HOVEDPROSJEKT 2005:

FORFATTERE:

CHRISTOPHER JOHNSRUD FULLU
KENNET FLADBY
HALVOR BORGEN
ROGER CARSON

Dato: 12. mai 2005

SAMMENDRAG AV HOVEDPROSJEKT

Tittel:	Skolelinux – Nettverksovervåkning	Nr: 8
		Dato: 12. mai 2005
Deltakere:	Christopher J. Fullu	
	Kennet Fladby	
	Halvor Borgen	
	Roger Carson	
Veiledere:	Intern veileder ved HiG: Erik Hjelmås	
	Ekstern veileder fra Skolelinux: Tosten Tøfte	
Oppdragsgiver:	Skolelinux	
Kontaktperson:	Tosten Tøfte	
Stikkord (4 stk.)	Nagios, Munin, Overvåkning, Administrasjon	
Antall sider:	Antall bilag:	Tilgjengelighet: Åpen

Kort beskrivelse av hovedprosjekt:

Prosjektgruppen har videreutviklet en overvåkningsløsning for å administrere skolelinux, først og fremst vha modulen Nagios som gruppen har tilpasset en Skolelinux Sarge–versjon

Gruppen har laget en server–klient applikasjon for å kunne utføre administrative oppgaver i et Skolelinux–nettverk Dette er gjort så generisk som mulig, slik at hva som skal overvåkes, og hvilke operasjoner som skal utføres, enkelt kan defineres og lages av en bruker, basert på scripts og konfigurasjonsfiler.

Applikasjonen består av en klient som samler data, en server som mottar data, og en GUI som behandler data. Disse er koblet til hverandre, og kalles henholdsvis Transmitter, Receiver og Listener under navnet AdminWorm

Forord

Hovedprosjektgruppe 8 ved dataingeniørlinjen ved HiG, har i sitt tredje og siste år ved høgskolen gjennomført et hovedprosjekt våren 2005 med tema nettverksovervåkning i Skolelinux. Gruppen ønsker å si takk til følgende fantastiske mennesker. Først og fremst ønsker gruppen å si takk til hele *Skolelinux* for et kjempefint prosjekt, og spesielt til *Tosten Tøfte* som har vært gruppens kontaktperson med Skolelinux gjennom prosjektperioden som også har hjulpet til med anskaffelse av viktig utstyr til gruppen. *Erik Hjelmås* som har vært gruppens interne veileder ved HiG. Takk til hovedprosjektgruppen *COPWALL*, som vi har delt grupperom med, og som har måtte holde ut med mye "plaging". Den andre Skolelinux gruppen, og ellers takk til alle fra "Fri programvare i skolen" som har bistått gruppen med svar på spørsmål i form av e–post eller news. Vil spesielt takke *Petter Reinholdtsen* og *Jimmy Olsen* som har bistått med hjelp rundt Munin, *Ragnar Wisløff* som har vært til stor hjelp i forbindelse med Nagios og under startfasen av prosjektet. Tusen takk til *vaktmesterene* som klarte å få frisk og passe temperert luft til å flyte gjennom "årene" til lufteanlegget. Hadde dere ikke kommet til unnsetning, ville vi strøket med alle sammen. Gruppen vil også si takk til *vaskedamene* som har tålt mye rot og som har bært bort, og opp, mange tallerkenstabler. Uten dere ville kantinen gått tom og det hadde blitt erklært krisetilstand på skolen ... Sist men ikke minst vil vi takke alle *kaffetraktere* rundt om på skolen som har måtte pumpe ut litervis med kaffe til ikke bare oss men også de fleste andre prosjektgrupper rundt om på skolen. Thor Hauknes for overnatting.
Underskrift fra alle gruppemedlemmer, samt sted og dato. også legge på underskrifter på alle fra gruppen.

Halvor Borgen	Christopher J. Fullu
Roger Carson	Kennet Fladby

Innhold

Tabeller

Figurer

Kapittel 1

Innledning

1.1 Problemområde

Dagens operativsystemer kan ofte deles i to. Den ene delen er de lukkede versjonene som for eksempel “Windows”[37] eller Mac OS[36]. Den andre delen består av *Open Source* operativsystemer som Linux[18], BSD[34] med mer. I henhold til GNU og GPL lisensiering, kan disse fritt kopieres, endres og tilpasses etter eget behov eller ønske[1]. Innenfor Linux finnes det utallige versjoner av operativsystemversjoner som for eksempel Mandrake(Mandriva)[25], Redhat[14], Suse[20], Debian[10]. Selv om de fleste av disse operativsystemene er noe mer kommersielle og utvikles av betalte utviklere, finnes de også som nedlastbare versjoner. Mange av disse utvikles av frivillige som igjen deler sitt bidrag med resten av verden. Debian er et av disse operativsystemene, og brukes av flere store bedrifter og private brukere over hele verden.

1.2 Avgrensning

Skolelinux er basert på Debian og er en ferdig konfigurert server/klient løsning som etter installasjon har det meste av tjenester og programmer ferdig satt opp og klar til bruk. Skolelinux er bra å bruke i skolesammenheng, da det er gratis og enkelt kan tilpasses forskjellige situasjoner. Det er billig i drift, og enkelt å både installere og administrere. Skolelinux baserer seg primært på å operere med halvtykke eller tynne klienter, som fører til at det kan benyttes eldre og ellers utdaterte maskiner som klienter. Det stilles noe større krav til servere, men er mye mindre krevende enn for eksempel Windows(2K,2003)[37] servere. I skolelinuxpakken følger også OpenOffice, Gimp, Munin, Nagios mm med. Munin og Nagios benyt-

[1] General Public Licens

tes i Skolelinux for å overvåke nettverket og lokale tjenester. Nagios kjøres på en hovedtjener med grafisk grensesnitt via Apache webserver/filtjener, mens Munin logger resultater fra systemtjenester lokalt, og legger disse i en database som kan åpnes via et web–interface.

1.3 Oppgavedefinisjon

Gruppens oppgave var å tilpasse og tilrettelegge Nagios for Skolelinux Sarge, som er neste versjon av Skolelinux, og for flerlags–overvåkning ved å ta i bruk nsca. For å kunne behandle dataen som kommer ut av Nagios må det utvikles en applikasjon for enkelt å kunne administrere brukere og prosesser over flere tjenere. Siste del av oppgaven var å tilrettelegge for bruk av data fra Munin slik at den kan samarbeide med Nagios ved hjelp av plugins og RRD–databaser.

1.4 Målgrupper

Det er to hovedmålgrupper for dette prosjektet. De som skal lese rapporten og de som skal bruke Nagios/Munin og AdminWorm. Det er jo selvfølgelig også mulig at fremtidige brukere leser rapporten, men gruppen anser det som mindre sannsynlig. Grunnen til dette er at rapporten inneholder mye teknisk informasjon om bla. hvordan applikasjonen er laget. Det er ikke sikkert fremtidige bruker(e) av applikasjonen er i stand til å forstå eller har behov for å vite noe om hvordan, og på hvilken måte, utviklingen har foregått.

1.4.1 Rapportleser

Denne rapporten er først og fremst skrevet med tanke på at sensor, veileder og oppdragsgiver skal lese den. Derfor er rapporten av en noe teknisk art, med beskrivelser av bl.a. valg som har blitt gjort underveis og tekniske spesifikasjoner. Rapporten er også av nytte for fremtidige videreutviklere. Det finnes mange dyktige utviklere der ute som kanskje ved et evt. senere hovedprosjekt eller i samarbeid med Skolelinux, har tid eller mulighet til å videreutvikle (Produktet). I den forbindelse er det veldig viktig å ha god dokumentasjon slik at eventuelle utviklere ikke trenger å bruke mye tid på å sette seg inn i tankegangen til de originale utviklerne. Det er også mulig at noen andre enn Skolelinux liker (Produktet) og vil implementere det i en annen versjon av Linux. Da vil det også være behov for å finne ut hvordan det fungerer. Selv om rapporten blir noe teknisk, er det ikke behov for en doktorgrad eller to for å kunne lese den, men litt erfaring eller kunnskap om systemutvikling og programmering er en fordel men ikke nødvendig. Mange

av de tekniske uttrykkene som er brukt, er forklart/definert i Appendiks A samt i (forklaringen av forkortelser hva det nå heter.)

1.4.2 Produktet

Mulige brukere av produktet vil først og fremst være lærere og/eller administratorer på grunnskole–nivå. Det regnes derfor med at den generelle datakunnskapen når det gjelder nettverksadministrasjon uheldigvis ikke er så veldig stor. Det å lage egne scripts kan kanskje være noe komplisert for en vanlig it–lærer, men ved hjelp av dokumentasjon og litt kunnskap innen programmmering bør det la seg gjøre. I en flerlags overvåkningssituasjon vil det antageligvis være mer erfarent datapersonell som bruker applikasjonen og Nagios/Munin, men det skal fortsatt være mulig for mindre erfarne bruker å kunne effektivt bruke produktet. Produktet er derfor laget for å være enkelt å bruke for et bredt spekter av brukere.

1.5 Formål/hvorfor valg av oppgave

Skolelinux benyttes mer og mer i grunnskolene, ikke bare i Norge men ellers i verden også. (Dette blir også forklart mer i dybden senere i kravspesifikasjonen.) Det er da også et økende behov for å kunne opprettholde sikkerheten på nettverket samt ha mulighet til å overvåke og administrere nettverket på en enkel og brukervennlig måte. Nagios og Munin er de foretrukne programmene/tjenestene som er valgt for bruk i Skolelinux Sarge. De følger med installasjonen men må tilpasses bla. flerlags administrering for å kunne la skolelinux være administrerbart fra en hovedsentral. “Problemet” er at de to programmene/tjenestene bare rapporterer om statuser og generelt informasjon om systemet. Det eneste en administrator kan gjøre med informasjonen er å se på den. Det er lite med muligheter for å enkelt gjøre noe med systemet, som for eksempel å fjerne en bruker fra systemet slå av applikasjonen eller starte stoppe tjenester. Foreløpig gjøres dette manuelt, er ofte vanskelig og tar en del tid. Siden Skolelinux skal benyttes av administratorer på skoler, hvor det er mangel på erfarne driftere, er det derfor behov for å lage en applikasjon hvor det kan legges til skripter som gjør en slik jobb automatisk.

1.6 Egen Bakgrunn og forutsetninger/Hva må læres

1.6.1 Generell Bakgrunn

Gruppen består av fire medlemmer som har forskjellige bakgrunn innenfor data. I løpet av høstsemesteret 2004 fordypet gruppen seg innenfor de områdene som

har vært av interesse. To av medlemmene er i hovedsak driftere og to er programmerere. Dette prosjektet er ideelt for gruppen, siden gruppen her har mulighet til å fordype seg i drifting av et alternativt operativsystem, og gjøre mye programmering/ skripting i form av plugins og andre tilleggsapplikasjoner. Programmering kan foregå på flere forskjellige språk som "C++, Java, Perl", og vil innebære at det taes i bruk avanserte programmeringsteknikker. Overvåkningsverktøyet "Nagios" er valgt som verktøyet Skolelinux skal bruke for overvåkning, som igjen passer veldig godt for drifterne i gruppen. Den har web-interface og gjør det mulig å få et godt innblikk i hvordan nettverk– og overvåkingstjenester fungerer. Det vil også kreve en god forståelse for nettverk og de tjenestene som kjøres hvis det skal lages spesifikke plugin og moduler, bl.a felles administrasjon, og ellers mer avanserte nettverksfunksjoner. (Da gruppen satte seg sammen for å komme på en oppgave, la gruppen frem et oppgaveforslag hvor ønsket var å lage et tilsvarende program fra bunnen. Gruppen foreslo oppgaven for Erik Hjelmås som allerede hadde fått en tilsvarende forespørsel fra Skolelinux. Etter en liten samtale med Erik Hjelmås, ble gruppen enige om at dette prosjektet var noe gruppen kunne tenke seg å gjennomføre.)

1.6.2 Gruppemedlemmene

Kennet Fladby

Kennet er en av to kodere som har hatt hovedansvaret for koding av plugins og skripter til applikasjonen. Han vil også bistå med administrering av nettverk og vil hjelpe til med dokumentasjon.

Christopher J. Fullu

Christopher er den andre av de to koderne og har i hovedsak stått for GUI-delen av applikasjonen samt hjulpet til med administrasjon av nettverket og dokumentasjon.

Halvor Borgen

Er Webmaster og er ansvarlig for oppdatering av gruppens hjemmeside i form av "blogg", linker, sider etc. Ellers har han tilrettelagt Nagios og for Sarge, utarbeidet et rammeverk for sammarbeid mellom Nagios og Munin, hjulpt til med koding av plugins og administrering av nettverket, samt lekt ordbok på gruppens dokumentasjon og prosjektdokumenter.

Roger Carson

Er gruppeleder og har ansvaret innkalling til møter, skrive referat fra møter, skriving av prosjektdokumenter ved bruk av LaTeX 2ε, og har sørget for at alle dokumenter er i henhold til regler/retningslinjer. Ellers har han hjulpet til med administrering av nettverk, oppdatering av hjemmesiden og med dokumentasjon.

1.7 Tidligere prosjekter om Skolelinux, administrasjon/overvåkning

Det er tidligere bare vært gjennomført ett skolelinux–hovedprosjekt som har fokusert på det med nettverksovervåkning og administrasjon men bare ett har tatt i bruk Nagios. Tittelen på hovedprosjektet var "Skolelinux–Overvåkningsstem"[27] og ble gjennomført våren 2003. I det prosjektet satte gruppen seg inn i overvåkningssystemer for Skolelinux generelt og spesielt Nagios med tilhørende funksjoner. De testet mange forskjellige nettverks–overvåkningssystemer som Big Brother[29], The Angel Network Monitoring[22], snort[28] mm. Overvåkningssystemene ble installert testet, vurdert og det endelige valget falt på Nagios. Målet var å tilrettelegge Nagios for den daværende Skolelinux versjonen, ved at de måtte i første omgang lage stabile debian pakker for Nagios. Nagios måtte testes og kompileres inn i Skolelinux helt fra bunnen av. De fikk laget debian pakkene men de rakk ikke å få fullstendig integrert Nagios med Skolelinux. De anbefalte/satte opp en del punkter om hva som burde gjøres videre mtp fremtidige prosjekter som mer avanserte Perl skripter og overvåkning av store nettverk. Det er her gruppen vår har tatt over. Gruppen har på en måte fortsatt der forrige gruppe sluttet. Det var et annet skolelinux hovedprosjekt samme året, hvor problemområdet omhandlet overvåknign/administrasjon. Prosjektet "Skolelinux–User Administration"[8] fokuserte på å implementere en automatisk og enkel løsning for printerkvote og LDAP [2] netgroups i Skolelinux.

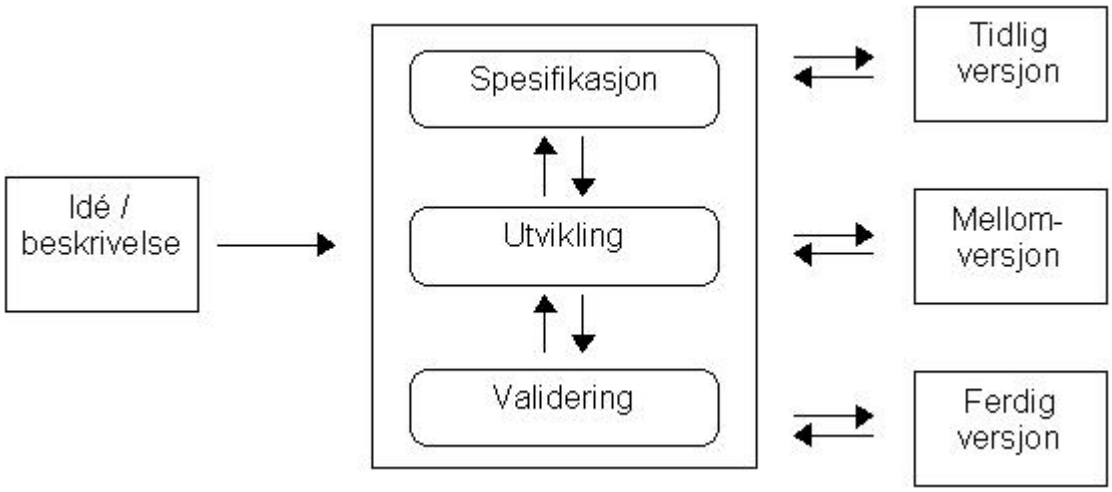
1.8 Utviklingsmodell

Inkrementell modell – kort forklaring av modellen

- Denne modellen som vist på figur 1.1 bygger på at prosjektet deles inn i flere små vannfallsmodeller, en for hvert inkrement.
- Oppdeling i kjerneprodukt + ekstramoduler.

[2] Lightweight Directory Access Protocol

- Hvert inkrement er et operativt produkt som kan leveres til kunden.
- Kjerneproduktet kan bli ferdig med lite folk, kan ha liten funksjonalitet.
- Delvis realiserte prosjekter er av verdi.
- Vanskelige deler kan utsettes, det man forstår kan gjøres først.
- Inkrementer kan tidsmessig være vanskelige å fastsette, siden oppgavene vi får varierer i vanskelighetsgrad og arbeidsmengde.



Figur 1.1: Inkrementell modell

Se vedlegg B.1 og B.2 for tidsskjema/gannt skjema, hvor de forskjellige inkrementene er illustrert.

1.9 Arbeidsformer

1.9.1 Grupperommet

Gruppen deler rom med en annen gruppe som sitter adskilt med skillevegger. Det er *fire* gruppemedlemmer som forklart over, som har hvert sitt ansvarsområdet og sine spesielle oppgaver. Utvikling/jobbing har i hovedsak foregått på grupperommet mandag til fredag mellom kl 10.00 og 16.00, mens helgejobbing og ettermiddagsjobbing har blitt gjort på eget initiativ. Gruppen jobber tett sammen, og valg og avgjørelser tas når alle er tilstede.

1.9.2 Møter

Gruppen har sammen med arbeidsgiver, så sant det har vært praktisk mulig, hatt ukentlige møter hvor gruppen har gitt en status på fremgangen. Hvis noen uforutsette spørsmål dukket opp har gruppen fått svar eller hjelp av oppdragsgiver under møtene. Veileder har også vært med på noen møter og har bidratt med kommentarer og synspunkter, og har hjulpet til hvis det var noe som gruppen sto helt fast på. Sammendrag fra disse møtene har blitt lagt ut på gruppens hjemmeside og lastet opp på CVS'n samt sendt ut til alle gruppemedlemmene, oppdragsgiver og veileder. I startfasen av prosjektet har gruppen også vært på Kapp som er et av Skolelinux–kontorene i Norge.

