Implementasjon av attributtbasert tilgangskontroll i elektroniske helsesystemer

André Sæ Kristensen
MASTEROPPGAVE VED INSTITUTT FOR INFORMATIKK
UNIVERSITETET I BERGEN

Lang oppgave
28. mai 2015

Innhold

Figurer

Tabeller

Eksempler

Forkortelser

ABAC Attribute Based Access Control.

ACL Access Control List.

AJAX Asynchronous JavaScript and XML.

AMPATH Academic Model for Providing Access to Healthcare.

API Application Program Interface.

CRUD Create Read Update Delete.

CSRF Cross-Site Request Forgery.

DAC Discretionary Access Control.

DDD Domain Driven Design.

DHIS2 District Health Information System 2.

DRY Don't repeat yourself.

GAE Google App Engine.

GPS Global Positioning System.

GWT Google Web Toolkit.

HERAS^{AF} Holistic Enterprise-Ready Application Security Architecture Framework.

HTTPS Hypertext Transfer Protocol Secure.

J2EE Java Platform Enterprise Edition.

JAX-RPC Java API for XML-based RPC.

JDBC Java Database Connectivity.

JNDI Java Naming and Directory Interface.

JPA Java Persistence API.

JSP Java Server Pages.

JSTL JSP Standard Tag Library.

KISS Keep It Simple Stupid.

MAC Manditory Access Control.

MD5 Message-Digest algorithm 5.

MFLAC Missing Function Level Access Control.

MVC Model View Controller.

MySQL My Structured Query Language.

NIST National Institute of Standards and Technology.

ODK Aggregate Open Data Kit Aggregate.

OID OpenID.

OpenMRS Open Medical Record System.

OWASP Open Web Application Security Project.

PAP Policy Administation Point.

PBAC Policy-Based Access Control.

PDP Policy Decision Point.

PEP Policy Enforcment Point.

PIP Policy Information Point.

PoC Proof of Concept.

PRP Policy Repository Point.

RAdAC Risk-Adaptable Access Control.

RBAC Role Based Access Control.

ReBAC Relationship Based Access Control.

RPC Remote Procedure Calls.

SOA Service-Oriented Architecture.

SOAP Simple Object Access protocol.

SpEL Spring Expression Language.

SQL Structured Query Language.

SRP Single Responsibility Principle.

SSFC Spring Security Filter Chain.

SVN Subversion.

UNIX Uniplexed Information and Computing System.

URL Uniform Resource Locator.

WSO_2 IS Web Services $Oxygen_2$ Identity Server.

XACML eXtensible Access Control Markup Language.

XML eXtensible Markup Language.

XSS Cross Site Scripting.

Forord

Jeg vil takke mine veiledere Federico Mancini og Khalid A. Mughal for god veiledning og inspirasjon i hele skriveprosessen til oppgaven. Jeg vil også takke PHD kandidat Samson H. Gejibo for god hjelp i oppstartfasen av prosjektet. En ekstra takk til Nicolai Gjellestad, min søster Anne Line og far Arild for korrekturlesing, og min mor Marianne for god støtte i hele studietiden. Til slutt vil jeg takke min kjære Mari I. Hodnefjell for støtte og motivasjon hun ga meg til å fullføre oppgaven.

Mai 2015
André Sæ Kristensen

Sammendrag

André S. Kristensen, Universitetet i Bergen

Sammendrag av masteroppgaven, innlevert 28. mai 2015:

Implementasjon av attributtbasert tilgangskontroll i elektroniske helsesystemer

Tilgangskontroll er et av de viktigste temaene innenfor informasjonssikkerhet[10]. Sensitiv data bør bare kunne aksesseres av autoriserte brukere eller programmer.

Denne oppgaven har som hovedmål å undersøke den nåværende tilgangskontrollen i to utbredte elektroniske helsesystemer: Open Medical Record System (OpenMRS) og Open Data Kit Aggregate (ODK Aggregate). Videre skal det utforskes muligheten for å implementere en mer fleksibel tilgangskontroll i overnevnte systemer med bruk av Attribute Based Access Control (ABAC). Før undersøkelsen blir utført skal det kartlegges eksisterende tilgangsmekanismer med deres fordeler og ulemper.

