NTNU
Det skapende universitet

Autentisering og tilgangskontroll i system for eSporing

Fredrik Folgerø Martinsen

Master of Telematics - Communication Networks and Networked Services (2010)
Juni 2010
Oppgaven levert: Finn Arve Aagesen, ITEM
Hovedveileder: Stig Frode Mjølsnes, ITEM
Biveileder(e):

Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

Oppgavetekst

eSporing har som mål å realisere en elektronisk infrastruktur for sporing i hele matkjeden i Norge. Systemets arkitektur baseres på en internasjonal standard. Systemet skal realiseres som en portalløsning med underliggende tjenester basert på Web Service spesifikasjonene. Tjenestene vil både være enkle, sammensatte og orkestrerte tjenester. Enkelte tjenester skal leveres på tvers av forskjellige sikkerhetssoner.

Oppgaven omfatter utarbeidelse av en løsning for autentisering og aksesskontroll.

1. Studer standard for sporing i matkjeden samt foreliggende spesifikasjoner for eSporing og utarbeid en initiell konseptuel arkitektur. Denne initielle arkitektur skal fungere som et begrepsmessig grunnlag for utarbeidelse av fullstendige krav til autentisering og aksesskontroll som skal brukes i en sikkerhetsanalyse.

2. Utarbeid krav til autentisering og aksesskontroll og gjennomfør en sikkerhetsanalyse av den initielle konseptuelle arkitektur. Utarbeid en revidert funksjonell arkitektur som tilfredsstiller kravene til autentisering og aksesskontroll.

3. Presenter sikkerhetsløsningen i eSporingsløsningen. Diskuter denne løsning med utgangspunkt i løsningen for den reviderte funksjonelle arkitektur.

Oppgaven gitt: 15. januar 2010
Hovedveileder: Finn Arve Aagesen, ITEM

Forord

Rapporten er rettet mot studenter og professorer ved Norges teknisk-naturvitenskapelige universitet ved avdelingen for telematikk. Det er gått ut ifra at leseren har god nok kunnskaper innenfor feltet til oppgaven. Rapporten inneholder derfor ikke definisjoner og forklaringer på de begrepene som er brukt. Til de tekniske begrepene brukes norske oversettelser av de engelske ordene. Dette er gjort for å holde en viss konsistens på rapporten siden den er skrevet på norsk.

Jeg ønsker og rette en stor takk til Professor Finn Arve Aagesen for all den hjelpen og oppfølgingen jeg har fått. En stor takk rettes også til Marijana Kosmerlj og Sturla Thung ved ErgoGroup for å ha kommet med en utrolig morsom og interessant problemstilling, og ikke minst for hjelpen og bidraget deres.

Sammendrag

eSporing er et prosjekt for å lage en nasjonal infrastruktur for elektronisk sporing i matkjeden. Denne infrastrukturen, betegnet som eSporingsløsningen, er basert på en tjenesteorientert arkitektur.

Med tjenesteorienterte arkitekturer stilles det nye krav til sikkerheten da tradisjonelle mekanismer for sikkerhet ikke er gode nok. Oppgaven har sett på sikkerhetsaspekter i tjenesteorienterte arkitekturer, med fokus på autentisering og tilgangskontroll. En løsning, betegnet som en revidert arkitektur, er presentert som et av resultatene til oppgaven. Arkitekturen er utarbeidet fra en sikkerhetsanalyse på en konseptuell arkitektur. Den konseptuelle arkitekturen, med utgangspunkt i eSporing, innlemmer den funksjonalitet og oppbygning som er spesifikk for tjenesteorienterte arkitekturer. Med dette viser den reviderte arkitekturen til en måte å sikre nettverket og tjenester på som oppgaven har kommet fram til er den mest ideelle for gitt situasjon.

Oppgaven har også sett på eSporingsløsningen og sammenlignet denne med den reviderte arkitekturen. Sammenligningen er det andre resultatet til oppgaven og er brukt til å vise til hvordan autentisering og tilgangskontroll kunne vært gjort annerledes i eSporingsløsningen.

Innholdsfortegnelse

Figurer

Tabeller

Forkortelser

EPC	Electronic Product Code
EPCIS	Electronic Product Code Information Services
RFID	Radio-Frequency Identification
SOA	Service-Oriented Architecture
WS	Web Services
W3C	World Wide Web Consortium
URI	Uniform Resource Identifier
XML	Extensible Markup Language
SSL	Secure Socket Layer
TSL	Transport Layer Security
RBAC	Role-Based Access Control
IM	Identity Management
IAM	Identity and Access Management
SOD	Separation of Duties
PEP	Policy Enforcement Point
PDP	Policy Decision Point
PAP	Policy Administration Point

XACML	Extensible Access Control Markup Language
WS-XACML	Web Services Profile of XACML
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
SSO	Single Sign-On

1 Innledning

1.1 Problemstilling

Autentisering og tilgangskontroll er et ungt tema innenfor tjenesteorienterte arkitekturer. I denne typen arkitekturer er ikke applikasjonene lenger tett koblete. Tradisjonell autentiserings- og tilgangskontrollmekanismer vil nødvendigvis ikke være tilfredsstillende. I tillegg kan andre sikkerhetsaspekter medføre nye utfordringer i distribuerte installasjoner. Oppgaven ser på sikkerhetsaspektene ut ifra en konseptuel arkitektur. Hovedfokuset er autentisering og tilgangskontroll i en tjenesteorientert arkitektur.

eSporing er et prosjekt for å utvikle og implementere en nasjonal elektronisk infrastruktur for sporing av varer i hele matkjeden. Løsningen, heretter kalt eSporingsløsningen, mottar og registrerer elektronisk sporingsinformasjon for varer som forflyttes mellom aktører. eSporingsløsningen er basert på en tjenesteorientert arkitektur.

Resultatet av oppgaven skal sammenlignes med eSporingsløsningen. Oppgaven ser på adresseringen av sikkerhetsaspektene for henholdsvis eSporingsløsningen og en funksjonell arkitektur utarbeidet fra sikkerhetsanalysen. Det vil bli sett på de metoder og mekanismer som er tatt i bruk. Med dette skal det dannes et bilde for hvordan autentisering og tilgangskontroll kan bli utført på en sikker måte i en tjenesteorientert arkitektur.

eSporing skal levere en løsning hvor Web Services tjenester leveres i en felles webportal. Tjenestene kan være enkle, sammensatte, og orkestrerte. Sikkerheten ivaretas ved bruk av et Identity and Access Management system. Systemet legger føringer på tilgangen til ressurser og rettigheter til identiteter i et nettverk. Oppgaven vil belyse disse egenskapene og eventuelt andre essensielle momenter.

1.2 Avgrensning

Sikkerhetsaspekter for vanlige systemer finnes det allerede god nok informasjon på og vil ikke bli berørt. Det er kun gått ut ifra en helhetlig arkitektur for å sette det tjenesteorienterte systemet opp mot de andre systemene i nettverket. I tillegg til å kunne utføre en fullverdig sikkerhetsanalyse.

For ikke å gjøre oppgaven for stor og kompleks kommer den konseptuelle arkitekturen til kun å inneholde enkle momenter. Til gjengjeld vil disse momentene være borti de fleste scenarioene for tjenesteorienterte arkitekturer.

Målet med oppgaven er ikke å finne den ideelle løsningen for autentisering og tilgangskontroll i en tjenesteorientert arkitektur. Isteden er det ønskelig å belyse hvordan beskyttelsen av ressurser kan best mulig oppnås i ulike situasjoner for ulike løsninger. Og eventuelt fylle inn manglende informasjon der hvor det trengs i etterkant.

Det er med dette ønskelig for resultatet av oppgave å gjelde for flere løsninger enn bare eSporingsløsningen. Skulle prosjektet kun ta for seg eSporingsløsningen ville det ikke være en teoretisk arkitektur, men en faktisk implementering av en arkitektur. I så fall ville det være nødvendig med en analyse av de løsningene, produktene og applikasjonene som er tatt i bruk også. Dette fordi en implementering kan introdusere nye svakheter direkte eller indirekte relatert til teknologien den implementere. Dette ville blitt altfor tidkrevende og komplekst.

Der hvor det eventuelt faktisk blir trukket inn standarder for teknologier, som SOAP for Web Services, er det kun på formålet med standardene som er gjeldende. Det vil ikke bli sett på eller diskutert hvorvidt standardene medfører nye trusler eller svakheter til arkitekturen. Med andre ord er det ikke oppgavens formål å vise til svakheter ved disse standardene med mindre det er sagt annerledes i oppgaven.

1.3 Organisering av oppgaven

I kapittel 2 presenteres bakgrunnsinformasjon som er nødvendig eller utdypende for når man leser oppgaven. eSporing i sin helhet vil bli presenter for å gi et bilde over prosjektet og dets forhold til oppgaven før Electronic Product Code Information Services og dets rolle i eSporingsløsningen forklares. Deretter kommer en kort introduksjon til den tjenesteorienterte arkitekturen, hvor komponentene og prinsippene for hvordan arkitekturen realiseres, blir presentert.

Kapittel 3 presenterer en initiell konseptuel arkitektur. Arkitekturen, heretter kalt den generelle arkitekturen, er basert på eSporingsløsningen. Den vil bli brukt til å gi en generell oversikt over de utfordringer autentisering og tilgangskontroll utgjør. Med arkitekturen innlemmes nye scenarioer og situasjoner utover de som det møtes på i sammenheng med eSporingsløsningen.

På bakgrunn av den generelle arkitekturen utarbeides en sikkerhetsanalyse. Resultatet av sikkerhetsanalysen presenteres i kapittel 4. Den generelle arkitekturen danner grunnlaget for definerte sikkerhetskrav som brukes i analysen av autentisering og tilgangskontroll.

I kapittel 5 presenteres en funksjonell arkitektur, heretter kalt referansearkitekturen. Arkitekturen er utarbeidet med utgangspunkt i den generelle arkitekturen. Den innlemmer nødvendig sikkerhetsmekanismer basert på sikkerhetsanalysen.

Sammenligningen av løsningen til eSporing med referansearkitekturen presenteres i kapittel 6.

Kapittel 7 gir avslutningsvis oppsummering og konklusjon.

1.4 Metode

Oppgaven tar i bruk en konseptmodellering for nettopp å kunne fremstille en arkitektur gjeldene for flere enn bare eSporingsløsningen. Arkitekturen gir bedre prinsipielt innblikk enn ved å se på en realisert arkitektur. Ulempen er at implementeringer og produkter brukt for å realisere arkitekturen kan medføre nye og uforutsette problemer. Elementer lar seg kanskje ikke gjennomføre eller kan ikke realiseres uten å introdusere svakheter eller sårbarheter.

Med utgangspunkt i en generell arkitektur tar oppgaven i bruk trusselmodellering. Den metoden legger grunnlaget for utarbeidelse av en sikkerhetsanalyse og etterfølgende referansearkitekturen. Det er noen viktige fordeler med å ta i bruk en allerede definert metode for sikkerhetsanalyse. Ved å ha en velkjente og utprøvd metode for å vurdere sikkerheten, finner man enklere sårbarheter og trusler som kanskje ikke er så opplagte. Man kan også gå systematisk til verks. Da blir det lettere å få oversikten over en stor og kompleks arkitektur.

Denne fremgangsmåten er derimot ikke helt ideell ettersom oppgaven tar utgangspunkt i en konseptuell arkitektur. Den generelle arkitekturen inneholder ikke all funksjonalitet som er nødvendig for en fullverdig løsning. Metoden kan derfor bli veldig målrettet og få en smal vinkling på sikkerhetsanalysen. Dette kan føre til enkelte kritiske momenter blir oversett eller utelatt.

Referansearkitekturen, gir en løsning på problemstillingen. eSporingsløsningen sammenlignes opp mot denne arkitekturen. Dette gir et bredere innblikk i problemene en tjenesteorientert arkitektur medfører og hvordan de kan løses.

2 Bakgrunn

2.1 eSporing[1]

eSporing er et nasjonalt prosjekt mellom norske myndigheter og deltakerne i matindustrien. Prosjektet ledes av Landbruks- og matdepartementet. I tillegg er Fiskeri- og kystdepartementet og Helse- og omsorgsdepartementet deltakere i prosjektets styringsgruppe sammen med sentrale aktører i matkjedene i Norge.

eSporing har som mål å etablere en løsning for effektiv elektronisk sporing av matvarer. Mange aktører har systemer for sporing internt, men ifølge [**1**] har ulike hendelser vist at det er nødvendig med en mer helhetlig løsning. Med en felles løsning kan det raskt og effektivt bli gitt en oversikt over vareflyten for hele verdikjeden. Dette medfører bedre beredskap, bedre mattrygghet, og et mer effektivt tilsyn for forbrukere og myndigheter. For bransjer, aktører i verdikjeden, og tilsynsmyndigheter vil man oppnå gevinster som:

- Bedre kontroll av produktenes innhold, holdbarhet og opprinnelsessted
- Dokumentere kvaliteten til et produkt
- Minke risiko for utrygg mat
- Redusere økonomiske konsekvenser ved tilbaketrekking av produkter
- Muligheter for økonomiske gevinster
- Effektivisere prosess- og produktoptimalisering
- Forbedre logistikk i matvareflyten, både nasjonalt og internasjonalt

Sporing

[**2**] viser til definisjonen for sporing i Matloven som:

> *"Muligheten til å spore og følge et næringsmiddel, et fôr, et dyr bestemt til næringsmiddelproduksjon eller et stoff som er bestemt til eller kan forventes å bli tilsatt næringsmidler eller fôr, gjennom alle ledd i produksjon bearbeiding og distribusjon."*

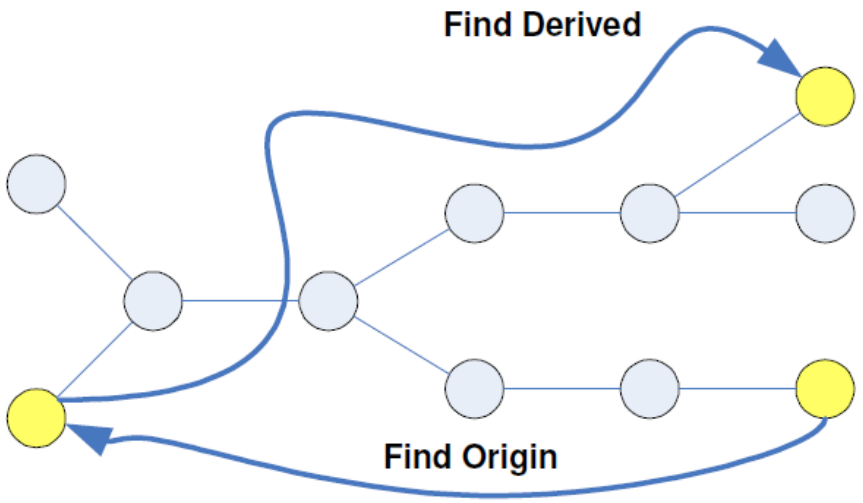
Prosjektet har som krav fra Matloven å kunne spore i alle ledd i verdikjedene. Hvert ledd i kjeden skal kunne spore et ledd bakover og et ledd framover. En bedrift skal med dette kunne se hvem de mottar råvarer fra og hvem de har sendt ferdigproduserte produkter til. eSporingsløsningen skal motta, identifisere og registrere elektronisk sporingsinformasjon av varer som forflyttes mellom

[1] Dette kapittelet er i basert på [**1**], [**19**], [**18**], og [**2**].

aktører. Denne sporbarhetsinformasjonen ligger tilgjengelig i infrastrukturen for eSporingsløsningen og muliggjør sporing av en vare gjennom hele verdikjeden.

Figur 2.1 illustrerer et bilde over vareflyten i en verdikjede. Ved å spore tilbake eller fremover i verdikjedene kan man enkelt danne et bilde over hendelsene til en vare. Et slikt bilde kan kun utarbeides når alle aktørene deltar i løsningen og lagrer sporbarhetsinformasjonen vedrørende varene sine. Hver aktør må lagre hvor varen kommer fra, eventuelle hendelser for hva som er gjort med varen, og til hvem varen er sendt videre til. Minimumssettet av sporbarhetsinformasjon som må tilgjengeliggjøres fra hver aktør er som følger:

- Identifikasjon av aktøren
- Identifikasjon av varen
- Informasjon om hendelsen (transport, lagring, splitting og lignende)
- Informasjon om tidspunktet for hendelsen

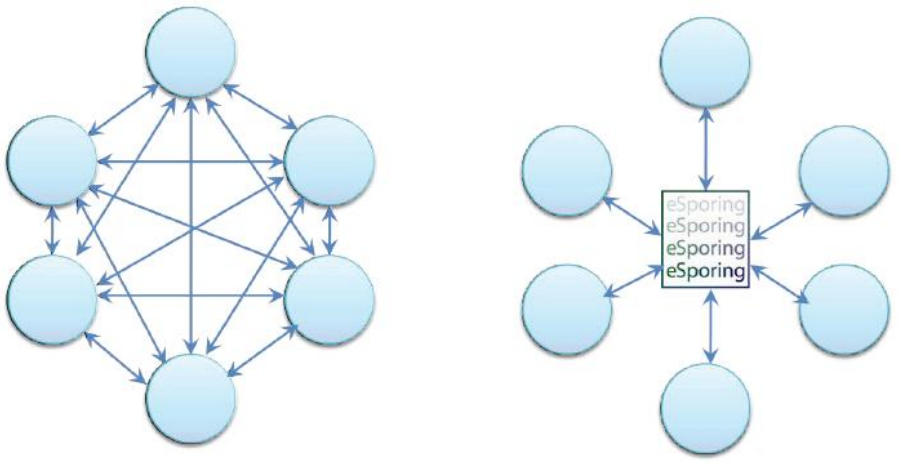


Figur 2.1: Sporing av en vare i en verdikjede. [2]

Infrastruktur

eSporingsløsningen vil bestå av en felles infrastruktur med grensesnitt mot deltakernes interne løsninger for elektronisk sporing. Løsningen skal gjøre sporingsdata tilgjengelig for å kunne følge utbredelsen av en vare mens den forflyttes mellom aktører i verdikjeden.

[2] Hentet fra [**2**], figur 5.



Figur 2.2: Forholdet mellom aktører, med og uten eSporingsløsningen. [3]

Fellesløsningen kjennetegnes av at det blir etablert et nav med et felles kommunikasjonsgrensesnitt basert på anerkjente, internasjonale standarder. Figur 2.2 illustrerer bruken av et nav for deling av informasjon framfor å etablere kommunikasjonskanaler til hver enkelt aktør. Aktørene vil få tilgang til løsningen gjennom bruk av standard grensesnitt, enten fra egne Enterprise Resource Planning systemer, gjennom en tredjepartsløsning, eller via en webportal.

Kravene til løsningen er:

- Bygge på eksisterende systemer
- Tilpasse internasjonale standarder
- Oppfylle kravene til sikker bruk
- Dele oppdaterte sporingsdata på tvers av aktørene
- Gi bedriftene mulighet til tettere samhandling gjennom utveksling av kvalitetsdata
- Ivareta fremtidige myndighets- og markedskrav
- Løsningen skal ikke være kostnadsdrivende

eSporingsløsningen er en tjenesteorientert arkitektur realisert med Web Services og en webportal hvor webtjenestene er tilgjengliggjort.

[3] Hentet fra [**18**].

Standarder
For å kunne realisere en løsning basert på de kravene nevnt ovenfor er det viktig med en standardisert form for identifisering av varer og hendelser som alle parter tar i bruk. Ettersom varene beveger seg mellom leddene i kjeden må de identifiseres med en global og ensartet identifikator. Identifikatoren må ikke bare være unikt for hele løsningen, men også for hele nettverket. Electronic Product Code (EPC) standarden brukes til dette. I tillegg står Electronic Product Code Information Services (EPCIS) standarden sentralt i eSporingsløsningen. Dette er en standard for deling av produktrelatert informasjon og er rettet mot deltakere i et globalt nettverk. Kapittelet 2.2 forklarer disse to standardene nærmere.

