BACHELOROPPGAVE:

Snortmanager

FORFATTERE:
Christer Vaskinn
Kristian Wikestad

DATO:
23.05.2012

Sammendrag av Bacheloroppgaven

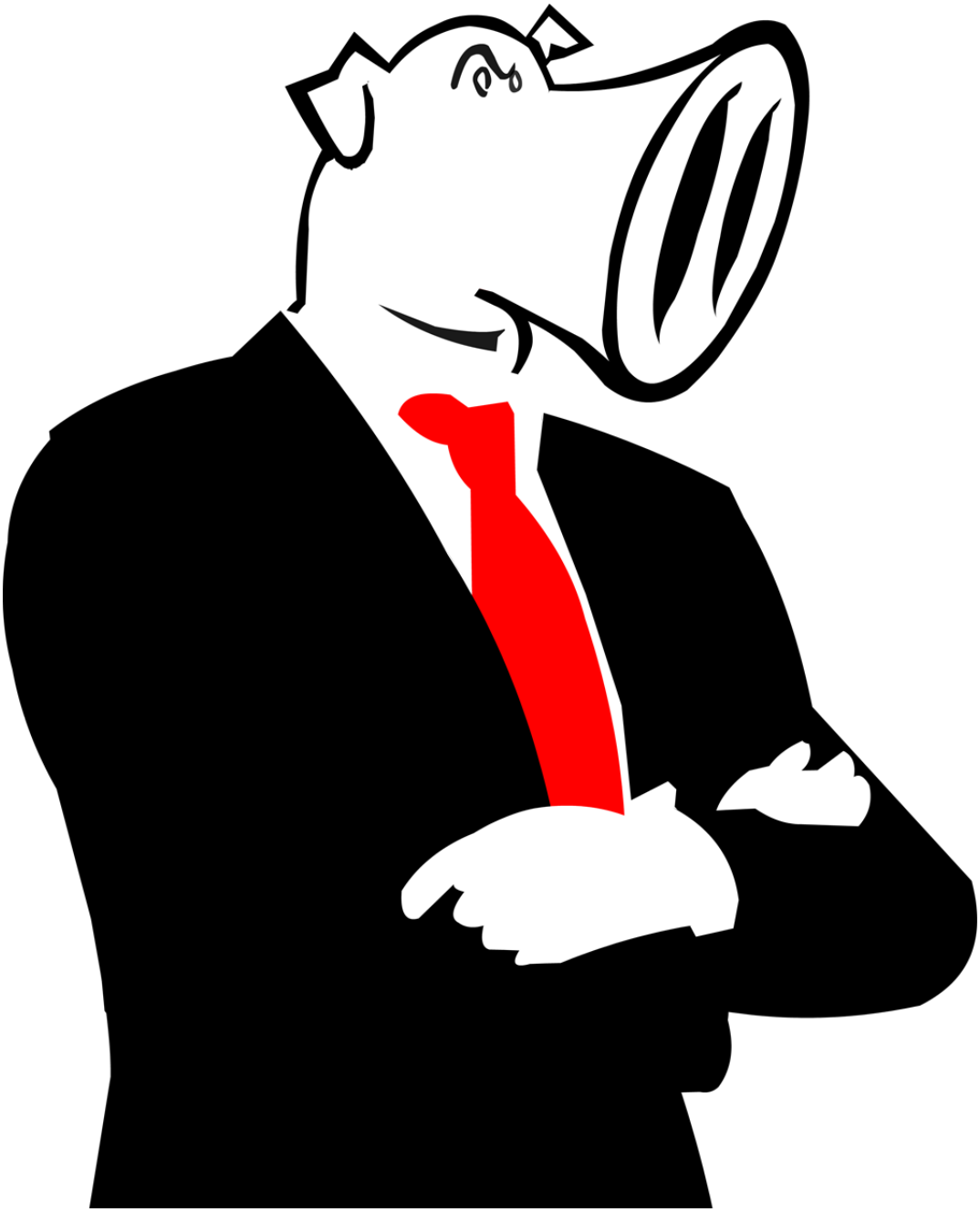
Tittel:	Snortmanager	Nr: -
		Dato: 23.05.2012
Deltakere:	Christer Vaskinn	
	Kristian Wikestad	
Veiledere:	Slobodan Petrovic	
Oppdragsgiver:	mnemonic as	
Kontaktperson:	Roger Storløkken	
Stikkord	Norway, Norsk	
Antall sider: 131	Antall vedlegg: 4	Tilgjengelighet: Åpen

Kort beskrivelse av bacheloroppgaven:
Det nordiske selskapet mnemonic as [sic] spesialiserer seg på tjenesteleveranser innen informasjonssikkerhet. En av tjenestene de leverer er nettverksovervåkning, hvor de blant annet benytter Snort for å analysere netttverkstrafikk. Dette anerkjente systemet basert på åpen kildekode har millioner av installasjoner på verdensbasis, med tilsvarende mange gode erfaringer innen nettverksanalyse. Økt kundeportefølje med behov for overvåkning av sine datanettverk har ført til behov for et godt verktøy for å administrere og konfigurere flere Snortsensorer. Det blir mer komplisert når kunder har flere nettverk med varierte behov og krav. Til nå har mnemonic brukt et egenprodusert verktøy for å utføre konfigurasjonen, men dette er basert på flere arbeidskrevende manuelle operasjoner. Dagens metode for å konfigurere sensorer er ineffektiv, lett utsatt for feil og uten god historikk. Selskapet trenger et enklere verktøy for å forbedre og forenkle oppgaven. Snortmanager er et webbasert verktøy for å administere og konfigurere Snortsensorer. Brukerene vil gjennom et enkelt grensesnitt kunne legge inn informasjon om sensorene. Det vil så produseres en konfigurasjonsfil for hver sensor registrert i systemet. De forskjellige behovene defineres med en policy, som er sett med regler skrevet i klartekst. Disse blir opprettet av brukerene, og brukt av Snortmanager når det skal produsere konfigurasjonsfiler for Snortsensorer. Filene kan senere overføres til sensoren som skal utføre analyse av datatrafikk. Snortmanager vil også laste ned Snort regler og signaturer brukt av sensorene for å kunne oppdage abnormaliteter i nettverkstrafikken. Disse reglene skal til enhver tid være oppdatert i ht. til kilder, men også ha et fullt bibliotek av tidligere regler. Under utviklingen har gruppen i hovedsak analysert mnemonics behov, men også de generelle behovene rundt administrasjon av flere Snortsensorer. Sluttproduktet er en enkel og oversiktlig måte å administere Snortsensorer på med full endringshistorikk.

Summary of Graduate Project

Title:	Snortmanager	Nr: -
		Date: 23.05.2012
Participants:	Christer Vaskinn	
	Kristian Wikestad	
Supervisor:	Slobodan Petrovic	
Employer:	mnemonic as	
Contact person:	Roger Storløkken	
Keywords	Snort, Administration, Python, Open Source, Linux	
Pages: 131	Appendixes: 4	Availability: Open

Short description of the main project:
mnemonic as [sic] provides information security service for the nordic market. One of the services they provide is network monitoring, where they use Snort to analyze network traffic. This well known open source IDS has millions of installations world-wide and just as many success stories. With an increase of customers in need of network monitoring, mnemonic realized that there are no good administration tools for the administration and configuration of multiple Snort sensors. It becomes more complicated when several customers have different networks, each with different needs and demands. mnemonic is currently using a tool developed in-house, but it is based on several complicated manual processes. The company requires a simpler tool to improve this task, thus making it more effective. Snortmanager is a Web based tool to administer and configure several Snort sensors at the same time. Through a simple user interface the user can add information about each sensor the system will administer. Snortmanager will use this information to produce a custom Snort configuration file. The different customizations are based on the needs and requirements of each sensor, network or company. It is defined by policies created in Snortmanager, containing a simple set of text based rules. When the configurations are created, they can be transferred to the sensor monitoring network traffic. Snortmanager downloads Snort signatures from external sources and stores them in a database. This database will be updated according to the external source, but also contain a full library of old signatures. During the development of Snortmanger we have analyzed mnemonics needs and requirments, and the general requirements with administrating several Snort sensors. Snortmanager gives IT professionals and administrators a simple way to administer their sensors, in addition to a full change history.



Forord

Når høgkolen presenterte bacheloroppgaver høsten 2011, var det å jobben med en av Nordens største av tjenester innen informasjonssikkerhet, vanskelig å motstå. Dette prosjektet går ut på å utvikle et nytt verktøy for å administrere og konfigurere Snort sensorer.

Prosjektet kunne ikke blitt hva det er, uten god støtte fra flere ansatte hos mnemonic as. Oppdragsgiver har gjennom hele prosjektet vært en nyttig ressurs, både på den utviklingen av Snortmanager og sluttrapporten. Gruppens kontaktperson hos oppdragsgiver Roger Storløkken, har hele tiden gitt raske tilbakemeldinger. Stian Jahr har vært prosjektets tekniske kontakt, host oppdragsgiveren. Under utviklingen av Snortmanager har han brukt sine kunnskaper om Snort for å kunne gi oss tips, og råd. Han har også gitt viktige tilbakemeldinger på designet, arkitekturen, og testene av Snortmanager.

Gruppens veileder Slobodan Petrovic har under prosjektet fungert som vitenskaplig rådgiver. Med hyppige statusmøter har han hele tiden holdt seg oppdatert med prosjektets fremgang, og har kommet med tilbakemeldinger på deler av rapporten fortløpende. Han var også viktig for å rettlede, og overbevise gruppen om at prosjektet var på rett vei.

Medlemmene av prosjektet ønsker også å takke følgende personer for deres hjelp under prosjektet:

- Ida Nyborg for korrekturlesning og tilbakemelding på sluttrapport.
- Tom Røise (HiG) for tips og råd under skrivingen av kravspesifikasjon og designdokument.
- CherryPy-samfunnet på IRC-nettverket irc.oftc.net.

Innhold

Forord . . . . . **iv**
Innhold . . . . . **v**
Figurer . . . . . **viii**
Tabeller . . . . . **x**
1 Innledning **1**
1.1 Problemområde 1
1.2 Oppgavedefinisjon 1
1.3 Oppdragsgiver 1
1.4 Formål 2
1.5 Bakgrunn 2
1.6 Avgrensning 2
1.7 Målgruppe 2
1.8 Valg av utviklingsmodell og organisering av arbeid 2
1.9 Snort 4
1.10 Litt om intrusion detection og prevention systemer generelt 4
1.10.1 Snort signaturer 5
1.11 Verktøy 5
1.11.1 Versjonskontrollsystem 5
1.11.2 Oversikt over gjøremål 6
1.11.3 reStructuredText 6
1.12 Konvensjoner brukt i denne rapporten. 7
2 Kravspesifikasjon **8**
2.1 Generell beskrivelse 8
2.1.1 Beskrivelse av bruker 8
2.1.2 Antagelser 8
2.1.3 Begrensninger 8
2.1.4 Krav til løsningen 9
2.1.5 Bruksmiljø 9
2.2 Funksjonelle krav 9
2.2.1 Liste over aktører 9
2.2.2 Overordnet use casebeskrivelser 9
2.2.3 Extended use case beskrivelser 13
2.3 Ikke-funksjonelle krav 15
2.3.1 Sikkerhet 15
2.3.2 Krav for vedlikehold av kode 15
2.3.3 Portabilitet 15
2.3.4 Miljøpåvirkning 15

2.4 Krav til enkeltkomponenter . . . 15
2.4.1 Rapportering . . . 15
2.5 Policy . . . 15
3 Design . . . 17
3.1 Beskrivelse av systemarkitektur . . . 17
3.2 Design av moduler . . . 19
3.2.1 Bakgrunnsjobber . . . 20
3.2.2 Systemlås . . . 21
3.2.3 Oppdatering av regler . . . 22
3.2.4 Administrasjonsverktøy for sensor . . . 25
3.2.5 Registrering av sensor . . . 25
3.2.6 Logging . . . 25
3.2.7 Rapportmodul . . . 26
3.2.8 Design av policy . . . 27
3.3 Logical view . . . 27
3.4 Deployment view . . . 28
3.5 Valg av teknologi . . . 29
3.5.1 Rammeverk for webpublisering . . . 30
3.5.2 Verktøy og rammeverk for håndtering av databaser . . . 30
3.5.3 Presentasjonsverktøy for mot sluttbrukere . . . 30
3.5.4 Planlagte oppgaver . . . 31
4 Implementering . . . 32
4.1 Valg av lisens . . . 32
4.2 Utviklingsmiljø . . . 32
4.2.1 Virtualisering . . . 32
4.2.2 Operativssystem . . . 33
4.2.3 Python . . . 34
4.2.4 MySQL . . . 34
4.2.5 SSH . . . 34
4.3 Inspirasjon og bruk av åpen kildekode. . . . 35
4.4 Implementering av CherryPy . . . 36
4.4.1 Mapping av URL og MVC-Controllere . . . 37
4.4.2 MVC-Controllere . . . 38
4.5 SQLAlchemy . . . 39
4.6 Visuelt design og templates . . . 42
4.7 Filstrukturen til Snortmanager . . . 44
4.8 Administrasjonsskriptet Snortmanager.py . . . 46
4.9 Implementering av bakgrunnsjobber . . . 48
4.10 Planlagte oppgaver . . . 50
4.11 Organisering av Snortsignaturer . . . 52
4.12 Implementering av policy . . . 54
4.13 Implementering av systemlås . . . 56

4.14 Implementering av sensor . . . 57
4.14.1 Opprettelse av ny sensor . . . 57
4.14.2 Endre sensor . . . 59
4.15 Implementering av logging . . . 59
4.15.1 Logging av handlinger gjort av bruker . . . 60
4.15.2 Logging av systemaktivitet . . . 61
4.16 Alvorlighetsgrad for logging av hendelser. . . . 62
4.17 Dokumentasjon av kildekode . . . 62
5 Testing og kvalitetssikring . . . 64
5.1 Testmetode . . . 64
5.2 Punkter som ble vektlagt under testingen . . . 64
5.3 Brukergrensesnittet . . . 64
5.4 Funksjonalitet . . . 65
5.5 Kildekode . . . 65
5.6 Kompatibilitet . . . 65
5.7 Testing av moduler . . . 65
5.7.1 Nedlastning og oppdatering av signaturer . . . 65
5.7.2 Testing av planlagte oppgaver . . . 68
5.8 Opprettelse av policy . . . 69
5.8.1 Test av produksjon av konfigurajonsfiler . . . 71
5.8.2 Testing av sensoradministrasjon . . . 78
5.8.3 Test av systemlås . . . 80
5.8.4 Testing av logging . . . 82
6 Avslutning . . . 85
6.1 Drøfting . . . 85
6.2 Resultater . . . 85
6.3 Valg tatt underveis . . . 86
6.4 Snortmanagers fremtid som åpen kildekode . . . 86
6.4.1 Kjente feil . . . 87
6.4.2 Distribusjon av kildekoden . . . 88
6.5 Evaluering av gruppens arbeid . . . 88
6.5.1 Organisering . . . 88
6.5.2 Fordeling av arbeidet . . . 89
6.5.3 Prosjekt som arbeidsform . . . 89
A Forprosjekt . . . 91
B Prosjektavtale . . . 108
C Møtereferater . . . 111
D Logg . . . 120

Figurer

1 Illustrasjon av inkrementell modell . . . 3
2 Use case diagram . . . 13
3 Use case diagram . . . 17
4 Use case diagram . . . 18
5 Use case diagram . . . 19
6 Use case diagram . . . 21
7 Use case diagram . . . 22
8 Sekvensdiagram for produksjon av konfigurasjonsfiler . . . 24
9 Illustrasjon av en policy . . . 27
10 Logical view . . . 28
11 Deployment view skisse . . . 29
12 Diagram av webdesign . . . 33
13 Diagram av webdesign . . . 43
14 Databasedesign av policy . . . 44
15 Screenshot av Snortmanager.py . . . 46
16 Screenshot av bakgrunnsjobber . . . 48
17 Skjermbilde av schedule . . . 51
18 Diagram over lagring av regler i databasen . . . 52
19 Databasedesign av policy . . . 54
20 Eksempel på add new sensor . . . 58
21 Eksempel på tabellen EventLog i database . . . 60
22 Skjermbilde av truncate . . . 66
23 Skjermbilde av første test . . . 66
24 Nye signaturer lagt til. . . . 67
25 Skjermbilde av kommandlinje etter andre test . . . 67
26 Skjermbilde av kommandlinje etter andre test . . . 68
27 Skjermbilde av policyen Stark . . . 69
28 Ferdig policy . . . 70
29 Struktur i databasen . . . 70
30 Skjermbilde og database etter fjernet policy . . . 71
31 Skjermbilde av policyliste . . . 72
32 Skjermbildepolicyen HiG . . . 72
33 Skjermbilde av policyen Stark . . . 73
34 Skjermbilde av bakgrunnsjobber . . . 74
35 Produsert konfigurasjon for Hig policy . . . 74
36 Produsert konfigurasjon for Stark policy. . . . 74
37 Produsert konfigurasjon for `Daily Bugle` policy. . . . 75

38 OSCORP Policy 76
39 Resultat sid 6499 77
40 Resultat av duplikatsjekk 77
41 Resultat av regex søk 78
42 Eksempel på opprettelse av ny sensor 78
43 Eksempel på opprettelse av ny sensor i database 79
44 Eksempel på opprettelse av ny lokasjon 79
45 Illustrasjon av tabellen SensorLocation 80
46 Systemet er låst 81
47 Systemet er låst 82
48 Test av logg til fil 83
49 Logging til database 84

Tabeller

1 Registrere ny sensor 14
2 Definere og vedlikeholde policy 14
3 Syntaks for policyproduksjon 16

1 Innledning

1.1 Problemområde

De fleste bedrifter har avanserte infrastrukturer og serverløsninger for å tilby sine ansatte og kunder digitale tjenester. Disse infrastrukturene inneholder ofte bedriftshemmeligheter og sensitive personopplysninger. Med det økte trusselbildet hvor bedrifter blir utsatt for spionasje og målrettede angrep fra kriminelle, utvikles et stort behov for en sikker infrastruktur.

Selskaper som mnemonic as [sic] tilbyr i dag tjenester innenfor sikring av nettverk for kunder som ikke har ressurser til å gjøre dette selv. mnemonic tilbyr overvåkning av nettverket til kunden, og for å gjøre dette benytter de Intrusion Detection Systemer(IDS). Disse systemene blir brukt til analysering av nettverkstrafikk for å finne abnormaliteter. Et eksempel på et slikt system er Snort, som er basert på åpen kildekode.

En av utfordringene til oppdragsgiver i dag, er at administreringen av signaturene brukt i Snort tar for lang tid. De vil effektivisere måten dette blir gjort på, men har ikke til nå funnet en komplett løsning.

Oppdragsgiver har av denne grunn etterlyst et system, som kan tilfredstille behovene. Systemet skal utvikles i åpen kildekode, så det kan videreutvikles etter brukers behov. Dermed blir det også tilgjengelig for andre som måtte ha bruk for et slikt system.

1.2 Oppgavedefinisjon

Systemet Snortmanager skal lages i henhold til vilkår og krav om funksjonalitetm sin avtalt med oppdragsgiver. Brukergrensesnittet skal være webbasert, for at majoriteten av funksjonaliteten skal være tilgjengelig fra en nettleser. Systemets hovedfunksjonaliteter er periodisk nedlastning av nye signaturer fra eksterne leverandører, administrering av sensorer, produksjon av policy og regelfiler, rapportering om bruk av regler og logging av bruker og systemaktiviteter.

1.3 Oppdragsgiver

Oppdragsgiver for prosjektet er selskapet mnemonic as, en av Nordens største leverandører innen IT og informasjonssikkerhet. Selskapet har rundt 100 ansatte, med kontorer i Oslo, Sandnes og Stockholm. mnemonic er et anerkjent selskap med kunder i de fleste størrelser og bransjer over hele verden.

De leverer tjenester innen sikkerhetsovervåkning, risikostyring, applikasjonssikkerhet og sikker infrastruktur. Sikkerhetsovervåkning innebærer en døgnbemannet Security Operations Centre (SOC), som analyserer logger og alarmer fra utstyr på deres og kundenes nettverk. Slikt utstyr er blant annet Intrusion Detection System(IDS) sensor, som overvåker nettverket for skadelig aktivitet.

1.4 Formål

Oppgavens formål er å skape en enklere og mer effektiv måte for administrering av systemet Snort. Bruker skal kunne bruke betraktelig kortere tid på hverdagslige gjøremål, og automatikk skal fjerne flere av oppgavene man tidligere har måttet gjøre manuelt. Denne innsparingen av arbeidstimer vil føre til lavere kostnader for bedriften, og motivere ansatte som slipper repetitivt og unødig arbeid.

1.5 Bakgrunn

Gruppemedlemmene studerer Bachelor i Informasjonssikkerhet ved Høgskolen i Gjøvik. Gruppen har variert erfaring fra programmering, databaser og webapplikasjoner i skole-, arbeids- og fritidssammenheng. I hovedsak er tidligere utviklingserfaringer i C-inspirerte objektorienterte språk som C++, PHP og Java. Kunnskap i databasemotorer varierer mellom medlemmene, men begge har middels til god erfaring med MySQL. Erfaringen med webapplikasjoner har tidligere vært xAMP-modellen, som baserer seg på Apache, MySQL og PHP.

1.6 Avgrensning

Oppgaven skal avgrenses for bruk av profesjonelle med erfaringer innenfor fagfeltet sikkerhetsovervåkning, og trenger derfor ikke simplifiseres for forbrukere. Grunnet oppgavens omfang og kompleksitet vil gruppen etter oppfordring fra oppdragsgiver ikke prioritere visuelt design av webgrensesnittet. Oppgaven vil i tillegg avgrenses til punkter nevnt i oppgavedefinisjonen.

1.7 Målgruppe

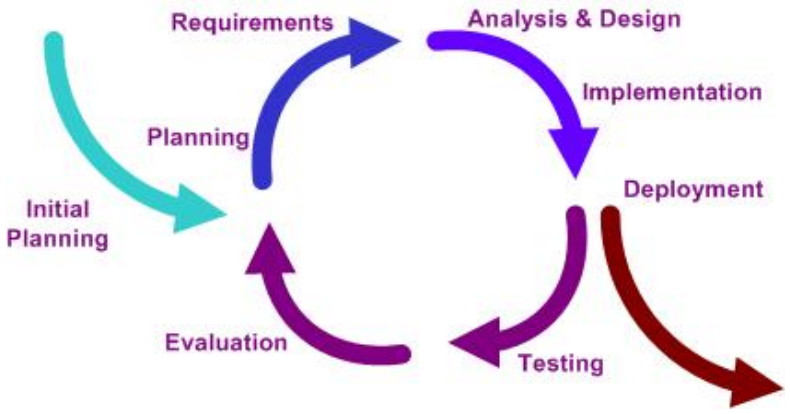
Oppgaven er i hovedsak utført for oppdragsgiver og det profesjonelle marked. Prosjektet vil være aktuelt for bedrifter innenfor samme fagfelt, og vil kunne videreutvikles og tilpasses etter behov.

1.8 Valg av utviklingsmodell og organisering av arbeid

Utviklingsmodellen vil definere hvordan gruppen skal planlegge utviklingen og framdriften av prosjektet. Spørsmålene gruppen tok i betraktning når utviklingsmodell skulle velges var:

- Hvor mye tid skal disponeres til utvikling?
- Hva passer best for en gruppe på to?
- Hva passer best for et prosjekt på denne størrelsen, og på modulbasert utvikling?

Gruppen mente inkrementell modell og Scrum var de beste valgene basert på kunnskapen gruppen hadde om modellene og på tilbakemeldinger. Inkrementell uvikling passet godt for gruppen på grunn av gruppens størrelse, og utviklingen vil skje modul for modul. Inkrementell utvikling kjennetegnes av repetitiv utvikling. Under forklares det kort hvordan gangen i inkrementell utvikling skjer.



Figur 1: lllustrasjon av en inkrementell modell [1]

1. Planleggingsfasen. Programmet planlegges grovt før en mer detaljert planleggingsfase. Her planlegges hvilke krav som skal settes til modulens funksjoner og oppgaver.
2. Designfasen. Utviklingen av modulen
3. Implementering og testfasen. Koden integreres med resten av systemet, testes for feil og andre bugs som kan forårsake ustabilitet. Etter implementeringen og testingen vurderes systemet og modulen som en helhet. Her vurderer man om programmet fungerer som planlagt, om man skal beholde, endre eller forkaste modulen.
4. Vurderingsfasen. Gruppen kommer til å ha hyppige møter med oppdragsgiver, og derfor få tilbakemelding på implementert funksjonalitet underveis i prosjektet.

Fasene gjentas til programmet er ferdigutviklet

Scrum legger til rette for utvikling i små grupper, og planlegger daglige statusmøter for at medlemmene av gruppen skal få en bedre oversikt over arbeidet som har eller blir gjort. Gruppen vil bruke deler av Scrummodellen i prosjektet. Et rapporteringsmøte skal skje hver uke, og det vil være et fysisk møte hver måned. Disse tidsperiodene vil fungere som Scrum sprints, og man vil ikke kunne komme med ønsker om endringer og forslag i løpet av denne perioden.

Gruppen planlegger å ha et daglig møte i likhet med daily Scrum, som ikke skal vare lengre enn 10 minutter. Her fortelles hva som er gjort siden dagen før, dagens plan og eventuelle problemstillinger og utfordringer.

Ettersom medlemmene på gruppen har forskjellige timeplaner vil ikke dette møtet ha et fast tidspunkt, og gruppemedlemmene trenger ikke engang være fysisk til stede. Ved hjelp av elektroniske verktøy kan deltakerne møtes fysisk, men også jobbe adskilt uten at dette skal påvirke planleggingen. En mer detaljert beskrivelse av disse verktøyene kommer under avsnittet for verktøy. Det benyttes f.eks. en elektronisk løsning for å holde orden på oppgaver som må utføres til enhver tid.

Inkrementell utvikling vil være mer fordelaktig som utviklingsmodell. Grunnen til dette er at Snortmanager baserer seg på at det blir implementert funksjonalitet inkrementelt og stykkvis, inkrementell modell tillater nettop denne tilnærmingen på utvikling.

Sammensettingen av inkrementell modell og Scrum-modellen er den beste måten å utføre utviklingen på, da inkrementell modell passer best for selve utviklingen av Snortmanager, og Scrum-modellen gir mulighet for å være en liten gruppe.

1.9 Snort

Systemet Snort [2] er et misbruksbasert (misuse-based) Intrusion Detection/Prevention System (IDS/IPS), utviklet i åpen kildekode av Martin Roesch i 1998. Snort utviklet seg til å bli et av dagens ledende misuse-baserte IDS/IPS. I dag blir Snort distribuert av selskapet Sourcefire, og benyttes av over fire hundre tusen brukere verden over.

Som andre IDS/IPS baserer Snort seg på forhåndssatte regler plassert på sensorer på nettverkene. Snort bruker disse reglene for å analysere datatrafikken som strømmer igjennom nettverket. Snort utfører i hovedsak protokoll og innholdsanalyse. Disse analysene er skapt for å oppdage ondsinnede angrep, blant annet Buffer Overflow og tjenestenektangrep. Ved deteksjon av slike angrep vil en eventuell operasjonssentral bli gjort oppmerksomme på et angrep eller en abnormalitet på nettverket. Snort har tre hovedfunksjoner: en pakkesniffer som presenterer pakker på skjerm, en loggfører av pakker som går over nettverket, samt et Intrusion Prevention System.

1.10 Litt om intrusion detection og prevention systemer generelt

IDS - Intusion Detection System

Et IDS-system kan ses på som en innbruddsalarm, dens oppgave er å detektere unormale aktiviteter innenfor nettverket. Et IDS er designet for å forhindre angrep fra utsiden, som for eksempel virus og DDoS angrep. IDS-systemets analyser baserer seg på signaturer for nettrafikk og interne aktiviteter satt av administrator. Disse reglene blir skreddersydd for kundens behov, og implementert på sensorer. Systemet jobber deretter ut fra disse reglene og sammenligner dem med aktivitetene som skjer for øyeblikket. Den analyserer pakkestrømmen gjennom nettverket, og søker gjerne etter bl.a. TCP forespørsler og SYN pakker. Hvis avvik forekommer, vil systemet gi fra seg en alarm etter ønske. Det kan være en stille alarm, der ansvarlig får beskjed via e-mail eller av annen kilde, vakthavende alarmeres i form av lyd og lys i operasjonsrommet. I Intrusion Detection systemet må truslene ofte fjernes manuelt av administrator.

IPS – Intrusion Prevention System

Intrusion Prevention System er et aktivt tiltak, i motsetning til IDS som er er et passivt tiltak. IPS ses på som en videreføring av IDS. IPS gjør mye av det samme som IDS, men i motsetning til IDS stopper IPS angrep i sanntid. IPS kan ses som en oppgradering og forbedring av brannmurer. I likhet med IDS har IPS forskjellige typer som baserer seg på forskjellige deler av nettverket. Et IPS vil hovedsakelig fokusere på følgende angrep; DoS angrep, bakdører, malware, trojanere og annen ondsinnet kode.

Det amerikanske instituttet National Institute of Standards and Technology har utarbeidet følgende argumenter:

[2] http://www.snort.org/snort

- For å skape frykt hos eventuelle angripere og økt sjangse for å bli tatt.
- For å fange opp angrep og andre sikkerhetsavvik som ellers ikke ville blitt fanget opp av andre sikkerhetstiltak.
- For å detektere og håndtere startfasen av angrepet.
- For å kartlegge de allerede eksisterende truslene.
- Kvalitetskontroll for sikkerhetsdesignet og administrasjonen.
- For å forsyne verdifull informasjon om instrukser som skal implementeres, og dermed øke kvaliteten på diagnostikken og berging av data.

Åpen kildekode

Åpen kildekode kjennetegnes av at kildekoden for et program eller system blir gjort tilgjengelig for offentligheten. Kopiering og endring gjøres etter ønske. For å beholde eierskapet på den utgitte koden, er utvikleren nødt til å lisensiere koden.

1.10.1 Snort signaturer

Snort benytter seg av regler for å oppdage mistenkelig, og skadelig aktivitet på nettverkene det analyserer. Dette er korte, enkle strenger med tekst, som beskriver hvordan Snort skal oppdage, reagere og behandle datatrafikk. Gjennom rapporten vil Snort regler omtales som signaturer, for å unngå forvirring med andre konsepter som benytter seg av konseptet regler.

De fleste Snort signaturer er kun en enkel tekstlinje, men den kan også være fler. Den inneholder to logiske seksjoner:

Header: Inneholder informasjon om regelens handling, hvilken protokoll som skal analyseres, kilde, destinasjon og porter for datatrafikken.

Options: Inneholder beskjeder som skal sendes når sensoren oppdager skadelig trafikk. Det vil også finnes informasjon om den skadelige aktivieten, og hvilke deler av datapakken som inspiseres.

Snortmanager leser hele regelen, men forholder seg kun til rule options. Spesielt to alternativer er viktig for Snortmanager, og resten av denne rapporten:

SID: Unik identifikator for Snort signaturer.

Rev: Unik identifikator av en Snort signaturs revisjon.

Ved hjelp av disse to alternativene kan én Snort signatur identifiseres i systemet. Signaturene kan oppdateres jevnlig, men regelens SID vil ikke forrandres. Man vil derfor finne flere situasjoner hvor en SID kan ha flere revisjoner.

1.11 Verktøy

For dette prosjektet har gruppen benyttet seg av flere verktøy, for å optimalisere planleggingen, og arbeidet med bacheloroppgaven.

1.11.1 Versjonskontrollsystem

For å ha kontroll på kildekode og andre dokumenter i prosjektet, var det nødvendig med et stabilt, effektivt, oversiktlig og letthåndterlig versjonskontrollsystem, Det var også et krav om

åpen kildekode.

Gruppen vurderte flere forskjellige teknologier og måter å løse dette på. En av løsningene var Subversion(SVN), et versjonskontrollsystem som er brukt blant utviklere på større utviklingsprosjekter. Subversion ble valgt bort som alternativ på grunn av manglende funksjonalitet ved sammenfletting av filer.

Gruppen gikk for Git. I likhet med SVN er også et versjonskontrollsystem, men i motsetning til SVN er dette et distribuert system, som vil si at istedenfor alle til én konsept som SVN, er Git et alle til alle-konsept. Alle data og endringer hos brukerene blir lagret lokalt på pc-en, brukeren kan føre flere revisjoner og jobbe på disse, uten at det vil påvirke Git-serveren eller andre brukeres kode.

Ved å ta i bruk Git vil gruppen få bedre kontroll på egen utvikling, det finnes alltid flere backup versjoner og det blir lettere å føre revisjonskontroll.

I Git brukes enkle kommandoer som git addfor å spesifisere hvilke filer som skal lagres i den lokale revisjonen, og git commit -mfor å lagre de spesifiserte filene, -m kommandoen gir brukeren mulighet til å skrive kort om hva oppdateringen omfatter. For handlinger gjort mot den eksterne git serveren bruker man git pushfor å lagre sin lokale revisjon på ekstern server, git pullfor nedlasting av oppdateringer fra server. Når man så laster ned fra server, har det ofte skjedd endringer i de samme dokumentene man selv har gjort forandringer i. Man vil da få beskjed fra revisjonskontrolleren om at det har blitt foretatt endringer, og man vil få muligheten til å sammenslå dokumentene .Det vil stå klart i teksten hvor sammenslåingen er blitt gjort.

1.11.2 Oversikt over gjøremål

Det var behov for et verktøy for å kunne holde tidsfrister, og ha kontroll på at oppgaver ble utført. Verktøyet måtte også være tilgjengelig for begge gruppemedlemmene, og kunne brukes uavhengig av plattform.

Trello [3] er et slikt system, med mulighet til å opprette flere tavler". Innenfor disse tavlene har du tre hovedmenyer, pending", doing", og done". Her opprettes poster med gjøremål under tiktige kategori. Gjøremålene kan også ha mer detaljerte underposter, og kan tilegnes brukere som deltar på samme prosjekt. Poster starter oftest i Pending"og flyttes når status endres.

1.11.3 reStructuredText

Rapporten og vedlegg skal skrives i det enkle formateringsspråket reStructuredText. Alt arbeidet kan lettere skrives i klartekst, og uten formatering. På denne måten kan medlemmene fokusere på innholdet, for så å prioritere formatering og utseende i slutten av prosjektet. Utveksling av arbeid med Git er også enklere når dokumenter er skrevet i åpen klartekst, og ikke binære formater som Microsoft Word eller lignende programmer.

reStructuredText fungerer med å definere innhold i dokumentet med en egenprodusert syntax. Hvilken som helst teksteditor kan brukes, så lenge den støtter UTF-8. Overskrifter, seksjoner, lister og annet innhold defineres av forfatterne med små tekstkommandoer. Senere vil dokumentet leses inn og konverteres til et annet format, ved hjelp av docutils [4]. Dette rammeverket kan konvertere rapporten til de fleste populære filformater.

[3] https://trello.com/
[4] http://docutils.sourceforge.net/

Under fasen med rapportskrivning vil dokumentet konverteres til en lett oversiktlig PDF fil, ved hjelp av **rst2pdf** [5] som bygger på docutils. Dette lager enkle og oversiktlige dokumenter som vil overleveres veileder og evt. oppdragsgiver. Når rapporten skal leveres vil dokumentet først bli konvertert til LaTeX[6], for videre å bli konvertert til PDF. Dette er valgt fordi LaTeXgir mer kontroll over utseende og muligheten til å legge med vedlegg i dokumentet.