1.9.3 Fremdriftsplan

Fremdriftsplan og valg av utviklingsmodell er beskrevet i *kravspesifikasjonen* kapittel 2. Gruppen har holdt seg godt etter planen. Det har vært vanskelig å vite hvor mye tid de forskjellige delene av prosjektet ville ta siden det bl.a har innvolverte mye avansert nettverksprogrammering.

1.10 Øvrige roller

1.10.1 Oppdragsgiver

Tosten Tøfte har vært gruppens oppdragsgiver fra Skolelinux. Han presenterte oppgaven for gruppen og har vært gruppens hovedkontakt med Skolelinux og "Fri programvare i skolen"–miljøet. På de ukentlige møtene, har Tosten kommet med innspill, kommentarer, forklaringer og ellers fulgt godt med på prosjektets progresjon. Tosten Tøfte har også med jevne mellomrom lagt ut en status på prosjektprogresjonen på Skolelinux–newsgruppen. Tosten Tøfte v/Skolelinux har bistått med en test–server og klienter på skolelinuxkontoret på Kapp. Han har også bistått med utstyr i form av en svitsj og en datamaskin som har vært kritiske komponenter for fremgangen av prosjektet. På "Fri programvare i skolen"–samling, som blir forklart senere i kapittel 6, hjalp Tosten Tøfte gruppen med å komme i kontakt med forskjellige mennesker innenfor dette miljøet. Noen av disse hadde spisskompetanse innenfor Munin og Nagios, og dette var til stor hjelp for gruppen.

1.10.2 Veileder

Erik Hjelmås har vært gruppens interne veileder ved HiG[15]. Han var den som først foreslo Skolelinux prosjektet og satte gruppen i kontakt med Tosten Tøfte.

Veileder har gitt gruppen ganske fri tøyler, men har fulgt nøye med på progresjonen og har kommet med kommentarer og innspill hele veien. Hvis gruppen hadde et problem, har veileder hjulpet til og svart på spørsmål. Spesielt i området med sikkerhet, har han vært til stor hjelp på "Perl" og sikker overføring med applikasjonen. Ellers under utforming av de forskjellige obligatoriske innleveringene som for eksempel forprosjekt og denne rapporten har Erik Hjelmås forklart og gitt tips til hvordan utformingen burde være.

1.11 Praktiske opplysninger

Hele prosjektrapporten er skrevet ved hjelp av LaTeX 2ε. Gruppen har brukt skrift størrelse "12", med *fire* overskriftsnivåer (kapittel, seksjon, underseksjon og underunderseksjon) hvor overskriftene har lik størrelse i forhold til hva slags type overskrift det er. Gruppen følger mest mulig standard LaTeX 2ε, men siden rapporten er skrevet på norsk, har pakkene "norsk, latin1, T1" blitt lagt med for å bl.a kunne bruke æøå og for å få riktig orddeling. Kodesnuttene som er tatt med som eksempel i teksten og i vedlegget, er lagt i et *alltt* eller *verbatim*. Det betyr at kompilatorene lar teksen, som for eksempel linjeskift avstand mellom ord og skrifttypen være mer eller mindre uforandret. Beskrivelse av figurene. Muligens med at vi har dem i begynnelsen.

1.12 Rapportorganisering

Denne rapporten bygger på en mal ("Vedlegg H"[16]) som høgskolen har lagt ut på prosjektets hjemmeside. Gruppen har basert seg på malen og tilpasset den etter behov slik at det passet til vårt prosjekt. Den inneholder i hovedsak en innledning, hoveddel, avslutning, litteraturliste og til slutt vedlegg. Noe av rapporten er skrevet på engelsk. Dette gjelder da dokumentasjon av koden/kommentering av koden. Grunnen til dette er at Skolelinux/Debian EDU er et internasjonalt produkt som utvikles og videreutvikles av utviklere fra mange forskjellige land. Det er derfor behov for å kunne ha et mer universelt språk som kan forstås av mange. De ulike kapitlene inneholder følgende, og er organisert slik:

Kapittel 1: Innledning

Den første delen av rapporten er en innledning, som er skrevet for å gi en kort og forståelig forklaring på bl.a oppgavebeskrivelsen, arbeidsformer, rapportorganiseringen mm. Innledningen er forsøkt å være skrevet slik at det ikke skal være behov for noen spesielle forkunnskaper, men hvis det er behov for forklaringer følger det med en ordforklaring i vedlegg A.

Kapittel 2: Kravspesifikasjon

Dette kapittelet beskriver mer nøye hva kravene fra oppdragsgiver er, og hva slags begrensninger og rammer prosjektet har. Det blir også gitt en mer grundig forklaring på oppdragsgiver og organisasjonen Skolelinux.

Kapittel 3: Design

Design kapittelet er konseptuell tankegang på hvordan løse oppgaven

Kapittel 4: Implementering

Her beskrives det hva som har blitt gjort og hvordan det ble løst.

Kapittel 5: Testing

Hvordan testing har foregått og hvordan feil eller *bugs* har blitt behandlet

Kapittel 6: Samarbeid med "Fri programvare i skolen" miljøet

Kapittelet om "Fri programvare i skolen" tar for seg hvordan det var å jobbe med "open source"–miljøet, utviklersamling, samt bruk av news og CVS.

Kapittel 7: Konklusjon

En konklusjon på hva gruppen har gjort. Kritikk av oppgaven, tanker om fremtidige påbygninger av oppgaven.

Kapittel 8: Litteraturliste

Referanser til bøker, web-sider ol. som ble benyttet under prosjektet.

kapittel 9: Vedlegg

Her ligger alle vedleggene, som defenisjoner brukermanualer, og tekniske beskrivelser.

Kapittel 2

Kravspesifikasjon

2.1 Introduksjon

2.1.1 Bakgrunn

Selve organisasjonen [27]

Fri programvare i skolen er en brukerforening som er en idealistisk organisasjon som arbeider for å tilrettelegge og informere brukere av "fri programvare i skolen" i norske skoler. SLX Debian Labs er en privat stiftelse, som fungerer som det juridiske hjemmet til Skolelinux (avtaler, kontaktpunkt for det offentlige osv.) Denne organisasjonene har nå 4 ansatte, hvorav 2 er utviklere, de andre jobber med samfunskontakt, markedsføring og nå salg av Skolelinux konseptet. Prosjektet Skolelinux bruker "gjørokrati" som arbeidsmetode. Det vil si at de som fysisk gjør noe, bestemmer. Selv om noen fort kan gå seg bort på veien, viser det seg at denne metoden, flere hoder arbeider raskere og bedre, fungerer bra. Denne metoden løser også problemet med venting i forbindelse med at "noen" er mer opptatt av styring og kontrol, enn å nå målene raskt og effektivt. Prosjektet er avhengig av at de frivillige bidragsyterne legger frem sine bidrag i versjonskontrollsystemet *CVS* med jevne mellomrom. Ved å bruke dette versjonskontrollsystemet, kan andre få innsyn i løsningene , dokumentasjon etc og fortsette arbeidet der en annen slapp. De som regelmessig deltar i utviklingen av Skolelinux består av ca 120 utviklere, 15 oversettere og de fire ansatte. Frem til 1. april 2004 har det blitt lagt ned 62 000 arbeidstimer i prosjektet.

Produktet "Skolelinux"

Selve produktet, Skolelinux, er et operativsystem som er laget for å enkelt kunne skreddersys etter skolens behov og ressurser. Skolen skal ikke behøve å sette

sammen mange forskjellige enkeltkomponenter, men kunne basere seg på én enkel løsning. Skolelinux er utviklet av "skolefolk", for skolefolk, og er derfor høyst tilpasset skolebruk og er laget for å være enkelt og billig i drift. Det er betraktelig mye billigere enn for eksempel mange Windows-løsninger. Skolelinux gir på en enkel måte elevene egne brukernavn, filer og webtjenester. Skolelinux brukes over hele verden og er tilgjengelig på 42 språk inklusive norsk og nordsamisk. Det leveres med OpenOffice, som er en gratis kontorpakke, og innholder bla. tekstbehandlere og andre MS–Office lignende programmer. Skolelinux er en utvidelse av *Debian*, som er en Linux distribusjon kjent for god sikkerhet og stabilitet.

2.1.2 Kort om krav til systemet

Vi skal lage en applikasjon og en modul, som brukes til hhv. administrering og overvåking av nettverk. Det er ment som hjelpeverktøy for primært systemadministratorer, men også til lærer som sitter i et klasserom og skal passe på elevene. Applikasjonen skal i utgangspunktet hente data fra hoster [1] og bruke informasjonen videre. Det skal være mulig å legge til mange forskjellige scripts, som betyr at GUI delen må være veldig fleksibel. På økonomi siden regner vi ikke med noen kostnader siden vi baserer oss på open–source med GPL lisensiering. Tidsplanmessing henvises det til fremdriftsplanen i forprosjektet.

2.1.3 Kort om systemets omgivelser

Modulen bygges fra to eksisterende tjenester, Nagios og Munin, som skal konfigureres og tilpasses hverandre og Skolelinux. Applikasjonen lages fra bunnen, og lages med tanke på at den skal brukes i Skolelinux, siden Skolelinux Sarge er basert på *Debian* vil den også fungere i et vanlig Linux miljø. Både applikasjonen og modulen kjøres fra hovedtjeneren. På en vanlig arbeidsstasjon vil det kjøre en DaemonD.1, som brukes av applikasjonen. Munin/Nagios modulen kjøres på hovedtjeneren og vil ha et web–interface som er tilgjengelig for alle med godkjent brukernavn og passord.

2.1.4 Dokumentets struktur

Dokumentet er delt inn i *fem* seksjoner.

- Introduksjon, hvor vi forklarer litt om bakgrunnen for prosjektet og forklarer litt fagspråk

[1] Hoster er nettverksenheter på et nettverket

- En brukerbeskrivelse som forklarer litt om hvem og hvordan det skal brukes samt noen overordnede begrensinger.
- Funksjonell spesifikasjon som forklarer hvordan vi skal håndtere sikkerhet, hvordan det hele er satt sammen.
- Den nest siste seksjonen tar for seg mer spesifikt begrensinger innen software, *OS* og hardware.
- Siste seksjon tar for seg aspekter rundt installasjon, med tanke på hardware, overgang fra det gamle til det nye, samt opplæring.

2.1.5 Referanser

Vi vil være ukentlig i kontakt med veileder og oppdragsgiver som kommer med hjelp innspill/kommentarer. Ellers bruker vi mye Internet og mye av informasjonssøking foregår ved hjelp av Google, som vi anser som vårt viktigste verktøy. Ellers har vi kontakt med diverse skolelinux– og Nagios/Munin–administratorer ved hjelp av mail og news. Munin Nagios manualer benyttes hele tiden.

2.2 Brukerbeskrivelse

2.2.1 Omgivelser

Det forutsettes at overvåkningsmodulen Munin/Nagios kjøres på et Skolelinux Sarge nettverk. Dette er da den nyeste versjonen av Skolelinux som fortsatt er i "testing" fasen. Vi følger derfor Skolelinux standarder mtp. nettverksoppbygning og konfigurasjon av systemet. Systemet vil da kunne kjøres på ellers utdaterte maskiner i form av tynne eller halvtykke klienter, en hovedtjener og en tynnklienttjener. Denne løsningen skalerer i stor grad, men vi har ikke mulighet til å lage et stort nettverk grunnet penger og plass, til å ha så mange enheter på nettverket vårt. Overvåkningsmodulen er beregnet på en flerlagsovervåkning som vil si at en overvåkningsmodul kan overvåke en annen. Det kan da brukes til intern administrasjon innad i en skole, og på en sentral enhet som overvåker flere enheter samtidig. Når det gjelder flerlagsovervåkning, bygger applikasjonen på noen av de samme prinsippene. Den skal kunne brukes av en lokal drifter, administrator og eller lærer, men skal samtidig også kunne benyttes av en sentral administreringsenhet. Den lages også primært for et Skolelinux miljø, men siden skolelinux i bunn og grunn er en *Debian* versjon, vil den også fungerer i et annet linuxmiljø uten større komplikasjoner. Både applikasjonen og modulen installeres på hovedtjeneren hvor informasjonen for modulene vil være tilgjengelig ved hjelp av et

webinterface mens et GUI–interface for applikasjonen lar brukeren benytte seg av de mulighetene som er der. Alle maskinene er koblet sammen og konfigurert etter Skolelinux spesifikasjoner og standarder, og det benyttes svitsjer og eller router for å sørge for kommunikasjon mellom enhetene. Applikasjonen og modulene overvåker en mengde tjenester, som for eksempel SMTP, POP3, HTTP, NNTP, PING og generelt Linux tjenester.

2.2.2 Systemets brukere

Brukere av Nagios/Munin modulen og applikasjonen vil være alt fra en lærer helt ned i grunnskolen som følger med på de maskinene som elevene bruker, samt det lokale it–personellet ved en skole, til en administrator på en sentrale administrasjonssentral. Både Nagios og Munin er enkle å ta i bruk via et webinterface. Det er veldig mye informasjon med i bildet her, og noe opplæring på hvordan finne frem og tolke det som Nagios rapporterer anbefales. Applikasjonen må være enkel i bruk og kreve lite opplæring og lages dynamisk slik at nye scripts enkelt skal kunne legges til. GUI–delen tilpasser seg automatisk deretter med nye knapper og valg ved hjelp av skripter. Den med størst innflytelse på utformingen av applikasjonen vil være vår oppdragsgiver, Tosten Tøfte. Siden han jobber for Skolelinux og bruker skolelinux jevnlig, kan han komme med synspunkter på GUI og generelt på hvordan gjøre applikasjonen mest brukervennlig for Skolelinux–brukere. Både modulen og applikasjonen vil være ferdig konfigurert og vil kreve minimalt med konfigurering og tilpasning etter installasjon av Skolelinux.

2.2.3 Operasjon

Modulene skal kunne brukes for overvåking av et lokalt nettverk (LAN) og et eksternt nettverk over Internet. Modulen skal oppdage feil på nettverket og på maskiner og skal rapportere status og evt feil. Munin skal være et grafisk hjelpemiddel til Nagios, oppdager Munin en verdi som er over en viss grense kan den sende melding til Nagios som rapporterer videre. Den rapporter med *tre* nivår, *OK,Warning, Critical*. En host kan også få tilstandene *State Down* hvis den ikke svarer på ping. Dette vises grafisk i form av grønne, gule og røde varsellamper. Applikasjonen er beregnet for bruk på et vanlig lokalt nett (LAN). Den skal også brukes til overvåking av tynne eller halvtykke klienter. Den skal hente inn informasjon om hvem som er logget på de forskjellige maskinene. Det mulig å avslutte prosesser eller logge ut brukere på en host. Her er sikkerhet viktig, siden det er mye informasjon som sendes til og fra serveren. Det vil da antageligvis bli benyttet SSH for å sikre overføringen.

2.2.4 Aspekter omkring livssyklus

Både applikasjonen og modulene vil muligens bli pakket sammen og lagt med i en Skolelinuxdistribusjon. Munin/Nagios er allerede open–source og applikasjonen vil også bli lisensiert under GPL lisensiering.

- Vedlikehold av systemet vil bli gjort etter behov, og siden det er open–source kan hvem som helst modifisere, rette opp feil osv.
- Modulen er åpen for nye plugins, og vil også fungere utenfor skolelinux men vil da antageligvis kreve litt modifikasjon. Applikasjonen er veldig fleksibel og er i bunn og grunn et skall som kan bruke mange forskjellige skripter. Brukere vil selv kunne lage egne skripter som kan legges til.
- Både modulen og applikasjonen er tenkt som en del av Skolelinux Sarge versjonen, og vil da konstant være under testing og kontroll. Under utvikling vil filer lastes opp på CVS'n. Der kan alle prøve ut det vi jobber med og gi tilbakemelding.
- Versjonskontroll utføres i form av CVS, og dokumentasjon skrives på engelsk og legges med.

2.2.5 Begrensning

De eneste begrensingene for bruk av modulen eller applikasjonen, er at det følger standard oppsett av Skolelinux og at applikasjonen ikke kan kjøres i konsoll.

2.3 Funksjonell spesifikasjon

2.3.1 Sikkerhet

Modulen bygger på standard Linux kommandoer og "regler", ved at du må være "root" for å endre på på konfigurasjonsfiler. Alle kan se på informasjonen som Munin har lagret, mens Nagios krever innlogging. Applikasjonen benytter seg av SSH og SSL for å sikre at autentiserte og krypterte meldinger kan sendes frem og tilbake.

2.3.2 Oppstart og nedtaging

Overgangen fra den eksisterende versjonen til den nye blir veldig enkel. Den vil følge med en ny versjon av Skolelinux Sarge CD'n. For å installere er det bare å installere en server på nytt.

2.4 Begrensninger

2.4.1 Softwaredesign begrensinger

Softwarestandarder og språk

GUI delen av applikasjonen lages i Qt [32], som er et bibliotek for C++. Scipts skal lages i Perl. Nagios/Munin sitt webinterface, kan åpnes ved hjelp av de fleste web–browsere som for eksempel Mozilla eller Firefox. Nagios kan kjøre alle eksekverbare filer, og scripts kan skrives i de fleste språk.

Software pakker/verktøy

Nagios og Munin med tilhørende pakker ligger ferdig med Skolelinux Woody, og må tilpasses Sarge.

Softwarekommunikasjonsstandarder og grensensitt

Siden Skolelinux benytter seg av Ethernet 10/100/1000 Mbps, benytter applikasjonen og modulen seg av TCP/IP protokollen for å kunne kommunisere med andre maskiner/nettverksenheter på nettverket.

Operativsystem

Både modulen og applikasjonen er laget for bruk i Skolelinux Sarge, men siden "SLX" bygger på Debian så vil også applikasjonen fungere bra i et debian miljø. For å kunne benytte applikasjonen anbefales det å benytte KDE vindushåndterer. I Skolelinux er KDE standard.

2.4.2 Hardwarekrav og omgivelser

Vi følger bare de standardkravene som Skolelinux har når det gjelder hardware.

