske i første omgang at være en lidt overflødig egenskab, men den viser sig at være utrolig nyttig, når man har brug for regnekraft fra en datamat, der er større end ens arbejdsstation. Endvidere gør princippet det muligt at fabrikere X terminaler, dvs. intelligente terminaler, som implementerer serverdelen af vinduessystemet i hardware.

MIT distributionen af X implementerer protokollerne TCP/IP, DECnet og Unix domain sockets, men i princippet kan X køre over en hvilken som helst datastrøm, der er hurtig nok.

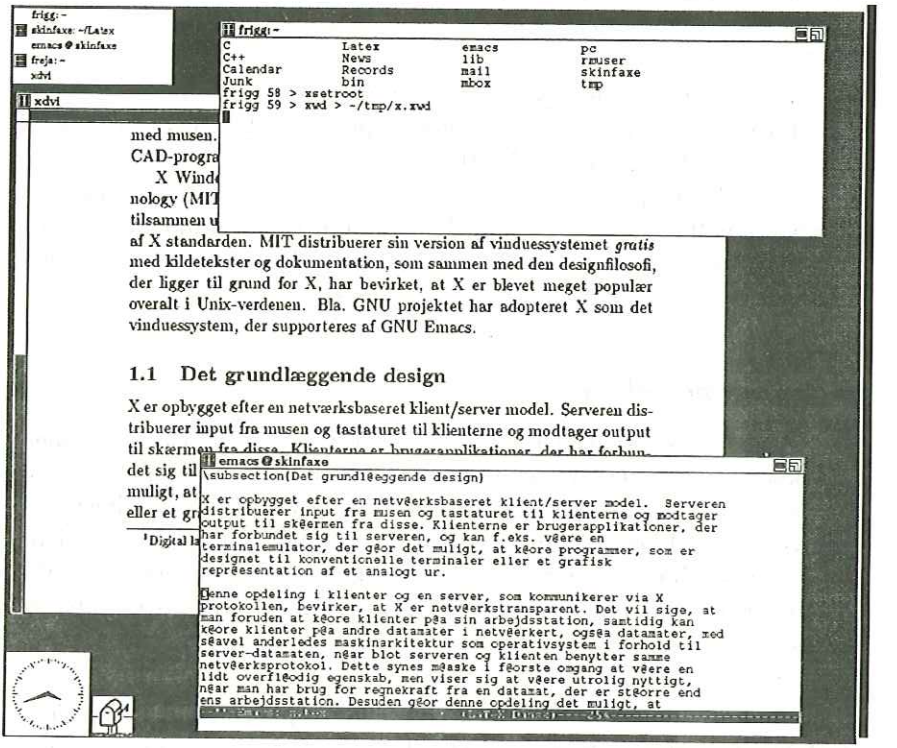
Server/klient princippet bevirker også, at de enkelte X klienter er hardware uafhængige, hvilket medvirker til at gøre dem portable. Netop portabiliteten var et krav, som MIT stillede sig selv ved designet af X og en medvirkende årsag til X's succes.

Brugergrænsefladen

Det grundlæggende princip i X er "*mechanism, not policy*". Systemet må ikke påtvinge brugeren en bestemt grænseflade, som det f.eks. er tilfældet i mange andre vinduessystemer. Under X findes en windowmanager, en speciel klient, som giver brugeren mulighed for at manipulere med sine vinduer i en grænseflade fastlagt af windowmanageren. På samme måde som man vælger shell i Unix, kan man så vælge og tilpasse sin windowmanager efter eget ønske og dermed opnå den grænseflade, der passer bedst til ens temperament.

Windowmanageren er ikke et priviligeret program, der har flere rettigheder end en almindelig klient. Den eneste måde at kommunikere med serveren på er via X protokollen, som er fælles for alle klienter.

Der findes flere forskellige windowmanagere. En populær en er twm. Et eksempel på skærmlayout kan ses på figur 2. Uwm er MIT's windowmanager, som i modsætning til twm ikke forsyner vinduerne med en såkaldt dragbar, men istedet kræver, at man skal bruge en kombination af musen og tastaturets meta-tast, når man skal flytte eller ændre størrelse på et vindue. Rtl er en højst utraditionel windowmanager, der



Figur 2: Eksempel på X med windowmanageren twm.

ikke tillader vinduer at overlappe hinanden, men istedet omorganiserer skærmen ved at flytte vinduerne, når en ny klient har brug for plads.

Mens mange vinduessystemer er snævre, med forudfastlagt politik om hvordan f.eks. en scrollbar fungerer, er X bredt med mulighed for mange variationer. Ulempen ved dette er, at X for førstegangsbrugeren virker stort og uoverskueligt, præcis som det er tilfældet for mange, når de første gang stifter bekendtskab men Unix.

X programmering

Designfilosofien i X kan på mange områder sammenlignes med filosofien i Unix. Inderst ligger kernen, som tager sig af fordelingen af resourcer samt fysisk in- og output. Unix systemkaldene svarer til X protokollen mellem server og klienter, som er implementeret i C i biblioteket Xlib.

I lighed med de mange biblioteksfunktioner under Unix, som supplerer systemkaldene, findes der over Xlib diverse toolkits. Toolkit'et Xt, der som Xlib hører med til X standarden, er et "objektorienteret" toolkit implementeret i C, hvoraf man bygger de såkaldte widgets, f.eks. scrollbars eller dialogboxe.

Med MIT distributionen følger et sæt widgets, Xaw, som er udviklet af Project Athena på MIT. Disse widgets er ikke en del af X standarden, men supporteres alligevel af MIT og ligger til grund for mange af de klienter, der følger med distributionen. Også andre end MIT har lavet widgets til Xt, f.eks. Sony og Hp.

Af andre toolkits kan nævnes InterViews, som er et C++ interface, der bygger på Xlib samt et Common Lisp interface, CLUE, der er en overbygning på CLX; en implementationen af X protokollen i Common Lisp.

MIT X distributionen

MIT's X distribution er delt op i to dele; en kernedel, som indeholder de dele, der supporteres af MIT og en del bidraget af andre, som indeholder forskellige toolkits, windowmanagere og applikationer. Distributionen kan skaffes på bånd fra DKUUG eller via anonym ftp til freja@diku.dk (129.142.96.1), hvor der også findes patches.

Serveren, der følger med distributionen, er i følge X11R3 Release Notes blevet bygget under følgende operativsystemer.

- 4.3+tahoe
- Ultrix 3.0 og Ultrix 2.0
- SunOs 3.4
- HP-UX 6.01
- Apollo Domain/IX 9.7 (også 9.6 iflg. udviklerne)
- IBM AOS 4.3 (iflg. udviklerne)
- A/UX 1.0

På Datalogisk Institut har vi MIT's X11R3 installeret på VAXstation2000's, Sun3's, Sun4's og en Vax785, samt IBM's version af X på en IBM RT. Som følge af standardiseringen af X er der ingen problemer i at lade en kommerciel udgave som IBM's køre sammen med MIT distributionen, og det har vist sig, at performance over nettet er lige så god som lokalt på arbejdsstationen.

Litteratur

[1] Robert W. Scheifler and Jim Gettys, "The X Window System" *ACM Transactions on Graphics* vol. 5, no. 2, 1986. En generel beskrivelse af X med udgangspunkt i version 10.

