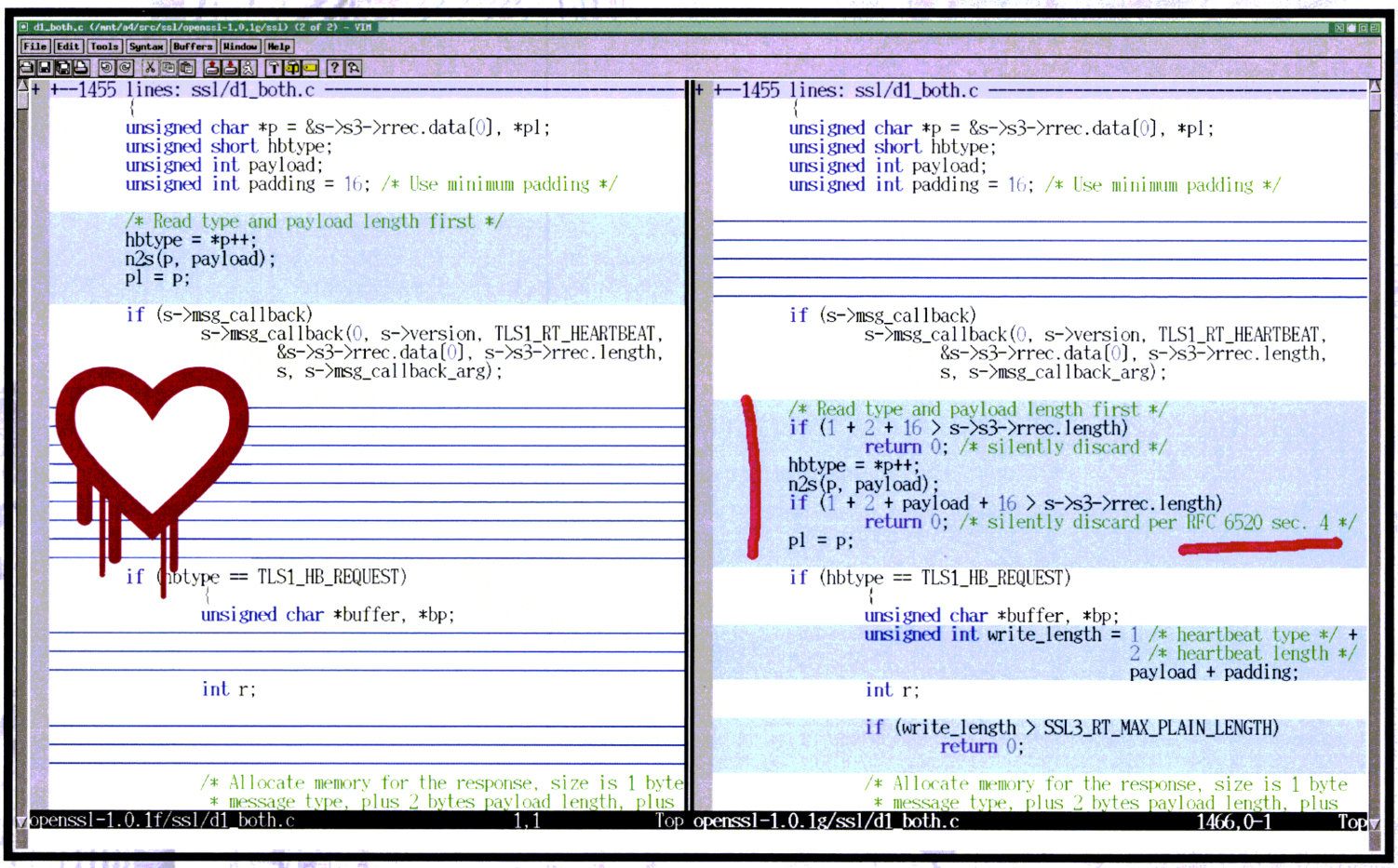
DKuug nr. 174 *Maj 2014*

NYT



Dansk Forum for Åbne Systemer

DKUUG - Unix Brugere (Linux/BSD/AIX) system administratorer

Community for IT-specialister og IT-interesserede.

DKUUG

Systemfejl

Hjerte- eller hjerneblødning?

Den danske ordbog siger om ***systemfejl:*** *fejl der skyldes forhold i et system, fx den måde det er sat sammen eller fungerer på; systematisk eller principiel fejl.*
"Fejlsikring er et ledelsesproblem. Når mange medarbejdere gentagne gange laver den samme fejl, er det en ***systemfejl****." (citat fra Dagens Medicin, nyhedsmagasin om sundhedssektoren, 2003).*

Heartbleed, den alvorlige sikkerhedsfejl i OpenSSL, offentliggjort i april (se s.8) var en systemfejl, forstået som manglende fejlsikring - for alle programmører kan jo lave en tanketorsk en gang imellem?

Hvad er de ledelsesfejl, som har ført til at OpenSSL er det mest udbredte system til krypteret trafik på Internettet?

Det korte svar er: Kompleksitet kræver øgede kompetencer og større ressourcer. Men hvordan har vi indtil nu kunnet klare kompleksiteten, og hvor stort er problemet?

Lad os se på Linux-kernen som et eksempel på en software-enhed med stigende kompleksitet. **Linux-1.0 er netop fyldt 20 år i marts** måned! Den er blevet voksen, så at sige, og er vokset fra 6 MB i udpakket tilstand til 622 MB i den nyeste version 3.14.2.

D.v.s. vi taler om en forøgelse på en faktor godt 100. Men man kan ikke bare forøge en stab til 100 mand. Der skal jo samarbejdes.

Kan mennesket rumme 100 gange et lille operativsystem i hovedet?

Forskellen fra 6 til 622 er umulig at rumme som andet end abstrakte tal. Vi ved fra astronomien, at vi mennesker har svært ved at forstå universets størrelsesforhold: Hubbles bevis for, at Andromeda tågen er en anden galaxe af samme slags som Mælkevejen, var kontroversiel i begyndelsen. Det var ufatteligt, at der kunne være noget udenfor vores "stjerneverden". Det er derfor man oftest illustrerer afstande i solsystemet i billeder, fx: solen som en tennisbold i midten af en rundkørsel og jorden som et 1mm. sandskorn 20-30 meter ude.

Vi har mulighed for at bygge store maskiner og for at rejse langt med vores moderne vidunderlige teknik, men grundlæggende arbejder vi mennesker ud fra vores kropsstørrelse, og megabygninger, megastrukturer og ***megaprogrammer*** kræver særligt værktøj.

Hvad slags værktøj kræver det at håndtere megaprogrammer?

SUN microsystems projektgrupper havde den tommelfingerregel, at et projekt højst måtte være 6 mand i 4 uger (citeret efter hukommelsen). En funktion burde iflg. lærebøgerne ikke være mere end en skærmfuld ... *(godt at vi har store skærme i dag eller hvad?)* og kunsten at lede et projekt består derfor i at designe det i mindre dele, som giver et overskueligt system. Hvis det kan tegnes på A3 uden grov forenkling, så er det godt.

Er vores computersystemer blevet for komplicerede?

Ja - men vi har brug for dem. Vi vil jo ikke undvære de nye smarte HW-devices, de avancerede programmer. For at klare komplexiteten kræves stadig større kamp for *modularisering, test, stress, multistage release.*

i den forbindelse er det interessant at høre, at Linus Torvalds ved flere lejligheder har nægtet at modtage (gode) drivere, som kom i store, uoverskuelige klumper - med andre ord, koden er ikke god bare fordi den er stabil og fungerer. Den skal være læselig.

Dertil kommer *konkurrence mellem flere teams.* Men hvis brugerne holder fast ved ét enkelt projekt, så ser vi ofte at kvaliteten forringes. Det var, i princippet, det, der skete med OpenSSL.

I "modern management" taler man om at *købe viden, hvor den er.* Men det er opskriften på fiasko (IC4, Polsag, Rejsekort etc.)

Tværtimod skal man opdyrke viden, videreudvikle kompetencer blandt de teams, som udgør en virksomhed eller et projekt.

Sikkerhed med OpenSSL

OpenSSL udviklerne var ikke grønne amatører, men der har været et problem med prioritering og modularisering og bemanding, måske afledt af sjusket projektledelse på grund af andre arbejdsopgaver eller livssituation. I så fald vil en ændring i bemandingen hjælpe, men det sker ikke altid automatisk i Open Source sammenhæng. OpenSSL kernegruppen er kun på 4 - 5 mand.

Der vil være magtkampe, prestigekampe, og en overdommer er ikke en realistisk løsning. Rigelig finansiering af alternative OpenSource projekter er den eneste løsning - selv om heller ikke det er uden problemer, fordi der vil kunne opstå nepotisme.

Skal Nordea, Danske Bank, Bank of England etc. betale for gen-udvikling af SSL?

Skal DKuug bruge sine ressourcer på støtte til et Open Source projekt som fx. LibreSSL?

■

I dette nummer ...

Redaktionen har experimenteret med fonte i dette nummer af bladet - ikke meget, for det var meningen at DKuug-NYT skal signalere kontinuitet. Her er to afsnit med forskellig font:

Men vi har været lidt bekymret over at bladet bruger en Helvetica font, hvor der ikke er forskel på I og l (stort i og ell). Et-taller kan man dog skelne fra I, 1og l bør være forskellige.

Men vi har været lidt bekymret over at bladet bruger en Helvetica font, hvor der ikke er forskel på I og l (stort i og ell). Et-taller kan man dog skelne fra I, 1 og l bør være forskellige.

Det var samme sætning med sans og bagefter en serif font, som har været den mest stabile (Open)Office font i 10-12 år: Nimbus Roman No. 9. Der må være en historie bag det navn - måske er der nogen af vores læsere, som vil oplyse læserne om det i næste nummer?

Vi har denne gang sat projektør på programmeringssproget Ruby, som kan en masse. I næste nummer skal vi så se på Ruby on Rails - web application framework. Det er vores erfaring fra kurser mv, at man lærer bedst ved en kombination af praksis og teori - og det prøver vi at leve op til med et eksempel med et tekst - adventurespil, hvor vores helt kan holde ting og sager i højre og venstre hånd og kan kæmpe med et uhyre.

Vi fortsætter med databaser, denne gang MySQL, som anvendes flittigt i Facebook. Artiklen giver en komprimeret påvisning af de utrolige krav, som visning af en FB side stiller til det bagvedliggende databasesystem.

Thomas Ammitzbøll-Bach har skrevet om den nye version af Ubuntu, som har profiler til forskellige typer hardwareplatforme. Det bliver spændende at se en smartphone med Ubuntu Mobil!

Mail til redaktionen: blad@dkuug.dk

Redaktionen

DKuug-NYT er medlemsblad for DKuug, foreningen for Åbne Systemer og Internet
Nr. 174 - Maj 2014

Udgiver:
DKUUG
Fruebjergvej 3
2100 København Ø
Tlf. 39 17 99 44
email: dkuugnyt@dkuug.dk
Redaktion:
Donald Axel (ansvarshavende)

Forsidecredits:
(redaktionen)

Design og layout:
DKUUG/Donald Axel med LibreOffice
Annoncer:
pr@dkuug.dk
Tryk:
Lasertryk i Aarhus
Oplag:
400 eksemplarer

Deadline for nr. 175: 10. Juni 2014

Medlem af Dansk Fagpresse
DKUUG-Nyt
ISSN-1395-1440

Vores møder og **foredrag** holdes - med mindre andet udtrykkeligt angives - på vores adresse:

DKUUG
SYMBION
Fruebjergvej 3
2100 København Ø

Hvis man kommer lidt før, er der tid til en snak på kontoret. DKUUG bor i en virksomhedsfarm, Symbion, hvor der er åbne døre indtil kl.18 eller 19 (afhængig af mødetidspunkt). Efter den tid har vi på foredragsaftener en vagt ved døren.

INDHOLD:

Arrangementer:

Onsdag d. 4 juni kl.19, Workshop om LibreOffice: hvordan man kan lave printfiler af pæn kvalitet med LibreOffice. Vanskeligheder med layout typer, cutmarks og illustrationer.

Fredag d. 13 Juni kl 18: Show and tell - vis og fortæl om et projekt, som du arbejder på.

DKUUG åbent hus onsdag d. 18 juni 2014 kl 18. på vores kontor i Symbion.

Andre arrangementer vil blive annonceret på web og via mail.

Gør-det-selv foredrag:

Vores kontor i Symbion giver mulighed for hands-on workshops, både dag og aften. Skriv til pr@dkuug.dk eller til bestyrelsen i DKUUG, bestyr@dkuug.dk, og hør om lokalet er ledigt den dag du vil arrangere et møde. Der er hurtig internetforbindelse, både wired og wireless. Er der større tilmelding, kan vi leje mødelokaler i Symbions mødested. Spørg os!