1.12 Konvensjoner brukt i denne rapporten.

Filer, o.l: Utheves med egen `font`. Eksempel: `Snortmanager.py`
Mapper, klasser og instanser: Utheves med samme font og *kursiv tekst*. Eksempler:

- *var/*
- *MetaData*
- *engine*

Metodenavn: Utheves med **fet tekst**. Eksempel: **index()**.
Kodeeksempler: Separeres fra øvrig tekst med tekstblokk og egen font. Eksempel:

```
    """Eksempel paa en blokk med kildekode."""
```

[5] http://code.google.com/p/rst2pdf/
[6] http://www.latex-project.org/

2 Kravspesifikasjon

Dette kapitlet inneholder en komplett beskrivelse av de funksjonelle og tekniske krav for Snortmanager. Hensikten er å forbedre kvaliteten av leveransen, og forsikre riktig funksjonalitet og teknisk design av løsningen.

2.1 Generell beskrivelse

Snortmanager skal bli et verktøy som håndterer administrasjonen av Snortsensorer. Det skal håndtere nedlastning og oppdatering av regler fra eksterne kilder, produksjon av konfigurasjonsfiler for sensorer, logisk organisering av sensorer, rapportering av signaturbruk og historikk over sensors livsvarighet.

Det skal være mulig for sluttbrukerene å definere eksterne kilder som snort.org. [1] eller øvrige sider som tilbyr Snortsignaturer. Disse skal deretter behandles av systemet for å kontrollere revisjon, innhold, om signatur skal deaktiveres sentralt, eller om ny signatur skal opprettes i databasen.

Hver sensor blir lagt inn i systemet og gruppert etter lokasjon. For hver enkelt sensor skal det deretter produseres en konfigurasjonsfil basert på unike policyer. En policy bygges opp av brukerene med gjenbrukbare komponenter med tekstregler.

All øvrig funksjonalitet skal være tilrettelagt gjennom et webbasert grensesnitt. Her skal det være mulighet for å begrense tilgangen med normal deny-all-prinsipp. Dette betyr at bare autoriserte, forhåndsbestemte brukere skal ha tilgang til systemet.

2.1.1 Beskrivelse av bruker

Snortmanager skal brukes av personell ved en IT-avdeling eller øvrige personer med god til meget god forståelse av IT. Brukerene har flere års erfaring og god teknisk forståelse av Snorts funksjonalitet og virkemåte.

2.1.2 Antagelser

Det antas at Systemet skal kjøres på et eget internt nettverk, sikret fra globalt eksponerte nettverk som Internett. For hvilke konsekvenser dette får for systemet, se Begrensninger.

2.1.3 Begrensninger

Ettersom systemet skal brukes på et internt sikret nett, blir det designet for å håndtere liten til middels last av brukere. Snortmanager skal kun benyttes av brukere ved endringer, som anses å være 1-3 brukere samtidig. Ellers vil det kun utføres automatiserte oppgaver. Sikkerheten i Snortmanager reflekterer også eksponeringen, og vil bare testes for potensielle sikkerhetsfeil. SQL Injection, og XSS er eksempler på dette.

[1] http://www.snort.org - Offisiell webside for Snort

2.1.4 Krav til løsningen

Løsningen skal være webbasert med mulighet til å utføre utvalgte jobber fra ett kommandolinjegrensesnitt. Hensikten er å forenkle og forene administrasjonen av regelsett for oppdragsgiver og deres kunder. Det skal utføres automatiske bakgrunnsjobber som laster ned og oppdaterer regelsett fra forhåndsdefinerte, eksterne tilbydere. Det skal daglig produseres nye konfigurasjonsfiler for hver enkelt sensor, som også skal plasseres på den respektive sensoren.

2.1.5 Bruksmiljø

Snortmanager skal utvikles i programmeringsspråket Python, etter ønske fra oppdragsgiver. Den ferdige løsningen skal settes i produksjon i et Linux-miljø, og derfor hovedsakelig testes mot denne plattformen. Samtidig vil koden skrives slik at den lett kan implementeres i andre plattformer.

Brukerene får tilgang til Snortmanager gjennom et webgrensesnitt som skal kunne kjøres i de fleste kjente nettlesere. Sluttbrukeren skal kunne gjøre alle operasjoner i webgrensesnittet. Det skal også være mulig å utføre bakgrunnsjobber fra kommandolinje, slik at jobber kan utføres av enn annen serveren den webapplikasjonen kjører på, startes manuelt eller av eksterne skript.

2.2 Funksjonelle krav

Funksjonelle krav omfatter hvilke funksjoner og tjenester Snortmanager tilbyr. Det beskriver hvordan hvilke data en bruker vil oppgi, hvordan de behandles og hva som er tillatt av løsningen. Kravene vil også beskrive hva Snortmanager ikke skal gjøre.

2.2.1 Liste over aktører

Følgende aktører vil bruke funksjonalitet i Snortmanager:

Sluttbruker: Brukere som logger seg inn til Snortmanager gjennom webgrensesnitt eller benytter kommandolinjeverktøy.

System: Triggere eller bakgrunnsprosesser som starter en bakgrunnsjobb.

2.2.2 Overordnet use casebeskrivelser

Use case beskrivelser viser hvordan Snortmanager skal utføre funksjoner og tjenester. Hensikten er å dokumentere handlinger utført av forskjellige aktører, i forskjellige omgivelser, og hvilket resultat handlingen får.

Use case: Administrere sensordata.

Aktør: Sluttbruker.

Trigger: Bruker ønsker å legge til, endre eller fjerne en sensor.

Mål: Forsikre seg om at sensordata er tilgjengelig, oppdatert og konsistent.

Beskrivelse: Siden systemet skal produsere konfigurasjonsfiler for sensorer, må disse registreres i systemet. Gjennom webgrensesnittet skal sluttbrukeren få en liste over alle sensorer registrert i systemet gruppert, med lokasjon. Fra dette grensesnittet skal det være mulig å legge til, og endre eksisterende sensor, og lokasjon. Brukeren vil bli spurt om nødvending informasjon som navn på sensor, lokasjon og IP-adresse, samt frivillig informasjon som beskrivelse. Dette blir lagret til databasen. En sensor blir identifisert med IP-adresse, som skal være unik i databasen. Den vil også få en ID, da det vil gjøre det enklere å endre IP-adressen og likevel vite hvilken sensor det gjelder. Grensesnittet for å opprette nye sensorer og endre

eksisterende sensorer bør være likt. Det skal ikke være mulig å slette en sensor fra databasen, den skal heller settes inaktiv. Dette forsikrer brukeren om at historikk og logg til enhver tid er korrekt og konsistent.

Use case: Administrere lokasjonsdata for sensorer.

Aktør: Sluttbruker.

Trigger: Det er nødvendig å opprette, endre en eksisterende eller deaktivere en lokasjon.

Mål: Ha en oppdatert oversikt over lokasjoner for enklere administrasjon av sensorer.

Beskrivelse: Ettersom alle sensorer skal grupperes etter lokasjoner, må disse dataene være korrekte. Man skal gjennom webgrensesnittet kunne opprette lokasjoner, med nødvendig data om lokasjonen. Brukeren vil også få en liste over alle lokasjoner i systemet. Her skal man kunne hente ut informasjon om lokasjonen og hvilke sensorer som ligger under den. Ved endringer endres kun lokasjonsobjektet, men ved deaktivering skal brukeren gis mulighet til å koble sensorene mot en ny lokasjon. Skjer ikke dette, blir alle sensorere deaktivert.

Use case: Administrere kilder for nedlastning av regler.

Aktør: Sluttbruker.

Trigger: Aktør ønsker å legge til, endre eller deaktivere eksterne kilder for nedlastning av Snortsignaturer.

Mål: Ha en oppdatert liste over kilder, slik at man har oppdaterte signaturer i systemet.

Beskrivelse: Gjennom webgrensesnittet skal man kunne legge til nye kilder for nedlastning av Snortsignaturer. Brukeren vil få en liste over alle **aktiverte** kilder, hvor man også kan legge til nye kilder. Da blir man presentert med et grensesnitt hvor man oppgir informasjon om kilden. Man skal også kunne endre adressen i en kilde, eller eventuelt deaktivere den, slik at den ikke blir brukt videre i systemet.

Use case: Opprette policyer for sensorer.

Aktør: Sluttbruker.

Trigger: Aktør ønsker å opprette en policy for en sensor.

Mål: Policyer skal brukes til å lage unike konfigurasjoner for de enkelte sensorene.

Beskrivelse: En policy skal opprettes for sensorene og brukes når nye konfigurasjonsfiler produseres. Her skal det gis mulighet for å aktivere/deaktivere enkeltregler, og filer i tillegg til å skrive om regler ved behov. Det skal også være mulig å legge innhold i toppen og bunnen av konfigurasjonsfilen.

Use case: Laste ned oppdaterte regler fra ekstern kilde

Aktør:
- System.
- Sluttbruker.

Trigger: Nye regler skal lastes ned fra eksterne kilder.

Mål: Ha en oppdatert database med Snortsignaturer.

Beskrivelse: Snortmanager skal kunne laste ned oppdaterte Snortsignaturer fra eksterne kilder. En ekstern kilde kan være på offentlige og private nett. Deretter skal alle signaturer i alle filer gjennomgås og behandles. Hvis det skjer innholds-, eller revisjonsendringer i signaturen, skal enkelte signaturer deaktiveres og nye legges inn. Hvis en aktiv signatur er aktivert i systemet, men deaktivert fra ekstern kilde, skal den deaktiveres. Etter nedlastning skal varslingsgruppe informeres om endringer, med e-post.

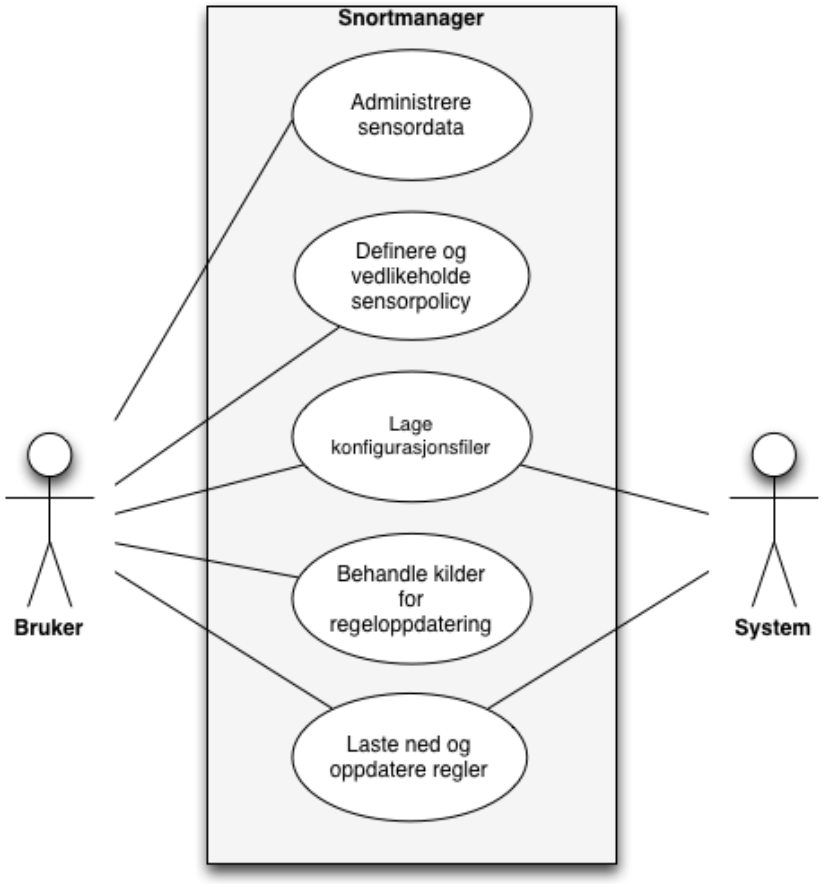
Use case: Produsere konfigurasjonsfiler for alle aktive sensorer
Aktør:
- System.
- Sluttbruker.

Trigger: Det skal produseres nye konfigurasjonsfiler for hver enkelt sensor registrert i Snortmanager.
Mål: Lage konfigurasjonsfiler som skal kunne distribueres til sensorer.
Beskrivelse: En konfigurasjonsfil skal produseres automatisk for hver enkelt sensor i systemet. Disse filene skal baseres på innholdet i policyen den enkelte sensoren er koblet sammen med.
Use case: Låse Snortmanager for endringer av data.
Aktør: System.
Trigger: Snortmanager skal utføre en bakgrunnsjobb som er avhengig av konsistente data.
Mål: Forsikre at ikke endringer skjer på data som benyttes av bakgrunnsjobber.
Beskrivelse: Systemet vil ha flere langvarige bakgrunnsjobber, som produksjon av konfigurasjonsfiler og nedlastning av regler. Det er derfor viktig at øvrige deler av systemet ikke endrer data brukt i disse jobbene. Det skal derfor være mulig å låse systemet for redigering gjennom en normal programvarebasert ressurslås.
Use case: Søke på regler for informasjon.
Aktør: Sluttbruker.
Trigger: Bruker ønsker å finne informasjon om regler.
Mål: Raskt finne informasjon om regler i systemet.
Beskrivelse: En bruker skal kunne søke frem informasjon om lagrede regler i databasen. Ved hjelp av et grensesnitt skal det kunne søkes på SID, for deretter å finne frem alle revisjoner av signaturen, deres innhold og hvorvidt de er aktive.
Use case: Rapportering av signaturbruk, policybruk, lokasjon eller lignende.
Aktør: Sluttbruker.
Trigger: Bruker ønsker en oppdatert rapport.
Mål: Presentere informasjon eller få en rapport over bruk og statistikk.
Beskrivelse: Det skal gjennom flere grensesnitt kunne hentes ut rapporter og statistikk over bruk av ressurser i Snortmanager, som signaturer, policy, lokasjon osv.
Use case: Logging av brukerhandlinger og operasjoner.
Aktør: Bruker.
Trigger: En hendelse utføres i systemet av bruker, som må logges.
Mål: Ha historikk over alle handlinger utført på systemet av bruker.
Beskrivelse: Loggføring av brukeraktivitet er nødvendig for å kunne ha en full historikk av all interaksjon mot systemet og databasen, gjort av bruker. Aktivtet fra hver modul blir logget til databasen.
Use case: Logging av hendelser i systemet.
Aktør: System.
Trigger: En hendelse utført av systemet, som må logges.
Mål: Ha historikk over alle hendelser forårsaket av systemet.
Beskrivelse: For å kunne ha en fullstendig historikk av alle hendelser forårsaket av systemet,

skal hver modul sørge for å logge exeptions og warnings til fil.

2.2.3 Extended use case beskrivelser

Dette er en mer utfyllende use case beskrivelse, og scenarioene blir utdypet med mer informasjon, hendelsesforløp, og sekvensdiagram. Rapporten inneholder kun to extended use case beskrivelser, men disse viser godt hvordan løsniningen skal brukes.



Figur 2: Snortmanagers use case diagram, som illustrerer hvilke funksjoner aktører har tilgang til.

:Use case: Administrere sensordata.
Aktør: Sluttbruker.
Hensikt: Registere, endre eller deaktivere en sensors data.
Trigger: Aktør ønsker å legge til, endre eller deaktivere sensor.
Beskrivelse: Systemet skal produsere konfigurasjonsfiler pr. sensor, og trenger derfor å ha informasjon om hver enkelt sensor lagret i databasen. Brukeren må derfor kunne registrere nye sensorer som skal behandles, endre eksisterende sensorer eller deaktivere sensorer som ikke skal tas ut av bruk.
Type: Essensiell.
Forhåndsbetingelse: Ingen.

Sluttbetingelse: Ingen.

Tabell 1: Registrere ny sensor

Aktør	**Systemrespons**
1. Brukeren ønsker å registrere en ny sensor.	
	2. Legger til ny sensor i systemet med nødvendige parametere
	3. Loggføre handling til sentral logg.
4. Brukeren får tilbakemelding om at forespørselen er utført.	

Handlingsforløp:

Alternativ handling: Steg 3: Hvis den registrerte sensoren allerede eksisterer, skal det vises feilmelding og gå til steg 1. Hvis brukere oppgir feilaktig informasjon, vis feilmelding og gå til steg 1.

:Use case: Definere og vedlikeholde policy.

Aktør: Sluttbruker.

Hensikt: Definere nye og vedlikeholde eksisterende policy for én eller flere sensorer.

Trigger: Aktør ønsker å opprette ny eller endre en policy.

Beskrivelse: Når det skal produseres en konfigurasjonsfil, skal den baseres på regler i en policy. Denne vil bestå av regler som skal brukes til å velge innholdet i filen, i tillegg til å manupulere det. Signaturene skal skrives som klartekst av sluttbrukeren, og vil inneholde informasjon om Snortsignaturer og sensordirektiver.

Type: Essensiell

Forhåndsbetingelse: Ingen

Sluttbetingelse: Ingen

Tabell 2: Definere og vedlikeholde policy

Aktør	**Systemrespons**
1. Bruker oppretter en tom policy med nødvendig informasjon.	
	2. Systemet lagrer informasjon om policy, og gir muligheten til å lage regler.
3. Bruker definerer hvilke regler og Snortsignaturer den ønsker å inkludere	
	4. Systemet lagrer objektene, og returnerer etter vellykket operasjon

Handlingsforløp:

Alternativ handling: Ingen.

2.3 Ikke-funksjonelle krav

Ikke-funksjonelle eller operasjonelle krav, beskriver hvordan Snortmanager skal implementere de funksjonalene kravene definert tidligere.

2.3.1 Sikkerhet

Snortmanager skal undersøkes for alvorlige sikkerhetshull og øvrige feil som kan føre til redusert stabilitet av systemet eller nødvendig infrastruktur.

2.3.2 Krav for vedlikehold av kode

Kode skal være godt dokumentert og lesbar, slik at andre utviklere senere kan vedlikeholde og videreutvikle systemet.

Kode skal i høyeste grad forholde seg til etablerte standarder for programmeringsspråket.

2.3.3 Portabilitet

Snortmanager skal i utgangspunktet fungere på Linuxbaserte operativsystemer og kvalitetssikring skal reflektere dette. Produktet skal utover dette programmeres med tanke på portabilitet mellom operativsystemer. Det bør ikke være mye arbeid å overføre løsningen fra et OS til et annet.

2.3.4 Miljøpåvirkning

Snortmanager vil kjøre på allerede eksisterende infrastruktur, og vil ikke påvirke natur og miljø på noen måte. Samtidig skal Snortmanager utvikles slik at det ikke bruker unødvendige ressurser, for å unngå ringvirkninger utenfor driftsmiljøet.

2.4 Krav til enkeltkomponenter

Dette er kravene som beskriver hvordan de individuelle komponentene av Snortmanager skal fungere. Disse kravene er mer spesifikke enn funksjonelle krav.

2.4.1 Rapportering

Det skal være mulig for sluttbrukere å hente ut portabilitet rapporter om følgende:

Regler: Det skal være mulig å hente ut informasjon om enkelte regler i systemet. Informasjonen er signaturinnhold, revisjoner, aktiveringsstatus, kilde og historikk.

Signaturfiler: Siden regler blir lastet ned i filer, skal det være mulig å hente ut informasjon om alle signatur som har kommet fra en enkelt fil.

Policy: Man skal kunne hente ut informasjon om en policy, hvilke elementer som er inkludert i denne og hvilke sensorer som benytter den.

2.5 Policy

En policy skal brukes under produksjonen av konfigurasjonsfiler for sensorene som Snortmanager administrerer. Disse policyene skal bygges opp av et eller flere sett med regler som oppgis av bruker gjennom grensesnittet. En sensor skal forholde seg til en policy i sin helhet, men deler av policyene skal kunne brukes i flere andre policyer.

Policyene skal kunne:

- Aktivere eller deaktivere filer som inneholder regler.
- Aktivere eller deaktivere enkelte regler basert på SID.
- Aktivere en signatur basert på regular expressions (regex).
- Skrive om deler av en signatur.

Syntaksen til reglene brukt av policyene skal i størst mulig grad være lik fra tidligere utgave av Snortmanager.

Handlingen av policyregelen vil si hvilken effekt den skal ha på konfigurasjonsfilen. Det kan være å deaktivere eller aktivere en signatur. Med dette menes hvorvidt noe skal skrives til konfigurasjonsfilen eller ikke. Deretter defineres kilden, som identifiserer hvor systemet skal hente det den skal handle ut fra. Til slutt kommer identifikatoren, som identifiserer den unike signaturen den skal bruke. I tilfeller hvor det finnes flere revisjoner av samme signatur, benyttes den høyeste revisjonen.

Tabell 3: Syntaks for policyproduksjon

Handling	**Kilde**	**Identifikator**	**Beskrivelse**
enable/disable	file	<filid>	Aktiverer eller deaktiverer standard versjon av en fil med regler. Den behandler bare regler aktivert i filen som er definert.
	sid	<sid>	Aktiverer eller deaktiverer en enkelt signatur basert på SID, uavhengig av om den er deaktivert i databasen.
	regex		Behandler regler basert på innhold i en regular expression. Denne behandler bare aktiverte regler i databasen.
rewrite	system	<erstatningstekst> <ny tekst>	Skrive om en signaturs innhold basert på erstatningstekst. Om systemet finner erstatningstekst blir den erstattet med ny tekst.

3 Design

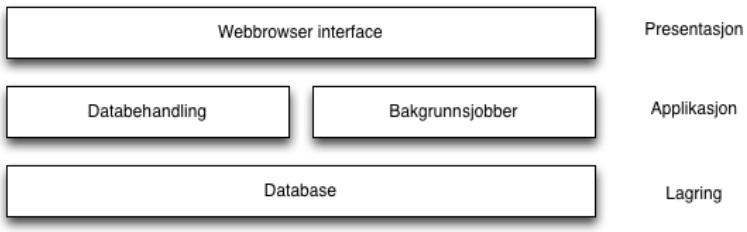
Hensikten med dette kapittelet er å beskrive designet og arkitekturen til Snortmanger i henhold til Kravspesifikasjonen. Snortmanager vil være et produkt basert på åpen kildekode, og være tilgjenglig over Internett.

Spesifikasjonen vil ikke være komplett, men reflekterer det arbeidet som skal gjøres får å lage sluttproduktet. Alle beskrivelser og figurer illustrerer den underliggende funksjonaliteten. Det visuelle designet på sluttproduktet vil utvikles over tid, basert på funksjonalitet og tilbakemeldinger fra sluttbrukere.

3.1 Beskrivelse av systemarkitektur

Snortmanager er et komplisert system, som reflekteres i arkitekturen som skal velges. Gruppen mener at løsningen skal kjøres fra én enkelt prosess, og at systemfiler skal ligge på ett enkelt sted på serveren. Prosessen skal være ansvarlig for brukeropplevelsen, lagring av data, loggføring og bakgrunnsjobber. Ettersom de forskjellige delene av løsningen har forskjellige krav, ble det valgt å bruke to modeller for å beskrive systemarkitekturen.

Lagmodellen ble valgt for å beskrive hele løsningen, mens Model-View-Controller (MVC)-arkitektur ble valgt for å beskrive selve websystemet. Bakgrunnsjobbene er en stor del av løsningen, men er ikke avhengig av grensesnitt eller presentasjon. Samtidig skal webapplikasjonen både presentere, og behandle data.



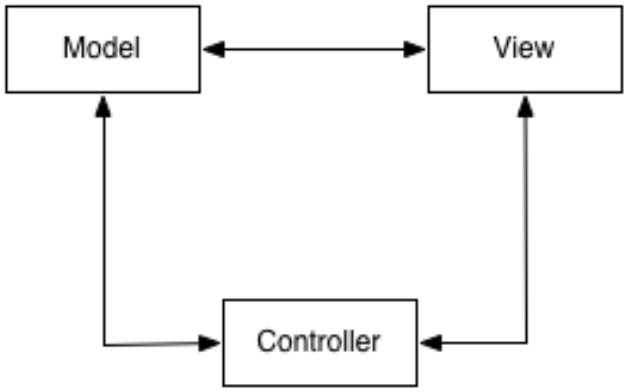
Figur 3: Hele Snortmanager representert med en lag modell arkitektur.

Sluttbruker blir presentert for et webgrensesnitt, hvor den skal kunne behandle data, starte jobber eller lese rapporter. Webgrensesnittet blir behandlet av applikasjonen, som er en webserver. I dette laget skal systemet kunne hente informasjon til sluttbrukeren og gjøre bakgrunnsjobber. Til slutt er databaselaget, som står for lagring og oppbevaring av data for hele Snortmanager.

Webserveren vil stå for overføring av data, presentasjon og utførelser av bakgrunnsjobber. Sistnevnte vil gjøres av samme prosess, men seperat fra grensesnittet. Systemet klarer da å kjøre jobber parallelt med at brukere utfører andre oppgaver i grensesnittet.

Webserveren står også for lagring av data til en database. Denne databasen skal inneholde alle data Snortmanager trenger som Snortsignaturer, data om sensorer, policykjeder, hendelseslogg

o.l.



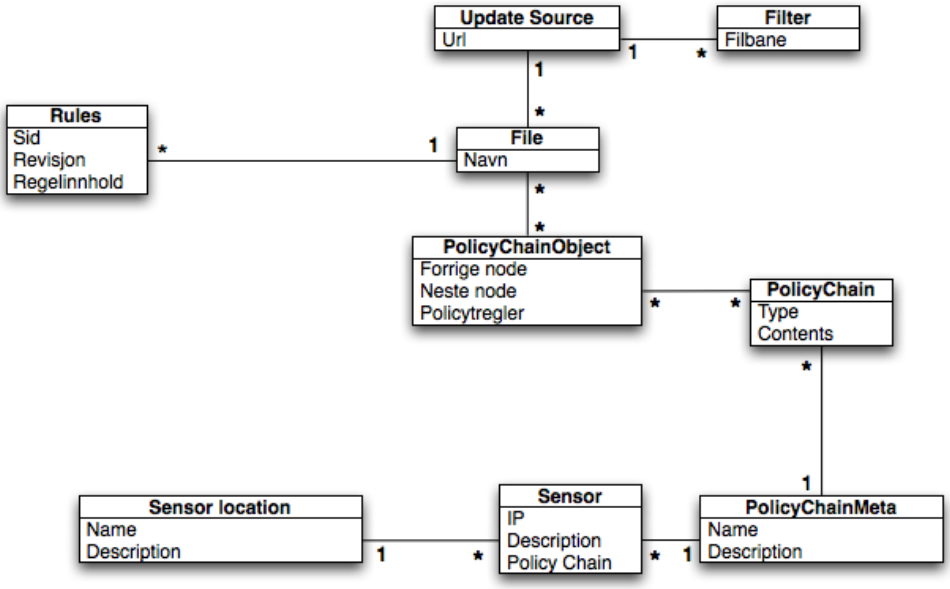
Figur 4: Webapplikasjonen designet med MVC modellen.

Arkitekturen for selve webapplikasjonen vil baseres på Model-view-controller, hvor utseende, brukerlogikk og data skilles i tre seperate deler.

Model er ansvarlig for den logiske delen av applikasjonen, som henting av data fra database og prosessering. Et view vil være ansvarlig for å presentere data til brukeren. Ved hjelp av HTML og CSS vil den behandle data og presentere brukeren for innhold.

Controller oversetter og videresender data fra view og fra model. Den er ansvarlig for å behandle data, fra både systemet og brukeren.

Ved å bruke MVC-modellen har Snortmanager separert presentasjon, databehandling og dataoppbevaring.



Figur 5: Snortmanagers overordnede klassediagram.

Gjennom analyser av oppdragsgivers behov og den gamle versjonen av Snortmanager kom det frem at de tre viktigste delene av systemet var policy, regler og signaturfiler. Det konseptuelle klassediagram er orientert rundt signaturfiler. Ettersom det i størst mulig grad skal aktiveres filer med underliggende regler, vil det konseptuelle diagrammet fokuseres rundt filobjektene.

En oppdateringskilde vil produsere et filobjekt for hver enkelt signaturfil som lastes ned under en oppdatering. En oppdateringskilde kan laste ned arkiver med flere filer, som gjør det viktig å gruppere dem deretter. Hver enkelt fil inneholder én til mange signaturer, så det blir opp til signaturfilobjektet å passe på alle reglene som ligger i systemet.

Hver enkelt signatur er ansvarlig for eget innholdet og visse metadata, som unik identifikator og versjon.

Policykjeder er konstruert som noder i en kjede som inneholder data om sine eksisterene objekter. De skal inneholde en link til det forrige elementet i kjeden, en link til det neste elementet i kjeden og hvilket policyobjekt denne kjedelinken inneholder.

Policyinnhold er en type som definerer om objektet skal puttes i toppen eller bunnen av filen systemet produserer. Deretter kommer innholdet i dette objektet i form av et tekstfelt, som inneholder reglene som skal brukes når filer blir produsert.

3.2 Design av moduler

Denne delen av designdokumentet beskriver konsepter og funksjonalitet som ligger i Snortmanager, samt hvilke kriterier, valg og tanker som ble gjort under utviklingen av systemdesignet.

3.2.1 Bakgrunnsjobber

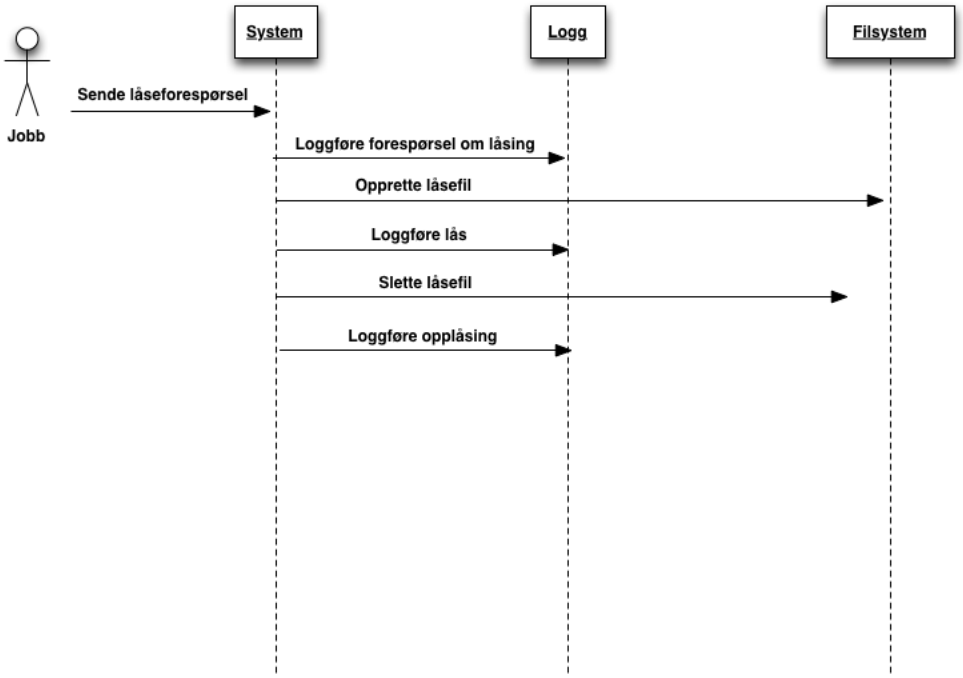
Snortmanager skal kunne kjøre flere oppgaver samtidig som det presenterer innhold til brukerene gjennom webgrensesnittet. Brukeren skal gjennom dette grensesnittet også ha mulighet til å starte en slik jobb, f.eks. å laste ned oppdaterte regler fra ekstern kilde, eller produsere konfigurajonsfiler for Snortsensorer.

Ettersom disse jobbene kan være både langvarige og omfattende fører dette til utfordringer med prosesseseringsressurser. Moderne webserveren kjører som en egen prosess, og oppretter en tråd for hver enkelt sesjon fra weblesere. Denne tråden er ansvarlig for alle handliger brukeren gjør, og for presentasjonen av innhold til sluttbrukeren. Hvis sluttbrukeren starter en slik jobb gjennom nettleseren, vil deres sesjon fryse mens oppgaven pågår, slik at den ikke kan gjøre andre handlinger på siden. Skulle også brukeren avslutte webleseren ved en feil, eller gå til en annen webside, vil oppgaven avsluttes for tidlig.

Bakgrunnsjobber må derfor kunne startes opp gjennom webgrensesnittet, men utføres utenfor denne tråden. Det finnes flere metoder å gjøre dette på, som å utføre oppgaven i en egen separat prosess som startes av Snortmanager. Dette kan føre til komplikasjoner med prosessynkronisering, potensielle konflikter og korrupte data hvis feil oppstår.

Snortmanager vil presentere et grensesnitt for sluttbrukeren som gir mulighet til å starte bakgrunnsjobber. Deretter vil systemet opprette en egen tråd hvor bakgrunnsjobben utføres. Sluttbrukeren kan dermed benytte seg av øvrig funksjonalitet i webgrensesnittet, lukke webleseren eller på annen måte avslutte sesjonen uten at dette avslutter bakgrunnsjobben.

3.2.2 Systemlås



Figur 6: Sekvensdiagram for systemlås.

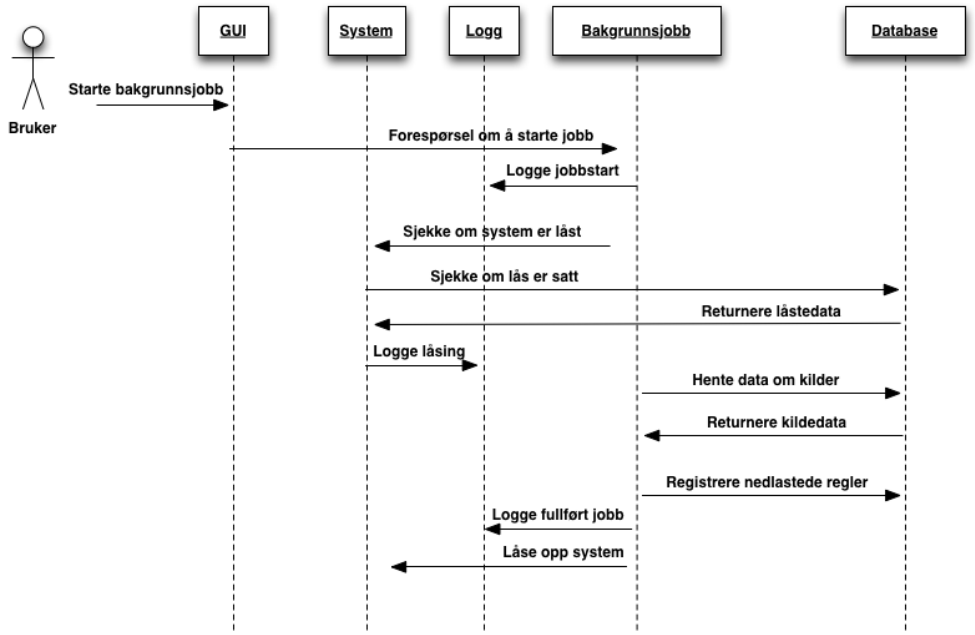
Snortmanager bruker bakgrunnsjobber for å utføre store oppgaver uten å påvirke brukeropplevelsen. For å ivareta integriteten til dataene og unngå feil, vil en systemlås benyttes. En systemlås vil begrense tilgangen til flere av Snortmanagers funksjoner, og sperre muligheten til å skrive endringer til databasen.

Når en bakgrunnsjobb starter, eller brukeren ønsker å gjøre endringer, vil det gjøres en sjekk av systemlås. Dette gjøres ved å se etter om en fil eksisterer på filsystemet. Om systemet er låst, får brukeren beskjed om dette, og operasjonen blir avbrutt. Hvis systemet ikke er låst, vil operasjonen bli utført. Hvis denne operasjonen er å starte en bakgrunnsjobb, vil Snortmanager loggføre at systemet blir låst. Deretter vil filen som indikerer at en systemlås er aktivert bli opprettet, og IDen til prosessen som starter jobben blir skrevet til den. Dette gjøres for å kontrollere at man kan spore opp hvilken prosess som opprettet filen.

Når bakgrunnsjobben er utført, skal systemlåsen fjernes, og endringer skal kunne gjøres. Dette gjøres ved at filen som først ble opprettet fjernes, og en oppføring blir skrevet til databasen for å informere om at systemet igjen er låst opp.