2.5 Aspekter omkring installasjon

2.5.1 Hardware installasjon

Vi har ikke behov for noen form for spesiell hardwareinstallasjon utenom det Skolelinux trenger til vanlig. Når vi er ferdig med applikasjonen og får testet den, lager vi en skolelinux pakke som følger med en skolelinuxdistribusjon. Det samme vil da også gjelde modulen som allerede følger med i standard skolelinux i form av Munin og Nagios.

2.5.2 Overgang/omlegging

Det vil være minimal merkbar forskjell på systemet. I det nåværende systemet finnes ikke tilsvarende applikasjoner som den vi lager. Modulen vil heller ikke føre til noen forandring. Web–interfacet vil forandres litt, men ikke mye.

2.5.3 Opplæring

Det vil heller ikke være noe stort behov for opplæring. GUI–delen av applikasjonen er veldig simpel og intuitiv, mens server delen krever litt mer innsikt i hvordan det skal settes opp. Det er forklart i brukermanualen hvordan dette skal gjøres. Nagios og Munin er mer eller mindre lik det nåværende systemet i utseende og bruk.

Kapittel 3

Design

3.1 Nagios

3.2 Videreutvikling av Nagios

I denne delen av *Design* kapittelet, skal vi forklare hva gruppen har fått gjort med Nagios, og hvorfor. Her følger en kort oversikt over temaer/oppgaver som gruppen har gjort:

- Satt oss inn i nagios–cfg fra Ragnar Wisløff, for så å lage en ny pakke helt fra bunnen.
- Modifisert eksisterende Nagios–oppsett for et Skolelinux Sarge–miljø.
- Tilrettelagt for distribuert overvåkning.
- Lagd en løsning for enkel navngivning av hoster for distribuert overvåkning.

3.2.1 Kort om Nagios

Nagios er laget for å overvåke hoster, deres tjenester og ressursbruk som for eksempel webserver, diskplass, load, sshd, dns etc. Programmet er en ren overvåkningstjeneste, dvs at det er ikke mulig å gjøre noe aktivt via Nagios med tjenester annet enn å se om de er oppe eller ikke, med mindre du ønsker å skrive tilleggsmoduler til Nagios. I Skolelinux brukes Nagios til akkurat dette siden løsningen er veldig fleksibel og relativt enkel å sette seg inn i med tanke på å skrive egne plugins og å konfigurere selve oppsettet.

3.2.2 Nagios i Sarge

Skolelinux Sarge var, ved prosjektets start, i et tidlig stadium, så utviklingen startet med en noe ustabil versjon, det ble oppgradert til nyere og bedre versjoner etterhvert. Det utvikles fremdeles for å få ut en ferdig stabil versjon av Skolelinux Sarge, og i siste stadium av testing med Nagios var prerelease pr01 installert på lab. I skrivende stund er nyeste debian–edu image *pr01* fra 11. mars 2005, samme som systemet på laben ble testet med, men daglige Sarge–image er å få tak i.

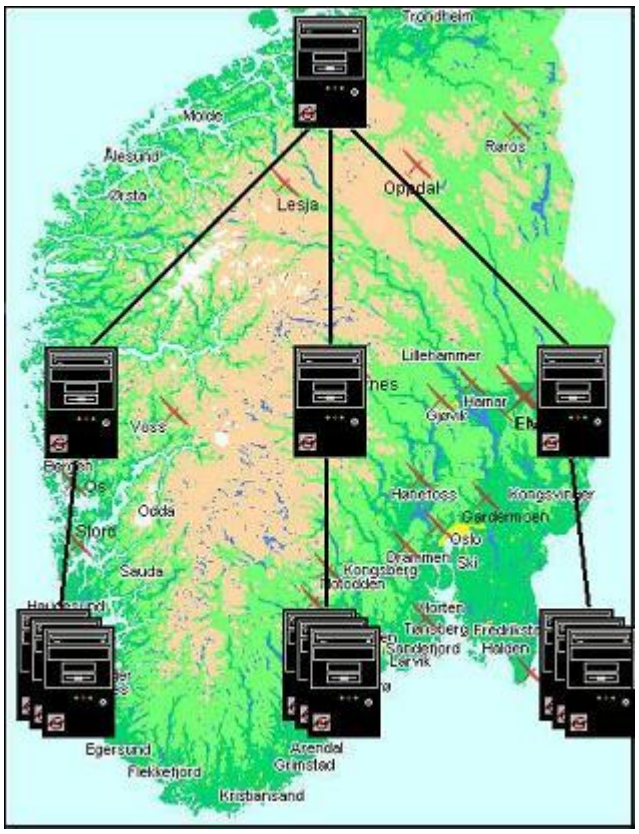
3.2.3 Hensikten med distribuert overvåkning

Distribuert overvåkning er generelt nyttig i små og mellomstore systemer, og da gjerne små og mellomstore systemer som segmenter i større systemer. Informasjonsflyt oppover i et trelignende hierarki er lønnsomt og ressursbesparende, som vist på figur 3.1 .I en Skolelinuxsetting, hvor det ofte er lite penger involvert, og ikke alle i it–seksjonen på barne–/ungdomsskoler er like kompetente vil dette være en stor fordel. Ved å sentralisere driften av et Skolelinuxsystem, og da gjerne slå sammen flere skoler under ett vil man utnytte sentral kompetanse i for eksempel kommune eller fylkeskommune. Det er ikke fast bestemt at det er slik distribusjon skal foregå, i praksis kan hvem som helst sitte som sentral med et visst antall skoler under seg, men ofte er det en dedikert it–seksjon i kommunen som sørger for administrasjon av skoler.

3.2.4 Valg av scriptspråk for Nagios

Gruppen har valgt å scripte alt i Perl[5]. Perl er et veldig fleksibelt scriptespråk, og er særdeles godt til modifisering av tekst og filer. Perl er standard i alle linuxdistribusjoner, og da også i Skolelinux. Det er ikke vanskelig å sette seg inn i, og gruppemedlemmene var delvis kjent med det fra før. Perl er et høytnivå språk som er stabilt og hurtig. Fleksibiliteten kommer ikke bare fra språket selv, men også i stor grad fra CPAN [1]. Her finner man nesten *6000* tredjeparts moduler for å utføre alle mulige oppgaver. nagios–cfg var fra før skrevet i Perl, så videre utvikling i Perl falt naturlig. Selv om plugins til Nagios er mulig å skrive i de fleste språk som kan generere en eksekverbar fil, mente gruppen det var best å holde seg til ett for ordens skyld.

[1] Comprehensive Perl Archive Network



Figur 3.1: Servere som overvåker flere servere

3.3 Munin

Hensikten med en trendanalyser[2] i et slikt system er å oppdage feil før de oppstår som kritiske systemfeil som i verste fall ville tatt ned tjenesten og maskinen. I samarbeid med Nagios kan det utvikles en god varsling og problemet blir løst før det i det hele tatt blir alvorlig.

3.4 Design av applikasjonen

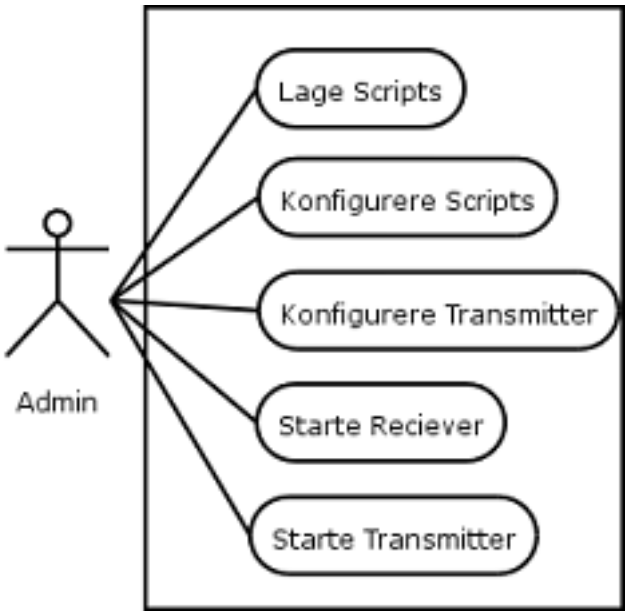
Etter en del samtaler med oppdragsgiver og tenkning, ble det klart at oppgaven ville omhandle tre temaer. Det føltes da logisk å dele opp applikasjonen i tre deler, samle, overføre og behandle data. Dette førte da til en klient til samling av data, en server til å motta data og en klient til å behandle data. Vi valgte å kalle disse

[2]Trendanalyser: Algoritme som detekterer eventuelle kommende feil så de kan rettes før de oppstår.

hendholdsvis Transmitter, Receiver og Listener. Alternativt kunne dette gjøres ved å kode det i færre deler, men dette ville gjort løsningen mindre brukervennlig og skalerbar. Et siste tenkt alternativ, ville vært å skrive om Nagios, men det blir for krevende for en oppgave av en slik art.

3.4.1 Bruk av scripts

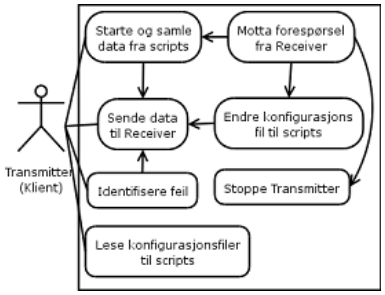
Ettersom arbeidsgiver ville at applikasjonen skulle gi brukere mulighet til selv å bestemme hva som skulle overvåkes og hvilke handlinger som skulle kunne utføres, valgte vi å basere dette på scripts. Dette fordi man da får muligheten til å lage forskjellige kilder som genererer data, uavhengig av eventuelle andre kilder. På denne måten er alt som trengs å vite, hvordan man starter scriptet. Videre skulle hvert script kunne konfigureres for å bestemme hvor ofte det skulle kjøres, hvilke linjer med data som skulle sendes og om det bare skulle sendes data dersom en endring ble oppdaget. Vi la derfor opp til at etthvert script skulle ha en egen konfigurasjonsfil. For å vite hvilke script som skulle startes og hvor de lå, gjorde vi det sånn at Transmitter (klienten som samler data) fikk en egen konfigurasjonsfil der dette skulle stå. Eventuelle handlinger som ville være aktuellt å utføre basert på data, for eksempel fjerning av en bruker eller stopping av en prosess, ble også basert på scripts, som vi for oversiktens skyld kaller CommandScripts. Vi kunne da lage en oversikt (figur 3.2) over hva en administrator må gjøre for å bruke applikasjonen.



Figur 3.2: Use case: Receiver Transmitter

3.4.2 Transmitter

Transmitter skulle, som vist på figur 3.3, samle data generert av scripts, og sende dette videre til Receiver, basert på oppsettet i hvert scripts konfigurasjonsfil. Transmitter skulle ha muligheten til å motta forespørsler fra Receiver om å endre et scripts konfigurasjonsfil og å starte et script uavhenging om det er satt til å bare starte på visse tider. Ettersom en bruker kan skrive feil i konfigurasjonfiler, skulle den også identifisere feil og sende det videre til Receiver. Receiver skulle også ha muligheten til å stoppe Transmitter.



Figur 3.3: Use case: Transmitter

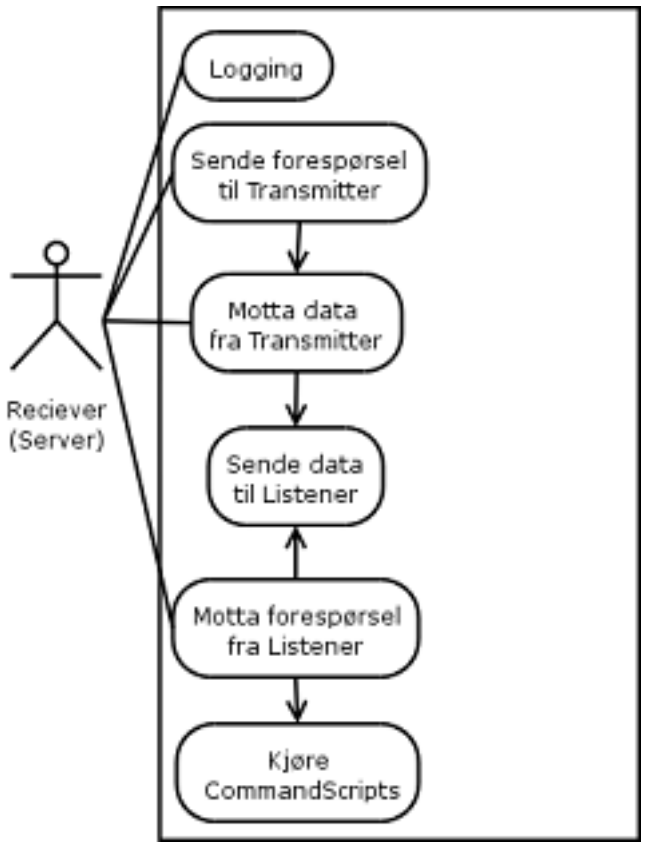
3.4.3 Receiver

Receiver sin oppgave skulle være å samle data fra èn eller flere Transmittere og logge interessante data. Eksempler på slik type data kan være for eksempel feilmeldinger og konfigurasjonen til scripts som illustrers under i figur 3.4. Ettersom arbeidsgiver ville ha muligheten til å overvåke fra for eksempel en web-side eller en GUI–applikasjon, skulle vi gjøre så Receiver kunne arbeide uavhenging av hva slags type applikasjon som var koblet til. Vi valgte Listener som et samlingsbegrep på slike applikasjoner, ettersom til skulle koble seg på Receiver og “Lytte” til data som ble generert av Transmittere. Listenere trengte også muligheten til å sende forespørsler til Receiver om å motta forskjellige typer data ved behov, for ikke å sende unødvendig data over nettet til enhver tid. CommandScripts skulle også kunne startes ved forespørsel fra en Listener.

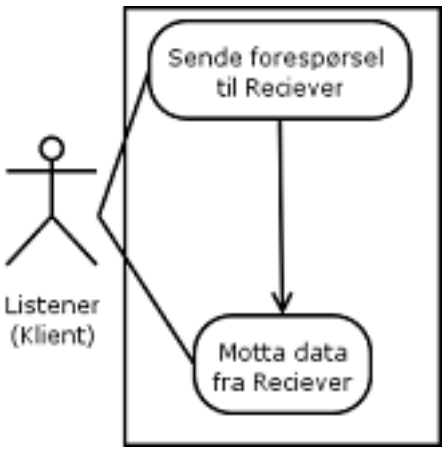
3.4.4 Listenere

En Listeners oppgave, som vist i figurt 3.5, skulle i utgangspunktet være veldig enkel, og kompleksiteten skulle ligge i hvordan Listener selv var oppbygd.

For videre løsning av oppgaven, skulle vi utvikle vår egen Listener som skulle være et grafisk–brukergrensesnitt (GUI) til å starte CommandScripts og vise



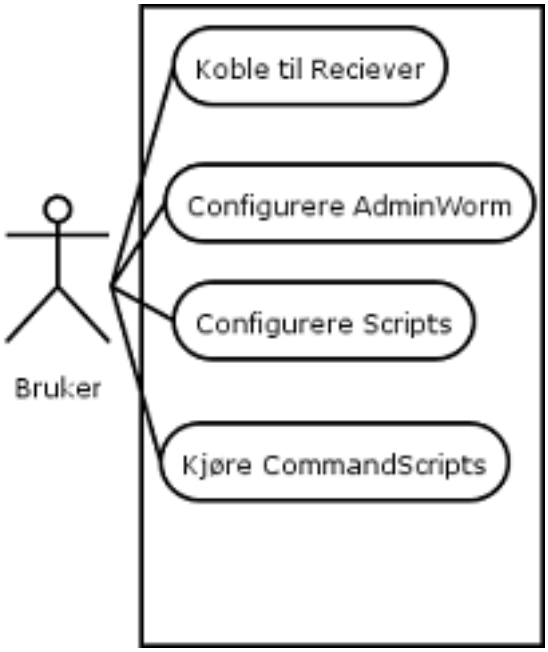
Figur 3.4: Use case: Receiver



Figur 3.5: Use case: Listenere

data generert av Transmitter på en oversiktlig måte. En bruker, som vist i figurt

3.6 ,skulle også kunne definere hvordan data fra et script skulle vises, og hvilke CommandScripts som skulle være tilgjengelig for å bruke basert på dataene. Dette skulle løses ved spesifisering i en konfigurasjonsfil.



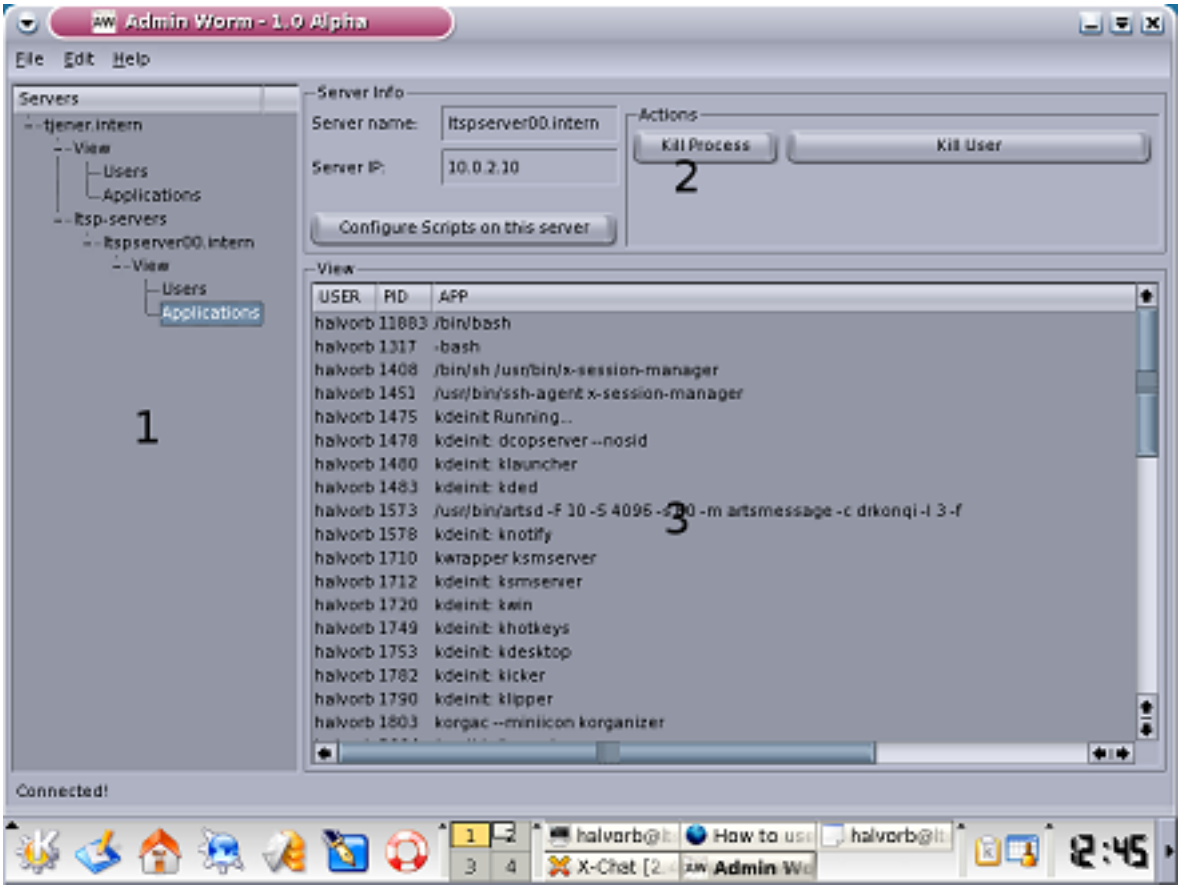
Figur 3.6: Use case: En bruker av AdminWorm

3.4.5 AdminWorm(Listener)

GUI–delen av Admin Worm for å være så enkel å bruke som mulig. Dette med tanke på at ikke alle innen Skolelinuxmiljøet er avanserte pc brukere. Dermed har gruppen lagt vekt på å utvikle GUI som er lett forståelig, med minst mulig uforståelige parametere som må fylles inn. GUI er dermed indelt i 3 hoveddeler, som vist på figur 3.7. Alle disse inngår i hovedskjermbildet. Indelingen er som følger.