Chunky Bacon! - Eller en hurtig intro til Ruby.

Af David Askirk

David Askirk er instruktør hos SuperUsers a/s og arbejder med både hardware, software, programmeringssprog og systemadministration.

Ruby er endnu et af de fantastiske sprog som findes derude. Ruby er et objekt-orienteret sprog, der har den filosofi at det skal være sjovt at programmere. I denne artikel skal vi lave et spil. Samtidig med at vi laver spillet kommer vi også ind på nogle af de spændende ting som Ruby kan.

Hvis du tager alt koden i rækkefølge og sætter ind i en fil vil det virke. Så kan det være du siger "Jamen det er der jo ikke noget fantastisk i", men når vi kommer til nogle af de tricks som vil blive vist, så vil man undre sig.

Det første vi gerne vil i vores spil er at printe et navn. Jeg er ret vild med Nintendos *Legend of Zelda*, og vil gerne lave et tekstbaseret adventure spil, så lad os kalde det Text-Zelda eller. Telda!

Så lad os udskrive spillets navn:

```
#List 1
puts "Welcome to the magical world of Telda"
```

Vi udskriver en streng til skærmen med `puts` PutString. Det giver dette output:

```
Welcome to the magical world of Telda
```

Lad os springe ud i det med at lave en klasse der repræsentere vores spiller, kaldet Text-Link eller Tink.

```
#List 2
class Tink
end
```

Det var jo billigt sluppet. Nu har vi en tom klasse. Men lad os bruge noget af det andet som ruby kan. Nemlig at genåbne klasser.

```
#List 3
class Tink
    attr_accessor :name
end

tink = Tink.new
tink.name = "The hero Tink"
puts tink.name
p tink
```

Måske tænker man nu. "Jamen du har jo bare gen-deklaret klassen". Det er faktisk ikke rigtigt. Det er stadig den samme klasse. Så nu har vores klasse fået et felt, som hedder name, som der automatisk er blevet lavet get og set til, ved at bruge `"attr_accessor"` metoden i klassen.

Med det efterfølgende kode laver vi en ny instans af klassen med `"new"` metoden. Så sætter vi navnet og udskriver først navnet og derefter hele objektet.

Alle objekter i ruby har en "inspect" metode som laver en tekst repræsentation af objektet.

I dette tilfælde resulterer det i dette output:

```
The hero Tink
#<Tink:0x23bb948 @name="The hero Tink">
```

Men vores helt skal have mere. Alle helte har et inventory, hvori de kan have alverdens ting. Samtidig har de fleste helte også to

```
#List 4
class Tink
    attr_accessor :name
    attr_accessor :inventory, :lefthand, :righthand

    def initialize
        @inventory = []
    end
end

tink = Tink.new
tink.name = "The hero Tink"
p tink
```

Her laver vi nogle flere gettere og sættere (de laves automatisk, fordi vi bruger attr_accessor til at erklære variable) samt vi laver en default constructor. I denne constructor initialiserer vi vores inventory til en tom streng.

Vi laver en ny instans af vores klasse, så vi er sikre på at vores inventory er klar til brug.

Nu ser output således ud:

```
#<Tink:0x2377908 @name="The hero Tink", @inventory=[]>
```

Så vi kan se at vi har en helt, han har et navn og et tomt inventory.

Et inventory er ikke meget bevendt hvis vores helt ikke kan bruge det han har i sit inventory. Samtidig må vi sørge for at vores helt ikke kan have den samme ting i begge hænder og i sit inventory på en gang. En sidste krølle er at vores helt selvfølgelig skal have den ting han gerne vil bruge i sit inventory.

Lad os tilføje noget mere funktionalitet til klassen, så vi kan putte noget i vores helts hænder.

```
#List 5
class Tink
    def equip(item, hand)
        if (!@inventory.include? item)
            puts "You do not have that item. You have:"
            puts @inventory
            return
        end
        if (hand == :left)
            @lefthand = item
            @inventory = @inventory - [item]
        elsif (hand == :right)
            @righthand = item
            @inventory = @inventory - [item]
        end
    end
end

p tink
tink.inventory << "sword"
p tink
tink.equip("sword", :left)
p tink
```

Koden List-5 skal tilføjes til List-4.

Her åbner vi klassen igen og tilføjer en metode, hvor vi tager en item og putter den i vores hånd. Først undersøger vi med "include?" om den ting vi gerne vil bruge overhovedet er til rådighed.

Dernæst kigger vi om det er højre eller venstre hånd som vi gerne vil have vores ting i. Vi bruger det fra ruby der hedder symboler, fx. ":left" for at angive om det er højre eller venstre hånd.

En symbol i ruby er en konstant, som bruges til at identificere noget. I C ville man skrive #define LEFT 1 eller noget lignende.

Vi laver et trick for at fjerne en ting fra vores inventory. I ruby kan man trække to arrays fra hinanden, som vi gør med linjen "@inventory = @inventory - [item]". Den skal læses som: Vi laver et nyt lille array der indeholder vores ting, og fjerner det lille array fra det store inventory array. Så står vi tilbage med et inventory der ikke har den ting i sig som vi lægger i hånden.

Efter vi har tilføjet metoden er det tid til at afprøve den.

"Men vent! du har jo ikke en ny instans så ved din helt da ikke at han kan bruge ting fra sit inventory?" Jo, det er nemlig ruby viser sin styrke. Jeg tilføjer en metode på en klasse og derefter er metoden til rådighed for alle instanser af den klasse.

Lad os kigge på output:

```
#<Tink:0x2403b28 @name="The hero Tink", @inventory=[]>
#<Tink:0x2403b28 @name="The hero Tink", @inventory=["sword"]>
#<Tink:0x2403b28 @name="The hero Tink", @inventory=[], @lefthand="sword">

$
```

Først så ser vi, at vi stadig har vores helt. Han har dog ikke noget i sit inventory. Lad os give ham et sværd. Det gør vi med "<<" operatoren. Den betyder at vi tilføjer et objekt til den collection der står før "<<".

Så er vores output som i linje 2.

Til sidst flytter vi vores helts sværd til hans hånd, og vores data er som ses i linje 3.

Lad os se nærmere på den metode, der putter en ting i vores helts hånd, den slags vil der fremover blive meget brug for. Lad os se om vi kan give ham noget i hånden på en lidt anden måde, så vi ikke skal skrive så meget en anden gang (List 6:)

```
#List 6
class Tink
    attr_accessor :name
    attr_accessor :inventory
    attr_accessor :hands
    def initialize
        @inventory = []
        @hands = {}
    end
    def equip(item, hand)
        if (!@inventory.include? item)
          puts "You do not have that item."
          puts "You have:"
          puts @inventory
          return
        end
        @hands[hand] = item
        @inventory = @inventory - [item]
    end
end

tink = Tink.new
tink.name = "The hero Tink"
tink.inventory << "sword"
tink.inventory << "shield"
tink.equip("shield", :right)
tink.equip("sword", :left)
p tink
```

Nu har vi tilføjet endnu et felt til vores klasse, denne gang hedder det "hands" og vi initialiserer det som et hash. Det betyder at vi vil kunne bruge det som "@hands[:left]" for at refere til den value der ligger med key ":left" som angiver venstre hånd. Nu bliver vores logik til at putte noget i en hånd virkelig simpel og vi gentager ikke koden. Det er et af de vigtige principper i ruby. DRY - Do not Repeat Yourself.

Fordi vi ændrer på initializeren må vi hellere lave en ny udgave af vores tink, så vores output bliver som følger:

```
#<Tink:0x23b7a30 @name="The hero Tink",
@inventory=[], @hands={:right=>"shield",
:left=>"sword"}>
```

Nu kan vi give vores helt et våben i den ene hånd og et skjold i den anden, så må det være på tide at vi lære at angribe med det våben. Lad os tilføje det til vores helt.

```
#List 7
class Tink
    def attack
        if (@hands[:left] == "sword")
            return 10
        end
        return 5
    end
end

puts "We attack for #{tink.attack} damage"
```

Vi tilføjer en attack metode hvor vi kigger på om vores helt bruger hans sværd, så giver han 10 i skade, og ellers giver han 5 i skade. Udover det, så kigger vi om sværdet er i hans venstre hånd, da vores helt er klart bedst til at kæmpe med venstre hånd.

Når vi udskriver med puts, bruger vi endnu et ruby trick for at injecte vores variable ind i vores streng vi udskriver. I ruby kan man snige en variabel ud ved at bruge "#{}".

Det giver dette output:

```
We attack for 10 damage
```

Nu har vi både sværd i hånd, og vi kan angribe med det, men vi mangler noget at angribe. Hvad er bedre at angribe end noget af det ondeste der findes, nemlig et monster.

Lad os lave en monster klasse.

```
#List 8
class Monster
    attr_reader :dead
    def initialize
        @hitpoints = Random.rand(100)
        @dead = false
    end

    def hit(hps)
        @hitpoints -= hps
        if (@hitpoints <= 0)
            @dead = true
        end
    end
end

ork = Monster.new
puts "The monster is alive" unless ork.dead
```

Her laver vi en monster klasse, hvor vi kan angive om monsteret er dødt, samt vi giver monsteret en tilfældig mængde hit points.

Vi sørger lige for at vores monster mister hitpoints når det bliver ramt, samt vi så sørger for at monsteret bliver markeret som dødt når det kommer under 0 hitpoints.

Så bruger vi vores monster klasse til at lave en ny ***ork*** [1] samt udskrive en status om denne ork. Her kan vi se endnu en styrke ved ruby, nemlig hvor nemt det er at læse. Lad os læse programlinjen (den sidste i List-8) højt (i naturligt dansk sprog). I Ruby programmet står der:

puts "The monster is alive" unless ork.dead.

og jeg læser det sådan: Udskriv "The monster is alive" ***medmindre*** ork.dead

Det er jo lige ud af landevejen, dog halter den lidt tilsidst så lad

```
#List 9
class Monster
    def alive?
        return !@dead
    end
end

while ork.alive?
    dmg = tink.attack
    ork.hit(dmg)
    puts "You hit the monster for #{dmg}"
end
puts "The monster died"
```

os lave en function der gør det endnu bedre.

Vi laver så et loop (*while ork.alive?*), hvor vi lader vores helt angribe, så længe der er liv i orken.

Output bliver så som dette:

```
You hit the monster for 10
You hit the monster for 10
You hit the monster for 10
The hero Tink have killed the monster!
```

Så vores helt *kunne* overvinde et monster! Lad os se hvordan han klarer sig mod en række af monstre.

```
#List 10
monsters = [Monster.new, Monster.new, Monster.new]
round = 0
while (monsters.size > 0)
    round += 1
    puts "Round #{round}"
    monsters.each do |m|
        dmg = tink.attack
        m.hit(dmg)
        puts "You hit the monster for #{dmg}"
        if m.dead
            puts "#{tink.name} have killed the monster!"
        end
    end
    monsters.select! {|m| m.alive?}
end
```

Her laver vi et array med tre monstre. Så lader vi tink slå må dem en af gangen indtil de alle er døde. Vi bruger det der hedder at vi piper ting ind i en blok. Det gør vi ved at sætte "|m|" ind i toppen af vores blok som i "monsters.each do |m|".

Vi bruger rubys array select metode efter hver runde til at kun bruge de monstre som stadig har noget liv tilbage til at møde tink i næste runde.

Det giver et output som dette:

```
Round 1
You hit the monster for 10
You hit the monster for 10
You hit the monster for 10
Round 2
You hit the monster for 10
The hero Tink have killed the monster!
You hit the monster for 10
You hit the monster for 10
Round 3
You hit the monster for 10
You hit the monster for 10
Round 4
You hit the monster for 10
You hit the monster for 10
The hero Tink have killed the monster!
Round 5
You hit the monster for 10
Round 6
You hit the monster for 10
The hero Tink have killed the monster!
```

---oooOOOooo---

Igennem denne artikel har vi kigget på, hvor nemt det er at lave nogle enkelte dele til et spil i ruby, ved at bruge nogle af de elementer som ruby stiller til rådighed til os som udviklere.

Hvis man har fået blod på tanden kan jeg anbefale at man kigger på why's poignant guide to ruby:

http://mislav.uniqpath.com/poignant-guide/

Det er en lidt anderledes guide til at lære ruby. Det er fra denne guide at overskriften Chunky Bacon stammer.

Et af de elementer vi ikke har berørt, er rubys irb, den interaktive shell til ruby. Den er god hvis man bare lige vil prøve nogle enkelte ting. Personligt har jeg den som regel åben ved siden af min almindelige kode lige til at prøve forskellige stykker kode af.

Folk der kan perl vil måske tænke "Jamen det lugter jo lidt af det jeg kan" og det har de egentlig ret i. Der er som regel mere end en måde at gøre tingene på, ligesom der er i perl.