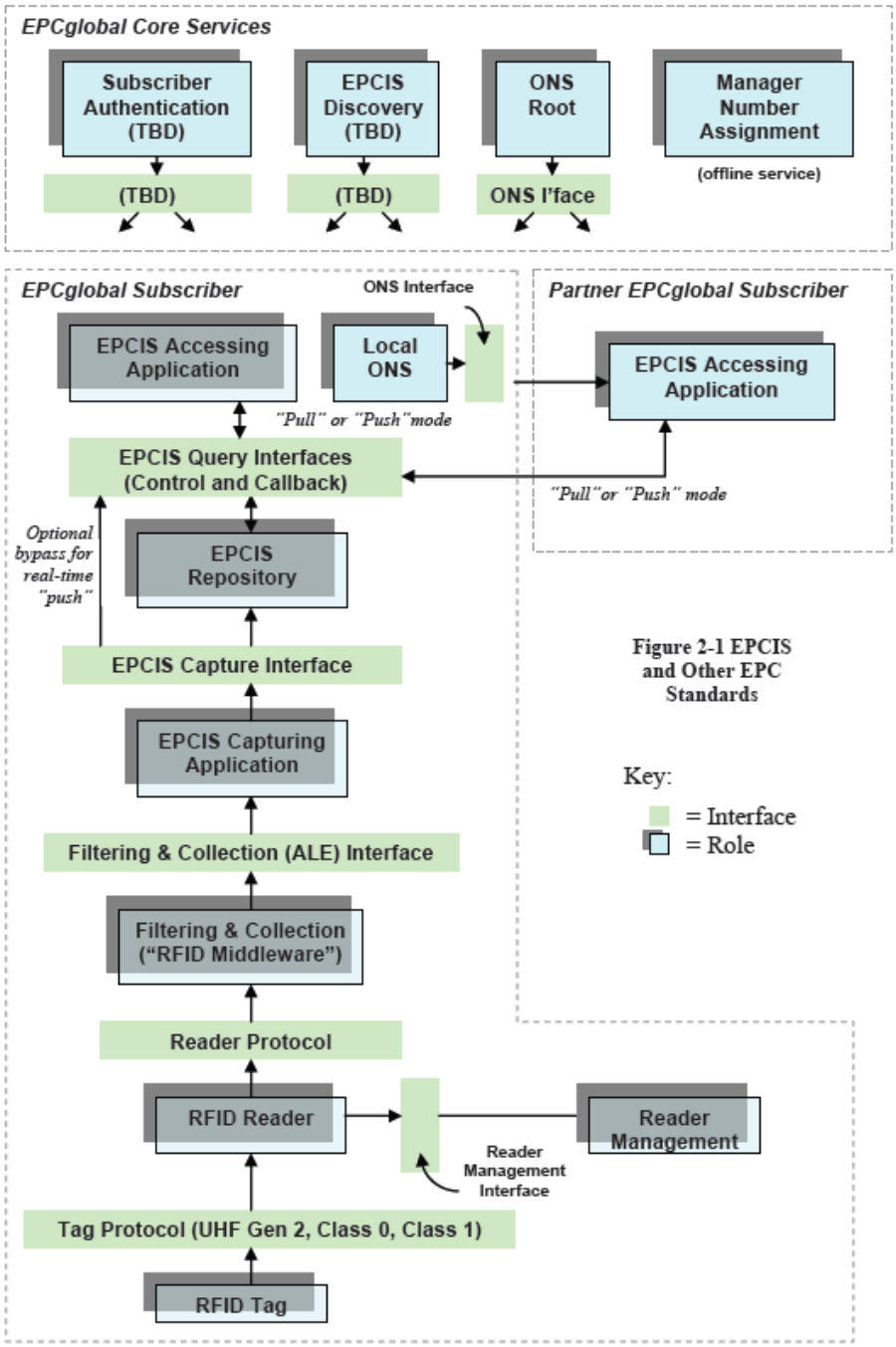
Sikkerhet
Ettersom all sporingsinformasjon samles på et sted for alle aktører, er det viktig å kunne gjøre informasjonen tilgjengelig for kun autorisert aktørene. En aktør står som eier av den sporingsinformasjonen den deler og kan gi tilgang til informasjon for spesifikke varer til andre samarbeidende aktører. En som er autorisert til å se sporingsinformasjonen er dermed enten den aktøren som opprettet sporingsinformasjonen, eller en som har fått tildelt tilgang av eieren til informasjonen. Fra kravet til eSporingsløsningen om sikker bruk innebærer dette at all data må beskyttes med sikkerhetsmekanismer som tilgangskontroll, kryptering og sikker overførsel.

2.2 Electronic Product Code Information Service[4]

EPC standarden er et sett med generelle koding konsepter med mål om å være etterfølgeren til strekkodeteknologien. EPC ble utarbeidet for blant annet å spore varer med Radio-Frequency Identification (RFID) teknologi. EPCIS standarden [**3**] er en global standard for å muliggjøre deling av EPC informasjon mellom ulike aktører. EPCIS definerer et felles syn på hvordan data ser ut og hvordan data utveksles. Målet med EPCIS er å kunne ta i bruk et EPC globalt nettverk. I nettverket vil deltakerne ha et felles syn på objekter med EPC-relatert data i en forretningssammenheng.

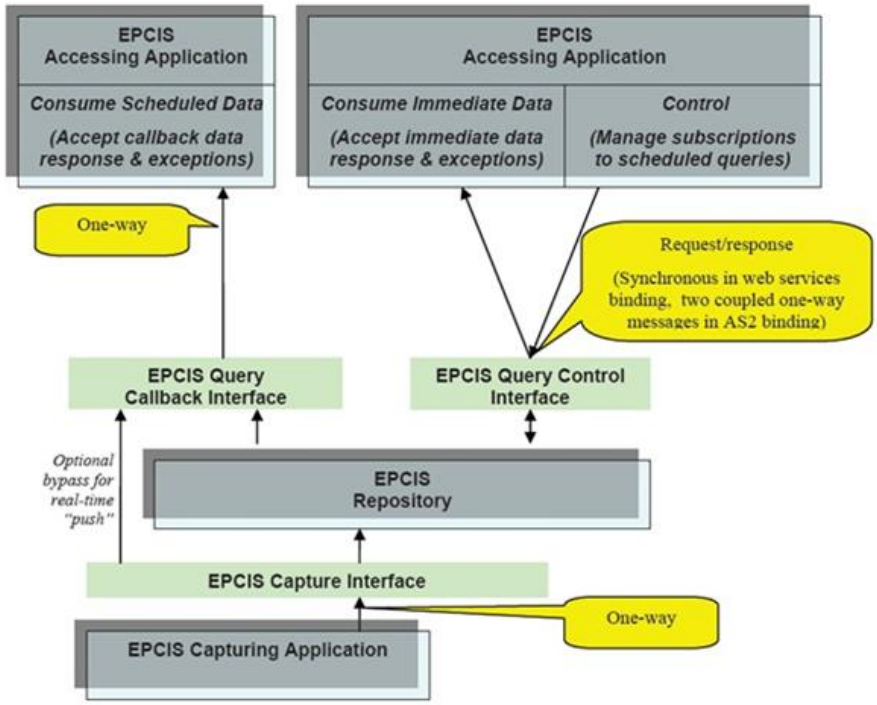
EPCIS standarden beskriver kun hvordan ulike applikasjoner skal kommunisere med hverandre. Den beskriver ikke hvordan EPCIS implementeres eller hvilke løsninger og teknologier som den er bygget opp av. Fordelen med dette er at man oppnår kompatibilitet mellom de ulike aktørene og samtidig inviterer til konkurranse mellom de som skal implementere standarden i sine produkter.

[4] Kapittelet er basert på [**3**].



Figur 2.3: Oversikt over EPCglobal Architecture Framework og EPCIS. [5]

[5] Hentet fra [**2**], kapittel 2.



Figur 2.4: EPCIS og de respektive grensesnittene til ECPIS Repository. [6]

Standarden

EPCIS spesifiserer to typer data kalt hendelses- og masterdata. Hendelsesdata viser til en prosess som er gjort med eller på varen mens masterdata er tilleggsinformasjon nødvendig for å kunne tolke hendelsene. Med dette danner EPCIS grunnlaget for å kunne spore varer. Fra denne sporing er det mulig å finne andre vare som inneholder et spesifikt produkt eller hvilke varer produktet inneholder. Med standarden er det også mulig å hente/lagre informasjon som for eksempel:

- Hvor varen kommer fra
- Hvilke steder (aktører) den har vært innom
- Av hvem og hvordan varen er blitt fraktet mellom aktørene

EPCIS må kunne tas i bruk i eksisterende systemer, uansett miljø. Rammeverket til EPCIS er derfor designet for å være lagdelt, utvidbar, og modulert. I figur 2.3 og figur 2.4 er EPCIS systemet og dets oppbygning illustrert. Figurene viser en

[6] Hentet fra [**2**], kapittel 8.

oversikt over et mulig system hvor det i de laveste lagene er spesifisert hvordan RFID kan brukes for å registrere og videresende data til de andre modulene. Med EPCIS Capturing Application modulen mottas og prosesseres innfanget data. Den neste modulen, EPCIS Repository, lagrer EPCIS relatert data. Til slutt blir hendelses- og masterdata tilgjengeliggjort med ECPIS Accessing Application til de applikasjonene som ønsker å ta i bruk disse dataene. Den siste modulen brukes av applikasjonene enten det er for å utføre spørringer eller for å motta notifisering på abonnerte hendelser.

I sammenhengen med denne oppgaven er det hvordan man henter ut hendelses- og masterdata fra en EPCIS Repository som er relevant. Fra figur 2.4 ser vi at disse dataene er tilgjengelig gjennom EPCIS Query Control Interface og EPCIS Query Callback Interface grensesnittene. Grensesnittene er henholdsvis for å utføre spørringer og abonnere på hendelser, og for å notifisere om hendelser som det er abonnert på.

2.3 Service-Oriented Architecture[7]

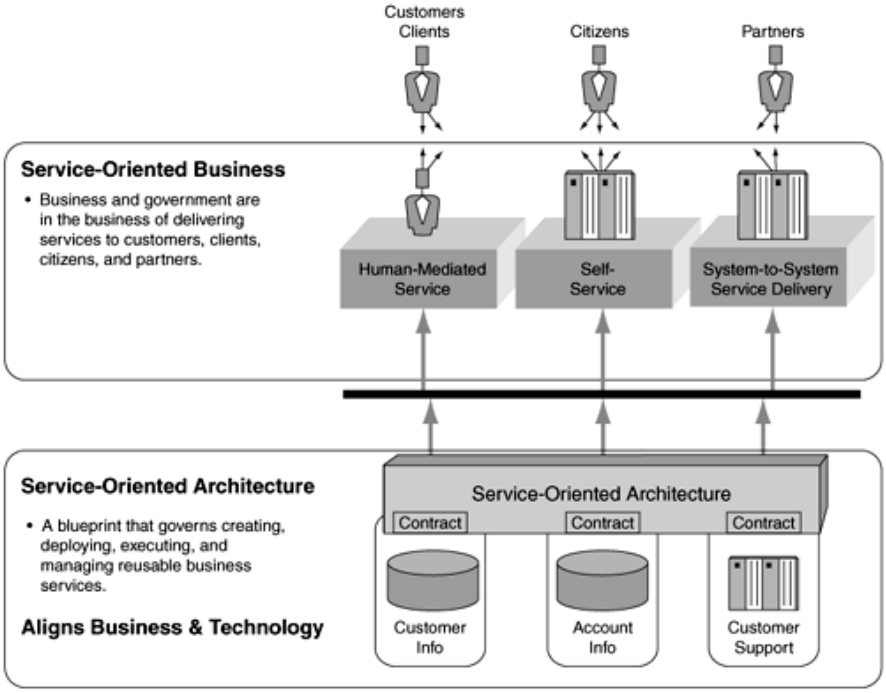
Service-Oriented Architecture (SOA), også kalt tjenesteorientert arkitektur, er ofte sett på i relasjon til forretningsdelen av en programvareløsning. I denne sammenhengen er SOA et verktøy for å levere de krav og den funksjonalitet klientene ønsker. Figur 2.5 viser en illustrasjon av hvordan SOA kan bli brukt for å levere løsninger til ulike domener med en felles infrastruktur og tjenester. SOA prøver å holde følge med endrende krav og miljø, både innenfor og utenfor forretningen, ved å lette utviklingen og håndteringen av løsninger. Dette gjøres ved å dele opp store prosjekter og oppgaver i mindre deler og samtidig få hver del til å fungere med hverandre, uavhengig av hvilke plattformer og teknologier som er brukt. I tillegg letter SOA integreringen av nye systemer og applikasjoner mot eksisterende systemer. SOA brukes også som en mekanisme for å kunne tilby interne tjenester til andre utenfor forretningen for å øke gevinsten og utnyttelsen av tjenestene.

2.3.1 Teknisk perspektiv

Fra et mer teknologisk ståsted kan SOA sies å være et sett med design prinsipper som fører til løst oppbygd sett med tjenester som kan bli brukt i ulike forretningsdomener. Med dette kan for eksempel en løsning bli realisert ved ulike tjenester som er laget med ulike implementeringsspråk og gjort tilgjengelig på ulike SOA plattformer. Isteden for å ta utgangspunkt i en spesifikk teknologi for å realisere en tjeneste, defineres tjenesten gjennom et grensesnitt.

[7] Kapittelet er basert på [**16**], [**15**], og [**17**].

Grensesnittet spesifiserer operasjonene til tjenesten og hvilke protokoller som må bli brukt for å ta dem i bruk. Med SOA oppnår man dermed tjenester som integrerer sømløst med hverandre.



Figur 2.5: SOA og dets forhold til løsninger på ulike nivåer. [8]

Ved å ta i bruk en arkitektur basert på SOA vil man promotere samspillevne, gjenbruk, og fleksibilitet, samt overse problemer som følge av:

- Distribuerte løsninger
- Applikasjonsintegrasjon
- Ulike plattformer
- Ulike protokoller
- Ulike enheter

Det som skiller tjenesteorienterte løsninger fra andre løsninger er hvordan separeringen av løsningen er oppnådd. Logikken for å utføre en oppgave er dekomponert i mindre, relaterte enheter hvor hver av dem adresserer en spesifikk del av et problem. På bakgrunn av dette oppfordrer tjenesteorienterte

[8] Hentet fra [**17**], figur 2-2.

løsninger til å la enheter eksistere autonomisk og uavhengig av hverandre. Samtidig må tjenestene være i stand til å kommunisere og virke sammen på en felles måte som om alle enhetene til sammen utgjør en stor ensartet enhet.

Prinsipper
Det er blitt anerkjent et sett med nøkkelprinsipper som skal forme og standardisere hvordan SOA komponenter lages. Prinsippene er presentert i Tabell 2.1.

Merk at disse prinsippene bygger på hverandre og når sett i helhet, vil de i hovedsak beskrive konseptet bak SOA og hvordan det er ment til å fungere. Med disse prinsippene etablert, kan man oppnå denne samspillevnen mellom ulike foretninger, applikasjoner, løsninger, og tjenester. Prinsippene kan sees på som retningslinjer for hvordan løsninger bør realiseres.

Hovedkomponenter
Hovedkomponentene i en SOA basert arkitektur er tjenester, meldinger, kontrakter og tjenestemeglere.

Tjenester er enheter med logikk som kan være satt sammen av andre tjenester eller være helt uavhengig. En tjeneste kan bli implementert på en hvilken som måte så lenge den følger de SOA prinsipper og retningslinjer som er gitt. Hvis disse prinsippene og retningslinje ikke er fulgt, kan ikke tjenesten lenger bli identifisert som en SOA tjeneste.

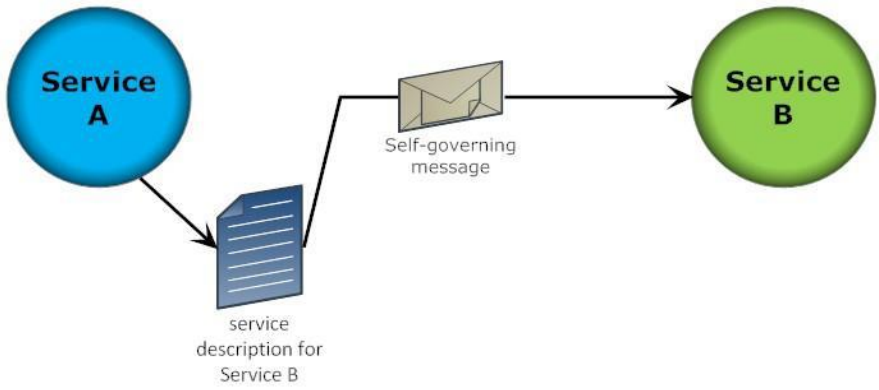
Avhenging av sammenhengen for hvordan en tjeneste blir brukt og dens rolle i en interaksjon, vil tjenesten enten være en tjenestebruker eller en tjenestetilbyder. En tjenestetilbyder er assosiert med ansvaret for å publisere sine egenskaper til andre tjenester gjennom en tjenestekontrakt. En tjeneste som tar initiativet til å starte en samtale med en annen tjeneste, er assosiert som en tjenestebruker.

Som følge av de prinsipper som er gitt er SOA en meldingsorientert arkitektur. Kommunikasjonen mellom de ulike tjenestene utførers med meldinger. Valget for å ta i bruk meldinger for å kommunisere innad i et SOA system, er blant annet for gjøre arbeidet med å lage tjenester minst mulig komplisert. Utviklerne kan fokusere på funksjonalitet til tjenesten istedenfor for på kommunikasjon og kommunikasjonsprotokoller.

Tabell 2.1: SOA prinsipper

Tjenestekontrakt	En representasjon av en tjenestes metadata som beskriver hvordan man kan ta i bruk tjenesten og hvilke betingelser som må være oppfylt for å utføre en operasjon. Kontrakten viser til hvordan interaksjonen med tjenesten skal utføres.
Svak kopling	Tjenester som har en relasjon med hverandre som minimaliserer avhengigheten mellom dem har en svak kopling. Ut ifra kontrakten til en tjeneste vet man at en tjeneste eksisterer og hvordan man kan kommunisere med den.
Abstraksjon	Innkapsler mulig kompleks prosesseringslogikk og eksponerer denne logikken gjennom et generisk og beskrivende grensesnitt. Dette skjuler logikken fra utsiden.
Autonomi	Rettet mot selve underlaget av tjenesten og viser til når en tjeneste har kontroll over logikken den omslutter. Tjenester realiseres ved å la dem ha fullstendig kontroll over sitt eksekverende miljø.
Tilstandsløs	Tjenester som minimalt trenger å ta vare på informasjon relatert til en aktivitet. Eventuell informasjon om tilstandene flyttes til meldingen isteden for å bli tatt vare på i selve tjenesten.
Oppdagbar	Tjenester skal kunne oppdages og brukes av andre uten videre kjennskap om tjenesten. Dette muliggjør for dynamisk interaksjon mellom tjenester etter hvert som de blir lagt til, endret, og fjernet i et system hvor oppdaging av andre tjenester er mulig.
Gjenbruk	Tjenesters omsluttende logikk kan bli brukt av flere enn en. Deler opp logikken til en oppgave for å slippe å måtte implementere de samme delene på nytt i en annen løsning.
Sammensatte	Tjenester kan bli satt sammen av eksisterende tjenester. En tjeneste kan bli linket og koordinert til et sett med spesialiserte tjenester for å utføre en sammensatt oppgave.

En beskrivelse av en tjeneste, kalt en kontrakt, inneholder metadata til tjenesten og viser til de karakteristikker tjenesten har. Kontrakten beskriver hvordan selve kommunikasjonen skal utføre, hvilke meldinger som skal bli brukt, og hvordan disse meldingene ser ut. Figur 2.6 viser til relasjonen mellom de tre komponentene tjeneste, kontrakt, og melding. Fra figuren ser vi tjeneste A forholder seg til kontrakten til tjeneste B og ikke selve tjenesten. Meldinger blir brukt for å kommunisere mellom de to tjenestene.



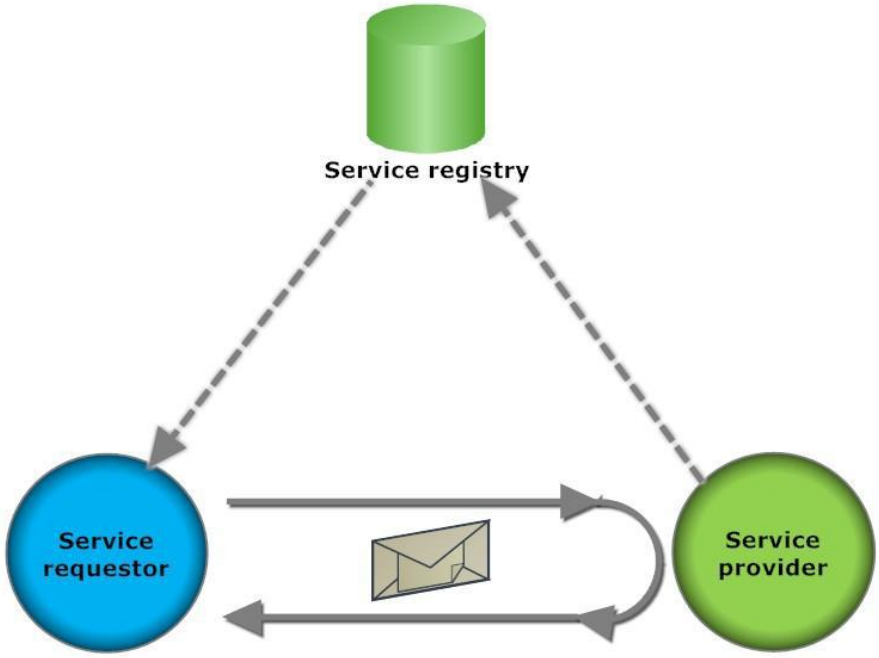
Figur 2.6: Relasjonen mellom tjenester, meldinger, og kontrakter i SOA.

For å gjøre tjenestene oppdagende av andre tjenester, tas det i bruk en tjenesteformidler i et SOA system. Se figur 2.7 for en illustrasjon av dette publiser-finn-bind paradigmet. Tjenesteformidleren vil ta vare på alle kontraktene til de tjenestene som er tilgjengelig slik at tjenester kan oppdages dynamisk. Ved å la en tjenestetilbyder publisere kontrakten sin til en tjenesteformidler, kan en tjenestebruker binde seg til denne tjenesten når den skal bli tatt i bruk. Dette fører til en arkitektur hvor tjenester ikke lenger er linket fast til hverandre, men er isteden linket til beskrivelsen av en tjeneste. Tjenester kan dermed bli lagt til eller fjernet uten å måtte gjøre endringer på de som tar i bruk tjenesten.

Selve programflyten for dette blir:

1. En eller flere tjenestetilbydere publiserer tilgjengeligheten av tjenestene sine
2. En tjenesteformidler registrerer og kategoriserer publiserte tjenester og tilbyr søketjenester mot disse registrene
3. En tjenestebruker kontakter en tjenesteformidler for å finne en ønsket tjeneste og for så å ta i bruk denne tjenesten

Ved å ta i bruk tjenesteformidlere kan dette også føre til et system hvor man har flere ulike tjenestetilbydere som har den samme beskrivelsen, men ulike egenskaper, og hvor en tjenestebruker kan velge den tjenesten som best passer til oppgaven.



Figur 2.7: Tjenesteformidler i et SOA system.

2.3.2 Web Services

Web Services (WS) [**4**] er en teknologi som kan bli brukt for å realisere en arkitektur basert på SOA og i hovedsak bygger på World Wide Web standarder. I [**4**] definerer World Wide Web Consortium (W3C) en WS tjeneste, heretter kalt en webtjeneste, som:

> *"A Web service is a software system identified by a Uniform Resource Identifier (URI), whose public interfaces and bindings are defined and described using Extensible Markup Language (XML). Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."*

Forskjellen mellom Web Services og SOA er at SOA er et konsept ment for å være teknologinøytralt mens Web Services er bygget opp rundt de webteknologier som finnes. Uansett, begge deler de sammen prinsippene og konseptene.