Konklusjonen av arbeidet er at implementasjon av ABAC i de overnevnte systemene ikke kan gjøres på en optimal måte uten å måtte redesigne dem. Deler av den eksisterende tilgangskontrollen må derfor beholdes, og bare en begrenset ABAC kan implementeres. Dette er fordi systemene er utviklet for en annen type tilgangskontroll og det er blitt tatt valg under utviklingen som gjør at tilgangskontrollen er for tett integrert i applikasjonen.

Basert på erfaringer med OpenMRS og ODK Aggregate ble det laget et elektronisk helsesystem med navnet Medical, der ABAC ble tatt med fra starten og implementert på en optimal måte. Medical prosjektet er et Proof of Concept (PoC) prosjekt som beviser hvor fleksibelt attributtbasert tilgangskontroll kan være når det blir tatt med i designfasen.

1 Introduksjon

Sikkerhet i applikasjoner er viktig i alle bedrifter, og tilgangskontroll med autorisasjon av brukere er et av de mest kritiske områder innenfor informasjonssikkerhet[10]. I bedrifter og applikasjoner hvor det er mange brukere som koordineres er effektiv tilgangskontroll en nødvendighet. Tilgangskontroll kontrollerer og definerer autorisasjonen brukeren trenger for å få tilgang til den rette informasjonen[11]. Tilgangskontroll brukes til å begrense risikoen for uautorisert tilgang til informasjon som kan resultere i tyveri og spredning av hemmelig og sensitiv data. Dette kan i verste fall medføre at bedrifter taper renommé og mister tillit. Alvorlige brist kan lede til stort erstatningsansvar og i ytterste konsekvens konkurs. Usikret data og svake tilgangskontroller er med på å svekke sikkerheten i dagens informasjonssamfunn[10].

«*Internet of Things*», helintegrerte systemer og brukernes krav om tilgjengelig informasjon over alt til enhver tid stiller nye og strenge krav til sikkerhet, og dette ikke minst i elektroniske helsesystemer[12][13].

I listen *Top 10 Application Security Flaws* som er utviklet av Open Web Application Security Project (OWASP), er tre av ti av de største sikkerhetsrisikoene forbundet med autorisasjon [14].

I denne oppgaven skal kandidaten se på åpne kildekodeprosjekter[1] (eng. open source) for innsamling av pasientdata i utviklingsland. Eksempler er Open Data Kit Aggregate (ODK Aggregate)[16], mUzima[17], District Health Information System 2 (DHIS2)[18], Open Clinica[19] og Open Medical Record System (OpenMRS)[20]. Prosjektene sliter med

[1] Prosjekter som er tilgjengelige for redistribusjon [15].

å ha nok kvalifisert personell til å samle inn data [21], og bruker derfor ufaglært personell på grunn av det er mer kostnadseffektivt. I denne sammenheng blir tilgangskontroll en kritisk komponent på grunn av de sensitive dataene som blir innsamlet. Det er stort behov for å skjerme pasientdata[22]. De som registrerer data trenger ikke full tilgang til systemet. I tillegg er prosjektene preget av mange utviklere med ulike kompetansenivå, og de drives uavhengig av hverandre. Dette medfører varierende kvalitet og et mindre fokus på datasikkerhet. I oppgaven vil kandidaten gå dypere inn i OpenMRS og ODK Aggregate.

1.1 Mål for oppgaven

Denne oppgaven har følgende hovedmål:

1. Utforske tilgangskontrollen i to elektroniske helsesystemer OpenMRS og ODK Aggregate.

2. Utforske muligheten til å integrere en mer fleksibel tilgangskontroll i overnevnte systemer med bruk av ABAC.

Et delmål er å kartlegge eksisterende tilgangsmekanismer med deres fordeler og ulemper. Det andre delmålet er å utvikle et Proof of Concept på elektronisk helsesystem basert på erfaringer tilegnet under implementasjonen av ABAC i OpenMRS og ODK Aggregate. Dette for å utforske mulighetene til ABAC når det er en del av designet til et nytt system.