[2] Oliver Jones, "Introduction to the X Window System", Prentice Hall 1989. Beskrivelse af Xlib med velvalgte eksempler og kommentarer.

[3] Joel McCormack and Paul Asente, "Using the X Toolkit or How to write a widget" procedings from *Summer USENIX '88*, 1988. Introduktion til toolkittet Xt og brugen af widgets.

[4] David S. H. Rosenthal, "Going for Baroque" *UNIX REWIEW* June 1988. "Hello World" programmet udsat for Xlib og Xt.

[5] Ed Lee, "Window of Opportunity" *UNIX REWIEW* June 1988. Diskussion af X vinduessystemets design og dets fremtid.

Internet-ormen

Af *René Seindal*

I sidste nummer af DKUUG-Nyt lovede vi at fortælle lidt om den famøse Internet-orm, som maltrakterede det amerikanske internet i november sidste år. Vi vil i denne artikel forsøge at beskrive hvad slags program ormen var, hvad det gjorde og ikke gjorde, hvordan det gjorde det og hvad konsekvenserne af det var.

Internet-ormen er et selvreplikerende program, som blev sluppet løs på det amerikanske internet d. 2. november, 1988. Programmet kunne overføre sig selv til andre maskiner via internettet, og spredte sig således til tusinder af maskiner over hele USA. Programmet var ikke i sig selv ondsindet, men fejl i det gjorde at det ofte inficerede samme maskine flere gange, hvilket forøgede belastningen drastisk.

Denne artikel er baseret på to tekniske rapporter om internet-ormen, som blev skrevet kort efter dens fremkomst på internettet. Deres titler kan findes sidst i artiklen.

Lidt terminologi

Før vi kigger nærmere på selve ormen, er det vigtigt at gøre sig klart hvad forskellen på en orm og en virus er.

Den vigtigste forskel ligger i iagtagelsen af at en orm kan eksistere (leve?) uafhængigt af andre organismer, og reproducere sig uden hjælp udefra, mens en virus altid er bundet til en værtsorganisme, uden mulighed for at overleve uden værtsorganismen. I den forstand passer ordene ganske godt på deres edb-sidestykker.

De angreb som kendes fra PC-verdenen har alle været vira, som har inficeret værtssystemet (DOS), og spredt sig via inficerede disketter og lignende medier. Dette faktum har medvirket til at gøre virusbegrebet kendt, og delvis forårsaget den begrebsforvirring, der har været omkring internet-ormen.

En orm, derimod, er en organisme (et program) som kan leve uden at være bundet til andre specifikke organismer (programmer). (At internet-ormen kun fungerede på Vaxe og Sun-3'ere under Berkeley

Unix ændrer ikke noget. Der findes også orme, som kun kan leve i subtropiske omgivelser. Det gør dem ikke til vira af den grund).

Det vil fremgå af det følgende at Internet-ormen, som navnet siger, var en orm, og ikke en virus, selvom den ofte bliver omtalt som en sådan.

Hvad ormen ikke gjorde

En ikke uinteressant ting ved et ukontrollabelt program som Internet-ormen, er hvad det *ikke* gjorde.

Først, så angreb ormen ikke alle systemer, men kun Vaxe under Berkeley Unix, og Sun-3'ere under SunOS. Disse begrænsninger bidrog naturligvis i nogen grad til at hindre programmet i at sprede sig, men var dog ikke bare tilnærmelsesvis nok til at stoppe programmet.

Dertil kommer at ormen var afhængig af internettet for at kunne sprede sig. Det betyder specielt at systemer, som ikke var forbundet til internettet ikke kunne inficeres, hvilket på det pågældende tidspunkt inkluderede Danmark. Så vidt jeg ved, har der ikke været nogle angreb i Europa overhovedet.

Når ormen først havde inficeret et system, forsøgte den ikke på nogen måde at modificere værtssystemet. Den ændrede ikke nogle systemfiler, og den slettede ikke andre filer end sine egne temporære filer. Ormen forsøgte heller ikke at installere trojanske heste eller at registrere eller opsamle de passwords, som måtte komme den i hænde. Interessant nok, så forsøgte den heller ikke at opnå root-privilegier. Den udnyttede dem ikke engang, hvis den fik dem i hænde alligevel.

Hvad ormen gjorde

Når man tager i betragtning hvad ormen ikke gjorde, så er det ret klart, at dens eneste formål er at reproducere sig, og at sprede sig. Den gjorde begge dele meget effektivt.

Før ormen kunne inficerer en anden maskine, måtte den først finde ud af hvilke potentielle ofre, der var tilgængelige fra den maskine, den befandt sig på. Når den havde fundet et potentielt offer, var den nødt til at forbinde til den anden maskine over nettet, og få startet en shell

i den anden ende. Hvis det lykkedes, kunne ormen overføre sig selv til den anden maskine, starte den nye orm, og afbryde forbindelsen. Den ene orm var så blevet til to, som begge ville fortsætte med at lede efter maskiner at bryde ind på.

Lokalisering af potentielle ofre

En stor del af ormens succes i at brede sig, skyldes dens metoder til at finde andre maskiner den kunne inficerer. Ormen brugte forskellige metoder alt efter om den ledte efter maskiner på samme lokalnet eller på andre net.

Berkeley Unix giver mulighed for at erklære to maskiner for 'ekvivalente,' dvs. at man ikke skal angive password ved login mellem to ekvivalente maskiner. Det er på tilsvarende vis muligt for individuelle brugere at erklære at de stoler på givne brugere på andre maskiner, som da vil få adgang til den pågældende brugers konto uden at skulle bruge password. Det er et system, som kræver omtanke i brugen, men på trods af de åbenbare problemer det kan åbne op for, er det en fuldstændig uundværlig facilitet i et netværksbaseret miljø.

Ormen kunne bruge disse oplysninger til at gætte hvilke konti der måske fandtes på andre maskiner, ved at antage at det var et gensidigt tillidsforhold mellem to maskiner (hvad det som oftest også er). Logikken i det kan kort skitseres som følgende: Hvis en bruger ved navn 'foo' på maskinen 'A' erklærer at han stoler på brugeren 'bar' på maskinen 'B,' er der en vis sandsynlighed for at det rent faktisk er den samme person, der har begge konti, og der er således en god chance for at 'bar' på 'B' også stoler på 'foo' på 'A.' I så tilfælde kan man blot starte en shell på den anden maskine som den pågældende bruger, ved hjælp af standard procedure til udførsel af shell-kommandoer på andre maskiner.

Selv i det tilfælde hvor der ikke var adgang til andre maskiner uden password, kunne ormen stadig bruge en oplysning som at 'foo' på 'A' fik sin post videresendt til 'bar' på 'B,' til at bryde ind på den anden maskine. Blot det at kende et gyldigt brugernavn på en anden maskine, er det første skridt på vejen til en succesfuld angreb.

Hvis ormen kun havde angrebet lokale maskiner, var den aldrig kommet så langt omkring. Ormen brugte flere forskellige ikke-trivielle

metoder til at finde ud af hvilke andre ikke-lokale maskiner den kunne komme i kontakt med. Den prioriterede specielt gateways højt (en gateway er en maskiner, der er forbundet til flere forskellige fysiske netværk, og den fungerer da som en bro mellem de to net), da det ville give bedst mulig adgang til endnu flere maskiner.