En tjeneste implementert som en webtjeneste, se figur 2.8, inneholder en helhetlig logikk som er tilgjengeliggjort gjennom en kontrakt til tjenesten, en definisjon. Definisjonen, i Web Services sammenheng, er et Web Services Description Language (WSDL) dokument som er bygget opp etter tilhørende XML Schema. I tillegg brukes WS-Policy for å spesifisere hvordan webtjenesten kan bli brukt. Disse standardene, bruken av SOAP som meldingsformat og URI for identifisering av tjenester, leder til universelle identifikasjons- og kommunikasjonsmekanismer.

Figur 2.8: Illustrasjon av en webtjeneste med tilhørende standarder.

3 Generell arkitektur

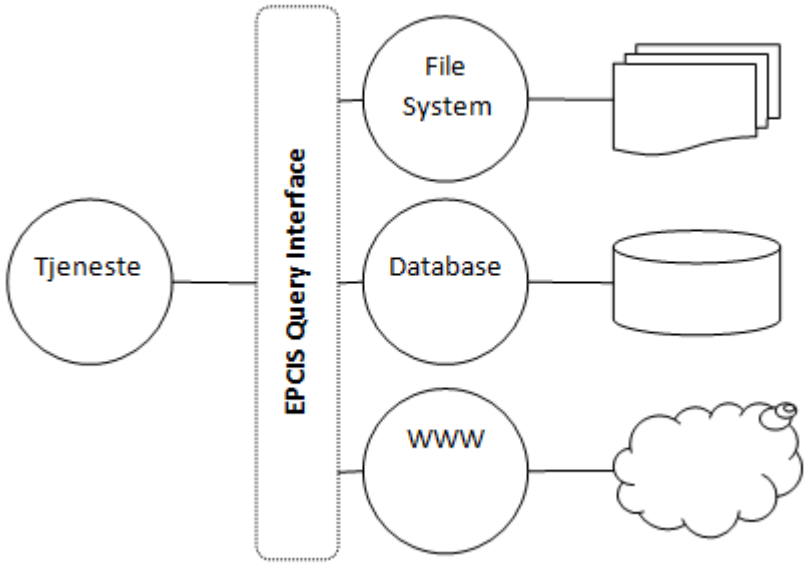
Utgangspunktet for den generelle arkitekturen er eSporingsløsningen. Med utgangspunktet utvides arkitekturen til å dekke de fleste variasjoner som kan spille en vesentlig rolle for autentisering og tilgangskontroll i en tjenesteorientert arkitektur. Dette gir nye scenarioer utover de som møtes på i sammenheng med eSporingsløsningen. Kun de mest brukte og avgjørende elementene i SOA tas med. Arkitekturen omhandler også de underliggende lagene som kan spille en rolle i autentisering eller tilgangskontroll. Dette gjøres for å trekke sammenhenger mellom det tjenesteorienterte systemet og de omsluttende systemene.

3.1 Grunnlaget

Basis for den generelle arkitekturen er EPCIS standarden. I sammenheng med oppgaven er det EPCIS Repository, se figur 2.4, som er det sentrale elementet fra EPCIS standarden. EPCIS Repository vil stå for lagringen av sporings- og vareinformasjon i arkitekturen. Standarden spesifiserer ikke hvordan komponenten implementeres, kun hvordan man kommuniserer med den. Denne kommunikasjonen kan sees på som en tjeneste i seg selv. Grunnlaget for arkitekturen er dermed lagt. Den generelle arkitekturen bygges opp rundt en SOA tjeneste som bruker en database hvor hendelses- og masterdata er lagret. Denne tjeneste, med databasen, utgjør EPCIS Repository.

Denne modellen likner på en hver annen modell i andre løsninger for eksempelvis lagring av data. Vi kan dermed utnytt dette til vår fordel for å lage en generell arkitektur. Begrunnelsen for dette er som i de fleste systemer eller løsninger hentes det gjerne data fra en fil, en database, et register, fra minnet, eller fra andre steder. Felles for de alle er bruken av grensesnitt mot oppgavene. I grensesnittene spesifiseres hvilke metoder eller funksjoner som kan brukes mot en datalagringsplass. Men, man spesifiserer ikke hvordan data skal lagres eller med hvilken teknologi eller format data skal lagres på.

Eksempelvis kan man se på operativsystemet som et grensesnitt til en harddiskstasjon. Grensesnittene i operativsystemet definerer metoder som blant annet oppdager filer og kataloger, finner størrelsen på filer, og åpner opp strømmer til eller fra en filer. Når en applikasjon tar i bruk disse metodene er det likegyldig hvilket format filsystemet harddiskstasjon har.



Figur 3.1: Bruk av grensesnitt for abstraksjon av lagringsmedium.

Figur 3.1 illustrerer dette ved at man har ulike tjenester som kommuniserer med ulike systemer og hvor tjenestene tar i bruk et felles grensesnitt innad i systemet. For denne figuren har tjenestene et EPCIS grensesnitt som lar andre tjenester hente ut hendelses- og masterdata fra henholdsvis et filsystem, en database, og fra en tjeneste eller applikasjon tilgjengelig fra et annet sted på weben.

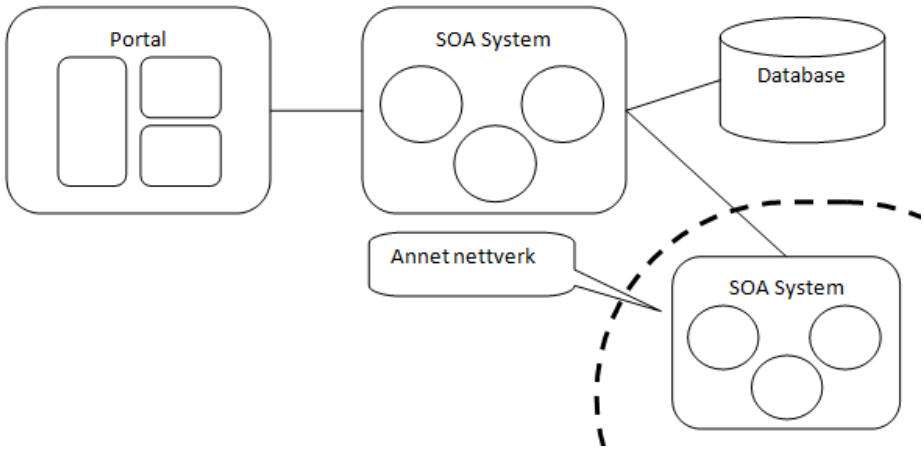
Med denne modellen er ikke arkitekturen begrenset til bare løsninger med EPCIS standarden. Formatet, lagringsmetodene, eller tjenestene kan byttes ut, men selve strukturen for hvordan man tar i bruk grensesnitt er den sammen.

3.2 Arkitekturen

Arkitekturen er laget på bakgrunn av eSporing sin mest sentrale oppgave, sporing, og vil kun tilby denne funksjonaliteten. Dette gjøres på en slik måte som tar for seg de mange ulike scenarioene som kan oppstå ved å ta i bruk SOA tjenester. Med andre ord sees det bort ifra annen funksjonalitet som er nødvendig for et helhetlig system. Eksempelvis hvordan innfanging og lagring av data utføres.

Figur 3.2 viser den generelle arkitekturen. Utenfra nettverket møtes det først på en portal. Portalen vil inneholde de SOA tjenestene en bruker kan ta i bruk. Tjenesten og eventuelle andre SOA tjenester som ikke er vist i portalen er alle tilgjenglig fra SOA systemet. Tjenestene kan ta i bruk databaser lokalt i

nettverket for å hente ut sporingsdata, eller de kan ta kontakt med systemer på andre nettverk. For eksempel kan en annen aktør ha lagret tilleggsinformasjon som EPCIS masterdata på et annet nettverk som tjenesten skal ha tak i.



Figur 3.2: Den generelle arkitekturen.

For å sette arkitekturen i perspektiv er målet lik målet til eSporingsløsningen: å opprette et felles nettverk hvor aktører legger inn hendelser, og eventuell tilleggsinformasjon, til varene for senere å kunne spore en varer gjennom hele verdikjeden. Dette felles nettverket betegnes som et nav. Med navet slipper man å måtte opprette forbindelser til alle de andre aktørene for å kunne utføre sporing av varer.

Det som er verdt å bemerke seg er hvordan det er SOA tjenesten som står sentralt og har hovedfokuset. De andre systemene kun gir et generelt syn på hvordan SOA systemet forholder seg til dem. Dette gjør det mulige å kunne utarbeide en helhetlig sikkerhetsanalyse.

Neste følger en mer detaljert beskrivelse av arkitekturen og de ulike komponentene for henholdsvis SOA systemet alene, og for de resterende systemene påfølgende.

3.2.1 SOA systemet

SOA systemet kan i realiteten bestå av kun en tjeneste. Hvis det er nødvendig eller påkrevd å ta i bruk et SOA system eller hvis ikke tjenesten er så stor med tanke på å dele den opp i mindre deler, er det ikke noe galt i kun å levere en enkel tjeneste. Det mer normale for SOA systemer er nok heller å ta i bruk mange mindre og mer spesialiserte tjenester som er blitt sammensatt til større

tjenester. De sammensatte tjenestene er så gjort tilgjengelig for applikasjoner og andre tjenester utenfor SOA systemet. Dette er mer i tråd med SOA filosofien og hvordan SOA bør implementeres.

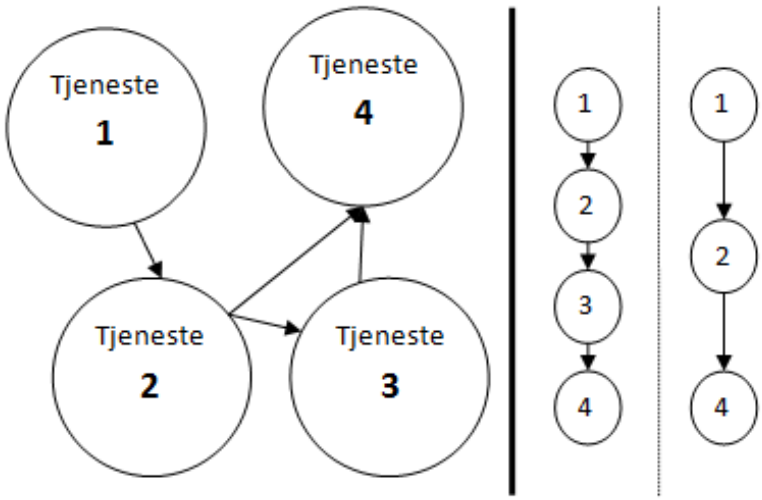
Grunntjenesten

Ettersom arkitekturen bygges opp rundt ECPIS Repository, lages det først en tjeneste som skal gi tilgang til et lagringsmedium gjennom ECPIS Repository grensesnittene. Denne tjenesten, i tro med SOA filosofien, kan gjenbrukes og er dermed ikke låst til for eksempel en spesifikk database. Ønsker man å ta i bruk enda en database kan man bare starte opp en ny instans av denne tjeneste rettet mot den andre databasen.

Sammensatte tjenester

På lik linje med EPCIS Repository tjenesten, lages det små og enkle tjenester som tar i bruk de eventuelle tjenestene som allerede er laget, kun for å tilføre ekstra funksjonalitet. Til slutt har man tjenester som kan tilgjengeliggjøres utenfor SOA systemet og utfører oppgaver som tar i bruk blant annet EPCIS Repository tjenesten.

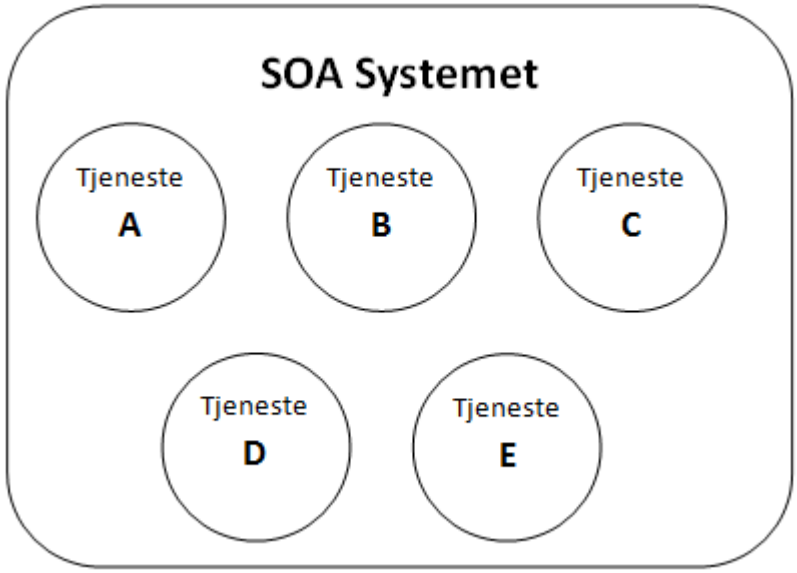
Figur 3.3 viser til en enkel sammensetning av tjenester og de tjenestekall som blir utført. Dette har ikke direkte noe med den generelle arkitekturen og gjøre, men er kun tatt med for å illustrere hvordan et tjenestekall på en sammensatt tjeneste kan se ut.



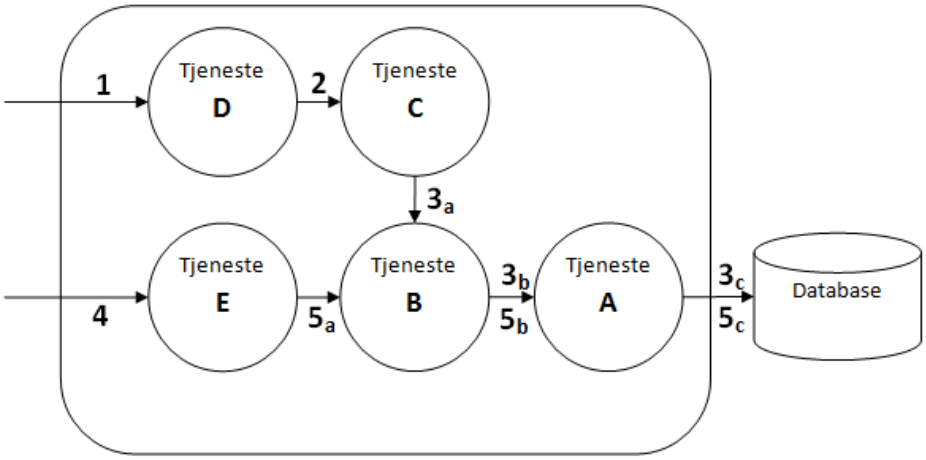
Figur 3.3: Eksempel på sammensatte tjenester og tjenestekall.

Ved å ta i bruk denne formen for sammensatte tjenester vil systemet trenge tjenester for å hente ut lagrede data, behandle data, og presentere resultatet. Figur 3.4 viser de SOA tjenestene som er med i systemet til den generelle arkitekturen. Følgende tjenester er tatt med, indeksert etter tjenestene i figuren:

A. Tjeneste som gjør det mulig å hente ut data fra en database. Tjenesten trenger nødvendigvis ikke kun være tilgjengelig for de lokale tjenestene, men kan også være tilgjengelig for andre tjenester på andre nettverk. EPCIS har blant annet denne funksjonaliteten for EPCIS Repository hvor den kan bli bruk av andre tjenester utenfor det lokale nettverket.
B. Tjeneste som kan spørre etter vareinformasjon eller hendelser til en vare. Bruker tjenesten som tilgjengeliggjør ECPIS grensesnittet mot et lagringsmedium.
C. Tjeneste for å hente alle hendelsene til en spesifisert vare.
D. Tjeneste for å spore en vares hendelser. Presenterer hendelsene til en vare på en oversiktlig og strukturert måte.
E. Tjeneste for å hente ut og presentere informasjon til en vare.



Figur 3.4: SOA systemet til den generelle arkitekturen.

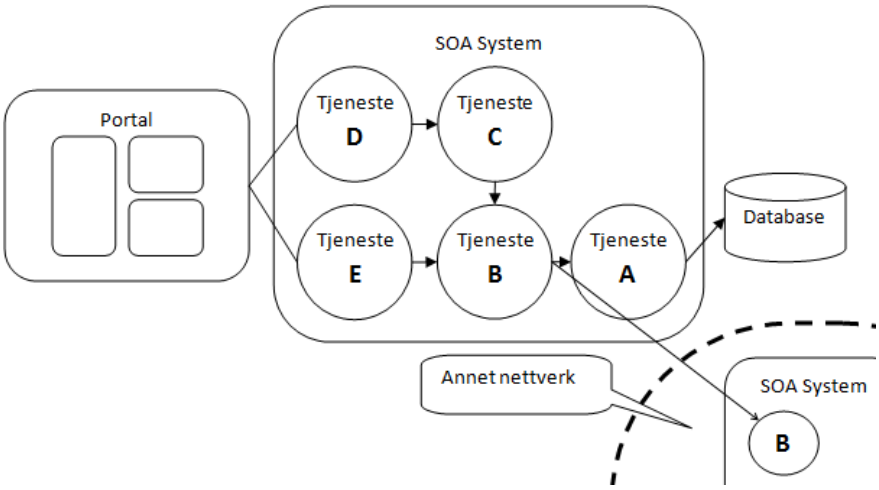


Figur 3.5: Eksempel på hendelsesforløp for sporing av en vare.

Merk at selv om det er sporingstjenesten som er hovedtjenesten, er det nødvendig med tjeneste C for å kunne utføre en helhetlig oppgave. Etter å ha sporet opp en vare, kan det være nødvendig med å hente ut informasjonen til varen. Eksempel på et hendelsesforløp, illustrert i figur 3.5, er:

1. Sporingstjenesten blir kalt for å spore en spesifisert vare.
2. Tjenesten sender en forespørsel til tjeneste C for å finne alle hendelsene til varen.
3. Tjeneste C tar i bruk tjeneste B, som igjen tar i bruk tjeneste A, for å hente ut hendelsene til varen fra database.
4. Etter at en vare er sporet opp, kalles en ny tjeneste for å vise vareinformasjonen til varen som er blitt sporet opp.
5. Tjenesten tar i bruk tjeneste C, men denne gangen for å hente ut vareinformasjonen fra databasen.

SOA systemet består nå av et antall mindre og mer spesialiserte tjenester som er satt sammen til å tilby ulike generelle tjenester. Hele SOA systemet, med alle tjenestene og relasjonene mellom dem, er illustrert i figur 3.6.



Figur 3.6: SOA systemet og relasjonen mellom tjenestene.

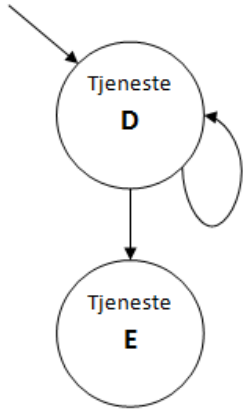
Orkestrering

For å ta i bruk flere av egenskapene som SOA tilbyr, tas det også med orkestrering av tjenester. Dette er en viktig del av SOA. Men orkestrering av store systemer kan ofte bli relativt komplisert. Derfor holder den generelle arkitekturen seg til en enkel orkestrering uten store finesser. Orkestreringen av tjenestene, illustrert i figur 3.7, er som følger:

1. Det startes med å spore en vare ved å kalle tjeneste D.
2. Deretter kan man velge å spore en ny vare fra verdikjeden som er utarbeidet av tjeneste D, eller man kan gå videre. Antall ganger man kan sporing en ny vare er det satt ingen begrensinger på.
3. Avslutningsvis kaller man tjeneste E for å hente vareinformasjonen.

Resultatet av dette SOA systemet er en dynamisk løsning hvor man kan introdusere eller fjerne lagringsmedier for EPCIS data dynamisk og hvor disse mediene ikke nødvendigvis trenger å være lokalt i nettverket.

Det er bevisst utelatt tre momenter ved SOA tjenesten. Det første er valget av ikke å ta i bruk tjenester med tilstand. Tjenester uten tilstand er mye enklere å utvikle og forholde seg til. Det andre er valget av ikke å bruke tjenester med konversasjon. Bakgrunnen for dette er den samme som for tjenester uten tilstand. Det tredje er ikke å ta i bruk tjenestemeglere i SOA systemet.



Figur 3.7: Orkestrering av SOA tjenester i den generelle arkitekturen.

3.2.2 Andre systemer

En arkitektur består som regel av andre komponenter også. Disse komponentene kan til en viss grad sees på som tjenester i seg selv, men er langt ifra å være SOA tjenester. Disse tjenestene hører til i andre systemer enn systemet for SOA tjenestene.

Ifølge beskrivelsen av eSporing sin løsning skal det leveres en webportal som tar i bruk de ulike webtjenestene for sporing og eventuelt andre tjenester. For å innlemme denne webportal i den generelle arkitekturen er det valgt å ha en portalkomponent som er teknologiuavhengig. Se figur 3.2 for oversikten over den generelle arkitekturen. For eksempel kan dette være en vanlig webserver eller det kan være nok en SOA tjeneste. Meningen er i hvert fall at komponenten ikke nødvendig vis er i det sammen system som SOA tjenestene og sees derfor på som en utenforstående komponent.

Det kan også hende løsningen er avhengig av tjenester eller komponenter som ligger et annet sted. For eksempel er det i den generelle arkitekturen tilgjengeliggjort tilgang til et eller flere eksterne nettverk. I eSporing kan dette da svare til systemene lokalt for hver enkelt produsent eller aktør. Er det ønskelig å få mer informasjon om et produkt eller sending kan SOA tjenesten ta kontakt med systemet til den aktøren som har data om det produkt. Systemet til aktøren kan for eksempel tilby en SOA tjeneste som implementer et EPCIS Query Control Interface og dermed gjør det mulig for eSporing og hente ut mer informasjon enn det som er lagret lokalt. Dette i seg selv kan fører til både bedre og dårligere sikkerhet, men er utelatt ettersom dette kan sees på som en vanlig tjeneste, tilhørende en annen sikkerhetssone.