- Til venstre finner man en liste over alle servere og de view de enkelte servere har.
- Til høyre finnes informasjon over den serveren man har valgt for øyeblikket, her finner man også knapper for å utføre oppgaver.
- Under denne finner man view. Denne gir en grei oversikt over output fra de ulike scripts som inngår i det valgte viewet. Kombinasjonen av hvordan

data fremstilles og hvilke opprasjoner som brukeren kan foreta på den har vi kalt et “View”. I “AdminWorm” har vi to typer scripts, som vi kaller *output scripts* og *commandscripts* “AdminWorm” har også en enkel konfigurasjons dialog, hvor man kan velge en ip/addresse til serveren man vil koble seg på, og om man vil koble på under oppstart.



Figur 3.7: Screenshot av AdminWorm

Kapittel 4

Implementering

4.1 Eksisterende, og ny, Nagios–løsning

Oppsettet for Skolelinux Woody var allerede pakket og klart, og hoveddelen er utviklet av Ragnar Wisløff fra Linuxlabs[4]. Det var denne pakken det ble tatt utgangspunkt i. Det første på agendaen var å teste om denne pakken lot seg installere i et Sarge miljø uten å gjøre noe som helst med den. Det gjorde den ikke på grunn av forskjeller fra Woody til Sarge. Løsningen ble derfor å bygge opp hele pakken fra bunnen av for å øke sjansen til å få et stabilt produkt. Dette ble naturligvis gjort på en hovedtjener som kjørte Skolelinux Sarge for å sikre kompatibilitet. Ved å granske oppbygningen til den eksisterende, gamle, pakken for Skolelinux Woody, fikk vi et godt utgangspunkt for den nye pakken. Gruppen valgte å beholde eksisterende kataloghierarki, men endre på konfigurasjonsoppsettet. Dette for å i størst mulig grad opprettholde gjenbruk av eksisterende kode/scripts, og gjøre det enklest mulig for de som allerede er kjent med det gamle oppsettet fra Skolelinux Woody. Kommer tilbake til endringene i seksjonen 4.2. Katalogstrukturen i pakken ser slik ut:

```
/
/etc/
/etc/Nagios/
/etc/Nagios/hosts/
/etc/Nagios/hostgroups/
/etc/Nagios/templates/
/etc/Nagios/default/
/usr/
/usr/sbin/
/usr/share/
/usr/share/Nagios/
```

```
/usr/share/Nagios/htdocs/
/usr/share/Nagios/htdocs/images/
/usr/share/doc/
/usr/share/doc/Nagios-cfg-debian-edu/
/var/
/var/log/
/var/log/Nagios/
/var/log/Nagios/rw
```

Alle katalogene innenfor Nagios er nye i forhold til en ren Nagiosinstallasjon. Det vil si at disse katalogene blir generert ved installasjon av pakken. Alle hoster som skal overvåkes får sin egen konfigurasjonsfil i mappen `hosts` og er definert i hostgroups som host–type server, gateway, workstation, printer eller infrastructure. Dette er gjort for å ha det så oversiktlig som mulig. Ved å la hver host ha sin egen fil med sine egne definerte tjenester er det enklere å holde styr på hver host enn om alle hoster skulle vært samlet i en stor fil. Det er også mulig å samle hoster i grupper hvis det må skaleres opp, gruppen har valgt å ikke implementere dette siden denne løsningen skalerer godt nok i Skolelinuxsammenheng. I vårt tilfelle dreier det seg hovedsaklig om å overvåke ulike servertjenester. Det er også fullt mulig å overvåke klienter, men de defineres ikke som servere, og med veldig få tjenester å overvåke. Hva som skal overvåkes avhenger av server–/klienttype, og defineres via nagios–cfg scriptet som setter opp hele Nagios–strukturen på nytt. Mer om dette kommer senere under seksjonen 4.2.7. I mappen templates ligger alle grunnfilene for de ulike host–typene. Ved kjøring av nagios–cfg vil scriptet gjenkjenne hoster basert på ip–adressen til hosten. Dette lar seg gjøre siden Skolelinux har et ip–oppsett basert på faste adresser og adressegrupper for ulike grupper hoster. Filer vil deretter kopieres over i mappen hosts, fra templates, og settes opp i henhold til host–type. Mappen default inneholder alle standard konfigurasjonsfiler. Kjekt å ha i tilfelle man har gjort en feil i konfigurasjonen av selve Nagios og måten programmet jobber på. Mappen `/usr/share/doc/nagios-cfg-debian-edu` inneholder dokumentasjon som følger med pakken, og mappen `/var/log/Nagios/rw` inneholder en “external command file”[1] som skal brukes ved distribuert overvåkning. I vårt tilfelle vil distribuert overvåkning fungerer slik at enhver Nagios som skal distribuere må ha “nsca” innlagt og konfigurert for å sende mot en bestemt sentral server. I pakken vår vil dette være lett å sette opp bare ved å editere en konfigurasjonsfil. Brukeren står selv fritt til å velge om informasjonen som sendes ut skal krypteres eller ikke, default er en metode som benytter seg av enkel XORA.1 .

[1] Denne filen inneholder alle eksterne kommandoer som skal kjøres. Eksempler er konfigurasjonsendringer via webgrensesnittet, sjekk av tjenester over distribuert overvåkning, og aktivering/deaktivering av tjenesten

Dette er en dårlig krypteringsmetode, og bør endres til et annet av valgene i konfigurasjonen etter eget ønske. Det er mulig å velge bl.a DES, 3DES, BLOWFISH, RC2 m.m. “nsca” er en daemon skrevet kun for å kunne sende informasjon i et distribuert overvåkningsnett via passive tjenestesjekker.

4.2 Endringer i Nagios og nagios–cfg

4.2.1 Den gamle Nagios pakken

Først måtte gruppen få den gamle pakken til å fungere for å kunne kartlegge hvilke endringer som måtte til for å få dette til å kjøre stabilt i et Skolelinux Sarge miljø. Siden det allerede var forsøkt å installere den gamle pakken var det bare å gå inn i oppsettet å kikke. Det første man ser er at pakken har forsøkt å overskrive alle konfigurasjonsfilene Nagios kom med. Dette er for det første ikke bra, en pakke skal ikke overskrive andre pakkers filer ved installasjon [17]. For det andre anses dette som en lite stabil løsning. Foreløpig merker vi oss dette og lar det gå. Alle de nye filene renames for å overskrive de gamle med kommandoen `rename -f 's/.dpkg-*\$//' *.dpkg*`. Mappen `rw` under `/var/log/Nagios` måtte opprettes i henhold til Nagios.cfg for “external command” filen Nagios.cmd. Endringen ble kartlagt og testingen gikk videre. Ved førstegangs kjøring av nagios–cfg kom det en feil, den reagerte på at NetAddr::IP ikke var inne, og nødvendig ip–håndtering fungerte da heller ikke. Pakken libnetaddr–ip–Perl ble installert for å inkludere dette. Etter andregangs kjøring feiler den på en av funksjonene for å legge til nye hoster. Etter litt “googling” og en samtale med vår veileder kom det fram at syntaxen er forandret fra “my $useip = new NetAddr::IP–>new($ip)” til “my $useip = new NetAddr::IP $ip”. Etter dette settes “$debug” til 1 for å håndtere eventuelle feilmeldinger ved neste kjøring. Neste kjøring viste seg å bli den siste, den fullførte uten feil og la til alle hostene den fant som egne filer i mappen hosts, hvordan dette fungerer kommer i et senere avsnitt(seksjon 4.2.7)

4.2.2 Et nytt web–interface

Neste punkt var å tilrettelegge web–interfacet for et Skolelinux–look. Det kom et eget oppsett med stylesheets og bilder med den gamle pakken, men det viste seg at dette ikke fungerte, i likhet med det meste andre fra den gamle pakken uten å endre på oppsettet. Ved første test manglet det knapper og bilder, og riktig stylesheet ble ikke tatt i bruk. Første steg var å kopiere over de rette html–filene for forside og meny fra `/etc/Nagios` til `/usr/share/Nagios/htdocs` og deretter å redigere `/etc/Nagios/apache.conf` slik at den inneholder linjen

`Alias/Nagios/stylesheets /etc/Nagios/stylesheets`. Etter dette måtte webserveren apache restartes, og webgrensesnittet så bra ut. Dette oppsettet er primært tilrettelagt apache, men kan også endres til andre webservere, noe dette prosjektet ikke omhandler. Det lokale oppsettet ser nå bra ut, og neste steg er den distribuerte overvåkningen.

4.2.3 nsca

Å installere nsca går kjapt og greit via apt–get[2], og gruppen har valgt å bruke standardoppsettet med tidligere nevnte krypteringsmetode (XOR 4.1). Data som per nå overføres ved nsca er ikke sensitive, så denne krypteringsmetoden må endres etter behov. Eneste endringen som er nødvendig i filen nsca.conf er å sette path til `nagios.cmd` til `/var/log/nagios/rw/nagios.cmd` lokalt på sentral server, forutsatt at sentralen kjører Nagios med et Skolelinux–oppsett. Hvis ikke må path endres til rett path for den aktuelle sentralen. Scriptet "submit_check_result_via_nsca" tar for seg selve distribusjonen. Dette scriptet følger med Nagios–pakken, og må bare modifiseres med rette paths og hostname/ip–adresse til sentral server, eks:

```
printfcmd="/usr/bin/printf"
NscaBin="/usr/sbin/send_nsca"
NscaCfg="/etc/send_nsca.cfg"
NagiosHost="xxx.xxx.xxx.xxx"
```

Det eneste scriptet gjør er å kjøre en kommando basert på dette og sender da informasjonen opp til NagiosHost etter hver kjøring av lokale sjekker. Deretter måtte filen "misccommands.cfg" under `/etc/Nagios` redigeres til å inneholde definisjonen av denne kommandoen. Parameterene som blir brukt er forklart i tabell 4.1

```
define command
   command_name submit_check_result_via_nsca
   command_line $PATH$/submit_check_result_via_nsca
   $HOSTNAME$ '$SERVICEDESC$' $SERVICESTATE$ '$OUTPUT$'
```

4.2.4 Aktivering av distribuert overvåkning

For å aktivere distribuert overvåkning må filen `/etc/Nagios/Nagios.cfg` editeres ved å sette "ocsp_command = submit_check_result_via_nsca" (forutsatt

[2]Program for å laste ned og installere pakker fra Debian sitt pakkearkiv

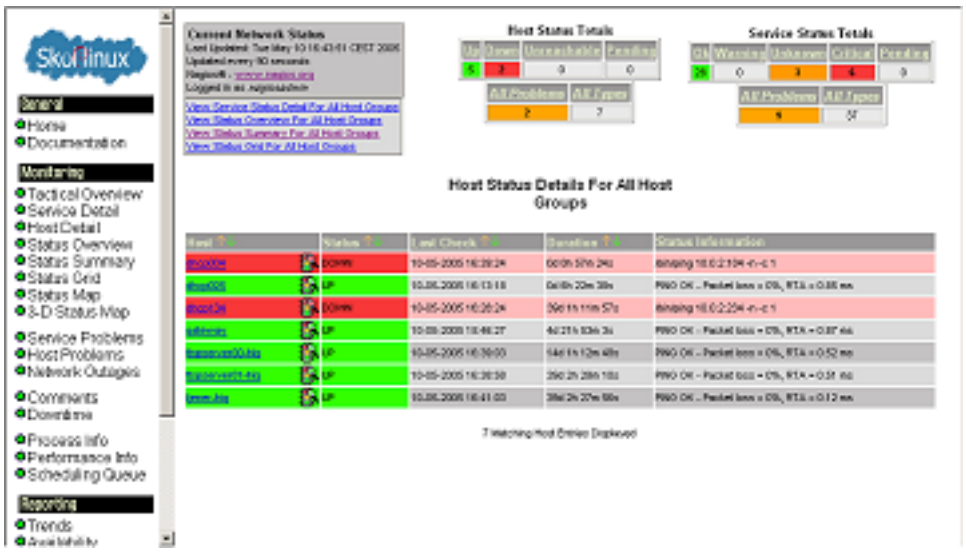
$HOSTNAME$	Navn på hosten som er sjekket
$SERVICEDESC$	Beskrivelse av tjenesten som er sjekket på host
$SERVICESTATE$	Tilstand på tjenesten, 0=OK,1=WARNING,2=CRitICAL,3=UNKNOWN
$OUTPUT$	Tekstoutput fra plugin, én linje fra STDOUT

Tabell 4.1: Beskrivelse av parametere

at det er dette den kalles i misccommands.cfg), og opsjonen ”obsess_over_services settes“ til 1 for å aktivere kjøring av ocsp_command. For å få Nagios til å kjøre “ocsp_command” da den skal er det ikke alltid nok å sette “obsess_over_services” til 1 i nagios–cfg. For å være sikker på at Nagios har “forstått” må dette sjekkes i “Process Info” i web–interfacet. Der skal “Obsessing Over Services?” stå til “Yes”, hvis ikke slås den på via menyen til høyre. Opsjonen enable_notifications settes til 0 eller 1 avhengig av om det skal varsles til noen om en tjeneste eller host lokalt er nede. Siden dette skal overvåkes sentralt kan man sette “enable_notifications” til 1 på sentral for å varsle alt derfra. Dette er lurt hvis det er ønskelig å spare lokal administrator for unødvendig mail/sms da den sentrale administratoren allikevel skal gjøre jobben. Hvis det er ønskelig å ha et rent distribuert system må “execute_service_checks‘” settes til 0 på den sentrale hosten for at det ikke skal kjøres noen form for sjekk av lokale tjenester på sentral. Gruppen har foreløpig valgt å ha en løsning basert på både lokale og passive sjekker på sentral. Siden sjekking av tjenester baserer seg på både ip–adresse og hostnavn lokalt måtte dette endres med tanke på både den lokale og den distribuerte overvåkningen. Ved å gå inn i checkcommands.cfg og endre alle `$HOSTNAME$` til `$HOSTADDRESS$` fikses en bug som kommer i sammenheng med navnebytte av lokale hoster. De navnene som endres ved hjelp av “name-cfg” finnes ikke via dns. Derfor må alle lokale sjekker basere seg på ip–adressen. Det retter da en feil man ville fått via distribuert overvåkning. Distribuert overvåkning baserer seg kun på hostnavnet lokalt, så ved å rette på sjekker lokalt ungår man en følgefeil oppover i treet. Utover dette er det ingen endringer som trengs i Nagios–oppsettet. Figuren under, figur 4.1, illustrerer det nåværende utseende av Nagios. I dette tilfellet detaljer for hostene på nettverket på hovedprosjektrommet.

Et viktig punkt er at alle tjenester som sjekkes rundt omkring på distribuerte hoster *må* være definert på sentral server, eks:

```
define host
   use                    server-host
   host_name              gateway-hig
   alias                  gateway ved testlab HiG
   address                10.0.2.1
```



Figur 4.1: Screenshot av Nagios

```
define service
   use                     p-service
   host_name               gateway-hig
   service_description     PING
   check_command           check_ping!100.0,20%!500.0,60%
```

Her er p–service en modifisert utgave av server–service for passive hoster. Eneste forskjellen er at active_service_checks satt til 0 slik at hostene kun skal sjekkes passivt via distribuert overvåkning. p–service som et barn av generic–serviceer er definert i filen `/etc/Nagios/servicetemplates.cfg`. Generic–service er en mal for ulike typer servicedefinisjoner. Alle de andre parametrene er standard, og opp til lokal administrator å velge. Beskrivelse av alle parametre finnes i dokumentasjonen til Nagios. Ved å bruke p–service vil alle hoster definert med denne typen service dukke opp i webgrensesnittet med en “P” ved siden av seg for å vise at denne hosten ikke sjekkes aktivt lokalt, men distribueres passivt. Status for slike hoster leses ut ifra den eksterne kommandofilen, Nagios.cmd, som nsca–programmet skriver til da den får service–sjekker fra distribuerte servere. Eks. på servicedefinisjon:

```
define service
   name                                p-service
   use                                 generic-service
```

```
max_check_attempts              3
normal_check_interval           5
retry_check_interval            5
contact_groups                  Nagios-admins
check_period                    24x7
notification_interval           30
notification_period             24x7
notification_options            u,w,c,r
active_checks_enabled           0
register                        0
```

4.2.5 Nye scripts i Nagios pakken

En annen ting som var ønskelig å løse var navngivning av hoster. For å overvåke hoster sentralt må det være mulig å skille ulike skolelinuxinstallasjoner fra hverandre. Siden alle skolelinuxinstallasjoner i prinsippet er satt opp med like ip–adresser og hostnavn var det ønskelig å lage en enkel måte å endre dette på både sentralt og lokalt. Dette ble gjort ved å lage to Perlscripts “name–cfg”(vedlegg E.1) og “add–passive”(vedlegg E.2). “name–cfg” benyttes etter man har kjørt “nagios–cfg”, for å endre hostnavn lokalt. Den endrer bare hoster av typen “ltspserver” og “tjener”, eks:

```
ltspserver00 --> ltspserver00-hig
ltspserver01 --> ltspserver01-kapp
tjener --> tjener-hio
```

Scriptet går inn i konfigurasjonsfilen til den aktuelle host og endrer navnet på host, på alle tjenester, og endrer til slutt navnet på fila. Se vedlegg C “add-passive” er et script som skal kjøres på den sentrale serveren for å enkelt kunne legge til hoster som skal overvåkes sentralt. Den oppretter da konfigurasjonsfiler på sentralen som samsvarer med type host og hostnavn fra den distribuerte serveren. Det er viktig at navnet man skriver inn her er akkurat det samme som på navnet på hosten lokalt siden distribusjonen baserer seg på hostnavnet lokalt fra de distribuerte serverene.