I et efterfølgende nummer vil vi kigge på det som har gjort ruby så stort, nemlig Ruby on Rails.

■

Heartbleed - århundredets værste sikkerhedshul

Første april rapporterede Googles sikkerhedsteam en sårbarhed i OpenSSL

af Donald Axel

... og det var ikke en aprilsnar. Fejlen vedrører Heartbeat funktionaliteten, som er keep-alive funktionen i HTTPS forbindelser. Sikkerhedshullet har fået navnet ***Heartbleed.*** Fejlen berører "kun" servere, som bruger **OpenSSL.** Der findes andre SSL implementationer. Det er en programmeringsfejl, som først blev opdaget efter 2 år.

Det er ikke vores almindelige PC'er eller smartphones, som har været i fare, men derimod vores private informationer som fx. passwords på servere som gmail, yahoo med flere; og hvad værre er, kan disse serveres certifikater være snuppet af kriminelle professionelle hackere, som vi skal se senere i denne artikel.

SSL står for Secure Socket Layer, Sikre forbindelser, hvor ordet Socket bruges om ***IP-nummer + port-nummer i begge ender af en forbindelse*** - entydig identifikation af en forbindelse. Port-numre er som bekendt ikke en fysisk port, men ren software kode - en slags "værelsesnummer", når datapakken er kommet indenfor i computersystemet.

Krypterede internetforbindelser kunne ved hjælp af den fejlagtige implementation af heartbeat misbruges til at hente indholdet af memory på de store servere.

Uden keep-alive kunne serveren lukke forbindelsen ned efter kort tid, og så skulle den genoprettes med udveksling af nøgler mv. for at man kunne kommunikere igen, heartbeat sparer derfor server-ressourcer, det er effektivt.

Opdagelsen

Det fremgik tydeligt af meldingen fra Googles sikkerhedsteam, hvad fejlen var og hvordan man kunne rette den - der medfulgte *patch.* Man finder en beskrivelse af opdagelsen i en pressemeddelelse fra Mark J. Cox https://plus.google.com/+MarkJCox/posts/TmCbp3BhJma

> *April 01 - Google contact the Google OpenSSL team members with details of the issue and a patch. This was forwarded to the OpenSSL core team members (1109 UTC). Original plan was to push that week, but it was postponed until April 09 to give time for proper processes. Google tell us they've notified some infrastructure providers under embargo, we don't have the names or dates for these.*

Direkte oversat står der, at *Google kontakter Google OpenSSL hold-medlemmer med detaljer om sagen og en patch (rettelse). Denne videresendes til OpenSSL core team (de ansvarlige for vedligehold af OpenSSL) kl. 11:09 UTC. Den oprindelige plan var at udsende rettelsen samme uge, men det blev udsat til 9. april for at give tid til den korrekte procedure. Google fortæller os at de har givet besked til nogle infrastruktur providere med besked om at de ikke må afsløre sårbarheden - vi har ikke navn og dato for disse meddelelser.*

Det lyder som om *Google* er en person. I Wikipedias redegørelse for forløbet siges det imidlertid, at det er Neel Mehta fra Googles sikkerhedsteam, som kontakter Mark J. Cox, som er medlem af OpenSSL teamet.

https://en.wikipedia.org/wiki/Heartbleed#cite_ref-25

Men er der nogen, der forinden har opdaget fejlen og givet besked til Google OpenSSL team? - det står åbent. Det er interessant, fordi det senere antages, at fejlen allerede er opdaget tidligere og besked er gået mellem de store udbydere, for derved at forhindre misbrug af sårbarheden ***inden*** den bliver offentliggjort. Derved har de store internet services med login via SSL kunnet sikre sig, inden offentliggørelsen kunne inspirere kriminelle til misbrug af sårbarheden.

Hvor længe har sårbarheden eksisteret?

Heartbeat har eksisteret siden marts 2012 (beta-versioner var ude forinden). På websitet for OpenSSL, openssl.org, kan man se nøjagtige datoer for releases (se nederst på siden). Heartbeat blev implementeret i version 1.0.1 der ses (med rød streg under):

4453920 **Mar 14** 14:34:38 **2012** openssl-1.0.1.tar.gz

For den kvikke - professionelle - kodebryder har der med andre ord været masser af tid til at udnytte heartbleed.

Source
Contribution
Support
Related

Bytes	Timestamp	Filename
4509047	Apr 7 19:21:29 2014	openssl-1.0.1g.tar.gz (MD5) (SHA1) (PGP sign) [LATEST]
4901640	Feb 24 14:54:01 2014	openssl-1.0.2-beta1.tar.gz (MD5) (SHA1) (PGP sign)
4089622	Jan 6 16:04:52 2014	openssl-1.0.0l.tar.gz (MD5) (SHA1) (PGP sign)
4509212	Jan 6 15:39:19 2014	openssl-1.0.1f.tar.gz (MD5) (SHA1) (PGP sign)
1442754	Jun 20 21:21:47 2013	openssl-fips-2.0.5.tar.gz (MD5) (SHA1) (PGP sign)
1421866	Jun 20 21:21:47 2013	openssl-fips-ecp-2.0.5.tar.gz (MD5) (SHA1) (PGP sign)
1421713	Apr 10 14:00:24 2013	openssl-fips-ecp-2.0.4.tar.gz (MD5) (SHA1)
1442721	Apr 10 14:00:13 2013	openssl-fips-2.0.4.tar.gz (MD5) (SHA1) (PGP
1442720	Mar 27 14:35:58 2013	openssl-fips-2.0.3.tar.gz (MD5) (SHA1) (PGP
1421714	Mar 27 14:35:58 2013	openssl-fips-ecp-2.0.3.tar.gz (MD5) (SHA1)
4459777	Feb 11 16:34:23 2013	openssl-1.0.1e.tar.gz (MD5) (SHA1) (PGP sig
4459791	Feb 5 13:17:07 2013	openssl-1.0.1d.tar.gz (MD5) (SHA1) (PGP sig
4050842	Feb 5 13:17:00 2013	openssl-1.0.0k.tar.gz (MD5) (SHA1) (PGP sig
3785001	Feb 5 13:16:55 2013	openssl-0.9.8y.tar.gz (MD5) (SHA1) (PGP sig
1421635	Oct 31 19:58:33 2012	openssl-fips-ecp-2.0.2.tar.gz (MD5) (SHA1)
1442642	Oct 31 19:58:19 2012	openssl-fips-2.0.2.tar.gz (MD5) (SHA1) (PGP
1422099	Jul 10 20:20:06 2012	openssl-fips-ecp-2.0.1.tar.gz (MD5) (SHA1)
1442377	Jul 10 20:19:33 2012	openssl-fips-2.0.1.tar.gz (MD5) (SHA1) (PGP
1407102	Jul 1 14:45:28 2012	openssl-fips-2.0.tar.gz (MD5) (SHA1) (PGP
4457113	May 10 17:20:24 2012	openssl-1.0.1c.tar.gz (MD5) (SHA1) (PGP sig
4047852	May 10 17:07:50 2012	openssl-1.0.0j.tar.gz (MD5) (SHA1) (PGP sig
3782486	May 10 16:41:36 2012	openssl-0.9.8x.tar.gz (MD5) (SHA1) (PGP sig
4456651	Apr 26 12:52:52 2012	openssl-1.0.1b.tar.gz (MD5) (SHA1) (PGP sig
3919005	Apr 25 14:37:05 2012	openssl-fips-1.2.4.tar.gz (MD5) (SHA1) (PGP
3782900	Apr 23 23:05:39 2012	openssl-0.9.8w.tar.gz (MD5) (SHA1) (PGP sig
4456456	Apr 19 14:20:37 2012	openssl-1.0.1a.tar.gz (MD5) (SHA1) (PGP sig
3782207	Apr 19 14:06:49 2012	openssl-0.9.8v.tar.gz (MD5) (SHA1) (PGP sig
4047721	Apr 19 13:55:31 2012	openssl-1.0.0i.tar.gz (MD5) (SHA1) (PGP sig
1386753	Mar 15 11:57:10 2012	openssl-fips-ecp-2.0.tar.gz (MD5) (SHA1) (P
4453920	Mar 14 14:34:38 2012	openssl-1.0.1.tar.gz (MD5) (SHA1) (PGP sign
3781776	Mar 12 16:43:06 2012	openssl-0.9.8u.tar.gz (MD5) (SHA1) (PGP sig
4048067	Mar 12 16:36:25 2012	openssl-1.0.0h.tar.gz (MD5) (SHA1) (PGP sig
4451351	Feb 24 00:07:06 2012	openssl-1.0.1-beta3.tar.gz (MD5) (SHA1) (PGP sign)
4447371	Jan 19 17:50:55 2012	openssl-1.0.1-beta2.tar.gz (MD5) (SHA1) (PGP sign)
4046513	Jan 18 14:43:42 2012	openssl-1.0.0g.tar.gz (MD5) (SHA1) (PGP sign)
3778943	Jan 18 14:21:58 2012	openssl-0.9.8t.tar.gz (MD5) (SHA1) (PGP sign)
3779406	Jan 4 20:28:25 2012	openssl-0.9.8s.tar.gz (MD5) (SHA1) (PGP sign)

Factbox

OpenSSL Project er et Open Source projekt for udvikling og vedligehold af en robust, højkvalitets Open Source implementation af protokoller for kryptering af net-trafik: Secure Sockets Layer (SSL version 2 og version 3) og Transport Layer Security (TLS version 1) samt et generelt krypterings-bibliotek.

Projektet ledes af et verdensomspændende fællesskab af frivillige, der bruger internettet til at kommunikere, planlægge og udvikle OpenSSL Toolkit og den tilhørende dokumentation.

Medlemmer af kerneholdet er også medforfattere til standard-specifikationen i IETF (RFC 6520). OpenSSL bruges af de fleste kommercielle sites, og bl.a. også i systemer, der fordeler belastningen, (load-balancing) *HAProxy* og *Nginx*

Hvad går fejlen ud på

Formålet med Heartbeat er som nævnt, at serveren ikke så ofte skal genforhandle en krypteret forbindelse. Brugeren (browseren) sender med jævne mellemrum en besked om at den gerne vil bevare forbindelsen - et hjerteslag, et heartbeat: Jeg lever endnu!

Datapakke 1 fra browseren er at forstå som et spørgsmål: Må vi opretholde forbindelsen?

Svar fra serveren kan være ja eller ingenting - Serveren kan droppe forbindelsen af den ene eller anden grund, måske fordi der er for travlt med andet. Men hvis den iøvrigt har en smule trafik med browseren, spares der en genforhandling af krypteringen, når man bruger denne "keep-alive" funktion.

Heartbeat_anmodning(1) fra browseren - fortsætter vi?

Heartbeat_svar(2) fra serveren - ja!

så simpelt er det.

Browseren spørger med et "payload", fx. en tekst som "Yellow Submarine". Længden af payload er i et felt i datapakken.

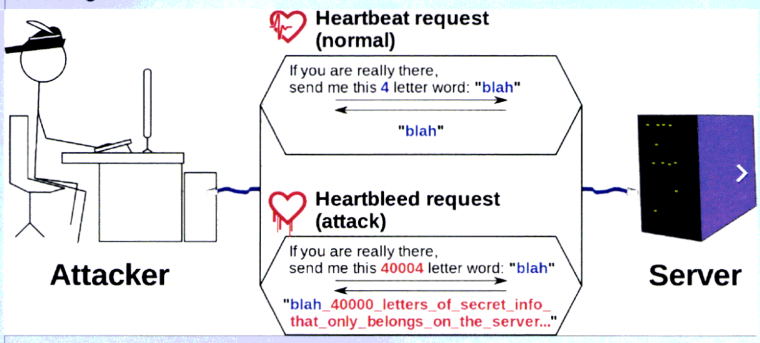
Fejlen i openssl koden

Men ak!

Payload længdefeltet bliver ***ikke*** verificeret i openssl-koden (frem til april 2014) Browseren (eller rettere hackeren, netklient-programmet) kan angive en større længde end payloads faktiske længde.

Serveren vil da lave et stort payload felt i returpakken, og de overskydende bytes (op til 64K) vil være memory, som har været brugt til noget andet - og som faktisk kan indeholde interessante, ukrypterede oplysninger.

Tegningen herunder (fra Wikipedia) viser flow *med data "blødning":*



Hvordan udnyttes denne fejl?

Skal man så sidde og kigge alle de modtagne datapakker igennem?