Hvordan man får en shell på en fremmed maskine

Når ormen havde lokaliseret en potentielt offer, forsøgte den som det første at forbinde til den anden maskines *telnet* program. Hvis det lykkedes, afbrød ormen forbindelsen med det samme, og hvis det fejlede opgav den at angribe den anden maskine. På denne måde fik ormen filtreret 'dumme' bokse, som f.eks. routere, fra.

Efter at være kommet i kontakt med en maskine, var ormens første opgave at få startet en shell i den anden ende. Ormen havde tre måder den kunne starte en shell på den anden maskine på. De er, i den orden ormen prøvede dem på, (1) at benytte oplysninger om ekvivalente maskiner og konti, (2) at benytte et sikkerhedshul i *fingerd* programmet, og (3) at benytte et sikkerhedshul i *sendmail*. Hvad disse sikkerhedshuller dækker over vil vi gennemgå i det følgende.

Ormen forsøgte først at bryde ind på en almindelig konto på den anden maskine, ved at bruge brugernavne, som på den lokale maskine var erklæret ekvivalente med offeret, eller hvor den kendte det lokale password (en af grundene til at ormen forbrugte så mange ressourcer, var at den gjorde ihærdige forsøg på at bryde folks password, for på den måde at kunne komme videre). Hvis det lykkedes ormen at få adgang til en konto på den anden maskine, startede den ved hjælp af en standard biblioteksfunktion en shell på den anden maskine, som den kunne bruge til at overføre sig selv med.

Hvis det ikke lykkedes ormen at få adgang til den anden maskine via en bruger konto, forsøgte den at udnytte en sikkerhedshul i *fingerd* programmet. *Fingerd* er en facilitet, som tillader folk udefra at se hvilke konti der er på en maskine, og hvem der er logget ind. Det er absolut ikke noget essentielt program, men de fleste sites har det alligevel kørende.

Ormens angreb på *fingerd* er nok den mest opfindsomme del af ormen. Det foregik som følger: *fingerd* fungere ved at *finger* programmet

fra en anden maskine forbinder over nettet til det. Det læser en linie fra netværksforbindelsen, giver den som argument til det lokale *finger* program, og sender uddata herfra tilbage over nettet. Det er et meget trivielt program. Sikkerhedshullet var også trivielt, idet det bestod i at programmet læser sine inddata med biblioteksfunktionen *gets*() med en lokal variabel som den modtagende buffer! Oplagt, ikke?

Den buffer, som *fingerd* brugte, var på 512 tegn. Det ormen gjorde, var at den sendte en nøje konstrueret linie, på 536 tegn, som da overskrev return adressen fra *main*(), så den pegede ned i bufferen. Nå *fingerd* returnerede, udførte den koden i bufferen, som simpelthen startede en shell. Ormen havde da en netværksforbindelse til en shell på den nye maskine.

Hvis *fingerd* angrebet fejlede, faldt ormen tilbage på et sikerhedshul i *sendmail*. *Sendmail* er et systemprogram, som varetager forsendelse af elektronisk post over et internet, ved hjælp af SMTP protokollen. En af faciliteterne i *sendmail* er, at det kan udføre en given shell kommando som respons på modtagelsen af en brev. *Sendmail* har også nogle indbyggede 'debug' faciliteter. Hvis *sendmail* er konfigureret rigtigt (forkert!), er det muligt at aktivere 'debug' faciliteterne over nettet, og dermed starte programmer på den anden maskine, som får et brev som inddata.

De færreste leverandører har været opmærksomme på denne lille detaille, så mange systemer kørte *sendmail* med 'debug' aktiveret. Ormen benyttede det hul til starte en shell på det udvalge offer. Den lavede en forbindelse til *sendmail* på den anden maskine, aktiverede 'debug,' og sendte et lille shell-script med en `/bin/sh` som modtager.

Ironisk nok, så skyldtes denne 'debug' bagdør i *sendmail* en meget sikkerheds-orienteret systemadministrator på Universitetet i Berkeley, som insisterede på at benytte en beta-test udgave af *sendmail* (uden tilladelse, forøvrigt), men ikke ville forsyne *sendmail*s forfatter med en konto til at teste den med. 'Debug' faciliteten blev derfor lagt ind for at kunne teste dette system, og den blev senere glemt [1].

Selvreplikationen

Nå ormen havde sikret sig at den anden maskine kunne inficeres, overførte den et lille 'vektor-program' til den anden maskine. Hvordan det

blev gjort i praksis afhang af hvilken af de tre angrebsstrategier, der lykkedes. Vektor-programmet blev oversat og startet på den anden maskine, og etablerede derefter en forbindelse tilbage til den maskine, den kom fra.

Når vektor-programmet og moder-ormen igen kom i kontakt med hinanden, blev der overført tre filer til den ny-inficerede maskine. Det drejer sig om vektor-programmets kildetekst, og objekt filer til selve orme-programmet, i en vax udgave og en Sun-3 udgave. Der er her at begrænsningen til Vaxe og Sun-3'ere kommer ind i billedet. Ormen kan naturligvis ikke overføre sine egne kildetekster, da den jo så ville være triviel at stoppe.

Efter at filerne var overført, udførte vektor-programmet en shell, som læste kommandoer fra nettet. Moder-ormen sendte så de nødvendige kommandoer til at lænke orme-programmet sammen, både Vax og Sun-3 udgaven. Højst en af dem vile lykkes, og det resulterende program blev udført. Hermed var den nye maskine blevet inficeret.

Resultatet

Resultatet af den fremgangsmåde, der er blevet skitseret er naturligvis en eksponentiel vækst i antallet af orme, og at mange maskiner ville blive inficeret flere gange.

Der var lagt en del arbejde i at forhindre at en maskine blev inficeret for mange gange. Ormen ville med jævne mellemrum checke om der var andre udgaver af ormen på den samme maskine, og hvis der var det, ville de slå plat og krone (bogstavelig talt) om hvem der skulle dø. Dog var det kodet sådan at en ud af syv orme ville nægte at dø, sandsynligvis for at forhindre en falsk orm i at stoppe det hele, og selv om en orm tabte, ville den dog alligevel fortsætte i op til 15 minutter endnu. Resultatet var at mange maskiner blev oversvømmet med orme, og på et eller andet tidspunkt ville ormene ganske simpelt forbruge alle systemets ressourcer. Det gik specielt ud over foretrukne ofre som f.eks. gateways.

Ormen var meget omhyggelig med at beskytte sig selv og sine data mod opdagelse og undersøgelse. Den havde ingen filer i filsystemet, undtagen i en kort periode onkring inficeringen, og alle data og tek-

ststrenge i programmet var kodet, og blev først afkodet når de skulle bruges. For at undgå at enkelte processer skulle tiltrække sig for meget opmærksomhed, rettede ormen i sin kommando-linie argumenter, så den lignede en almindelig shell, og den skiftede med jævne mellemrum sit process nummer ud, for ikke at akkumulere et for stort cpu-forbrug.

En ting som ormen ikke kunne kamouflere, var den meget store aktivitet som programmerne *telnet*, *sendmail* med 'debug' aktiveret, og *rexecd* (remote execution daemon) udviste, og det var da også sådanne observationer, som ledte folk på sporet af ormen.