4.2.6 Pakking av ny nagios–cfg

Første steg var å opprette et eget katalogtre i en mappe hvor det eksisterende oppsettet og alle mappene beskrevet under seksjon 4.1 ble kopiert, som videre skulle brukes til utviklingen. Dette ble gjort for å starte med blanke ark. Konfigurasjonsfilene ble kopiert over, og som tidligere nevnt bestemte gruppen seg for å beholde

den eksisterende debian–edu konfigurasjonen for å gjøre det mest mulig uniformt med resten av Skolelinux. Debian–edu konfigurasjonen er Skolelinux sin egen struktur for konfigurasjonsfiler og oppsett. Etter utpakking og oppretting av kataloger ble alle konfigurasjonsfiler og script for Nagios kopiert over i sine respektive mapper. Siden oppsettet katalogmessig er likt trengs ingen endringer i konfigurasjonsfilene, men disse er gått igjennom for å forsikre brukerne av pakken om at alt er OK. Alle endringene som har blitt gjort, og alle filer som måtte kopieres manuelt og endres på etter forsøk på å installere den gamle pakken har blitt kopiert over i det nye treet pakken skal lages fra. Denne pakken er bygd slik at den ikke vil overskrive noen av de eksisterende filene ved installasjon/utpakking. Etter strukturen er lagd må kontrollfilen som lagrer informasjon om pakken opprettes og redigeres. Filen heter "control" og ligger i mappen DEBIAN i roten av pakketreet `/usr/src/nagios-cfg-debian-edu/DEBIAN`. Her skal all informasjon om pakken, pakkens vedlikeholder og skaper, versjonsnummer, avhengigheter etc. ligge. Denne katalogen vil ikke bli synlig etter at pakken er bygd.

Selve pakkingen foregår slik

- Sourcekatalogen ligger i
  `/usr/src som /usr/src/Nagios-cfg-debian-edu`
  Stå i `/usr/src` og skriv
  `dpkg-deb --build Nagios-cfg-debian-edu tmp.deb`

- Test pakken, og endre navn på tmp.deb til hva du vil pakka skal hete, man kan selvfølgelig gjøre dette med en gang i selve byggingen, men om den er pakket fra før så bør ikke den gamle pakken overskrives uten at den nye testes på forhånd.

Siden Skolelinux er på vei til å gå over i "debian–edu", og da bli en del av debian sentralt, er det ikke lett å få egne pakker inn i distribusjonen. Denne pakken skal etterhvert inn i Skolelinux Sarge, men videre derfra har ikke gruppen selv mulighet til å få pakken inn i løpet av hovedprosjektperioden. Hvis noen i miljøet tar på seg å være vedlikeholder av pakken, og å "renske" den for eventuelle bugs så vil pakken permanent havne hos "debian–edu" hvis det er behov for den.

4.2.7 Funksjonalitet

nagios–cfg, som er kjernen av oppsettet, fungerer slik at den pinger alle ip–adresser som skolelinux bruker på nettverket, og legger de som svarer inn i systemet på bakgrunn av ip–adressen de har, ip–adressene fordeles slik:

```
Tjener : 10.0.2.2
LTSP   : 10.0.2.10-10.0.2.29
Printer: 10.0.2.30-10.0.2.49
Static : 10.0.2.50-10.0.2.99
WS     : 10.0.2.100-10.0.3.243
```

- Tjener[2] er hovedtjeneren i nettverket. Denne håndterer fil–/nettverksdeling, inneholder brukerens hjemmeområder og hele oppsettet for nettverket. Systemtjenester og godkjenning av brukere ligger her.
- LTSP er tynnklienttjenerne. Disse kjører alle prosessene til tilhørende tynnklienter. Det er mulig å ha flere tynnklienttjenere i samme nettverk.
- Printer er skrivere som kan kobles til arbeidsstasjoner, tynnklienter eller nettverket. De administreres sentralt fra Tjener.
- Static er et område satt av til maskiner/enheter som trenger statisk ip–adresse, mye forskjellig utstyr kan ha behov for dette.
- WS er arbeidsstasjoner som kjører alle prosesser selv. All autorisering av brukere og lagring av data håndteres av Tjener.

Alle hoster som svarer på “ping” dyttes inn i en array basert på ip–adresse og hostname. Deretter trekkes en og en host ut av arrayen, og ip–adressen sjekkes opp mot host–type (forklart i seksjon 4.2.7). Riktig host–type returneres og konfigurasjonsfilene opprettes i henhold til host–type. Disse konfigurasjonsfilene opprettes fra filene i katalogen `/etc/Nagios/templates`, som tidligere nevnt inneholder alle grunnfilene for host–typene. I tillegg til å opprette filer for hoster vil også konfigurasjonsfilene for Nagios overskrives. Det er derfor viktig at riktige kopier av disse filene eksisterer i mappen `/etc/Nagios/default` så ingen konfigurasjonsendringer forsvinner i prosessen. Hvis det derimot er ønskelig å legge til en og en host vil ikke hele systemet overskrives. Til slutt legges hostene til på riktig sted i `/etc/Nagios/hostgroups`. En host kan legges til som server, gateway, workstation, printer eller infrastructure, og disse host–typene har alle ulike tjenester som skal overvåkes. Foreløpig er det ikke gjort noen endringer i forhold til Skolelinux Woody på dette, men dette kan gjøres lett ved å redigere templates for de ulike hostene, eventuelt legge til tjenester individuelt i hver host sin konfigurasjonsfil. Dokumentasjonen til Nagios har en god forklaring på dette, evt. så er konfigurasjonsfilene veldig lette å forstå.

4.3 Munin med Nagios, trendanalyzer

4.3.1 Kort om Munin

Munin[30] er en applikasjon utviklet av linpro[3] som samler systemdata og lagrer dette unna i RRD–databaser[21]. Disse dataene blir dratt ut og displayet i en samling grafer som kan vise det meste av systemtjenester på en host.

4.3.2 Munin mot Nagios

Til Nagios finnes det en plugin som heter Munin–Nagios. Denne sender passive advarsler til Nagios hvis verdier overstiger en grense basert på data fra Munin. Denne baserer seg også på "nsca" som tidligere er omtalt i rapporten, og er en slags lokal distribuert overvåkning.

4.3.3 Trendanalyzer

Gruppen fikk i oppgave å utvikle et rammeverk for å lage noe lignende, en trendanalyzer. Denne skal skrives som en Nagios–plugin, og hente ut data fra Munin sine RRD–databaser. En slik analyzer er foreløpig tenkt å lages for å måle CPU–load og diskbruk over tid. Denne har vi ikke utviklet ferdig pga at idéen til oppgaven kom for sent i prosjektet. Den ble da sett på som en oppgave gruppen kunne se på om tiden strakk til.

Foreløpige trendanalyzer

Foreløpig er analyzeren tenkt slik for CPU-load:

- Først skal trendanalyzeren samle inn informasjon for hver dag mellom 0800 og 1700.
- Deretter hentes det ut, og det regnes snitt av de fire mest belastede timene iløpet av denne perioden. Målingene tas fra cpu_idle_load målinger. Dagsverdiene samles inn over en hel uke, og til slutt regnes det snitt av denne uken.
- Den første uken settes til middelverdien de andre skal måle seg mot.
- Det er videre tenkt at over tid skal man kunne se på slike ukeverdier og se om de har en trend, enten opp eller ned i forhold til den første uken, og et antall uker som har vært.

For vår modell er det tenkt at verdiene skal dyttes tilbake i en RRD–database og hentes ut ved behov, så ikke scriptene behøver å parse alt av informasjon hver gang, for det vil ligge en god del dager der etterhvert, men foreløpig baserer det seg på tekstdatabaser. Løsningen har nå to scripts. Det ene kjøres i en cron–job og samler data fra RRD–databasen til Munin for cpu_idle_load og lagrer data fra 0800 til 1700 hver dag i en fil med snittverdier tatt hvert femte minutt over den aktuelle timen. Det er disse dataene som er tenkt å lagres i en RRD–database om løsningen skal implementeres skikkelig. Det andre scriptet benytter seg av disse dataene og beregner firetimers–snitt. Foreløpig lagres ikke disse snittene noen sted, men en tenkt løsning kan være å også ha disse i en RRD–database. En administrator skal selv kunne bestemme terskelverdier og deltaverdier mellom ulike målinger, samt antall uker en måling skal sjekkes mot, på en enkel måte i scriptene. Output fra scriptet vil være en tekststreng som beskriver trenden i forhold til de satte terskel– og deltaverdier, og en returncode, 0/1/2/3. Dette er verdier som Nagios trenger for å kunne gi eventuell alarm og en forståelig status ved hjelp av en tekststreng.

Eksisterende løsninger

Det finnes lignende løsninger allerede for eksempel *cfenvd*. Dette følger med cfengine, som på samme måte samler inn data og kan se en trend. [7] Munin har også en lignende funksjon.

4.4 Valg av pakker til AdminWorm

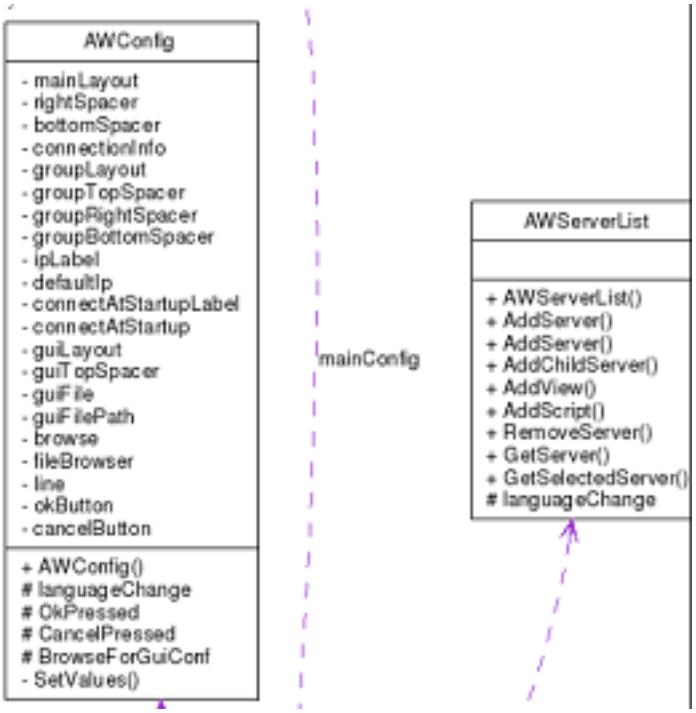
Hele GUI og dens funksjonalitet er utviklet ved bruk av Qt fra Trolltech [31]. Qt er et kryss–kompatibelt GUI bibliotek for C++. Dette gjør at “AdminWorm” kan kompilere og kjøre på både Linux, BSD, Windows og Mac OS. Selv om “AdminWorm” primært skal kjøre, og bare vil ha funksjonalitet, på linux (Skolelinux). I tillegg er libxml2 tatt i bruk for lagring av konfigurasjon som språkvalg, serveradresse etc. under kjøring av applikasjon. Denne løsningen ble valgt fordi xml er et lett, og standard, format

4.5 Kommentering av kode i AdminWorm

Kommentering av C++ koden, som blir benyttet i GUI-delen av AdminWorm, beskrevet i et eget dokument som ligger på en CD som følger med rapporten. Dokumentet er laget ved hjelp av “Doxygen”, som har produsert en LaTeX 2ε kode, som har blitt kompilert og lagt til denne rapporten. Ved hjelp av Doxygen blir det

generert fine klassediagrammer som vist under i figuer 4.2 som viser dialog med script og dialog ved valg av server, og i figur 4.3 som viser selve konfigurasjons-dialogen til AdminWorm og dialogen som viser lister over servere(del 1 som det ble henvist til i seksjon 3.4.5).

Figur 4.2: Klassediagram: Server og valg av script



Figur 4.3: Klassediagram: Server list og konfig.dialog

4.6 Konfigurasjonsfilene til AdminWorm

Alle konfigurasjonfilene er på formen[3]:

```
[overskrift]
<nøkkel>=<verdi>
```

for eksempel:

```
[Path]
RUNTIMER=10
```

Det er da en enkel oppgave for koden å lese inn en slik fil, og bruke ønskelige nøkler og verdier. Og ettersom vi holder en standard er det mye lettere å jobbe med forskjellige typer konfigurasjonsfiler.

4.7 Script som verktøy i AdminWorm

Applikasjonen benytter seg av ulike uavhengige scripts kodet i Perl. Dette er fordi mange har kjennskap til å lage slike, og dermed med letthet kan legge til nye

[3]Syntaksen er på på samme form som .inf i Windows og DOS

moduler. Det gjør også at ting kan bli mer oversiktlig ettersom scripts ikke trenger å vite noe om hverandre. Man kan også forandre på ett, uten at det påvirker alle de andre.

4.8 Sikkerhet i AdminWorm

Ettersom dataoverføring over nettverket er en stor del av oppgaven, kommer spørsmålet om sikkerhet fram. Dette med sikkerhet er ikke noen enkel oppgave, og etter diskusjon med arbeidsgiver, ble gruppen enige om at det skulle legges til rette for å implementere sikkerhetsmessig funksjonalitet til slutt, enten av oss eller andre, og heller konsentrere utviklingen om å få en funksjonell løsning, uten sikkerhet. Det er kodemessing lagt til rette for å endre på hvordan data overføres over nettverket, uten å måtte forandre på resten av koden.

4.8.1 De ukritiske delene av overføringen

Trafikken som går gjennom Transmitter, Receiver og Listener er altså ren tekst, og det sikkerhetsspørsmålet som i all hovedsak stilles, er hvorvidt en utenforstående skal kunne se disse dataene eller ikke. Det var snakk om å kjøre disse dataene gjennom SSL, slik at bare brukere med de aktuelle rettigheten skulle ha tilgang. Fordi det i utgangspunktet ikke er sikkerhetsmessig utsatt data som overføres, ble det gjort et valg om å la dette ligge og fokusere på funksjonalitet fremfor sikkerhet.

4.8.2 De kritiske delene av overføringen

Et annet problem var hvordan en bruker skulle kunne gjøre handlinger på systemet. Det gis ikke direkte tilgang til å gjøre endringer eller kjøre “kommandoer” på systemet gjennom en tilkobling på Transmitter, Receiver eller Listener. Istedet er løsningen på at det sendes en forespørsel til Receiver om å starte et CommandScript, som forklart i seksjon 4.10.6. Det er da opp til hvert enkelt CommandScript å implementere sikkerhetsmessige rutiner. For eksempel ved å “kicke” en bruker trengs det en form for autentisering via SSH. Dette implementeres da i scriptet.

4.9 Forklaring til oppgaven

4.9.1 Valg av scriptspråk i AdminWorm

Gruppemedlemmene hadde tidligere ikke laget noen klient–server applikasjoner, men hadde kjennskap til litt Perl og visste at dette var et språk godt egnet for å be-

handle tekst. Perl har også funksjoner og moduler for å håndtere kommunikasjon over sockets [35].

4.9.2 Hvorfor lage moduler ved hjelp av scripter

Ettersom arbeidsgiver ville ha et generisk system, med mulighet for at brukere enkelt skal kunne legge til egne moduler for hva som skal overvåkes, er løsningen basert på scripts, se seksjon 4.12.1. Transmitter ble deretter utviklet for å starte disse scriptene, og sende dataene de genererte, over til Receiver. Port 4244 viste seg å være ledig, så valget falt på denne, for denne overføringen, uten noen spesiell grunn. Det var også et ønske fra arbeidsgiver at det skulle kunne defineres for hvert script hvor ofte det skal kjøre, hvilke linjer i generert data som skal sendes, om det bare skal sendes dersom en endring oppdages osv. Derfor har hvert script muligheten til å ha en konfigurasjonsfil som leses av Transmitter før scriptet kjøres. Siden forskjellige scripts kan generere forskjellig data og bruker varierende tid for å gjør dette, er alt lagt i tråder A.1. Dette vil sikre at et script kan bruke den tiden det trenger, uten å sperre for andre.

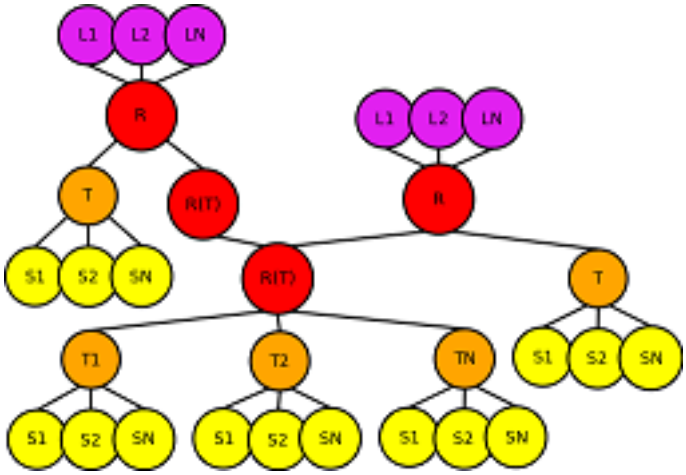
4.10 Kort forklart om Transmittere og Receivere

Serveren består nå av to deler. En sender (klient) som vi kaller “Transmitter” (seksjon 4.12) og en mottaker (server) som vi kaller “Receiver”. Kort fortalt kan én eller flere Transmittere starte én eller flere kjørbare filer for eksempel scripts (se seksjon 4.12.1) skrevet i *Perl*, og sende dataene disse genererer, videre til Receiver. En eller flere applikasjoner (klienter) kan da koble seg opp mot Receiver og motta all data generert av Transmittere. Disse applikasjonene kaller vi for “Listener”. Vi velger å kalle dem det ettersom GUI er forbundet med synlige knapper, vinduer osv, mens en Listener kan være hva som helst (også en GUI) som kan koble seg opp mot en socet. Listener blir nærmere forklart i seksjonen 4.14. Det er altså fritt fram for andre å utvikle en Listener til å koble seg opp mot Receiver og gjøre det som måtte ønskes med data som mottas. Dette ble gjort siden arbeidsgiver ønsket muligheten for å lage løsninger rettet mot web o.l. En Receiver kan også koble seg på en annen Receiver, og på denne måten sende data videre i flere ledd. Viser her et eksempel på hvordan et en slik kobling kan se ut:

4.10.1 Oversiktsbilde for scripts, Transmittere, Receivere og Listenere

Figuren 4.4, viser hvordan en kobling mellom Transmittere, Receivere og Listenere kan se ut.