4 Sikkerhetsanalyse

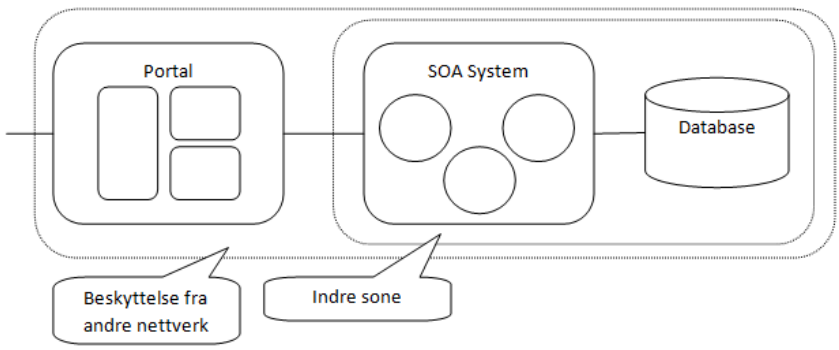
I dette kapittelet presenteres resultatet av sikkerhetsanalysen. I den første delen av kapittelet presenteres et trusselbilde for den generelle arkitekturen. I likhet med en trusselmodell vil denne delen belyse sider ved arkitekturen som kan lede til sikkerhetsrisikoer. Her identifiseres også de ressursene man ønsker å beskytte. Trusselbildet gjelder for hele arkitekturen mens de resterende delene er kun rette mot den tjenesteorienterte delen av arkitekturen. Etter trusselbildet vises det til de sikkerhetskrav det er gått ut ifra. Avslutningsvis vil de to siste delene presentere resultatet av analysen for henholdsvis autentisering og tilgangskontroll.

4.1 Trusselbilde

Med trusselbilde er målet å peke på de generelle punktene som bidrar til økt sikkerhet. Bildet viser også til de svakheter arkitekturen står overfor og gjør oppmerksom på eventuelle spørsmål som det må tas stilling til ved realiseringen. Trusselbildet er med andre ord ikke uavhengig av de realiseringsteknologier som er brukt, men danner grunnlaget for en ny sikkerhetsvurdering.

Figur 4.1 viser til den generelle arkitekturen, men nå med sikkerhetssoner for de ulike systemene og ressursene. De ressursene som det i første omgang er ønskelig å beskytte, er:

- Systemene fra angripere som ønsker å ta kontroll over dem.
- Hindre brukere fra å ta i bruk tjenester som de ikke har rettigheter til.
- Hindre tilgang til sensitiv data og data som brukere ikke er eiere av eller er gitt tilgang til.



Figur 4.1: Den generelle arkitekturen, med sikkerhetssoner.

4.1.1 Sikkerhetsprofil

En sikkerhetsprofil beskriver fremgangsmåten for hvordan sikringen av resursene utføres. Innholdet i en slik profil omhandler blant annet input validering, autentisering, tilgangskontroll, konfigurasjonshåndtering og andre områder som kan være sårbare. De kontrollene og mekanismene man kommer fra til er ment for å sikre applikasjonene mot eventuelle sårbarheter.

Ettersom sikkerhetsanalysen er utført på en teoretisk arkitektur er det begrenset hvor mye av sikkerhetsprofilen som kan utarbeides. Det vises derfor til en blanding av momenter for hvordan man kan sikre applikasjonene og hvilke sårbarheten som er gjeldende. De sårbarhetene som i første omgang applikasjonene kan være mottakelige mot, er presentert i Vedlegg A. Tabellen viser til spørsmål som bør stilles når man analysere en faktisk implementering av arkitekturen. Spørsmålene kan deles inn i kategorier for applikasjonssårbarhet og er behjelpelig for å komme fram til design- og implementeringstilnærmelser mot sårbarhetene.

4.1.2 Trusler

For å gi en oversikt over de truslene arkitekturen står ovenfor, inneholder tabell 4.1 de mest brukte truslene per dags dato. Tabellen er delt inn i tre kategorier etter hvilken komponent som er mottaklig for trusselen. Det gis i tillegg en enkel beskrivelse av hver trussel. Merk at dette bare er et utdrag av de topp rangerte truslene. I tillegg kan man ta i bruk andre kilder som SANS liste over de mest kritiske sårbarhetene [**5**] og de mest gjeldende programmeringsfeilene [**6**] for å danne gode rutiner for sikker programmering og realisering av løsninger.

Hvordan man skal forsvare seg mot slike trusler kan utføres på mange forskjellige måter. En naturlig måte å gjøre dette på, uten å gå spesielt inn på detaljer, er å la hvert system stå for en del av sikkerheten og dermed lagvis sikre systemene. Et eksempel på dette kan være en arkitektur hvor:

- Brannmur og svitsjer sikrer de andre systemene i nettverk fra omverdene og eventuelt fra hverandre med sikkerhetssoner
- Web serveren står for autentisering av brukere ved bruk av sterk autentiseringsmekanisme og eventuelt nekter anonyme brukere tilgang
- SOA systemet sørger for hele tiden å sjekke om tjenestene kan kalles ved å ta i bruk tilgangskontrollmekanismer

- Databasen sjekker om bruker har de rettighetene som trengs til å hente ut eller sett inn data

Tabell 4.1: Mulige trusler mot den genrelle arkitekturen. [9]

	Trussel	Beskrivelse
Nettverk	Information Gathering	Nettverksenheter kan bli oppdaget og profilert (Profiling). På bakgrunn av denne informasjonen kan en angriper finne og eventuelt angripe sårbarheter til de ulike enhetene. Eksempelvis vil en angriper starte med å finne hvilke typer system nettverket inneholder, hvilket operativsystem og hvilke applikasjoner som kjøres. Deretter søker angriperen etter kjente sårbarheter som kanskje ikke er blitt utbedret med sikkerhetsoppdateringer.
	Session Hijacking	Også kjent som "man-in-the-middle" angrep. Kjennetegnes ved at en angriper plasserer seg mellom kommunikasjonen til to parter og utgir seg for å være den respektive motparten. Ved å manipulere kommunikasjonen kan en angriper lure en part til og tror at de kommuniserer med den riktige motparten.
	Denial of Service	Går ut på å nekte tilgangen til et system eller applikasjon for en berettiget bruker. En angriper kan oppnå dette ved for eksempel å sende flere forespørsler til serveren enn den kan håndtere. Et annet eksempel kan være å utnytte mekanismen for å autentisere en bruker til å få den aktuelle brukeren til å bli sperret ute.
	Spoofing	Går ut på å skjule sin riktige identitet. En angriper kan sende pakker med uriktig avsenderadresse for å utgi seg for å være en annen. På denne måten skjules hvor angrepet kommer fra.
	Sniffing	Også kalt tyvlytting, er en form for angrep hvor man overvåker nettverkstrafikken for data som kan kompromittere applikasjonene eller systemene. Man leter gjerne etter passord og konfigurasjoner i klartekst, eller pakker som er kryptert med svake algoritme.
Vert	Virus, Trojaner, Ormer	Programmer som på en aller annet måte har til hensikt å utføre skade eller ta over systemer og applikasjoner. Virus er designet til å utføre skade eller ondsinnet handlinger mot operativsystem og applikasjoner.

[9] Basert på kapittel IV i [**9**].

		Trojaner har samme hensikt som et virus, men utgir seg for å være en harmløs data fil eller kjørbart program. Ormer oppfører seg på lik linje som med trojanere, men de har i tillegg en replikasjonsegenskap som gjør de i stand til å infisere nye systemer etter et vellykket angrep.
	Profiling	En angriper prøver å finne hvilke systemer og applikasjoner som kjøres på verten i håp om at det muligens finnes sårbarheter i systemet eller applikasjonen som gir tilgang eller kontroll til angriperen. For dårlig sikkerhetsoppdatering av systemer og applikasjoner utgjør en stor trussel.
	Passordgjetting	Brukes når angriperen ikke kan etablere en anonymisert tilkobling til serveren. Angriperen prøver isteden å etablere en autentisert tilkobling ved å gjette seg fram til eventuelle brukernavn og passord kombinasjoner.
	Uautorisert tilgang	Med manglende eller for dårlig tilgangskontroll kan en angriper få tilgang til begrenset informasjon eller kjøre begrensede operasjoner.
Applikasjon	Input validering	Mangelfull sjekking av input parametre kan la en angriper utføre oppgaver han ikke er ment til å kunne gjøre. Eksempelvis kan angrep som SQL Injection og Buffer Overflow gjøre en angriper i stand til å se begrenset data eller i verste fall kjøre ondsinnet koden som lar angriper ta kontroll over verten.
	Svak eller egendefinert kryptering	Dårlig genererte nøkler, svak eller egendefinert kryptering kan føre til at en angriper utnytter dette og oppnår autentisert tilkobling.
	Sensitiv informasjon i klartekst	Ved ikke å kryptere sensitiv data kan en angriper få tak i disse dataene ved tyvlytting på nettverkskommunikasjonen, aksesser lagringsmediet eller angriperen kan endre på dataene.
	For dårlig revisjon og logging eller svak unntakshåndtering	Indirekte kan dette føre til at bruker kan nekte for å ha utført operasjoner, angrep blir for seint eller aldri oppdaget, eller uønsket informasjon blir utgitt når en angriper utfører gale operasjoner.
	Svake passord	Ved å la brukeren spesifisere svake passord kan angripere lett gjette seg til dette ved for eksempel Dictionary Attack eller Social Engineering.

Trusselmodell
Når man har en faktisk løsning å gå ut ifra kan det tas i bruk trusselmodeller som går mer spesifikk inn på de ulike sårbarhetene og kategoriserer dem på en helt ny måte. Microsofts STRIDE trusselmodell [**7**] er en slik modell og brukes for å analysere angrep under kategorier for hva det er ment å oppnå med angrepet. Man ønsker å identifisere de mest tenkelige angrepene og angrepsrettingene. STRIDE brukes for å identifisere og kategorisere disse angrepene og gir også retningslinjer for hvordan egnede mottiltak kan tas i bruk for å redusere risikoen. STRIDE står for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Evelation of privilege og viser til de trusselkategoriene trusselmodellen baserer sikkerhetsanalysen på.

4.2 Sikkerhetskrav

Før en analyse kan gjennomføres må det defineres sikkerhetskrav. De sikkerhetskravene som det er gått ut ifra, er som følger:

- *Subjektet er autentisert* – Der hvor det utføres tilgangskontroll må subjektet kunne vise til informasjon om de rettigheter som er gitt til subjektet. Denne informasjon brukes da for å sjekke om subjektet er gitt tilgang til de ressurser som det er forespurt om. Subjektet må ha autentisert seg først før informasjonen om hvilke rettigheter subjektet har er gyldig.
- *Kun autoriserte subjekter har tilgang* – Systemet skal kun gi tilgang til ressursene hvis subjektet kan bevise for å ha rettighetene til å ta i bruk dem. Hvis ikke subjektet har alle de rettighetene som det er påkrevd, nektes det tilgang og forespørselen blir forkastet.
- *Ikke mulig å opphøye tilgang* – Det skal ikke være mulig for subjektet å kunne opphøye eller utvide rettighetene sine. Med andre ord skal ikke subjektet kunne få tilgang til andre ressurser enn de som det allerede er gitt til av systemet. Den informasjonen som på bakgrunn av autentiseringen er kommet fram til, skal da ikke være mulig å endre på. Dette medfører blant annet sikre mekanismer for overføring av informasjonen mellomelementene som utfører autentisering og håndhever tilgangen.

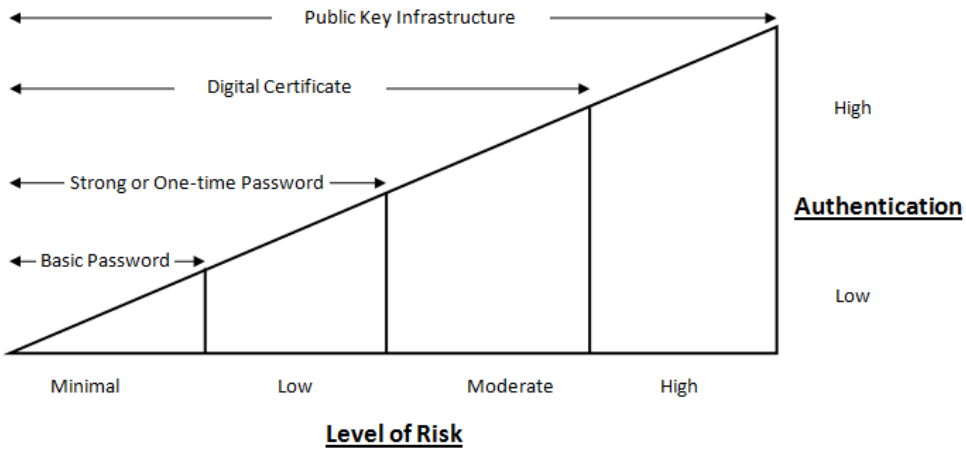
Disse sikkerhetskravene er kun gitt i henhold til oppgaven og er rettet mot den tjenesteorienterte delen av arkitekturen. Dette kan ikke sees på som krav gjeldende for hele arkitekturen.

4.3 Autentisering

Felles for hvert system er spørsmålet som blir stilt ved designet av systemet: hvor sikkert skal systemet være? Det å velge riktig autentiseringsmetoder for systemene er avgjørende for å trygge de ressursene man ønsker å beskytte og samtidig tilby gode løsninger. Valg av feil metode kan fort føre til svekket eller ingen sikkerhet. Eller, en dårlig brukeropplevelse på grunn av kronglete metoder for noe som skulle være enkelt.

Sikkerhetsnivåer

For å kunne ta det rette valget er man nødt til å forstå hva de ulike autentiseringsmetodene utfører og bidrar til. Man kan som regel kan gå ut fra for hver metode som tas i bruk øker sikkerheten, men som følge blir autentisering mer besværlig for brukeren. Det er ønsket sikkerhet, eller sikkerhetsnivå, som avgjør hvilke metoder som skal bli tatt i bruk.



Figur 4.2: Sikkerhetsnivå for ulike autentiseringsmekanismer. [10]

Figur 4.2 viser til et eksempel på et utvalg av autentiserings metoder og hvilke sikkerhetsnivåer disse metodene svare til. Ut ifra disse nivåene kan man identifisere tilhørende egenskaper til hver og en av metodene som viser til styrken deres. Med dette har man en forståelse av sikkerheten til hvert nivå og kan finne det nivået og den metoden som passer best for å sikre ressursene. I [**8**] tabell 4.1 identifiseres enkelte konsekvenser som kan være en pekepinne på hvilket sikkerhetsnivå man bør velge. Tabellen er gjengitt i tabell 4.2.

[10] Hentet fra [**8**], figur 4.2.

Tabell 4.2: Risikonivå og assosierende konsekvenser.

Risk level	Consequences of compromise
Minimal	Insignificant inconvenience to either party
	No release of private or sensitive information
	No threat to commercial or government interests
	No opportunity for associated criminal activity
Low	Possible inconvenience to either party
	No release of private or sensitive information
	Minor threat of financial loss to either party
	No threat to government interests
	No opportunity for associated criminal activity
Moderate	Significant inconvenience to either party
	Possible release of private or sensitive information
	Threat of significant financial loss to either party
	Threat to non-national security government interests
	Possible opportunity for associated criminal activity
High	Major inconvenience to either party
	Release of private or sensitive information
	Significant financial loss to either party
	Threat to government interests
	Threat to national security
	Opportunity for associated criminal activity

Kritiske momenter

De områdene som utgjør er trussel for sikkerheten, kan oppsummeres som følgende:

- Validering – Når identiteten opprettes kan det være at brukeren utgir seg for å være noe den ikke er, enten det er en annen identitet eller har andre rettigheter. Det er derfor viktig at valideringen svarer til sikkerhetsnivået av autentiseringen.
- For svake metoder – Bruken av metoder som ikke holder mål utgjør en trussel.
- Fornektelse – Ved å ta i bruk for svake metoder kan man nekte for å ha utført en handling. Eksempelvis kan man påstå å ha mistet eller blitt frastjålet brukernavnet og passord. Dette utgjør en trussel på lik linje med for svake metoder fordi man ikke har klart å se hele trusselbildet.

- Mistet berettigelsesbevis – Dette utgjør en trussel hvis man ene og alene setter seg avhengig av kun en autentiseringsmetode og denne metoden ikke svarer til det nødvendige sikkerhetsnivået.
- Håndtering – Ved ikke å håndtere de sensitive dataene og metodene som utgjør autentiseringsmekanismene på riktig måte, kan dette i seg selv utgjøre en trussel.
- Brukerfeil – Mekanismer kan være nødvendig for å fange opp, varsle og eventuelt rette opp feil som en bruker eller administrator gjør og som kan skade eller utsette systemet eller brukeren for sårbarheter. Eksempelvis svake eller opplagte passord.

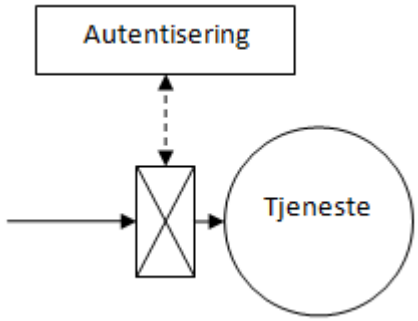
Faktumet er at autentisering, sett i forhold til den generelle arkitekturen, er løst. Det finnes et mangfold av metoder og mekanismer for autentisering som både er studert og utprøvd grundig. De eneste utfordringene som utgjør en risiko, er å ta i bruk feil metode sett ut ifra hvor sterk sikkerhetene skal være og gal implementering eller bruk av metoden.

4.4 Tilgangskontroll

For sikkerhetsanalysen er det gått ut ifra enkelte forutsetninger. Uten disse forutsetningene ville det ikke vært mulig å realisere en arkitektur uten kritiske sårbarheter. Disse forutsetningene er: (i) autentisering er gjennomført før tjenestekallet, (ii) tjenestekallets melding samt resultatet av autentisering er sikret både med tanke på integritet og konfidensialitet, og (iii) konfigurasjonen av tilgang til ressursene samt de rettigheter som er gitt til subjektene er riktig satt.

Med disse forutsetningene kan arkitekturen ha et SOA system som tar i bruk en felles autentiseringsmekanisme. Autentiseringen kan dermed bli utført av tilgangskontrollen, eller av et system eller en tjeneste på forhånd. Dette er illustrert i figur 4.3 hvor tilgangskontrollen til en tjeneste har muligheten til å autentisere subjektet. Hvorvidt de andre systemene tar i bruk denne fremgangsmåten er uvesentlig.

Det er gått ut ifra en sort boks for autentisering som kan ta i bruk en hvilken som helst anerkjent og sikker autentiseringsmekanisme. Resultatet av autentiseringen er enten bli lagt til i selve meldingen til tjenestekallet eller er blitt gitt direkte til den entiteten som skal ta autorisasjonsbeslutningen.



Figur 4.3: Autentisering og tilgangskontroll før tjenestekallet.

Det fine med denne løsningen er å kunne dele opp autentiseringen og tilgangskontrollen. Dette gir muligheten for å ta i bruk en eller flere autentiseringsmekanismer og samtidig gjøre håndhevingen av tilgang enkel. Ved å skille mellom autentisering og tilgangskontroll blir designet, implementeringen, forvaltningen, vedlikeholdet, og utviklingen lettere. Dette er også hensiktmessig i sammenheng med en tjenesteorientert arkitektur ettersom det ikke lenger er mulig å se på interaksjonen som enkel toveis klient-server interaksjon.

Sikker utveksling

Utvekslingen av sikkerhetsinformasjonen er sikret både med tanke på integritet og konfidensialitet, til den grad det er nødvendig. Fordi autentiseringen og tilgangskontrollen gjøres hver for seg, er det nødvendig å sikre at autentiseringsinformasjonen ikke endres før selve tilgangskontrollen utføres. I tillegg kan det være ønskelig i noen situasjoner å skjule sikkerhetsinformasjon fra andre enn selve tilgangskontrollen. Hvis angripere hadde klart å avlytte og få tak i denne informasjonen, kunne den bli misbrukt eller gi angriperne informasjonen om hvordan systemene ser ut.

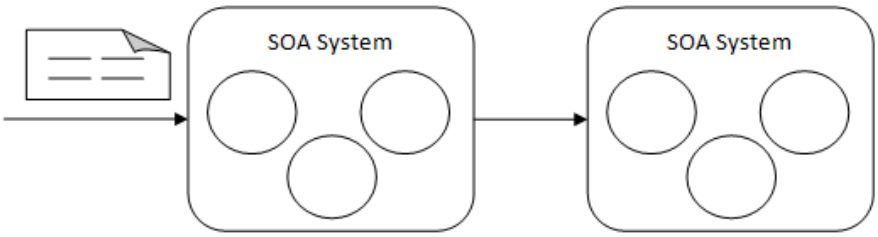
Det er ikke tatt stilling til hvordan disse mekanismene er tatt i bruk. Enten det er direkte med eventuelle kryptografiske funksjoner, eller indirekte ved å sikre systemene og kommunikasjonen mellom systemene. Det eneste som er viktig i oppgavens sammenheng er å ha en arkitektur hvor angripere utenfra ikke skal kunne manipulere eller få innblikk i sikkerhetsinformasjon som blir utvekslet mellom systemene og tjenestene.

Rett konfigurasjon
Til slutt er det gått ut ifra en riktig konfigurering av systemene der de sikkerhetsattributter som skal bestemme tilgangene til ressursene er riktig satt. De rettighetene som er tildelt subjektene må være riktige og ikke ha blitt gitt ved en feil eller på feil grunnlag. Hvis av en eller annen grunn en menneskelig feil fører til en autentisering og tilgangskontroll som ikke lenger fungerer som den skal, kan ikke selve arkitekturen og systemene klandres. Selvfølgelig kan man argumentere for at et system som gjør dette mulig, ikke er et sikkert system fra begynnelsen av, men dette er noe oppgaven ikke tar stilling til.