Lidt kronologi

De første kendte spor af ormen stammer fra den 2. november omkring klokken 18 (alle tider i amerikansk vestkyst tid), og omkring klokken 22 var flere systemer ubrugelige på grund af belastningen fra ormen. Den første dokumenterede erkendelse af at en orm/virus var løs på internettet er fra omkring klokken 23.30, i form af et elektronisk brev, som mange dog ikke fik, fordi de havde koblet sig af nettet i panik. Denne besked indholdt ellers meget vigtige oplysninger om hvordan ormen kunne begrænses i sin vækst.

Omkring klokken 3 om morgenen, den 3. november, har Berkeley et patch til *sendmail* klar, og poster det på nettet. Flere folk blive i løbet af morgenen opmærksomme på *finger*-angrebet, men af forskellige årsager bliver det ikke kendt før om eftermiddagen. Et officielt patch fra Berkeley bliver postet omkring klokken 19.

I løbet af natten mellem den 3. og 4. dekompilerer folkene i Berkeley og en gruppe på MIT ormen. De bytter kode et par gange i nattens løb. Klokken 6 om morgenen, den 4., er ormen stort set helt disassemblet, og næsten helt dekompileret. Omkring middagstid har både MIT gruppen og Berkeley gruppen dekompileret ormen til omkring 3200 linier C-kode.

Konsekvenserne

De kortsigtede konsekvenser af internet-ormen var naturligvis, at der gik panik i folk, da de så deres systemer opføre sig helt tosset. Mange

lukkede deres systemer ned, og afbrød deres forbindelse til internettet. Meget store dele at internettet blev koblet af, og flere vigtige meddelser om hvordan ormen angreb blev forskinket i op til to dage.

De mere langsigtede konsekvenser er sværere at vurdere. Internettet kom meget hurtigt på benene igen, mest takket være den indsats som de to grupper i Berkeley og på MIT ydede, idet de meget hurtigt fandt ud af hvordan ormen virkede, og hvordan man kunne stoppe den.

En meget interessant observation er, at ormen sandsynligvis ville være blevet stoppet endnu hurtigere, hvis mange maskiner ikke var blevet lukket i panik. Internettet var godt nok det middel, som ormen brugte til at sprede sig over, men det var også det våben, som stoppede den. Hurtigt kommunikation mellem geografisk spredte grupper var essentiel for den hurtige forståelse for hvad slags program ormen var, og hvordan den skulle stoppes. Mange vigtige informationer strandede, fordi vigtige gateways blev koblet af internettet.

Litteratur

[1] Eric Allman, *Worming my way...*, *UNIX Review*, vol. 7, no. 1, Januar 1989.

[2] Donn Seeley, *A Tour of the Worm*, Department of Computer Science, University of Utah.

[3] Eugene H. Spafford, *The Internet Worm Program: An Analysis*, Department of Computer Science, Purdue University, Purdue Technical Report CSD-TR-823.

Uddrag af rapport om Internet-ormen

Af *Dennis Meredith*
Cornell Chronicle
Cornell University

Denne artikel er et resumé af en rapport om skyldsspørgsmålet i forbindelse med Internet-ormen. Internet-ormen blev udførligt omtalt i DKUUG-Nyt nr. 20. Artilken i DKUUG-Nyt nr. 20 skal dog suppleres med nogle få oplysninger, før man kan få det fulde udbytte af resuméet.

Det var stort set fra starten kendt, at en studerende ved Cornell universitetet, Robert T. Morris, sandsynligvis var ophavsmanden til ormen, uden der dog var blevet fremlagt nogle endegyldige beviser for det. Sådanne foreligger efter min bedste viden endnu ikke offentligt.

Rapporten er udarbejdet af et lærer-panel på Cornell universitetet, og skal således ikke tages hverken som et upartisk, eller et juridisk gyldigt dokument. Resuméet af rapporten blev bragt i "The Cornell Chronicle," som er universitetets administrations blad.

Resten af denne artikel er selve resumét, det indledende afsnit undtaget, idet det er tilføjet at Manny Farber, som sendte resuméet ud på USENET.

The Cornell Chronicle is the Administration's organ. As such, their coverage of the Bob Morris report may be relatively onesided, but since they got the report in advance, they summarized it. I'll put the last paragraph right here: Copies of the report are available from the Office of the Vice President for Information Technologies, 308 Day Hall, [area code 607] 255-3324.

CORNELL PANEL CONCLUDES MORRIS RESPONSIBLE FOR COMPUTER WORM

(By Dennis Meredith, Cornell Chronicle, 4/6/89)

Graduate student Robert Tappan Morris Jr., working alone, created and spread the "worm" computer program that infected computers nationwide last November, concluded an internal investigative commission appointed by Provost Robert Barker.

The commission said the program was not technically a "virus"—a program that inserts itself into a host program to propagate—as it has been referred to in popular reports. The commission described the program as a "worm," an independent program that propagates itself throughout a computer system.

In its report, "The Computer Worm," the commission termed Morris's behavior "a juvenile act that ignored the clear potential consequences." This failure constituted "reckless disregard of those probable consequences," the commission stated.

Barker, who had delayed release of the report for six weeks at the request of both federal prosecutors and Morris's defense attorney, said, "We feel an overriding obligation to our colleagues and to the public to reveal what we know about this profoundly disturbing incident."

The commission had sought to determine the involvement of Morris or other members of the Cornell community in the worm attack. It also studied the motivation and ethical issues underlying the release of the worm.

Evidence was gathered by interviewing Cornell faculty, staff, and graduate students and staff and former students at Harvard University, where Morris had done undergraduate work.

Morris declined to be interviewed on advice of counsel. Morris had requested and has received a leave of absence from Cornell, and the university is prohibited by federal law from commenting further on his status as a student.

The commission also was unable to reach Paul Graham, a Harvard graduate student who knew Morris well. Morris reportedly contacted Graham on Nov. 2., the day the worm was released, and several times before and after that.

Relying on files from Morris's computer account, Cornell Computer Science Department documents, telephone records, media reports, and technical reports from other universities, the commission found that:

• Morris violated the Computer Sciences Department's expressed policies against computer abuse. Although he apparently chose not to attend orientation meetings at which the policies were explained,

Morris had been given a copy of them. Also, Cornell's policies are similar to those at Harvard, with which he should have been familiar.

- No member of the Cornell community knew Morris was working on the worm. Although he had discussed computer security with fellow graduate students, he did not confide his plans to them. Cornell first became aware of Morris's involvement through a telephone call from the Washington Post to the science editor at Cornell's News Service.
- Morris made only minimal efforts to halt the worm once it had propagated, and did not inform any person in a position of responsibility about the existence or content of the worm.
- Morris probably did not indent for the worm to destroy data or files, but he probably did intend for it to spread widely. There is no evidence that he intended for the worm to replicate uncontrollably.
- Media reports that 6,000 computers had been infected were based on an initial rough estimate that could not be confirmed. "The total number of affected computers was surely in the thousands," the commission concluded.
- A computer security industry association's estimate that the worm caused about $96 million in damage is "grossly exaggerated" and "self-serving."
- Although it was technically sophisticated, "the worm could have been created by many students, graduate or undergraduate ... particularly if forearmed with knowledge of the security flaws exploited or of similar flaws."