T = Transmitter
R = Receiver
L = Listener
S = Script
(T) = Receiver oppfører seg som en Transmitter for tilkoblet Receiver. For å utføre handlinger basert på mottatt data har vi laget muligheten til å sende en forespørsel (se seksjon 4.11) til Receiver om å starte scripts som vi kaller CommandScripts (se seksjon 4.13.1).



Figur 4.4: Koblingsoversikt

4.10.2 Konfigurasjonsfilen til Transmitter

Siden det av og til er ønskelig å separere forskjellige typer filer, for oversiktens skyld, eller andre grunner, ga fikk Transmitter en egen konfigurasjonfil “Transmitter.conf”. Der står det i hvilken mappe scripts ligger, hvilke scripts som skal startes og hvilken mappe konfigurasjonsfilene deres ligger og hva de heter.

4.10.3 Receivere og Listenere

For at data skal kunne vises på en oversiktlig og logisk måte, trengs det en eller annen form for GUI. Derfor kan applikasjoner koble seg på Receiver på port 4243, og motta all data sendt av Transmittere gjennom den. Disse applikasjonene kalles Listenere. Gjennom tråder er det mulig for flere Listenere å koble seg på en Receiver. En Listener kan også sende forespørsler til Receiver, om å motta for eksempel en liste over alle feil som eventuelt har oppstått. Så i hovedsak kan det å lage en

Listener mot Receiver, sammenlignes med å lage en GUI for IRC–protokollen. Vi har lagd vår egen Listener som kalles AdminWorm.

4.10.4 Listenere, Receiver og Transmitter sammen

Som en følge av at en Listener mottar all data generert av Transmittere koblet til en Receiver, ble det en lett prosedyre å gjøre det slik at en Receiver faktisk fungerer som en Listener og kan koble seg på en annen Receiver, samtidig som den tillater tilkoblinger fra Transmittere. En Receiver som kobler seg på en annen, vil da se på denne som en Transmitter, som fører til at data kan sendes videre i mange ledd. Dette illustrers i det tidligere figuren 4.4 i seksjon 4.10.

4.10.5 Feilsjekker i AdminWorm

Ettersom det er mulig for brukere å skrive feil, er det implementert mange feil–sjekk prosedyrer. Alle feil som oppstår vil bli sendt til Receiver og logget der. For en mer detaljert beskrivelse av dette, se seksjon 4.15.

4.10.6 CommandScripts

Som tidligere nevnt(seksjon 3.4), ønsket arbeidsgiver ønsket muligheten til å utføre handlinger basert på mottatte data. for eksempel fjerne en bruker, stoppe en prosess, flytte over en fil osv. Dette skulle også være et generisk system, altså gi brukere mulighet til å legge til egne handlinger. Dette løste seg ved å basere det på scripts som vi kaller “CommandScripts” (se seksjon 4.13.1) Receiver fikk derfor muligheten til å starte eksterne scripts, ut i fra en forespørsel fra en Listener.

4.10.7 Nye funksjonaliteter

Ettersom nye funksjonaliteter blir lagt til, er koden designet med tanke på dette. Det kan for eksempel lett legges til forandre hva som er gyldige key’er og verdier i en konfigurasjonsfil og ta disse i bruk. Å forandre på hvordan teksten som sendes mellom Transmitter, Receiver og Listener er også enkelt. Vi har jobbet mye frem og tilbake, og lært nye ting underveis, så det å legge til denne muligheten har vist seg å være høyst nødvendig, og har spart oss mye tid ved endringer og nye implementasjoner.

4.10.8 Utseende og funksjonalitet

De følgende punktnene omhandler de forskjellige GUI–delene av AdminWorm. Fiureren 3.7 viser hvor på skjermen de tre forskjellige delene under, er plassert på

skjermen.

Serverlist

Etter oppstart vil programmet være koblet til, vel og merke hvis brukeren har valgt det alternativet, ellers må brukeren selv koble til. Dette kan gjøres enkelt ved og trykke “F5” tasten. Brukeren har også mulighet til å koble fra med “F6” tasten (for overskikt over shortcut keys, se brukerveiledning for “AdminWorm”). Uansett om brukeren valgte å koble til under oppstart eller å gjøre dette manuelt, vil brukeren få opp en liste over servere og de “view” som hver server har i serverlisten. Denne listen vil oppdatere seg dynamisk, for eksempel hvis en annen server i nettverket kobler seg på, vil denne komme opp her automatisk. Hvis en server detter ned, vil denne forsvinne fra listen. Dette er oppnådd ved at serveren (hovedserveren som GUI er koblet på, noe som gruppen har kalt en “receiver”) vil rapportere om endringer i serverstatus. All informasjon som mottas fra server vil ha ett “keyword” foran seg, slik at “AdminWorm” kan scanne input og avgjøre hva som skal gjøres med det. Disse “keyword” benyttes også når gui sender forespørsler til server.

Informasjonsfeltet og knapper

Når brukeren velger en server i server listen, vil kort (hostname og ip) informasjon om denne dukke opp øverst til høyre. Brukeren vil da også få mulighet til å konfigurere script som den valgte serveren har tilgjengelig. Hvis brukeren har valgt en server uten script, for eksempel hovedserveren (ofte kalt tjener.intern), vil brukeren derimot få informasjon om at det ikke finnes script på denne serveren. Hvis brukeren derimot har valgt en server med script tilgjengelig vil det dukke opp en enkel dialog som gir brukeren mulighet til å forandre parametere i scriptenes konfigurasjonsfiler. Denne dialogen er dynamisk og bygger GUI på informasjon fått fra server. Denne informasjonen inneholder feltene som brukeren kan forandre og navnet på de. En liste holder oversikt over alle script på den valgte serveren og gir brukeren mulighet til å velge mellom dem. Parametrene til hvert script har da allerede blitt lest fra server, og vil forandre seg fra hvert enkelt script som velges. Det kan bare lagres forandringer gjort på ett script for hver gang. Hvis flere script skal forandres, må brukeren klikke på knappen for å få opp dialogen på nytt. Denne er plassert lett tilgjengelig under informasjonen for server. For hver gang brukeren åpner denne dialogen sender GUI en forespørsel til server og å sende over innholdet i konfigurasjonsfilene til de scriptene som er tilgjengelig på den valgte serveren. Dette er gjort for å få mest mulig oppdatert informasjon til brukeren hele tiden, og er også grunnen til at dialogen lukkes for hver gang brukeren har valgt å lagre endringene.

View

Nederst til høyre finnes “view” listen. Her får man se all output som det valgte “view” sender. Dette er enkelt bestående av en list–view. Kolonnene derimot blir lagd dynamisk for hvert “view”, for eksempel: hvis man ser på eksemplet på “gui.conf” litt lengre opp i teksten, ser man at under [View2] finnes det “Column1”, “Column2”, “Column3” med navn. Navnet etter “=” blir det som blir navn på kolonne(ne). “AdminWorm” leser dette og lager kolonner ut i fra dette. For hver gang brukeren velger ett nytt view i server listen, oppdateres dette, dvs kolonnen blir opprettet. Utifra det valgte view, vil det finnes mulige opprasjoner som kan utføres på dataene, for eksempel: fjerne bruker eller drepe prosesser. Disse mulighetene vil forandre seg fra “view” til “view”. Brukeren vil se dette i form av knapper som dukker opp øverst til høyre. Disse gjemmes bare, hvis den aktuelle opprasjonen ikke er mulig for det aktuelle “view”.

4.10.9 Konfigurasjonsmuligheter for utseende, og funksjonalitet for GUI

Når “AdminWorm” startes vil det (hvis ikke brukeren har valgt noe annet i konfigurasjons dialogen) forsøke å koble seg på serveren på valgt ip. “AdminWorm” benytter sockets til å opprette forbindelse. Dette vil være “localhost” som standard, men dette kan også forandres i konfigurasjons dialogen. “AdminWorm” benytter port 4243 for å koble seg på og for å sende kommandoer og motta informasjon fra serveren. Serveren vil hele tiden sende ut info som “AdminWorm” kan plukke opp og gjøre noe med. Vanligvis vil dette bety å liste det opp ordnet i “View” skjermbildet, men dette avhenger av konfigurasjonen for GUI. Denne konfigurasjonen finnes i “gui.conf” filen, og denne leses under oppstart av programmet. Brukeren har derimot muligheten til å kunne velge plassering og navn på denne filen under konfigurasjons dialogen i programmet, så det er ikke nødvendig at den heter “gui.conf”. Dette er bare standard. Et annet viktig punkt er at uten denne filen vil brukeren ikke kunne se noen view. Det kommer av at programmet tolker manglende fil som ingen view definert. Dermed vil man kun få listen med servere, uten denne filen. Et eksempel på gui.conf er vist under:

```
[Views]
View1=Users
View2=Applications

[Commands]
Kill_process=Kill Process
Kill_user=Kill User
```

```
[View1]
Column1=USER
OutputScript=return_users
Command1=Kill_user

[View2]
Column1=USER
Column2=PID
Column3=APP
OutputScript=return_apps
Command1=Kill_user
Command2=Kill_process

[Parameters]
Kill_user1=USER
Kill_process1=PID
Kill_process2=APP
```

I dette eksemplet er alle mulige “headere” (det som står i [ ]) vist. Forklaring til “headere”:

- **[Views]:**
  `Format: View<nr>=<navn på view>` (nr begynner på 1). Her finnes alle views som programmet skal behandle. Hvis man vil legge til flere gjøres dette her.

- **[Commands]:**
  `Format:<navn på script fil>=<navn på knapp>` Her legges alle kommandoer (scripts) som “AdminWorm” skal kunne kjøre inn. Det som ikke står her, vil ikke kunne kjøres. Det som legges inn her kommer fram som knapper under “Actions” i gui, hvis det valgte viewet definerer denne kommandoen. Infoen som er satt inn under de to neste “headere” er bare ment som et eksempel:

- **[View1]:** Her legges info inn om View1. View defineres på denne måten `[View<nr>]` (nr begynner på 1). En “view” definisjon har noen parametere, og de kommer under headeren. `Column<nr>=<navn på kolonne>` (nr starter på 1). Dette blir kolonner i “View” listen i GUI
  `OutputScript=<output script navn>`. Med “output script navn” menes navnet på det scriptet som kjøres for å hente informasjonen som dette viewet skal vise. Eks: return_users returnerer brukere på den gitte server.
  `Command<nr>=<navn på outputscript>` (nr begynner på 1). Her listes kommandoer som dette viewet kan kjøre. De som ikke er definert under Commands, men er definert her, vil ikke bli vist, og motsatt.

- **[View2]:** enda et eksempel på hvordan man definerer et “view”.

- **[Parameters]:**
  `Format:<navn på script fil<nr>>=<kolonne navn i view hvor parameterne til script skal hentes>` (nr begynner på 1, og trengs bare hvis flere parametere skal sendes). Her definerer man hvilke parametere som skal sendes tilbake til script når brukeren velger å kjøre en kommando. Eksempelvis: Drepe bruker. Dette vil da bli hentet ut av “view” listen, i gui. Hvis det ikke står noe her må brukeren selv skrive inn informasjonen som skal sendes tilbake.

Det er ikke nødvendig at “headere” kommer i den rekkefølgen som vist i dette eksemplet. “AdminWorm” sin gui konfigurasjonsfil leser vil finne fram til de riktige headere, uavhengig av rekkefølge. Det finnes en konfigurasjons fil til som berører gui, “preferences.xml”. Denne lagrer unna brukervalgte preferanser om hvordan

programmet skal starte (språk, koble til på oppstart og velge hvor gui konfigurasjonsfilen befinner seg). Denne har ikke noe med selve utseende på gui (utenom språk) og finnes derfor i egen fil. Brukeren har heller ikke mulighet til å kunne velge hvor denne skal befinne seg, eller velge et annet navn på den. Finnes ikke filen med navnet “preferences.xml” i samme mappe som programmet blir alt bare satt til standard verdier, og en ny fil blir opprettet. De standard verdiene som er tilgjengelig nå er vist i tabell 4.2. Brukeren har mulighet for å velge server som

Språk	Engelsk
Server	localhost
Koble til under oppstart	Ja
Plassering av gui konfigurasjons fil	./gui.conf

Tabell 4.2: Standard verdier i AdminWorm

skal kobles til, om man skal koble til under oppstart, og plassering av “gui.conf” i konfigurasjons dialogen for “AdminWorm”. Denne kan man finne ved å velge “Edit” og Config på menyen, eller trykke Ctrl+C (på engelsk språk vel og merke). Denne dialogen vil lese “preferences.xml” filen og vise brukeren denne informasjonen på en enkel måte. Brukeren kan da gjøre endringer og lagre disse ved å trykke OK. Ikke ialle tilfeller, men i de fleste må brukeren da starte programmet på nytt, for å se endringer. Men for alt untatt “koble til under oppstart” vil man oppleve endringene bare ved å koble fra og koble til server på nytt. Valg av språk ligger under “Edit” menyen og under “Languages”. Det språket som er valgt lese under oppstart ut av “preferences.xml” og blir valgt i menyen. Valg av språk kan skje under run-time. Hvis brukeren velger ett nytt språk vil alle strenger automatiskbytte til det valgte språket (hvis de har blitt oversatt). Dette vil også bli lagret unna i preferanse filen slik at programmet neste gang vil starte med dette språket. Dette er gjør mulig ved bruk at Qt sin translator klasse. Denne klassen vil oversette alle strenger med macroen tr(“ <text> ”) rundt seg. Et krav er derimot at hver klasse har en egen “SLOT” som kan kjøres hver gang språk bytte. Dette betyr at alle strenger må initialiseres i en egen funksjon som må kjøres i klassens konstruktor, og når bruker bytter språk. Derfor har alle klasser som inneholder strenger i “AdminWorm” en egen “SLOT” som heter “languageChange()”. Denne kobles da opp mot klassen som er ansvarlig for å bytte språk. For mere forklaring om “SIGNAL”/“SLOT”, se kort forklaring under “Nettverks kode”. Klassen og koden for håndtering av xml og språk bytte er skrevet av Fredrik B. Kjølstad og Bjørn Nilsen, “Stopmotion”.

4.10.10 Nettverks kode

GUI (og servere) benytter socket til opp og ned kobling. Protokollen som brukes er TCP. Vi har valgt å la kommunikasjon foregå på port 4243 [4], dette er tilfeldig valgt. Som nevnt tidligere brukes Qt biblioteket til all koding for GUI og dens funksjonalitet. Dermed benyttes også dette til nettverkskoden for gui'en. Qt sin socket kalles QSocket, denne gjør det lett å jobbe med socket løsninger. Qt sender såkalte "SIGNALS" mellom hverandre, og disse kobles opp mot såkalte "SLOTS". QSocket sender signaler om når den er koblet til, koblet fra, klar til å motta data, klar til å lese, feil,... Eksempel på en klasse med "SIGNAL" og "SLOTS":

```
class EksempelKlasse
{
Q_OBJECT
private slots:
void VisDialog();

signals:
void KnappTrykket();

public:
EksempelKlasse();
};
```

Q_OBJECT er en macro, og den er helt nødvendig for å få "SLOTS" og "SIGNALS" til å virke som de skal. "AdminWorm" hører etter såkaldte "ReadyRead" signaler fra socket klassen. Når det får et sånt betyr det at data (i "AdminWorm" sitt tilfelle betyr dette kun i tekst form) kan bli lest fra socket. Data som kommer inn kjøres gjennom en såkalt scanner for å definere typen på data som har blit mottat. Etter dette sendes data videre til sine parse funksjoner, basert på hvilken type det er. Parsing av tekst skjer ved bruk av QString (som er Qt sin egen string klasse) sin section funksjon. Denne deler opp strenger basert på hva den skal lete etter, for eksempel mellomrom. Dette gjøre tekstparsing enkelt, i forhold til vanlige c strenger.

4.10.11 Kompilering av Adminworm

Qt benytter sin egen makefile generator, og man må derfor ha disse pakkene `libqt3-mt-dev, qt3-dev-tools` og `libqt3-headers`. I tillegg be-

[4] Både port 4244 og 4243 er sjekket mot RFC1700 og disse er ledige [19]

nytter programmet libxml2 så pakken libxml2–dev er nødvendig. For å kompilere kjører man `qmake --project` så "qmake <navn på .pro fil>" og så make.

Forklaring på .pro fil

Qt benytter sin egen prosjekt fil. For å generere denne så kjører man "qmake -project" i den mappen hvor alle kildekode filene ligger. Hvis man skal oversette programmet til et annet språk må man legge til dette et sted i .pro filen:

```
TRANSLATIONS = <navn på program>_<språk>.ts
               <navn på program>_<språk>.ts
```

Eksempelvis fra "AdminWorm":

```
TRANSLATIONS = AdminWorm_no.ts
               AdminWorm_de.ts
               AdminWorm_se.ts
```

no = norsk
de = tysk
se = svensk
Engelsk er standard språket og man trenger ikke oversette dette da alle strenger i programmet er initialisert med dette. For å oversette programmet er et tre steg som man må gjøre i tillegg til å legge til "TRANSLATIONS" i .pro:

1. Kjøre lupdate <navn på pro fil>.pro i mappen hvor kildekoden ligger
2. Kjøre "Qt linguist", åpne for eksempel "AdminWorm_no.ts" som er generert av det forrige steget, og oversette alle strenger. "Qt linguist" er en enkel gui.
3. Kjøre lrelease <navn på pro fil>.pro. Dette vil generer eksempelvis "AdminWorm_no.qm" som inneholder oversettelsen. Denne må da kopieres dit som programmet krever at oversettelser skal ligge. I "AdminWorm" sitt tilfelle er dette ./translations.