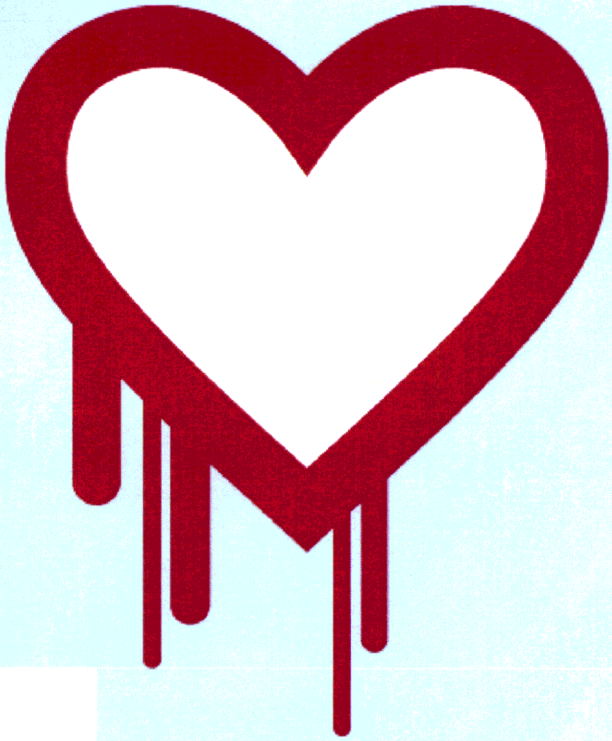
Nej, en rigtig professionel kodebreaker har selvfølgelig programmer, som klarer alt det kedelige, men det er dog gennemsøgningen, som tager mest "opmærksomhedstid".

CloudFlare, et firma, som optimerer og sikrer websites med videre, indbød til et forsøg på at hacke en server med Heartbleed buggen (det omtales bl.a. i Infoworld og Washington Post).

De opfordrede Internet brugere til at køre deres egne test på en dummy-server, som havde Heartbleed buggen. Hackerne skulle så "stjæle" security certifikatet fra serveren og så sende besked til CloudFlare, som var "signeret" med dette certifikat for at bevise, at de faktisk havde fået fat i det. Indenfor ni timer efter "konkurrencens" offentliggørelse - og tre timer efter, at han begyndte at arbejde på sagen - havde en hacker ved navn Fedor Indutny allerede løst opgaven.

"Det var såmænd bare en sjov måde at bruge fredag aften, og en god lejlighed til at demonstrere mine evner i en lovlig cracking opgave" skrev Indutny i en e-mail til Washington Post. "Efter at have startet et script på en cloud-server, så jeg film og glemte alt om konkurrencen. Efter at checke logfilerne i ca. en time opdagede jeg til min overraskelse en private key!"

Det krævede godt nok et par millioner "Heartbeat" udvekslinger, før Indutny fik en secure key, men selv med en langsommere fremgangsmåde ville det ikke være urimeligt vanskeligt. Indutny var ikke den eneste, det lykkedes at fiske de private nøgler ud af dummy-serveren.



Indutny var ikke den eneste, som meldte tilbage til CloudFlare at de havde fundet en private key fil.

Kan man ikke bare skifte private nøgler?

Jo, selvfølgelig kan man skifte private krypteringsnøgler ud, og det bør man også gøre. Men hvis andre har fået nøglen, skal den trækkes tilbage, den skal *blokeres* ligesom et stjålet Dankort, ellers kan tyven sætte et website op, som hedder det samme, og med det narre folk til at logge ind. På engelsk hedder *revoke,* når den key og det certifikat, som bekræfter dens ægthed, trækkes tilbage.

Opsætning af et "fup-site" er ikke for begyndere, det kræver bl.a. injection af falske DNS oplysninger, men det kan gøres (endda forholdsvis nemt, siger eksperterne).

Det er ikke alle browsere, som automatisk checker, om et certificat for et website er trukket tilbage. Revocation listen (listen over blokerede sites) ville med et være over en halv million. Nogle browsere har verifikation slået fra. Det forøger risikoen for at nogle mennesker falder i fælden.

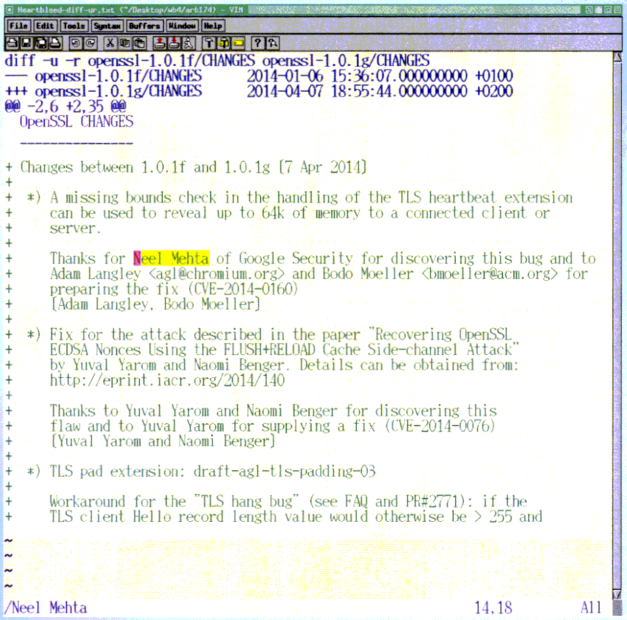
Med CloudFlare udfordringen in mente er det derfor tydeligt, at Heartbleed er meget farlig og kan have langvarige, skadelige følger.

Nysgerrig? - se med egne øjne!

Hvis man er nysgerrig, kan man selv finde fejlen ved at sammenligne openssl-1.0.1f fra januar 2014 med den nyeste, patchede version 1.0.1g fra 7.april 2014.

Openssl source kan downloades frit:

```
4509047 Apr  7  19:21:29 2014  openssl-1.0.1g.tar.gz
4509212 Jan  6  15:39:19 2014  openssl-1.0.1f.tar.gz
```



Ved at sammenligne de to (og bruge gvim for at få et screenshot med fremhævelser, som gør det lettere at læse) ser man straks, at OpenSSL team i loggen har takket Neel Mehta fra Google Security for at have meddelt fejlen til OpenSSL-core team. Vi er på rette spor!

Fejlen beskrives sådan: A missing bounds check in the handling of the TLS heartbeat extension can be used to reveal up to 64k of memory to a connected client or server.

> *Et manglende bounds check i håndteringen af TLS heartbeat udvidelsen can bruges til at afsløre op til 64k memory til en opkoblet klient eller server.*

Lad os se, om vi kan finde fejlen blandt de 1700+ linier, som kommer ud af en sammenligning af de to udpakkede versioner af OpenSSL.

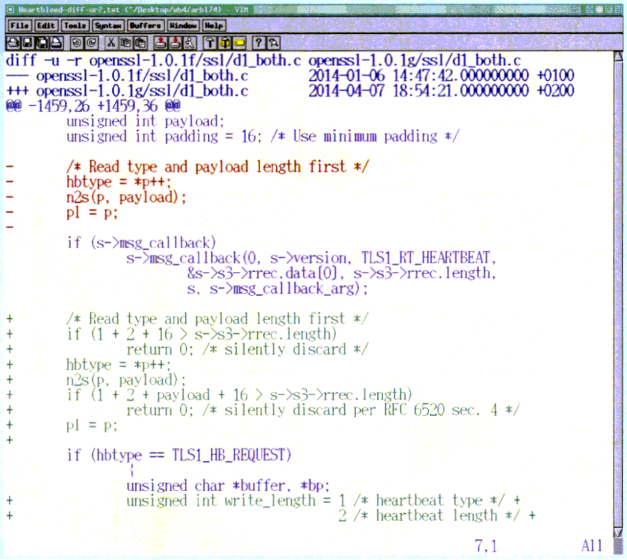
Vi har udpakket de to versioner, typisk i et tomt arbejds-directory, og ls viser:

```
sat2:/usr/local/src/ssl #ls
openssl-1.0.1f  openssl-1.0.1g
sat2:/usr/local/src/ssl #
```

Først kigger vi med diff -ur på sidste version med Heartbleed og så den nye, rettede version fra 7.april 2014:

```
diff -u -r openssl-1.0.1f/ssl/d1_both.c \
           openssl-1.0.1g/ssl/d1_both.c
```

og i output, skønt lidt forvirrende, kan vi forholdsvis hurtigt spotte de steder, hvor der står "HEARTBEAT":



Jeg har, for at det skal være lettere at læse, cut-pasted output fra ***diff*** programmet ind i en gvim-editor med syntax highlighting. diff -u formatet er det, der kaldes unified patch format.

Her ses linier med ***minustegn*** foran - de skal væk, hvis man vil patche (rette) den gamle kode, så den svarer til nyeste version. Linier med ***plustegn*** foran skal indsættes.

Det, der straks fanger opmærksomheden, er at der i plus-linierne står at hvis 1+2+payload+16 er større end rrec.length, så lad være med at gøre mere - med andre ord, kassér indkommende heartbeat-anmodning.

Den linie mangler helt i ***minustegn*** linierne, og der er heller ikke noget lignende i den bevarede kode, som er uden margin-tegnmarkering.

Men det er lidt svært at læse, også for en, der er vant til at læse C-programmer. De forskelle, man kan se, kan kun bruges til at spore én ind på, hvad der er sket med koden og hvor det er sket.

Vi kan bruge ***gvim*** editoren til at se nærmere på koden, og filnavnet for denne kode ses øverst, d1_both.c

Kommandoen hedder **`gvim -d fil1 fil2`**

Altså:

```
$ gvim -d openssl-1.0.1f/ssl/d1_both.c \
          openssl-1.0.1g/ssl/d1_both.c
```

Det giver et editor vindue afbildet nederst på siden. Det er beskåret for ikke at blive for småt. Jo større skærm, desto bedre overblik.

Hvis man vil se de linier, der er foldet sammen (altså de linier, som ikke vises fordi de er éns i begge filer) kan man klikke på et plus-tegn i marginen (som ikke er vist af pladshensyn).

Nu ses det forholdsvis hurtigt, at der er de samme data-deklarationerne øverst i en funktion (men hvilken funktion? - når man "folder ud" ser man, at det **er** heartbeat implementationen:

```
#ifndef OPENSSL_NO_HEARTBEATS
int
dtls1_process_heartbeat(SSL *s)
```

Ligeledes kan vi forholdsvis let se at noget, der kunne ligne en funktion, *n2s,* hiver en variabel-værdi *payload* ud af en pointer *p.

En erfaren C-programmør kan se, at *n2s* er en macro. Man får ikke en værdi retur i en variabel som er parameter i et funktions-kald - det skulle have været en pointer, hvis *n2s* var en funktion.

Ved at følge programmet tilbage kan vi se, at *p peger på den modtagne datarecord.

Nu ved vi, at den værdi, som *n2s* lægger i variablen *payload* **kommer fra klienten.** Værdien *payload* er en del af den modtagne datapakke, som spørger: *Kan vi vedligeholde forbindelsen?*

Men det er det røde felt, der fanger opmærksomheden; ***gvim -d*** er i stand til at vise **hvor** på linien der er forskel, og det er længdeangivelse til OPENSSL_malloc() funktionen, som er det præcise sted, at fejlen sker.

Dér allokeres memory i størrelsen ***payload*** + padding etc. Husk på, at værdien *payload* kommer fra klienten - og den længdeangivelse behøver ikke at svare til længden af den faktisk tilsendte datapakke (med fx. string "Yellow Submarine", men hvor længden er angivet til at være 64000).

Altså: I stedet for at tage længden af den modtagne datapakke (som selvfølgelig fås fra den funktion, der læser fra net-forbindelsen) bruger programmet en ukontrolleret værdi.

En ondsindet pakke med større længdefelt vil med denne kode bevirke, at der allokeres mere memory; den vil indeholde rester fra tidligere programkørsler, eller data fra tidligere kørte funktioner i samme server-program.

```
Normal pakke:

+-----+---------------+----------------+
|type | payload længde| payload etc.   |
+-----+---------------+----------------+
|   1 |       10      | Yellow Sub  ...|
+-----+---------------+----------------+
|<----------fx-37bytes---------------->|

Ondsindet pakke:

+-----+---------------+----------------+
|type | payload længde| payload etc.   |
+-----+---------------+----------------+
|   1 |    64000      | Yellow Sub  ...|
+-----+---------------+----------------+
|<------stadig-37bytes---------------->|
```

For at undgå fejlen skal man selvfølgelig bare afvise pakker, hvor længdefeltet ikke svarer til faktisk modtagne pakke.

Nu har vi - meget kortfattet - set på koden, som er årsag til Heartbleed. Det tager et par timer (eller dage, hvis man vil ned i substansen) og det var ikke vanskeligt at få fat i koden. Lad mig så lige tilføje, at man ved at undersøge nærmere finder nøjagtig samme kode i en anden sourcefil, som danner et library til ssl-applikationer, så systemet er stort, og man skal kende det overordnede design.