For å forsikre mot potensielle sikkerhets- og systemfeil, skal det legges inn logikk for å avverge race conditions. Når systemet får en forespørsel og har hentet statusen til systemet, vil den vente et kort, tilfeldig tidsintervall før den sjekker om filen eksisterer. Dette gjør at man ikke kan planlegge et tjenestenektsangrep på systemet ved å manipulere låsefilen.

3.2.3 Oppdatering av regler



Figur 7: Sekvensdiagram som viser nedlastning av regler fra eksterne kilder.

Systemet trenger jevnlig å laste ned nye og oppdaterte regler fra eksterne kilder. Disse reglene kommer i form av tekstfiler, hvor hver linje er en signatur. En oppdateringsjobb kan startes av en bruker, eller på et bestemt tidspunkt.

En jobb kjører i en egen separat tråd i prosessen. På denne måten påvirkes ikke jobben av brukerens handlinger, f.eks. at nettleseren blir lukket. Brukerene kan også benytte webapplikasjonen mens bakgrunnsjobbene pågår, men får ikke gjort noen endringer på lagrede data, siden systemet blir låst.

Selve prosessen starter med å hente alle aktiverte kilder fra databasen, for så å laste ned filen. Ettersom filer kan være både enkeltfiler og arkiv, må filtypen undersøkes og behandles deretter. Når filen er ferdig pakket ut, skal systemet traversere alle filene i mappen. Alle filer av filtypen .rules skal gjennomgås og leses inn til systemet. Det skal også være mulig å ignorere filer og mapper, slik at de ikke leses inn i systemet.

Hver enkelt fil skal leses linje for linje, og skal først undersøkes om de er en Snortsignatur. Alle regler skal leses, uavhengig av om de er deaktivert i filen. Deretter sjekkes de enkelte reglene for følgende, før de skrives til databasen:

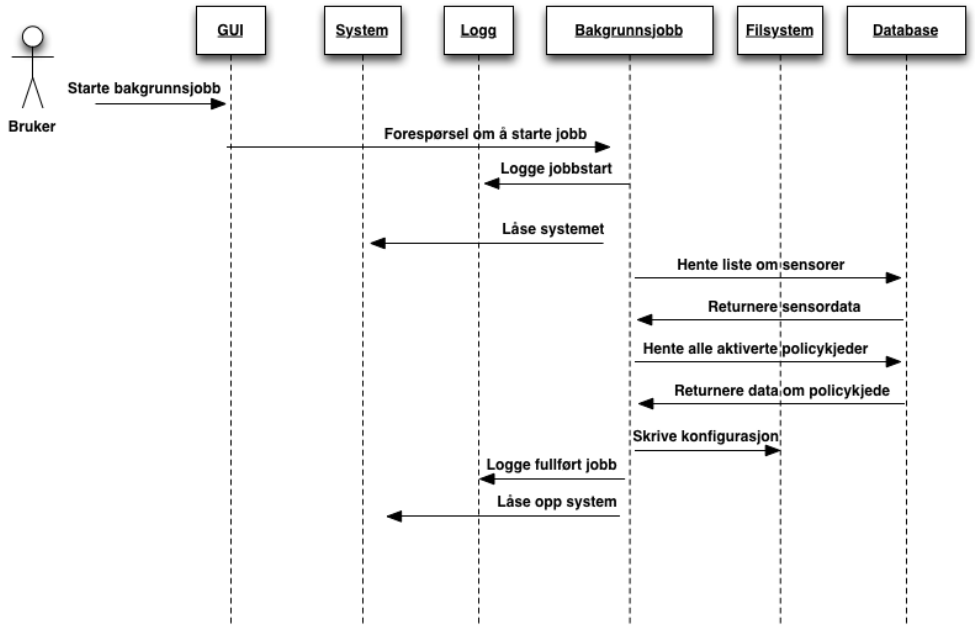
- Er signaturen deaktivert?
- Eksisterer signaturen fra før?
- Er dette en ny revisjon av signaturer?

- Har det skjedd endringer i signaturer uten at revisjonstallet har endret seg?

Om signaturen ikke finnes fra før, skal den legges til, om om nødvendig som deaktivert. Finnes signaturen fra før, med samme metadata, vil den nye signaturen forkastes. Der signaturer eksisterer, men den innleste har et høyere revisjonstall, eller endringer i signaturstrengen, skal eksisterende signatur deaktiveres, og ny signatur lagres i databasen.

Til slutt låses systemet opp, og brukeren får igjen jobbe med systemet.

Produksjon av konfigurasjonsfiler



Figur 8: Sekvensdiagram som viser produksjon av konfigurasjonsfiler.

Systemet skal produsere en tekstfil med regler og konfigurasjon, basert på regler definert i en policy. Jobben startes enten av en bruker eller automatisk av systemet. Deretter overføres ansvaret til bakgrunnsjobben, som er ansvarlig for logging til hendelseslogg og låsing av systemet. Låsing av systemet gjør at det ikke kan gjøres endringer til databasen, og forhindrer at data endrer seg under utføring av systemjobber.

For å redusere tiden det tar å produsere konfigurasjonsfiler for flere sensorer, skal systemet først lage en liste over hvilke sensorer som er registrert med en policy. Dette gjør at produksjonstiden reduseres betraktelig, siden det er stort gjennbruk av policyeelementer. Deretter traverseres policykjeden i henhold til kravene i kravspesifikasjonen.

Når policykjeden er konstruert henter systemet konfigurasjonsdata og Snortsignaturer fra databasen. Den kan også skrive om regler eller aktivere deaktiverte regler. Til slutt skrives alle dataene til én fil for hver sensor. Denne filen vil ha samme navn som IP-adressen til sensoren.

Til slutt loggfører systemet jobben og låser opp systemet, slik at brukerene kan gjøre endringer.

3.2.4 Administrasjonsverktøy for sensor

Sensorer er en vital del av systemet, og vil bli hyppig benyttet av bruker. Derfor trenger systemet et enkelt og oversiktlig administrasjonsverktøy for behandling av sensorer, som er intuitivt og funksjonelt. På Snortmanagers webgrensesnitt får bruker først en liste over sensorer som tidligere er lagt til i systemet. Samme side vil også ha to undersider, henholdsvis "Add Sensor", og Edit Sensor". De to neste avsnittene vil forklare nærmere hvordan disse fungerer.

3.2.5 Registrering av sensor

For å legge til sensor vil brukeren bli møtt med et oversiktlig registreringsskjema, der brukeren skal skrive inn data som trengs for å registrere en ny sensor. Informasjon brukeren må fylle inn er IP adressen til den nye sensoren. Denne IP adressen vil også bli brukt som identifikator for sensoren. Det skal kunne velges mellom eksisterende policyer fra en liste som bruker vil at sensor skal benytte. Brukeren må også fylle ut sensorens lokasjon. Det kan være stedsplass, eller navnet på kunden. Sensorene vil også bli sortert etter lokasjon, for at bruker enklere skal finne riktig sensor. En kort beskrivelse av sensor skal også være mulig å tilføye.

Endre sensor

Endringen av sensor skal, i likhet med registrering av sensor, bli gjort i webgrensesnittet til Snortmanager. Her vil brukeren få en oversikt over tidligere registrerte sensorer. Brukeren kan da endre dataene for en gitt sensor. Disse endringene blir så oppdatert i databasen. Brukeren skal også ha muligheten til å sette en sensor til aktiv eller inaktiv, som gjøres ved hjelp av en knapp i systemet. Denne knappen vil ha en fargekode som forteller om sensoren er aktiv eller ikke, henholdsvis grønn for aktiv, rød for inaktiv.

3.2.6 Logging

Systemet skal kunne logge alle hendelser gjort av systemet selv, og aktiviteter som brukeren utfører på systemet. Gruppen har valgt å dele opp loggingen fra system, og bruker, fordi å logge all programaktivitet til database vil skape enorme mengder data i databasen. Samtidig vil programmet også være mer effektivt når systemaktivitet skrives til fil.

Logging av brukeraktivitet vil lagres i database. Grunnen er at det skal være enklere å søke etter endringer og hendelser. Data som skal logges til databasen er navnet på modulen som hendelsen inntraff på, samt alvorlighetsgrad, tids/datostempel, hvilken type hendelse som inntraff og en melding forhåndsdefinert av gruppen. Det vil også genereres en automatisk ID hver gang en hendelse blir skrevet til database.

Logging av systemaktivitet fungerer ved hjelp av Pythons innebygde loggingfunksjon. Denne lar deg velge utskriftsformat, og muligheten til å skrive til fil ved hjelp av en handler. Hendelsen vil så bli lagt til i slutten av logfilen med nødvendig informasjon som dato og tidspunkt, alvorlighetsgraden, navnet på modulen, samt en melding med en beskrivelse av hva som skjedde.

Med tanke på alvorlighetsgrader innenfor logging, har vi brukt standarddefinisjonene i Python. De er Debug, Info, Warning, Error, og Critical, rangert fra minst til mest alvorlig i den rekkefølgen.

3.2.7 Rapportmodul

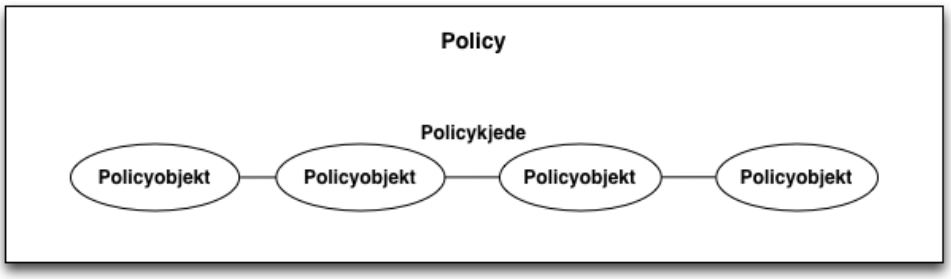
Det skal være mulig å hente ut rapporter om forskjellige deler av systemet. Rapportmodulen skal hente informasjon fra de øvrige modulene i systemet, og produsere en gjennomført visuell oversikt over dataene den hentet ut. Alle rapportene skal basere seg på en hovedrapport, og henting av data fra underliggende moduler.

Regler blir lagt inn ved å lese signaturfiler lastet ned fra tilbydere, og blir også organisert etter disse i systemet. Det skal derfor være en liste over alle filene som er registrert i systemet. Når brukeren velger fra listen skal alle regler som er i denne filen hentes ut. Den skal vise hver enkelt signaturs innhold, revisjon og SID. Det skal også vises hvor mange revisjoner som finnes i systemet. Ved å trykke på en signatur, skal man kunne få opp en liste over alle revisjoner av den enkelte signaturen som finnes i systemet.

Det skal finnes en rapport for bruk av policykjeder, og hva kjedene inneholder. En bruker skal få en liste over alle definerte policykjeder, som skal holde metadata om policykjede: Navn, beskrivlse og antall sensorer som bruker den. Når brukeren velger en kjede fra listen, skal det hentes informasjon om alle policyelementer i riktig rekkefølge, og en liste over alle sensorer som bruker kjeden.

3.2.8 Design av policy

Policyer skal brukes når Snortmanager produserer konfigurasjonsfiler for registrerte sensorer. Policyene skal være basert på egen syntaks (definert i Kravspesifikasjonen), som leses inn og blir brukt av Snortmanager for å gjøre valg under produksjonen av filer.



Figur 9: Illustrasjon av en policy med policykjede og policyobjekter.

For at det skal være mulig å gjenbruke deler av en policy i andre policyer, har følgende begreper blitt definert:

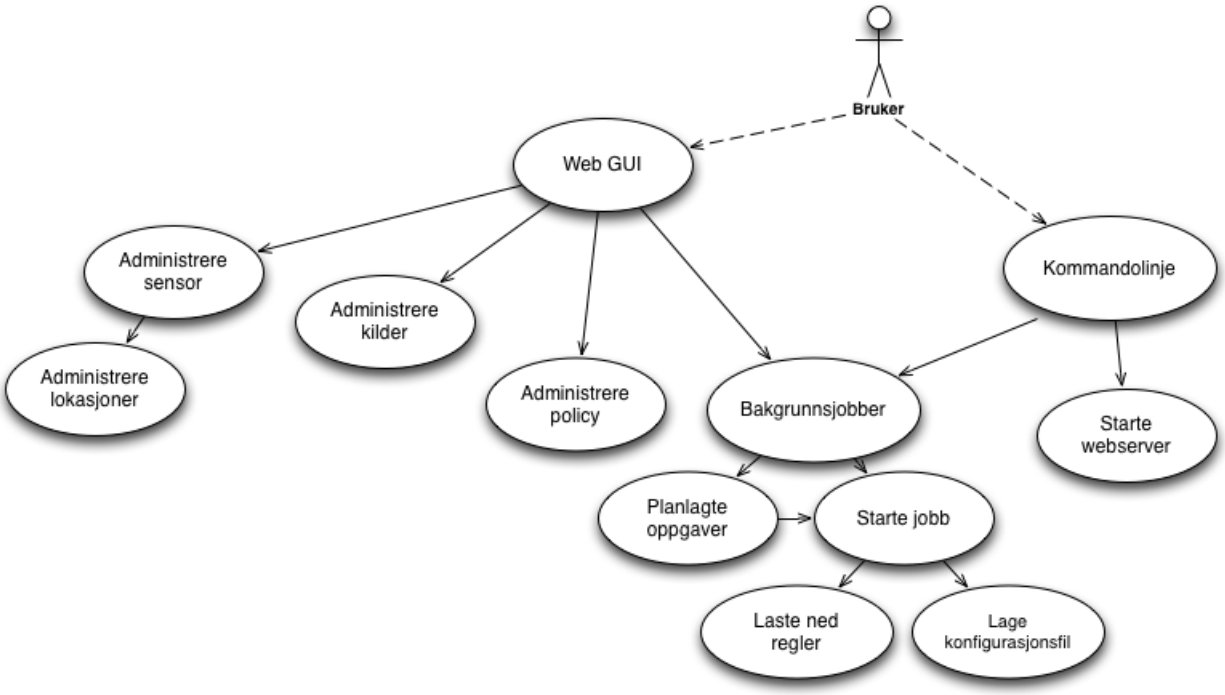
Policy: Et samlebegrep for en hel policy i systemet. Denne inneholder alle policyregler og informasjon om policyen. En policy vil inneholde strukturen og alle policyregler, og defineres med navn og beskrivelse.

Policyobjekter: Små, selvstendige enheter med policyregler. Objektene vil inneholde reglene i som klartekst, og kan brukes i alle policyer, uavhengig av hverandre. Alle endringer som gjøres i et objekt, vil reflekteres i alle policyene den er med i.

Policykjeder: Inneholder policyobjekter koblet sammen i en kjede. Kjedestrukturen er basert på listekonseptet (linked list), og vil gjøre det lettere å lagre strukturen i en policy og sortere elementene i kjeden etter behov. Når Snortmanager skal bygge opp en hel policy, vil den hente en policykjede som består av policyelementer. Policyelementer er kun et ledd i kjeden, og kan sammenlignes med noder i en liste. Elementene inneholder en kobling til et policyobjekt, og sin posisjon i kjeden. En forandring av kjedens sortering eller et av elementene vil ikke påvirke policyobjektene.

3.3 Logical view

Et logical view er en del av RUP 4+1 prosessen, og illustrerer funksjonaliteten til Snortmanager. Hensikten med diagrammet er å vise funksjonaliteten, fra brukerens perspektiv.



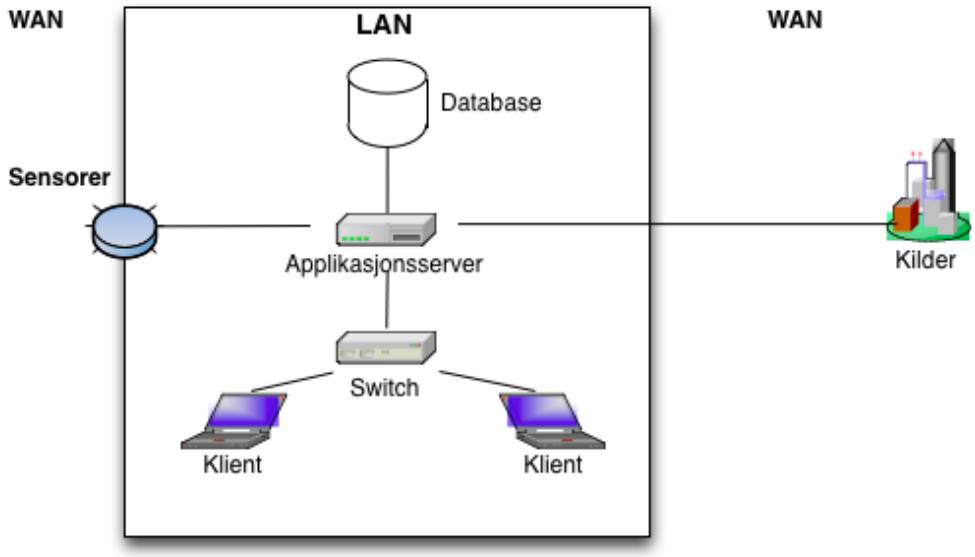
Figur 10: Illustrasjon av Snortmanagers logical view.

Diagrammet over viser at brukeren forholder seg til webgrensesnitt, eller kommandolinje. Gjennom førstnevnte kan den utføre endringer i systemet og starte bakgrunnsjobber. Kommandolinjen er tilrettelagt for at brukerene kan starte webtjenesten, og utføre bakgrunnsjobber.

Endringer omfatter handlingene legge til, fjerne og endre, som kan utføres på sensor, signaturkilder, planlagte oppgaver og policy. Brukeren kan også starte en oppdatering av signaturer, eller produksjon av konfigurasjonsfiler.

3.4 Deployment view

Et deployment view er også en del av RUP 4+1, og illustrerer Snortmanagers fysiske komponenter, og hvordan de samarbeider.



Figur 11: Illustrasjon av Snortmanagers deployment view.

I denne skissen anser vi brukerenes browsere som klienter, ettersom web-grensesnittet er en tjeneste på applikasjonsserveren. Databasen kan både være plassert på selve applikasjonserveren, eller en ekstern server. For å skildre dette har vi valgt å vise databasen separat i diagrammet.

En Snortsensor kan være plassert på eget eller eksterne nettverk, som er illustrert med å plassere sensor i grensen LAN/WAN.

Eksterne kilder er plassert utenfor det lokale nettverket, og er utenfor Snortmanager ansvarsområde. Disse serverene er plassert over hele verden, og Snortmanager bruker ikke noe spesiell webteknologi for å laste ned oppdateringer.

3.5 Valg av teknologi

Snortmanager er avhengig av mange ulike moduler for å tilby riktig funksjonalitet. Gruppen har gjennom hele prosjektet vurdert moduler som kan tilby nødvendig funksjonalitet. Grunnlaget for valgene som ble gjort er basert på følgende kriterer:

Teknologien og programvaren som velges skal være stabil, og i aller største grad feilfri. Dette sørger for sikker drift av Snortmanager, og at det ikke oppstår uforutsette feil eller stans av programvaren.

Videre er det et krav om modulers fleksibilitet. Dette bidrar til at de til enhver tid samarbeider sømløst med øvrig teknologi.

Simplisitet er også essensielt i et prosjekt som Snortmanager, hvor utviklingstiden er kort. Det skal være enkelt å sette seg inn i og lære teknologien. Et simplere design har ofte færre feil, ettersom det er lettere å oppdage feil i kildekoden.

Snortmanager skal også kunne skaleres ved behov. Ettersom gruppen ikke har mulighet til å konkret fastsette lasten og bruken av Snortmanager utover bachelorprosjektet, må teknologien

ha dokumentert at det tåler ulike mengder last, og kan utvides.

Gruppens siste kriterie er kontinuerlig utvikling og et aktivt utviklingsmiljø. Dette forsikrer at funksjonalitet ikke blir utdatert, feil utbedres fortløpende og at en kan få støtte og hjelp ved behov.

3.5.1 Rammeverk for webpublisering

Gruppen brukte mye tid på å finne et rammeverk for webutvikling under forprosjektet. Etter å ha undersøkt og vurdert flere alternativer var det CherryPy som best oppfylte til kriteriene og kravene. CherryPy karakteriseres som et mikrorammeverk, som betyr at bare det essensielle tilbys for å lage en applikasjon. Dette reduserer kravene til ressursene til serveren som skal kjøre Snortmanager.

CherryPy anses å være et av de moderne Pythonrammverkene med fortsatt aktiv utvikling. Dette medfører at det også er et av de mest stabile, både som egen webserver og koblet sammen med andre. Flere andre Pythonrammeverk benytter seg av komponenter fra CherryPy basert på stabiliteten og lave ressurskrav.

Når et produkt kun leveres med grunnleggende funksjonalitet, er fleksibilitet nødvendig for å kunne samarbeide med andre produkter og teknologier. CherryPy har løst dette med et modulært design, og tilrettelegging av funksjonalitet i separate verktøy. Dette kalles en Toolbox, og fungerer ved å separere ut kritisk funksjonalitet med øvrig funksjonalitet. APIet tilrettelegger for at utviklere skal kunne implementere øvrig teknologi og rammeverk, for så å gjøre det tilgjengelig i hele CherryPytjenesten.

3.5.2 Verktøy og rammeverk for håndtering av databaser

For å forenkle tilkobling og håndtering av database- og datastrutur ble SQLAlchemy valg. Som et av de meste brukte verktøyene, tilbyr det et sett med verktøy for å koble til database, manipulere data og tilby databasefunksjonalitet for hele applikasjonen. SQLAlchemy er også en Object Relational Mapper (ORM), som betyr at databasestruktur defineres i kildekoden til applikasjonen. På denne måten kan databaseobjekter kan behandles som Pythonobjekter.

SQLAlchemy støtter de fleste databaser på dagens marked, med planer om å støtte flere i fremtiden. SQLAlchemy kan også enkelt implementeres med CherryPy. Til slutt er det godt dokumentert, med mange oversiktlige programmeringseksempler.

3.5.3 Presentasjonsverktøy for mot sluttbrukere

Jinja2 [1] er et HTML templatingverktøy for webapplikasjoner skrevet i og for Python. Det brukes til å separere utviklingen av prossesering og presentasjon mot sluttbrukerene. Rammeverket er basert på presentasjonsdelen av et annet populært rammeverk; Django. Dette fungerer ved å ha egne HTMLfiler som tar variabler fra webtjenesten, og med en egen syntaks blir variablene skrevet ut som nettinnhold. Snortmanager velger en HTMLfil når en metode er avhengig av å presentere innhold, og sender med data som brukes til å presentere innholdet.

Gruppen valgte dette rammeverket for Snortmanager mest for simplisitet, da syntaksen for HTML-filene er lik vanlig HTML. Det vil si at HTML-filene ser helt normale ut, bortsett fra egen syntaks for rammeverket som skriver ut data. Det er også enkelt å lage komplette templates og

[1] http://jinja.pocoo.org/

definere filer som inkluderes i andre filer. Dette kombinert med gjennomført god dokumentasjon gjør at mindre utviklingstid går med på å lage webinnhold. Jinja2 integreres også lett i CherryPy som et eget verktøy i Toolbox-systemet.

3.5.4 Planlagte oppgaver

Planlagte oppgaver i Snortmanager skal implementeres med modulen Advanced Python Scheduler (APS) [2]. Modulen er enkel å implementere, uten "harde"avhengigheter til øvrige pakker. Den gir også mulighet til å planlegge oppgaver basert på dato/tidspunkt, intervaller eller cron-syntaks.

APS har også støtte for SQLAlchemy, som gjør at planlagte oppgaver kan lagres i Snortmanagers database. Dette forenkler både organiseringen og administrasjonen av planlagte oppgaver, ettersom man enkelt kan presentere til sluttbrukeren gjennom grensesnittet. APS er også thread safe", som betyr at det fungerer utmerket med separate tråder, og alles egne tråder effektivt og sikkert.

[2] http://packages.python.org/APScheduler/

4 Implementering

Denne delen av rapporten tar for seg hvordan Snortmanager er implementert, basert på kravspesifikasjonen og designdokumentet. Det inneholder skjermbilder, illustrasjoner og utsnitt fra applikasjonen og kildekoden.

4.1 Valg av lisens

Ettersom Snortmanager skal bli tilgjengelig som åpen kildekode, måtte det vurderes og velge en lisens som tillater dette. Følgende kriterer ble lagt til grunn for valget:

- All modifikasjon av kildekode skal kunne intergreres i alle prosjekter som benytter kode fra Snortmanager.
- Et produkt som inneholder kode fra Snortmanager skal kunne være kommersielt.
- At utviklerne av produktet ikke er ansvarlige ved evenutelle komplikasjoner som oppstår.

Gruppen gikk gjennom flere lisenser under utviklingen, men reduserte alternativene til to populære lisenser:

- BSD lisensen utviklet av University of California, Berkeley.
- GNU GPL lisensen utviklet av Richard Stallmann.

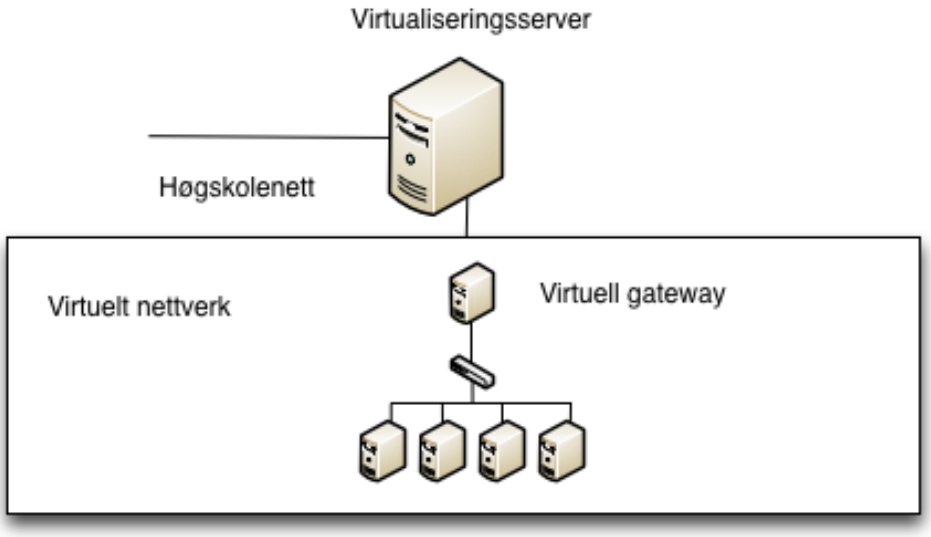
Begge lisenser innholder ansvarsfraskrivelse, og restriksjoner om behandlingen og distribusjonen av kildekode. Til slutt ble en GPL-lisens valgt, da den krever at kildekoden må publiseres, uavhengig om koden brukes kommersielt. Dette kan ikke garanteres med en BSD lisens, siden nye utviklere kan ta eksisterende kildekode og gjøre modifikasjoner uten å dele det. Med GPL-lisensen forventes det at flere prosjekter kan videreutvikle Snortmanagers, samtidig som alle kan dra nytte av arbeidet. På sikt vil dette forbedre kildekoden og funksjonaliteten.

4.2 Utviklingsmiljø

Denne delen tar for seg teknologi, systemer og oppsett av infrastrukturen brukt under utviklingen av Snortmanager.

4.2.1 Virtualisering

Oppdragsgiver gikk til innkjøp av en server som skulle benyttes til utvikling og testing av Snortmanager. Det ble bestemt at denne skulle brukes som virtualiseringsserver. Virtualiseringsløsningen som ble valgt var gratisversjonen av VMWare VSphere 5, et anerkjent og mye brukt virtualiseringsprodukt. Produktet er kraftig, og designet for å simulere enorme driftsmiljø.



Figur 12: Diagram som viser den virtuelle testinfrastrukturen.

Høgskolen har kun et begrenset antall IP-adresser på skolenettet. Ettersom det var vanskelig å beregne hvor mange servere prosjektet trengte, ble det opprettet et eget internt nettverk på virtualiseringsserveren. Gruppen kunne også på denne måten redusere risikoen for sabotasje, ved å begrense tilgangen til testmaskiner. Selv om testservere ikke kunne gjort stor skade på skolens nettverk, var det også beroligende å kunne begrense tilgangen ut av infrastrukturen.

Testnettet ble satt opp med IP-segmentet 10.0.0.0/24, og var kun eksponert mot øvrige nettverk med nettets gateway. Dette var kun en virtualisert server som kjørte med samme operativsystem og tilnærmet likt oppsett som de øvrige virtuelle serverne. Gatewayen kjørte ikke noe programvare brukt til testing, men hadde ansvaret for DNS, utdeling av IP, internettilgang og tilgangen til testnettet fra utsiden. Ved hjelp av dette oppsettet brukte testinfrastrukturen kun to av Høgskolens IP-adresser, én for virtualiseringsserveren og én for gatewayen.

Selv om virtualiseringsløsningen er gratis, er den kraftig med mye funksjonalitet. Samtidig har utviklerne fjernet noe funksjonalitet, som kun tilbys til betalende kunder. Et eksempel er templating, hvor man kan opprette en servermal som brukes til å lage nye servere ved behov. Siden gruppen ikke kunne betale for dette, ble tiltak gjort for å emulere funksjonaliteten.

Til å begynne med ble det opprettet en virtuell server på virtualiseringsserveren, som hadde ferdig installert operativssystem og Python. Ved å aktivere SSH på virtualiseringsserveren, kunne gruppen kopiere denne og opprette en ny maskin ved hjelp av Unix-kommandoer. Deretter kunne man bruke administrasjonssgrenesnittet til virtualiseringstjenesten for å behandle den nye serveren.

4.2.2 Operativssystem

Alle virtuelle maskiner i miljøet kjører CentOS Linux 6.0. Dette er etter ønske fra oppdragsgiver, da operativsystemet brukes i deres driftsmiljø. Gruppekompetansen på denne Linux-distribusjonen

er i tilllegg høy. Operativsystemet har også et omfattende tredje-partsbibliotek av programvare som gjør det enkelt å installere programvare utviklingen er avhengig av. Med en bred installasjonsbase og aktivt brukersamfunn, er hjelp og støtte lett å oppdrive.

4.2.3 Python

Versjon 2.6 og 2.7 av programmeringsspråket Python ble brukt under utviklingen. Disse er de mest stabile versjonene av 2.0-generasjonen, og anbefalt av Python-utviklerne av språket. Forskjellen mellom dem er minimal, bortsett fra noen små feilutrettelser. Begge versjonene av Python ble valgt ettersom forskjellige pakkebiblioteker inneholder forskjellige versjoner. Det var ikke nødvendig å installere Python på testserverne, ettersom dette er en del av operativssytemet.

4.2.4 MySQL

MySQL fra Oracle ble valgt som databasesystem, da det er den mest utbredte databasemotoren basert på åpen kildekode. Det er også den databasen gruppen har mest erfaring med, da den er brukt i skole- og/eller privat sammenheng. Databasen ble installert med ferdige pakker med operativsystemets bibliotek.

4.2.5 SSH

Filoverføring og tilgang til miljøet ble gjort med SSH. Dette er en sikker, stabil og enkel måte å få tilgang til utviklingsmiljøet. SSH krever i tillegg lite, til ingen konfigurasjon. SSH blir også brukt til å lage en sikker, kryptert tunnel til testmiljøet slik at løsningen kan testes med datamaskiner utenfor testmiljøet.

4.3 Inspirasjon og bruk av åpen kildekode.

Deler av prosjektet er inspirert av andre prosjekter som også er åpen kildekode.

I oppdatering av Snortsignaturer fra eksterne kilder kom gruppen over et problem med representasjon, og å hente ut relevant informasjon fra signaturene. Etter flere søk på Internett kom ble prosjektet Rule2Alert [1] oppdaget, et system som genererer trafikk basert på Snortsignaturer. Dette hadde en klasse med konstruktor, som leser inn en Snort signatur som tekststreng og henter ut relevante data. Denne konstruktoren er lagt som egen fil i modulen Downloader.

Da arkitekturen av løsningen skulle designes fantes ingen god dokumentasjon for korrekt oppsett av CherryPy. Etter flere dager med leting ble det bestemt å velge en ny strategi. GitHub [2] som hoster mange av verdens softwareprosjekter, ble brukt til å finne prosjekter som benyttet seg av CherryPy. Der ble prosjektet CouchPotato [3] funnet, som inneholdt oversiktlig kode, teknologi lik den valgt for Snortmanager og lettfattelig struktur. Dette prosjektet ble derfor brukt som inspirasjon flere steder i Snortmanager.

Som nevnt tidligere finnes det ikke mange løsninger basert på åpen kildekode, som kunne brukes som referanse. Pulled Pork [4], som leveres av Sourcefire Inc, ble brukt tidlig i prosjektet. Det ble kun brukt som refereanse under utviklingen av signaturoppdatering.

Prosjektet er også hatt tilgang til oppdragsgivers gamle løsning for å administrere Snortsensorer, men dette ble i likhet med Pulled Pork kun brukt som referanse tidlig i prosjektet.

[1] http://code.google.com/p/rule2alert/
[2] http://www.github.com
[3] http://www.github.com/RuudBurger/CouchPotato
[4] http://code.google.com/p/pulledpork/

4.4 Implementering av CherryPy

CherryPy er et minimalistisk rammeverk for webutvikling med Python, men er fortsatt kraftig og lett å implementere. For å forstå hvordan gruppen har intergrert CherryPy i prosjekt, bør man først vite hvordan CherryPy fungerer.

```
import cherrypy

class HelloWorld:

    @cherrypy.expose
    def index(self):
        return "Hello world!"

cherrypy.quickstart(HelloWorld())
```

Ovenfor ser man en modifisert utgave av CherryPys "Hello Worldeksempel fra dokumentasjonen [5] deres. CherryPy integreres i et prosjekt, ved å importere modulen i prosjektet. Deretter defineres et objekt som skal presentere innhold for siden, som i eksemplet over er en instans av **HelloWorld()**-klassen. Metodene i klassen brukes til å behandle og publisere innhold til brukeren.

I eksemplet deklareres metoden **index()**, som er en standard metode for å vise en root-side i CherryPy. Dette er tilsvarende andre webservere, hvor root-siden er en fil som heter index.html. Metoden returnerer kun teksten "Hello World!", som vises når noen kobler til serveren med nettleseren. Over deklarasjonen av klassen finner man Python-direktivet *@cherrypy.expose*, som forteller serveren at denne metoden er en nettside og skal eksponeres til brukere. Til slutt startes CherryPy-serveren med **quickstart()**, som tar root-klassen som attribut, og begynner å tilby siden til brukerene.

Hello World-eksemplet viser at konfigurasjon av CherryPy er enkelt, og egner seg for små websider med lite innhold. Arkitekturen av Snortmanager gjør derimot at denne typen implementasjon fungerer dårlig. Arkitekturen er Model-View-Controller, og bare kontrollerene alene ville resultert i en rotete kildekode. Hvordan dette ville sett ut kan man se i eksemplet under:

```
import cherrypy

class Snortmanager():
    """ Root klasse for eksemplet """

    @cherrypy.expose
    def index(self):
        """ Siden som vises til brukerene aksesserer siden. """
        return 'Welcome to Snortmanager'
```

[5] http://docs.cherrypy.org/stable/concepts/basics.html

```
class Policy():
    """ Kontroller for policyadministrasjonen av eksemplet. """

    @cherrypy.expose
    def index(self):
        """ Siden som vises hvis folk henter ut /policy/ siden """

        return 'Administer sensor'

root_page = Snortmanager()
root_page.policy = Policy()

cherrypy.quickstart(root_page)
```

4.4.1 Mapping av URL og MVC-Controllere

Som kodeeksemplet viser, ville det med Snortmanager blitt mye spaghettikodefor kontrollerne. Det ville blitt nødvendig å opprette en instans av hver kontroller, som på sikt ville blitt uoversiktlig. Dette er også bare et enkelt eksempel som viser publisering av innhold med enkle URLer. Når Snortmanager trenger funksjonalitet til å ta imot data i fra webskjemaer og hente variabler for URLer, blir kildekoden enda mer komplisert. For å redusere kompleksiteten for koden, ble den normale dispatchern (måten man publiserer metoden, bygger og mapper URLer på) erstattet med CherryPys RouteDispatcher [6] .