1.2 Motivasjon

Hvem som har tilgang til de ulike ressurser er ofte ikke en prioritet i web utvikling, men det er en kritisk faktor for å kunne behandle sensitiv data[23]. Motivasjonen for å implementere ABAC i eksisterende systemer som OpenMRS og ODK Aggregate er for å forbedre informasjonsdelingen, gjøre autorisasjonen dynamisk og ivareta kontroll på informasjonen. Ved å implementere ABAC er teorien at det skal bli enklere å vedlikeholde og utvide informasjonssikkerheten.

1.3 Utfordringer med ABAC og implementasjon i eksisterende prosjekter

Det er mange utfordringer knyttet til å implementere i tredjepartssystemer da en på forhånd ikke kan vite hvordan den nye autorisasjon vil fungere i det gamle systemet. Det samme gjelder for kommunikasjonen og integrasjonen mellom de forskjellige komponentene i systemet. Videre er det å lære seg XACML språket en utfordring i prosjektet. Dette på grunn av at språket er sterkt syntaksdrevent og det er lite dokumentasjon hvordan komponentene skal fungere sammen.

1.4 Omfang / avgrensninger

Omfanget til masteroppgaven vil være ODK Aggregate, OpenMRS, eXtensible Access Control Markup Language (XACML), Web Services Oxygen$_2$ Identity Server (WSO$_2$ IS) og følgende utvalgte tilgangskontroller; Role Based Access Control (RBAC) og Attribute Based Access Control (ABAC). Andre tilgangskontroller vil bli diskutert men ikke gått i dybden på. Oppgaven vil legge vekt på informasjonssikkerhet, autorisasjon og implementasjonen av ABAC i eksisterende systemer og sammenligne RBAC og ABAC.

1.5 Relatert arbeid

I *Attribute Based Access Control (ABAC) for Web Service* [24], blir det diskutert hvor lite dynamisk tilgangskontroller er i Service-Oriented Architecture (SOA). ABAC løsningen som blir beskrevet bruker Simple Object Access protocol (SOAP) og Java API for XML-based RPC (JAX-RPC) fremgangsmåte. Den formulerer også policyer som logiske utrykk.

I *Attribute Based Access Control for APIs in Spring Security* [25], diskuteres det attributtbasert tilgangskontroll for API med bruk av et kjent sikkerhetsrammeverk (Spring Security). Det diskuteres også hvordan Spring Expression Language (SpEL)[26] kan gi skreddersydd tilgang.

I *Comparative analysis of RBAC and ABAC*[27] blir tilgangskontrollene sammenlignet og fordeler og ulemper diskutert. En ulempe med RBAC i forhold til ABAC er at den ikke støtter kontekst.

I *ABAC and RBAC: Scalable, Flexible, and Auditable Access Management*[28] diskuteres en kombinasjon av ABAC og RBAC der det kombineres det beste fra begge modellene.

I *A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC* [29] presenterer de ABACα, et konsept som vil innkapsulere DAC, MAC, og RBAC med ABAC.

I masteroppgaven *Policy based Electronic Medical Record System* [30] blir Policy-Based Access Control (PBAC) implementert i OpenMRS. Implementasjonen bygger på eksisterende tilgangskontroll. Oppgaven blir sammenlignet med implementasjonen kandidaten gjorde i kapittel 4 i seksjon 4.6.