Attributter for beskrivelse av egenskaper, rettigheter og krav
I oppgaven brukes begrepet sikkerhetsattributt, eller bare attributt, for å vise til hvordan tilgangen til ressurser uttrykkes, samt hvordan rettigheter og egenskaper er tildelt subjektet. Det er ikke ment for å vise til en spesifikk mekanisme eller teknologi men er kun brukt for å beskrive helt generelt hvem, hva eller hvordan tilgangen til en ressurs er satt.

Attributter er et sett med merkede egenskaper som kan bli bruk til å beskrive en hvilken som helst entitet. En entitet tatt i betraktning i en autorisasjonsprosess gjelder ikke bare for subjektet, men også for andre elementer som er med i avgjørelsen. Dette gir en mer finkornet og kontekstrettet tilgangskontroll som kan tilpasses til dynamisk endrede behov. Tilgangen til ressurser kan bli beskrevet med et sett med attributter som må være oppfylt for å kunne ta i bruk eller få tak i ressursen. Subjektet må eksempelvis ha matchende attributter som de til ressursen som skal bli brukt.

4.4.1 SOA systemet
For å gjøre analysen så enkel og oversiktlig som mulig jobbes det trinnvis opp fra det minste funksjonelle SOA systemet til man har et fullverdig system lik den generelle arkitekturen. Derifra inkluderes annen vesentlig funksjonalitet spesifikt for SOA. I alle scenarioene er det gått ut ifra en implementering hvor det er SOA systemet som håndterer og ruter meldingene til SOA tjenestene. Dette er illustrert i figur 4.4. SOA systemet kan med dette utføre felles håndtering av meldinger inn til systemet og spesiell håndtering på forespørselen til tjenesten.

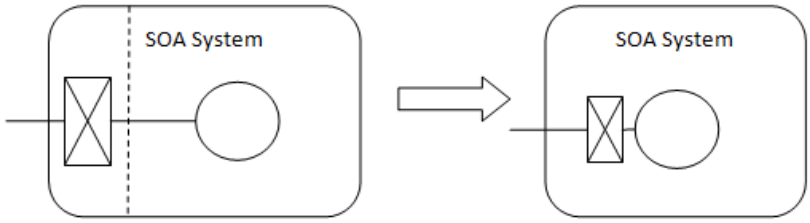


Figur 4.4: Håndteringen av meldinger mellom to SOA system.

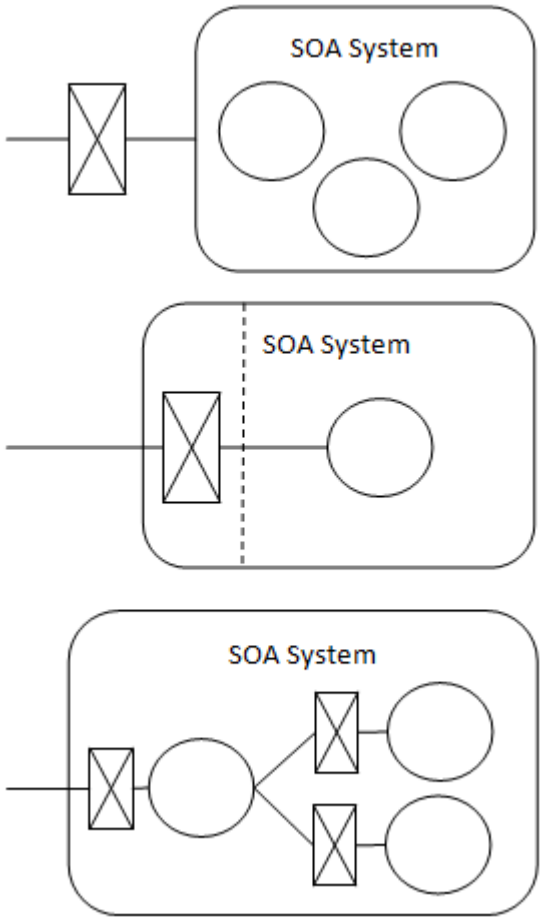
Med denne modellen er det mulig å utføre tilgangskontroll på tre forskjellige steder, figur 4.6:

1. Det kan gjøres utenfor systemet av eksempelvis et dedikert system
2. Det kan gjennomføres av selve SOA systemet
3. Hver enkel tjeneste kan utføre tilgangskontrollen selv

Ut ifra den SOA funksjonaliteten som er tatt i bruk er det mange fordeler og ulemper med hver av de tre måtene. Ettersom fokuset nå er på SOA systemet alene, kommer ikke den eksterne tilgangskontrollen til å bli nevnt. Av praktiske grunner er det også valgt å se på tilgangskontroll av systemet og av tjenestene som det samme. Figur 4.5 illustrerer dette. Selv om dette ikke er riktig ettersom systemet eller tjenestene vil bli mer komplisert alt ettersom valget. Men, for enkelthetens skyld vil det å utføre tilgangskontrollen i den aller første tjenesten svare enten til tilgangskontroll i systemet, eller av tjenestene.



Figur 4.5: Tilgangskontroll utført av systemet eller tjenesten.



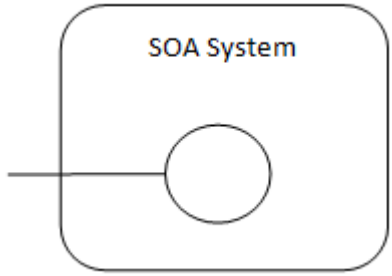
Figur 4.6: Muligheter for tilgangskontroll i SOA systemet.

4.4.1.1 Enkle tjenester

Figur 4.7 viser det enkleste og mest minimalistiske SOA systemet som kan lages. Systemet består av kun en enkel tjeneste. Tjenesten utfører en operasjon, eksempelvis multiplikasjon av to store tall. Den tar ikke i bruk andre tjenester, enten det er innenfra eller utenfra SOA systemet.

Scenario 1:

Tilgangskontrollen i dette scenarioet kan utføres både av tjenesten selv eller av systemet uten signifikante forskjeller. Tilgangskontrollen vil sjekke om subjektet er autentisert og om den har tilgang til de ressursene som vil bli brukt i tjenestekallet.

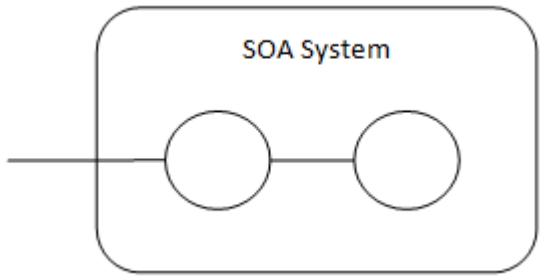


Figur 4.7: SOA system med en enkel tjeneste.

Styrken med dette scenarioet er hvor lite komplisert det vil være for tilgangskontrollen å få tak i de nødvendige attributtene. Både for attributter som beskriver restriksjonene til tjenestene, og for egenskapene og rettighetene til subjektet. Det eneste som kan feile i dette scenarioet her er menneskelig svikt ved feil konfigurasjon av attributtene.

4.4.1.2 Sammensatte tjenester

SOA systemet består nå av to eller flere tjenester som er satt sammen til en stor tjeneste. Det er den sammensatte tjenesten som er synelig utenfor SOA systemet. Dette kan være et eksempel på gjenbruk av tjenester der den innerste tjenesten kanskje blir brukt i andre applikasjoner også. I de påfølgende scenarioene inneholder systemet to tjenester. Den ene tjenesten tar i bruk den andre og begge tjenestene er satt til å begrense tilgangen, men har forskjellige attributter som beskriver tilgangskravene. Se figur 4.8.



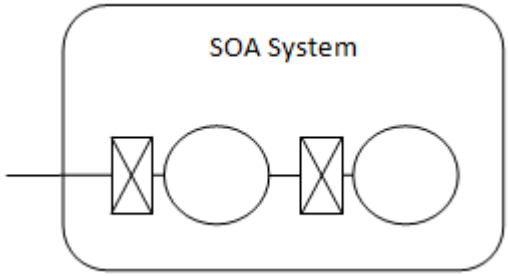
Figur 4.8: SOA system med sammensatte tjenester.

Opp mot den generelle arkitekturen kan dette scenarioet for eksempel omhandle tjenestene for å finne EPCIS hendelser til et produkt og for å gjennomføre spørringer mot en database. Den første tjenesten vil da utføre spørringer om hendelsene til et produkt mens den andre tjenesten omslutter en database med EPCIS grensesnittene og dermed lar den første tjeneste gjennomføre EPCIS spørringer mot databasen.

Med tanke på tilgangskontroll har den første tjenesten ansvaret for å sjekke om subjektet har tilgang til databasen mens den andre tjenesten vil sjekke om subjektet har tilgang til de hendelsene som det spørres etter. Ved å ta i bruk denne modellen kan både database og dets innhold beskyttes uten komplisert tilgangskontroll i selve database.

Med dette har systemet en sammensatt tjeneste med muligens ulike sikkerhetskrav

Scenario 2:
Går ut ifra en modell, figur 4.9, hvor det utføres tilgangskontroll for hver tjeneste. Det tas dermed ikke hensyn til de kontrollene som er eller vil bli utført før eller senere i kjeden til tjenestekallene.



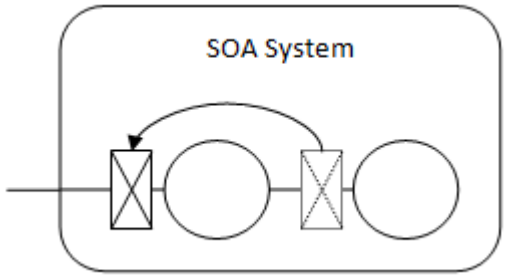
Figur 4.9: Tilgangskontroll for hver tjeneste som er sammensatt.

Problemet som oppstår nå er muligheten for inkonsistent data for de ulike tjenestene. Dette kommer av at det nå er mulig for hvilke som helst av tjenestene som er satt sammen til å forkaste tjenestekallet. Hvis tilgangs til en av tjenestene blir avslått før oppgaven er ferdig kan det oppstå inkonsistens i dataene. Med mindre det finnes en form for tilbakekalling av tjenestekall hos de tjenestene som allerede er kallet, kan man ikke lenger være sikker på om systemet og applikasjonene fungerer som det skal. Dette problemet vil med andre ord føre til

mer komplekse tjenester fordi det trengs mekanismer for å tilbakestille et tjenestekall.

Det største problemet er derimot hvor usikre tjenestene blir av å ta i bruk denne måten for tilgangskontroll. Det kan være mulig for den sammensatte tjenesten å motta et nytt tjenestekall før den forrige oppgaven er avsluttet. Når man så må tilbakestille en melding kan dette ikke la seg gjøre fordi det nye tjenestekallet allerede har gjort endringer som ikke kan reverseres. Eller, mer kritisk er det å ha tjenester som utfører viktige operasjoner før det er vist om oppgaven kan i det hele tatt utføres.

Scenario 3:
I dette scenarioet går det ut ifra en litt annerledes løsning enn i scenario 2. Nå akkumuleres istedenfor attributtene fra hver tjeneste som har tilgangskontroll og kontrollen gjennomføres kun i den sammensatte (første) tjenesten. Fra eksempelet gitt tidligere vil man nå samle opp attributtene fra de to tjenestene og enten utføre tilgangskontrollen ved inngangen til SOA systemet eller ved den første tjenesten. Dette er illustrert i figur 4.10.



Figur 4.10: Akkumulering av attributter for tilgangskontroll.

Styrken med denne løsningen er en effektiv tilgangskontroll for hele den sammensatte tjenesten der det aldri vil oppstå situasjoner hvor bare deler av oppgaven tjenesten er satt til å gjøre blir utført. I tillegg blir det enklere å se hvilke attributter som er gjeldende for hver enkelt sammensatt tjeneste. Ettersom de ulike kontrollene er satt sammen til en, blir også systemet raskere samt de tjenestene som er satt sammen blir enklere siden de ikke lenger må implementere sin egen tilgangskontroll.

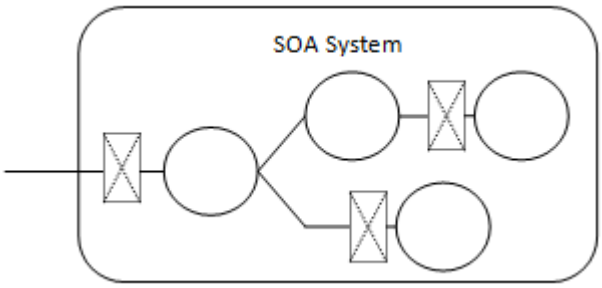
Svakhetene med denne løsning er derimot todelt og har svakheter og fordeler som er hverandres motpart.

Statisk
Den første løsningen er en statisk variant hvor man samler opp attributtene første gangen den sammensatte tjenesten tas i bruk for så å beholde dem for resten av tiden den er i bruk. Mer normalt kan det være at tilgangen til de sammensatte tjenestene konfigureres første gangen og må oppdateres når tjenestene de er satt sammen av får endret kravene sine.

Svakheten med denne løsning er mulig sikkerhetssvikt når tjenester oppdateres eller får endret attributtene. Hvordan skal man vite hvilke sammensatte tjenester som må oppdateres når attributtene til en tjeneste endres er noen av utfordringene man kan møte på. Denne svakheten er veldig farlige og kan fort føre til katastrofale utfall hvis ikke det følges nøye med på hva som til en hver tid blir gjort med SOA system.

Dynamisk
Den andre løsningen er en dynamisk variant hvor man henter inn attributtene fra alle tjenestene som utgjør den sammensatte tjenesten ved hvert eneste tjenestekall. På denne måten er man sikret for alltid å ha de gjeldende attributtene for hver enkel tjeneste. Denne løsningen er i tillegg mer attraktiv da den, i motsetning til den statiske løsningen, er i tro med prinsippene bak SOA.

Scenario 4:
Sett en sammensatt tjeneste hvor det ikke er vist på forhånd hvilke av de sammensatte tjenestene vil bli brukt. Eksempelvis en løsning som tar i bruk en tjeneste for enkle oppgaver og en ny sammensatt tjeneste for mer avanserte oppgaver. De to tjenestene vil mest sannsynlig være konfigurert med ulike attributter og man vet ikke hvilke av disse attributtene som gjelder før den første tjenesten har behandlet forespørselen. Se figur 4.11 for en illustrasjon. Dette er en mer avansert variant av scenario 2 og 3 og som byr på nye problemer.



Figur 4.11: Tilgangskontroll for mer komplekse sammensatte tjenester.

Problemet som oppstår nå er hvilke løsninger man skal ta i bruk for å fikse dette problemet:

- Man kan ta i bruk en funksjon som gjør det mulig for den første tjenesten å beregne hvilke attributter som er gjeldende før den behandler selve meldingen. Dette vil løse problemet, men vil kreve mer kompliserte og avanserte tjenester.
- Man kan akkumulere alle attributtene og gå ut ifra at alle gjelder. Dette blir ikke riktig da den av de to tjenestene som har lavest kun vil bli kalt hvis subjektet har tilgang til begge tjenestene. Med andre ord vil subjekter som kun har rettigheter til den tjenesten med laveste sikkerhet aldri få tilgang til noen av tjenestene.
- Man kan gå ut ifra en løsning lik scenario 2 og dermed risikere å få inkonsistent data hvis dette er mulig. Dette kunne faktisk vært en fin løsning hvis det ikke var mulig å få inkonsistent data.

4.4.1.3 Orkestrerte tjenester

Ettersom orkestrering kun bestemmer rekkefølge på hvilke tjenester som kan bli kalt, må hver enkelt tjeneste gå ut ifra at det ikke er blitt gjennomført noen kontroll på forhånd. Med andre ord vil man ikke oppleve den samme situasjonen som for eksempel i scenario 2 hvor man kan få inkonsistent data. Dette fordi tjenesten er laget uavhengig av hverandre og er laget slik at de ikke avhengig av hva de andre orkestrerte tjenestene gjør. De er kun fokusert på sin egen oppgave.

Men, det finnes situasjoner som rettferdiggjør hvorfor dette scenarioet er tatt med. Det kan for eksempel være ønskelig å sjekke om subjektet kan nå, og ta i bruk, en av de endelige tjenestene i orkestreringen. Det er med andre ord ikke ønskelig å la subjektet kalle tjenestene hvis den ikke kan oppnå noe – gjennomføre en endelig oppgave.

Fra den generelle arkitekturen kan det for eksempel være ønskelig å nekte tilgang til tjenesten for sporing av varer hvis brukeren ikke kan hente ut vareinformasjon etterpå.

Et annet eksempel er orkestrering av to tjenester hvor den ene henter ut kryptert konfidensielt materiale og den andre tjenesten dekrypterer materialet. Det er ikke ønskelig å la subjekter kalle den første tjenesten hvis den ikke har rettighetene til og dekryptere innholdet etterpå.

Scenario 5:
Ut ifra orkestreringen vet man hvilke tjenester som kanskje kan bli kalt og det er dermed mulig å avslå eller i beste fall advare brukeren om mangel på rettigheter for å utføre en fullverdig oppgave. Det eneste som gjør denne mekanismen vankelig er hvis det er tatt i bruk kompliserte orkestreringer hvor det finnes forskjellige tjenester som kan betegnes som endelige. Det blir i disse tilfellene vanskelig å kunne finne de rette attributtene som gjelder og man kan ende opp med å utføre tilgangskontroll for kun deler av orkestreringen.

4.4.1.4 Tjenestemeglere (dynamiske tjenester)
I det neste scenarioet er SOA funksjonen for dynamisk oppdaging av tjenester tatt med. Nå er ikke tjenestene i den sammensatte tjenesten bundet fast sammen. Isteden må tjenestene oppdages før de kan bli brukt.

For eksempel kan dette scenarioet være gjeldende for en bedrift som spesialiserer seg på å levere tjenester som utfører samme oppgave, men hvor hver tjeneste har ulike egenskaper. Ved å spesifisere blant annet pris og ytelse kan kunder velge hvorvidt oppgaven skal utføres fort og dyrt, eller sakte og billig.

Scenario 6:
Dette scenarioet opplever man de samme problemene som man møter på i scenario 4. Det er ikke lenger mulig å vite hvilke tjenester som vil bli kalt for på dem måten finne alle de nødvendige attributtene som trengs for å kunne fullføre oppgaven.

Den eneste forskjellen i dette tilfellet er at det ikke lenger mulig å ta i bruk en løsning hvor man finner hvilke tjenester som kommer til å bli kalt før selve kallet gjennomføres. Bakgrunnen for dette er at man er nødt til å gjennomføre spørringene på tjenestemegleren, men de tjenestene som man finner vil ikke lenger være gyldige. Dette fordi hver enkelt tjeneste vil på ny gjennomføre en spørring for å finne neste tjeneste. Man kunne selvfølge gjort det mulig å spesifisere resultatet av spørringene ved tjenestekallet for på den måten unngå en ny spørring, men dette vil da føre til en sikkerhetsrisiko da tjenesten ikke lenger selv står for oppdagingen av den neste tjenesten som vil bli kalt.

Den eneste fordelen med denne løsningen for dynamisk sammensatte tjenester, er hvis det faktisk er mulig å ta i bruk en løsning som gjør det mulig å vite hvilke tjenester eller hvilken type tjenester som vil bli kalt. Da er det mulig å gjennomføre tilgangskontroll på den første tjenesten ved bare å vite hvilke

sikkerhetsattributter den dynamiske tjenesten vil ha. Det mer reelle for dynamiske tjenester er nok at de ikke vil la sammensatte tjenester utføre tilgangskontrollen på veiene av dem.

4.4.1.5 Konversasjon

På lik linje med orkestrering finnes det tjenester hvor man må utføre ulike operasjoner på før man har det endelige resultatet. Eksempelvis må kan man lage en tjeneste hvor man må oppgi parametre i tre forskjellig tjenestekall før man har utført en oppgave.

Scenario 7:
Hvis de ulike kallene har ulike attributter dukker det opp et problem hvis det er ønskelig å nekte tilgang til subjekter som ikke kan fullføre en oppgave: hvordan man skal finne hvilke attributter som gjelder? Enda vanskeligere blir det hvis tjenesten har flere kall som gir et endelig resultat og man ikke lenger vet eksakt hvilke av kallene som vil bli brukt.

4.4.1.6 Tilstand

For tjenester med tilstand blir tilgangskontrollen mer kompleks. Avgjørelsen for om tilgangen skal godkjennes må I tillegg være basert på tilstanden til tjenesten. På en eller annen måte må den som håndhever tilgangen få den nødvendige informasjon i form av attributter om tilstanden til tjenesten i relasjon til tjenestekallet. Denne informasjonen trenger ikke være direkte relatert til selve tjenesten. Håndhevingen av tilgang til en tjeneste kan også være avhengig av miljø til tjenesten eller statusen til en ressurs, eksempelvis.

Scenario 8:
Et eksempel på dette kan være en tjeneste med to tilstander for å utføre en signering av en avtale mellom to parter. Det første kallet kan alle utføre og vil bare utlevere en avtale som kan signeres. Det andre kallet krever derimot at begge partene er medlemmer i organisasjonen, og vil signere avtalen basert på signaturer til de to partene spesifisert i tjenestekallet. Med dette er tilgangskontrollen avhengig av å vite om det gjeldende tjenestekallet bare skal hente ut avtalen eller om det er signeringen av avtalen som skal fullføres. Tilgangskontrollen vil dermed være implementert på en slik måte at den kan kontakte tjenesten og få de nødvendige sikkerhetsattributtene for å kunne ta avgjørelsen på om tjenestekallet skal godkjennes eller ikke.

4.4.1.7 Sentralisert

For en sentralisert tilgangskontroll oppstår det også nye utfordringer. Med sentralisert menes en arkitektur hvor flere systemer som ikke nødvendigvis er heterogene tar i bruk samme mekanisme for tilgangskontroll. Problemet med dette oppstår når systemene ikke har samme definisjon på de sikkerhetsattributtene som er gitt.

Scenario 9:
Skal dette fungere er man ente avhengig av å kunne legge føringer på alle systemene som tar i bruk tilgangskontrollen, eller så må man kunne oversette betydningen av attributtene mellom systemene.