The commission was led by Cornell's vice president for information technologies, M. Stuart Lynn. Other members were law professor Theodore Eisenberg, computer science Professor David Gries, engineering and computer science Professor Juris Hartmanis, physics professor Donald Holcomb, and Associate University Counsel Thomas Santoro.

Release of the worm was not "an heroic event that pointed up the weaknesses of operating systems," the report said. "The fact that UNIX ... has many security flaws has been generally well known, as indeed are the potential dangers of viruses and worms."

The worm attacked only computers that were attached to Internet, a national research computer network and that used certain versions of

the UNIX operating system. An operating system is the basic program that controls the operation of a computer.

"It is no act of genius or heroism to exploit such weaknesses," the commission said.

The commission also did not accept arguments that one intended benefit of the worm was a heightened public awareness of computer security.

"This was an accidental byproduct of the event and the resulting display of media interest," the report asserted. "Society does not condone burglary on the grounds that it heightens concern about safety and security."

In characterizing the action, the commission said, "It may simply have been the unfocused intellectual meanderings of a hacker completely absorbed with his creation and unharnessed by considerations of explicit purpose or potential effect."

Because the commission was unable to contact Graham, it could not determine whether Graham discussed the worm with Morris when Morris visited Harvard about two weeks before the worm was launched. "It would be interesting to know, for example, to what Graham was referring to in an Oct. 26 electronic mail message to Morris when he inquired as to whether there was 'Any news on the brilliant project?'" said the report.

Many in the computer science community seem to favor disciplinary measures for Morris, the commission reported.

"However, the general sentiment also seems to be prevalent that such disciplinary measures should allow for redemption and as such not be so harsh as to permanently damage the perpetrator's career," the report said.

The commission emphasized, that this conclusion was only an impression from its investigations and not the result of a systematic poll of computer scientists.

"Although the act was reckless and impetuous, it appears to have been an uncharacteristic act for Morris" because of his past efforts at Harvard and elsewhere to improve computer security, the commission report said.

Of the need for increased security on research computers, the commission wrote, "A community of scholars should not have to build walls as high as the sky to protect a reasonable expectation of privacy, particularly when such walls will equally impede the free flow of information."

The trust between scholars has yielded benefits to computer science and to the world at large, the commission report pointed out.

"Violations of that trust cannot be condoned. Even if there are unintended side benefits, which is arguable, there is a greater loss to the community as a whole."

The commission did not suggest any specific changes in the policies of the Cornell Department of Computer Science and noted that policies against computer abuse are in place for centralized computer facilities. However, the commission urged the appointment of a committee to develop a university-wide policy on computer abuse that would recognize the pervasive use of computers distributed throughout the campus.

The commission also noted the "ambivalent attitude towards reporting UNIX security flaws" among universities and commercial vendors. While some computer users advocate reporting flaws, others worry that such information might highlight the vulnerability of the system.

"Morris explored UNIX security amid this atmosphere of uncertainty, where there were no clear ground rules and where his peers and mentors gave no clear guidance," the report said.

"It is hard to fault him for not reporting flaws that he discovered. From his viewpoint, that may have been the most responsible course of action, and one that was supported by his colleagues."

The commission report also included a brief account of the worm's course through Internet. After its release shortly after 7:26 p.m. on Nov. 2, the worm spread to computers at the Massachusetts Institute of Technology, the Rand Corporation, the University of California at Berkeley and others, the commission report said.

The worm consisted of two parts—a short "probe" and a much larger "corpus." The probe would attempt to penetrate a computer, and if successful, send for the corpus.

The program had four main methods of attack and several methods of defense to avoid discovery and elimination. The attack methods exploited various flaws and features int he UNIX operating systems of the target computers. The worm also attempted entry by "guessing" at passwords by such techniques as exploiting computer users' predilections for using common words as passwords.

The study's authors acknowledged computer scientists at the University of California at Berkeley for providing a "decompiled" version of the worm and other technical information. The Cornell commission also drew on analyses of the worm by Eugene H. Spafford of Purdue University and Donn Seeley of the University of Utah.

GNU projektet

Af *René Seindal*
seindal@diku.dk

Vi bringer her på opfordring en artikel om hvad GNU projektet er for noget, da der tilsyneladne er mange, som ikke ved hvad det er. Artiklen vil dels tage udgangspunkt i mine egne erfaringer med GNU programmellet, og dels i den seneste udgave af GNUs nyhedblad, "GNU Bulletin," fra januar i år.

Indrykket tekst er ordrette uddrag af "GNU Bulletin." Da disse afsnit er på engelsk, skulle forveksling ikke være mulig.

Hvad er GNU

GNU er en projekt, igangsat af Free Software Foundation (FSF), med det formål at udvikle og distribuere gratis programmel. Med deres egne ord er ideen med Free Software Foundation:

> The Free Software Foundation is dedicated to eliminating restrictions on copying, redistribution, understanding and modification of computer programs. We do this by promoting the development and use of free software in all areas of computer use. Specifically, we are putting together a complete integrated software system called "GNU" (GNU's Not Unix) that will be upward compatible with Unix. Some large parts of this system are already working and we are distributing them now.
>
> The word "free" in our name refers to two specific freedoms: first, the freedom to copy a program and give it away to your friends and co-workers; second, the freedom to change a program as you wish, by having full access to source code. Furthermore, you can study the source and learn how such programs are written. You may then be able to port it, improve it, and share your changes with others.

> Other organizations distribute whatever free software happens to be available. By contrast, FSF concentrates on development of new free software, building toward a GNU system complete enough to eliminate the need to purchase a proprietary system.

Selvom programmellet er gratis, og der derfor ikke er nogen service, er man dog ikke helt på herrens mark, idet:

> After we create our programs, we continually update and improve them. We release between 2 and 20 updates a year, for various programs. Doing this while developing new programs takes a lot of work, so any donations of pertinent source code and documentation, machines, labor or money are always appreciated.

Service og support på programmellet bygger således på brugernes egne rettelser og tilføjelser. Det forventes at man, hvis man retter fejl eller forbedre programmerne, sender sine rettelser tilbage til Free Software Foundation, hvor de så måske finder sin vej ind i distributionen. Man skal ikke regne med at blive betalt for sit arbejde, idet det forventes at man, da man jo har haft glæde af det Free Software Foundation har givet en gratis, vil være villig til at givet noget tilbage.

Personerne

Der er efterhånden mange personer involveret i Free Software Foundation og GNU, men nogle er mere fremtrædende end andre.

Den utvivlsopmt vigtigste person i Free Software Foundation er Richard Stallman, tidligere ved MIT, idet han har stiftet Free Software Foundation. Stallman har selv bidraget til GNU med programmel som GNU Emacs, som er en udvidbar editor, og GNU CC, som er en ANSI kompatibel C oversætter. Derudover arbejder han som koordinator for alle de personer, der skriver programmel og dokumentation til GNU. Til sin hjælp har han Leonard Tower, som bl.a. tager sig af meget af det administrative arbejde.

Af de mange personer, der bidrager med programmel og dokumentation, kan nævnes Michael Tiemann, fra Stanford University, som har

lavet 'G++,' der er GNUs C++ oversætter; Roland McGrath, som har lavet GNUs make og som arbejder på et ANSI kompatibelt C bibliotek; Doug Lea, som har skrevet 'libg++,' et C++ bibliotek; samt mange andre, som oftest arbejder på mindre separate projekter.