Routes dispatcher er basert på Routes, originalt implementert i Ruby on Rails [7], men senere portert til Python i Routes-modulen [8]. Routes fungerer slik at URLer mappes mot metoder og handlinger på forhånd. Dette betyr at URLene ikke er avhengig av metodenavn og klassen de er i. Routes-konseptet forenkler også videre utvidelser, f.eks. subdomener for et fremtidig API ved å tillate nye URLer uten store endringer av CherryPys eller Snortmanagers konfigurasjon.

Snortmanger definerer alle URLer og mappingen mellom controllere og metoder i filen `routes.py`. Modulen importerer alle controllerne til Snortmanager, og ved hjelp av metoden **connect()**, som er en del av *RoutesDispatcher*. Denne oppretter et internt namespace for kontrolleren, og definerer URL og hvilken metode som skal brukes.

```
try:
    import cherrypy
    from webapp.controller.background_jobs import JobsController
except ImportError as e:
    print 'Error while importing error in', __file__
    print e

def initialize_routes():
    """ Initialiserer routes for Snortmanager """

    mapper = cherrypy.dispatch.RoutesDispatcher()
```

[6] http://docs.cherrypy.org/stable/refman/_cpdispatch.html#cherrypy._cpdispatch.RoutesDispatcher
[7] http://guides.rubyonrails.org/routing.html
[8] http://pypi.python.org/pypi/Routes/

```
    mapper.minimization = False
    mapper.explicit = False
    mapper.append_slash = True

    #Dispatcher og mapping for bakgrunnsjobber
    mapper.connect('jobs', '/jobs/',
                                controller = JobsController(), action='index')
    mapper.connect('jobs', '/jobs/start_job/',
                                controller = JobsController(), action='start_job')
    mapper.connect('jobs', '/jobs/schedule/',
                                controller = JobsController(), action='schedule'
```

Eksemplet over viser hvordan mapping blir initialisert og definert for *JobsController*. som tilbyr funksjonalitet for administrering av bakgrunnsjobber. Den første attributen er namespacet mapperen skal opprette og bruke. Deretter defineres URLen brukeren skal benytte i sin nettleser. Til slutt brukes *controller* parameteren for å definere hvilken kontroller som benyttes, og *action* parameteren for å definere riktig metode.

4.4.2 MVC-Controllere

Kontrolleren i systemarkitekturen er bindeleddet mellom model og view. Det er ansvarlig for å behandle data og forberede presentasjon. Snortmanagers kontrollere ligger i mappen*webapp/controller*, og det finnes én klasse for hver enkel kategori av løsningen. For eksempel har policyadministrasjon sin klasse, mens sensoradministrasjon har sin.

Alle klassene for funksjonalitet er en underklasse av *BaseController()*, som kun inneholder funksjonalitet som brukes av alle kontrollerne. Underkontrollerne inneholder metoder og data som er relevant for hovedfunksjonalitet de skal tilby sluttbrukeren, og metodene som er mappet opp mot URLer er eksponert.

```
try:
    from webapp.controller import BaseController
    import cherrypy
except ImportError as e: #If there's an error
    print 'There was a importerror in', __file__
    print e


class MainController(BaseController):
    """ Kontroller som brukes for aa presentere Snortmanagers hovedside """

    @cherrypy.expose
    @cherrypy.tools.jinja(template='main/index.html') # Henter korrekt template
    def index(self):
        """ Hovedside som skal vises til sluttbrukere """
        event_content = Session.query(EventLog).limit(10) # Henter logg fra database

        return (self.render({ 'page_title': 'Main page',
                                    'event_content': event_content}))
```

Eksemplet over viser hvordan konrolleren presenterer innhold for Snortmanagers landingsside (eller hovedside). Før metoden ser man to Python-dekoratorer. Den første forteller som tidligere CherryPy at metoden skal eksponeres slik at det blir en webside, mens det andre henter template for siden. Dette presenteres senere i rapporten, men forklares kort i paragrafen under.

Metoden **index()** henter en liste med ti linjer hendelseslogg fra databasen. Dette returne-

res til templatefilen i form av en Pythonliste. Rendermetoden som blir brukt er nedarvet fra BaseController, og brukes av alle kontrollerne for løsningen.

4.5 SQLAlchemy

Snortmanager skal lagre all informasjon i en MySQL database, og bruker SQLAlchemy for å blant annet manipulere data og koble til databasen. SQLAlchemy har to hoveddeler: Core er komponeneter som brukes for databasekoblinger, transaksjoner og sesjoner. Med andre ord blir disse brukt for å koble til og bruke databasen. Den andre delen er ORM, (Object Relational Mapper). Disse komponentene brukes for å definere tabeller og databasestruktur, slik at de kan brukes som Python-objekter.

All kode for behandling av database ligger i `dbhandler.py`, som ligger i *webapp/config* mappen. Denne ene filen er tilkobling til databasen, og deklarasjonen av strukturen.

Måten man definerer tabeller og deres struktur på kalles declarative. Det fungerer ved at man definerer en instans med metoden **declarative_base()**, som alle klasser som skal definere databasetabeller vil arve fra. Alle attributter som skal være en del av tabellene, defineres i klassen med egne SQLAlchemy typer, som blir oversatt til databasespesfikke verdier av SQLAlchemy. Disse typene kan ha attributer som definerer hvordan de skal fungere i databasen, for eksempel at de ikke skal være tomme (NOT NULL). Til slutt deklareres en constructor og en **__repr__()**-metode, som forteller hvordan objekter skal defineres og brukes av SQLAlchemy.

Følgende eksempel viser hvordan tabellen Sensor defineres for Snortmanager. Eksemplet viser at å bruke SQLAlchemy reduserer tiden man bruker på utvikling og testing, og gjør det lettere å manipulere rader i en database:

```
Base = declarative_base() # Oppretter en instans av Declarative() klassen

class Sensor(Base):
    """ Klasse for Sensor tabellen i databasen. """
    __tablename__ = 'Sensor' # Setter navnet paa tabellen i databsen
    id = Column('id', Integer, primary_key = True, autoincrement = True)
    name = Column('name', String(50), nullable = False)
    ip = Column('ip', String(20), nullable = False)
    location = Column('location', INTEGER(unsigned = True),
                      ForeignKey(SensorLocation.id), nullable = False)
    description = Column('description', String(140), nullable = False)
    policychain = Column('policychain', Integer, nullable = False)

    def __init__(self, name, ip, location, description, policychain = 0):
        """ Konstruktor slik at man kan opprette i databasen. """

        self.name = name
        self.ip = ip
        self.location = location
        self.description = description
        self.policychain = policychain

    def __repr__(self):
        """ Representasjon av objektet """

        return "<Sensor('%i','%s',␣'%s',␣'%s',␣'%s')>" % (self.id,
        self.name, self.ip, self.location, self.description)
```

Når hver tabell er deklarert og definert, må de opprettes i databasen. Dette gjøres gjennom metoden **create_all()** i Base-objektet. Denne utføres av metoden **init_database()**, som kjøres av CherryPy ved hver oppstart. Utdraget under viser hvordan tabeller blir opprettet.

```
try:

# Initialize database
def init_database():
    try:
        Base.metadata.create_all(engine)

except OperationalError:
    print 'Error while connecting to database'
```

Eksemplet viser at **create_all()** tar *engine* som parameter. Dette er en instans av SQLAlchemy Engine, som inneholder grunnleggende funksjonalitet til SQLAlchemy. Klassen oppretter en tilkobling til databasen, oppretter en pool med tråder som kan brukes for å gjøre operasjoner mot databasen, identifiserer databasetypen og oppretter de øvrige APIene benyttes av de øvrige delene til Snortmanager.

```
try:
    engine = create_engine('mysql://passord:brukernavn@server/database',
    echo=False, pool_recycle=3600)

    metadata = MetaData(engine)
    Session = scoped_session(sessionmaker(bind = engine, autocommit = True))
except OperationalError as e:
    print 'Error while connection to database, shutting down!'
    print e
    exit(1)
```

Eksemplet over viser hvordan en engine opprettes for SQLAlchemy. Metoden **create_engine** viser at den får en URL-streng med type dialekt, brukerinformasjonen, serveren og databasen som skal brukes. Deretter opprettes et *MetaData*-objekt, som tar engine-instansen som parameter. Instansen metadata brukes av SQLAlchemy for å kartlegge databasens struktur, opprette nye tabeller og manipulere strukturen til eksisterende ved behov.

Til slutt opprettes et Scoped Session-objekt, som brukes for å forbedre databasetilkoblingen for applikasjonen. Scoped session fungerer ved at all håndtering av databasetilkoblinger skjer pr. tråd, som passer ypperlig for en webapplikasjon hvor all aktivitet skjer i tråder. Når denne instansen skapes av **sessionmaker**, tar den også parameteren *autocommit = True*, som betyr at SQLAlchemy automatisk oppretter og avslutter alle transaksjoner. Dette fordi Snortmanager på ingen måte er avhengig av transaksjoner for å utføre oppgavene sine.

Når øvrige moduler som kontrollere og bakgrunnsjobber trenger å arbeide med databasen, importeres kun de instanser og klasser fra `dbconfig.py` som er nødvendige for å utføre selve jobben. All øvrig funksjonalitet importeres direkte fra SQLAlchemy-modulene.

```
try:
    from webapp.controller import BaseController, url, redirect
    from webapp.config.dbconfig import Session, EventLog
    from sqlalchemy.orm import query
```

```
    import cherrypy
except ImportError as e: #If there's an error
    print 'There was a importerror in', __file__
    print e

class MainController(BaseController):

    @cherrypy.expose
    @cherrypy.tools.jinja(template='main/index.html')
    def index(self):
        event_content = Session.query(EventLog).limit(10) # Sporring mot databasen

        return (self.render({ 'page_title': 'Main page', 'event_content': event_content}))
```

Eksemplet over viser igjen et utdrag fra Snortmanagers hovedside. Her importeres først *Session*-instansen og *EventLog*-klassen fra `dbconfig.py`. Deretter importeres **query()**-metoden fra SQLAlchemys ORM modul. Deretter ser vi i **index()**-metoden at det kjøres en spørring med *Query*-klassen i *Session*-instansen. Denne tar klassen til tabellen den skal gjøre spørringer mot, som parameter. Videre benyttes **limit()** metoden, som begrenser antallet returnerte rader til 10.

All form for innsnevering av søk fungerer med metoder i *Query* klassen. I eksemplet over ble **limit()** brukt, men andre metoder brukes også:
Query.filter(): Brukes når man trenger å filtere et søk.
Query.one(): Brukes når man trenger å kun hente en rad.
Query.all(): Brukes når man trenger å hente alle rader.
Query.first(): Brukes når man trenger å hente de første radene.
Query.order_by(): Brukes når man trenger å sortere etter en attribut.
Query.join(): Brukes når man trenger å koble to tabeller sammen.

Vi har nå beskrevet detaljert hvordan man data hentes, men Snortmanager skal også legge til og manipulere eksisterende data i databasen. For dette brukes ikke **Query** klassen, men øvrige metoder i *Session*-instansen.

```
try:
    from webapp.config.dbconfig import Session, PolicyChainMeta
except ImportError as e: #If there's an error
    print 'There was a importerror in', __file__
    print e

    @cherrypy.expose
    def add_policy(self, name='', description=''):

        if len(name) > 0 and len(description) > 0: # Sjekker lengden til objektene
            new_policy = PolicyChainMeta(escape_string(name.capitalize()), escape_string(description))
            Session.add(new_policy) # Legger det til i databasen
            Session.flush() # Tvunget SQL Flush
```

Eksemplet over viser hvordan en ny policy opprettes av Snortmanager. Det vil først opprettes en ny instanse av *PolicyChainMeta* -klassen med navnet og beskrivelsen av policykjeden. Deretter vil dette legges til databasen med **add()**-metoden i *Session* instansen. Til slutt utføres en **flush()** av databasen for å forsikre seg om at bufferet tømmes, og dataene blir korrekt skrevet til databasen. Det siste steget er ikke nødvendig, men noe som ofte gjøres for å skrive data til databasen med en gang.

Hvis man trenger å manipulere en rad i databasen som allerede eksisterer, henter man denne ut med med *Query* for så å manipulere objektet som hvilket som et helst Python-objekt. Deretter skrives det tilbake til databasen med metoden **Session.merge()**, som vist i eksemplet under:

```
from webapp.config.dbconfig import Session, PolicyObject, PolicyChain

    def choose_object(self, **kwargs):
        try:
            if 'object-id' not in kwargs and 'policy-id' not in kwargs:
                raise KeyError

            object_id = kwargs['object-id']
            policy_id = kwargs['policy-id']

            policy_element = Session.query(PolicyChain).filter(
            PolicyChain.id == policy_id).one()

            policy_element.policyobject_id = object_id

               Session.merge(policy_element)
            Session.flush()

        except KeyError:
            print 'KeyError'
```

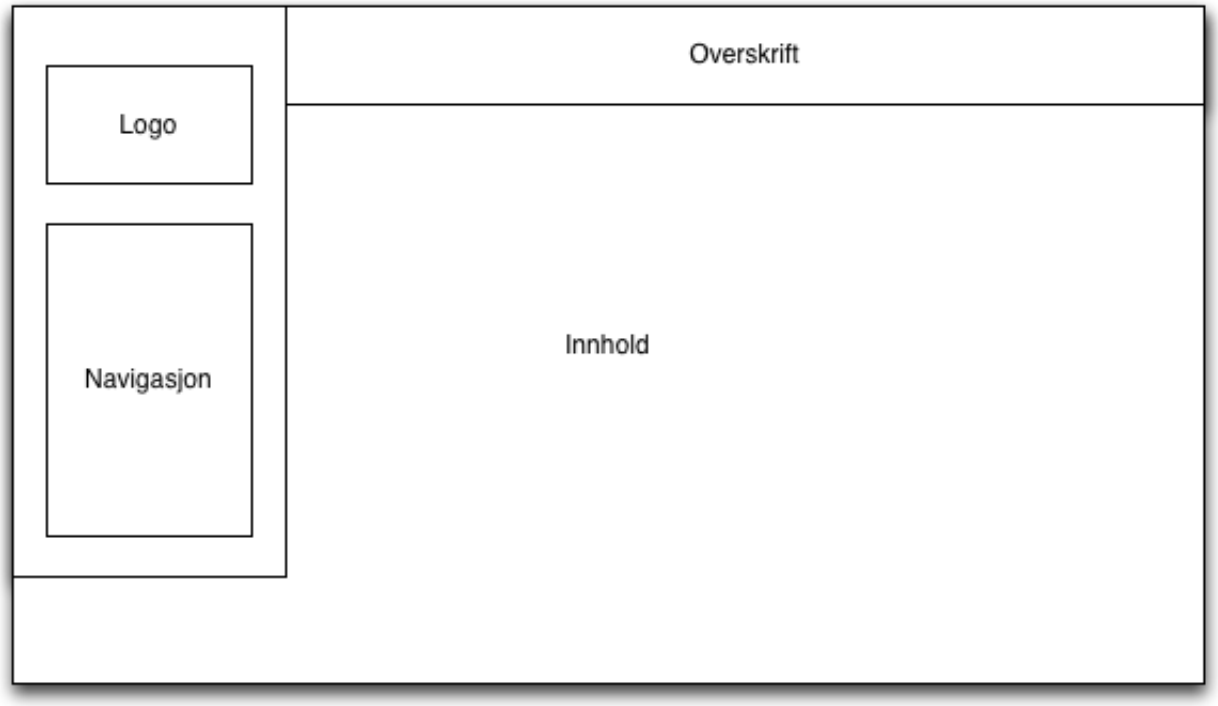
Utdraget viser hvordan en bruker kan velge et nytt policyobjekt for et policyelement. Først blir det eksisterende policyelementet hentet ut, deretter manipuleres objektets *policyobjecy_id* atributt, før det skrives tilbake til databasen.

Som demonstrert er SQLAlchemy et omfattende rammeverk for å behandle data i en database. Det har vært enkelt å implementere og ikke minst enkelt å ta i bruk. Det har redusert risikoen for SQL-feil betraktelig, og ikke minst uviklingstiden ved å forenkle blant annet feilsøking.

4.6 Visuelt design og templates

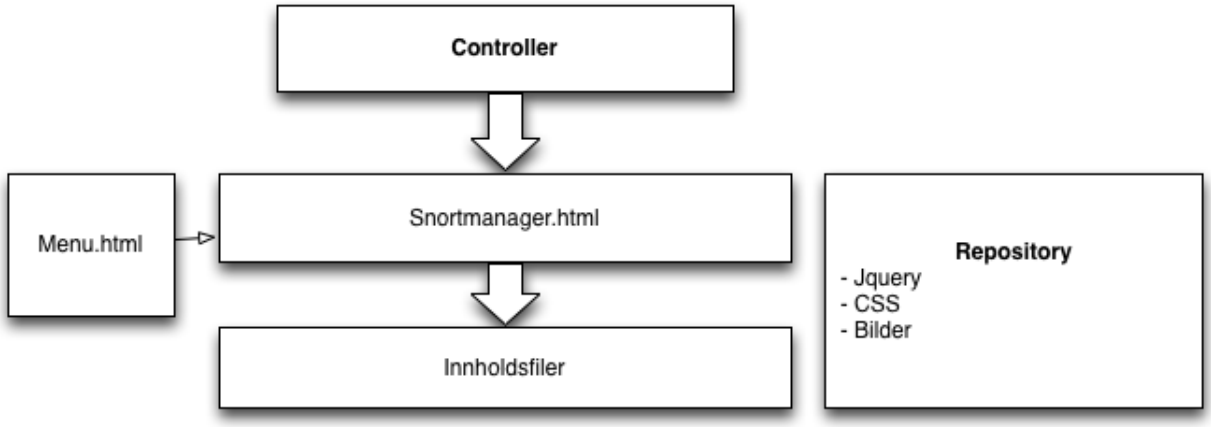
Det visuelle designet og presentasjonen er implementert med HTML5, CSS og presentasjonsverktøyet Jinja2. Noe funksjonalitet er implementert med javascript og jQuery. Det er ikke krav om at Snortmanager skal være visuelt pent. Det skal derimot være oversiktig og praktisk strukturert.

Flere forskjellige måter å strukturere og organisere grensesnittet på ble vurdert. Til slutt ble det besluttet å implementere et design hvor grensesnittet har en header med overskrift for funksjonen man bruker. Man finner også listebasert navigasjon og logo i venstre kolonne, og hovedinnhold i øvrig grensesnitt.



Figur 13: Diagram av webgrensesnittets layout.

Den listebaserte navigasjonen er laget som en statisk unummerert liste med to nivåer. Det første nivået beskriver hvilken del av løsningen man administrerer, det andre nivået beskriver underfunksjonalitet. Listen er laget for å kollapse når man ikke bruker en en modul. For å spare utviklingstid, ble det ikke implementert lister som kollapser. Isteden presenteres alle nivåer til enhver tid. Menyen oppbevares i filen menu.html, og inkluderes i alle views.



Figur 14: Diagram som viser hvordan elementer arves i Snortmanager.

Diagrammet ovenfor viser den hierarkiske modellen for presentasjon av Snortmanager. Hver enkelt metode som er eksponert definerer hvilken HTML-fil de skal bruke som template for å presentere innhold. Dette gjøres med Python-direktivet *@cherrypy.tools.jinja()*, som tar den ønskede templaten som parameter.

Ser man nøye på direktivet over, oppdager man at denne egentlig er et verktøy i CherryPys Toolbox. Hver gang direktivet blir brukt, vil CherryPy-serveren bruke en HTML-rendrer for å produsere og presentere innholdet til sluttbrukeren. Denne renderen er egentlig en del av controllerne, men er lagret i *webapp/config*-mappen i filen `render.py`. Eksempelet under viser hvordan en side henter korrekt template, og populerer det med data.

```
@cherrypy.expose
@cherrypy.tools.jinja(template='background_jobs/index.html')
def index(self):

    return ({ 'page_title': 'Background jobs'})
```

4.7 Filstrukturen til Snortmanager

Webløsnigen er designet med MVC arkitektur, samtidig som applikasjonen i sin helhet baserer seg på lagmodellen. Det var også viktig at applikasjonen skulle være mest mulig selvstendig, uavhengig av installasjonsskript og være enkel å implementere.

Dette har ført til en filstruktur som ser slik ut:

```
| - \texttt{Snortmanager.py} - Administrasjonsskript for Snortmanager.
| - \texttt{configuration.conf} - Konfigurasjonsfil for Cherrypy.
| - \textit{jobs/}- Mappe med programfiler for bakgrunnsjobber.
| - \textit{library/}- Bibliotek med kildekode for tredjeparts bibliotek.
| - \textit{logs/}- Mappe for lagring av logfiler.
| + \textit{resources/}- Ressurser med statisk innhold for bruk av wepapplikasjon.
```

```
   | -  \texttt{favicon.ico} - Favicon presentert til brukeren.
   | - \textit{images/}- Bilder brukt av webapplikasjon.
   | - \textit{js/}- Javascriptbibliotek og filer.
   | - \textit{stylesheet/}- CSS filer for webapplikasjon.
| - \textit{templates/}- HTML-filer brukt for å presentere innhold til sluttbruker.
| - \textit{var/}- Filområdet Snortmanager bruker for filer som oppretter og/eller endrer seg und
| + \textit{webapp/}- Programfiler og kildekode for webapplikasjons.
   | - \textit{config/}- Hjelpefiler til webapplikasjonens controllere.
   | - \textit{controller/}- Controllere brukt av webappen.
| - \texttt{webapp.conf} - Konfigurasjonen for selve applikasjonen Snortmanager.
```

For å forenkle administrasjon av Snortmanagertjenesten, gjøres alle operasjoner gjennom et enkelt administrasjonsskript som heter `Snortmanager.py`. Dette vil beskrives i detalj senere i rapporten.

Videre har Snortmanager to konfigurasjonsfiler, anbefalt av CherryPys dokumentasjon [9]. De er delt opp i global konfigurasjon (configuration.conf), som har konfigurasjonsdirektiver for blant annet webserveren til CherryPy. For Snortmanager er dette porter den skal lytte til, IP-adresser og øvrige konfigurasjon av selve webserveren. Den andre konfigurasjonsfilen er for selve webapplikasjonen Snortmanager, og inneholder direktiver som hvor statiske filer er og hvilke mapper som er tilgjengelig i nettleseren. Konfigurasjonsinformasjon for databasen og informasjon om feilmeldinger ligger også i denne filen.

Det er viktig at Snortmanager ikke støter på konflikter med tredjeparts kodebiblioteker på serveren den skal kjøre på, og at det tar kort tid å ta det i bruk etter installasjon. Det er derfor valgt å inkludere biblioteker som ikke følger med i Python. Disse blir kompilert og lagt i mappen *library*. Dette fører til at Snortmanager benytter mer plass på harddisken, men reduserer risikoen for potensielle driftsproblemer, og gjør det lettere å sette opp Snortmanager.

De tre mappene som de tre delene av MVC-arkitekturen er *webapp/config*, *webapp/controller/* og *templates/*. Filer som brukes av modeldelen ligger i *webapp/config/*. Denne er ansvarlig for koblingen mot verktøy som brukes for presentasjonsinnhold, tillatte webaddresser, SQL databasen og dens struktur.

Viewsdelen ligger i mappen *templates/*, og inneholder HTML-filer som kalles templates. Filene inneholder kun HTML-kode og syntaks fra Jinja2, og bruker ressurser i *resources/*som Javascript og CSS for å bedre kunne presentere innholdet til sluttbrukeren.

Controllere benytter seg av views og model for å bygge opp siden for sluttbrukeren, og ligger tilgjengelig i *controller*mappen under webapp. Det finnes én controller for hver enkelt del av løsningen, som for eksempel policy, jobber, regler og sensorer. Alle disse arver fra et controllerobjekt, som inneholder felles metoder og attributer.

Mappen *jobs* inneholder kildekode og nødvendig informasjon som Snortmanager bruker for å utføre bakgrunnsjobber. Disse filene er også til dels avhengige av filer fra *webapp/config*for å kommunisere med databasen.

[9] http://docs.cherrypy.org/stable/concepts/config.html

4.8 Administrasjonsskriptet Snortmanager.py

I beskrivelsen av filstrukturen ble administrasjonsskriptet Snortmanager.py introdusert. Dette skriptet benyttes for å redusere kompleksiteten rundt administrasjon av Snortmanagerprogrammet. Ved å bruke dette skriptet kan brukeren utføre operasjoner som:

- Starte Snortmanger med og uten utskrift av hendelseslogg til skjerm.
- Starte og stoppe Snortmanager som bakgrunnstjeneste (daemon).
- Laste ned og oppdatere Snortsignaturer fra eksterne kilder
- Starte produksjon av konfigurasjonsfiler for Snortsensorer

Hver gang skriptes startes, oppretter operativssystemet en prosess. Skriptet avgjør hvilke operasjoner som skal utføres med kommandolinjedirektiver. Hvis brukeren ikke oppgir noen slike direktiver, vil skriptet starte Snortmanager med utskrift av brukeraktivitet til skjerm. Dette er standardfunksjonalitet for CherryPy.

```
[root@TestServer1 KodeBase]# python Snortmanager.py -h
usage: Snortmanager.py [-h] [-d] [-u] [-p] [-s] [--status] [--stop]

Administration script for the Snortmanager uility

optional arguments:
  -h, --help  show this help message and exit
  -d          Run server as a daemon
  -u          Update rules from sources
  -p          Produce and push policy chain
  -s          Silent operation, log nothing to screen
  --status    Check the running status of Snortmanager
  --stop      Stop Snortmanager if running as daemon
[root@TestServer1 KodeBase]#
```

Figur 15: Hjelpemenyen som viser hvilke operasjoner brukeren kan gjøre med Snortmanager.py

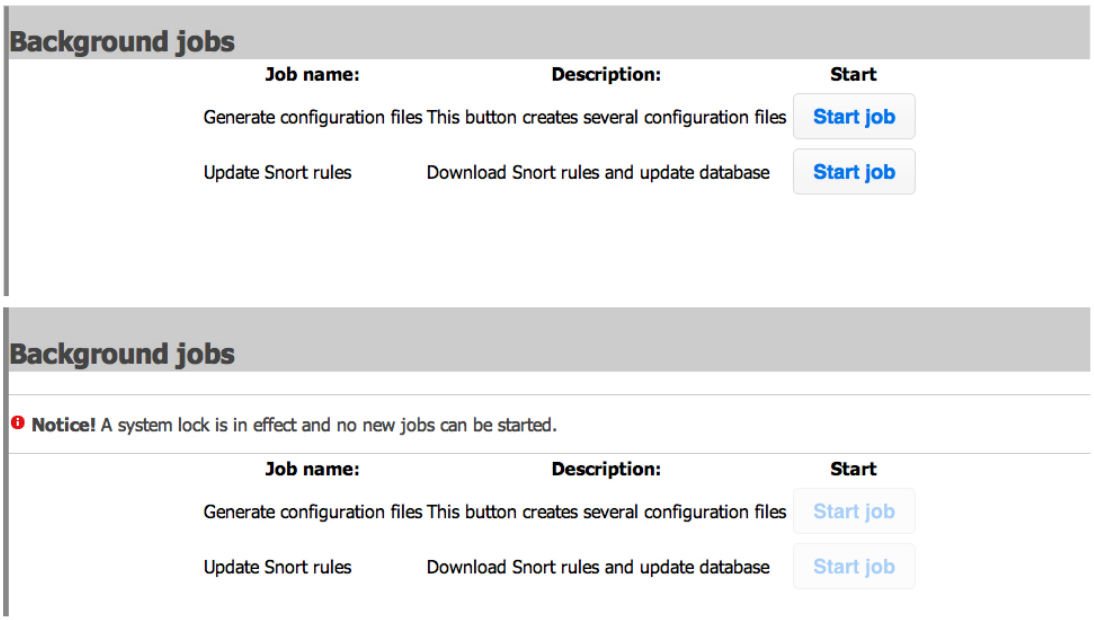
I de fleste tilfeller skal Snortmanager kjøre som en bakgrunnstjeneste. Dette betyr at en ny prosess som inneholder webserveren opprettes og separerer seg fra prossessen brukeren startet. Når Snortmanager.py fullfører eksekvering, vil den slette sin egen prosess, mens selve webtjenesten fortsatt kjører. Denne funksjonaliteten fungerer kun på operativssystemer som tilbyr metoden **fork()**, og vil derfor ikke fungere på f. eks. Microsoft® operativssystemer.

Når Snortmanager starter opp i daemonmodus opprettes, en PID (Proess Identifier)-fil, som inneholder prosessens unike identifikasjon på operativssystemet. Ved hjelp av denne identifikatoren kan administrasjonen lett finne tilbake til prosessen og avslutte den senere. Filen opprettes

i Snortmanagers*var/*-mappe.

4.9 Implementering av bakgrunnsjobber

Bakgrunnsjobber skal enkelt kunne startes i webgrensesnittet og utføres i egen tråd i samme prosess. Webgrensesnittet presenterer en liste med alle tilgjenglig jobber, og funksjonene som kan utføres er implementert i controllern og lagt i en Pythonliste. Brukeren får beskjed i grensesnittet om systemet er låst for endringer eller utførelse av nye jobber.



Figur 16: Listen over tilgjengelige bakgrunnsjobber i Snortmanager. Hvis systemlås er aktivert får man en beskjed.

I skjermbildet over ser man at brukeren får en liste over tilgjengelige bakgrunnsjobber. Brukeren kan da starte en bakgrunnsjobb ved å trykke på knappen til jobben som ønskes utført. Jobben må så godkjennes i en dialogboks, hvor brukeren også har mulighet til å avbryte jobben.

Hvis brukeren godkjenner starten av jobben, sendes data via en jQuery AJAX-funksjon til en metode i klassen *JobsController*. Denne verifiserer at innholdet i dataene er korrekt ved å benytte listen over tilgjengelige jobber nevnt over. Deretter vil den opprette en egen tråd ved hjelp av Pythons egen modul for trådhåndtering, Threading [10]

CherryPy er utviklet slik at nye tråder som opprettes på denne måten må avsluttes før prosessen kan avsluttes. Dette kan unngås, men er beholdt, ettersom det reduserer det risikoen for systemfeil og korrupte data.

Samtidig ser man fra bildet over at når systemlåsen er aktivert, vil sluttbrukeren bli presentert med en feilmelding, og knappene for å starte en jobb blir deaktivert. På denne måten får sluttbrukeren nødvendig informasjon om aktivitet, og Snortmanager har ekstra sikkerhet mot at

[10] http://docs.python.org/library/threading.html

nye jobber startes før andre er utført.

4.10 Planlagte oppgaver

Planlagte oppgaver er implementert med modulen Advanced Python Scheduler (APS), en mye brukt modul for håndteringen av planlagte jobber i Python. APS er ikke en del av Pythons standardbibliotek, og Snortmanager inneholder derfor en kompilert versjon under *Library*mappen.

APS er avhengig av å kjøre i egen tråd i prosessen, slik at det ikke begrenser eksekvering av øvrig kode. Den må også utføre noen operasjoner under oppstart og avslutning. Derfor vil CherryPy opprette en tråd som abonnerer på en egen kanal i CherryPys motorstruktur. Dette medfører at APS får beskjed når brukeren starter og stopper Snortmanagertjenesten, og kan utføre nødvendige opperasjoner som å avslutte jobber, stoppe scheduleren eller hente og lese inn planlagte oppgaver.

Rammeverket for planlagte oppgaver og APS ligger i mappen webapp/config/Schedule.py. Dette bryter med MVC-strukturen, siden planlagte oppgaver ikke er en del av model. Det er likevel en såpass sentral del av webapplikasjonen at det er en praktisk plassering. I denne modulen deklareres klassen *SnortScheduler()*, som arver fra CherryPys SimplePlugin. Dette baseobjektet brukes til interaksjon med CherryPys motor, og gjør at APS får tilgang til kanalen. CherryPy har nå mulighet til å gjøre et begrenset antall operasjoner på instansen av klassen.

Under initiering av SnortScheduler blir det gjort kall til moderobjektets constructor for å koble seg inn på CherryPys bus, hvor kanaler opprettes og beskjeder blir sendt. Deretter opprettes en *Scheduler*-instans fra APS. Denne utfører operasjoner som henting og utførelsen av planlagte oppgaver.

Planlagte oppgaver oppbevares i APS JobStores, og er kun en lagringsplass for planlagte oppgaver. Det finnes tre typer JobStores: Binærfil, mongoDB (noSQL) og SQLAlchemy. Ettersom sistnevnte allerede benyttes av Snortmanager, vil også planlagte oppgaver bli lagret i databasen med SQLAlchemy.

SnortScheduler-klassen inneholder metodene *start()* og *stop()* som benyttes når Snortmanager starter og avslutter CherryPy-motoren. Under oppstart vil instansen først koble til databasen for å hente ut jobber fra JobStore. Om ikke det jobber i databasen, vil det opprettes to jobber. Den første jobben som opprettes er oppdatering av Snortsignaturer som planlegges til klokken 09.00 hver dag, og en som produserer nye konfigurasjonsfiler klokken 09.00 hver dag.

Støtten for SQLAlchemy i APS er noe annerledes enn den utviklet for Snortmanager, ettersom det ikke benytter seg av Scoped Sessions. For å kunne bruke de samme objektene og instansene som resten av Snortmanager, inneholder konfigurasjonsfilen for databasen en egen klasse som arver fra APS' SQLAlchemyJobStore. Dette gjør at all funksjonalitet er tilgjengelig for scheduleren, men at konfigurering av JobStore forenkles. Dette fordi samme Engine og Metadata-instanser kan brukes uten store modifikasjoner av APS' klasser. Nedenfor vises hvordan JobStore er implementert i Snortmanager:

```
class ScheduleStore(SQLAlchemyJobStore):
    """ -- Docstring fjernet for rapporten -- """

    def __init__(self):
        """ -- Docstring fjernet for rapporten -- """
        SQLAlchemyJobStore.__init__(self, engine=engine,
        metadata=metadata, tablename='Schedule')
```

Sluttbrukere skal også kunne administrere planlagte oppgaver gjennom webgrensesnittet. Kontrolleren som håndterer denne funksjonaliteten er *JobsController*, som også er ansvarlig for bakgrunnsjobber. Denne importerer SnortScheduler-instansen, for så å benytte funksjonalitet i APS' Scheduler()-klasse til å lage, slette og endre planlagte oppgaver.

Figur 17: Grensesnittet viser at man får en oversikt over alle planlagte oppgaver, hvilken type de er og når de skal utføres.