For eksempel kan to løsninger geografisk brukt i to forskjellige land ha ulik oppfatting av når en person er myndig. Hvis disse applikasjonene skal ta i bruk en tredjepart for å utføre en tjeneste som kun skal være er tilgjengelig for myndige personer, kan det oppstå konflikter. Ettersom ulike land har ulik oppfattning av når en person er myndig, er man nødt til å deklarere en felles definisjon for når en person er myndig og eventuelt omgjøre de attributtene til hver av applikasjonene før håndhevingen av tjenestekallet.

4.4.2 Andre systemer

Generelt for de andre systemene utføres tilgangskontroll på mange ulike måter. Ulike mekanismer og modeller for tilgangskontroll finnes, er godt utprøvd og undersøkt, men ikke alle passer til hvert system. De fleste systemer har derfor alle ulike måter å løse problemet på. Denne delen vil kun vise til sårbarheter og trusler for tilgangskontroll i arkitekturen sett i en helhet.

4.4.2.1 Kriterier

Aller først kan det nevnes et par selvforklarende kriterier som må være oppfylt for sikker tilgangskontroll:

- *Sanntid*: når en bruker ønsker å få tilgang til eksempelvis en applikasjon må tilgangkontrollsystemet kunne finne de gyldige rettighetene tilhørende brukeren. Hvis denne informasjon er gått ut på dato eller er blitt endret siden sist gang brukeren tokk i bruk applikasjonen, skal ikke brukeren lenger få tilgang til dette hvis den nye informasjonen tilsier det. Når en utfører tilgangskontroll er man derfor avhengig å ha den seneste informasjonen for å kunne foreta et riktig valg.
- *Tiltrodd*: de handlingene som utføres, enten det er validering av subjektets identitet, autentisering, eller autorisasjon, må alle være utført

av en tiltrodd part. Hvis ikke kan hvem som helst utgi seg for å ha berettigelsesbevis de egentlig ikke har. Med andre ord kan man la et annet system eller en tredjepart utføre disse oppgavene så lenge man kan stole på at jobben blir utført riktig og med de kontroller som svarer til det sikkerhetsnivået man har.

- *Sikkert*: den informasjonen som brukes for å autentisere og autorisere subjekter skal ikke kunne endres på med mindre det blir gjort av noen som er autoriserte til å gjøre det. Et subjekt skal for eksempel ikke kunne legge til ekstra berettigelsesbevis etter en autentisering, men før selve tilgangskontrollen. Gjerne er det også ønskelig å holde disse dataene i kryptert form for ikke å røpe sensitiv data som for eksempel brukernavn og passord.

Spørsmål som dukker opp på bakgrunn av disse kriteriene er:

- Hvem skal håndtere validering, autentisering, autorisasjon?
- Hvilke mekanismer skal tas i bruk for sikring av sensitiv informasjon, både med tanke på integritet, så vel som konfidensialitet?
- Hva slags rettigheter skal man bruke, hva må lagres, og hvordan er formatet?
- Skal hvert system stå for sin egen tilgangskontroll eller skal det brukes dedikerte tjenester?
- Hvis tilgangskontroll er samordnet mellom ulike tjenester, hvordan vet man hvilken sikkerhetsinformasjon som er nødvendig for å kunne utføre tilgangskontroll uten å røpe for mye?
- Er det nødvendig med støtte for komplekse vurderinger?

I tillegg kan mekanismer som Single Sign-On (SSO), føderasjon og lignende introdusere nye utfordringer.

4.4.2.2 Felles identitetsregister

Normalt utføres tilgangskontrollen av hvert system. Problemet er å måtte ha flere systemer som tar i bruke flere mekanismer for tilgangskontroll og samtidig har sine egne identitetslagre.

De største truslene ved å ha mange ulike identitetslagrer er å ende opp med dupliserte identiteter. I tillegg er det lett å miste oversikten, og løsningene kan fort bli både komplekst og tidkrevende å håndtere. Da er det lett å gjøre feil og det er bare et spørsmål om tid før konsekvensene viser seg. Eksempler gitt i [**8**]

viser til mange slike situasjoner hvor blant annet dupliserte identitetsinformasjon utgjør en skjult trussel.

Dette er et spørsmål som har en betydelig signifikans på arkitekturen og ikke minst på sikkerheten. Kan hvert system stå for sin egen sikkerhet eller er det kanskje best å ta i bruk en felles løsning. Eksempelvis, brannmuren og web serveren har sine egne konfigurasjoner eller står et annet system for sikkerhetsbeslutningene. Eller, kanskje er det ideelt med en mellomting hvor et system står for håndteringen av sikkerheten, men hvert system står selv for å håndheve den.

En ting er i hvert fall sikkert. Ved å gjøre det minst mulig komplisert kan man redusere muligheten for brukerfeil og menneskelig svikt. Mennesket har tross alt det lett for å glemme og å gjøre feil når ting blir for komplisert eller når man ikke helt har oversikten over situasjonen. I [**8**] konkluderes det i hovedsak med en felles løsning som den rette veien og gå.

Men, for mange av dagens systemer er i mange situasjoner løsningen en del av systemet og bygget for å operere med kun et lokalt datalager. Dette er systemer som det ville være dumdristig å prøve å modifisere til å fungere med andre systemer og datalager. I slike sammenhenger er det kun synkronisering som er den foretrukne måten hvis man absolutt skal ta i bruk et sentralt identitetslager.

4.4.2.3 Overføring av sikkerhetsinformasjon

Med en gang man tar i bruk sikkerhetsmekanismer utenfor systemet vil informasjon som for eksempel identitets- og sikkerhetsattributter være utsatt for angrep. Dette gjelder også for sikkerhetsmekanismer for klient-server paradigmet hvor det utveksles av sikkerhetsinformasjon for autentisering av brukere, eksempelvis.

Som en løsning på dette tas det gjerne i bruk krypterte kommunikasjonskanaler, men dette igjen fører til nye problemer for felles eller sentrale sikkerhetsmekanismer. Selv om vanlige mekanismer som Secure Socket Layer (SSL) og Transport Layer Security (TLS) er ganske sikre for end-til-ende kommunikasjon kan de ikke ene og alene tilby den ende-til-ende sikkerheten som trengs i tjenesteorienterte arkitekturer.

Faktumet er at en melding kan bli rutet og prosessert av flere mellomliggende applikasjoner og tjenester før den når sitt mål. Fordi SSL/TLS kun tilbyr sikkerhet mellom to entiteter som kommuniserer, er det mulig for de

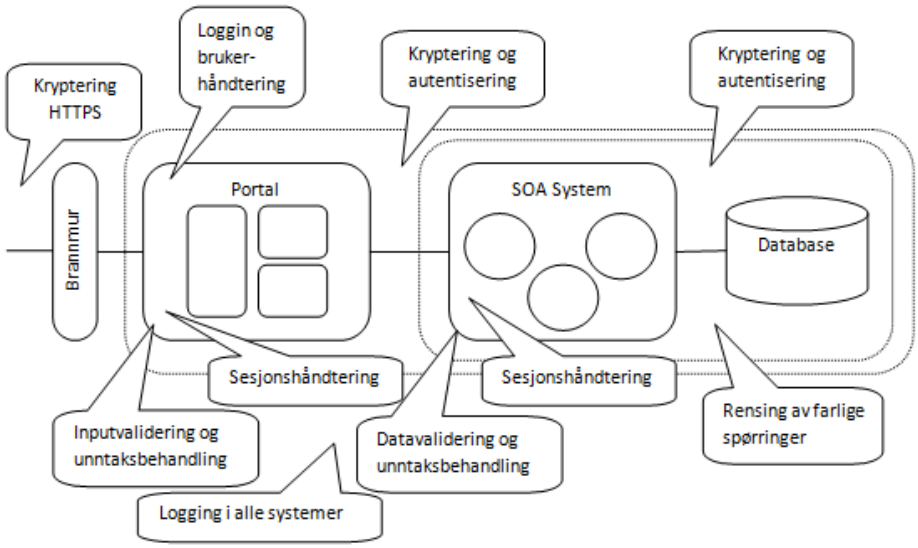
mellomliggende applikasjonene eller tjenestene og uoppmerksomt granske og endre på meldingen med onde hensikter. Hvordan man skal da få den nødvendige informasjonen fram uten å gå på bekostning av andre ting, samtidig som sikkerheten opprettholdes, er viktige og veldig essensielle utfordringer som man kan møte på i tjenesteorienterte arkitekturer.

5 Referansearkitektur

Basert på sikkerhetsanalysen er det tatt i bruk to hovedelementer for sikringen av tjenestene. Det første er et identitets- og tilgangskontrollsystem. Dette systemet står overordnet for identitetene som er brukt i nettverket og håndteringen av tilgang til de ulike systemene og tjenestene. Det andre er en tilgangskontrollmodell for det tjenesteorienterte systemet. Modellen viser til hvordan håndhevingen av tilgang til tjenestene utføres.

De to hovedelementene er supplert med mer normale og tradisjonelle sikkerhetstiltak. For sikkerheten innad i arkitekturen brukes mekanismer for sikring av kommunikasjonen mellom systemer, og sikring av selve systemene og tjenestene. Referansearkitekturen er basert på den metodologien og de stegene som er gitt i [**9**][11] og tar for seg en helhetlig sikring av arkitekturen.

Med referansearkitekturen, illustrert i figur 5.1, ønskes det å sikre applikasjonene og ressursene på en sikker på. Uten å gå i for mye detalj gjøres dette ved å ta i bruk komponenter og mekanismer som forklart i tabell 5.1:



Figur 5.1: Referansearkitekturen, med sikkerhetsmekanismer.

[11] Selv om det er gått ut ifra produkter fra Microsoft vil mekanismene og metodene til en viss grad være uavhengig av hvem som har levert systemet.

Tabell 5.1: Mekanismer for sikring av nettverk og applikasjoner.

Brannmur, rutere svitsjer	Selve nettverket sikres med disse komponentene og er den første forsvarslinjen som en angriper utenfra møter på. Disse komponentene vil sørge for tilgangskontroll til de forskjellige systemene i nettverket. De kan også brukes for å opprettholde forskjellige sikkerhetssoner innad i nettverket. Med dette blir nettverksenhetene og data i transmisjon sikret.
Portalen	Som det system som er verten for applikasjonene, har den en fundamental rolle for beskyttelsen av applikasjonene. Webserveren sikres ved eksempelvis å ta i bruk riktige konfigurasjonsinnstillinger, men også ved å tvinge bruk av siker kommunikasjon med HTTP Secure (HTTPS). Autentisering og tilgangskontroll utføres også av systemet for kun å tilby de tjenestene en bruker har rettigheter til.
SOA systemet	Avgjørende for sikkerheten til systemet er riktig konfigurasjonsinnstillinger og lagvise sikkerhetsmekanismer som sikring av kommunikasjonskanalen (mot systemet), autentisering (mot brukeren), og tilgangskontroll (mot tjenestene).
Database	Riktig konfigurasjonsinnstillinger utgjør en viktig del for å sikre systemet. Samtidig kan andre mekanismer for sikring av systemet bli brukt. For eksempel ved å rense spørringer for farlige uttrykk, eller adskilles av tilgangskode fra databasen. Sikkerhetsmekanismer mot resten av nettverket kan også tas i bruk.
Identitetshåndtering	System for felles og sikker håndtering av identiteter. Utvides til et system som også håndterer tilgangskontrollen til de ulike systemene og tjenestene.
Retningslinjer for sikker kode og konfigurering	Hvordan man skal for eksempel sikre tjenestene med inputvalidering eller sikre konfigurasjonsinnstillinger for applikasjonen slik at ikke tjenestene i seg selv ikke utgjør en trussel.

Kort oppsummert blir nettverket og sikkerhetssonene sikret med brannmurer, rutere, svitsjer. Systemene og tjeneste er riktig konfigurert, har de nyeste sikkerhetsoppdateringen, og tar i bruk kjente sikkerhetsmekanismer. Kommunikasjonen sikres med krypterte kanaler mellom brukeren og nettverket. Deler eller hele meldinger sikres med kryptering og signaturer innad i nettverket og eventuelt også mellom brukeren og tjenestene. Det brukes et felles identitetsregister for å samkjøre de ulike systemene med tanke på autentisering og håndtering av identiteter og rettigheter. Policy brukes for å definere og håndheve restriksjonene på ressursene og hva ulike identiteter og rettigheter har lov til å gjøre.

Policy
[**10**] viser til en definisjon av policy, eller rettere sagt en sikkerhetspolicy, som:

> *"... statement of what is, and, what is not allowed."*

Ut ifra definisjonen gir sikkerhetspolicyer en formell og generell måte å fremstille hva som er lov og hva som er ikke lov. Med formell menes for eksempel at den kan leses og forstås av maskiner. Den er generell ut ifra at det ikke er sagt hva som må utgjøre en policy.

Det negative med denne måten å beskrive restriksjonene for tilgang på er at dette ikke setter restriksjoner på hvordan en policy blir formulert. Dette kan føre til tvetydige formuleringer av policyer som kan utgjøre en sikkerhetstrussel.

Uansett, policy tilbyr en anvendbarhet og brukbarhet som gjør den i stand til å håndtere og integrere et flertall av ulike modeller for identitet og tilgangskontroll. Role-Based Access Control (RBAC) er en slik sikkerhetsmodell.

Neste følger en mer detaljert beskrivelse av de to hovedelementene til referansearkitekturen.

5.1 Identitetshåndtering

Det er viktig å kunne avgjøre om en bruker har rettighetene til å ta i bruk en tjeneste. Denne avgjørelsen tas ved å sjekke rettighetene til brukerens identitet opp mot de kravene som er gitt av tjenesten og de ressurser som vil bli berørt. Identiteten til en bruker er bestående av data kalt identitetsattributter og viser til de sikkerhetsrelevante egenskaper brukeren er tildelt. Håndtering av identiteter, eller identitetshåndtering, er organisering og lagring av data spesifikk for hver

enkelt bruker. En bruker kan for eksempel være en ansatt innenfor organisasjonen eller klienter som kunder og leverandører.

Et system for identitetshåndtering er et ledd i å håndheve tilgangen til en organisasjons ressurser. Et miljø for identitetshåndtering vil typisk kunne tilby funksjonalitet som proviantering, rollehåndtering, overholdelseshåndtering, og arbeidsflyt. I tillegg blir det lettere å kunne overvåke og logge brukeres identiteter, rettigheter, og handlinger. Dette fører til bedre sikkerhet, samt kan bli brukt for lettere å overholde lover, regler og initiativ som påkrever denne funksjonaliteten. Sarbanes–Oxley Act i USA [**11**] er et eksempel på dette.

Ofte har mer tradisjonelle systemer sitt eget identitetsregister og autentiseringsmekanismer. Fra sikkerhetsanalysen er det å håndtere flere identitetsregistre en mulig sårbarhet. Et system for identitetshåndtering, også kalt et Identity Management (IM) system, har som mål å forene alle disse registrene gjennom en felles løsning. Et identitetslager utgjør grunnlaget for et IM system og det generelt sett finnes to typer identitetslagre. Identitetslagrene er typisk bestående av enten et register eller en database.

Med dedikert system for identitetshåndtering åpner det opp muligheter for enklere og mer effektiv autentisering. Eksempler på dette er for eksempel multidomene-identitetshåndtering for bedre sikkerhet og brukeropplevelse, lik føderasjon.

Sikkerhetsmodell

For tilgangskontroll er det veldig viktig med en effektiv og oversiktlig sikkerhetsmodell. I [**10**] begrunnes valget av riktig sikkerhetsmodell som:

> *"En effektiv administrasjon av autorisasjonspolicyer i et distribuert miljø er det vesentlige kriteriet for valget av egnede sikkerhetsmodeller."*

I [**10**] presenteres fire ulike sikkerhetsmodeller for håndtering av komplekse autorisasjonsbeslutninger. Disse modellene er Discretionary Access Control, Mandatory Access Control, RBAC, og $UCON_{ABC}$. Fra oppgavens ståsted er det RBAC som er mest interessant. Med RBAC håndteres ikke brukere individuelt, men man definerer roller som kan bli gitt til hver enkelt bruker. Det er dermed rollene som viser til rettighetene man har. Dette gjør det mye lettere å håndtere et mindre antall roller enn et mangfold av brukere. Håndhevingen defineres av RBAC gjennom bruk av sikkerhetspolicyer. Ved å ta i bruk policy blir det lettere og raskere å endre tilgangen til eksempelvis grupper som er blitt tildelt roller.

Sikkerhetsmodellen RBAC muliggjør også for Separation of Duties (SOD). Med SOD er det mulig å tildele roller til hver enkelt organisasjon i løsningen og la dem håndtere sin egen tilgang. Med et rollehierarki vil det eksempelvis være mulig å tildele roller til organisasjon A for håndtering av tilgang til egne ressurser. Organisasjon A kan med dette definere nye roller som gir begrenset innsyn til spesifikke ressurser og tildele disse rollene til andre organisasjoner etter ønske.

Men RBAC har også enkelte svakheter, spesielt ved bruk i en tjenesteorientert arkitektur. Som forklart i [**10**] kan avgjørelsen for tilgangen til en tjeneste også være avhengig av tilstaden til tjenestene eller miljøet til tjenesten. Man kan også være avhengig av å måtte definere dynamiske restriksjoner for når en rolle har rett til å ta i bruken en ressurs. I tillegg kan det være nødvendig med å definere mer spesialisert tilgang. For eksempel kan det være ønskelig å spesifisere hva man kan gjøre med ressursen.

Identity and Access Management

Et Identity and Access Management (IAM) system er en utvidelse av et IM system og kan bli brukt til å håndtere tilgangen til en arkitektur på ulike nivåer. Med IAM kan tilgangskontrollkravene være gitt til elementer på det fysiske laget, til applikasjonslaget, eller til forretningslaget. For referansearkitekturen kunne et IAM systemet ha lagt føring for tilgangskontroll for både brannmur, portalen, og databasen. I tillegg til selve SOA systemet. Et IAM system typisk innholder:

- Et autentiseringslager for informasjon som lar brukere logge inn på nettverket
- Et autorisasjonslager som inneholder en brukers attributter for tilgang til ressursene i nettverket og er mer dynamisk og flyktig

Målet med et IAM system er å forenkle tilgangen til ressurser samtidig som tilgangen blir nektet for de som ikke har rettighetene. Dette muliggjøres ved bruk av policy for økt transparens og skalerbarhet. Andre grunner for å ta i bruk et IAM system, er:

- Det er ønskelig å beskytte brukerens identitetsattributter mot tjenester som kan misbruke denne dataen.
- For løsninger over ulike domener og sikkerhetssoner er det nødvendig for brukeren å måtte autentiseres flere ganger. Ved å ta i bruk en felles autentiseringsmekanisme slipper man dette.

- Ta i bruk felles definisjoner av identitets- og autorisasjonspolicyer til ulike tjenester for å unngå heterogen navngiving av attributtene. Viktig for alle parter, tjenestene, brukerne og tilgangskontroll mekanismene å ha den samme forståelse for hva attributtene viser til.

5.2 Tilgangskontroll i SOA

Tilgangskontroll i SOA utføres på lik linje som i andre arkitekturer. Når en entitet ønsker å få tilgangen til en ressurs, må entiteten først autentisere seg selv. I denne prosessen vil entiteten ut ifra en sikkerhetsmodell få tildelt sikkerhetsattributter. Attributtene viser til de rettighetene entiteten har. Sikkerhetsinfrastrukturen vil etterfølgende sjekke om de attributter entiteten kan fremstille tilfredsstiller de krav som er gitt av sikkerhetspolicyene for gjeldende ressurser. I tillegg kan sikkerhetsinfrastrukturen også hente inn informasjon i form av attributter fra tjenesten, fra miljøet, eller fra andre ressurser.

Målet er å ha et SOA system hvor tilgangskontroll til ressursene er dynamisk og avhengig av sammenhengen ressursene de blir brukt i. Dette er funksjonalitet som ikke bør bli implementert av tjenestene. En tjeneste er best realisert hvis den kun utfører oppgaven sin og ikke noe mer. For en sikkerhetsinfrastruktur i en tjenesteorientert arkitektur er det ønskelig å skille sikkerheten fra selve tjenesten. Bruk av policy og automatiserte mekanismer istedenfor å hardkode tilgangskontroller er viktige elementer.

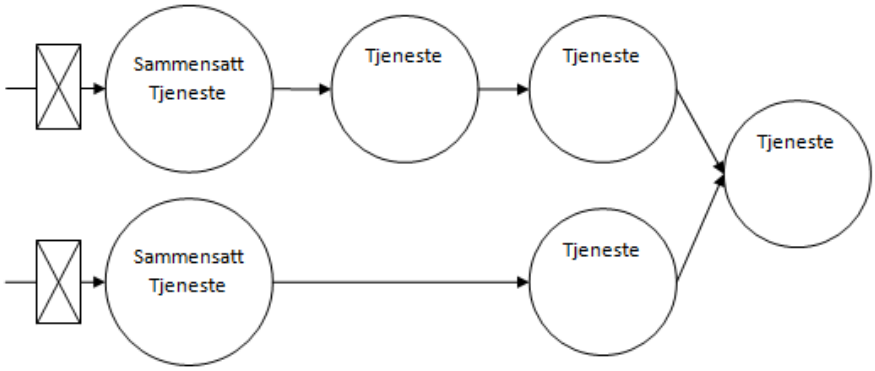
Med tilgangskontroll for SOA systemer og SOA tjenester er det enkelte momenter som bør følges. Disse er:

- Ressursene skal ikke være direkte berørt i håndhevingen av tilgang. Eksempelvis ved bruk av autorisasjonspolicy istedenfor å implementere tilgangskontroll i koden til tjenesten.
- Håndtering av tilgang skal være dynamisk. Man skal kunne endre på de krav og rettigheter som er påkrev uten store endringer i systemet.
- Mulighet for komplekse autorisasjonsavgjørelser. Håndhevingen kan også være avhengig av tilstanden til ressursen eller annet informasjon.
- Elementet som håndhever tilgangen skal ikke nødvendigvis være kjent med hvilke teknologi eller mekanismer som er tatt i bruk for håndhevingen av tilgang.