1.6 Hypoteser

På bakgrunn av tidligere arbeid [24], [25], [27], [30] postulerer kandidaten følgende hypoteser for oppgaven ved implementasjon av ABAC i OpenMRS og ODK Aggregate:

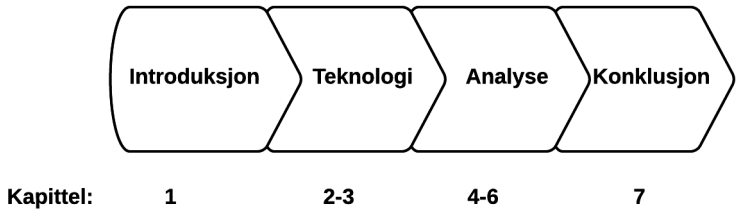
1. Ved implementasjon av ABAC i OpenMRS vil en kunne skille pasienter på behandlingsavdeling.
2. Ved implementasjon av ABAC i OpenMRS vil en kunne frigjøre rolleprivilegiumproblemet[2].
3. Ved implementasjon av ABAC i ODK Aggregate vil en kunne få dynamisk tilgangskontroll.
4. Ved implementasjon av ABAC i ODK Aggregate vil en kunne begrense tilgang til skjemaene med rolleattributter.

Videre vil de samme hypotesene bli testet i Medical Proof of Concept prosjektet.

[2]Rolleprivilegiumproblemet er mange statiske privilegier som må kobles til egendefinerte roller.

1.7 Strukturen til oppgaven

Oppgaven er organisert i syv kapitler. Figur 1.1 viser hvordan kapitlene er oppdelt.



Figur 1.1: Strukturen til oppgaven.

Kapittel 2 Utvalgte tilgangskontrollmodeller: Beskriver eksisterende tilgangskontroller og fordeler og ulemper med dem.

Kapittel 3 Implementasjon av ABAC og relevante teknologier: Belyser overveielser en utvikler må ta når det skal implementeres ABAC i tredjepartssystemer, og gir en grundig innføring i XACML sine hovedprinsipper og begreper. I tillegg vil det bli gitt en innføring i WSO_2 IS

Kapittel 4 Implementasjon av ABAC i OpenMRS: Gir en oversikt over OpenMRS, blant annet hvilken tilgangskontroll den bruker og problemer med nåværende tilgangskontroll. Det vil også bli beskrevet en tidligere masteroppgave og hvordan den har implementert PBAC. Oppgaven blir sammenlignet med hva kandidaten gjorde i prosjektet.

Kapittel 5 Implementasjon av ABAC i ODK Aggregate: Presenterer ODK Aggregate og beskriver hvilke styrker og svakheter tilgangskontrollen har. Videre blir implementering av tilgangskontrollen forklart, og deretter følger en sammenligning av gammel og ny versjon av ODK Aggregate.

Kapittel 6 Proof of Concept: Medical prosjekt: Omhandler implementeringen av et selvlagd prosjekt som bygger på erfaringene fra implementasjonen fra OpenMRS og ODK Aggregate. Her blir også sikkerheten med ABAC i prosjektene evaluert med OWASP.

Kapittel 7 Erfaringer, videre arbeid og konklusjon: Beskriver erfaringer med implementasjonene, videre arbeid og konkluderer masteroppgaven.

2 Utvalgte tilgangskontrollmodeller

Det finnes per dags dato flere ulike tilgangskontroller. Mange er definerte og har en klar standard, andre er mer diffuse og vanskelige å tolke. Tilgangskontrollene implementeres ulikt i hvert system og det er stor variasjon om man følger de etablerte standardene.

2.1 Hva er tilgangskontroll?

Tilgangskontroll bygger på fire prosesser[27]:

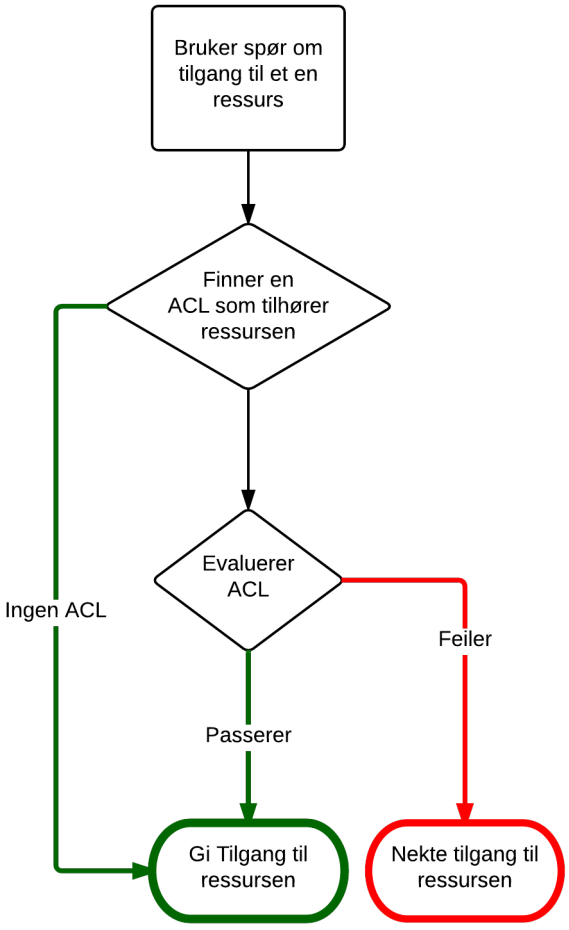
1. **Identifisering** av personer eller applikasjoner.
2. **Autentisering**: en person eller applikasjon må bevise hvem de er.
3. **Autorisering**: personen eller applikasjon må ha rettighetene som ressursen krever.
4. **Tilgangsavgjørelse** er en kombinasjon av de andre tre prosessene for å avgjøre om forespørselen er godkjent.

2.2 Access Control List (ACL)

ACL er den eldste og mest grunnleggende form for tilgangskontroll. Den blir brukt i systemer for å begrense tilgang til filer og data. ACL ble tidlig tatt i bruk i Uniplexed

Information and Computing System (UNIX) systemer. ACL er for det meste assosiert med operativsystemer, men den brukes også i databasesystemer. Et eksempel er at et dokument kan ha en ACL som sier hvem som har tilgang. Strukturen til ACL gjør at det ikke kreves mye underliggende arkitektur [31].

Figur 2.1 viser hvordan en avgjør om en bruker får tilgang til en ressurs. Først spør brukeren om tilgang, også letes det etter en ACL som passer til ressursen som skal aksesseres. Hvis det ikke finnes en ACL vil brukeren få tilgang med en gang. Hvis det finnes en ACL vil den bli evaluert. Hvis brukeren ikke har tilgang blir ressursen ikke gitt til brukeren og omvendt hvis vedkommende får tilgang.



Figur 2.1: Tilgangskontroll flyten til ACL.

2.2.1 Fordeler og ulemper med ACL

Fordeler med ACL er at man kan ha skreddersydd tilgang til hver ressurs. Prinsippet til ACL er enkelt, hver ressurs som skal aksesseres har en assosiert ACL som sier hvem som har tilgang og hva de kan gjøre[31].

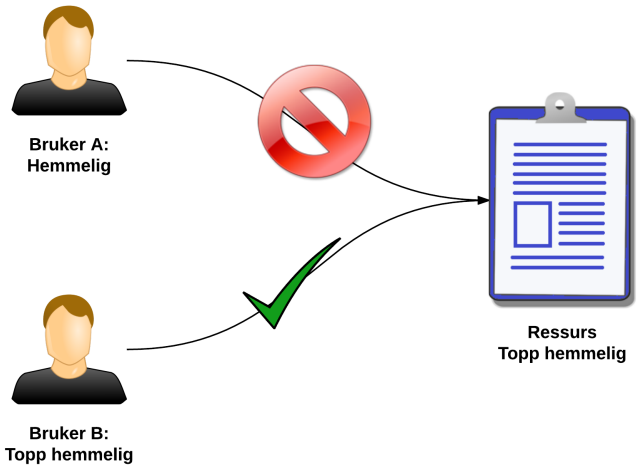
ACL er en utbredt tilgangskontroll men den har sine begrensinger. ACL for en gitt fil må sjekkes hver gang en ressurs blir aksessert. Dette er en ineffektiv og en ressurskrevende måte å gi tilgang til ressurser.