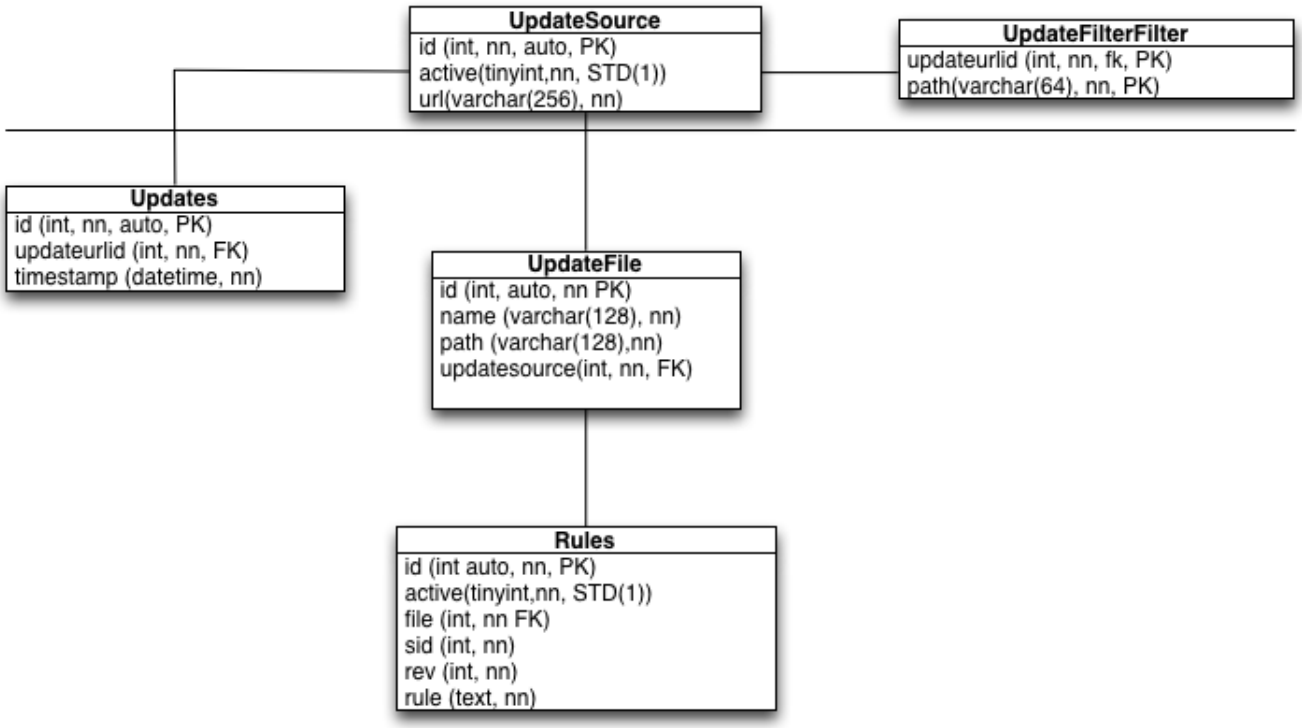
Når brukeren legger til nye jobber i grensesnittet, får den valget om hvilken type oppgave som skal lagres. Dette kan være intervall, dato/tidspunkt og cronmetoden. Deretter kan den velge mellom allerede etablerte bakgrunnsjobber. For dette prosjektet vil planleggeren kun kjøre bakgrunnsjobber definert i Snortmanager, men det er tilrettelagt for at man senere også kan kjøre eksterne oppgaver og tjenester.

Når jobben lagres vil kontrolleren for siden omformatere dataene fra brukeren til en Job-instanse som lagres i APS. Dette vil også bli skrevet til databasen av APS parallelt.

Det vil ikke være mulig å endre jobber i denne versjonen av Snortmanager. Operasjonene APS bruker for å endre prosjekter, er å fjerne det gamle og skrive et nytt. Dette, kombinert med at brukeren sjelden skal utføre denne oppgaven, har gjort at funksjonaliteten er deprioritert.

4.11 Organisering av Snortsignaturer

Snortsignatuer er essensielle for at Snortsensorer skal kunne oppdage skadelig aktivitet under nettverksanalyse, og det er derfor nødvendig at Snortmanager har en konsistent signaturdatabase. Ettersom prosessen for å oppdatere regler ble presentert i designdokumentet, vil kun databasestrukturen og lagring av signaturer presenteres her.



Figur 18: Databasediagram for oppdatering og lagring av regler.

Alle kilder Snortmanager skal bruke når den laster ned nye Snortsignaturer, ligger i tabellen *UpdateSource*. En kilde inneholder en aktiveringsstatus, og en URL som Snortmanager skal bruke til å laste ned filer fra.

Ettersom det skal være mulig å ignorere filer i en oppdatering, er tabellen *UpdateFileFilter* opprettet. Den inneholder en identifikator til kilden den tilhører, og en bane i oppdateringen som skal ignoreres.

Linjen i diagrammet illustrerer delen av løsningen som brukeren har direkte kontroll over. Alle tabellene under populeres og behandles automatisk når Snortmanager oppdaterer signaturer.

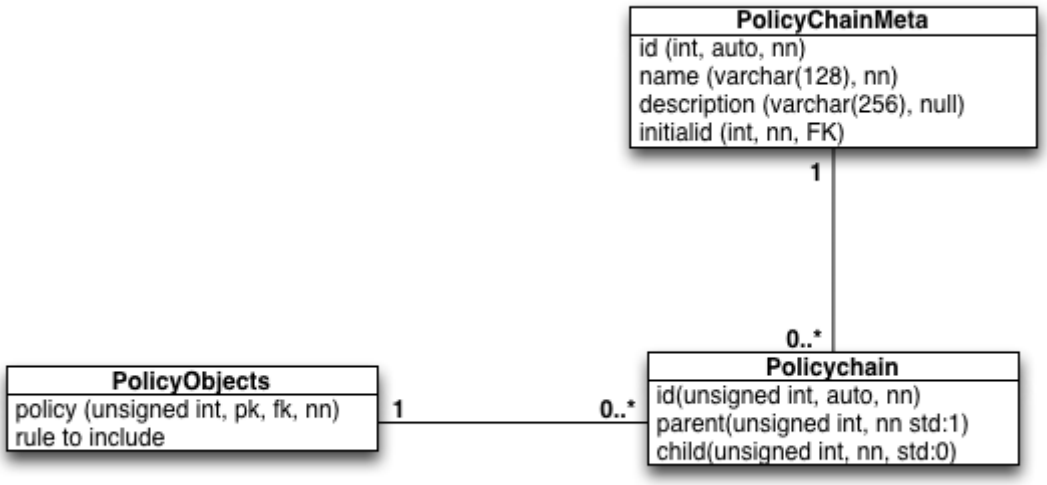
UpdateFile inneholder alle filer som Snortmanager har lest signaturer fra under en oppdatering. Når disse registreres under oppdateringen, legges også identifikatoren til kilden de er lastet ned fra i tabellen *UpdateSource*. Videre inneholder et filobjekt navnet på filen og banen den hadde til den originale filen som ble lest inn. Dette gjør at Snortmanager kan simulere en filstruktur

ved behov.

Hver enkelt signatur blir lagret i *Rules* tabellen. Ettersom en Snortsignatur kan ha mange forskjellige valgfrie direktiver, er det valgt å bare inkludere de mest nødvendige attributtene i databasen. En signatur vil knyttes mot filen den ble lest inn fra med fjernnøkkelen *file*. Signaturer identifiseres av databasen og Snortmanager med sin unike identifikator, men sluttbrukere vil forholde seg til SID og rev. *Rule* atributtet inneholder hele Snortsignaturen som er lest inn. Dette forenkler lagring og innlesning betraktelig.

Tabellen *Updates* inneholder bare en logg over oppdateringer. For hver enkelt kilde som brukes i en oppdatering, vil Snortmanager skrive en oppføring til denne tabellen.

4.12 Implementering av policy



Figur 19: Illustrasjon av policyene lagret i databasen.

Fordi alle data i Snortmanager skal være lageret i databasen, var det nødvendig med en måte å få kjedekonstruksjonen uten at det gikk på bekostning av dataene og databasenormaliseringen. Det ble derfor valgt å organisere policydata i tre forskjellige tabeller.

Policy: Inneholder alle datene som definerer en hel policy. De er id, navn og en beskrivelse. En beskrivelse er ikke nødvendig, men anbefalt. Et PolicyChainMeta-objekt er unikt for hver enkel policy. Skjer det en korrupsjon i denne, vil ikke policyen kunne brukes av Snortmanager.

PolicyKjede: Selve kjeden bygget opp i databasen. Det vil finnes et element i denne tabellen for hvert enkelt policyobjekt som benyttes av én enkelt policykjede, kjedeelementet. Policyen som elementet skal brukes i defineres med at attributtet *chain_id*, settes til policyens id fra PolicyChainMeta-tabellen.

Hvert enkelt element inneholder en link til sitt forrige og neste element i kjeden. Dette er et heltall som identifiserer en rad i den samme tabellen. Det første elementet i kjeden identifiseres med at parent er satt til 0. Det siste elementet identifiseres identifisres med at child er satt til 0. Ved å organisere strukturen på denne måten, kan kjeden traverseres begge veier. Det forenkler også sorteringen av elementer, ettersom man kan forandre ordenen på elementene i en kjede, uten at det påvirker selve policyobjektet eller andre policyer objektet er brukt i.

Objektet som det enkelte kjedeelementet skal bruke ligger i tabellen *PolicyObject*, og kalles fortsatt policyobjekter. Hvert enkelt policyobjekt inneholder først en type som definerer om objektets innhold skal skrives i starten (prepend), midten (contents) eller slutten (append) av en konfigurasjonsfil. Bare objekter av typen contents inneholder policyregler om Snortsignaturer. De øvrige typene vil brukes til direktiver og konfigurasjonsdata.

Det neste attributet til policyobjektet er active, som ikke brukes av Snortmanager nå. Dette er lagt til for å senere kunne deaktivere policyobjekter i flere kjeder ved behov. Til slutt er attributen contents, som inneholder regler og øvrig konfigurajonsdata brukt under produksjon av konfigurasjonsfil. Dette lagres i form av en tekstblokk, og har ingen plassbegrensing.

4.13 Implementering av systemlås

Systemlås er implementert med klassen *SystemLock*, og ligger i `__init__.py` i *jobs*-mappen. Denne klassen initieres i samme fil, og instansen heter *ServiceLock*. Instansen blir importert av kontrollerne til webapplikasjonen og bakgrunnsjobbene.

Når en bakgrunnsjobb starter, eller sluttbruker ønsker å gjøre endringer vil Snortmanager kontrollere om en systemlås er aktivert. Hvis filen `DB_LOCK.pid` eksisterer i mappen *var/*, er en systemlås aktivert. Dette gjøres av metoden **lock_system()**, som benytter seg av metoden **is_locked()** for å kontrollere om Snortmanager allerede er låst. Hvis Snortmanager er låst, får brukeren beskjed om dette, og operasjonen blir avbrutt.

```
class SystemLock:
    """ Locks the system so users can't start new jobs or write to database """
    pid_file = os.path.join(VARDIR, 'DB_LOCK.pid')

    def is_locked(self):
        """ Return the status of the lock. True if system is locked."""
        try:
            f = open(self.pid_file, 'r')
            f.close()
            return True
        except IOError as e:
            return False

    def lock_system(self, name):
        """ Creates a system lock and locks database for changes. """
        if self.is_locked(): # Kontrollerer om database er laast
            raise SystemLockedException(self.process_id, self.process_name)

        try: # oppretter filen som indikerer laas
            lock_file = file(self.pid_file, 'w')
            lock_file.write(os.getpid())
            lock_file.close()
        except IOError:
            print 'Error while creting PID'
```

Utsnittet over viser at systemet sjekker om det er låst ved å forsøke å åpne PID-filen. Hvis filen ikke eksisterer, vil en exception oppstå, og metoden returnerer at filen ikke eksisterer. Denne måten å kontrollere om filer eksisterer på er anbefalt av Python i dokumentasjonen deres.

Hvis systemet ikke er låst, vil Snortmanager fortsette eksekveringen. Hvis operasjonen innebærer å starte en bakgrunnsjobb, vil Snortmanager benytte den innebygde loggingløsningen for å registrere at en lås aktiveres. Deretter vil filen `DB_LOCK.pid` opprettes og IDen til prosessen som låser systemet blir skrevet til den. Dette gjøres for å kunne kontrollere at man kan spore opp hvilken prosess som opprettet filen.

```
    def unlock_system(self):
       """ Unlocks the system so changes can be written to database. """
        if self.is_locked():
            self.process_name = ''
            self.process_id = 0
            os.remove(self.pid_file)
```

Når bakgrunnsjobben er utført, skal systemlåsen fjernes og Snortmanager åpnes igjen. Dette

gjøres ved å fjerne `DB_LOCK.pid`, og loggføre at låsen er fjernet.

4.14 Implementering av sensor

Håndtering av sensorer skal skje ved hjelp av webgrensesnittet, og kan aksesseres fra menyen på venstre siden under Sensor". Her får brukeren opp en liste over sensorer som allerede er registrert i databasen. Informasjonen i listen blir hentet fra databasen Snortmanager og tabellene *Sensor*, *SensorLocation*, og *PolicyChain*, ved at det kjøres en spørring til databasen. Her hentes så data fra attributtene IP, location, policychain og description, hvor location og policychain inneholder en ID hver som representerer poster i tabellene *SensorLocation* og *PolicyChain*. Resultatet av denne spørringen blir så presentert av Snortmanager i grensesnittet som en tabell.

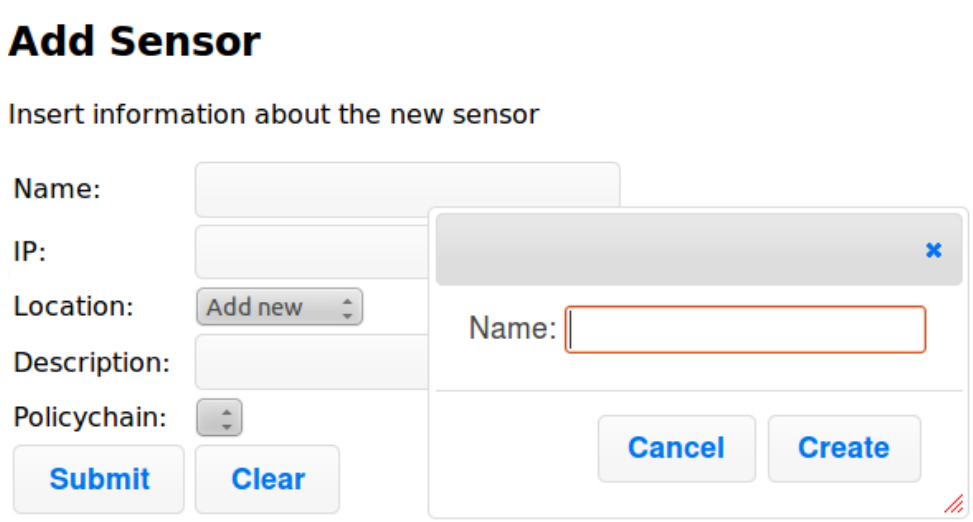
4.14.1 Opprettelse av ny sensor

I undermenyen "Add Sensorfår brukeren muligheten til å legge til en ny sensor. Her blir brukeren presentert med et utfyllingsskjema, som finnes i HTML-filen "addsensor.html". Etter å ha oppgitt nødvendig informasjon, blir informasjonen sendt til kontrolleren *SensorController* i *sensor.py*-skriptet.

Brukeren skal fylle ut navnet på sensoren, IP-adressen, velge lokasjonen sensoren befinner seg på, samt skrive en kort beskrivelse av sensoren. Siden sensorene skal sorteres etter lokasjon, vil det bli brukt en egen tabell for dette. Her vil det bare finnes ett enkelt tilfelle av hver lokasjon. Brukeren vil få valget mellom å bruke en eksisterende lokasjon, eller opprette en ny. Denne funksjonaliteten blir benyttet ved å presentere en nedtrekksmeny med en liste over allerede registrerte lokasjoner. I samme nedtrekksmeny vil brukeren også ha mulighet til å legge til en ny lokasjon.

Ved opprettelse av ny lokasjon, vil det blir kjørt en sjekk på om den nye lokasjonen finnes fra før. Hvis den gjør det, vil den eksisterende lokasjonen benyttes. Så legges informasjonen i tabellen *Sensor*, *SensorLocation* og *PolicyChain*, ved hjelp av funksjonen **getsensordata()**.

Bruker vil deretter få en status om dataen ble lagt til i databasen, samt informasjon om hva som ble lagt til.



Figur 20: Eksempel på skjema ved opprettelse av sensor og lokasjon

Under vises en kodesnutt fra sensor.py, som mottar data fra skjema og legge til medsendt data til database. Det vises også et eksempel fra dbconfig.py, som viser hvordan klassen Sensor oppretter tabellen Sensor i databasen ved hjelp av SQLAlchemy.

```
@cherrypy.expose
@cherrypy.tools.jinja(template='content.html')
def getsensordata(self, addName, addIp, sensor_location, addDescription):

    try:
        check_location = Session.query(SensorLocation).filter(
        SensorLocation.name == sensor_location.capitalize()).one()
    except NoResultFound, e:
        new_location = SensorLocation(sensor_location.capitalize())
        Session.add(new_location)
        Session.flush()

        Session.refresh(new_location)
        locationId=new_location.id
        Session.flush()

         new_sensor = Sensor(addName, addIp, locationId, addDescription)
        Session.add(new_sensor)
        Session.flush()
     else:
        existing_location = check_location.id
        new_sensor = Sensor(addName, addIp, existing_location, addDescription)
        Session.add(new_sensor)
        Session.flush()

    return ({ 'page_title': 'Add Sensor', 'secondmenu': self._create_sidemenu() })
```

Eksempel på opprettelse av sensor og lokasjon

Kodesnutten under viser hvordan klassen Sensor mottar og skriver data til databasen ved

hjelp av SQLAlchemy.

```
Base = declarative_base() # Definerer et base objekt

class Sensor(Base):
    """ Klasse for Sensor tabellen i databasen. """
    __tablename__ = 'Sensor' # Setter navnet paa tabellen i databsen
    id = Column('id', Integer, primary_key = True, autoincrement = True)
    name = Column('name', String(50), nullable = False)
    ip = Column('ip', String(20), nullable = False)
    location = Column('location', INTEGER(unsigned = True),
    ForeignKey(SensorLocation.id), nullable = False)
    description = Column('description', String(140), nullable = False)
    policychain = Column('policychain', Integer, nullable = False)

    def __init__(self, name, ip, location, description, policychain = 0):
        """ Konstruktor slik at man kan opprette i databasen. """

        self.name = name
        self.ip = ip
        self.location = location
        self.description = description
        self.policychain = policychain

    def __repr__(self):
        """ Representasjon av objektet """

        return "<Sensor('%i','%s',␣'%s',␣'%s',␣'%s')>" % (self.id,
         self.name, self.ip, self.location, self.description)
```

4.14.2 Endre sensor

På siden Endre sensor"vil brukeren kunne endre data i en valgt sensor. Det blir presentert en nedtrekksmeny med en liste over registrerte sensorer. Brukeren velger IP-adressen til den sensoren det skal bli utført endringer på, og trykker på Edit". Deretter fyller brukeren ut ny informasjon i feltene som skal endres. Denne funksjonaliteten blir presentert av skriptet *editsensor.html*.

Den nye dataen blir så sendt til funksjonen **editsensor()** i sensor.py-skriptet. Denne funksjonen vil da finne sensoren valgt av brukeren. Selve sensoren har en ID som primærnøkkel, så selv om IP-adresse endres, vil sensoren fortsatt ha samme ID.

Grunnen til at sensor identifiseres i system på en egen ID istedet for IP-adressen, er at Snortmanager da vil føre en mer oversiktlig og komplett logg. Når funksjonen har funnet riktig sensor, vil den erstatte postene data utfylt av bruker.

4.15 Implementering av logging

Logging er et viktig verktøy ikke bare under utvikling, men også i testfasen. I testfasen trenger man informasjon som beskriver gangen steg for steg, som gjør det lettere å feilsøke. Logging kan brukes som en detaljbeskrivelse av programmets gang. Dette fører til at du vet hvor langt programmet har kommet før det feiler. Logging er også viktig i den hverdagslige bruken av programmet. Alt av hendelser gjort av bruker eller systemet skal logges. Loggingen gjør det oversiktlig å følge aktiviteter og endringer. Som beskrevet i kravspesifikasjonen har gruppen lagd to forskjellige loggfunksjoner; én funksjon for logging av brukeraktivitet, og én for logging av systemaktivitet.

4.15.1 Logging av handlinger gjort av bruker

Denne loggeren tar for seg endringer i løsningen eller i databasen utført av bruker. Eksempler på dette kan være å legge til en ny sensor, endringer gjort på en eksisterende sensor, eller opprettelse av nytt policyobjekt.

Nå loggeren legger til en ny sensor, bruker den loggfunksjonen log_to_database() i filen dbconfig.py. Funksjonen blir tilkalt i den eksekverende koden, som sender med data til loggfunksjonen om hva som er blitt gjort. Loggfunksjonen oppdaterer så databasetabellen *EventLog* med den motatte dataen, hvor den motatte dataen er endringer i databasen.

Data som blir lagt til i EventLog er ID, alvorlighetsgrad, dato/tid, navnet på modulen funksjonen ble benyttet i og en beskrivende tekst om hendelsen. *ID* blir opprettet først, og autoinkrementert etterhvert som tabellen blir oppdatert. ID fungerer også som primærnøkkel.

Deretter blir *severity* oppdatert med alvorlighetsgraden til hendelsen. For å minimere kodemengden blir det brukt en "ordbok"(dictionary). Dette er en datatype som mottar et tall. Tallene symboliserer tekstverdier, i dette tilfellet ser ordboken slik ut:

.. code:: python

severityDict=0:'debug',1:'info', 2:'warning', 3:'error', 4:'critical'

Deretter blir *timestamp* oppdatert, denne autogenereres med klokkeslettet til systemet programmet kjører på. Så blir navnet på modulen skrevet til database, som gjøres ved å benytte en spesiell variabel i Python kalt __name__. Her sendes modulnavnet og hvilken mappe den ligger i, med til databasen. Til slutt blir *text* oppdatert med en kort beskrivelse av hendelsen.

```
mysql> select * from EventLog;
+----+----------+---------------------+--------+---------------------------------+
| id | severity | timestamp           | module | text                            |
+----+----------+---------------------+--------+---------------------------------+
|  1 | Warning  | 0000-00-00 00:00:00 | Sensor | Could not write data to database |
|  2 | Warning  | 0000-00-00 00:00:00 | Policy | Failed - adding new policy       |
|  3 | Error    | 0000-00-00 00:00:00 | Sensor | Could not reach database         |
+----+----------+---------------------+--------+---------------------------------+
3 rows in set (0.00 sec)
```

Figur 21: Eksempel på tabellen EventLog i database, ved logging til database.

```
class EventLog(Base):
    __tablename__ = 'EventLog'

    id = Column('id', Integer, primary_key=True, autoincrement=True)
    severity = Column('severity', String(50), nullable = False)
    timestamp = Column('timestamp', DateTime, nullable = False)
    module = Column('module', String(50), nullable = False)
    text = Column('text', Text, nullable = False)

    severityDict={0:'debug',1:'info', 2:'warning', 3:'error', 4:'fatal'}

    def __init__(self, severity, module, text):
        try:
            if type(severity) is not int:
                raise TypeError('Severity is not a int')
```

```
            if severity < 0 or severity > len(self.severityDict):
                raise ValueError('The severity has to be between 0 and',
                 str(len(self.severityDict)))
            self.severity = severity
            self.module = module
            self.text = text
            self.timestamp = datetime.now()
        except TypeError as e:
            print e
        except ValueError as e:
            print e



    def __repr__(self):
        severity = self.severityDict[self.severity]
        return "<EventLog(%i, '%s', '%s', '%s')>" % (self.id,
         severity, self.module, self.text)
```

4.15.2 Logging av systemaktivitet

I motsetning til logging av brukeraktivitet, skal systemaktivitet logges til fil. Dette blir gjort for å spare plass, og for å skape en ryddigere logg. For å logge systemaktivitet til fil, bruker gruppen Pythons egen modul for logging. For å utnytte denne på en mest mulig effektiv måte, skal et eget skript ta seg av selve loggingen. Dette vil fungere slik at enhver modul vil kunne importere skriptet ved hjelp av from log import debug, info, warning, error, critical". Ved å importere disse snarveiene", vil loggingen enkelt kunne implementeres i modulene.

For å benytte seg av funksjonen i skriptene, skriver man for eksempel "warning("Meldingen du ønsker å sende med")". Denne dataen blir så sendt til skriptet log.py. Her tilkalles Pythons integrerte loggfunksjoner, filehandler og loggformater. Filehandles jobber er å eksportere data til fil. Før dataen når filen, blir den sendt til funksjonen formatter, som organiserer den medsendte dataen til riktig output.

En logginstans vil se slik ut: **[2012-05-07 20:05] DEBUG sensor: starter hjemmeside**, hvorav formatet [Dato/tidspunkt] "AlvorlighetsgradModul": "Melding".

Under vises koden for logging til fil i Python

```
import logging
import sys


DEBUG_LOG_FILENAME = os.path.join(rundir,'logs','SMLOG.log')

#Formatet paa utskriften til fil
formatter = logging.Formatter('[%(asctime)s] %(levelname)s %(module)s: %(message)s')

#Valg av filhandler for skriving til fil
fh = logging.FileHandler(DEBUG_LOG_FILENAME)
fh.setLevel(logging.DEBUG)
fh.setFormatter(formatter)


mylogger = logging.getLogger('MyLogger')
mylogger.setLevel(logging.DEBUG)
```

```
mylogger.addHandler(fh)

#"Snarveier" for bruk av funksjonene i andre skript
debug = mylogger.debug
info = mylogger.info
warning = mylogger.warning
error = mylogger.error
critical = mylogger.critical
```

4.16 Alvorlighetsgrad for logging av hendelser.

Det er flere alvorlighetsgrader innenfor logging. Under nevnes standard alvorlighetsgrader i et Python-system.

Debug: Det laveste nivået når det gjelder logging. Blir som regel brukt til detaljerte beskrivelse av programflyten.

Info: Her defineres ofte input og output gjort av brukerinteraksjon mot løsningen. Info blir også ofte brukt til å verifisere at operasjoner fungerer som de skal.

Warning: Blir brukt for å varsle om uønskede eller uventede feil. Warning er ikke en fatal hendelse; løsningen vil fortsatt kjøre, men administrator av systemet bør ta denne varselen alvorlig, og undersøke hendelsen. Eksempel på hendelse for warning; Korrupt data, manglende data eller lite diskplass.

Error: Den nest alvorligste hendelsen. I motsetning til warning, er en error-hendelse fatal; ikke nødvendigvis for løsningen, men for operasjonen hendelsen skjedde i. Eksempler på error-hendelser kan være at Snortmanager ikke får tak i en fil eller manglende tilkobling mot eksterne enheter.

Critical: Alvorligste hendelsen. Blir også kalt fatal", da denne hendelsen inntreffer ved systemsvikt og ved store tap av data.

4.17 Dokumentasjon av kildekode

Snortmanager er åpen kildekode, som betyr at nye utviklere kan være interessert i å videreutivkle eller vedlikeholde kildekoden. Dette krever utfyllende dokumentasjon.

Kildekoden til Snortmanager skal dokumenteres med Docstring [11], som er standard for Python-prosjekter. I likhet med andre velkjente kodedokumentasjonsløsninger i andre programmeringsspråk (som Javadoc [12]), skrives dokumentasjon direkte i kildekoden, slik at utviklere kan bruke den direkte. Docstring kompileres også sammen objektet, slik at får den sitt eget attributt (__doc__) i modulen, klassen eller metoden. Nye utviklere kan da bruke Pythons egen **help()**-metode til å lese dokumentasjon. Det kan også eksporteres til andre formater, som dokumentasjonsverktøyet Sphinx [13].

Docstrings fungerer slik at man i en constructor eller metodedeklarasjon skriver en tekststreng som starter og slutter med tre gåsetegn ("). En docstring kan være både en enkel linje (one-line), eller være skrevet utfyllende på flere linjer (multiline).

Snortmanager forholder seg til retningslinjene i PEP257 [14] (Python Enhancement Proposals),

[11] http://www.python.org/dev/peps/pep-0258/
[12] http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html
[13] http://sphinx.pocoo.org/
[14] http://www.python.org/dev/peps/pep-0257/

som omhandler anbefalt bruk av Docstring. Alle moduler er dokumentert i sin `__init__.py`. Videre blir hver enkelt klasse nøye beskrevet og forklart med en multiline docstring. De fleste metoder blir beskrevet med en multiline docstring, bortsett fra metoder hvor det er åpenbart hva som utføres. Da brukes en one-line docstring.

5 Testing og kvalitetssikring

Testing er en svært viktig del i sluttfasen ved utvikling av software. I denne fasen skal man gå gjennom kildekode og benytte enhver funksjonalitet for å forsikre seg om at stabiliteten, integriteten, og sikkerheten til produktet er som det skal være i henhold til kravspesifikasjonen utviklet i starten av prosjektet.

Ettersom Snortmanager skal tas i bruk i et produksjonsmiljø etter endt prosjekt er det viktig at gruppen forsikrer seg om at funksjonalitet fungerer i hennhold til spesifikasjoner, og at systemet ikke inneholder feil som påvirker negativt.

5.1 Testmetode

Gruppen har som tidligere nevnt valgt en blanding av scrum og inkrementell modell for prosjektet, og utviklingen har skjedd modulbasert, med større tester etter sammenslåingen av modulene, for en mer helhetlig test av systemet, og for å sikre at integrasjonen av modulene har vært vellykket. Denne modulbaserte utviklingen har ført til at testingen har forekommet hyppig, ettersom utviklingen har skjedd stykkvis, og man underveis tar seg tid til grundig testing av enkeltmoduler. Den måten å utvikle på har vist seg svært positiv for gruppen, da vi har måtte lære mye av skript og programmeringsspråkene underveis, har denne metoden sørget for mindre feil underveis i utviklingen.

5.2 Punkter som ble vektlagt under testingen

Med flere komponenter som skal samarbeide, og mye informasjon er det viktig å prioritere. Før testingen ble flere kriterier vurdert, og gruppen kom frem til å vektlegge følgende:

- Funksjonaliteten til Snortmanger, og at disse fungerer.
- Kildekodes lesbarhet og konsistens.
- Kompatibilitet og portabilitet med plattformer og løsninger.

5.3 Brukergrensesnittet

I testingen av brukergrensesnittet ble designet og utformelsen av siden testet, og vurdert. Designet på siden skal være funksjonelt og intuitivt. Med dette mener gruppen at alt av funksjonalitet skal fungere i henhold til kravspesifikasjonen, alle linker, og funksjonalitet som dropdownmenyer og andre grafiske designelementer skal være operative. Gruppen gikk stykkvis igjennom de forskjellige sidene og sub-sidene, samt igjennom hver eneste link og funksjon for å forsikre seg om at alt fungerte.

5.4 Funksjonalitet

Testingen av funksjonaliteten ble gjort på samme måte som testingen av designet, å stykkvis gå igjennom funksjonene til systemet, hovedsaklig funksjonaliteten som kjører i bakgrunnen.

5.5 Kildekode

Under testingen av kildekode ble kodens integritet og effektivitet testet. Om den koden vi utviklet var den beste og mest effektive måten å gjøre det på.

5.6 Kompatibilitet

Systemet vi ble også testet med tanke på kompatibilitet. Her testes det om systemet vil fungere på tvers av plattformer og programmer, nærmere bestemt operativsystemer og nettlesere. Ved testing av kompatibilitet ble også alle overnevnte punkter testet gang på gang, grunnet at gruppen måtte forsikre seg om at all funksjonalitet fungerte i flere settinger.

5.7 Testing av moduler

Snortmanager er bygget opp av forskjellige moduler som tilbyr funksjonalitet, og disse testes individuelt. Denne måten er mer fokusert, raskere og det gjør det enklere å rette feil.

5.7.1 Nedlastning og oppdatering av signaturer

Disse testene var omfattende, ettersom det er flere suksessfaktorer som kan være vanskelige å måle. For oppdatering av signaturer, skal følgende testes:

- Hvor mange regler som er lest inn etter hver kjøring.
- Hvor lang tid jobben tar.

Det publiseres ikke antall signatur i oppdateringer som lastes ned, derfor må det testes at det ikke skjer noen forrandringer i antall ved oppdatering. Oppdragsgivers database med signaturer kunne ikke brukes som referanse, ettersom de benytter kilder som krever betalt abonnoement. For dette prosjektet kunne utviklerne bare bruke regler som tilbys gratis.

Det er også viktig at Snortmanager laster ned regler og oppdaterer databasen raskt. Derfor har gruppen tatt tiden mellom oppdateringens start og slutt. Resultatene brukes for å sammenligne hvor lang tid en oppdatering tar, både når databasen populeres med regler første gang, og ved senere oppdateringer.

Testen av signaturoppdatering avhenger av kildene som benyttes. Hvis man laster ned offisielle signaturer fra Snort, kan testen kun utføres hvert 15. minutt på grunn av tidsrestriksjoner. For å omgå dette ble en lokal kopi av signaturene lagret i testnettet, under tidligee tester. Senere i prosjektet ble offisielle signaturer brukt.

Modulen for å oppdatere regler var en av de tidligste modulene som ble utviklet, som gjorde at det måtte utvikles uten webgrensesnitt. Kommandolinjen ble derfor brukt for å teste oppdateringen av signaturer. Deretter ble administrasjonsverktøyet for SQL databaser **mysql** brukt.

Alle testene ble kjørt mot to offisielle kilder som tilbyr gratis signaturer. Den første kilden er Snorts offisielle signaturer, den andre er Emerging Threat. Før testene ble utført første gang ble alle dataene fra nødvendige tabeller for signaturer fjernet. Dette ble gjort ved å kjøre MySQL-kommandoen **truncate** mot tabellene med signaturer (Rules), registrerte filer (UpdateFiles) og oppdateringsloggen (Updates).

```
mysql> TRUNCATE Rules; TRUNCATE UpdateFile; TRUNCATE Updates;
Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>
```

Figur 22: Tabellen for signaturer er tømt, og klar for test.

Alle testene utført fra kommandolinjen ble utført sammen med Unix-kommandoen **date**, som viser nåværende data og tidspunkt. Dette vil vise nøyaktig tidspunkt før og etter fullført oppdatering. Man vil da komme frem til nøyaktig hvor lang tid utføringen av jobben tok.

```
mysql> SELECT * FROM Updates;
+----+-------+---------------------+
| id | urlid | timestamp           |
+----+-------+---------------------+
|  1 |     1 | 2012-05-14 10:48:51 |
|  2 |     2 | 2012-05-14 10:49:13 |
+----+-------+---------------------+
2 rows in set (0.00 sec)

mysql> SELECT now() as Tid, count(*) FROM Rules;
+---------------------+----------+
| Tid                 | count(*) |
+---------------------+----------+
| 2012-05-14 10:51:04 |    35293 |
+---------------------+----------+
1 row in set (0.00 sec)

mysql>
```

Figur 23: Den første testen kjørt fra kommandolinjen, med start- og sluttidspunkt.

Skjermbildet over viser at jobben er utført med en database tømt for signaturer. Testen viser at oppdateringen kun tok noen få sekunder. Samtidig viser bildet under under at 35294 ble lagt

til databasen.

```
mysql> SELECT * FROM Updates;
+----+-------+---------------------+
| id | urlid | timestamp           |
+----+-------+---------------------+
|  1 |     1 | 2012-05-14 10:48:51 |
|  2 |     2 | 2012-05-14 10:49:13 |
+----+-------+---------------------+
2 rows in set (0.00 sec)

mysql> SELECT now() as Tid, count(*) FROM Rules;
+---------------------+----------+
| Tid                 | count(*) |
+---------------------+----------+
| 2012-05-14 10:51:04 |    35293 |
+---------------------+----------+
1 row in set (0.00 sec)

mysql>
```

Figur 24: 35294 signaturer er lagt til databasen.

I databasen er det registrert to oppdateringer, én for hver kilde med tidsspunktet oppdateringen ble registrert. Før neste test kan utføres, må man vente minst 15 minutter på grunn av restriksjonene nevnt tidligere.

Den andre testen er nesten identisk med den første, men databasen blir ikke tømt for signaturer.

```
[root@TestServer1 KodeBase]# echo START `date`; python Snortmanager.py -
START Mon May 14 10:48:49 CEST 2012
Starting download of Rules
STOPP Mon May 14 10:49:25 CEST 2012
[root@TestServer1 KodeBase]# echo START `date`; python Snortmanager.py -
START Mon May 14 11:09:37 CEST 2012
Starting download of Rules
STOPP Mon May 14 11:12:55 CEST 2012
[root@TestServer1 KodeBase]#
```

Figur 25: Andre test er utført, og tok noe lenger tid.

Når andre test er fullført, ser man, som forventet, at jobben har brukt litt mer tid på å fullføre. Dette skyldes at eksisterende signaturer skal hentes, kontrolleres og sammenlignes med

signaturene fra den eksterne kilden. Kontrollen vil sjekke dataene i begge signaturene, eventuelt deaktivere den gamle og legge til den nye i databasen. Prosessene er omfattende, men oppdateringen tar ikke mye lenger tid, selv om det er over 35 000 regler.