I alle tilfeller er ikke dette like lett å følge. Avhengig av hvordan tjenestene er realisert og tatt i bruk kan det bli vanskelig og altfor komplisert å lage mekanismer for håndheving av tilgang. Fra sikkerhetsanalysen påpekes det blant

annet for vanskeligheten med tilgangskontroll for sammensatte eller dynamisk oppdagede tjenester. Det er derfor viktig å designe og utvikle smarte og enkle tjenester. Selv noe som er veldig komplisert og krevende kan deles opp og forenkles. Dette gjelder også for tilgangskontroll.

For sammensatte tjenester er det ikke åpenbart for en tjenestebruker om hvilke tjenester som utgjør den sammensatte tjenesten. Enten må den sammensatte tjeneste stå for tilgangskontrollen, eller så må autorisasjonen bli delegert til de tjenestene som utgjør den sammensatte tjenesten. I referansearkitekturen er det gått ut ifra en modell for håndheving hvor tilgangskontrollen kun utføres på de sammensatte tjenestene. De tjenestene som utgjør den sammensatte tjenesten blir laget for generelt bruk og har ingen tilgangskontroll.



Figur 5.2: Tilgangskontroll for tjenestene i referansearkitekturen.

Figur 5.2 viser til hvordan tilgangskontrollen utføres for tjenestene i SOA systemet. Her designes tjenestene slik at det er kun de sammensatte tjenestene tilgjengelig til brukeren som har tilgangskontroll. Tilgangen til tjenestene vil bli bestem av policyen til tjenesten og policyene håndteres av at IAM systemet. Dette ble valgt for å unngå komplekse tjenester med tilbakekalling og for enklere håndtere tilgangen. Man slipper også situasjoner hvor policyene til tjenestene er i konflikt med hverandre eller har forskjellig betydning.

Håndheving

Hvordan tilgangen skal håndheves er opp til dem som leverer produktet for SOA systemet. Det er imidlertid en generisk modell for tilgangskontrollen som brukes av mange. For en tjenesteorientert arkitektur er det spesielt utenforstående mekanismer som står for tilgangskontrollen. Disse mekanismene bygger på en modell hvor håndhevingen av tilgang utføres ved å avskjære tjenestekallet før

tjenesten. Normalt vil håndhevingsmekanismen være basert på autorisasjonspolicyer.

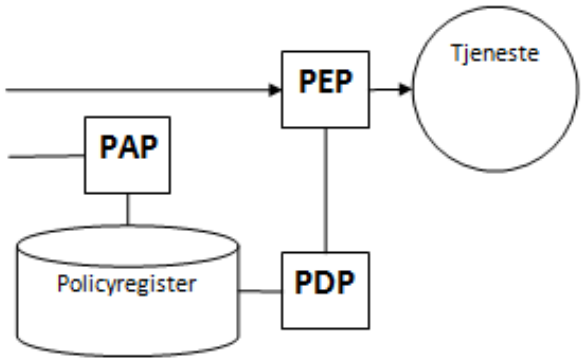
En policyhåndheving sørger for tilgangskontroll basert på berettigelsesbevis av subjektet samt policy for ressursen uten direkte å involvere selve ressursen. En policy for tilgangskontroll kan bli spesifisert for selve tjenesten, for en type tjenester, for en enkel operasjon, eller for en parameter til en operasjon. Avgjørelsen for om en operasjon kan utføres eller ikke er avhengig av autorisasjonspolicyene som er berørt av operasjonen. Tjenestekallet godtas kun hvis tjenestebrukeren har de riktige attributtene som er spesifisert i policyene.

Generell modell for tilgangskontroll

En tilgangskontrollmodell begrenser settet med tjenestebrukere som kan ta i bruk operasjoner til en SOA tjeneste. Den viktigste forskjellen mellom de ulike modellene for tilgangskontroll er om tjenesten er tilstandsløs, og om håndhevingen er sentralisert eller desentralisert. For tjenester med tilstand er det nødvendig med en tilleggsfaktor i avgjørelsen om hvorvidt det skal gis tilgang eller ikke. Avgjørelsen må baseres på om hvorvidt forespørselen tilfredsstiller de krav som er gitt av autorisasjonspolicyen og på tilstanden til tjenesten. I en desentralisert modell vil hver enkelt tjeneste stå for sin egen tilgangskontroll. Med en sentralisert modell utføres kontrollen på bakgrunn av policyen til den sammensatte tjenesten. Eventuelt kan også policyene til tjenestene som utgjør den sammensatte tjenesten bli tatt med. Men selv med denne forskjellen har de to modelltypene veldig lik struktur.

Den generelle modellen for tilgangkontroll er typisk delt opp i tre elementer som vist i figur 5.3. Til sammen utgjør disse elementene den minimale funksjonaliteten som er nødvendig for å håndheve tilgangen til ressursene. Ved å dele opp mekanismen muliggjør dette for bedre design av systemene samt bedre og flere valg for løsninger og produkter til hvert av disse elementene. Modellen har en sentralisert tilgangskontroll og tilgangen håndheves ut ifra de autorisasjonspolicyer som er definert.

Policy Enforcement Point (PEP) er elementet som håndhever tilgangen til tjenesten. PEP er elementet hvor et tjenestekall til en tjeneste enten bli forkaster eller godtatt. Avgjørelsen for dette valget blir tatt av Policy Decision Point (PDP) som sammenligner attributtene til entiteten med de kriteriene som er gitt i policyen for tjenesten. For håndteringen av alle disse policyene har man et eget element kalt Policy Administration Point (PAP). PAP fungerer som et slags administrasjonssystem for definisjonene av de ulike policyene.



Figur 5.3: Sikkerhetsmodell for tilgangskontroll i SOA.

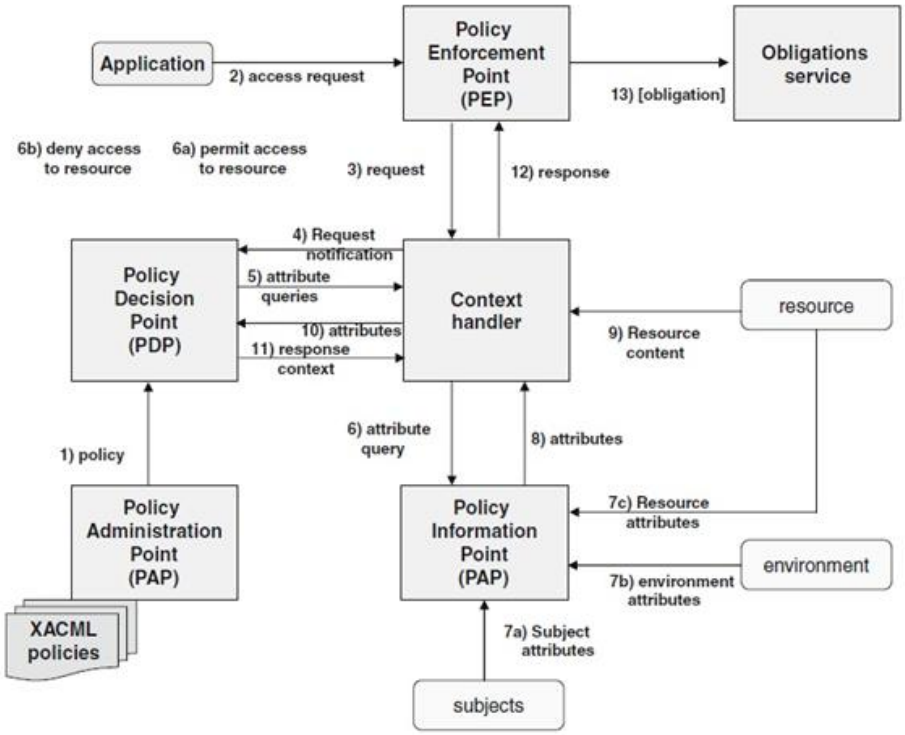
Sikkerhetsinfrastrukturen sørger for at de tjenestene som er begrenset, blir håndtert av et PEP element. PEP blir et ”single-point-of-entry” som alltid sørger for å utføre tilgangskontroll til tjenesten. I et SOA system vil en indre PEP sørger for og mappe og håndheve tilgang til det lokale miljøet. En ekstern PEP står for oppgaver som autentisering, utveksling av data, og andre sikkerhetsmekanismer. Dette kan være alt fra dekryptere deler av meldingen, validere signaturer og tokens, sørge for riktig autentiseringsnivå, eller påføre tilleggsinformasjon som dato og klokkeslett. Den eksterne PEP kan bli brukt til å avskjære alle tjenestekall inn til systemet, eller til spesifikke områder i systemet.

Normalt utføres autentiseringen på vanlig måte og resultatet blir lagt til i meldingen. Dette gjøres for å unngå flere autentiseringer i samme tjenestekall kjede. Ved å legge til autentiseringsinformasjonen til entiteten vil dette effektivisere tilgangskontrollen. Dette stiller krav til løsningen ettersom elementene ikke nødvendigvis trenger å ligge på sammen plass. Det er derfor også nødvendig med sikkerhetsmekanismer som sørger for at riktige elementer brukes og at meldingene ikke kan bli endret av andre enn de som faktisk har lov. Kanskje er det også nødvendig med kryptering av meldingene siden de inneholder sensitiv informasjon som identitets- og sikkerhetsattributter.

Det er viktig å bemerke seg at modellene for tilgangskontroll er respektive for de teknologiene som er tatt i bruk for å realisere SOA arkitekturen. I tillegg er det ikke sikkert det finnes en standard modell for hvordan tilgangskontroll bør utføres. For Web Service er det i [**12**] vist til et flertall av fremgangsmåter for å realisere tilgangskontroll. Felles for de alle er at de omhandler policy og håndtering for å utføre tilgangskontroll.

5.2.1 Extensible Access Control Markup Language

Extensible Access Control Markup Language (XACML) [**13**] er en modell for tilgangskontroll som virker veldig interessant. XACML er en OASIS standard som kan bli brukt for å spesifisere autorisasjonspolicy for håndheving av tilgang til tjenester. Fra sikkerhetsanalysen var det kommet fram til at tilgangskontroll for tjenesteorientert arkitektur er komplekst og kan bli håndhevet på mange forskjellige steder. XACML tar sikte på blant annet å løse disse problemene.



Figur 5.4: XACML sikkerhetsmodellen for tilgangskontroll i SOA.

Standarden utgjør et enkel generelt språk for å formulere policyer. Den beskriver også hvordan meldingene som sendes mellom komponentene i sikkerhetsinfrastrukturen er bygget opp. XACML er utarbeidet som en komponent i en sentral løsning for tilgangskontroll og er likegyldig til om tjenestene er tilstandsløs eller ikke. Standarden innholder spesifikasjonen på den funksjonaliteten som er nødvendig for å prosessere autorisasjonspolicyer og definere en abstrakt modell for hvordan data utveksles mellom komponentene. Figur 5.4, hentet fra figur 4.15 i [**12**], viser til modellen for XACML. Modellen inneholder PEP, PDP, og PAP for henholdsvis håndheving, avgjørelse og

håndtering av tilgang. For mer komplekse autorisasjonspolicyer har den også muligheten til å basere avgjørelsen på attributter fra miljøet til tjenesten, fra selve tjenesten, og fra andre ressurser.

XACML er basert på følgende ideer presenter i[**12**]:

- En policy må ikke være innebygget eller sterk linket til systemet som de hører til.
- Policyer skal kunne bli brukt på ulike typer ressurser uten noen begrensninger. Eksempelvis på XML dokumenter, databaser, applikasjonsservere, webtjenester, og med ulike nivå.
- Policyene skal kunne ta i betraktning spesielle karakteristikker til miljøet. For eksempel belastningen på systemet som tjenesten kjører på.
- Det skal være mulig å utveksle og dele autorisasjonspolicy, selv når systemene og policyene som er definer ikke er heterogene. Policyene internt i en organisasjon skal kunne bli oversatt til et felles policyspråk som har den samme betydningen.

Med dette standardiserer XACML syntaksen for hvordan policyspråket er definert, så vell som hvordan policyene blir behandlet og utveseklet mellom PEP og PDP. Standarden støtter for policyer hvor reglene kan være desentralisert. En policy kan bli definert av ulike organisasjoner hvor hver av dem beskriver kravene til spesielle grupper. Flere policyer kan også bli brukt i en og samme tilgangskontroll. XACML støtter med andre ord mekanismer for å spesifisere hvordan man skal ta en avgjørelse basert på flere policyer og regler fra ulike parter.

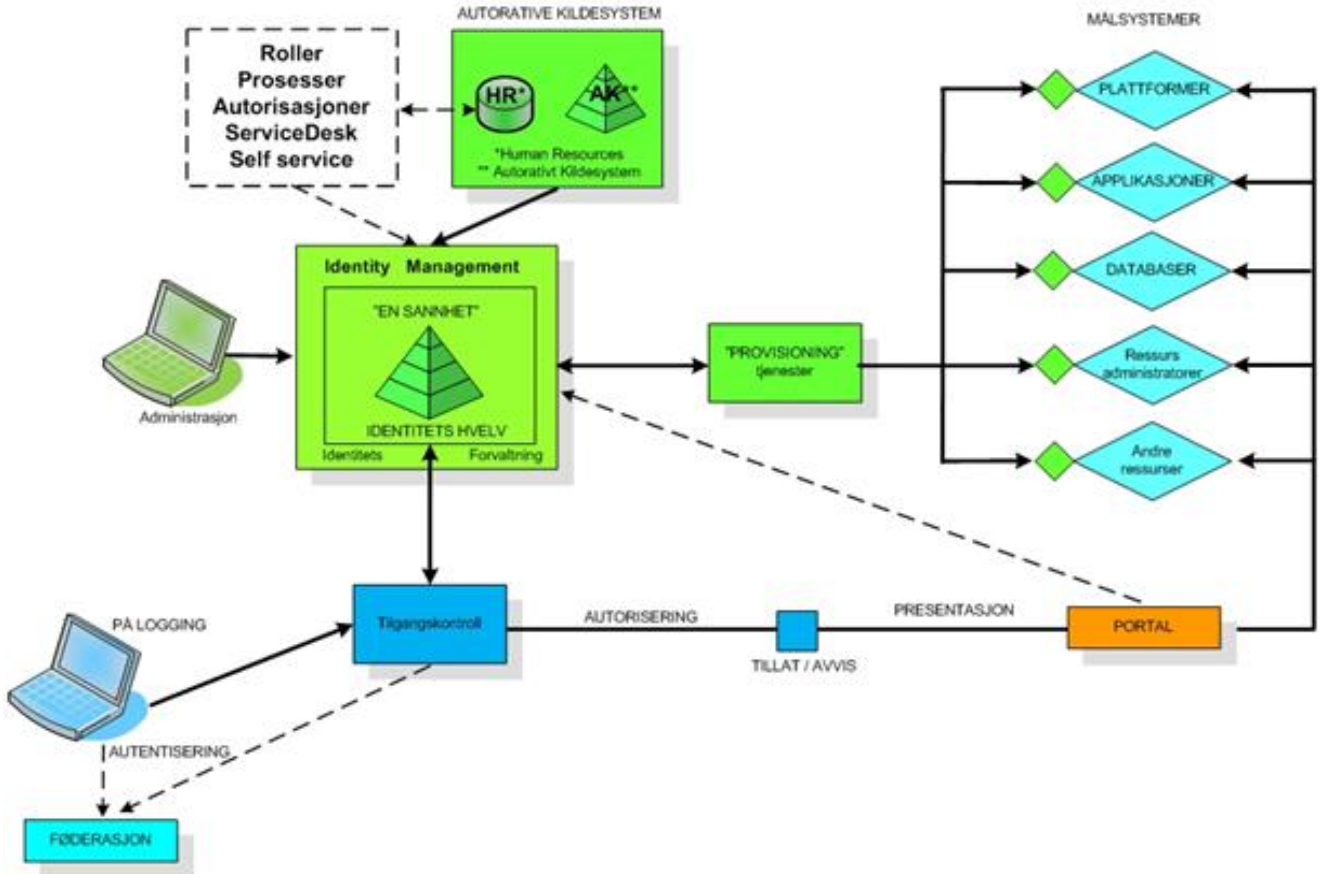
6 Sikkerhet i eSporingsløsningen

I eSporingsløsningen er den tjenesteorienterte arkitekturen realisert med Web Services. Webtjenestene er kun tilgjengelig fra en webportal. Hendelsesdata, samt masterdata, er lagret i databaser og blir tatt i bruk av webtjenestene. Figur 6.1 viser til sikkerhetsmodellen som er brukt i løsningen.

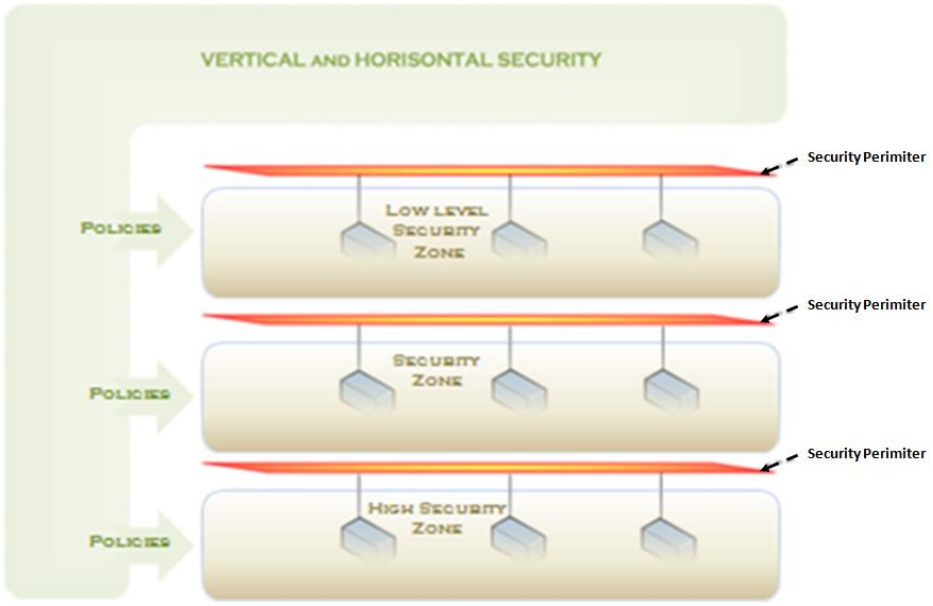
Sikkerheten i eSporingsløsningen er litt spesiell i forhold til referansearkitekturen og andre typiske arkitekturer. I modellen betraktes både infrastrukturen og tjenestene som elementer i et sikkert rom. Dette sikre rommet er illustrert i figuren som målsystemer og svarer til alle tjenester og systemer innenfor portalen. Med denne modellen oppnår man et lukket system. For eSporingsløsningen fører dette til at det bare er webtjenestene som er tilgjengelige, og det kun gjennom webportalen.

Lik referansearkitekturen sikres eSporingsløsningen med tradisjonelle mekanismer og komponenter. I eSporingsløsningen er det et IAM system som står for sikkerheten og overordnet legger føringer på tilgangen til ressursene i det sikre rommet. Systemet utfører autentiseringen og er en viktig del av tilgangskontrollen for alle ressursene i nettverket. IAM systemet kan med dette sees på som en sikkerhetstjeneste. Som i referansearkitekturen brukes også policy for å spesifisere hva som er lov og hva som ikke er lov i løsningen. Tilgangen til ressursene styres av veldefinerte policyer og roller. I tillegg brukes det et sentralt register for identitetshåndtering som er basert på NIST RBAC sikkerhetsmodellen.

For eSporingsløsningen er sikkerhetsbildet noe endret i forhold til tradisjonell vertikal sikkerhet. På bakgrunn av tjenestelaget i eSporingsløsningen settes kravene til både vertikal og horisontal sikkerhet. Se figur 6.2 for en visualisering av sikkerhetsmodellen. Med tradisjonell vertikal sikkerhet menes soneinndelinger med bruk av sikkerhetsbarrierer mellom sonene. Dette gjelder både lokalt og mot ekstern tilgang. For en tjenesteorientert arkitektur er ikke dette godt nok. På bakgrunn av hvordan tjenestene er brukt, kan det være nødvendig med sikkerhet for tjenester i samme sikkerhetssonen. Horisontal sikkerhet er derfor også et krav. Det er IAM som opprettholder denne sikkerheten med tradisjonell vertikal sikkerhet (sonemodell) og horisontal sikkerhet (tjenesteorientert). Dette gjøres ved å benytte samme policy for likeverdige tjenester og tilganger i samme sikkerhetssone, samtidig som den vertikale sikkerheten er ivaretatt.



Figur 6.1: Sikkerhetsmodellen brukt i eSporingsløsningen.



Figur 6.2: Vertikal (infrastruktur) og horisontal (tjenester) sikkerhet.

Bakgrunnen for valget av en arkitektur hvor all autentisering og tilgangskontroll styres av et overordnet system, var i hovedsak todelt:

- Et relativt høyt antall tjenestekall skulle håndteres. Dagens løsninger for Web Service ville ikke klare å oppfylle dette kravet. Isteden tar man i bruk et spesialisert system for tilgangskontroll. IAM er et slikt system og skal ikke ha problemer med å oppfylle kravet.
- For organisasjonene som skal ta i bruk eSporingsløsningen var det viktig med et sikkerhetssystem som stod for tilgangskontrollen til alle ressursene. For ikke å få bedriftshemmeligheter avslørt måtte sikkerhetsmodellen beskytte alle ressursene i systemet. Med modellen verifiseres alltid rettighetene til en entitet med IAM systemet.

Neste følger en nærmere beskrivelse av autentisering og tilgangskontrollen i eSporingsløsningen og forholdet deres til referansearkitekturen.

6.1 Autentisering

Når en bruker første gang kommer inn til nettverket, vil brukeren først være innom IAM systemet for autentisering. Autentiseringen utføres i en egen avgrenset og sikker tjeneste i IAM systemet. Systemet autentiserer brukeren og legger til nødvendig sikkerhetsinformasjon relatert til sesjonen i HTTP headeren.