I tillegg for å få libxml2 til å linkes riktig må dette legges til et sted i pro filen:

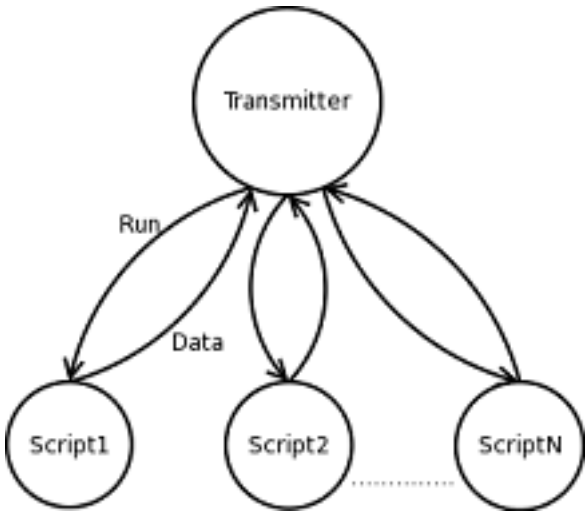
```
INCLUDEPATH += $$system(xml2-config --cflags | sed -e 's/-I//g')
LIBS += $$system(xml2-config --libs)
```

4.11 Forespørsler

Noe av teksten som sendes mellom Transmitter, Receiver og Listener vil kun sendes dersom en forespørsel sendes. Det vil si at for eksempel Listener sender en tekst til Receiver som forteller hva Receiver skal sende tilbake. Et eksempel på forespørsel: Listener sender teksten “AWGetErrorLog” til Receiver. Receiver vil da svare med tekst som viser eventuelle feil som har oppstått, for eksempel “AWTError: Key: RNTMER in return_apps.conf is not valid!)”. Videre i teksten hvor vi beskriver forespørsler mer detaljert, er det i prinsippet helt likt som i eksempelet over.

4.12 Transmitter

Det første Transmitter gjør er å lese konfigurasjonsfilen “Transmitter.conf”. Denne inneholder informasjon om hvilke scripts som skal startes og hva konfigurasjonsfilene deres heter. Det ble bestemt at scripts som skal startes må ligge i samme mappe, og det samme for konfigurasjons–filene. Ethvert script skal altså ha en egen konfigurasjonsfil som forteller hvor ofte det skal kjøre, hvilke linjer av data som skal sendes osv. For mer detaljert beskrivelse av disse konfigurasjonsfilene under seksjon D.2.2 For at Receiver skal kunne identifisere hva slags type



Figur 4.5: Transmitter kan starte ett eller flere scripts

informasjon Transmitter sender, begynner hver linje med en identifikasjonstekst. Receiver forstår da om en linje med data er en feilmelding, data fra et script osv. For en mer detaljert beskrivelse se seksjon D.3. En Transmitter har noen tilgjengelige forespørsler som Receiver kan sende for å stoppe en Transmitter. Den kan

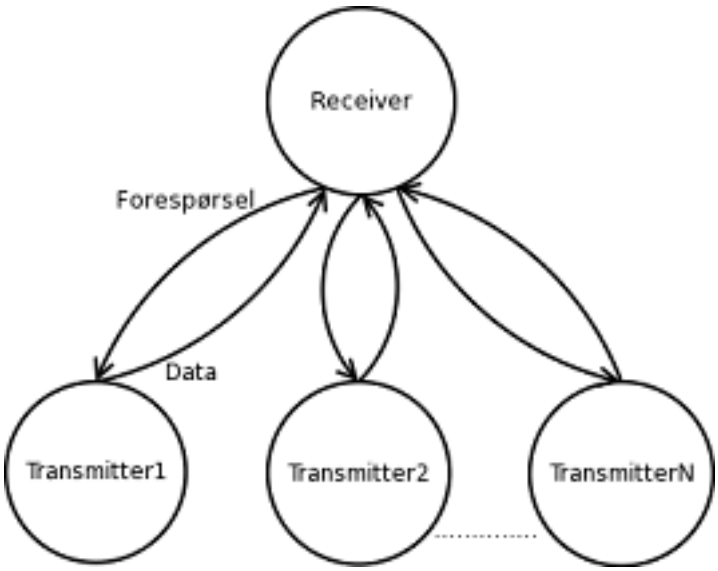
starte et script, selv om det for eksempel er satt til å bare kjøre hvert 10 sekund, og endre i et scripts konfigurasjonsfil.

4.12.1 Scripts

Transmitter samler informasjon basert på på data generert av kjørbare scripts for eksempel laget i Perl. Det er ikke begrenset til denne typen scripts, men det må være filer på en slik form at man kan starte de som en vanlig applikasjon, og de må generere data og deretter stoppe. for eksempel et lite Perl script som kjører kommandoen “ps -e ux” og deretter finner ut hvilke brukere som er pålogget. Nedenfor vises et kodeeksempel på et sånt script: Det er ikke laget noen grense i Transmitter for hvor tregt et script kan være, men det må til slutt stoppe, ellers vil det bare kjøres en gang.

4.13 Receiver

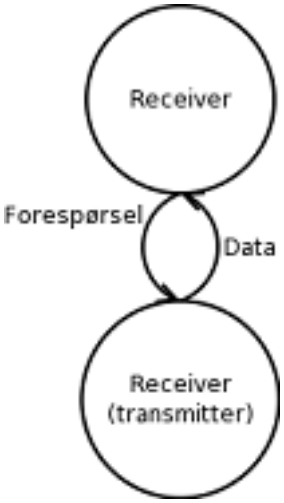
Receiver , som vist under i figur 4.6, er spesielt designet for å motta og behandle output fra én eller flere Transmittere. Output kan for eksempel være hvilke scripts som er startet, deres konfigurasjon, hva som er gyldige key’er og feilmeldinger. Dette blir lagret slik at en Listener kan få tak i det. All data en Receiver mottar



Figur 4.6: Receiver kan motta data fra en eller flere Transmittere

fra Transmittere blir sendt videre til alle Listenere som er tilkoblet. Dette gjør at en Receiver også kan fungere som en Listener. Receiveren som kobler seg på en

annen Receiver vil se på denne som en Transmitter, samtidig som den kan motta data fra andre Transmittere. Det vil si at man kan koble Receivere og Transmittere på mange forskjellige måter, og på denne måten sende data mellom forskjellige maskiner i så mange ledd man ønsker. Dette vises i den tidligere viste figuren 4.4. Hvordan gangen i det hele er, med tanke på en Receiver, er illustrert i figur 4.7 For



Figur 4.7: En Receiver kan koble seg på en annen Receiver. (Transmitter i parentes betyr at Receiveren oppfører seg som en Transmitter for den tilkoblede Receiver)

at Receiver skal kunne skille mellom en Transmitter og Listener, ble det gjort slik at Transmittere kobler seg på port 4244, mens Listenere kobler seg på port 4243. Det er mulig for Listener å sende en rekke forespørsler 4.11 til Receiver. Dette kan for eksempel være en forespørsel om å motta en liste over alle feil som eventuelt har oppstått, eller liste over hvilke Transmittere som er tilkoblet Receiveren. For en mer detaljert beskrivelse av forespørsler kommer senere under seksjon

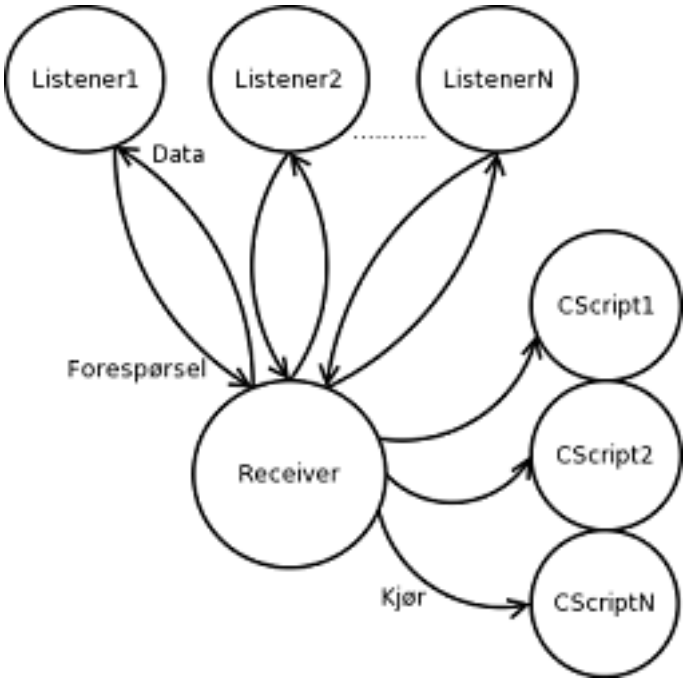
4.13.1 CommandScripts

En Receiver har muligheten til å starte scripts ved forespørsel fra en Listener på følgende måte: “AWCommand <hostname> <command_script> [<parameter-list>]” Receiver vil da lete etter <command_script> i mappen “CommandScripts” og prøve å kjøre dette med <hostname> som første parameter etterfulgt av eventuelle andre parametere i [<parameterlist>]. Det vil si at CommandScripts i seg selv må utvikles for å motta <hostname> som første parameter. Ved å ha en slik løsning, hvor CommandScripts kan kjøres ved en forespørsel, gir det muligheten for brukere å lage egne handlinger basert på data eventuelt mottat av en Transmitter. Grunnen til at hostname må komme som første parameter er for å identifisere

hvilken host handlingen skal gjøres mot. Vi hadde som oppgave å gi muligheten for en administrator å fjerne brukere, og løste dette ved å lage et CommandScript som gjennom SSH utfører den ønskede handlingen .

4.14 Listenere

En Listener, som vises i figur 4.8, kan koble seg på en Receiver på port 4243. Den vil da automatisk motta all output generert fra script på alle Transmittere som også er koblet til Receiver. Det er ikke satt noen begrensninger for hva Listener kan



Figur 4.8: En Receiver kan sende og motta data fra én eller flere Listener, og starte CommandScripts (CScript) på forespørsel.

være. Den trenger kun å ha mulighet for å koble seg på gjennom en TCP–socket. Dette gir muligheten for å utvikle senere løsninger rettet mot for eksempel web, hvor man har en web–side som jobber mot Receiver. Listener kan sende en rekke forespørsler til Receiver for å motta informasjon om et scripts konfigurasjonsfil.

4.15 Feilmeldinger

Serveren er forsøkt lagd så robust som mulig, med tanke på at en bruker kan skrive en del feil. En rapport over feil som er gjort i for eksempel et scripts konfigura-

sjonsfil, ikke–eksisterende filer og så videre, vil bli logget i Receiver. Listener kan motta denne rapporten ved å sende kommandoen “AWGetErrorLog” til Receiver. Receiver skal starte uansett, ettersom den ikke har konfigurasjonsfiler det kan gjøres feil i. Under er det listet opp noen av feilene som blir oppdaget og logget:

- Konfigurasjonsfilen til selve Transmitter mangler. Dette er det eneste som får Transmitter til å ikke starte, men feilen blir fortsatt logget av Receiver.
- Et script som er spesifisert til å starte finnes ikke, eller blir borte etter at det har blitt startet.
- Et script har ikke fått spesifisert konfigurasjonsfil, eller konfigurasjonsfilen mangler.
- Ved feil key, eller verdi i en konfigurasjonsfil, brukes standard.
- Hvis en Listener sender en ugyldig kommando, eller mangler et parameter, sendes en melding om dette tilbake.

Tilsammen utgjør dette at feil kan identifiseres på Listener nivå. Hvis en Listener sender en ugyldig kommando, eller mangler et parameter, sendes en melding om dette tilbake.

4.16 Installasjon

For å begynne å bruke Transmitter og Receiver er det en relativt enkel fremgangsmåte. Man må ha de tre filene “transmitter”, “transmitter.conf” og “receiver”, og legge disse i ønsket mappe. “transmitter.conf” D.2.1 inneholder informasjon om hvor script og deres konfigurasjonsfiler finnes, og hvilke scripts som skal startes. Dette må spesifiseres etter eget behov. Deretter er det bare å starte “receiver” som en vanlig kjørbar fil, eks: “./receiver”. Så kan Transmitter startes med parameter som forteller hvilken Receiver den skal kobles til, eks: “transmitter 10.0.2.2”. Hvis Transmitter startes uten parameter, kobler den til 10.0.2.2 som standard. Receiver og Transmitter trenger ikke å ligge på samme maskin, og flere Transmittere kan koble seg på samme Receiver. Viser her et eksempel av en struktur på hvordan alt ligger lagret på en maskin:

```
tjener.intern(10.0.2.2):
/usr/share/aw-Receiver/receiver
/usr/share/aw-Receiver/CommandScripts/CommandScript1
/usr/share/aw-Receiver/CommandScripts/CommandScript2
.
```

```
.
/usr/share/aw-Receiver/CommandScripts/CommandScriptN

ltspserver00.intern(10.0.2.10):
/usr/share/aw-Transmitter/transmitter
/usr/share/aw-Transmitter/transmitter.conf
/usr/share/aw-Transmitter/Scripts/Script1
/usr/share/aw-Transmitter/Scripts/Script2
.
.
/usr/share/aw-Transmitter/Scripts/ScriptN
/usr/share/aw-Transmitter/ScriptConfig/Config1
/usr/share/aw-Transmitter/ScriptConfig/Config2
.
.
/usr/share/aw-Transmitter/ScriptConfig/ConfigN

‘‘transmitter.conf’’ kunne da sett slik ut:

[Paths]
Script=/usr/share/aw-Transmitter/Scripts
ScriptConfig=/usr/share/aw-Transmitter/ScriptConfig
[Scripts]
Script1=Config1
Script2=Config2
.
.
ScriptN=ConfigN
[Receiver]
IP=10.0.2.2
```

4.17 Sikkerhet

Det ble bestemt av oppdragsgiver, at fokusen på prosjektet skulle ligge mer på funksjonalitet enn sikkerhet da dette var noe som kunne løses senere. Det er funnet følgende sikkerhetsrisikoer:

- Data sendes ukryptert
- Ingen verifisering av hoster

- Ingen flooding sperre[5]

Ellers er det f

[5]Store mengder data som sendes mot en Socket med intensjon å knekke systemet

Kapittel 5

Testing

5.1 Testing av Nagios og Munin

5.1.1 Nagios

Testing generellt

Etter at pakken var ferdig måtte den testes. Testplattformen besto i Skolelinux Sarge på hovedtjener (Tjener) og tynnklienttjenere (LTSP) lokalt, og tilsvarende på vår eksterne testlab på Kapp. Image lokalt var debian–edu av 11. mars 2005, på Kapp har det blitt brukt flere dagsferske images fra starten av mai.

Installasjonen av pakken foregikk slik:

`dpkg -i nagios-cfg-debian-edu_0.1.deb` og deretter `apt-get --f install` for å installere alle pakkene som nagios–cfg avhenger av. Installasjonen lokalt gikk bra første gangen, pakken, og avhengighetene, installerte og den spurte om den skulle starte konfigurasjonen av nagios–cfg. Etter konfigurasjonen hadde kjørt ferdig første gangen fant den alle hostene på Skolelinux–nettet lokalt, og web–interfacet kom opp på riktig måte. Pakken er bygd slik at dette er alt som trengs å gjøres såfremt man sitter i et Skolelinuxsystem. Scriptet "name–cfg" fungerte også etterhvert, se bugliste, og ønsket navn på lokale hoster ble endret. For å aktivere distribuert overvåkning oppover fra lab mot Kapp må dette settes opp i Nagios.cfg. obsess_over_services settes til 1, og dobbeltsjekkes mot "Process Info" i web–interfacet, se bugliste.

Testing på Kapp

Installasjonen på Kapp foregikk på samme måte, og hoster lokalt ble funnet og lagt inn. For å sjekke at informasjon kommer fra lab overvåker vi /var/log/syslog, og ser at "nsca" mottar informasjon. I brannmuren til Kapp må også riktig port som

"nsca" lytter på, åpnes. Hvilken port det dreier seg om settes selv til en ledig port over 1024. Etter å ha forsikret oss om at det lokale systemet faktisk distribuerte informasjon, ble hoster lagt til via "add–passive" scriptet, som fungerte som det skulle fra start. Hostene kom opp i web–interfacet på Kapp som de skulle med "P" ved siden av seg. Informasjonen på alle tjenestene i web–interfacet på Kapp ble sjekket mot web–interfacet her på lab, og det stemte. Informasjonen ble distribuert på riktig måte i henhold til oppsettet. Underveis avdekket gruppen følgende bugs:

- Selv om obsess_over_services er satt til 1 i Nagios.cfg kjører den ikke alltid ocsp_command som definert for distribuert overvåkning. Dette bør sjekkes via "Process Info" i web–interfacet, og er en bug i Nagios.
- Hadde litt problemer med "name–cfg" og filhåndteringen. Virket som den bare lagde tomme filer etter å ha lest inn informasjon fra de gamle. Gruppen løste dette ved å lagre all informasjon i arrayer før skriving til fil.

Siden denne pakken ikke har hatt noen eksterne testere annet enn på Kapp, selv om det har blitt forespurt ved flere anledninger, så har det ikke blitt avdekket flere bugs foreløpig.

5.1.2 Hvordan testes applikasjonen

Applikasjonene (Transmitter,Receiver og AdminWorm(GUI)) ble testet underveis i utviklingen ved hver nye kodesnutt. Etter en endring eller ny innlagt funksjonalitet, ble dette testet med en gang. For å oppdage hvor i koden eventuelle feil lå, ble det benyttet en funksjon som viste tekst, for å se hvor langt i koden man kom før en feil oppstod. I QT ble dette gjort ved å vise en "QMessageBox" og i Perl "print <tekst>". Underveis ble det avdekket følgende bugs:

- Receiver slutter av og til å sende data fra Transmittere av ukjent grunn. Den kan fortsatt motta forespørsler.
- Det kan oppstå feil hvis GUI–delen av applikasjonen kobbler seg på Receiver samtidig som en Transmitter.
-

5.1.3 Hvem tester applikasjonen

Applikasjonen er testet av gruppen selv og veileder, ettersom det ikke fantes tid eller ressurser til å kjøre større eksterne tester, for eksempel brukerne applikasjonen er rettet mot. Det er også et relativt komplisert oppsett som kreves for å få alt til å kjøre som det skal, noe som gjør at det å la tilfeldige personer teste systemet ikke lar seg gjennomføre ved dette stadium.