I databasen, ser man at ingen nye regler er lagt til i databasen. Dette viser at ingen feil har oppstått under oppdateringen, og ingen nye regler er lagt til databasen. At ingen duplikater av signatuerer finnes i databasen er kritisk for at Snortmanager skal fungere normalt, og ikke bruke unødvendig plass i databasen.

```
mysql> SELECT * FROM Updates;
+----+-------+---------------------+
| id | urlid | timestamp           |
+----+-------+---------------------+
|  1 |     1 | 2012-05-14 10:48:51 |
|  2 |     2 | 2012-05-14 10:49:13 |
|  3 |     1 | 2012-05-14 11:09:39 |
|  4 |     2 | 2012-05-14 11:11:55 |
+----+-------+---------------------+
4 rows in set (0.00 sec)

mysql> SELECT now() as Tid, count(*) FROM Rules;
+---------------------+----------+
| Tid                 | count(*) |
+---------------------+----------+
| 2012-05-14 11:15:03 |    35293 |
+---------------------+----------+
1 row in set (0.00 sec)

mysql>
```

Figur 26: Det er nå registrert to nye oppdateringer i databasen.

5.7.2 Testing av planlagte oppgaver

Det viktigste med testing av planlagte oppagver er:

- Tjenesten starter og stopper sammen med CherryPy
- Alle jobber leses inn og blir utført på korrekt tidspunkt
- At nye jobber kan legges inn gjennom grensesnittet

At scheduler-tjenesten starter opp sammen med CherryPy testes først og fremst ved å kontrollere de sentrale loggene etter at tjenesten er startet. Deretter venter man til det definerte

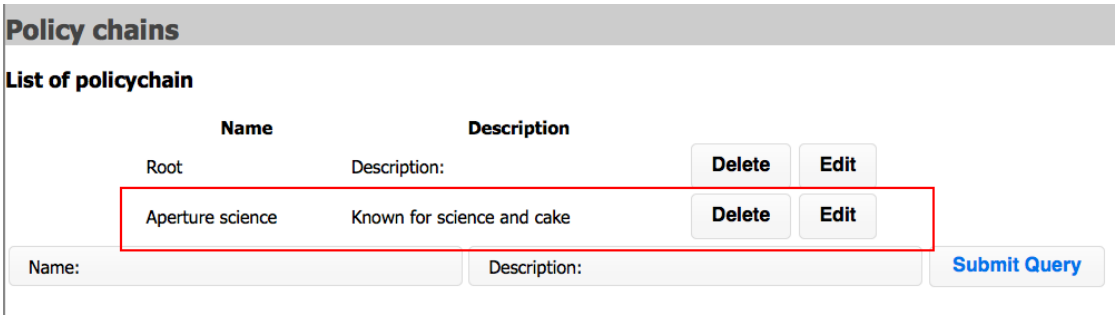
tidspunktet for jobbene, og sjekker i logger og databasen at jobben er riktig utført.

For å kontrollere at jobber blir lest inn riktig fra JobStore, ser man på listen over planlagte jobber i webgrensesnittet. Dette hentes ikke direkte fra databasen, men fra minnet til scheduleren. Hvis planlagte oppgaver ligger i listen, har man bekreftet at de er lest inn riktig.

5.8 Opprettelse av policy

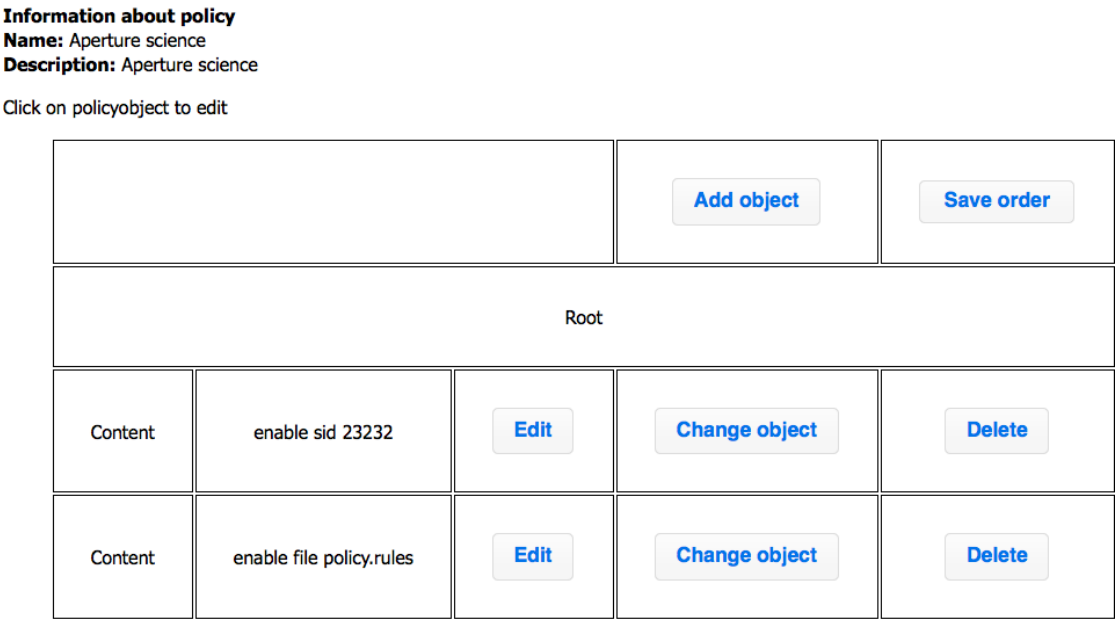
En policy kan oppfattes som en komplisert enhet med mange samarbeidene komponenter. De er kritiske for produksjon av konfigurasjonsfiler. Det må derfor testes at en bruker lett kan opprette, endre og evenutelt slette en policy, uten at det får konsekvenser for de øvrige delene av systemet.

Testen starter med å opprette en ny policy som heter `Aperture Science`. I bildet under vises den nye policyen markert i listen.



Figur 27: Policien `Aperture Science` er lagt til, og er aktiv i Snortmanager.

Videre må man trykke på **Edit** til høyre for policyen for å kunne redigere den. Man blir da presentert med en policy uten innhold. Her gjøres flere operasjoner: Først opprettes det to nye objekter. Deretter legges det til et allerede eksisterende objekt, og til slutt blir et av objektene slettet. Til slutt ser policyen ut som på bildet under.



Figur 28: Etter at testen er utført ser innholdet i `Aperture Science` slik ut.

I databasen kan det dobbeltsjekkes at policystrukturen er korrekt med følgende SQL spørring:

```
SELECT c.id, c.chain_id, o.contents, c.parent, c.child FROM PolicyChain as c
INNER JOIN PolicyObject as o ON c.policyobject_id = o.id
WHERE c.chain_id = (SELECT id FROM PolicyChainMeta where name = 'Aperture Science');
```

Resultatet av spørringen viser at strukturen av policykjeden ble korrekt, som vist skjermbildet under. Det første elementet i kjeden har parent = 0. Deretter er det lett å følge strukturen i kjeden ved å gå til det første elementets child. Dette elementet er det siste i kjeden, med en child = 0.

```
mysql> SELECT c.id, c.chain_id, o.contents, c.parent, c.child  FROM PolicyCha
erture Science');
+----+----------+--------------------------+--------+-------+
| id | chain_id | contents                 | parent | child |
+----+----------+--------------------------+--------+-------+
| 32 |       13 | enable sid 23232         |      0 |    33 |
| 33 |       13 | enable file policy.rules |     32 |     0 |
+----+----------+--------------------------+--------+-------+
2 rows in set (0.00 sec)
```

Figur 29: Resultatet av SQL-spørrigen utført tidligere viser at strukturen til policyen er korrekt.

I listen over policyer deaktiveres den nye policyen, ved å trykke på Delete. Policien er nå fjernet fra grensesnittet, men policy og struktur er fortsatt tilgjengelige i databasen.

Figur 30: Viser at policy er fjernet fra listen, men fortsatt er tilgjengelig i databasen.

5.8.1 Test av produksjon av konfigurajonsfiler

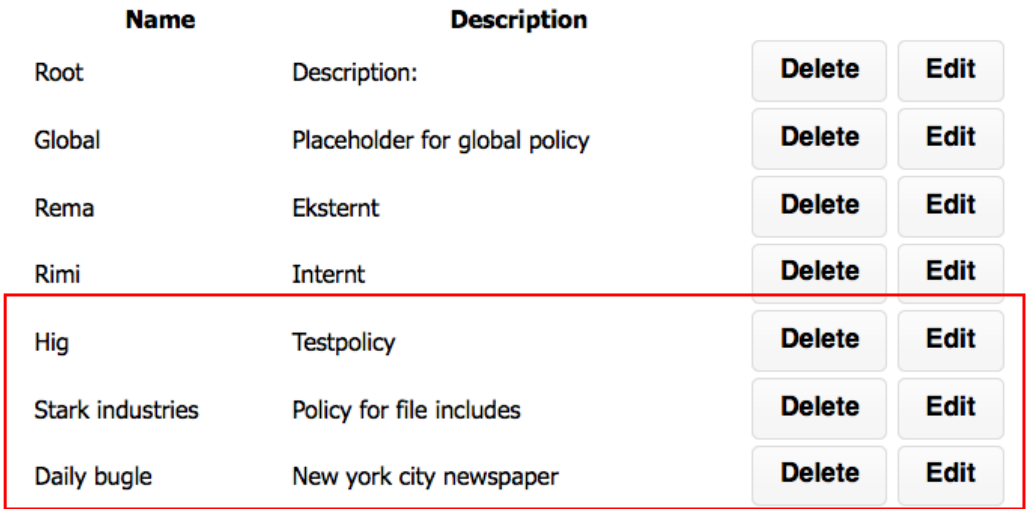
Potensielle feil i produksjon av konfigurasjonsfiler kan få alvorlige konsekvenser for sensorer. De kan ignorere skadelig aktivitet eller helt slutte å fungere. Testene har derfor blitt nøye laget og utført for å forsikre at konfigurasjonsfiler blir korrekt produsert, og at riktige signaturer blir skrevet til den.

Testene i prosjektet er i hovedsak fokusert på signaturdelen til produksjonen, ettersom direktiver skrevet ut før og etter skrives direkte til filen uten behandling. Det skal ikke måles tid produksjonen av konfigurasjonsfiler tar, ettersom det varierer med innholdet i policyene. Det er også vanskelige å produsere et reellt test-scenario, ettersom oppdragsgiver kun kan gi prosjektgruppen fiktive policyer, grunnet taushetsplikt.

Følgende funksjonalitet karakteriseres som basisfunksjonalitet, og ble testet først:

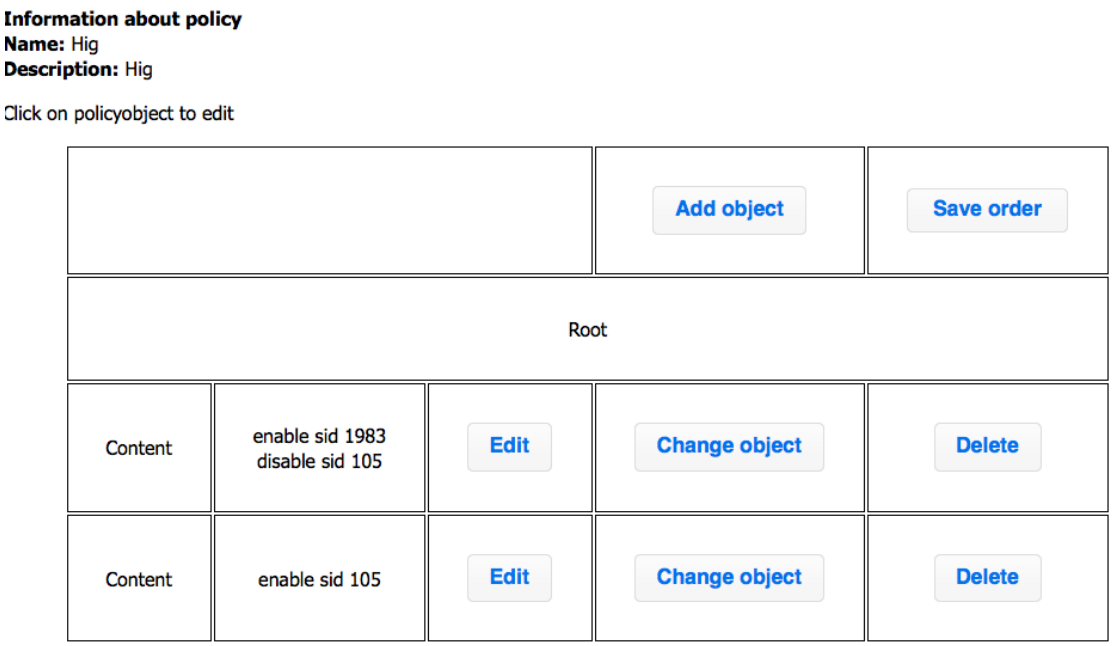
- At enkelt regler ble skrevet til fil basert på SID.
- At regler deaktivert tidigere ble aktivert senere ved behov.
- At hele filer ble skrevet ut til fil.
- At deaktiverte signaturer **ikke** ble skrevet til fil.
- At enkelte signaturer kunne bli deaktivert fra filer ved behov.
- At systemet ikke stopper hvis filer eller signaturer som ikke finnes blir forsøkt aktivert.

For å teste dette opprettet ble tre forskjellige policyer opprettet, som ville bli knyttet opp mot tre forskjellige sensorer. De tre policyene er markert i listen under:



Figur 31: Liste over aktiverte policyer i Snortmanager, hvor de tre nye policyene er markert i rødt.

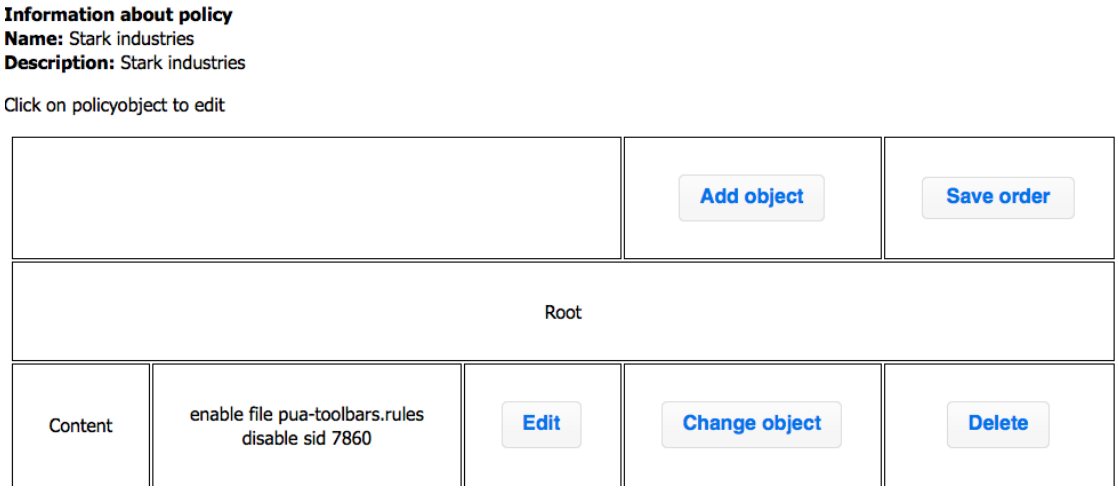
Den første policien opprettet har fått navnet `Hig`, og skal teste om en signatur som ble deaktivert i et tidligere objekt blir reaktivert senere. Den vil derfor deaktivere signaturen med SID 105, for så å reaktivere den senere.



Figur 32: Et skjermbilde av den definerte `HiG` policyen.

Videre opprettes en policy som skal teste Snortmanagers evne til å skrive alle aktive Snort-

signatuerer ut fra filnavnet den først ble registrert som, og at den samtidig klarer å deaktivere en enkelt signatur fra filen, basert på SID. Denne policyen har fått navnet `Stark Industries`, og vil inneholde signaturer fra filen *pua-toolbars.rules*. Filen er registrert i databasen med tre signaturer (7858-7860), og vi velger å deaktivere signaturen med SID 7860.



Figur 33: `Stark Industries` har aktivert filen *pua-toolbars.rules*, men deaktivert signaturen med SID 7860.

Til slutt opprettes policyen `Daily Bugle`, som skal teste om systemet klarer å aktivere en fil med regler. Den skal også teste om systemet tåler at brukeren definerer regler som ikke finnes i databasen. To filer er dermed aktivert; *web.rules*, som ikke eksisterer, og *policy.rules*, som innholder 36 aktiverte regler.

Følgende tabell viser hvilke sensorer som er koblet mot hvilken policy:

IP	**Policy**
192.2.2.2	Hig
192.165.2.3	Stark Industries
172.16.34.3	Daily Bugle

Tiden det tar å produsere en policy er ikke viktig for denne delen av testen, og jobben startes derfor fra webgrensesnittet. På denne måten får man også testet at start av bakgrunnsjobber også fungerer gjennom webgrensesnittet.

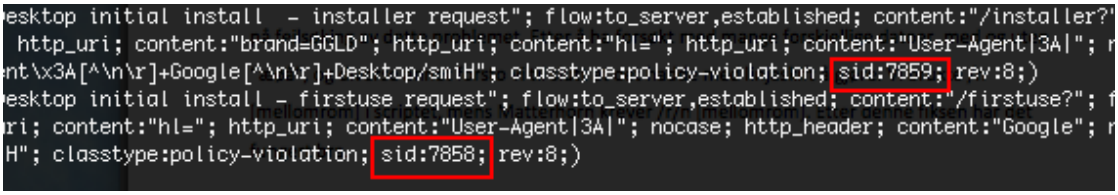
Figur 34: Listen over bakgrunnsjobber i webgrensesnittet hvor produksjonen vil startes.

Når produksjonen er fullført, åpnes mappen Snortmanager benytter for å skrive konfigurasjonsfilene. På testserveren er dette operativsystemets område for midlertidige filer. Her er det opprettet en mappe med et tilfeldig valgt navn, og i denne finner man en konfigurasjonsfil for hver sensor registrert i Snortmanager. Det er kun produsert konfigurasjonsfiler for sensorer registrert i systemet som aktiv, og med koblet sammen en policykjede.

Den første konfigurasjonsfilen vi ser på er produsert med policyen `Hig`. Vi forventer her å finne to regler, hvor én har SID 1983, mens den andre har SID 105. Skjermbildet under er blitt kuttet for å få plass i dokumentet, men SIDene til reglene i filen er markert. Dette viser at Snortmanager klarer å deaktivere en signatur, for så å aktivere den igjen.

Figur 35: Utsnitt av konfigurasjonsfilen produsert av `Hig`, som inneholder to signaturer.

Videre ser vi på konfigurasjonsfilen produsert av policyen `Stark Industries`. I denne filen skal man finne to regler fra filen *pua-toolbars.rules*, som kun har tre regler. Som vist i skjermbildet under, finnes det kun to signatuerer; 7858 og 7859. Den siste signaturen fra denne signaturfilen er deaktivert i policyen, og derfor ikke skrevet til konfigurasjonsfilen



Figur 36: Filen produsert av `Stark Industries` inneholder to av tre signaturer.

Til slutt undersøkes konfigurasjonsfilen produsert av policyen `Daily Bugle`, som inneholdt

den ikke-eksisterende signaturfilen *web.rules* og signaturfilen *policy.rules*, med 36 aktiverte signatur. Dette betyr at konfigurasjonsfilen kun skal inneholde 36 signaturer, noe som verifiseres med at innholdet til filen skrives ut med kommandoen **cat** og sendes til programmet **wc**, som teller linjene. Skjermbildet under viser da at det er 36 linjer i konfigurasjonsfilen, og en linje representerer en signatur.

```
[root@TestServer1 tmp]# cd tmpaKVyAQ/
[root@TestServer1 tmpaKVyAQ]# ls
172.16.34.3.txt  192.165.2.3.txt  192.2.2.2.txt
[root@TestServer1 tmpaKVyAQ]# cat 172.16.34.3.txt |wc -l
36
[root@TestServer1 tmpaKVyAQ]#
```

Figur 37: Konfigurasjonsfilen produsert av `Daily Bugle` inneholder 36 signaturer.

Når den grunnleggende funksjonaliteten er testet, skal den mer avanserte funksjonaliteten i policyobjekter testes. Snortmanager skal kunne skrive om hele, deler og aktivere signaturer basert på regular expressions (regex). For å gjøre dette, opprettes en ny sensor med IP 10.0.0.123, og en ny policy med navnet `OSCORP`.

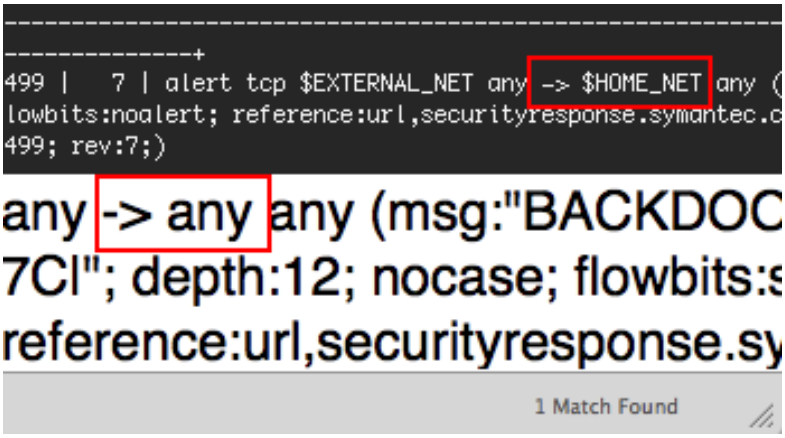
Den nye policyen skal teste at man kan aktivere én enkelt signatur, men omskrive deler av innholdet. Den skal også teste at man kan aktivere og skrive signaturer til konfigurasjonsfilen, basert på et regular expression. Samtidig skal testen kontrollere at en signatur ikke skrives til filen flere ganger, ved å aktivere en av reglene som systemet ville aktivert med regular expressionssøket.

Figur 38: Den avanserte funksjonaliteten skal testes med policyen `OSCORP`.

I bildet over viser at den først aktiverer to regler; den frittstående signaturen med SID 6499, og signaturen med SID 2008953, som vil bli aktivert av regular expressions-regelen. Deretter defineres utrykket som skal brukes for å aktivere alle signaturer med spesifikk innhold. Til slutt aktiveres en omskriving av signaturen med SID 6499, som omskriver teksten *$HOME_NET* til teksten "*any*".

Etter å ha utført en ny produksjon av konfigurasjonsfiler, vil en ny bli lat til de eksisterende. Man teller antall linjer i filen med **cat** og **wc**, og kommer frem til at det er 6 signaturer i filen. For å forenkle kontrollen av konfigurasjonsfilen, kopieres hele innholdet inn i programmet Reggy [1], som er beregnet på å utvikle og teste regular expressions. Først kontrolleres at SID 6499 har blitt skrevet til konfigurasjonsfilen.

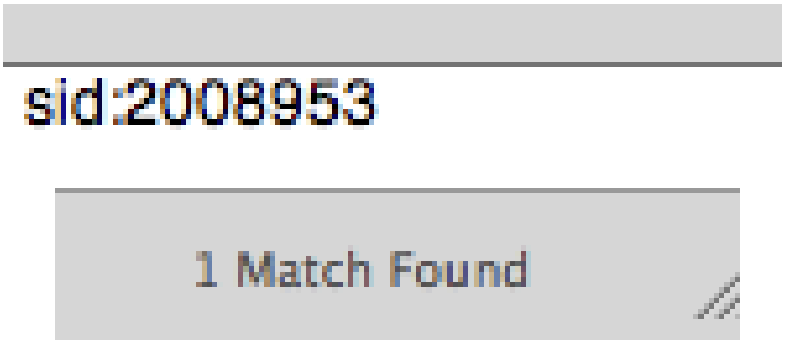
[1] http://reggyapp.com/



Figur 39: Sammendrag av flere skjermbilder som viser at signaturen med SID 6499 ble korrekt skrevet til fil.

Bildet over viser at søket etter SID 6499 finner kun én signaturen. Ved å sammmenligne signaturen lagret i databasen (øverst) med resultatet fra konfigurasjonsfilen filen (midten), finner man at signaturene ikke er identiske. Den riktige delen av signaturen er skrevet om, og ligger korrekt i konfigurajonsfilen uten at databasen er forandret.

Videre undersøkes resultatet av selve regex-søket, og flere signaturer er aktivert. Søket finner totalt 5 signaturer, som viser at antallet i konfigurasjonsfilen stemmer. Ingen ekstra regler er tatt med i produksjon. Til slutt viser skjermbildet under at kun én signatur med SID 2008953 er aktivert i filen. Dette viser at det ikke skrives duplikater av regler til en konfigurasjonsfil, selv om en SID kan være definert flere ganger.



Figur 40: Kun én signatur med SID 2008953 er skrevet til konfigurasjonsfilen, selv om den var aktivert flere steder.

Figur 41: Sammendrag av to skjermbilder med flere signaturtreff og 5 treff.

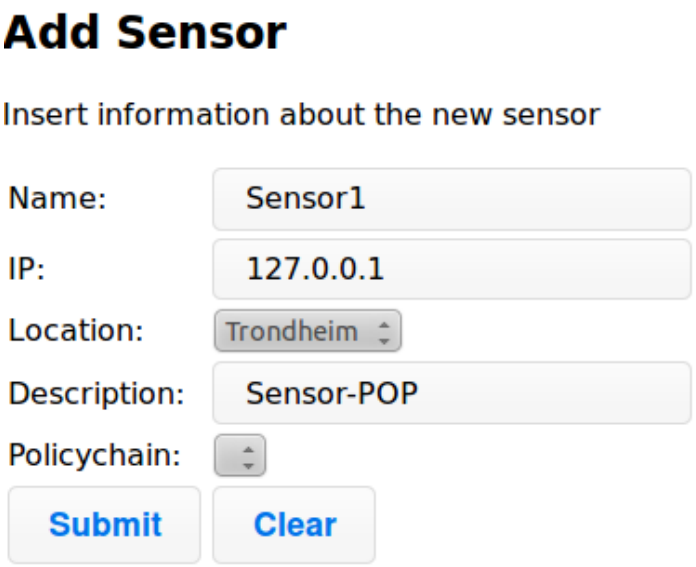
5.8.2 Testing av sensoradministrasjon

Testingen av sensor skulle forsikre at funksjonaliteten og integriteten til funksjonen fungerte korrekt. Administreringen av sensordata er en viktig del av systemet, og ble derfor testet nøye.

Funksjonalitet som testes for sensor er:

- Det skal være mulig å legge til en ny sensor.
- Bruker skal kunne endre eksisterende data på en valgfri sensor.
- Sensor skal kunne settes til inaktiv.

Når en bruker skal legge til en sensor, velges "Add sensor"i menyen til venstre i brukergrensesnittet. Brukeren vil da bli møtt av et utfyllingsskjema, hvor han kan legge inn data om en ny sensor. Det har blitt testet på bildet under, som illustrerer hvordan grensesnittet ser ut med utfylling av data.



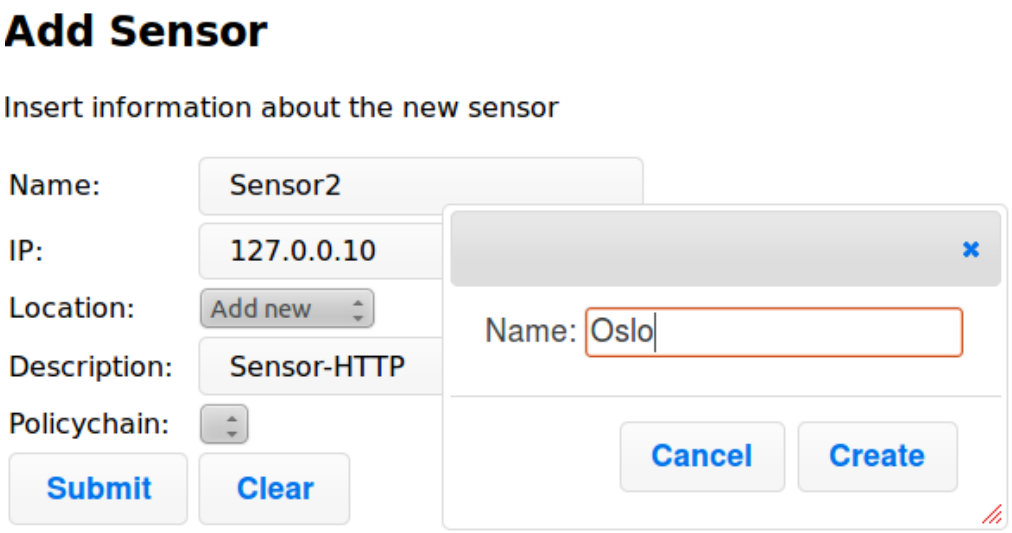
Figur 42: Eksempel på opprettelse av ny sensor.

Bildet over viser hvordan bruker kan taste inn data i skjema for å opprette en sensor. For å bekrefte at data ble skrevet til databasen, viser bildet under en spørring til database for å hente frem den gjeldende sensoren.

```
mysql> select * from Sensor;
+----+---------+-----------+----------+-------------+-------------+
| id | name    | ip        | location | description | policychain |
+----+---------+-----------+----------+-------------+-------------+
|  1 | Sensor1 | 127.0.0.1 |        2 | Sensor-POP  |           1 |
+----+---------+-----------+----------+-------------+-------------+
1 row in set (0.00 sec)
```

Figur 43: Eksempel på opprettelse av ny sensor i database

Spørringen bekrefter at sensor er blitt lagt til i database, og at funksjonen fungerer. Deretter må funksjonen legge til lokasjon på den samme siden som testes. Under vises et bilde hvor brukeren ønsker å legge til "Oslosom ny lokasjon. For å kunne gjøre dette, trykker brukeren på **Add new** i nedtrekksmenyen. Brukeren fyller da inn lokasjonen som skal legges inn i boksen som dukker opp.



Figur 44: Eksempel på opprettelse av ny lokasjon

I likhet med opprettelse av ny sensor, må det også testes om lokasjonen er blitt opprettet i databasen for å bekrefte at funksjonen virker.

```
mysql> select * from SensorLocation;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | Trondheim |
+----+-----------+
1 row in set (0.00 sec)
```

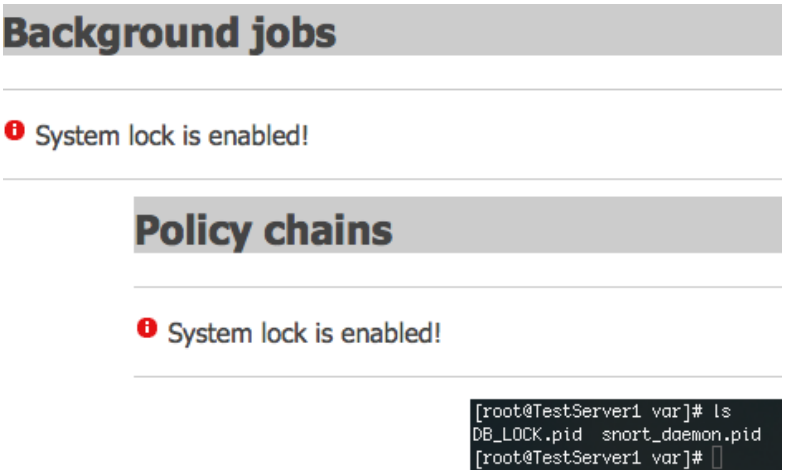
Figur 45: Eksempel på opprettelse av ny lokasjon i database

Bildet over bekrefter at lokasjonen "Osloer blitt lagt til i databasen med riktige verdier.

5.8.3 Test av systemlås

Systemlåsen må fungere til enhver tid for å redusere risikoen for feil på systemet og i dataene. Hvis Snortmanager ikke blir låst for endringer, kan data bli korrupte, og feil kan oppstå senere eller på Snortsensorene. Hvis Snortmanager ikke blir låst opp igjen, vil det ikke være mulig å gjøre endringer eller utføre operasjoner.

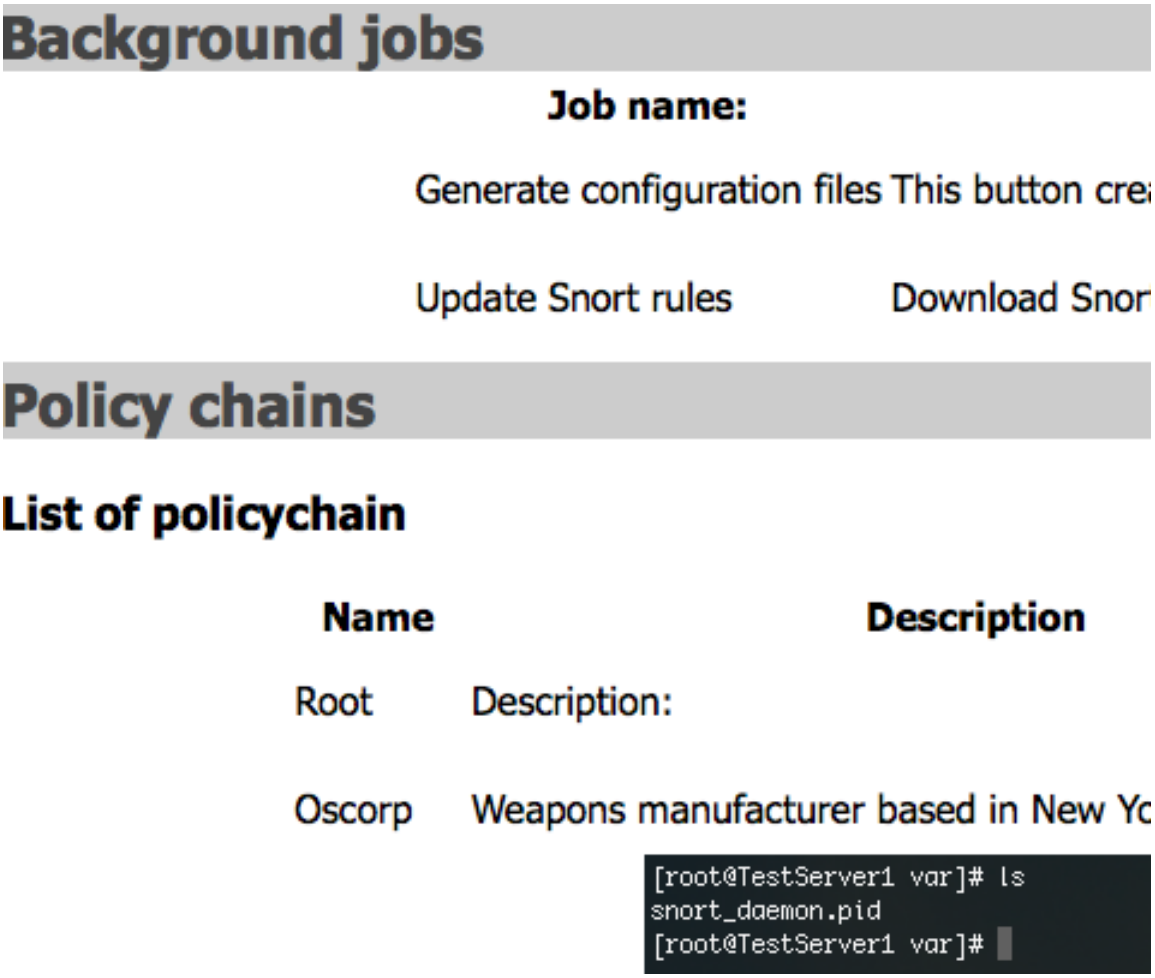
For å teste systemlåsen, blir bakgrunnsjobben som tar lengst tid startet. Dette er oppdateringen av Snortsignaturer, og vil ta flere minutter. Når jobben startes, får brukeren opp en dialogboks hvor den blir bedt om å bekrefte at jobben skal starte. Deretter vil systemet i bakgrunn opprette en låsefil i *var/*-mappen til Snortmanager, som indikerer for andre tjenester at en systemlås er aktivert.