(fortsættes side 18)

```
lines: ssl/d1_both.c ------------------------------------------------------------
{
unsigned char *p = &s->s3->rrec.data[0], *pl;
unsigned short hbtype;
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                &s->s3->rrec.data[0], s->s3->rrec.length,
                s, s->msg_callback_arg);

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
if (hbtype == TLS1_HB_REQUEST)
        {
        unsigned char *buffer, *bp;
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
        int r;

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
        /* Allocate memory for the response, size is 1 byte
         * message type, plus 2 bytes payload length, plus
         * payload, plus padding
         */
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;
```

```
+ +--1455 lines: ssl/d1_both.c ---------------------------------------------------
        {
        unsigned char *p = &s->s3->rrec.data[0], *pl;
        unsigned short hbtype;
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

        /* Read type and payload length first */
        if (1 + 2 + 16 > s->s3->rrec.length)
                return 0; /* silently discard */
        hbtype = *p++;
        n2s(p, payload);
        if (1 + 2 + payload + 16 > s->s3->rrec.length)
                return 0; /* silently discard per RFC 6520 sec. 4 */
        pl = p;

        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
                unsigned int write_length = 1 /* heartbeat type */ +
                                            2 /* heartbeat length */
                                            payload + padding;
                int r;

                if (write_length > SSL3_RT_MAX_PLAIN_LENGTH)
                        return 0;

                /* Allocate memory for the response, size is 1 byte
                 * message type, plus 2 bytes payload length, plus
                 * payload, plus padding
                 */
                buffer = OPENSSL_malloc(write_length);
                bp = buffer;
```

Ubuntu 14.04 LTS er på gaden

Af Thomas Ammitzbøll Bach

Det engelske firma Canonical har midt i påsken frigivet den seneste udgave af deres linuxdistribution, Ubuntu, i version 14.04 LTS med kodenavnet Trusty Tahr. I denne prosaiske betegnelse ligger følgende information: At den er frigivet i år (20)14 i måned 4 (april), og at der er tale om en "Long Term Support" udgave, hvilket vil sige, at Canonical forpligter sig til at udgive opdateringer i 5 år. Kodenavnet er som sædvanlig et dyrenavn og følger konventionen, at begyndelsesbogstavet skal være næste i rækken efter sidste release.

Den seneste udgave er udgivet til fire platforme/anvendelsesformål: Cloud, Server, Desktop og Mobile (hvilket vil sige smartphone og tablets). Det er selvfølgelig en marketingstrategi, for de fire platforme er ikke helt sammenlignelige, men det giver et fingerpeg om, at computerverdenen skifter gear i disse år, og den traditionelle server og desktop er på vej mod elefantkirkegården, og fremtiden er en blanding af SaaS (Software as a Service), PaaS (Platform as a Service), og en masse andre [A-Z]aaS'er kombineret med de nye letvægtsenheder som smartphones og tablets. Det er ikke en ændring, der sker i løbet af de næste 5 år, men udviklingen er allerede i gang.

Desktoppen

Det er nærliggende at begynde med at se på desktoppen, for dels er det der, man opdager de mest håndgribelige ændringer, og dels er det desktoppen, der gav Ubuntu en stormende modtagelse for næsten 10 år siden. Og desktopudgaven er da også pudset op, og der er kommet et par nye ting til. Men det virker lidt i småtingsafdelingen, for Linux er ikke kvantespringenes teknologi, men er nærmere som en lava, der langsomt men sikkert indtager nye områder. Derfor er mange "nyheder" i virkeligheden bare boblere, der er modnet og forfremmet.

Unity, der er Ubuntus grafiske grænseflade, har altid været genstand for debat. Da man flyttede minimér-, maksimér- og luk-knapperne ("kryds-og-bolle-knapperne") over til venstre side af vinduet, var der mange, der var uenige i den beslutning. Da man flyttede menuerne fra toppen af vinduet til toppen af displayet, var der ligeledes mange, der var uenig i det. Men tilsyneladende har Unity-udviklerne lyttet til kritikken, og de har tilføjet en mulighed for at ændre menuplaceringen tilbage til toppen af vinduet. En anden ting, der har været kritik af, er at menuerne i forskellige applikationer (f.eks. Nautilus filebrowseren) har været meget sparsomme til fordel for kontekstmenuer og knapper i toolbaren. Her er filmen også rullet lidt tilbage, for menuerne er knap så sparsomme mere. Forklaringen er, at forskellige add-ons har ikke kunne placere deres menupunkter, når menuerne er væk med en masse problemer til følge.

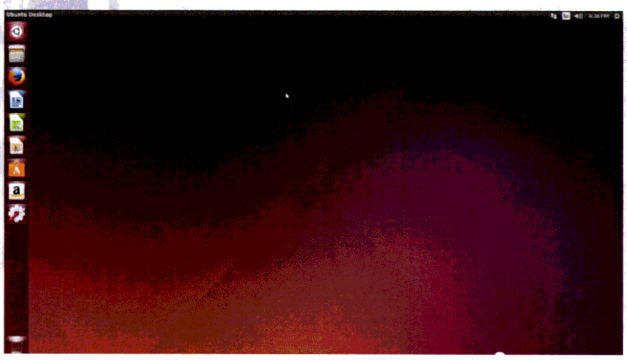


Foto: Simon Schytt

Thomas Ammitzbøll Bach, konsulent, programmør, instruktør - og skribent :)

En ny indstilling er et generelt "zoomlevel", hvor man kan justere knapper og tekster til forskellige skærmopløsninger og tæthed (DPI). Det har mere og mere betydning, fordi mange kompakte skærme i dag kan fås med meget stor tæthed, og det medfører et krav om at kunne justere den generelle størrelse af elementer på skærmen.

Ubuntu har "opfundet" en letvægtsbrowser til at integrere webapplikationer med desktoppen. Denne integration var med i 13.10, men det fungerede ikke helt optimalt, fordi den var baseret på at anvende eksisterende åbne browservinduer. Derfor er Oxide, som letvægtsbrowseren hedder, en tilpasset udgave af Google's Chromium. Umiddelbart er det let at være skeptisk, for KDE gjorde det samme på et tidspunkt, og det gav ikke en naturlig oplevelse, men tiden vil vise, om det lykkedes for Ubuntu. Det er svært at få webapplikationer til at virke som en integreret del af en desktop, med eller uden letvægtsbrowsere.

Ellers er der nye udgaver af stort og småt. LibreOffice er rykket et par trin frem, Firefox og Thunderbird er rykket et par trin frem, og der er opdateringer rundt omkring. Det ændrer dog ikke ved, at desktoppens opdateringer er konservative og "lava-agtige".

Cloud

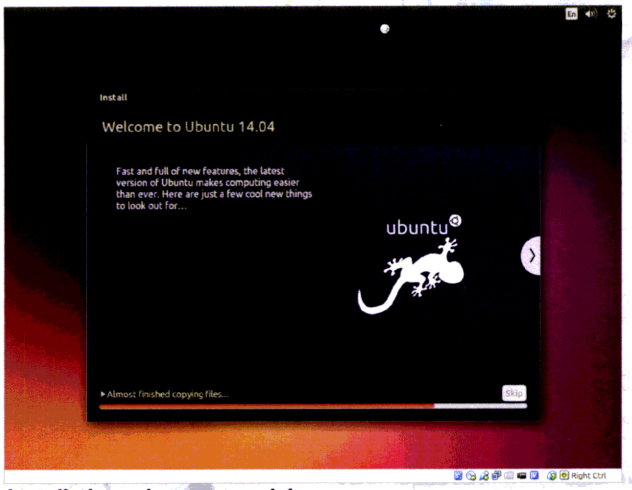
Canonical er helt oppe på beatet, når det drejer sig om at sælge Ubuntu som den foretrukne cloud-platform, og integrationen med OpenStack bliver stærkere og tættere. Hvis man undrer sig over denne satsning, så er det fordi, det er her de store servicekontrakter er at sælge. Canonical er en forretning, og selv om vi er mægtig glade for, at de vil give verdenen en åben og gratis distribution, så handler det om at omsætte varen på hylden, og det er partnerskaber, servicekontrakter og uddannelse. Og for tiden er de bedste kunder store datacentre, der har brug for at holde mange servere og tjenester i luften.

Openstack er netop samme dag (17. april) blevet frigivet i version 2014.1 med kodenavnet Icehouse. Om det er tilfældigt vides ikke, men Canonical har lagt et stort arbejde i at få integrationen med OpenStack til at være meget tæt. OpenStack er en platform til at administrere virtualiseringen af forskellige dele af cloud service. Det vil sige CPU, storage og netværk samt administration og flytning af de forskellige dele. OpenStack er ikke i sig selv et virtualiseringssystem, men kan anvende KVM, Xen, LXC eller endda Hyper-V som host. Med OpenStack gør det det nemmere at lave images, instantiere images, oprette, flytte, starte/stoppe og nedlægge instanser.

Installation og tilpasning

Alle, der har installeret Ubuntu, ved, at det er let nok at lave en grundinstallation, men det tager tid at sætte en installation op til en bestemt service. Det tager tid at finde ud af, hvilke pakker

der er nødvendige for at køre f.eks. en Puppet master, et django-site, et compile-miljø, etc. For at gøre den slags nemmere, har Canonical investeret en masse kræfter i deres Juju, der er et værktøj til deployment. Med Juju installerer man en eller flere "charms", der er en slags meta-pakker. Juju letter virkelig arbejdet med at lave images. Med få kommandolinje-instruktioner er alle pakker installeret og klargjort.



Installation - den nemme del

Virtualisering

Som nævnt er der flere muligheder for virtualisering. Canonical spiller på flere heste her, men det er tydeligt, at KVM er deres hovedfokus. KVM er står for Kernel-based Virtual Machine, og som navnet antyder, så er det en integreret del af linuxkernen og udvikles af kerneudviklerne. Det gode er, at det nok er et ret fremtidssikret valg, men det er ikke et buzzword blandt topledere på linje med VMware og XenServer. Derfor er forståeligt, at Canonical har en strategi på disse virtualiseringssystemer. Der er arbejdet meget med at glatte rynker ud, så både host-tools og guest-tools installeres (næsten) automatisk.

Der er brugt mange ressourcer på at få LXC til at køre godt. LXC står for LinuX Containers og er en form for virtualisering, hvor gæstsystem og hostsystem deler samme kerne og filsystem. Det gør det muligt at køre rigtig mange gæstsystemer på det samme hostsystem, og det kan fint lade sig gøre at køre 300 gæstsystem på en enkelt server. LXC har ikke spillet "out-of-the-box" i tidligere udgaver af Ubuntu, hvor man typisk skulle tweake en del for at få gæstsystemerne til at køre ordentligt, men det virker nu meget mere strømlinet end tidligere.

Puppet - Multi-platform Open Source Configuration Management

Puppet er opgraderet til version 3.4. Der er sket meget siden 2.7, bl.a. er der kommet Hiera, der gør data til konfiguration mere hierarkisk, MCollective, som er et dataopsamlingssystem. Hvis man ikke tidligere har brugt Puppet, så er det en god anledning til at få kigget på det. Administration af mange hosts og/eller servere lettes væsentligt, og det er lettere af holde styr på versionering af konfigurationsfiler og deployment.

Mobile enheder

Smartphones blev *in* fra omkring 2009, og interessen har samlet sig om tre platforme, nemlig Apples iPhone, Googles Android og Microsofts WP8. Det er derfor en satsning for Canonical at gå ind i det marked, som tilsyneladende allerede har delt forbrugerne i mellem sig. Og der er nok heller ikke nogen, der spår at Ubuntu Touch får nogen nævneværdig markedsandel af det brede salg det første lange stykke tid. Men det er nok heller ikke målet i sig selv. For at kickstarte udbredelsen af Ubuntu på mobile enheder, har Canonical valgt at anvende Nexus 4 som udviklingsplatform, der som bekendt er Googles referenceplatform til Android. Det er et veldefineret stykke hardware, som er brugbar som daglig telefon, hvilket er vigtigt, når test af Ubuntu Touch skal ud af laboratoriet og ned i folks lommer.

Hvad kan Ubuntu Touch så? En hel del! De grundlæggende funktioner er stabile og velkørende, og der er en række apps til rådighed allerede, som folk kender fra andre smartphones som Youtube, Facebook, Evernote, Gmail og Maps. Ubuntu Touch understøtter både "native apps", som er skrevet i C++ og/eller Javascript, og "web apps" i HTML 5, CSS og Javascript. De to typer apps kan frit blandes og placeres i home screen efter ønsker og behov.