5.1.4 Hvilke system kan dette kjøres på

For å få til en fullverdig test i det miljøet det er ment for, kreves det tilgang til et Skolelinux system med Hovedservere og Tynnklienter. Gruppen har ikke hatt tid til å kjøre en så omfattende test, ettersom dette krever mye tid og ressurser.

5.1.5 Videre testing

For å få testet dette på en god måte, bør det testes på et Skolelinuxnettverk i større skala enn på labben vår. Det vil si hovedtjener med mer enn 5 tynnklienttjenere. Dette fordi det går ca 50 brukere på en tynnklienttjener og for en grunnskole er dette stort nok. Spesielt med tanke på applikasjonen bør det også testes med flere hovedtjenere samtidig for å prøve ut videresending av data i flere ledd.

Kapittel 6

Samarbeid med “Fri programvare i skolen” miljøet

6.1 Utviklingssamling

“Fri programvare i skolen” arrangerte helgen den 28. -30. januar 2005 en utviklersamling ved “Bøler” videregående skole i Oslo. Alle som bidro til prosjektet var velkommen. Det kom utviklere oversetter og testere fra bla. Canada, Hellas og hele Norge. Samlingen er i utganskpunktet gratis og gruppemedlemme som var med fikk refundert reisekostnadene og gratis overnatting hos en av de andre deltakerne av samlingen. Deltakerne på samlingen ble delt inn i grupper avhengig av hva det var de skulle gjøre på samlingen. Det var egne rom for de som skulle være med på drifting, oversetting, testing, skoleprosjekt osv. Halvor Borgen og Roger Carson var de to medlemmene som var med på samlingen. De fikk stort utbytte i form av god innføring i hvordan oppbygning av skolelinux nettverket skal være. (Selv om det ikke fungerte. . . hehe) Ellers fikk gruppemedlemmene snakket med bla *Ragnar Wisløff* som hjalp til forklaring av Nagios/Munin og prosjektoppgaven generelt.

6.2 News

Et viktig punkt som må tas med når det er snakk om Skolelinux og “Fri programvare i skolen” er *News* Det finnes alt i alt *åtte* forskjellige news grupper for skolelinux. Gruppene tar for seg ting som utvikling, feil, CVS osv.

- Hva er news? [6] News kan sees på som en forvokst elektronisk oppslagstavle som er delt opp i mange forskjellige emner(Nyhetsgrupper). Alle nyhetsgruppene vil kunne poste egne meldinger og svar, som alle kan lese. Det

finnes utallige forskjellige som kan være alt fra slektsforskere til orienteringsløperer. Newsgruppene er organisert i et hierarki og det er enkelt å søke gjennom dem. For å benytte seg av News trengs det for eksempel en browser, e–post program. gmane.linux.skolelinux.(cvs,devel, linux-i-skolen)

- Skolelinux benytter seg av dette til det fulle. Ved hjelp av News, er det enkelt å knytte sammen mange forskjellige utviklere fra forskjellige land på et sted. Her kan det diskuteres, kommenteres samt på en enkel måte gi ut vikig informasjon til mange mennesker samtidig. Blir en ny versjon av noe lastet opp på CVS'n kan alle se det for så å ta den nye versjonen/oppgraderingenen i bruk.

- Iløpet av prosjektperioden har gruppen benyttet seg av news iform av spørsmål på problemer som har oppstått. Spesielt når det gjelder Nagios/Munin, har vi kunnet enkelt "poste" et spørsmål/problem ut til newsgruppen og fått svar fra diverse Nagios/Munin brukere eller vedlikeholdere. Påmeldingen til utviklersamlingen ble også gjort via news. Mailen ble sendt og den ble postet på newsgruppen slik at alle kunne se hvem som skulle være med.

Kapittel 7

Avslutning

7.1 Drøftinger og valg underveis

Oppgaven forandret seg etterhvert litt etterhvert. Veileder, oppdragsgiver kom hele tiden med nye ideer ønsker Var et klart mål men med uklare spesefike oppgaver Valg av utv. modell som har fungert bra med ukentlig møter.

7.2 Kritikk av oppgaven

Definering av opggaven

Vi kom litt for sent i gang fordi det manglet klarhet i hva som skulle gjøres og hvordan det skulle løses. Det ble i utgangspunktet bestemt at vi skulle bruke Nagios til å løse administrasjonsoppgaver, for eksempel fjerne brukere og stoppe prosesser, i et Skolelinux miljø. Etter utviklingssamlingen, og en innføring i hva Nagios faktisk gjør, viste det seg å være en umulig oppgave, uten å skrive om selve koden i Nagios. Det ble da bestemt å utvikle en egen separat løsning for å administrere brukere og applikasjoner på grunnlag av data lest ut av Nagios. Dette førte da til at ting tok mye lenger tid, enn det som var planlagt.

Arbeidsmengden

Arbeidsmengden har vist seg å være passe for en gruppe på inntil 4 personer, Det har vært mye dokumentasjon å skrive og lese, mye å kode, samt ulike systemer å sette seg inn i, og har alle hatt nok å gjøre fra prosjektets start til slutt.

7.3 Hva har vi lært

Gruppen har måtte sette seg inn i Perl, C++ med QT–biblioteket, Nagios og Munin, i tillegg til å gjøre seg kjent med Skolelinux sin struktur og funksjonalitet. Ikke alle på gruppen hadde kjennskap til Linux fra før av, så utvikling på/for andre platformer har vært en lærerik opplevelse. Vi har innsett viktigheten av god planlegging, strukturert arbeide, og et godt samhold innad i gruppen. Alt i alt har dette vært gode erfaringer å ta med seg videre etter dette prosjektet, og erfaring som bil komme til nytte i ulike jobbsituasjoner.

7.4 Videre arbeid

- AdminWorm
  - Implementere sikkerhet i overføring av data mellom klient og server, data sendes nå ukryptert fra klienter mot server, og fra server mot GUI. En løsning gruppen har diskutert med oppdragsgiver er muligheter for å sende informasjon over SSH/SSL.
  - Implementere sikkerhet i utveksling av kommandoer mellom GUI og server, kommandoer med prosessID, brukernavn, og eventuelle passord bør sendes over SSH. Gruppen har foreløpig tenkt et system basert på passordløse nøkler og en dedikert bruker for applikasjonen. En GUI kan være alt som kan motta tekst eks. telnet/netcat, så det er ønskelig å implementere en sikrere løsning her.
  - Skalering. Gjøre det enkelt å legge til overvåkning av en ny maskin, ved at man kun trenger å spesifisere ip-addressen til maskinen som deretter automatisk får opprettet nødvendige mapper og tilsendt scripts.
  - Direkte konfigurering. Gjøre det mulig å konfigurere scriptene som kjører på en Transmitter, uten å måtte restarte Transmitter for at endringene skal tre i kraft.
  - Legge nettverkskode i en egen tråd, all lytting mot socket bør gå i egen tråd for å forsikre at alt av mottatte data blir tatt imot og prosessert på riktig måte hvis programmet driver med noe annet
  - Testing i større skala
- Nagios
  - Videreutvikle løsningen for distribuert overvåkning i større skala
  - Få pakken med i Debian–edu (Skolelinux) Sarge

  - Testing i større skala

- Trendanalyzer
  - Fullføre rammeverket og bygge en løsning basert på RRD–databaser
  - Bygge et system for å bestemme grense–/deltaverdier for målinger
  - Implementere løsningen som en del av Debian–edu, evt. i samarbeid med Munin.

7.5 Evaluering

7.5.1 Organisering

Gruppeleder har ikke fungert som en overordnet sjef. Det har ikke vært noen så store problemer at det ikke kunne løses uten bruk av dobbeltstemme. Arbeidsmodellen vi har valgt, åpnet for ganske fri fordeling av arbeidet.

7.5.2 Fordeling av arbeidet

Fordeling av arbeidet i gruppen har ende opp tilnærmet likt det som ble forklart over i seksjone 1.6.2. Dette har fungert veldig bra siden hver av gruppemedlemmen var filnke på området sitt. Det har vært ukentlige møter hvor kritiske problemer har blitt løst ved hjelp av. oppdragsgiver. Gruppen har fungert bra sammen og miljøet innad i gruppen har vært bra hele prosjektperioden.

7.5.3 Prosjekt som arbeidsform

Vi føler at et prosjekt som dette er en bra arbeidsform. Vi hadde sett frem til dette og håpet det ville være spennende og lærerikt. I løpet av de tre årene på HiG har det foreløpig bare vært gjennomført mindre prosjekter i form av innleveringer osv. Et hovedprosjekt gir en forsmak på hvordan det er å jobbe ute i det virkelige arbeidslivet, i form av samarbeid, planleggning, og ikke minst det å jobbe med et større prosjekt. Det vi har gjort, har potensialet til å påvirke hvordan Skolelinux–brukere jobber, og vil forhåpentligvis gjør livet til administratorer enklere.

7.5.4 Subjektiv opplevelse av hovedprosjektet

Det viktigste vi sitter igjen med etter løsing av denne oppgaven, er at planlegging og en klar definisjon av hva oppgaven går ut på står sentralt for god kommunikasjon i videre arbeid. Jo fler man er på en oppgave, jo viktigere er det at alle hver

for seg er klar over hva de skal gjøre. Vi har som oftest jobbet med en innstilling om at dersom man bare begynner å jobbe, vil alt falle på plass etterhvert. Etter denne oppgaven har vi forandret denne instillingen noe, og skjønner nå at en god krav–spesifikasjon er utrolig viktig. Etter en noe treg start på grunn av en nærmest ubrukelig problembeskrivelse fikk vi en forsinkelse på et par uker. Først etter utviklersamlingen på Bøler ble den endelige beskrivelsen fattet. Disse ukene kunne blitt brukt til å implementere feks. noe sikkerhet eller mer på trendanalyzeren, men vi har gjort alt vi i utgangspunktet skulle gjennomføre, og mere til, fordet. Utover dette har vi lært mye Perl og QT, noen mer enn andre, og hvordan man kan kode funksjonalitet i to forskjellige språk og få de til å jobbe sammen. Vi hadde heller aldri kodet nettverks-kode direkte, samtidig som det skulle fungere sammen med tråder, men har nå lært en hel del om dette. Så hovedsaklig er vi veldig fornøyd med å jobbet med denne oppgaven. Den har gått innom mange forskjellige områder innenfor sikkerhet, administrasjon og brukervennlighet/ergonomi, og vi har alle lært av hverandre underveis. Det har også vært en utfordring å lage en god og oversiktlig rapport, ettersom oppgaven har så mange forskjellige deler. Det er mye lettere å lage en løsning og rapport til én enkeltstående applikasjon, så det har vært spennende å prøve noe ganske annet. Alt i alt synes vi at hovedprosjekt er en god måte å ta til seg lærdom på. Det vil for eksempel være mye lettere å jobbe med et neste prosjekt siden man lærer å jobbe bedre sammen, og forstår verdien av planlegging.

7.6 Konklusjon

Gruppen har hatt muligheten til å være med på et større prosjekt som kan ha innvirkning på mange Skolelinux–administratorer over hele verden. Gruppen har fått mye nyttig læring om hvordan nettverksovervåkning kan fungere, samt scripting i Perl, programmering i C++, og dokumentasjon ved hjelp av LaTeX 2ε. Vi føler at vi har lært mye om hvordan det er å jobbe i prosjekt sammen med andre, og ikke minst om hvordan det er å jobbe i et prosjekt som har potensialet til å påvirke andre mennesker.

Gruppen er veldig fornøyd med resulatet. Vi har levert et produkt som tilfredsstiller de krav, og ønsker, oppdragsgiver i utgangspunktet stilte og enda litt til. Oppdragsgiver virker fornøyd med resultatet og vi håper at oppdragsgiver, og administratorer av Skolelinux–nettverk, vil kunne benytte eller videreutvikle produktet slik at administratorer skal få en enklerer og bedre hverdag.

Kapittel 8

Litteraturliste

Bibliografi

[1] Thread (computer science), 2005. URL: http://en.wikipedia.org/wiki/Thread_%28computer_science%29 (11.05.05).

[2] Vidar Andersen Alexander Limi, Alan Runyan. Arkitektur, 2005. URL: http://www.skolelinux.org/no/product/architecture/document_view (08.05.05).

[3] Lindpro AS. Lindpro, 2004. URL: http://www.linpro.no/ (08.05.05).

[4] LinuxLabs AS. Linuxlabs as, 2005. URL: http://www.linuxlabs.no/(08.05.05).

[5] Perl Training Australia. Perl training australia - why perl?, 2005. URL: http://perltraining.com.au/whyperl.html (08.05.05).

[6] Lars Arne Brekken. Hva er news, 1999. URL: http://www.polydesign.no/npn/index.asp?artikkel=13 (08.05.05).

[7] Mark Burgess. cfenvd, 2005. URL: http://www.cfengine.org/manpages.phtml?page=cfenvd.html.txt (12.05.05).

[8] Morten Sporild Christian Frøhaug, Martin Dahl. Skolelinux - user admin, 2003. URL: http://developer.skolelinux.no/info/studentgrupper/2003-HiG-useradmin/ (08.05.05).

[9] Cprogramming.com. Xor-encryption, 2005. URL: http://www.cprogramming.com/tutorial/xor.html (08.05.05).

[10] Debian. About debian, 2005. URL:http://www.debian.org/intro/about (08.05.05).

[11] R. Droms. Rfc 2131, 2005.
URL: http://www.faqs.org/rfcs/rfc2131.html (08.05.05).

[12] The Apache Software Foundation. Welcome! - the apache software foundation, 2005.
URL: http://www.apache.org/ (08.05.05).

[13] Google. Google, 2005.
URL: http://www.google.no/ (08.05.05).

[14] Red Hat. Red hat enterprise linux, 2005.
URL: http://www.redhat.com/software/rhel/ (08.05.05).

[15] HIG. Høyskolen i gjøvik, 2005.
URL: www.hig.no (08.05.05).

[16] HiG. Retningslinjer for skriving av prosjektrapporten, 2005.
URL:
http://www.hig.no/dav/AF204D9F70EA454E851EF09EDEB3EC92.pdf
(08.05.05).

[17] Christian Schwarz Ian Jackson. Debian policy manual, 1998.
URL: http://www.debian.org/doc/debian-policy/ch-binary.html (08.05.05).

[18] Linux Online Inc. What is linux, 2005.
URL: http://www.linux.org/info/index.html (08.05.05).

[19] J. Postel J. Reynolds. Rfc 1700 (rfc1700), 2005.
URL: http://www.faqs.org/rfcs/rfc1700.html (12.05.05).

[20] Inc. Novell. Suse linux professional 9.3, 2005.
URL: http://www.novell.com/products/linuxprofessional/overview.html
(08.05.05).

[21] Tobias Oetiker. About rrdtool, 2005.
URL: http://people.ee.ethz.ch/ oetiker/webtools/rrdtool/ (08.05.05).

[22] Marco Paganini. The angel network monitor, 1998.
URL: http://www.paganini.net/angel/ (08.05.05).

[23] Prashant. Cvs - concurrent versions system, 2005.
URL: http://www.gnu.org/software/cvs/ (08.05.05).

[24] Kirsten Ribu. Inkrementell utvikling, 2005.
URL: http://www.iu.hio.no/ kirstenr/systemutvikling/slides2005/2_Systemutviklingsmetoder.ppt#34
(08.05.05).

[25] Mandriva/Mandrakesoft SA. The mandriva linux concept, 2005.
URL: http://www.mandrivalinux.com/en/concept.php3 (08.05.05).

[26] Skolelinux. Arkitektur, 2005.
URL: http://www.skolelinux.org/no/product/architecture/document_view
(08.05.05).

[27] Skolelinux. Organisasjonen skolelinux, 2005.
URL: http://www.skolelinux.org/no/about/organization(08.05.05)/.

[28] Snort. Snort, 2005.
URL: http://www.snort.org/ (08.05.05).

[29] Quest Software. Big brother professional edition, 2005.
URL: http://www.bb4.com/aboutus.html (08.05.05).

[30] SourceForge.net. Sourceforge.net cvs repository - directory - cvs: munin/munin, 2005.
URL: http://cvs.sourceforge.net/viewcvs.py/munin/munin (08.05.05).

[31] Trolltech. Qt overview, 2004.
URL: http://www.trolltech.com/products/qt/ (08.05.05).

[32] Trolltech. Qt overwiew, 2005.
URL: http://www.trolltech.com/products/qt/ (08.05.05).

[33] Wikipedia. Advanced packaging tool, 2005. URL:
http://en.wikipedia.org/wiki/Apt-get (11.05.05).

[34] Wikipedia. Berkeley software distribution, 2005. URL:
http://en.wikipedia.org/wiki/BSD (11.05.05).

[35] Wikipedia. Computer sockets, 2005.

[36] Wikipedia. Mac os x, 2005. URL: http://en.wikipedia.org/wiki/OSX
(11.05.05).

[37] Wikipedia. Microsoft windows, 2005. URL:
http://en.wikipedia.org/wiki/Microsoft_Windows (11.05.05).

[38] Wikipedia. Transport layer security, 2005. URL: http://en.wikipedia.org/wiki/Secure_Sockets_Layer (11.05.05).

[39] the free encyclopedia Wikipedia. Graphical user interface, 2005. URL: http://en.wikipedia.org/wiki/GUI (08.05.05).

[40] the free encyclopedia Wikipedia. Network address translation, 2005. URL: http://en.wikipedia.org/wiki/NAT (08.05.05).

[41] the free encyclopedia Wikipedia. Network switch, 2005. URL: http://en.wikipedia.org/wiki/Network_switch (08.05.05).

[42] the free encyclopedia. Wikipedia. Open source, 2005. URL: http://en.wikipedia.org/wiki/Open_Source (08.05.05).

[43] the free encyclopedia Wikipedia. Router, 2005. URL: http://en.wikipedia.org/wiki/Router (08.05.05).

[44] the free encyclopedia Wikipedia. Secure shell, 2005. URL: http://en.wikipedia.org/wiki/Ssh (08.05.05).