Etter vellykket autentisering blir brukeren sendt til webportalen. All kommunikasjons mellom bruker og webportal utføres med HTTPS. All informasjon vedrørende autentiseringen blir også kommunisert med HTTPS og denne sikkerhetsinformasjonen er gyldig for sesjonens varighet.

IAM systemet brukt i eSporingsløsningen er et dedikert system. Systemet støtter ulike nivåer og former for autentisering. Alt fra brukernavn og passord, som den svakeste metoden, til sterkere metoder som Public Key Infrastructure eller digitale sertifikater kan bli brukt. For mer distribuerte løsninger er det også mulig med føderasjon og SSO mekanismer.

Med IAM systemet har eSporingsløsningen et felles system for identitetsforvaltning, håndtering av sikkerhetsmekanismer, og logging av sikkerhetsrelevant data. Selv om det ikke er mulig å se noen direkte svakheter med dette systemet, kan det bemerkes to ting som er felles for alle slike systemer. Fra sikkerhetsanalysen har man, for det første, det å samle all sikkerhetsinformasjon i et system kan bli sett på som en sårbarhet. For det andre er man avhengig av ikke å gi administrative rettigheter til personer som kan misbruke den tillitten de har fått. Dette er den avgjørende faktoren for en løsning som tar i bruk et sentralt register. Uansett hvor sikkert systemer er kan det alltids omgås for en person som har administrative rettigheter.

Når dette er sagt, er det viktig å gjøre en bemerkning vedrørende eSporingsløsningen. De rettighetene (rollene) man blir tildelt er definert på en slik måte at man ikke skal ha mulighetene til å gjøre noe utenom det man er berettiget til. En bruker i en organisasjon som har eierrettigheter vil ikke kunne gjøre noe annet enn å fordele rettigheter til andre brukere for de ressursene som organisasjonen er eiere av. Med andre ord, ved å være nøye på defineringen av rettighetene til rollene skal det mye til for en enkel person å gjøre ondsinnede handlinger.

Utover dette er det ikke store forskjeller mellom eSporingsløsningen og referansearkitekturen med tanke på autentisering. Begge har et felles register for identiteter og det er gått ut ifra et sentralisert system. Den eller de som håndhever tilgangen må kontakte systemet for å få autentisert brukeren.

6.2 Tilgangskontroll

Den største forskjellen mellom referansearkitekturen og eSporingsløsningen, er utførelsen av tilgangskontroll. I eSporingsløsningen er det gått ut ifra et sikkert rom hvor brukere får tilgang til tjenester og ressurser gjennom webportalen. Referansearkitekturen er ikke designet slik ettersom en mer åpen modell er valgt. For referansearkitekturen er det ikke en overordnet tjeneste som alle må forholde seg. Derimot er alle tjenestene tilgjengelig så lenge kravene for når tjenestene kan tas i bruk, er opprettholdt.

Den første tilgangskontroll en bruker møter på er ved første inngang til eSporingsløsningen. Dette er det første og viktigste punktet for tilgangskontroll og fungerer som den første forsvarslinjen. En bruker blir her autentisert og får tildelt de rettigheter den har krav på, før den blir sendt videre til webportalen. På bakgrunn av sikkerhetsinformasjonen fra autentisering i IAM systemet vil webportalen kun vise de tjenestene som er tilgjengelig for brukeren. Webportalen blir et ledd i tilgangskontrollen ved å begrense innsynet til tjenestene for en bruker.

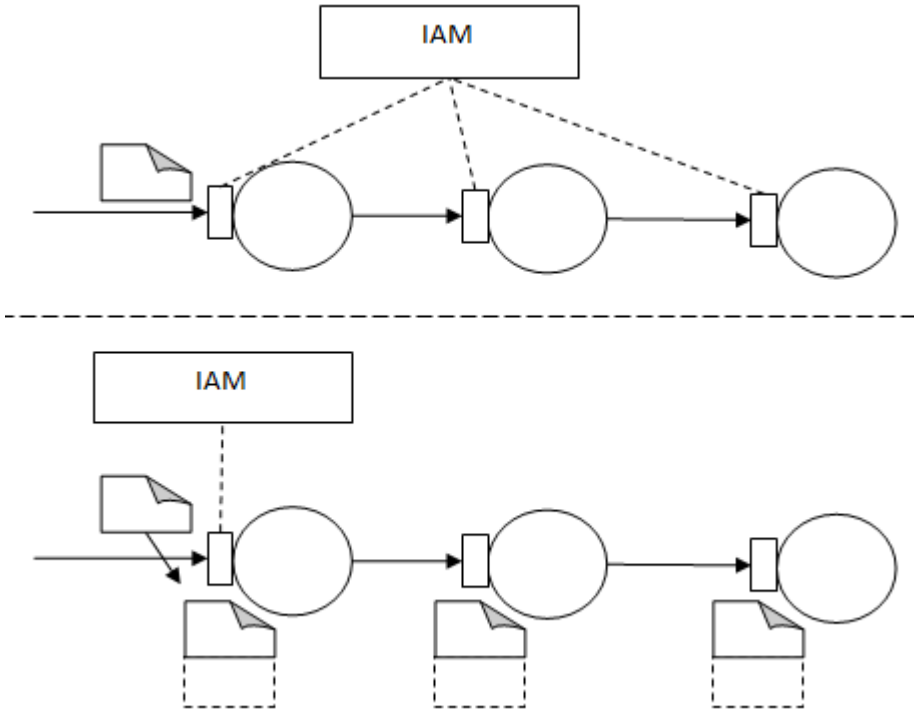
Dette er den ytre tilgangskontrollen til eSporingsløsningen og er teknisk sett lik typisk brukte mekanismer. Det sikre rommet og den indre tilgangskontrollen er derimot litt mer spesielle. For i tillegg til å legge begrensinger på at det er kun webportalen som kan ta i bruk webtjenestene, utføres det også tilgangskontroll ved hver ressurs inne i det sikre rommet. Ved et tjenestekall vil portalen sende med autentiseringsinformasjonen fra IAM systemet. Før en autorisasjonsavgjørelse blir tatt vil tilgangskontrollen med dette utføre et API kall mot IAM for å verifisere om brukeren har de nødvendige rettighetene som er påkrevd. Med andre ord vil man alltid sjekke opp mot IAM om den sikkerhetsinformasjonen som er gitt i tjenestekallet stemmer. Brukeren skal ikke kunne opphøye sin egen tilgang.

I eSporingsløsningen blir sikkerhetsinformasjonen lagt til tjenestekallet. Det er dermed opp til hver enkelt tjeneste å verifisere denne sikkerhetsinformasjonen med IAM systemet før tjenesten kan bli tatt i bruk. Stemmer ikke informasjonen, eller har ikke bruker de riktige rettighetene, vil den få tjenestenekt. På denne måten er alle tjenestene sikret. Skulle en bruker prøve å endre på informasjonen vil dette bli oppfanget av IAM systemet. Eller, hvis brukeren prøver å omgå enkelte tjenester, vil de neste tjenestene uansett håndheve sin egen tilgang og fange opp dette.

Med dette er selve tilgangskontrollen i eSporingsløsningen forklart. Den består av en ytre og en indre tilgangskontroll, separert på hver sin side av webportalen. Den ytre tilgangskontrollen utføres når en bruker første gangen kommer inn til nettverket og en sesjon med webportalen oppretts. Den indre tilgangskontroll, betegnet som en verifisering, utføres ved alle tjenester etterfølgende webportalen. Man kommer kun inn til den indre delen ved å gå igjennom webportalen.

I tillegg er det enda en vesentlig forskjell mellom eSporingsløsningen og referansearkitekturen. I eSporingsløsningen er ikke tilgangskontrollen basert på meldingen i tjenestekallet. For å sjekke om brukeren har rett til å ta i bruk tjenesten, må man alltid innom IAM for å verifisere de påstandene som er gitt av brukeren i tjenestekallet. Dette er ikke direkte galt, men fordi man har to parter å forholde seg til er dette ikke det mest optimale. Det kan være misvisende og lede til gal håndheving. Men aller mest er det en unødvendig oppgave å måtte verifisere gang på gang i samme sammenheng, for samme forespørsel. En mer effektiv måte ville vært å gå ut ifra en verifisering kun når man kommer inn i det sikre rommet. Etter verifiseringen sørges det for at sikkerhetsinformasjonen i meldingen ikke kan endres. Figur 6.3 illustrerer dette. Man går dermed ut ifra de rettighetene bruk i verifisering gjelder for hele oppgaven som skal utføres og ikke bare for det ene tjenestekallet. Denne formen for å legge ved nødvendig informasjon i meldingene er også mer i tro med designprinsippene for en tjenesteorientert arkitektur.

Et annet problem med designet slik eSporingsløsningen er per dags dato er hvordan selve tilgangskontrollen i hver enkelt tjeneste utføres. Dette stiller krav til å implementere tilgangskontrollen riktig for hver enkelt tjeneste. Hvis ved en programmeringsfeil tjenesten utelater eller har gal håndheving, vil det ikke lenger hjelpe med å verifisere brukeren mot IAM systemet først. Det ville vært mer i tro med en tjenesteorientert arkitektur hvis dette var for eksempel utført av en automatisert mekanisme som omslutter tjenesten. Men, på grunn av designet av eSporingsløsningen vil kun tjenesten være kompromittert. De andre tjenestene og systemene vil fortsatt være sikre.



Figur 6.3: Vanlig tilgangskontroll og tilgangskontroll i SOA sammenheng.

6.3 Videre utvikling

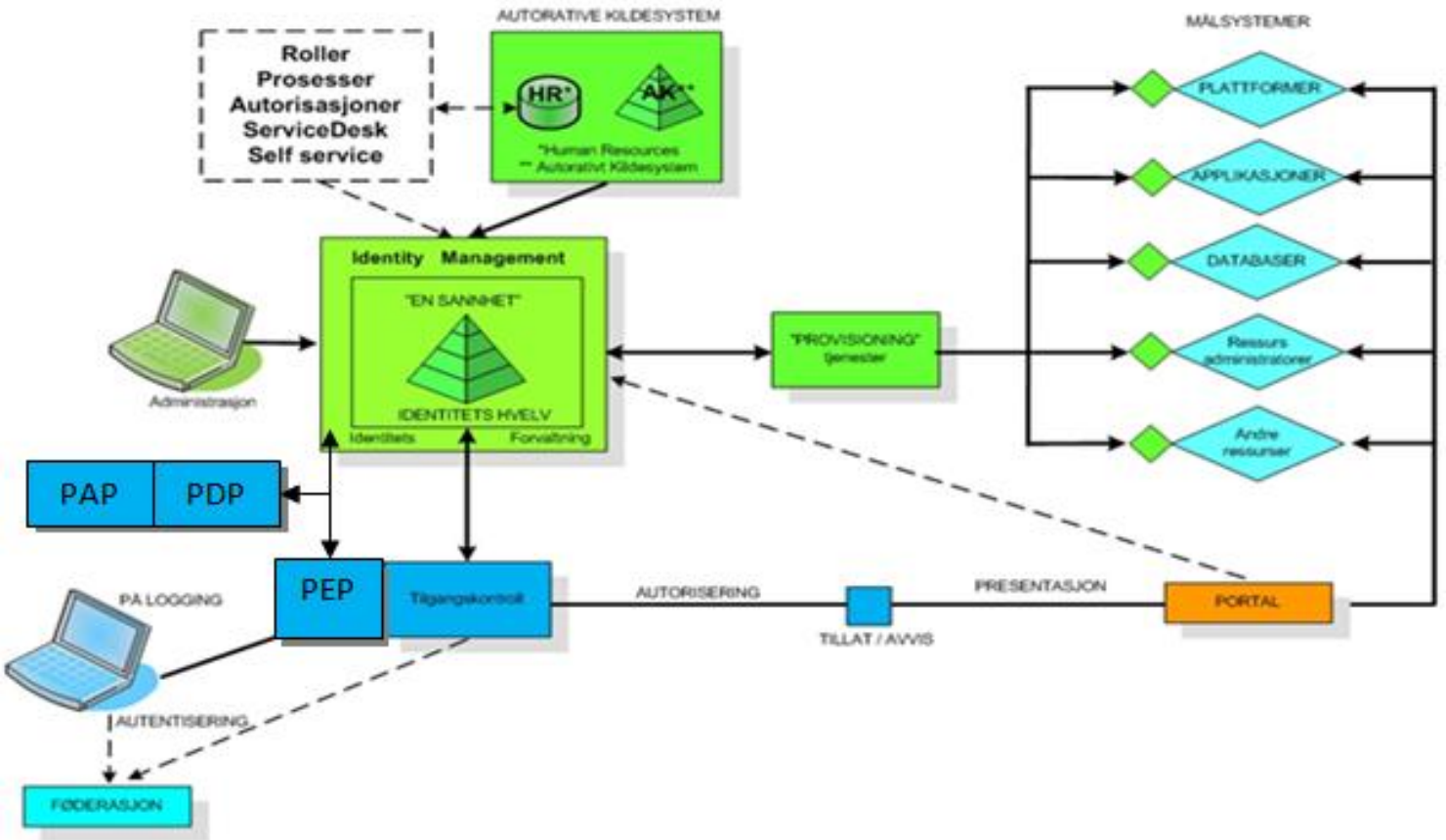
Hvis det en gang i fremtiden er ønskelig å tilby flere applikasjoner, kan det være at man må gjøre betydelige inngrep i eSporingsløsningen. For eksempel kunne man laget en tjeneste hvor kunder i en butikk kan skanne produktet og få opp hvor innholdet i varen kommer fra. Hvis dette var tilfellet ville webportalen mest sannsynlig ikke klare den nye mengden forespørsler som vil bli generert.

En løsning hadde vært å åpne opp deler av det sikre rommet med eksempelvis dedikerte webtjenester. Dette ville krevd en annen måte å verifisere tjenestekall på. Her ville man vært nødt til å ta i bruk en mer tjenesteorientert løsning for autentisering og tilgangskontroll. For eksempel med en modell hvor sikkerhetsinformasjon følger med meldingen. Med dette kan man utføre tilgangskontroll en gang og la de etterfølgende tilgangskontrollene være basert på melding.

Selv om eSporingsløsningen er bygget rundt en tjenesteorientert arkitektur, er den langt ifra å utnytte alle de muligheter som arkitekturen muliggjør og

oppfordrer til å ta i bruk. Det som ville vært mest interessant å se på, hadde vært en mer avansert og automatisert tilgangskontroll rettet mot den tjenesteorienterte arkitekturen. Dette ville ikke by på de altfor store utfordringer.

Web Services Profile of XACML (WS-XACML) [**14**] er en kandidat som kunne passet ypperlig inn. Dette er en standard utarbeidet av OASIS for å kunne ta i bruk XACML sikkerhetsmodellen i Web Services. Denne modellen er allerede presentert i referansearkitekturen, se kapittel 5.2.1, og har egenskaper som kanskje kunne ha passet for et slikt scenario. Avhengig av de produkter som finnes på markedet kunne man kanskje også implementere deler av modellen i IAM systemet. For eksempel kunne man se for seg en løsning hvor systemet for webtjenestene sørger for PEP mot selve tjenesten mens PDP og PAP er dedikerte elementer innebygget i IAM systemet. Figur 6.4 viser hvordan dette kunne sett ut i sikkerhetsmodellen til eSporingsløsningen.



Figur 6.4: WS-XACML i sikkerhetsmodellen til eSporingsløsningen.

7 Oppsummering og konklusjon

Dagens sikkerhetsmodeller og mekanismer er ikke beregnet for tjenesteorienterte arkitekturer. Med egenskaper som ubegrenset tilgjengelighet til data og ressurser, dynamisk konfigurasjon av applikasjoner, rekonfigurasjon gjennom arbeidsflyt, løs kopling og autonome tjenester stilles det nye utfordringer. I tillegg til de mer tradisjonelle sikkerhetsmetoder som lag- og soneinndeling må også nye mekanismer for tilgangskontroll tas i bruk. Tjenester skal blant annet kunne brukes i ulike domener.

Mekanismer som beskytter tjenestene må kunne ta i bruk sikre metoder for autentisering og tilgangskontroll. Dette må nødvendigvis ikke bli gjort av selve tjenesten, men kan for eksempel bli delegert til en sikkerhetsinfrastruktur. En sikkerhetsinfrastruktur vil normalt håndheve tilgangen til tjenestene basert på sikkerhetspolicyer. Med en tjenesteorientert arkitektur er det ønskelig med en kompleks sikkerhetsinfrastruktur fremfor tjenester som håndhever sin egen tilgang.

Den presenterte referansearkitekturen viser til en slik sikkerhetsinfrastruktur. Med arkitekturen har oppgaven kommet frem til en fremgangsmåte for å sikre tjenester som også letter håndteringen av identiteter og tilgang.

eSporingsløsningen tar i bruk mange av de samme mekanismene og metodene som det er kommet fram til i oppgaven. Men, selv om IAM systemet brukes av webtjenestene for autentisering og verifisering av rettigheter, er det tjenestene selv som står for tilgangskontrollen. eSporingsløsningen har ingen mekanismer for å lette utviklingen og håndhevingen av tilgang til webtjenestene.

Referanser

[1] eSporing. (2009, Juli) Faktaark 1: eSporingsprosjektet. [Online]. http://www.regjeringen.no/upload/subnettsteder/esporing/Vedlegg/Faktaark_1_eSporingsprosjektet.pdf

[2] eSporing. (2009, April) eSporing Veileder. [Online]. http://www.kunnskapsnettverk.no/C10/C19/E-sporing/Lists/Kunngjringer/Attachments/13/Veileder%20eSporing_versjon%202.0.pdf

[3] EPCglobal. (2007, September) EPC Information Services (EPCIS). [Online]. http://www.epcglobalinc.org/standards/epcis/epcis_1_0_1-standard-20070921.pdf

[4] W3C Web Services Architecture Working Group. (2004, Februar) Web Services Architecture. [Online]. http://www.w3.org/TR/ws-arch/wsa.pdf

[5] SANS. (2009, September) The Top Cyber Security Risks. [Online; aksessert 12 Juni 2010]. http://www.sans.org/top-cyber-security-risks/

[6] SANS. (2010, Februar) CWE/SANS TOP 25 Most Dangerous Programming Errors. [Online; aksessert 12 Juni 2010]. http://www.sans.org/top25-programming-errors/

[7] Shawn Hernen, Scott Lambert, Tomasz Ostwald, and Adam Shostack, "Threat Modeling: Uncover Security Design Flaws Using The STRIDE Approach," *MSDN Magazine*, November 2006.

[8] Graham Williamson, David Yip, Ilan Sharoni, and Kent Spaulding, *Identity Management: A Primer*. Mc Press Online, 2009.

[9] J.D Meier et al., *Improving Web Application Security: Threats and Countermeasures*. Microsoft Corporation, 2003.

[10] Michael Hafner and Ruth Breu, *Security Engineering for Service-Oriented Architectures*. Springer-Verlag Berlin Heidelberg, 2009.

[11] Wikipedia. Sarbanes–Oxley Act. [Online; aksessert 13 Juni 2010]. http://en.wikipedia.org/wiki/Sarbanes%E2%80%93Oxley_Act

[12] Elisa Bertino, Lorenzo Martino, Federica Paci, and Anna Squicciarini, *Security for Web Services and Service-Oriented Architectures*. Springer-Verlag Berlin Heidelberg, 2010.

[13] OASIS. (2010, Mars) eXtensible Access Control Markup Language (XACML). [Online]. http://www.oasis-open.org/committees/download.php/37639/xacml-3.0-core-spec-cd-03-en.zip

[14] OASIS. (2007, August) Web Services Profile of XACML (WS-XACML). [Online]. http://www.oasis-open.org/committees/download.php/24951/xacml-3.0-profile-webservices-spec-v1-wd-10-en.pdf

[15] Erl Thomas, *SOA Design Patterns*. Prentice Hall, 2009.

[16] Erl Thomas, *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.

[17] Eric Pulier and Taylor Hugh, *Understanding Enterprise SOA*. Manning Publications, 2005.

[18] eSporing. (2009, Juli) Faktaark 3: Effekter og gevinister. [Online]. http://www.regjeringen.no/upload/subnettsteder/esporing/Vedlegg/Faktaark_3_Effekter_og_gevinister.pdf

[19] eSporing. (2009, Juli) Faktaark 2: eSporingsløsningen. [Online]. http://www.regjeringen.no/upload/subnettsteder/esporing/Vedlegg/Faktaark_2_eSporingsløsningen.pdf

Vedlegg A Spørsmål til sikkerhetsprofil

Category	Considerations
Input validation	Is all input data validated? Could an attacker inject commands or malicious data into the application? Is data validated as it is passed between separate trust boundaries (by the recipient entry point)? Can data in the database be trusted?
Authentication	Are credentials secured if they are passed over the network? Are strong account policies used? Are strong passwords enforced? Are you using certificates? Are password verifiers (using one-way hashes) used for user passwords?
Authorization	What gatekeepers are used at the entry points of the application? How is authorization enforced at the database? Is a defense in depth strategy used? Do you fail securely and only allow access upon successful confirmation of credentials?
Configuration management	What administration interfaces does the application support? How are they secured? How is remote administration secured? What configuration stores are used and how are they secured?
Sensitive data	What sensitive data is handled by the application? How is it secured over the network and in persistent stores? What type of encryption is used and how are encryption keys secured?
Session management	How are session cookies generated? How are they secured to prevent session hijacking? How is persistent session state secured? How is session state secured as it crosses the network? How does the application authenticate with the session store? Are credentials passed over the wire and are they maintained by the application? If so, how are they secured?
Cryptography	What algorithms and cryptographic techniques are used? How long are encryption keys and how are they secured? Does the application put its own encryption into action? How often are keys recycled?

Parameter manipulation	Does the application detect tampered parameters? Does it validate all parameters in form fields, view state, cookie data, and HTTP headers?
Exception Management	How does the application handle error conditions? Are exceptions ever allowed to propagate back to the client? Are generic error messages that do not contain exploitable information used?
Auditing and logging	Does your application audit activity across all tiers on all servers? How are log files secured?