I en bedriftsløsning kan en ACL bli vanskelig å behandle, det kan være mange brukere og hver bruker trenger ulik tilgang til ressursene. Håndtering av alle ACLer tidkrevende og det oppstår rom for feilbehandling av tilganger.

2.3 Mandatory Access Control (MAC)

MAC blir beskrevet som Bell-La Padula modellen [32]. Den består av et sett med klassifiseringer eks: Topp-Hemmelig, Hemmelig, Konfidensielt og Klassifisert. Denne listen er sortert, og det finnes i tillegg et undersett med kategorier som blir kalt sikkerhetsnivåer. Sikkerhetsnivåer er delvis sortert og fungerer som et gitter. Alle dokumenter som skal være sikret får en etikett som sier hvilke klassifikasjon man trenger og hvilket sikkerhetsnivå det skal ha. MAC hadde sin glanstid på 70–90 tallet.

Figur 2.2 viser Bruker A og Bruker B, der Bruker A har tilgang til å se Hemmelig, og Bruker B har tilgang til Topp Hemmelig. Ressursen har stempelet Topp Hemmelig og vil bare kunne vises til brukere med samme stempel, i dette eksempelet vil ikke Bruker A få tilgang til ressursen.



Figur 2.2: Tilgangskontroll flyten til MAC.

2.3.1 Fordeler og Ulemper med MAC

En fordel med MAC er at det har veldefinerte sikkerhetsregler. MAC benytter seg av en enkel sikkerhetsstruktur der en person bare kan lese et objekt hvis han har større eller lik sikkerhetsklassifiseringsnivå.

Ulempe med MAC er at taktiske klassifikasjoner plasserer alle objekter i utvalget klassifikasjoner. Det er vanskelig å gi skreddersydd tilgang til individer og objekter. Det er lett å gi for mye tilgang til systemet.

2.4 Discretionary Access Control (DAC)

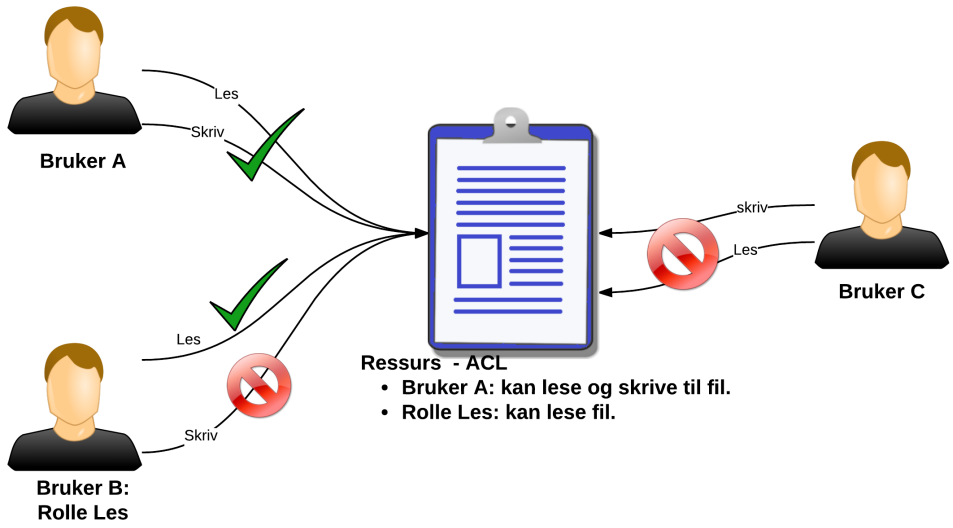
DAC kontrollerer tilgang til ressurser ved å begrense tilgangen basert på identitet til subjekter og grupper. Et subjekt kan gi sine tillatelser indirekte til et annet subjekt hvis det ikke blir begrenset av MAC. Programmer som jobber på vegne av brukeren har de samme tilgangene som brukeren. Tilgangen baseres på tilgangsmatriser. DAC er mye brukt i dagens operativsystemer, databasesystemer og andre informasjonssystemer. I tilgangsmatrisen får hvert subjekt en rad, og hver ressurs en kolonne. Innholdet består av de tillatte handlingene et subjekt kan utføre på ressursene [33]. Tabell 2.1 viser at John Doe har full tilgang til Fil 1, men ikke til Fil 2 (r=read, w=write, x=delete). ACLen sier hvilke brukere som er i systemet og hvilke tilgang de har til filen. Tilgangsmatrisen illustrere

hvordan det hele henger sammen.