Filosofien

Drivkraften i hele projektet er idealisme, da der jo per definition ikke er penge i at udvikle gratis programmel. De bagved liggende tanker er blevet udtrykt meget præcist af Doug Lea, i en artikel på USENET, som det følgende er et uddrag af:

> I am, primarily, a teacher in a liberal arts college. As such, I stand for the 'free' dissemination of ideas. Historically, (please forgive any botching of historical facts to suit my needs, but that's what history is for!) the main tool by which intellectual property has been allowed to be widely disseminated (read 'taught') while at the same time both crediting originators, and protecting the works from corruption, mis-attribution, and so on, has been the notion of Copyright. For these reasons, the introduction of copyright laws is widely considered to have been an important step in accelerating intellectual and scientific progress.
>
> Sadly, in the science of computing, this solution has not stood up well. While, in many disciplines, the price of a copyrighted work to be used for study is well within the reach of those who could best benefit from it (e.g., a copy of "War and Peace" might be $5, or even $50, but not $50,000), the economics of computing have, for the most part, priced copyrighted software out of the reach of students (and most others). Most readers would agree that the study of high-quality existing programs is among the better methods for learning about the art of programming. These days, one cannot legally show, discuss, and teach from, say, Unix or Lotus source code.
>
> I believe that Stallman's notion that the economics of copyright can be separated from its role in the protection

> and propagation of intellectual property is as good a solution to this dilemma as we are likely to get. There are many of us, especially those of us in academe, who are actually very pleased to devote some time and effort to writing software without any direct monetary compensation. For all sorts of reasons. (For example, in my case, with 'libg++,' as a means to further investigate the pragmatics of object-oriented programming and so on. Or maybe it's just incorrigible hacking. Whatever.)
>
> Now I, and many others, I suspect, are not terribly worried about maintaining proper authorship credit, etc., of such work. The reason that the GNU License Agreement is attractive is mainly that it keeps accessible the work that I intended to be accessible, but also generally offers all other benefits that Copyright engenders, but that the mere act of placing work in the 'public domain' would not.

GNUs copyleft

Free Software Foundation forsøger at beskytte deres programmel mod misbrug, herunder salg, med noget de kalder en "copyleft." Hovedformålet med en "copyleft" er at sikre at det programmel, som forfatteren ønskede skulle være gratis, vil vedblive at være gratis. Free Software Foundations egen forklaring følger:

> The simplest way to make a program free is to put it in the public domain. Then people who get it from sharers can share it with others. But bad citizens can also do what they like to do: sell binary-only versions under typical don't-share-with-your-neighbor licenses. They would thus enjoy the benefits of the freeness of the original program while withholding these benefits from the users. It could easily come about that most users get the program this way, and our goal of making the program free for *all* users would have been undermined.
>
> To prevent this from happening, we don't normally place GNU programs in the public domain. Instead, we protect

> them by what we call "copylefts." A copyleft is a legal instrument that makes everybody free to copy a program as long as the person getting the copy gets with it the freedom to distribute further copies, and the freedom to modify their copy (which means that they must get access to the source code). Typical software companies use copyrights to take away these freedoms; now we software sharers use copylefts to preserve these freedoms.

Projektets status

Indtil nu har vi mest set på hvad Free Software Foundation og folkene bagved ønsker at lave. Hvad har de så opnået? Indtil nu har GNU produceret en meget stor del programmel. Det meste af det er programmørorienteret. Der er ingen spread-sheets og regne-programmer.

GNU Emacs er nok det mest kendte GNU produkt. Den er skrevet af Richard Stallman, og har i tidens løb (siden 1984) gennemgået en omfattende udvikling. Den er i dag nok den mest omfattende editor der findes. Den har specialle 'modes' til mange forskellige typer tekst, og kan bl.a. lave automatisk indrykning af C og lisp programmer. Stort set hele Emacs' funktionalitet er implementeret i Emacs' egen indbyggede lisp variant, og det er derfor fantastisk let at lave nye funktioner, eller modificere de allerede definerede (som man naturligvis har kildetekster til). Emacs kan stort set alt (ellers kan man få den til det), så nogle folk starter efter sigende en Emacs om morgenen, og forlader den ikke før de har fri. Emacs kan vist køre på stort set alle 32 bit maskiner, og den er i hvert fald portet til hundreder forskelige arktekturer. Emacs kommer med en on-line manual, som udskrevet fylder 300 sider. Emacs Lisp er dokumenteret med et 400 siders dokument.

GNU har efterhånden et komplet sæt programmer til programudvikling i C. Det inkluderer en portabel, optimerende ANSI C oversætter, som er både hurtigere og laver mere effektiv kode end de fleste kommercielt tilgængelige oversættere. GNU CC kommer med en lænker, som bl.a. kan angive uløste reference med linie henvisning til kildeteksten. Der er bison, som er GNUs yacc (?) og flex, som er en "fast lex." Begge har udvidet funktionalitet i forhold til deres Unix modstykker,

og er samtidigt hurtigere. GNU har en version af make, som er den mest omfattende make jeg nogen sinde har set.[2] GDB er GNUs debugger, som arbejder på kildetekst-niveau, og man behøver derfor stort set ikke kende til maskinsprogs-programmering for at kunne bruge den. Til sidst, så indeholder sættet programmer som size, nm, mm., fordi GNUs oversættere og lænkere ikke vil arbejde med objektfiler i COFF-format.

G++ er GNUs C++ oversætter, som i den nuværende udgave implementere en større del af sproget end nogen anden C++ oversætter. G++ benytter GNU CCs kodegenereringsmodul, og kan derfor foretage mange af de samme optimeringer. Sammen med G++ kommer libg++, som er et meget omfattende C++ bibliotek, som indeholder klasser til alt hvad hjertet kan begære.

Af resten af GNUs programmel kan nævnes GNUs diff og grep, som efter sigende er verdens hurtigste. GNU leverer også programmer som ls, tar, sed, et interaktivt plot-program, en PostScript fortolker, flere netnews programmer, og meget mere.

Programmellets kvalitet

Programmernes kvalitet er til enhver tid varierende. Flere af de nyere programmer er af Free Software Foundation blevet mærket som beta-test distributioner, og programmerne kan være meget ustabile. På dette punkt adskiller Free Software Foundation sig fra de fleste kommercielle firmaer, idet de gerne frigiver programmel i *meget* tidlige faser i udviklingsforløbet. Det giver fremtidige brugere mulighed for at bedrage til den videre udvikling, og medvirker til at øge de færdige programmers kvalitet.

Generelt kan det siges at med tiden er de fleste af de ældre programmer blevet meget stabile, heriblandt Emacs, GNU CC, bison, flex, og mange af programmeringsværktøjerne. Jeg bruger selv mange af programmerne i det daglige, og oplever kun få problemer, som mest opstår på de områder, hvor GNUs programmel ikke er bagudkompatibelt med det tilsvarende Unix programmel.

[2] Der er vist ikke det den ikke kan, måske med undtagelse af at lave kaffe, og punktsvejse under vandet.