Der er mange ergonomiske beslutninger, der lægger sig tættere op af Unity end af andre smartphones, så det kræver lidt tilvænning, hvis man ellers bruger f.eks. Android, men man overraskes over, hvor god udnyttelse af skærmarealet det er lykkedes at opnå. Der er flere swipes (fra venstre, fra højre, nedefra) til forskellige funktioner, og det betyder, at mange kontroller er gemt "i gardinet". I Ubuntu Touch finder man en meget klassisk forståelse af multitasking, nemlig at en app startes og kører, indtil man afslutter den, og alle kørende apps listes side om side. Det er på en måde befriende, for man behøver ikke at gætte på, om en app er blevet zappet af systemet, fordi noget andet havde brug for noget mere RAM eller CPU. Men det betyder også, at man skal huske at afslutte sine apps og ikke bare lade dem hænge i gardinet.

Tærsklen til at lave sine egne apps er bemærkelsesværdig lav. Ubuntu Touch er baseret på Qt, og man kommer meget langt med bare en simpel QML-fil på et halvt hundrede linjer. Der er et SDK, som for den størstedels vedkommende blot er en standard Qt-Creator.

Konklusion

Canonical har grund til at være stolte over deres seneste release. Som nævnt er der ikke de store synlige forskelle fra release til release. For at få øje på fremdriften skal man kigge på, hvad der var boblere i sidste release og hvor langt de er nået i modenhed nu. Og anskuer man det sådan, så er der sket meget fremdrift.

Det er synlige tegn på, at Canonical satser på at tjene penge og mere end nogensinde. Det er ikke ment som noget negativt, for der anvendes open source løsninger alle steder, hvor det overhovedet er muligt; men de teknologier, der er mest fokus på, er tydeligere end nogensinde dem, der er til fordel for stordrift. Canonical tjener deres penge på netop at sælge løsninger til større organisationer i form af konsulenttjeneste, uddannelse, partnerskaber og specialløsninger. Men det, der var storserverteknologi for et par år siden, er nu blevet almindeligt i selv små installationer, så når det regner på præsten, drypper det på degnen.

■

MySQL, MariaDB

Open Source og database-standard

Engang for mange år siden var der ofte diskussion om hvilke sprog, der var "bedst", forstået som størst udtrykskraft, størst aflastning af programmøren og mindre risiko for sjuskefejl, og hvilke sprog, der gav størst chance for kode-genbrug på andet isenkram.

Der gik nogle år før det gik op for flertallet af programmører, at det snarere handlede om at finde det sprog, som egner sig bedst til den type opgave, man står med. I begyndelsen af "computer-alderen" (60-70) handlede det om at se mulighederne, forstå hardwaren, men i løbet af 70'erne og 80'erne voksede presset for at få standardiserede programmeringssprog.

Da jeg i 80'erne havde kontakt til Data General, var der en venlig og meget alsidig programmør, som gav mig en kort introduktion til Cobol - det interessante, sagde han, er at Cobol er det første sprog, som har defineret et standardiseret interface til en database. Cobol, DB2 og SQL og mange andre sprog (fx. PL/1) var alle udbredt på IBM produkter. Men det var så som så med at overholde standarder: konkurrenternes implementationer af fx. SQL havde altid extensions, som gjorde at man blev låst fast til systemet, hvis man brugte alle features.

MySQL blev grundlagt i 1995 af David Axmark, Allan Larsson and Michael "Monty" Widenius - med det formål at give Open Source miljøet en SQL-compliant database. Første release i 1995.

MySQL drop-in for mSQL

MySQL var fra begyndelsen SQL i en forenklet udgave - med en file-storage back-end (ISAM). MySQL var et drop-in replacement for mSQL, mini-SQL, som er et database-system, der er frit for non-kommerciel anvendelse, tilgængelig source med copyright.

Men netop ved at være en forsimplet SQL var brugere af MySQL bundet til systemet. (Ingen klagede dog, for MySQL var stabil og meget hurtig i forhold til alternativerne.) Fx. blev *transaktioner* implementeret meget senere (ca. 2006). For nye projekter betød manglerne ikke meget, for man kan emulere de mere komplekse, rekursive søgninger og operationer med flere, på hinanden følgende, forespørgsler og opdateringer, men en overgang fra fx. Oracle SQL til MySQL var (og er) vanskelig.

I løbet af nullerne skete der meget med MySQL: den fik en alternativ back-end, InnoDB, og der blev desuden etableret samarbejde med en af tidens 3 største database-produkter:

> *In 2003 SAP AG and MySQL AB joined a partnership and re-branded the [Adabas] database system to MaxDB. In October 2007 this reselling was terminated and sales and support of the database reverted to SAP. SAP AG is now managing MaxDB development, distribution, and support. Source code of MaxDB is no longer available under the GNU General Public License. SAP also stated that "Further commercial support concepts to cover mission critical use requirements outside of SAP scenarios are currently subject to discussion."*

(https://en.wikipedia.org/wiki/MaxDB)

Dynamiske websites

Software as a Service (SaaS) er jo egentlig bare applikationer, som kører centralt med en GUI på en anden device. Det svarer til "gamle dages" multi-user systemer, med den lille forskel, at transporten til skærmen sker via (trådløst) netværk.

Databaser er rygraden i dynamiske websites. I sidste nummer så vi på en simpel web-applikation, som benyttede PHP, men det er ikke det eneste sæt Open Source programmer, som kan bruges til SaaS.

PHP har samme sæt database-access funktioner til flere typer databaser, enten et eget sæt funktioner (MySQL & Postgres) eller gennem ODBC (Open Database Connectivity). Med PHP programmer er man derfor i stand til at skifte database-server.

Men SQL på fx. PostgreSQL og MySQL er alligevel ikke helt ens, så hvis man har været ude i hjørnerne af sin database, så bliver man "gift med den".

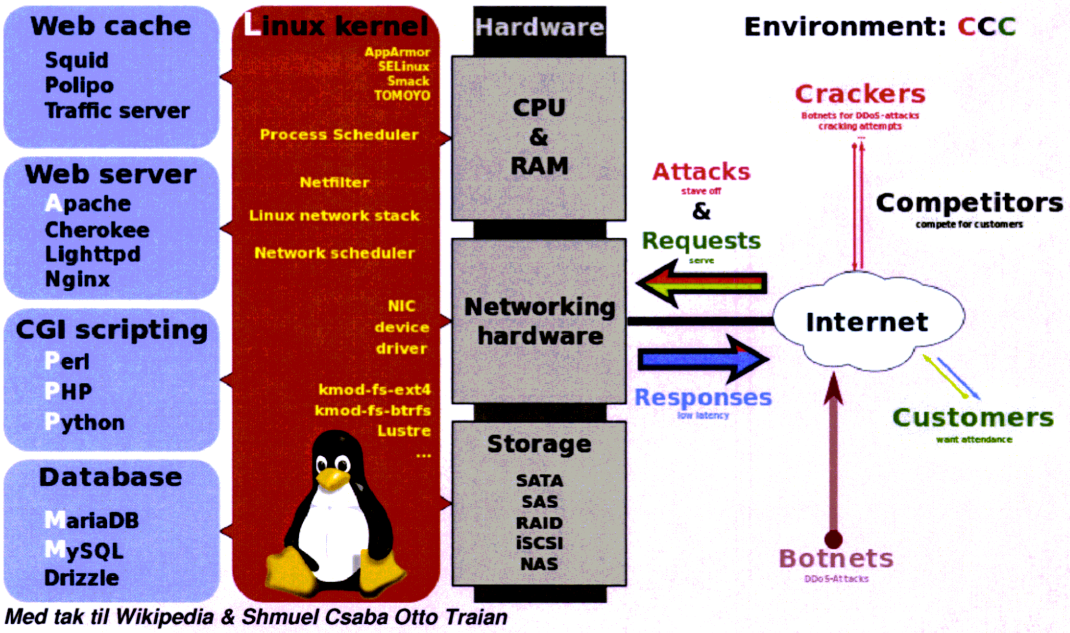
LAMP stakken

Allerede for 10-15 år siden brugte NASA og ministerier i USA LAMP (Linux, Apache, MySQL og Perl) og anbefalede det til skoler og institutioner som en stabil og billig internetplatform. Siden er det blevet til Apache, MySQL og PHP, eller Python, eller Ruby on Rails - og der er mange flere alternativer, se illustrationen nedenfor (med tak til *Shmuel Csaba Otto Traian*).

Man ser, at MySQL er suppleret med MariaDB. For de fleste læsere er det nok ikke en nyhed, at MySQL grundlæggerne Widenius, Axmark og Allan Larsson etablerede en virksomhed,

Som et kuriosum kan nævnes at redaktionens performance forsøg med 6 millioner records i MySQL, som den kommer pr. standard med Debian Linux, gav nøjagtig samme antal inserts pr. sekund (1500) som forsøgene med Postgres (på samme system, samme disk etc.) 6mio. af denne type inserts:

INSERT INTO visitors VALUES (520315, 'Torlaborg', 'Torlaborg00000005',0);



Med tak til Wikipedia & Shmuel Csaba Otto Traian

SkySQL, for at udvikle videre på en branch-off af MySQL, ***MariaDB***.

Sidste år begyndte Google at samarbejde med MariaDB:

> *Google senior systems engineer announced the news at the Extremely Large Databases (XLDB) conference in Stanford, CA as part of his MySQL presentation (PDF Link). A Google spokesperson said, "Google's MySQL team is in the process of moving internal users of MySQL at Google from MySQL 5.1 to MariaDB 10.0. Google's MySQL team and the SkySQL MariaDB team are looking forward to working together to advance the reliability and feature set of MariaDB."*

(http://www.zdnet.com/google-quietly-dumps-oracle-mysql-for-mariadb-7000020670/)

Google er **i færd med at flytte** fra MySQL 5.1 til MariaDB [...] Det var i september 2013, og der er stadig ikke nogen nyheder om endelig migrering. Det er essentielt, fordi vi her taler om den type serversystemer, som kan køre med hundrede-tusindvis af samtidige brugere.

Facebook er totalt afhængig af MySQL i en egen, ombygget version, som kan fordele belastningen over hundreder af sites forskellige steder på jorden.

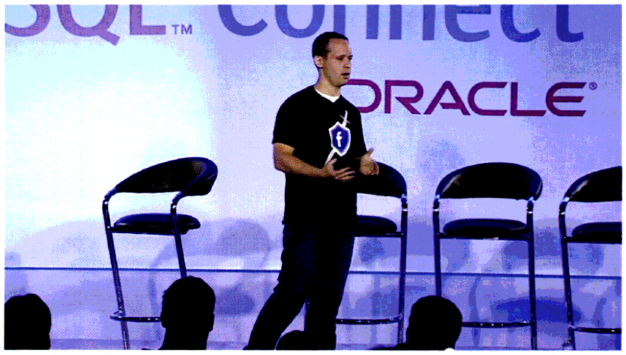
Harrison Fisk om MySQL og FB

På en konference for MySQL *(MySQL Connect 2013, Oracle Openworld 2013)* fortalte Harrison Fisk, Manager of Data Performance at Facebook:

"Jeg er involveret i flere teknikker, MySQL (selvfølgelig) Hbase og Hadoop.

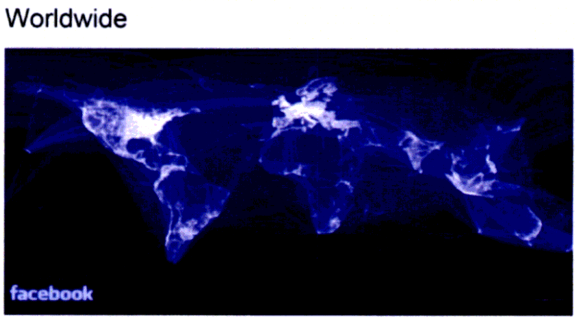
- *Vi har flere petabytes (mange 1000 Terabytes) data;*
- *vi ændrer 11.2 mio rækker (records/rows) i sekundet;*
- *vi læser 2500 millioner rækker i sekundet, det er altsammen efter cache, for vi har forskellige caching systemer,*
- *det er ca. 41.25 rows pr. query -*

Det er ialt ca. 24.2 millioner disk IOPS (In/Out operationer pr. sekund), så diskene har meget travlt."



Harrison Fisk, Facebook, fortæller at Oracle MySQL på flash bruger compression hele vejen - giver øget performance

Diagrammet nederst giver *kun* et overblik over applikations-stakken hos Facebook. Data kan ikke partitioneres geografisk.