Figur 46: Når systemet er låst, får man melding om dette i brukergrensesnittet, og en fil blir opprettet i filsystemet.

Når jobben er aktivert vil man i hele webgrensesnittet få melding om dette. I skjermbildet over vises et eksempel hvor man får samme feilmeldingen i både administrasjonen for bakgrunnsjobber og policy. Man ser også at knapper er deaktivert i systemet slik at man ikke kan gjøre endringer. Til slutt inneholder *var/*-mappen filen `DB_LOCK.pid`, som indikerer et låst system.

Om systemet er låst sjekkes opp hver gang en bruker åpner en side i webgrensesnittet. Dette betyr at brukere som har gått inn på en side før systemlåsen er aktivert, fortsatt har mulighet til å trykke på knappene. Derfor sjekker systemet i tillegg om systemet er låst hver gang en operasjon utføres. Brukeren vil da få melding om at systemet er låst og endringer ikke kan utføres.



Figur 47: Etter fullført bakgrunnsjobb blir systemet låst opp, beskjeden til brukeren borte og låsefilen slettet.

Når jobben er utført, skal systemet låses opp, og filen skal fjernes fra filsystemet. Etter at testjobben er utført, fjernes meldingen i webgrensesnittet og knapper reaktiveres. Ved å sjekke filsystemet oppdager man også at filen som indikerer et låst system, er fjernet.

Testene har også kontrollert at en bakgrunnsjobb ikke kan startes med administrasjonsskriptet mens webapplikasjonen kjører. Det betyr også at en bakgrunnsjobb som startes av administrasjonsskriptet, låser systemet slik at brukerene av webgrensesnittet får beskjed.

5.8.4 Testing av logging

Det er viktig for sluttbruker å kunne lese gjennom logger, for å holde orden på aktiviteter gjort av systemet eller bruker. Det finnes to loggmoduler, henholdvis logging til database og logging til fil, og kommende avsnitt vil ta for seg testingen av begge to.

Testing av logging til fil

Logging til fil baserer seg på Pythons sin innebygde loggingfunksjon. Det gjorde det viktig å teste gjentatte ganger på flere systemer, for å forsikre seg om at dette fungerte hver gang uten at sluttbruker skal måtte foreta endringer og forbedringer i sitt system for å få logging til fil til å fungere tilfredstillende.

Følgende ble testet:

- Integritet, fungerer logging hver gang?
- Hva blir skrevet til fil?
- Kompatibilitet. Fungerer logging på flere Linuxdistroer, og på det gruppen har satt som standardinstallasjon (les Standardinstallasjon under Innledning)?

Testingen av integritet ble utført ved å innføre funksjonen for loggingen i de fleste moduler, for å forsikre at alle nødvendige filer kunne bruke funksjonene i log.py-skriptet. Funksjonen ble også innført i flere klasser og funksjoner innenfor disse skriptene, for å sikre at logging kunne utføres.

Under vises et testscenario hvor logging til fil blir benyttet. Her blir funksjonen error tilkalt i filen `dbconfig.py`, og databasens navn blir endret med vilje, så den ikke skal klare å finne databasen. På denne måten kunne gruppen teste at det logges til fil hver gang det skjer noe feil ved tilkobling til database.

```
34 try:
35     engine = create_engine('mysql://root:snortmanager@127.0.0.1/nortmanager', echo=False, pool_recycle=3600)
36     #     engine = create_engine('mysql://snortmanager:snortmanager@localhost/snortmanager', echo=False, pool_recycle=3600)
37     metadata = MetaData(engine)
38     Session = scoped_session(sessionmaker(bind = engine, autocommit = True))
39 except OperationalError as e:
40         error("Failed connecting to database")
41     print 'Error while connection to database, shutting down!'
42     print e
43     exit(1)
44
```

Figur 48: Eksempel på loggfunksjon i kode

På bildet over vises et eksempel på funksjonen logg til fil. Linje 35 viser "mysql://root:snortmanager@127.0.0.1/**sn**
Den uthevede teksten er navnet på databasen skriptet prøver å koble til. Her har gruppen med vilje fjernet forbokstaven til Snortmanager for å teste at loggfunksjonen fungerer. Selve loggfunksjonen er implementert på linje 40 under excepten. Denne excepten inntreffer når Snortmanager ikke klarer å koble til database.

Resultatet av denne loggingen ble en ny post i loggfilen `smlog.log`, som ser slik ut "[2012-05-10 19:50:14] ERROR dbconfig: Failed connecting to database".

Innholdet i de genererte tekstfilene måtte også testes, for å verifisere at riktig data blir skrevet til fil.

Data som skrives til fil er: Dato og klokkeslett, alvorlighetsgrad for hendelsen, modulen hendelsen inntraff på og selve meldingen. Loggfilen blir lagret som en tekstfil, og en post ser slik ut:

"[2012-05-10 09:08:02] INFO sensor: This is a testlog message".

Å teste logging til fil var en lang prosess, da gruppen måtte forsikre seg om at alle data skrevet til fil stemte. Det ble derfor for eksempel utført tester på andre nødvendige moduler.

Ved testingen av kompatibilitet ble standardinstallasjonen installert på både Fedora, Ubuntu og CentOS-servere. Tidligere i dette kapittelet er generell test av kompatibilitet beskrevet. Det er likevel nødvendig å teste noe funksjonalitet på andre Linuxdistroer. Dette ble gjort for å finne ut om de samme programmene brukt på CentOS under utviklingen fungerte i andre Linuxdistroer, med særlig tanke på Python-installasjonen.

Testing av logging til database

Måten gruppen testet logging til database på ligner logging til fil. Følgende ble testet:

- Integriteten til loggfunksjonen.
- Hva blir skrevet til database?
- Kompatibilitet.

Logging til database skiller seg fra logging til fil ved at denne funksjonen er utviklet i Python, Cherrypy og SQLAlchemy. Med andre ord er hele denne funksjonen produsert av gruppen, skreddersydd for å kunne skrive til database.

For å logge til database ble funksjonen **log_to_database()** bruk. Funksjonen finnes i filen dbconfig.py. Kall til denne funksjonen ble implementert i de forskjellige modulene og klassene, for å sikre at også denne funksjonen fungerte på tvers av mapper og filer. Det ble gjennomført flere tester, med forskjellige kombinasjoner av meldinger og alvorlighetsgrader, for å forsikre om at funksjonen i et hvert tilfelle skriver riktig informasjon til databasen. Det ble også her, i likhet med logging til fil, utført tester på forskjellige installasjoner på flere Linuxdistroer.

Under vises et eksempel på en logging til database når en bruker legger til en ny sensor gjennom grensesnittet til Snortmanager.

```
log_to_database(1, __name__, "Added new sensor")
```

Etter denne funksjonen er kjørt, legger resultatet seg i database, som vist under.

```
mysql> select * from EventLog;
+----+----------+---------------------+--------------------------+------------------+
| id | severity | timestamp           | module                   | text             |
+----+----------+---------------------+--------------------------+------------------+
|  1 | 1        | 2012-05-17 21:36:26 | webapp.controller.sensor | Added new sensor |
+----+----------+---------------------+--------------------------+------------------+
1 row in set (0.00 sec)
```

Figur 49: Hendelse logget til database

6 Avslutning

6.1 Drøfting

Under kapittelet drøfting skal gruppen redegjøre for resultater oppnådd i det endelige produktet Snortmanager, og valg tatt underveis. Det vil også bli drøftet om hvorfor valgene er gjort og om de kunne blitt gjort annerledes.

6.2 Resultater

Oppgaven var å utvikle et system for behandlingen av sensorer og regler for Intrusion Detection Systemet (IDS) Snort. Med tanke på den korte tiden gruppen hadde på å utvikle systemet, og på systemets omfang, er gruppen fornøyd med det endelige resultatet. Systemet vil gjøre det enklere for oppdragsgiver å utføre tidligere tidkrevende oppgaver.

Etter kravene spesifisert i kravspesifikasjonen, har systemet blitt et godt og stabilt produkt. Etter tilbakemelding fra oppdragsgiver, innfrir systemet oppdragsgivers behov i form av effektivitet, funksjonalitet og ytelse. Resultatene presentert i resten av kapitlet baserer seg på tilbakemeldinger fra oppdragsgiver.

Gruppen har i stor grad effektivisert nedlasting av regler. Tidligere brukte oppdragsgiver lang tid på nedlasting og implementering av signaturer. Snortmanager gjør dette betraktelig raskere. Denne prosessen brukte oppdragsgiver tidligere i underkant av 10 minutter på å utføre. Snortmanager utfører den samme prosessen på mellom to og tre minutter.

Opprettelsen av policy og signaturer er også mer effektiv. Snortmanagers grensesnitt er oversiktlig og intuitivt, og lar brukeren raskt og enkelt opprette nye policyer.

Snortmanagers grafiske verktøy vil også effektivisere måten oppdragsgiver utfører sine rutiner på. Ved enkel navigering i webgrensesnittet vil oppdragsgiver kunne opprette, endre og deaktivere signaturer og policyer.

Gruppen har også utviklet automatisering av arbeidet til oppdragsgiver. Det har ikke blitt utført en on-site-test hos oppdragsgiver av dette, så gruppen har ingen konkrete testresultater å presentere. I henhold til krav fra oppdragsgiver er scheduler like fullt en funksjon som vil automatisere arbeidsoppgaver oppdragsgiver gjør manuelt i dag. Snortmanagers-scheduler funksjon tillater brukeren av systemet å automatisk utføre nedlasting av regler i et gitt intervall. Dette vil utelukke enkelte oppgaver som tidligere måtte utføres manuelt av mnemonics ansatte.

Administrering av sensorer er også enklere og mer oversiktlig i Snortmanager. Bruker vil enkelt kunne se hvilke sensorer som finnes i systemet, og manipulere disse etter ønske. Dette var tidligere basert på tekstfiler; en relativt tungvinn måte å gjøre det på.

Loggingen tillater oppdragsgiver å holde oversikt over endringer utført av Snortmanager. Dette er en fordel for oppdragsgiver i sammenhenger der det må foretas inspeksjoner av problemer som oppstår, og for å kunne ha en oversikt over hva som har blitt gjort når.

Funksjonaliteten Snortmanager tilbyr vil redusere arbeidstimer relatert til behandling av sensorer og signaturer.

Brukeren av systemet vil i stor grad ha nytte av å lese rapporten, som dokumenterer hvert aspekt av Snortmanager og dens funksjonalitet, med detaljer og eksempler.

6.3 Valg tatt underveis

Gruppen brukte lang tid på å kartlegge systemet og på å finne passende rammeverk og teknologi som skulle benyttes under utviklingen. Valgene har vært tilfredstillene for gruppen, og bidratt positivt til utviklingen av Snortmanager. De viste seg å passe inn i prosjektet, og var relativt enkle å lære og bruke. En utfordring var at noen av utviklingsspråkene ikke var godt dokumentert, eller hadde få eksempler på bruk. Noe ekstra tid ble derfor brukt på å løse enkelte problemer som oppsto under utviklingen.

Python ble brukt som programmeringsspråk etter oppfordring fra oppdragsgiver. De fleste systemene de kjører i dag er skrevet i Python, og de ansatte har mye erfaring innenfor språket. Python viste seg å være et godt programmeringsspråk for Snortmanager, og satt seg inn i språket raskt. Språket i praksis var behagelig, oversiktlig, og godt dokumentert.

Siden gruppen skulle bruke Python som programmeringspråk, måtte gruppen velge rammeverk ut fra dette. Gruppen valgte å bruke Cherrypy som rammeverk for webgrensesnittet. Rammeverket er enkelt, men har samtidig god funksjonalitet. Gruppen vurderte også de populære rammeverkene Django, og Pylons. Disse ble valgt bort fordi det er for omfattende for oppgaven, krever for mye ressurser og ville være vanskelig å sette seg inn i på kort tid

Gruppen er fornøyd med valget om å bruke Cherrypy som rammeverk, og det viste seg å fungere godt. Dokumentasjonen var en utfordring. På noen områder var den tynn, og kodeeksemplene var for simple. Mye tid ble derfor brukt på å finne løsninger på problemer. Skulle prosjektet gjentas ville fortsatt Cherrypy blitt valgt.

Snortmanager trengte også en template engine utviklet for Python, og med støtte for CherryPy. Valget falt derfor på Jinja2, et funksjonabelt og velutviklet verktøy. Det har fungert utmerket for prosjektet, og er godt dokumentert. Gruppen er fornøyd med valget av Jinja2, og ville brukt det igjen ved senere anledninger.

Gruppen trengte også et verktøy for å kunne simplifisere dialogen med databasen. Det ble derfor bestemt å bruke verktøyet SQLAlchemy, som tilbyr databasefunksjonalitet i Python, uten å skrive SQL-spørringer. SQLAlchemy var et godt verktøy, og effektiviserte utviklingen betydelig. Dette verktøyet vil bli brukt igjen ved senere anledning.

MySQL ble valgt som database, siden begge gruppemedlemmene har tidligere erfaringer med denne. Valget av database hadde lite å si, fordi SQLAlchemy fungerer mot de fleste databaser.

6.4 Snortmanagers fremtid som åpen kildekode

Snortmanager skal eksistere videre som åpen kildekode, fritt tilgjengelig på Internett. Tanken med dette er at nye produktet skal kunne brukes av andre med samme problemstilling som oppdragsgiver. Nye utviklere skal kunne bidra til ny funksjonalitet og feilretting. Hele eller deler av Snortmanager kan også brukes i andre prosjekter, så lenge disse også er åpen kildekode.

Underveis i prosjektet kom det ønsker fra oppdragsgiver som ble nedprioritert eller fjernet fra bachelorprosjektet. Det har også oppstått situasjoner, som tidsmangel, som har ført til omprioritering.

Slik funksjonalitet blir overført til åpen kildekodeprosjektet, hvor det vil ble designet, utviklet og implementert i Snortmanager. Eksempler på slik funksjonalitet er.

API: Det var tidlig et ønske om et API, slik at Snortmanager kunne integreres med andre løsninger oppdragsgiver bruker. Tanken var at alle operasjoner og informasjon tilgjengelig gjennom webgrensesnittet, også skulle være tilgjengelig gjennom APIet. Dette ble ikke med i sluttleveransen, på grunn av manglende tid og ressurser. Ved å modifisere deler av prosjektet og Routes-mappingen skal dette implementeres i en senere utgave.

Portabilitet: Snortmanager er designet og testet for Unix-basert operativssytemer. Samtidig er det bygget på teknologier som kan brukes på de fleste operativsystemer. Det er derfor planlagt å kartlegge hvor mange systemer som er støttet, for så å gjøre nødvendige modifikasjoner for å kunne kjøres på flere operativsystemer.

Søk etter signaturer: Det ble planlagt en søkefunksjon, som skulle gjøre det lettere å finne informasjon om signaturer, basert på innhold eller SID. Det er tilrettelagt for funksjonaliteten, men den ble ikke med i sluttleveransen, grunnet tidsmangel.

Utfyllende rapport: Snortmanager skulle kunne levere en rapport over bruk av signauterer, policybruk og sensorer. Dette ble ikke tatt med i sluttleveransen, men er planlagt i en senere utgave.

Autentisering: Brukere skal logge seg inn med personlig brukernavn og passord, og kun ha tilgang spesifisert funksjonalitet i Snortmanager. Infrastrukturen for dette ble implementert, men funksjonaliteten ble aldri utviklet, grunnet tidsmangel.

Validering av policyobjekter: Policyregler kan være omfattende, og vanskelig å skrive. Det skal derfor utvikles en validator som kontrollerer regler, mens de skrives av brukeren, og når de lagres.

Administrasjon av planlagte oppgave: Det er nødvendig å kunne administrere planlagte oppgaver bedre, dette forbedres i senere versjoner.

6.4.1 Kjente feil

Under testingen har forskjellige feil oppstått i applikasjonen, og ble utsatt til prosjektet ble åpen kildekode.

Generelt

- Ikke all input fra sluttbrukerene blir validert, og får fjernet skadelige karakterer
- SQLAlchemy klarer ikke konvrtere skandinaviske karakterer i noen grensesnitt.

Policy

- Det er mulig å registrere en policy flere ganger med samme navn.
- Det finnes ingen liste over deaktiverte policyer.
- Det er mulig å deaktivere og endre navnet på Root policyen.

6.4.2 Distribusjon av kildekoden

Snortmanager er et åpen kildekodeprosjekt, og trenger derfor å være tilgjengelig for nye utviklere som ønsker å forbedre det, gjøre endringer og bruke det i andre prosjekter. Koden vil være tilgjengelig på GitHub [1], en tjeneste og samfunn for utveksling av kildekode på Internett.

GitHub ble valgt da de baserer utveksling av kildekode på versjonshåndteringssystemet Gits lette administrasjon, gode oversikt og gode løsninger for samhandling for flere utviklere.

6.5 Evaluering av gruppens arbeid

Prosjektgruppen sto ovenfor mange utfordringer gjennom prosjektperioden. Først og fremst er prosjektgruppen liten, med bare to medlemmer. Fordi oppgaven er tilrettelagt for 3 personer, ville dette bety at arbeidsmengden ville være større for hver person. I tillegg hadde ikke medlemmene jobbet sammen tidligere, og kjente hverandre ikke godt. Noe tid ble derfor brukt til å bli bedre kjent og utbedre rutiner for å kunne samarbeide best mulig.

Tidlig i prosjektet ble det opprettet et gruppereglement med retningslinjer, regler og sanksjoner. Reglementet og retningslinjer ble fulgt nøye, og det har ikke vært behov for revisjon. Gruppen har også forsøkt å til enhver tid holde hverandre, oppdragsgiver og veileder oppdatert på prosjektstatus.

6.5.1 Organisering

Prosjektgruppen var tidlig bestemt på at man i hovedsak skulle sitte sammen på felles grupperom, hvor det arbeidsoppgaver ble gjort individuelt og med samarbeid. Gruppen tilrettela også også for hjemmekontor eller å jobbe et annet sted enn grupperommet. Arbeidstiden medlemmene har tilbragt på grupperom har vært varierende, men i de fleste tilfeller vært 08 - 16. Arbeid er også blitt utført individuelt utenom arbeidstiden, blant annet kvelder og helger.

Ved å bruke samhandlingsverktøyene nevnt i innledningen, har det ikke vært nødvendig å sitte sammen til enhver tid. Ettersom medlemmene har hatt forelesninger, på forskjellige tidspunkt, har hjemmekontor blitt brukt på faste tidspunkt.

Loggføring av arbeid ble utført av individuelt på dagsbasis. I de fleste tilfeller er også tidsbruk dokumentert, i tillegg til aktiviteter og fremgang. Gruppemedlemmene har ikke lest hverandres arbeidslogger, før sluttrapporten skal leveres.

Gruppen har hatt faste møter med både veileder og oppdragsgiver. Møter med veileder har vært korte statusmøter på hans kontor. I løpet av prosjektets siste måned ble også tiden brukt til å diskutere utfordringer, og få løpende tilbakemelding. Disse har i stor grad vært positive, og veiledermøtene har vært nyttige for å bekrefte at prosjektet var på rett vei.

Gruppen har også hatt ukentlige statusmøter på Skype med teknisk kontakt host oppdragsgiver. Denne tiden ble brukt til å presentere status på prosjektet, utviklingen og diskusjon rundt problemstillinger og utfordringer. Det ble også et fysisk møte ca. hver fjerde uke, hvor møtet ble holdt annenhver gang på Gjøvik og på oppdragsgivers hovedkontor i Oslo. Disse møtene viste seg å være mest effektive når de ble avholdt i Oslo, ettersom representant brukte mye tid på reise til Gjøvik og kun hadde kort tid sammen med prosjektgruppen. Selv om det var mindre effektivt, var det fortsatt nyttig for å teste og presentere Snortmanager. Prosjektgruppen har fått god støtte

[1] http://www.github.com

fra oppdragsgiver, som hele tiden har vært tilgjengelig for å besvare spørsmål over e-post. De har kommet med informative tilbakemeldinger, og vært klare på ønsker og krav. Det eneste man kunne gjort annerledes var å flytte alle møtene på Gjøvik til Oslo.

Ved å bruke reStructuredText og versjonhånderingssystemet Git, har gruppen hele tiden hatt full endringshistorikk på dokumentet med kommentarer. Alt arbeidet ble formidlet ved hjelp av en server på Internett som tilhørte et av medlemmene, men har i tillegg ligget på medlemenes personlige datamaskiner. For å ytteligere redusere risikoen for datatap, har arbeidet også ligget sikkerhetskopiert i Cloud og på to individuelle testservere.

Denne måten å arbeide på har fungert godt, da det har vært enkelt å kunne arbeide individuelt, for så å flette sammen arbeidet automatisk ved deling. Ved å ha full endringslogg med kommentarer fra medlemmene på alt arbeidet ved hjelp av funksjonalitet i Git, har det vært enkelt å holde oversikt på hva som er utført og hvilke endringer som er gjort.

I ettertid skulle det vært definert retningslinjer for bruken av kommentarer i arbeidet, og hva som skal skrives i endringsloggen. Med begrenset tid og ressurser kan det være vanskelig å skrive informative meldinger i endringsloggen. reStructuredText har også god støtte for kommentarer som ikke ville blitt vist i det ferdige dokumentet. Dette skulle blitt brukt mer for å plassere elementer og utveksle informasjon mellom deltakerne. Det burde også vært et krav om å laste opp arbeidet til arbeidsserveren etter hver arbeidsdag, for å ha en sikkerhetskopi.

Gruppen avtalte tidlig å ha daglig møte om foregående og dagens gjøremål, men dette ble ikke fulgt opp. Det skjedde fordi medlemmene for det meste satt fysisk sammen, hadde god dialog, og lett kunne se forandringer i arbeidet. Gruppen fulgte opp utviklingsmodellen etter beste evne, men situasjoner har oppstått som faller utenom. Flere tidsfrister fra prosjektets fremdriftsplan ble brutt, på grunn av oppgavens kompleksitet. Dette virker i ettertid uoverkommelig, og var vanskelig å forutse. Tidsplanen ville blitt endret for å ta hensyn til tiden som ble brukt på å lære programmeringsspråk og teknologier.

6.5.2 Fordeling av arbeidet

Fordeling av arbeidsoppgaver ble forsøkt gjort fortløpende. Gruppen forsøkte tidlig å vurdere arbeidsmengden til hver enkelt oppgave, og sammen fordele basert på ønsket arbeidsmengde, område og prioritering.

På grunn av arbeidsmengden, gruppens størrelse og kompetanse, ble fordelingen skjev over tid. Nye oppgaver ble valgt etter behov og prioritering. Selv om medlemmene jobbet hver for seg, hjalp de hverandre hvis én sto fast. Gruppen diskuterte løsninger med hverandre gjennom hele prosjektet, og presenterte også sammen for både veileder og oppdragsgiver.

6.5.3 Prosjekt som arbeidsform

Prosjektet har vært krevende, og mye tid er brukt på sette seg inn i oppdragsgivers problemstilling. Det har også vært krevende å sette seg inn i et nytt programmeringsspråk samtidig med utviklingen. Det har uansett vært lærerikt å måtte veksle mellom forskjellige tenkemåter, finne løsninger på problemer, vurdere teknologi og ha løpende dialog med flere interessenter under utviklingen.

Prosjektet har vært en god erfaring, og medlemmene er i enige om at erfaringen som ble opparbeidet vil være nyttig i arbeidslivet. Som nevnt tidligere, er det ting som ville blitt gjort

annerledes, men gruppen er stolte av arbeidet de leverer i form av sluttrapport, og produktet Snortmanager.

A Forprosjekt

Forprosjekt Snortmanager

Christer Vaskinn & Kristian Wikestad

25. januar 2012

Innhold

Sammendrag **4**
Tittel . . . 4
Prosjektdeltakere . . . 4
Oppdragsgiver . . . 4
Veileder . . . 4
Problemstilling . . . 4
Bakgrunn for problemstilling . . . 4

Prosjektgruppen **5**
Gruppens medlemmer . . . 5
Veileder . . . 5
Kontaktperson mnemonic . . . 5

Mål og rammer **6**
Bakgrunn . . . 6
Prosjektmål . . . 6
Resultatmål . . . 6
Effektmål . . . 6
Rammer . . . 7

Omfang **8**
Oppgavebeskrivelse . . . 8
Funksjonalitet i Snort . . . 8
Avgrensning . . . 9

Prosjektorganisering **10**
Ansvarsforhold og roller . . . 10
Oppdragsgiver . . . 10
Prosjektgruppen og prosjektleder . . . 10
Rutiner og reglement . . . 10
Grupperegler . . . 10
Retningslinjer . . . 10

Planlegging, oppfølging og rapportering **11**
Valg av softwareuviklingsmodell . . . 11
Utfordringer . . . 11
Behov . . . 11
Modell . . . 11
Plan for statusmøter og beslutningspunkter . . . 12

Organisering av kvalitetssikring **13**
Dokumentasjon, standardbruk og kildekode . . . 13
Standarder . . . 13
Kildekode . . . 13
Konfigurasjonsstyring . . . 13
Risikoanalyse . . . 13

A Gantt prosjektplan **15**

B Grupperegler for bacheloroppgaven Snortmanager **16**

Sammendrag

Tittel

Snortmanager

Prosjektdeltakere

Christer Vaskinn
Kristian Wikestad

Oppdragsgiver

mnemonic as
v/ Roger Storløkken

Veileder

Høgskolen i Gjøvik
v/ Slobodan Petrovic

Problemstilling

Lage et webbasert system for å forenkle driften av Snort sensorer

Bakgrunn for problemstilling

mnemonic as [sic] leverer tjenester innenfor informasjonssikkerhet som inkluderer overvåkning av kundenes nettverk for ondsinnet aktivitet med systemet Snort. Dette skaper utfordringer med kunders behov, administrasjon og konfigurasjon av overvåkningssystemets sensorer.

Prosjektgruppen

Gruppens medlemmer

Christer Vaskinn
Tlf: 918 48 384
E-post: christer.vaskinn@hig.no

Kristian Wikestad
Tlf: 416 30 307
E-post: kristian.wikestad@hig.no

Veileder

Slobodan Petrovic
Tlf: 478 20 950
E-post: slobodan.petrovic@hig.no

Kontaktperson mnemonic

Roger Storløkken
Tlf: 950 50 875
E-post: roger@mnemonic.no

Mål og rammer

Bakgrunn

Som en av Nordens største leverandører av produkter og tjenester innenfor IT- og informasjonssikkerhet, leverer mnemonic as [sic] blandt annet sikkerhetsovervåkning av nettverk. Slike systemer fungerer ved å analysere nettverkstrafikk for ondsinnet og skadelig innhold. For å utføre dette bruker de flere verktøy, som inkluderer open source intrusion detection systemet (IDS) Snort.

Med 4 millioner nedlastninger og 400 000 registrerte brukere er Snort det mest utbredte systemet for nettverksovervåkningen på markedet. Systemet fungerer ved å plassere sensorer på nettverket, denne vil så sammenligne trafikken med signaturer. Oppdager sensoren mistenkelig trafikk, vil den rapportere til en sentral server. Den sentrale servern vil da advare en ansatt som kan analysere funnet og gjøre tiltak deretter.

Administrasjonen av sensorer gjøres ved hjelp av individuelle konfigurasjonsfiler som produseres lokalt på sentral server og distribueres til sensorene. Denne prosedyren er meget omfattene da den baserer seg på kommandolinjeskript skrevet i forskjellige språk og en manuell overføring. Ofte må også konfigurasjonsfilene redigeres manuelt, for å tilpasses kundenes behov før de distribueres til sensorene.

Dette omfattende manuelle arbeidet gjør konfigureringen av sensorer tidkrevene for mnemonic. Ved en eventuell hendelse eller normalt vedlikehold bruker prosessen en stor del ressurser. Den manuelle redigeringen av filer kan også medføre feil i konfigurasjonen, som kan skape konsekvenser for kunden.

Prosjektmål

Resultatmål

Ved å forenkle administrasjonen av denne meget viktige oppgaven ønsker mnemonic at ansatte kan bruke tid på andre oppgaver. Som en bonuseffekt vil deler av funksjonaliteten gjøre det lettere for selskapet å utføre endringer og reagere ved sikkerhetshendelser. En annen forventet effekt vil være en redusert sjanse for feilkonfigurasjon av sensorene.

Effektmål

Prosjektets hovedmål er å utvikle en open source webapplikasjon som forenkler administrasjon av sensorer. I denne applikasjonen skal det være følgende funksjonalitet:

- Opprette og redigere konfigurasjonsfiler basert på hirarkisk modell av sensorer og servere.
- Opprette regelsett pr. hirakiske nivå som alle sensorene arver.
- En endringslogg hvor man kan se konfigurasjonsendringer utført av hvilke ansatte/brukere.
- Staistikk over bruker av konfigurasjonsalternativer

Det er også et ønske om følgende funksjonalitet:

- Verifisering og automatisk distribusjon av konfigurasjonsfiler.
- Verifisering av regelsett.
- Statistikk over signaturbruk
- Et API for kombinering med øverige løsninger.
- Redigeringsverktøy for å kunne legge inn egne signaturer.
- Mulighet til å redigere signaturer lastet ned fra leverandører.

Rammer

Prosjektet starter med å designe og utvikle løsningen, og skal startes senest 1. februar 2012. Systemet skal være tilgjenglig med en standard open source lisens, og valget av teknologiske løsninger må reflektere dette. Det skal ikke være komplisert for en bruker med middels Snort kjennskaper å sette seg inn i løsningen å utføre administrasjon. Løsningen skal være ferdig i starten av juni 2012.

Omfang

Oppgavebeskrivelse

Oppgaven går ut på å lage et webbasert verktøy for å enkelt kunne produsere konfigurasjonsfiler for Snort sensorer. Backend av løsningen skal utvikles i Python 2, mens frontend utvikles med HTML5 og Javascript (jQuery). Programmeringsspråk ble valgt av hennsyn til ønske fra oppdragsgiver, men vi velger selv hvilket rammeverk som brukes. Databasen som blir valgt må reflektere systemets åpne natur, være i stand til å håndtere mengden data, være pålitlig og støttes av rammeverket.

For prosjektet har vi valgt mikrorammeverket CherryPy som egner seg utmerket for små prosjekter. CherryPy har lang fartstid, er veldig stabilt, har mye funksjonalitet og er veldig fleksibielt. Rammeverket kommer også med en meget fleksibel stabil som vi ønsker å utnytte.

Grunnet rammeverkets minimalistiske natur kommer det ikke med standard funksjonalitet for templates og databaselagring. Vi har valgt Jinja2 som template motor. Denne er også meget fleksibel, liten men også veldig sterk. Ikke minst har den veldig enkel syntax, som minner om PHP/HTML programmering vi er vant med.

Til slutt har vi valgt SQLAlchemy for databasehåndtering, da den har god støtte for CherryPy og mange populære databasesystemer. SQLAlchemy kommer også med en omfattende ORM (Object Relational Mapper) som er enkel å bruke. SQLAlchemy gjør det enklere å skalere løsningen på et senere tidspunkt, og bytte databasemotor skulle dette være ønskelig.

Til slutt har vi valgt MySQL som databasemotor, en av verdens mest utbredte. Databasen er å finne i de fleste driftsmiljøer, og er den prosjektgruppen har lengst erfaring med. Ved å bruke en slik database i motsetning til for eksempel den filbaserte løsningen SQLite, kan vi i større grad forsikre integriteten av dataene.

Løsningen skal basere seg på en hierarkisk modell for arv, som betyr at en konfiurasjonsfil produseres ved å arve spesifikke alternativer. Dette vil forenkle opprettelse av nye kunder, behov og soner samt gjøre det lettere å endre alternativer globalt for sensorene.

Funksjonalitet i Snort

Automatisk oppdatering av signaturer: Det finnes flerfoldige tusen signaturer som benyttes i Snort, og de fleste av signaturene blir utviklet av eksterne leverandører som Snort.org (VRT) og Emerging Threats. Det er derfor viktig og opprettholde og oppdatere signaturene, for å sørge for best mulig sikkerhet.

Distribusjon av regelsett: mnemonic har sitt hovedkontor i Oslo, og må derfor kunne ha en enkel og effektiv måte å distribuere tilpassede regelsett til en eller

flere sensorer.

Det har vært et ønske fra oppdagsgiver at konfigurasjon skal kunne verifieres for syntaxfeil før de distribueres til sensorene. Denne verifiseringen omfatter bare endringer som gjøres i webgrensesnittet av brukerne.

Endringslogg: Oppdragsgiver ønsker å kunne beholde historikk over tidlige endringer. Denne må også inneholde informasjon om endringer en ansatt gjør, slik at dette ikke kan tilbakevises.

Avgrensning

Ettersom prosjektet er omfattende er det viktig at vi har begrensninger for å ikke bli overvelmet. Løsningens design skal være praktisk og funksjonell.

Prosjektorganisering

Ansvarsforhold og roller

Oppdragsgiver

Oppdragsgiver skaffet utstyr for testing av løsningen, og vil gjennom hele prosjektet tilby rådgivning, støtte og tilbakemelding. Ettersom de skal ta i bruk løsningen vil de også være sentral i testingen.

Prosjektgruppen og prosjektleder

Ettersom gruppen bare består av to personer, har vi valgt å ikke ha en prosjektleder. Dette betyr at det er de enkelte medlems ansvar å utføre sine tildelte arbeidsoppgaver.

Rutiner og reglement

Grupperegler

Gruppen har samarbeidet, diskutert og etablert et foreløpig reglement. Begge medlemmene kan til enhver tid komme med forslag til endringer av reglementet. Dette må i såfall diskuteres av gruppen og det reviderte reglementet ville ikke være gyldig før det blir signert av alle medlemmene.

Retningslinjer

Gruppen har opprettet retningslinjer underlagt og basert på gruppereglementet. Disse rutinene omfatter bruken og formidling av kildekode, dokumentasjon, arbeidsoppgaver o.l.

Planlegging, oppfølging og rapportering

Valg av softwareuviklingsmodell

For at gruppen skal kunne utføre oppgaven innen etablerte tidsramme er det nødvendig at gruppen velger et systemutviklingsmodell og et rammeverk.

Utfordringer

Vår gruppe står ovenfor flere utfordringer som må gjennspeile valget vårt av systemutviklingsmodell. Den åpenbare utfordringen vår er gruppens størrelse på. Dette tilsier at vi har reduserte ressurser og potensielt dårlig tid. Vi er derfor avhengig av god planlegging tidlig for å unngå feilberegninger og tidstyver.

Gruppens størrelse har gjort at vi har valgt å ikke ha prosjektleder, som overfører ansvaret for oppgavens utførelse til de indiviudelle medlemmene og til en hvis grad gruppen som en helhet. Dette betyr at vi er avhengige av å vite hva det andre medlemmet gjør og om han klarer å overholde tidsfrister.

Oppdragsgiver har stilt sin kompetanse og erfaring til rådighet, og har derfor kommet med ønsker angående rapportering og statusoppdateringer. Dette fører til at vi må loggføre og dokumentere fremgangen, i tillegg til å planlegge møter og skrive rapporter.

Den siste utfordringen er at deltakerne har øvrige fag ved Høgskolen med egne krav. Det er derfor viktig at den enkelte deltakeren kjenner til hva det skal gjøre.

Behov

Våre kriterier for valg av systemutviklingsmodell gjenspeiler utfordringene prosjektet står ovenfor.

- Korte daglige møter
- Ukentlig rapportering til oppdragsgiver med virtuelt møte
- Månedlig fysisk møte med oppdragsgiver.
- Utveksling av informasjon internt i gruppen

Modell

Etter å ha gjennomgått behov og utfordringer så vi gjennom flere systemutviklingsmodeller som kunne dekke behovet. Vi så tidlig at Scrum inneholdt mye som ville dekke store deler av behovene våre, men vi kunne ikke velge en ren Scrum modell. Løsningen i prosjektet er avhengig av stykkvis utvikling og implementering. Dette førte til at vi bestemte oss for modellen inkrementell utvikling for dette prosjektet.