Det generelle mønster er, at de tidlige udgaver er praktisk taget ubrugelige til daglig brug. De sendes ud for at få foreslag, fejlreportinger og rettelser tilbage og man kan deltage i arbejdet, hvis man er interesseret. De senere udgaver er gerne ved at stabilisere sig, og kan bruges til almindelig brug, men man skal dog være forberedt på at støde ind i fejl. Efterhånden som fejlene bliver rettet, bliver programmerne dog mere og mere stabile.

Distribution af GNU programmel

GNU programmel distribueres på flere måder. Da programmerne ifølge deres copyleft må kopieres frit, kan man til enhver tid blot få et kopi af naboens eksemplar. Hvis man ikke kender nogen, som kan eller vil forsyne en med det man ønsker, kan man bestille det på bånd fra Free Software Foundation, EUUG eller i visse tilfælde fra DKUUG.

Hvordan man bidrager til GNU

Hvis man har noget, som måske kan have interesse for Free Software Foundation, kan man skrive til `gnu@prep.ai.mit.edu` og give dem en beskrivelse af hvad det er man har at tilbyde. Hvis man blot ønsker at lave noget arbejde for dem, kan man også skrive og bede om deres "wish list" over de ting, som de godt kunne tænke sig, men endnu ikke har nogle til at lave.

Annoncer i DKUUG-Nyt

Det er muligt at få bragt annoncer i DKUUG-Nyt. Bladet udkommer for tiden i et oplag på 600 eksemplarer, dvs. at man kun betaler lidt over 1 krone pr. eksemplar for en halvsides-annonce. Endvidere bliver hvert eksemplar af bladet normalt læst af mere end én person.

Prisen for en halv side er pt. på kr. 500,– og redaktionen forbeholder sig retten til at anbringe annoncerne hvor *den* har lyst—vi vil dog naturligvis såvidt muligt anbringe annoncerne på fremtrædende pladser, men såvel forsiden som bagsiden er annoncefrit område.

Annoncerne skal indleveres til den almindelige deadline i reproklar tilstand. Vi vil normalt påskønne at få et praj på forhånd om, at der er materiale på vej. Redaktionens addresse findes på side 2.

Unix standardiseringsorganer og interesseorganisationer

Af *Kim F. Storm*
Texas Instruments A/S

De mange organer og organisationer, der deltager i udviklingen og standardisering af Unix-styresystemet er nu kommet under en fælles paraply, hvilket yderligere vil styrke Unix-systemets forankring på markedet for administrative edb-systemer.

Åbne systemer

Den væsentligste årsag til Unix-systemets voksende udbredelse er indeholdt i begrebet åbne systemer, dvs. systemer der ikke er bundet til en bestemt leverandørs specifikke datamat-arkitektur. Unix udmærker sig her ved at give programmer en helt ensartet grænseflade til maskinens ydre enheder som f.eks. pladelagre, bånd og terminaler, til filer og til kommunikation mellem programmer. Dette gør at programmer skrevet til et Unix-system på en maskine fra leverandør X stort set uden ændringer kan flyttes over på en maskine fra leverandør Y og bringes til at køre der.

For en køber af edb-systemer er der mange fordele ved at anskaffe et åbent system fremfor et lukket, leverandørspecifikt system. Dels er det muligt for en køber at sammenligne leverandørerne udfra en fælles målestok, dvs. primært se på om de tilbudte programmer opfylder de stillede krav, dels vil der ikke kræves store nyinvesteringer i programmer, hvis man får behov for at anskaffe en større maskine eller man måske ønsker at skifte maskinleverandør, og endelig vil der også ofte være et større udbud af gode programmer, der kan løse ens opgave, da de åbne systemer giver programleverandører mulighed for at levere deres programmer til mange forskellige leverandørers maskiner.

X/Open

En række europæiske edb-leverandører indså i 1984 at selv det at flytte programmer mellem leverandørens egen model A og B krævede enorme ressourcer. Faktisk regnede man ud at omkring 90% af al tid gik til at vedligeholde eksisterende programmer på leverandørens forskellige maskiner, mens kun 10% blev brugt på udvikling af nye (og tidssvarende) programmer. Alt dette fordi systemerne var forskellige fra leverandør til leverandør, og i mange tilfælde også forskellige fra model til model hos den enkelte leverandør.

Disse leverandører dannede organisationen X/Open, som opstillede en række specifikationer for hvilke værktøjer man bør bruge for at sikre at programmer nemt (og dermed billigt) kan flyttes mellem maskiner af forskelligt fabrikat eller model. Da Unix var og stadig er det eneste styresystem, der uden store problemer kan bringes til at køre på enhver datamat uanset dens opbygning, blev Unix naturligt nok udpeget som styresystemet, der skulle danne basis for udviklingen mod åbne systemer.

Udover Unix valgte man i starten en række specifikke databaseværktøjer (ISAM) og programmeringssprog (C, MF-COBOL, FORTRAN), som man samlede i en såkaldt "Portability Guide 1", dvs. en specifikation for hvilke værktøjer der bør være til rådighed i det åbne system. Siden har man i 2. og 3. udgave af guiden udvidet mængden af værktøjer, f.eks. med sproget PASCAL og databaseværktøjet SQL, således at man har et meget komplet sæt af værktøjer til opbygning af administrative edb-systemer.

X/Open kredsen er også vokset, således at også de største amerikanske og japanske edb-leverandører er kommet med. Senest er også de to interesseorganisationer UNIX International og Open Software Foundation (OSF) blevet optaget i X/Open. Begge disse organisationer står bag udviklingen og markedsføring af Unix (eller Unix-kompatible) styresystemer, og gennem optagelsen i X/Open har de begge tilkendegivet at de vil følge de anvisninger for åbne systemer som X/Open har givet.

UNIX International

Unix er udviklet på et af den amerikanske telefongigant AT&Ts laboratorier, og det er stadig AT&T der har rettighederne til Unix-operativsystemet. En lang række leverandører af Unix-systemer har valgt at bakke op om AT&T som primus motor i videreudviklingen af Unixsystemet, og de har dannet organisationen UNIX International som forum for udveksling af synspunkter og ideer indbyrdes, med AT&T og udadtil. Dvs. UNIX International står med AT&T i spidsen for den udviklingslinie, der bygger videre på det eksisterende, velfungerende og gennemtestede Unix-system.

Open Software Foundation

En anden gruppe af leverandører har ikke været helt så begejstrede for hvad de har set som AT&Ts monopol på Unix-markedet. De har dannet organisationen Open Software Foundation (OSF) med det mål at kunne levere et Unix-kompatibelt styresystem, som ikke ejes af AT&T. OSF står altså for en udviklingslinie, der begynder på en frisk, må igennem en formodentlig lang udviklings- og testperiode før et stabilt og brugbart system er til rådighed, men så på længere sigt måske vil have et teknologisk mere moderne system end AT&Ts. Indtil da vil man dog kun kunne levere systemer, der stadig bygger på AT&Ts Unix-udgave.

Unix-Standardisering

X/Opens specifikationer for hvordan man sikrer maksimal flytbarhed af programmer mellem åbne systemer er baseret på eksisterende, tilgængelige værktøjer. Disse værktøjer følger i videst muligt omfang internationale standarder, men på områder hvor der ikke findes en international standard anvender man såkaldte "de fakto" standarder, dvs. standarder som ikke er formelt vedtagne i f.eks. ISO-regi, men som er vidt udbredte og akcepterede.