Dette er ikke et kort - men visualisering af "friendships"

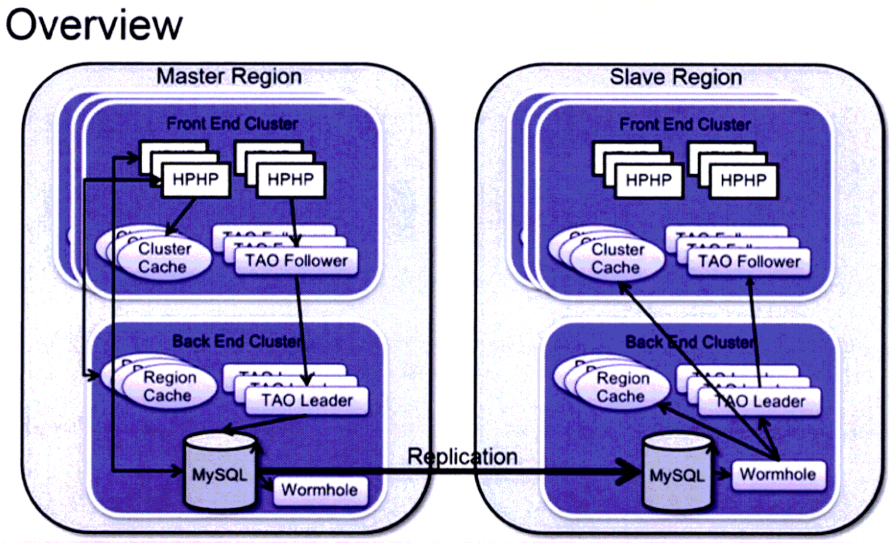
Facebook har iflg. http://www.statisticbrain.com/facebook-statistics/ 1.3 milliarder af brugere verden over. For at fordele belastningen slår danske brugeres søgninger /visninger først op i lokale driftscentre. Der er tusindvis af servere fordelt over mange lande. Overbliksskemaet viser, hvordan en bruger bliver for-bundet til ét front-end cluster, og derefter guides hans query (opslag på venner etc.) først til database med lokale, derefter - hvis der er behov pga. "friendships" - kan den guides videre ud til andre områders driftscentre.

Man bemærker også, at Fisk skriver HPHP på diagrammet, det er Facebooks patchede, kompilerbare PHP.

Replikeringen er ikke (kun) backup, men videresendelse af data til andre lokationer.

MySQL bruges kun når der skal laves disk-opslag for de brugere, der kun lejlighedsvis benytter Facebook. De ting, der spørges efter hele tiden, ligger i cache.

■



Facebooks applikationsstak i skemaform

Ruby, objektorienteret, med rødder i Perl, Smalltalk, Eiffel, Ada og Lisp

Ruby var navnet på en sten som symboliserer et stjernetegn - for en kollega til skaberen af programmeringssproget

Med andre ord, Ruby er ikke et akronym for noget i retning af "Radical Ultimative Best Yield" eller hvad man nu kunne forestille sig. Det er nærmest et tilfældigt navn, fordi et programmerings-sprog jo skal hedde et eller andet. Navnet blev fundet, før der var skrevet en eneste linie for sproget!

I de første år, fra 1993, var Ruby mest et lokalt japansk projekt; den første mailingliste blev åbnet i december 1995, samtidigt med offentliggørelse af version 0.95, men det var stadig udelukkende på japansk.

Skaberen af Ruby, Yukihiro Matsumoto, blev i 1997 ansat som fuldtidsudvikler for netlab.jp, og i 1998 fik Ruby sin egen engelske hjemmeside (på "Simple English", som er lettere at forstå for asiater og andre, der ikke taler flydende engelsk). Desuden fik Ruby et arkiv for applikationer, *Ruby Application Archive,* som blev standset i 2013 på grund af manglende ressourcer; man kan finde det på *archive.org,* eller man kan bruge en pakke-manager, RubyGems, hvorfra man kan hente programmer i sourceform. Der medfølger compile-vejledning, *.gemspec* - men man kan også bygge programmerne uden.

Den aktuelle version er 2.1.1 fra 24. februar 2014, det er et *semantisk versionsnummer,* det vil sige, at første tal er en ændring af interface (API, funktionsdefinitioner ændres, så man ikke kan bruge samme Ruby til gamle programmer), andet tal er en ændring af funktionalitet, men stadig bagud-kompatibelt.

Det tredie tal er fejlretning, bug-fixes.

Den nye version indeholder er ikke så forskellig fra 2.0. Man kan læse i NEWS filen, hvad de vigtigste ændringer er. En af dem berører at funktioner vil blive checket for om argumenter kan udelades - og omvendt, nødvendige argumenter vil udløse en fejl, (det skete ikke tidligere).

Ruby er

- ***objektorienteret,***
- ***dynamisk,***
- ***reflektivt.***

Objekt-orienteret betyder som bekendt, at man kan definere program-objekter, der behandles med egne, tilhørende funktioner. Et eksempel, kendt fra de fleste programmerings-sprog, er floating point og integers, som kan indgå i samme regnestykke, og hvor compileren automatisk vælger de rigtige konverteringer og aritmetiske funktioner. Objekt-orientering er praktisk for matematisk modellering af fysiske objekter.

Dynamisk betyder at Ruby kan ændre programmets opførsel på en måde, som statiske programmer får fastlagt ved compile-time, typisk en *eval* funktionalitet, som udfører kode, der genereres mens programmet kører - men der er også mange andre anvendelser af den *dynamiske funktionalitet* i et programmeringssprog som Ruby. Implementering af reflektivitet er et eksempel på anvendelse af den dynamiske funktionalitet, men går et skridt videre.

Dynamisk betyder også at man kan skrive ***a = 2*** og så er a en integer med værdien 2 - det vil sige uden at fortælle systemet hvad type ***a*** er.

Reflektivt betyder, at Ruby kan ændre på definitionen af en klasse undervejs i programmet, typisk fordi man udvikler et program fortløbende og tilføjer en funktionalitet. Det specielle er at man kan gøre det, mens programmet allerede kører.

Inspirationskilder

Ruby er påvirket af en lang stribe sprog, fra C++ over Ada og Perl til Lisp.

README filen beskriver Ruby på denne måde:

* Simpel Syntax
* **Normale** Objekt-Orienterede features(fx. class, method calls)
* **Advancerede** Objekt-Orienterede features(fx. Mix-in, Singleton-method)
* Operator Overloading
* Exception Handling
* Iterators og Closures
* Garbage Collection
* Dynamisk indlæsning af objekt filer(på nogle arkitekturer)
* Portabelt — kører på mange Unix og POSIX kompatible platforme, på Windows, Mac OS X, BeOS etc.

Det kan opsummeres til at Ruby supporterer flere forskellige programmerings-paradigmer (design-filosofier), funktions-programmering, objekt-orienteret, og imperativ programmering (kommando-programmering). De forskellige betegnelser skal tages med et gran salt, et objekt-orienteret program set som en helhed, er et imperativ, og skal sættes igang - med en kommando.

Den vigtigste egenskab for et programmeringssprog er lethed i koncept, vedligeholdelse og stabilitet, selvfølgelig. Og her scorer Ruby pænt! (eller bedre!) Ruby er skrevet i C programmerings-sproget.

Grundlæggende regler for Ruby

Man kan lave et ruby program ved at starte "ruby" og så ellers indtaste, - men det er ikke en interaktiv kommandofortolker. Det er først i det øjeblik, at man taster *end-of-file* (Ctrl-D) at programmet bliver kørt. ***irb*** er en interaktiv Ruby fortolker.

Men at Ruby kan læse fra tastaturet (standard input, stdin) gør at vi kan skrive et program på samme måde som et shell-script, - ligesom Perl, Awk, og mange andre fortolkere.

```
xterm
#!/usr/bin/ruby

puts "programlinie 1 koerer."
```

Gem filen som hallo-1.ruby og ***chmod hallo-1.ruby*** - derefter kan programmet køres fra kommandolinien:

```
xterm
sat2:/wb4/arb174 #./hallo-1.ruby
programlinie 1 koerer.
sat2:/wb4/arb174 #
```

At lave talvariable er - ligesom i awk og javascript - bare at tildele en talværdi til en identifier - i det øjeblik du nævner variabelen, bringes den til at eksistere.

```
#!/usr/bin/ruby

puts "programlinie 1 koerer."
c = Math.sqrt(3**2 + 4**2)
print "c er "
puts c

puts "c er #{c}"
```

3 i anden skrives 3**2, og 9+16 er selvfølgelig 25, hvilket programmet udtrækker kvadratroden af, den lægges i variabelen med navnet 'c'

Output fra kørsel af ovenstående program viser at sidste linie giver samme output som de to foregående linier.

```
sat2:/wb4/arb174 #vi hello-math.ruby
sat2:/wb4/arb174 #./hello-math.ruby
programlinie 1 koerer.
c er 5.0
c er 5.0
sat2:/wb4/arb174 #
```

Det interessante her er, at man kan skrive c ud med den almindelige puts() funktion, som automatisk konverterer tal-variabelen til en text-streng. For at få et tal ind i en streng kan vi bruge den specielle #{variabelnavn} notation.

I sidste linie bruges syntax med specialtegnet hash + krøllede parenteser #{*var*} til at få den numeriske variabel indsat som tekst i en anden tekst-streng. Hash-tegnet betyder i denne sammenhæng kun noget særligt hvis det efterfølges af krøllede parenteser (braces).

Sigils

Ruby ligner Perl i mange henseender. En af de forskelle, man straks får øje på, er at hvor Perl bruger ***sigils*** *(specialtegn foran identifiers som typemarkering)* bruger Ruby ikke nogen form for typeangivelse.

Som man kan se af artiklen om Ruby med kodeeksempler, er sproget intuitivt, men hvis man er vant til programmering, studser man i første omgang over hvad meningen med specialtegnene er, fx. kolon'er ":" foran en variabel i en klasse:

```
class Tink
    attr_accessor :navn
    attr_accessor :inventar, :vhaand, :hhaand

    def initialize
        @inventar = []
    end
end
```

Det er faktisk heller ikke til at se på en variabel, hvad den er for en. Det kunne man selvfølgelig klare ved at tilføje type-markeringer på identifiers.

Snabel-a "@" bruges for at vise, at en variabel er en lokal variabel (man kunne fristes til at tro at det er en arraymarkering). Med dollar-tegn "**$**" kan man angive, at variablen har globalt scope (kan ses overalt i programmet).

Arrays er ikke arrays på samme måde som i C-sproget. Det er som i Lisp (og Awk) lister og/eller associative arrays, som kan gennemløbes med en objekt-metode **.each** - fx.:

```
#!/usr/bin/ruby

samplearray = []
samplearray << "skruetraekker"

# p printer indholdet af hele arrayet:
p samplearray   # print viser ["skruetraekker"]

samplearray << "hammer"

puts samplearray[1]  # array index begynder med 0.

samplearray.each  do |tool|
  puts tool
end
```

Eksemplet viser, hvor mange måder man kan håndtere disse associative arrays.

Output er som man kan forvente det:

```
sat2:/wb4/arb174 #./hello-array.ruby
["skruetraekker"]
hammer
skruetraekker
hammer
sat2:/wb4/arb174 #
```

Ulempen ved Ruby (hvis der er nogen) er, at objekt-notationen meget let kan gå hen og blive ulæselig - især hvis man er vant til fx. C-programmering, hvor gode navne-konventioner ofte bevirker at man læser program source uden besvær. Men string-handling i C-libraries er til gengæld ulæselige, når man kommer ud over basis-ting, der skrives på stdout.

Ruby et stærkt værktøj til højniveau programmer, der håndterer store mængder tekst - som fx. web-serverside programmer gør.

■

Donald Axd

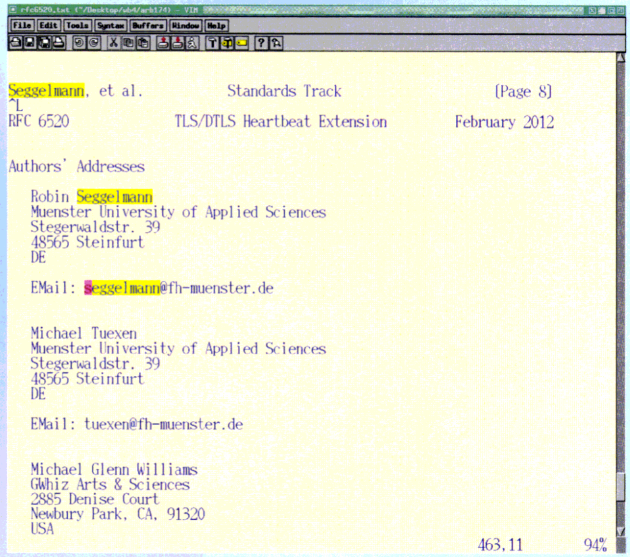
Faldgruber:

Identifiers for konstanter skrives med stort begyndelsesbogstav. En klasse er en "konstant" (det skulle man ellers ikke tro, når man rent faktisk kan lave om på den hele tiden!)

Redaktionen måtte experimentere i ganske mange minutter for at finde ud af at ***initialize*** *i array-eksemplet er et reserveret ord.*

Heartbleed: Procedurefejl eller forståelsesfejl/systemfejl?