Når det er sagt skal i også låne forskjellige egenskaper av Scrum. Vi har allerede avtalt med oppdragsgiver at vi skal ha periodiske måter. Et rapporteringsmøte skal skje hver uke og det vil være et fysisk møte hver måned. Disse tidsperiodene vil fungere som Scrum sprints og man vil ikke kunne komme med ønske om

endringer og forslag i løpet av denne perioden.

Vi planlegger å ha et daglig møte i likhet med daily Scrum, som ikke skal vare lengre enn 10 minutter. Daily Scrums pleier å vare 15 minutter, men ettersom vi bare er to i prosjektgruppen bør det holde med 5 minutter pr. deltaker. Her skal man fortelle hva som er gjort siden i går, dagens plan, eventuelle problemstillinger og utfordringer.

Da vi har forskjellige timeplaner vil ikke dette møtet ha et fast tidspunkt, og vi trenger ikke være fysisk tilstede under møte. Ved hjelp av elektroniske hjelpemidler kan deltakerne møtes og jobbe adskilt å ikke påvirke prosjektet negativt. Vi benytter f.eks. en elektronisk løsning for å holde orden på oppgaver som må utføres til enhver tid.

Plan for statusmøter og beslutningspunkter

Etter planen vil det forekomme statusmøter hver 2. og 4. uke med oppdragsgiver. Det er avtalt å ha statusmøter over Skype annenhver uke, men hver 4.uke skal vi ha fysiske møter med oppdragsgiver. Dette vil enten være i Oslo ved oppdragsgivers hovedkontor eller på Høgskolen i Gjøvik. Vi har også avtalt statusmøter med veileder annenhver uke.

Når det gjelder beslutningspunkter vil de generelle beslutningene hovedsaklig tas i gruppa. Hvis det gjelder saker som berører prosjektet iform av endringer av funksjonalitet mm. vil dette avklares med oppdragsgiver og veileder informeres om eventuelle forandringer.

Organisering av kvalitetssikring

Dokumentasjon, standardbruk og kildekode

Dokumentering vil foregå fortløpende i prosjektets gang for å ha en bedre oversikt når prosjektrapporten skal gjøres. Alle gjøremål vil dokumenteres iform av en detaljert logg, vi vil også bruke en tjeneste som heter Trello for bedre oversikt og hva som gjøres av hvilke parter, hva som er ferdig, og hva som ikke er påbegynt.

Standarder

Prosjektet skal følge retningslinjer og standarder som forsikrer en åpen kildekodeløsning.

Kildekode

Dette prosjektet skal som tidligere nevnt utvikles som et open source prosjekt. Noen funksjoner i prosjektet vil også videreutvikles fra noen av funksjoner allerede laget av oppdragsgiver. Vi tester hverandres kode kontinuerlig for å unngå feil og usikker kode.

Konfigurasjonsstyring

Selve produktet i prosjektet skal utvikles i python, og vi må derfor ha en grei måte å samle de forskjellige filene og informasjonen på, samt. holde orden på hva som blir utviklet og endret. Derfor har vi valgt å bruke en løsning som heter GIT. GIT er et open source versjonshåndterings system, som holder orden på, og logger endringer og tilstander på de forskjellige dokumentene og mappene som ligger på serveren.

Risikoanalyse

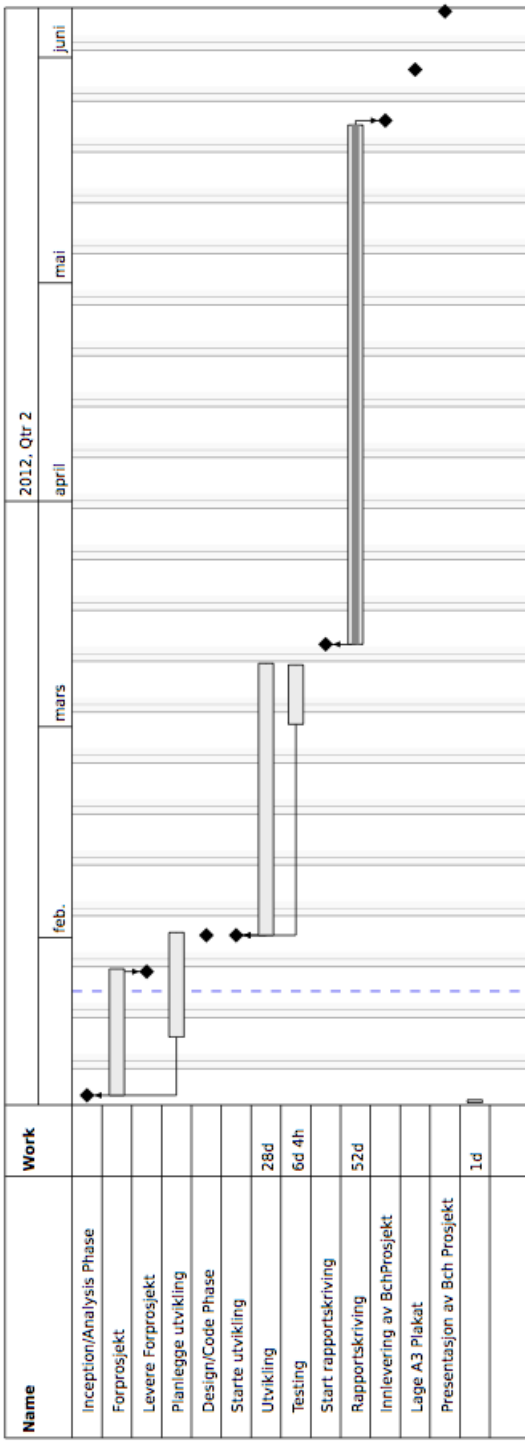
Gruppen har vurdert følgene ting som risikobelagte

Risikobeskrivelse	**Risiko**	**Konsekvens**	**Forebyggende strategi**
Kortvarig sykdom	Middels	Middels	Ha et sunt arbeidsmiljø, formidle arbeidsoppgaver, arbeidsdata og status ved hjelp av elektroniske midler.
Langvarig sykdom	Middels	Høy	Ved alvorlig langvarig sykdom kan prosjektets fremgang svekkes.
Oppdragsgiver går konkurs	Lav	Lav	Firmaet har god likviditet og produktet skal lanseres som frittstående open source produkt.

... fortsetter neste side

Risikobeskrivelse	**Risiko**	**Konsekvens**	**Forebyggende strategi**
Tap av data	Lav	Alvorlig	Sikkerhetskopiering, bruk av cloudtjeneseter, replikering av arbeidsdata og sikkerhetskopier.
Tap av sensitive opplysninger	Lav	Lav	Vi lagrer og oppbevarer ingen sensitive opplysninger for oppdragsgiver eller øvrige deltakere.
Misfornøyd oppdragsgiver	Lav	Middels	God dialog med oppdragsgiver, jevnlig rapportering, møter og applikasjonstesting.
Tapt utstyr	Middels	Alvorlig	Forsikre oss om at vinduer og dører er lukket til ennhver tid når rommet er tomt. Ta backup av kritisk data.

A Gantt prosjektplan



B Grupperegler for bacheloroppgaven Snortmanager

Versjon: 1
Disse reglene er utarbeidet av deltakerne i bacheloroppgaven Snortmanager. Alle medlemmene er pålagt å underskrive og følge reglementet under hele prosjektet. Innholdet i reglementet kan endres i samråd med de øvrige deltakerne men er ikke gyldig før alle har underskrevet den reviderte versjonen.

1. Ved uenighet mellom medlemmene kan det ikke benyttes stemmerett grunnet gruppens størrelse. Slike uenigheter løses derfor ved hjelp en av følgende metoder:

    - Håndbak
    - Myntkast
    - Russisk roulette

2. Det skal ikke være en prosjektleder i gruppen, begge medlemmene skal utveklse og formidle oppgaver i samråd og er pliktig til å utføre disse innen gitte tidsfrister. 1. Hvis man i god tid ser at man ikke klarer å overholde fristen må man informere om dette. Eventult også reprioritere og replanlegge med resten av gruppen, evt. veileder.

3. Ved situasjoner hvor avtalte tidspunkter og tidsfrister ikke blir overholdt blir det benyttet sanksjoner. Dette medfører at medlemmetet spanderer lunsj (m. kaffe) på øvrige medlemmer. Et tidsbrudd annses å være alt over 15 minutter.

4. Hvis man har skjelig grunn eller informerer på forhånd (i god tid) blir ikke punktet ovenfor fulgt. God tid annses å være minst 4 timer i normal dagtid (8-22).

5. Medlemmene er pliktig til å holde gruppekalenderen oppdatert med tidsfrister og planlagte personlige avtaler som kan påvirke arbeidet.

6. I situasjoner hvor gruppen påløpes kostnader deles det likt (50/50) mellom medlemmene. Kostnader for indiviudelt utstyr, rekvisita o.l. håndteres av den enkelte.

7. Kommunikasjon mellom medlemmene skjer henholdsvis via. Skype, e-post og SMS.

8. Medlemmene skal oppdatere Trello med oppgaver som utføres, er planlagt og ferdigstilt.

9. All dokumentasjon, øvrige dokumenter og kildekode formidles og versjonshåndteres med Git i hennhold til brukerveiledning.

10. Begge medlemmene har retten til å signere dokumenter på vegne av hele gruppen. I situasjoner hvor dette kan medføre kostnader eller lignende må det først rådføres.

B Prosjektavtale

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

mnemonic as, Wergelandsveien 25, 0167 Oslo

(oppdragsgiver), og

Christer Vaskinn, Kristian Wikestad

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 01.01.2012 til 12.06.2012.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Slobodan Petrovic

Oppdragsgivers kontaktperson (navn): Roger Storløkken

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): Roger Storløkken dato 11.01.2012

IMT Dekan/prodekan (signatur): _____ dato _____

C Møtereferater

Møtereferater Snortmanager

Møtereferat

Sted: mnemonics hovedkontorer
Tilstede: Christer Vaskinn, Kristian Wikestad, Stian Jahr
Dato: 08.03.2012

Agenda

1. Demonstrere løsningen og status
2. Diskutere utfordringer med teknologivalg
3. Diskutere policykjeden
4. Planlegge fremover

Gjennomgang

1. Vi viste frem hvor langt vi hadde kommet i prosjektet, og hva som er gjort. Vi fant to feil i nedlastningen av regler. Vi fikset den ene på stedet, men lot den andre bli en sak for videre utvikling.
2. Vi fortalte og demonstrerte teknologien brukt (CherryPy, SQLAlchemy, Jinja2) og fortalte hvorfor vi har brukt så lang tid på å sette oss inn i det.
3. Vi diskuterte policykjeden og kom flere en fornuftig måte å designe det på.
4. Vi planla og prioriterte moduler for videre fremgang.

Møtereferat

Type: Skype
Dato: 22.03.2012
Deltakere:

- Kristian Wikestad
- Christer Vaskinn
- Roger Storløkken
- Stian Jahr

Statusrapport

- Diskuterte problem rundt APPEND/PREPEND i policyobjekter, bestemte at dette skal være ett atributt i objektet
- Diskuterte ting rundt open source, oppdragsgiver var ikke klar over at de hadde eierskap i kode. Vi bestemte å ha med vurdering av open source deling på nettet

Møte med veileder

Dato: 28.02.12
Lokasjon: Veileders kontor
Deltakere:

- Christer Vaskinn
- Kristian Wikestad
- Slobodan Petrovic

Sluttrapport

- Veilder likte strukturen og det nåverende innholdet i rapporten. Forstå at det var flere mangler.
- Gruppen fikk beskjed om at Miljøhensyn skal nevnes. Vi skal hvertfall vise at vi har tenkt på det.
- Testing av effektivteten til systemer er viktig.

Møtereferat

Dato: 29.02.2012
Lokasjon: mnemonics kontorer
Deltakere:

- Christer Vaskinn
- Kristian Wikestad
- Stian Jahr

Agenda

- Presentere rapport
- Status på Snortmanager
- **Diskutere problemer**
    - Problemer med nedlastning
    - Problemer med produksjon av policy

Gjennomgang av rapport

- **Christer og Kristian presenterte det første utkastet av rapporten. Leste gjennom i plenum.**
    - Oppdaget at oppdragsgiver er beskrevet i oppdavedefinisjon, mens det skal være i egen del av rapporten.
    - Stian mente at kompabilitet i del om Python ikke kom godt nok frem.
    - Under inspirasjon kunne vi ta med Oinkmaster og Snortmanager.
    - Som en del av problemstilling kan vi ta med at eksisterende produkter er bygget for instalalsjoner med en sensor men at det ikke skalerer veldig bra.
    - Syntax av policyobjekter må komme bedre frem i implementasjon.
- **Diskusjon rundt del om effektivitet**
    - Vanskelig å definere effektivitet
    - Vi bør kunne forklare Snortmanagers tilpassnynigsdyktighet
    - Bedre tuning for kunder, raskere leveranser
- **Hvordan måle effektivitet?**
    - Hvor mange regler i database? I dag lagres bare aktive regler, sammenligne dagens aktive regler med fremtidens aktive og deaktive regelsett.
    - Ta tiden på produksjon av nye konfigurasjonsfiler og oppdatering av regler.

Problemer med system

Utfordringer med oppdatering av regler

Utviklerne oppdaget at systemet fikk problemer med å sammenligne regler lastet ned fra eksterne kilder og regler i databasen. Ettersom regler i oppdatering alltid inneholder siste revisjon er det viktig at man bare sjekker siste revisjon mot databasen med den nye regelen.

Utfordringer med policyproduksjon

- Inkludering av regler og filer skjer oftere en deaktivering, derfor viktig å ta med dette.
- Siste ledet i produksjon er det gjeldene. Dvs. inneholdet i et kjedeelement lenger ned overskriver øvrige bestemmelser
- Skal ikke være regler i prepend og append, men det er teknisk sett mulig.
- Bytt filendelse fra .txt til .conf
- Rewrite i pre- og append skal kunne overskrive øvrige policyelementer. For eksempel overskriving av variabler:

var HOME_NET [10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/17] rewrite var HOME_NET* "var home_net [172.23.23.22]

Et reélt eksempel finnes i local_sid.conf

Møte med veileder

Dato: 24.04.12
Lokasjon: Veileders kontor
Deltakere:

- Christer Vaskinn
- Kristian Wikestad
- Slobodan Petrovic

Veileder har lest hele sluttrapporten

- Skriv om hva en IDS er i definisjon
- Fjern beskrivelse av policyobjekt ettersom dette kommer bedre frem i design.
- Brukerbeskrivelse har en "bug"
- Hva slags trusler tas det ikke forbehold om siden det er lokalt nettverk.
- **Beskrivelse virker som en gjenntagelse av krav**
    - Bruk heller generell beskrivelse
- **Oppdatere krav til løsningen med tekst fra beskrivelse slik at alle krav samles på ett sted**
    - Lurt å ha alle krav på ett sted
- Hvordan tilfredstiller use case kravene fra kravspec
- Trigger er ikke klar i beskrivelse av lås
- Snortmanager kommer ikke tydlig frem når vi definerer system (nevner system i dokument)
- **Må rette tekst om bruk, det er ikke presist om.**
    - Bruk av regler annses å være bruk av Snort til analyse av nettverk.
- Liste over aktører er fornuftig
- Sjekk tekst som omhandler søk av regler
- **Ikke-funksjonelle krav egner seg kanskje bedre tidligere i planen**
    - Slik at det samles med øvrige krav
- Låsing må spesifiseres bedre
- Forklare hva RUP 4+1 er.
- Ta vekk kunnskapsmangel om programmeringsspråk.

Møte med veileder

Dato: 05.05.12

Lokasjon: Veileders kontor

Deltakere:

- Christer Vaskinn
- Kristian Wikestad
- Slobodan Petrovic

Sluttrapport

- **Designdelen trenger å være mer utfyllende!**
    - Design skal forklare ideén.
    - Implementering er hvordan den ble gjennomført.
- Forklar bedre hva Snort er, hvordan det funker og ting relatert til systmet som regelfiler og signaturer.
- Mer tekst før punktlister
- **Testedelen trenger mye mer innhold.**
    - Viktig å ta med resultater.

Møte med veileder

Dato: 03.05.12
Lokasjon: Veileders kontor
Deltakere:

- Christer Vaskinn
- Kristian Wikestad
- Slobodan Petrovic

Agenda

1. Rask gjennomgang av endringer i rapport og nåværende status
2. Ønske om å flytte neste møte

Gjennomgang av rapport

- Gruppen hadde sendt over ny rapport i forkant av møtet.
- Frist for å overlevere til veileder ble flyttet til 18. mai.
- Gruppen må sette inn flere eksempler i implementasjonsdelen.
- Ta med flere screenshots i implementering
- **Ta med case study i implementering**
    - Ta for oss hvordan Stian forholder seg til produktet
- Ta med skjermpdumper av hvordan man f.eks. bruker policier.
- Fortelle om hvordan mnemonic skal ta i bruk løsningen.
- Viktig nå å utvide informasjon, struktur og innhold er bra.

Nytt møte

Vårt nye veiledningsmøte flyttes til kl 13, onsdag 9.05

D Logg

Logg

Dette er loggene skrevet av medlemmene under prosjektet frem til 19 mai 2012. Hvert enkelt medlem skrev logg individuelt, deretter ble dette flettet sammen til dette dokumentet.

09.01.2012

Gruppen arbeidet med planlegging av prosjektet, og skriving av forprosjektsrapport.

10.01.2012

Gruppen arbeidet med planlegging av prosjektet, og skriving av forprosjektsrapport.

11.01.2012

Gruppen arbeidet med planlegging av prosjektet, og skriving av forprosjektsrapport. Git repo settes opp for arbeidsdokumenter på sentral server. Også jobbet med å sette opp Docutils for ReStrcturedText. Skrev ferdige, og signerte gruppereglement og retningslinjer. Oversendt prosjektavtale til mnemonic.

12.01.2012

Fortsetter arbeidet med planlegging, og forprosjekt. Kontrakter mottatt fra oppdragsgiver. Fordeler oppgaver for forprosjektet.

13.01.2012

Kristian har timer mye av dagen og Christer har møte om karrieredag og fag. Gruppen fortsetter med utbedringen av forprosjekt.

16.01.2012

Kristian jobber med oppgavebeskrivelsen, Christer med mål og rammer.

17.01.2012

Kristian er i begravelse hele dagen, Christer arbeider med forprosjektet. Henholdsvis Softwareutviklings- og risikomodell.

18.01.2012

Gruppen arbeider med prosjektplanen, sammendrag og begynner å sy sammen dokument.

19.01.2012

Møte med veilder, presenterer status på forprosjekt. Syr sammen dokumentet, kompilerer det med latex. Forrandringer skjer etter møte med veilder.

20.01.2012

Prosjektplan - Timer mye av dagen

Kristian jobber med prosjektplan, utenom timer i andre forelesning. Christer jobber med korrekturlesing av forprosjektsrapport.

23.01.2012

Christer finner informsjon om teknologien og rammeverk gruppen skal velge til prosjektet. Kristian jobber med prosjektplan.

24.01.2012

Kristian skirver prosjektplan. Christer jobber med gruppens hjemmeside, rammeverk for rapport og kompilering av forprosjekt.

25.01.2012

Kristian skriver prosjektplan ferdig. Gruppen leverer og får godkjent forprosjekt.

27.01.2012

Christer har hjemmekontor og leser om Python og teknologier. Kristian har forelesninger hele dagen.

30.01.2012

Planlegger framdriften ift. Snortmanager, lage usecaser. Begynner på kravspesifikasjon

31.01.2012

Gruppen Leser meg opp på Python, teknologier og lager usecaser.

01.02.2012

Kristian Syk. Christer jobber setter seg inn programmeringsspråk og teknologier

02.02.2012

Kristian Syk. Christer leder høgskolens karrieredag for IT-studenter.

03.02.2012

Kristian har timer mye av dagen, men får lest litt på python. Christer jobber med webrammeverk.

07.02.2012

Stian Jahr fra oppdragsgiver kommer til HiG for å møte gruppen. Gruppen får Presentert fremgang. Designet datamodell for nedlastning av filer. Diskutert modell for policyhåndtering. Sett på demonisering og planlagte oppgaver i Snortmanager.

13.02.2012

Christer jobber med Design av downloader og updater, lærer seg forskjellige biblioteker og rammeverk.

Kristian hobbet med innlevering av oblig i Database og applikasjondrift, men også lest på dokumentasjon.

14.02.2012

Christer utvikler og designer modul for å laste ned og oppdatere Snort signaturer.

Kristian leser på sqlalchemy dokumentasjon.

15.02.2012

Gruppen har statusmøte med veileder. Kristian Leser på sqlalchemy, cherrypy.

Christer har Programmert updatering av signaturer, brukte mye tid på feilsøking av utviklingsserver. Måtte installere MYSQLLIB for at SqlAlchemy skulle fungere

15.02.2012

Kristian: Leser på sqlalchemy - Jobbe med www teknologi mye av dagen.

Christer jobber med modul for å laste ned Snort signaturer.

17.02.2012

Kristian har timer mye av dagen.

Christer har hjemmekontor og tester regelnedlastning. Leser om CherryPy. Begynner å lete etter andre prosjekter som bruker det.

20.02.2012

Christer fortsetter arbeidet med nedlastning av signaturer.

Kristian leser dokumntasjon på teknologiene før lunsj, www teknologi prosjekt resten av dagen.

21.02.2012

Kristian er hos tannlegen, men leser dokumentasjon.

02.21.2012 laster over regelnedlastning til testserver. Gjør modifikasjoner for å gjøre den raskere, samt bedre utnyttelse av SQLAlchemy

22.02.2012

Kristian driver med Dokumentasjonslesing.

Christer har fikset litt av koden til snortmanager, deriblandt SQLAlchemy oppsett, regelnedlastning og begynner å implementere rammeverk for web.

23.02.2012

Gruppen har Skype møte med Stian Jahr

Kristian setter seg inn i andre løsninger som bruker CherryPy for å lære hvordan det fungerer.

Christer fikser kode, leser om CherryPy og Jinja2. Setter seg inn i prosjektet CouchPotato som har en tiltalende arkitektur for Snortmanager.

24.02.2012

Kristian har timer mye av dagen, studering av dokumentasjon og kildekode til andre programmer.

Christer begynner å implementere MVC strutkruen til Snortmanager.

27.02.2012

Kristian starter jobben med sensormodulen.

Christer fortsetter arbeidet med sensormodul.

28.02.2012

Christer programmeret webgrensesnitt og implementerer MVC

Kristian jobbet med Sensormodulen, problemer med tilkobling til database, feil med port i virtuell maskin.

29.02.2012

Gruppen hadde møte med veilder

Kristian jobbet med sesormodul.

Christer implementerer custom rendrer, og forbedrer SQLAlchemy konfigurasjonen.

01.03.2012

Gruppen hadde Skype møte med Stian Jahr

Kristian jobbet med sensormodul.

Fortsetter arbeidet med å implementere webgrensesnitt.

02.03.2012

Kristian timer mye av dagen, jobber med ferdigstiling av sensormodul.

Christer ser på teknologier for å løser planlagte bakgrunnsjobber. Begynner å implemetere produksjon av konfigurasjonsfiler.

05.03.2012

Kristian jobbet med sensormodul.

Christer implementerer ny funksjonalitet for databasekoblinger med SQLAlchemy. Implementerer Jinja2 rendrer og får den til å fungere. Legger til kontroller for bakgrunnsjobber. Begynte å lage skjemaer

06.03.2012

Kristian jobbet med sensormodul, klargjør for presentasjon til oppdragsgiver. Lagt til loggingfunksjon.

Christer forbereder presentasjon for oppdragsgiver. Vurdererer visuelt design.

07.03.2012

Kristian Fikset problemer med feilinformasjon om sensor blir postet til database.

Christer fortsetter forberedelsen av presentasjon. Arbeider med routes, sqlalchemy og CherryPy.

08.03.2012

Møte med oppdragsgiver ved deres hovedkontor i Oslo.

Gruppen presenterer Snortmanager, blant annet på database og nedlastning av signaturer. Forenkler noe kode for å redusere tiden det tar å oppdatere signaturer. Tester øvrige deler av løsningen.

Oppdragsgiver kommer med gode tilbakemeldinger.

11.03.2012

Christer jobber med regelnedlastning og normalisering av database. Begynner å vurdere sluttrapport,

12.03.2012

Kristian jobber med oblig i database og applikasjonsdrift, men får gjort en del på loggmodulplanleggingen.

Christer Jobbe med policynedlastning, skriver sluttrapport. Oppretter git repo for Rapport.

13.03.2012

Kristian planlegger logging modul, leser meg opp på sqlalchemy, python, cherrypy.

Christer jobber med rapportskrivning, startet på kravspesifikasjon. Skrevet litt på innledning.

14.03.2012

Gruppen har møte med veileder

Kristian endrer sensormodul til å ha sensor lokasjon i egen tabell.

Christer fortsetter arbeidet med rapporten.

15.03.2012

Møte med Stian Jahr; Skype.

Kristian jobber med sensormodulens og samspllet mellom tabellene sensor og sensorlocation.

Christer fortsetter arbeidet med kravspesifikasjonen og øvrig deler av rapporten.

16.03.2012

Kristian skriver logging modul, samt leser meg opp på sqlalchemy.

Christer fortsetter på design av produksjon av konfigurasjonsfiler, skriver deler av kravspesifikasjonen. Implementerer flere systemer i test.

19.03.2012

Kristian er på jobbintervju i oslo, jobber med loggingmodulen på bussen og litt på kveldstid.

Christer fortsetter med filproduksjon, og sluttrapport.

20.03.2012

Kristian jobber på loggingmodulen for systemaktivitet. Jobber med første utkast av rapporten.

Christer skrev ferdig første utkast kravspesfikasjon, og god del på implementinger og design delen.

21.03.2012

Kristian jobber med loggingmodulen, for systemaktivitet. Jobber med første utkast av rapporten, lagt til noe innledning.

Christer ppprettet og skrevet en god del om implementering, opprettet og skrevet innledning til design, opprettet diagram for virutell serverarkitektur. Lagt til oppsummering på norsk og engelsk.

22.03.2012

Møte med stian, skype

Kristian jobber med loggingmodulen. Jobber med første utkast av rapporten.

Christer jobber med strukturen av rapporten, docutils rammeverket. Oppretter egen dokumentkopilatir, samt ny kravspec.

23.03.2012

Kristian Jobber med logginmodulen. Jobber med første utkast av rapporten.

Christer utvikler på forskjellige deler av systemet.

26.03.2012

Christer Laget PDF, endret på kompilerskript, inkludert tekst fra Kristian

Kristian skriver tekst til rapporten.

26.03.2012

Kristian jobber med loggingmodulen.

Christer fullfører policyproduksjon, forbereder teter. Endret litt på signaturoppdatering, og øvrige bugs. Fikset noe konfigurasjon.

27.03.2012

Kristian prøver å ferdigstille moduler og første utkast av rapport til møte med veileder og oppdragsgiver.

Christer: Mange store endringer på kravspec, nytt compilescript, slått sammen endringer i innledning. Fikset problem med sletting av filter.

28.03.2012

Møte med veileder.

Kristian skrevet en del innledning og testing til første utkast av rapporten.

Christer har gått gjennom store deler av rapporten.

29.03.2012

Møte med oppdragsgiver i Oslo.

Lagt til møtereferat for møte med Stian hos mnemonic

30.03.2012

Christer: Endringer på Rule constructor, klasse og ruleupdater jobb, fikset error exception i downlaoder.

05.04.2012

Møte med Stian Jahr på Skype.

09.04.2012

Kristian Jobbet med rapportmodulen, og www-tek prosjekt på slutten av dagen.

Christer: Jobbet med MVC rammeverk.

10.04.2012

Kristian jobbet med rapportmodulen.

Christer gjorde meg ajour etter påskeferien, gikk gjennom dokumentasjon hittil. Forsøkte å sette meg inn i designdokument og tegne UML-skjema i kravspace. Satte igang tester.

11.04.2012

Kristian jobbet med rapportmodulen.

Christer satt seg mer inn i designdokumentasjon. Gikk gjennom flere tester, endret litt på policyproduksjonskoden

12.04.2012

Skypemøte med Stian

Kristian jobbet med raportmodulen.

Christer har lagt til funksjonalitet store mengder funksjonalitet - Policyproducer kan legge til og fjerne sid og files - Policyproducer kan skrive om filer basert på input - Bug som skaper krasjer i policynedlastning er muligens fikset, fikset bug med operational error. Hoppet over designdokumentasjon, skrev store deler av policyproduksjon som rewrite og enable filer. Fungerer med tester. Mnemonic vil at vi skal besøke de 31. mai.

13.04.2012

Kristian: Rapportmodul, mye av dagen går til Database og applikasjonsdrift.

Christer: Fortsetter arbeidet fra i går.

14.04.2012

Christer: Forbedrer full test av automatiserte jobber.

15.04.2012

Christer Oppdatering av signaturer tar for lang tid. Gjør koden raskere ved å modifisere algoritme og implementere chaching. Testing.

16.04.2012

Kristian: Rapportmodul, skriver feil til fil, slutten av dagen går til jobbe med oblig i annet fag.

Christer Designdokument, implementering og kravspec

17.04.2012

Kristian fikset feil med rapportmodul.

18.04.2012

Christer: Store endringer på rapport - Skrevet en del på design - Laget med flere skisser - Benyttet bananer

19.04.2012

Skypemøte med Stian Jahr

Begge Jobber med rapporten.

20.04.2012

Kristian: Database og applikasjonsdrift mye av dagen, jobber med rapport på kvelden.

Christer jobber med rapport.

23.04.2012

Kristian jobber med rapport innledning.

Christer har lagt til deployment view, skrevet en god del på design og oppdatert spec litt. Lagt til nye bilder og diagramer. Lagt til blant annet krav for policyproduksjon, store deler av designdokument og flere ting

24.04.2012

Gruppen har møte med veileder.

Kristian jobber med rapport innledning, testing.

Christer jobbet med design av løsning, implementerte JQuery UI. Lag til nye møtereferater.

24.04.2012

Christer: Jquery og visuelt design av grensesnitt

25.04.2012

Kristian jobber med rapport og testing, leser meg opp på javascript.

Christer fortsetter arbeidet med visuelt design og funksjonalitet i grensesnitt med jQuery.

26.04.2012

Gruppen har møte med Stian Jahr over Skype.

Kristian jobber med rapport og testing, leser meg opp på javascript.

Christer Skriving av objekter til database, design, error code.

28.04.2012

Christer grensesnitt og utvikling

27.04.2012

Kristian endrer addsensor, for å passe med javascript, Database og applikasjonsdrift mesteparten av dagen.

Christer fortsetter arbeidet med grensesnitt og utvikling.

29.04.2012

Christer har gjort store endringer på hele systemet. Utviklet webgrensesnitt og funksjonalitet. Løste bug med legge/endre til kilder

- Kan endre objekt i eget vindu
- Kan fjerne objekt
- Kan deaktivere policy
- Kan legge til og endre kilder for nedlastning
- Mye JQuery
- Json deler av systemet lagt ut
- Lagt til infratruktur for å deaktivere objekter

30.04.2012

Kristian jobbet med rapporten.

Christer jobbet med rapport

01.05.2012

Kristian jobbet med rapporten.

Christer Jobbet med grensesnitt og rapport

02.05.2012

Skypemøte med Stian Jahr

Kristian jobbet med rapport og testing. Skrevet en del på testing

Christer har lagt til mulighet til å opprette og slette nye policuer samt oprette og velge objekter, Små endringer, mye innhold

03.05.2012

Gruppen har møte med veileder.

Kristian jobbet med rapport og testing.

Christer jobbet med grensesnitt

04.05.2012

Kristian jobbet med wwwteknologi prosjekt hele dagen.

Christer jobbet med grensesnitt og trådhåndtering

05.05.2012

Christer jobbet med håndtering av bakgrunsjobber og tråder

06.05.2012

Christer har lagt til bakgrunnsjobber i interface og i systemet. Kan aktivere to bakgrunnsjobber pr. dags dato. Også lagt til scheduler og mulighet til å lagre dette i database. Schduler er koblet sammen med cherrypy.engine

07.05.2012

Kristian jobbet med prosjektrapporten implementering og testing.

Christer skrevet flere sider med design og implementasjon

08.05.2012

Kristian fikser feil i kode, skriver rapport på designdelen. Oppdatert Kravspec med to logg use caser, samt implementering med implementering av logging

Christer: Gjort følgende endringer:

- Fikset daemonizer
- Lagt til at systemet lager en PID fil
- Muligheten for å sjekke systemstatus gjennom PID
- Muligheten til å stoppe en daemon basert på PID
- DAEMON fungerer som normalt
- Slettet gamle filer fra systemet
- Endret oppdatering av signaturer.
- Fikset en skrivefeil, og referansefeil i rapport.
- Skrevet om implementering, testing og design.

Tue May 8 12:39:19 2012 Christer Vaskinn Christer Vaskinn

09.05.2012

Gruppen har møte med veileder.

Kristian jobber med dokumentering og implementering av logging.

Christer jobber med bugfixing, testing og dokumentasjon av testing. Skriver i tillegg om implementasjon.

10.05.2012

Gruppen får besøk av Stian Jar. Presenterer snortmanager for tilbakemelding.

Kristian møter Stian, www-tek prosjekt resten av dagen.

Christer fikser bugs oppdaget i testingen med stian, deriblandt med tråder, logging, database og øvrige bug. Diskuterer design av lås med Stian. Implementerer første versjon av systemlås, og escaping av input strenger. Skrevet mye om testing.

11.05.2012

Kristian jobbet med prosjektrapporten implementering og testing.

Christer gjennomgang av dokument, kode. Skrevet om implementasjon av cherrypy og SQLAlchemy

12.05.2012

Christer: Skrevet ferdig om SQLALchemy og implementasjon.

13.05.2012

Christer Fikset feil med rule updater pluss flere bugs: siste policyobjekt fjerner tidligere objekter. Append og prepend regler fungerer normal. Policyproducter ignorerer kommentarer (strenger som starter med #).Skrevet på implementering, fikset rule nedlastning og policyproduksjon.

Kristian har skrevet en del i innledning, kravspec, design, implementering og testing.

14.05.2012

Kristian har jobbet med prosjektrapporten innledning.

Christer Skrev ni sider, mest på test. Fikset enable by regex, rewrite testet Renamet object-content i javascript fil, Fikset bug med policy ,fikset navnefeil. Fikset litt på Policyproduksjon, lagt til regex enable

15.05.2012

Kristian jobbet med prosjektrapporten implementering og testing. Jobbet med sensoradministrasjon.

Christer laget grensesnitt for location. Skrevet om aktivering i rapporten med regex og omskrivning, lagt til screenshots. Skrevet en god del om testing og implementingring. Startet så vidt på avslutning. Gått gjennom hele dokumentet for å finne feil

16.05.2012

Kristian jobbet med prosjektrapporten implementering og testing.

Christer har skrevet en del, spes innledning og impl. samt lagt til noen bilder. Forberedt systemlås for testing etter ferdigsettelse. Startet på avslutning, skrevet om gruppens erfaringer osv. Skrevet en god del på testing.

17.05.2012

Kristian jobbet med prosjektrapporten avslutning, implementering og testing

Thu May 17 00:42:24 2012 Christer Vaskinn Christer Vaskinn

Christer lagt til avslutningsutkast, skrevet om avslutning.

18.05.2012

Gruppen har levert endelig utkast.

Kristian har ferdigstilt rapport for levering til veileder.

Christer forberedte rapporten for lesing av veilder. Skrev om deler av test. Skrev også om nettverksdesignet på testnettete, RST. Hjulpet Kristian med Drøfting

19.05.2012

Kristian jobbet smått med feilretting.

Christer fullfører arbeidet med administrasjon av sensor. Kan nå legge til, fjerne og endre sensor. Korrekturlest og gjort endringer på sluttrapporten.