Selv om mange varianter af Unix har set dagens lys gennem årene, er grundstammen i systemerne stadig den samme, og holder man sig indenfor de rammer som er afstukket af X/Open, er Unix i dag en veletableret "de fakto" standard.

De mange varianter af Unix har naturligt givet anledning til en del tvivl om, hvorvidt man nogensinde ville komme frem til en fælles international standard for Unix, men dels arbejder AT&T ihærdigt på at samle de forskellige varianter i et fælles system (kendt som Unix System V release 4), dels arbejder man nu i IEEE- og ISO-regi på en formel international standard for åbne systemer under navnet Posix, der naturligvis har Unix som forbillede. Posix kommer kun til at afvige fra eksisterende Unix-systemer i detaljer, da et af målene for arbejdet er i videst muligt omfang at sikre at eksisterende Unix-programmer vil fortsætte med at virke under Posix.

Når Posix standarderne foreligger i løbet af de nærmeste år vil X/Open naturligt medtage disse i deres "Portability Guides", og både UNIX International og OSF vil derefter hurtigt kunne justere deres systemer til at opfylde standarderne og leverandørerne vil følge efter med.

Der er imidlertid ingen grund til at vente på at denne formelle standardisering er på plads før man satser på et åbent system ved anskaffelsen af et nyt edb-system. Næsten uanset hvor store ændringer Posix vil foreskrive i forhold til de eksisterende Unix-systemer vil de mulige nødvendige tilretninger af programmel skrevet til Unix være helt ubetydelige sammenlignet med det arbejde det vil være at flytte tilsvarende programmel fra et leverandørspecifikt, lukket system til et åbent Posix/Unix-baseret system.

Boganmeldelse — Objekt-orienteret programmering

Af *René Seindal*

Anmeldelse af: Bertrand Mayer — *Object-oriented Software Construction*, Prentice-Hall, 1988, 500 sider. ISBN 0-13-629031-0.

Bertrand Mayer er en af de store inden for objekt-orienteret programmering. Ud over at have skrevet denne bog, har han designet sproget Eiffel, som er det gennemgående eksempel i bogen.

Bogen en primært en introduktion til objekt-orienteret system konstruktion. Den er en uformel, praktisk orienteret gennemgang af ideerne og teknikkerne bag objekt-orienteret system konstruktion. Bogen fungerer samtidig som en introduktion til Eiffel.

Bogens opbygning

Bogen er delt op i fire dele. Den første del omhandler dels de generelle problemstillinger og principper, som leder frem til metoden objekt-orienteret programmering, og dels de fundamentale begreber i objekt-orienterede udviklingsmetoder.

Den anden del af bogen, den største, gennemgår de teknikker man har til rådighed ved objekt-orienteret system konstruktion. Den dækker ting som objekter og klasser, parametriserede klasser, design af klasse grænseflader, "inheritance" og dynamisk lager-administration. Eiffel bliver brugt som det gennemgående eksempel hele bogen igennem. Denne del af bogen introducerer de forskellige dele af Eiffel efterhånden som de tilsvarende grundlæggende begreber gennemgås.

Den tredie del omhandler objekt-orienterede metoder i andre programmerings miljøer end Eiffel. Det dækker bl.a. muligheder for at benytte objekt-orienterede metoder i Pascal, C, Ada, Fortran, mm., samt en oversigt over hvilke andre objekt-orienterede programmeringssprog der findes.

Endelig består den fjerde del af flere appendicies, om bl.a. Eiffels standard biblioteker og grammatik.

Bogens anvendelighed

Bogen er helt klart en undervisningsbog, og må vurderes som sådan. Den er dels en lærebog i Eiffel, og dels en grundbog i objekt-orienterede teknikker og metoder.

Den virker i begge henseender meget vellykket. De enkelte begreber og teknikker introduceres i en logisk og velgennemtænkt rækkefølge, fulgt af de tilsvarende faciliteter i Eiffel. Bogen indeholder flere gennemgående eksempler, som raffineres og forbedres efterhånden som flere og flere nye begreber indføres, hvilket er en storartet hjælp til forståelse af hvorfor disse nye begreber er nødvendige. Den komplette kode til alle eksemplerne findes i et appendiks.

En god ting er den udstrakte brug af diagrammer og program eksempler til at illustrere begreberne med. Tegninger og eksempler siger ofte mere end mange ord, og de bliver benyttet flittigt i bogen.

Eiffel

Det er en svært ting at vurdere et programmingssprog, som man ikke har prøvet, så det vil jeg undlade. Bertrand Mayer påstår selv at Eiffel både kan bruges, og bliver brugt med succes, til såvel undervisning som industriel systemudvikling.

En interessant facilitet i sproget et muligheden for at knytte formelle betingelse til en klasse, som man kan vælge at få kontrolleret på kørselstidspunktet. Det giver gode muligheder for at finde fejl, der opstår når to dele af et programkompleks ikke er enige om hvilke betingelser, der skal være opfyldt for at en given operation kan udføres. Den slags fejl er erfaringsmæssigt meget svære at finde, da begge dele af systemet hver for sig ser korrekte ud. Først når de to programdele kombineres, vil fejlen optræde. Med de faciliteter, som Eiffel tilbyder, er det muligt at få kontrolleret på kørselstidspunktet at en funktion kun kaldes i en lovlig tilstand, og at den, når den returnerer, efterlader systemet i en lovlig tilstand.

Konklusion

Personligt har jeg haft meget glæde af at læse Bertrand Mayers bog. Mine ideer om hvad objekt-orienteret programmering er, er blevet meget klarere efter at jeg har læst bogen. Den har helt klart givet mig en indsigt i nogle ting, som jeg ikke havde før. Ud over lidt programmering i C++, har mit kendskab til objekt-orienterede teknikker ikke været omfattende. Det føler jeg i høj grad at der er blevet gjort noget ved. På en måde har bogen givet en et lidt anderledes syn på hvordan programmer og systemer bør opbygges.

Til folk, der føler at objekt-orienteret programmering er et hul i deres viden, kan jeg absolut anbefale Bertrand Mayers bog. Hvis man mere konkret ønsker at lære at programmere i Eiffel, virker bogen også udmærket, uden at jeg dog vil ligge hovedet på blokken på det (jeg har som sagt aldrig prøvet at programmere i Eiffel).

Til de personer, som allerede føler sig trygge ved objekt-orienterede teknikker og metoder er bogen måske ikke helt så interessant. Den indeholder dog mange kloge ord om design og implementation af komplekse systemer, som måske vil være til glæde for selv viderekommende.

Under alle omstændigheder: til dem, som ikke føler at de er kloge nok, der er Bertrand Mayers bog ganske givet en god måde at blive lidt klogere på.

Oversigt over medlemsmøder afholdt i 1989

Dato	Sted	Tema
30/3	København	Systemudviklingsværktøjer (CASE).
19/4 †	København	UNIX i kontormiljøer.
31/5	Odense	Netværk.
1/6	Odense	Datasikkerhed.
14/6 †	København	Migration til UNIX.
28/9	København	Distribuerede systemer.
11/10 †	Provinsen	Industrielle systemer.
27–28/11	København	Årsmøde med netindslag.

De med † markerede møder var eftermiddagsarrangementer, som typisk var af ca. 2 timers varighed, placeret efter normal arbejdstid. Disse møder var gratis. De øvrige møder var heldagsarrangementer.