Programmøren er medlem af OpenSSL teamet og er hoved-forfatter på specifikationen af Heartbeat extension, RFC-2620, se skærmdump nedenfor, og koden er gennemset af peer reviewer Stephen Henson ***inden*** den bliver brugt i næste release.



Seggelmann, et al. Standards Track [Page 8]
RFC 6520 TLS/DTLS Heartbeat Extension February 2012

Authors' Addresses

Robin Seggelmann
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

EMail: Seggelmann@fh-muenster.de

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstr. 39
48565 Steinfurt
DE

EMail: tuexen@fh-muenster.de

Michael Glenn Williams
GWhiz Arts & Sciences
2885 Denise Court
Newbury Park, CA, 91320
USA

463,11 94%

I et interview med avisen Sydney Morning Herald fortæller Robin Seggelmann: "Jeg arbejdede på at forbedre OpenSSL og havde afleveret flere fejlrettelser og tilføjet nye funktioner. I én af de nye funktioner kom jeg uheldigvis til at glemme at kontrollere længden på et array."

Alle kan lave fejl; men denne slags fejl må ikke slippe ud i en version til almindelig drift. Hvordan man kan undgå det?

Selv det simpleste program indeholder komplicerede design-beslutninger, skrev Kernighan & Ritchie i *The C Programming Language,* og viste, hvordan et lille program som udskriver den længste linie i en textfil, må beskytte sig mod input-linier, der er for lange. Var det den lektion, som OpenSSL programmøren ikke har forstået? Ja, åbenbart, eller også var han træt.

Men 1) hvordan kunne fejlen passere gennem review og
2) hvordan kunne den passere release proceduren, og
3) hvordan den blev opdaget, af hvem, 4) og under hvilke omstændigheder. Desværre er der ikke oplyst hvordan de trin er sket og derfor har vi kun redaktionens kvalificerede gæt:

1) Fejlen passerede gennem én reviewer, Stephen Henson, som ikke så fejlen. Det tyder på, at *koden er ulæselig.*

2) Release proceduren har muligvis slet ikke været igennem andre programmører. *Koden er ulæselig og release proceduren styres ikke af kritiske kompetente programmører.*

3) Fejlen blev opdaget af Google Security Team, Neel Mehta, eller nogle i samme type job (drifts-sikkerhed) før ham. *Udefra kommende sikkerhedsfolk med programmerings-indsigt kan finde fejl og rette dem - men vi ved ikke om heartbleed blev opdaget pga. at der var nogen, som var begyndt at udnytte sårbarheden.*

4) Omstændigheder ved opdagelsen (et kvalificeret gæt): Neel og de, der måtte have fundet fejlen før ham, har skullet overvåge en service, og har opdaget en eller flere forbindelser med alt for mange heartbeat-requests.

Kode review

Kode review forudsætter godt kendskab til alle niveauer i koden. Hvis en kode-review'er skal bedømme en sætning som vi så den tidligere,

```
buffer =
OPENSSL_malloc(1 + 2 + payload + padding);
```

så skal han vide hvad de forskellige variable står for, hvor de defineres, hvor de initieres, og hvad de skal bruges til. Man kan komme langt med navnekonventioner, d.v.s. man kalder variable noget, som antyder, hvad type og synlighed (scope) - og endda også hvad de indeholder, fx. kunne variablen payload her hedde ***plength_from_packet.*** Og hvis man vil have korte navne, er der rigeligt forkortelses-konventioner: ***pkt_plen*** ville være en stor forbedring. Det er iøvrigt én af den slags ting, en reviewer ville opdage uden at kende ret meget af koden.

Der har været kommentarer om, at et andet programmerings-sprog ville forhindre sådanne fejltagelser, men det er dels forkert og dels irrelevant ud fra hensynet til at dette er low-level kode, der skal være så effektivt som muligt.

Version2 og andre konkluderer, at det viser at der ikke er ressourcer nok til Open Source projekter. Desværre er det ikke hele forklaringen (så ville vi have en nem løsning) selv om det er påfaldende at der **kun** er 4 mand på OpenSSL teamet, som bruges af millioner af servere.

Både IBM og andre store firmaer har releaset professionel, velbetalt kode, som hovedsageligt er skrevet af en enkelt programmør, og som indeholdt fejl.

Kompetence kan fordeles på flere, når man opdeler et projekt i små bidder, ved at definere gennemtænkte funktionsopdelinger (API) og navnekonventioner. Det er her, at OpenSSL bløder mest.

API (og dermed design) for OpenSSL er blevet kritiseret af bl.a. Poul Henning Kamp i en nylig blog-entry på Version2, men allerede for 5-6 år siden kaldte PHK OpenSSL for skodkode.

Der er - som følge af offentliggørelsen af heartbleed - dannet et team af Theo de Raadt OpenBSD, som branch fra OpenSSL. De arbejder på at reducere koden (der er på ca. 440000 linier incl.test mv.) og de har allerede klippet 90000 linier ud ved at fjerne support for mindre brugte systemer og lign.

En ændring af alle de tre forhold: forståelsesgrad (større respekt for sikkerhed/kodekvalitet) design og procedure (eller systemfejl: ledelse, review, pre-release og netværks-overvågning) vil formindske risikoen for gentagelser.

Hvorfor gik der to år før den blev opdaget?

Det specielt uhyggelige Heartbleed er, at den ikke ses som en afvigelse i error-logningen. Det er jo ikke en "fejl" i datakommunikationen - det er simpelthen en forespørgsel om serveren venligst vil holde forbindelsen åben.

Alternative projekter

GnuTLS er ren GPL (Open Source licens) og foretrækkes af nogle distroer. Debian pakketeam anbefaler skift til GnuTLS. Den er ikke en drop-in erstatning for OpenSSL, så andre applikationer skal ændres så de ikke bruger /usr/lib/libssl.so*

- I parentes bemærket har mange servere været konservative og var endnu ikke kommet til anvendelse af ***libssl.so.1.0.1*** - og har derfor ikke været berørt af heartbleed.

■

Kommandocentralen

Monitorering af temperatur

i sidste nummer så vi lidt på uptime, det allerførste overblik over systemets belastningsgrad, som viste længden af *runqueue.*

```
sat2:/arb #uptime
 01:49:25 up 55 days, 11:12, 14 users,  load average: 0.10, 0.14, 0.11
```

Med uptime runqueue ser man hurtigt om overhead, system-belastningen, er opad- eller nedadgående, og programmet belaster ikke maskinen.

Hvis runqueue er 1, er systemet 100% belastet, CPU'en kører hele tiden. Hvis runqueue er 2, er systemet overbelastet, men ikke urimeligt. Den enkelte bruger vil knap nok bemærke det; det sker først ved runqueue 5+.

Men runqueue er lavet i en tid, hvor CPU'er kørte med samme frekvens hele tiden.

En almindelig PC har skalerbar CPU-frekvens, og når belastningen stiger, sættes frekvensen op. Derved øges temperaturen.

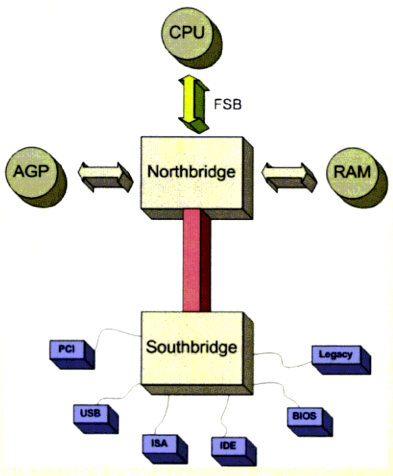
For at få aflæst CPU'ens temperatur, kan man bruge ***sensors,*** fra program pakke lm-sensors, og ***hddtemp.***

LM-sensors kommer med perl-script # ***sensors-detect*** til at finde ud af, hvilket chipsæt, computeren bruger. Når det er kørt, kan man fra kommandolinien køre $ ***sensors***

```
[...]
fan3:           0 RPM  (min = 10546 RPM, div = 128)  ALARM
fan5:           0 RPM  (min = 10546 RPM, div = 128)  ALARM
temp1:        +38.0°C  (high = +122.0°C, hyst =  +9.0°C)  sensor = thermistor
temp2:        +54.0°C  (high = +80.0°C, hyst = +75.0°C)  sensor = thermistor
temp3:        +33.5°C  (high = +80.0°C, hyst = +75.0°C)  sensor = thermistor
cpu0_vid:     +1.350 V
```

Der er kun 2 ventilatorer (fans) i den workstation, som har kørt ovenfor, og derfor kan man ignorere advarslerne for fan3-4 og fan5. At de er med, skyldes, at chipsættene er i stand til at kontrollere flere ventilatorer, hvis det er nødvendigt (fx. kan det være nødvendigt med en harddisk-fan.)

Måske er dette tidspunktet for en oversigt over PC'ens busser og chipsæt:



Man ser, at CPU'en "gemmer sig" bag en "bro", northbridge. Det skyldes at CPU'en, der internt kører 1 til 3 GHz, ikke kan kommunikere over boardets længere strækninger med den høje frekvens; det er radiobølge-frekvenser, og det er ikke nok med afskærmning af banerne på printkortet, for bølgelængden af 2.7 GHz er kun 11 cm og enhederne skal jo følges ad. Hvis man kan få RAM (DDR2) til at køre 800 MHz er det fint. En "bridge" er en bus-transfer enhed som kan videresende data fra CPU'en til de ydre enheder - men i langsommere tempo. Når det sker, må CPU'en vente hvis ikke den har data i sit egen interne lager, cache level1 eller leve2, der er en "hurtig kopi" af extern RAM.

Og **southbridge** er så kommunikation til de ydre enheder, som er endnu langsommere. Southbridge kaldes også I/O Hub, og styrer forbindelse til både ACPI (Advanced Configuration and Power Interface), USB, PCI mv. Det er kombinationen af ACPI digitale temperatur-sensorer i CPU og chipsæt at systemet kan styre strømforsyning til ventilatorerne.

Det er jo meget smart og kan reducere støjen i et serverrum væstentligt.

Fjern falske alarmer

Da man som regel får nogle warnings og alarmer på almindelige billigere workstations, laver vi et lille script, fx. sådan,

```
#!/bin/bash -a
sensors | grep -i -e temp1: -e temp2: -e Vcore:
```

hvorved vi kun viser de linier, som er interessante.

Hvis vi kalder filen Temp og gør den eksekverbar:

```
$ Temp
temp1:      +35.0°C  (high = +70.0°C)
Vcore:      +1.09 V  (min =  +0.00 V, max =  +1.74 V)
temp1:      +35.0°C  (high = +122.0°C, hyst = +9.0°C)  sensor = thermistor
temp2:      +45.5°C  (high = +80.0°C, hyst = +75.0°C)  sensor = thermistor
```

Og hvis vi ikke ønsker at blive mindet om at sensorerne iflg. lm-sensors databasen er en thermistor, så fjerner vi også det:

```
#!/bin/bash

sensors |
grep -i -e temp1: -e temp2: -e Vcore: |
gawk '{ print substr($0,1,56);}'
```

Hvis man laver mange scripts til aflastning i det daglige arbejde, er det min erfaring at det kan betale sig at

1) lægge dem i /usr/local/bin + checke at det er i PATH variablen

2) sætte dem i en menu, få en visuel reminder om at de er der,

3) arkivere dem i en distributionsfil, som følger med setup af nye maskiner.

I næste *kommandocentral* vil vi se på deployment systemer.

Donald Axel

Kære SuperUsers Venner

SUPERUSERS

Mit kære SuperUsers fylder 30 år:
Vi fejrer Super-WoodStock, en aften på gamle Karlebogaard, hvor mit orkester Santanas Venner spiller koncert, grillen er tændt, der er pølser, salat m.m. samt et godt glas eller to. Masser af tid til at hygge sammen med gode gamle venner.

INVITATION:
Jeg vil være meget glad for at se alle tidligere SuperUsers-kolleger, SuperUsers-kursister og andre gode venner af huset inkl. jeres mænd og fruer.

HVOR OG HVORNÅR:
Lørdag den 31. maj kl. 19:00 på Karlebogaard, Karlebovej 91, 3400 Hillerød.

TILMELDING:
Blot send en mail på 30@superusers.dk (skriv navne/antal)

30 glade hilsener fra
Brian

PS: Santanas Venner er trommer, congas, timbales, bas, piano/orgel, guitar, sang og Carlos Santanas musik, som jeg har spillet siden jeg var 14.
Se mere på:
W: www.santanasvenner.dk
FB: www.facebook.com/SantanasVennerOfficial
YT: www.youtube.com/watch?v=I6nQGYyfThI

Karlebogaard: Karlebovej 91 • 3400 Hillerød & **Kampehøjgaard**: Krajbjergvej 3, Vorre • 8541 Skødstrup
Tlf.: 48 28 07 06 • **Mail:** super@superusers.dk • **Web:** www.superusers.dk