Subjekt / Objekt	Fil 1	Fil2
John Doe	rwx	r-x
Jane Doe	r–	rw-

Tabell 2.1: Tilgangsmatrise.

Figur 2.3 viser hvordan en ressurs er beskyttet. Ressursen benytter seg av en ACL som sier hvilke brukere som har tilgang og hvilke handlinger de kan gjøre. ACL kan også gi tilgang til roller. Bruker A er nevnt i ACL med navn og har full tilgang, Bruker B er ikke nevnt med navn, men har rollen Les. Rolle Les gir tilgang til å lese filen, men ikke skrive til den. Bruker C er ikke nevnt i ACL og har ikke rollen Les, og får dermed ikke tilgang til ressursen.



Figur 2.3: Tilgangskontroll flyten til DAC.

2.4.1 Fordeler og ulemper med DAC

En fordel er at DAC kan gi skreddersydd tilgang til ressurser. I motsetning til MAC kan DAC styre tilgangen og rettighetene selv på hvert objekt.

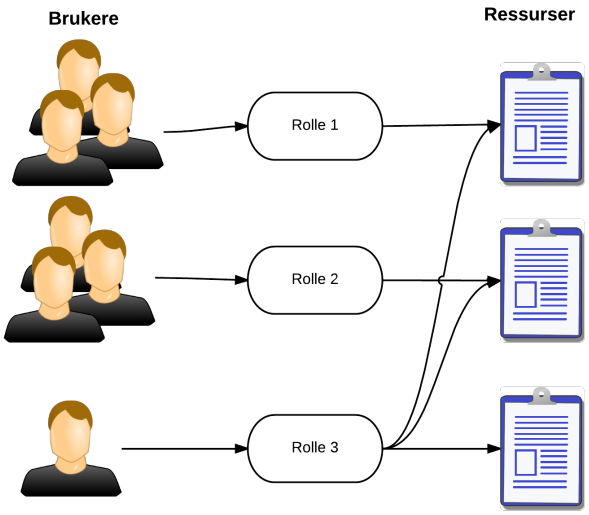
Ulemper med DAC er at den er veldig svak mot trojanske hester[34]. Trojanske hester er programmer som ser ut som vanlige programmer men gjør andre ting i bakgrunnen

med rettighetene brukeren har. For å kjempe imot den trojanske hesten introduserte de MAC for å få konfidensielle nivåer på ressursene, slik at en trojaner ikke kunne få tilgang til mer en det brukeren hadde tilgang til. Tilgangsmatrisen kan bli svært tidkrevende å vedlikeholde etter hvert som den vokser.

2.5 Role Based Access Control (RBAC)

I 1992 ble RBAC presentert som en tilgangskontroll av Ferraiolo og Kuhn. Artikkelen de skrev var et forslag til et alternativ til de mest utbredte tilgangskontrollene MAC og DAC [35]. Rollebasert tilgangskontroll er et begrep som var i bruk i systemer fra 1970-tallet, men det ble ikke en National Institute of Standards and Technology (NIST) standard før i 2000. Rollebasert tilgangskontroll er i dag den mest utbredde tilgangskontrollen. De fleste programvareleverandørene leverer løsninger som støtter rollebasert tilgangskontroll.

Figur 2.4 viser hvordan ulike grupper med brukere har ulike roller og rollene er knyttet til ressursen. Rolle 3 har tilgang til alle ressurser. Med rollebasert tilgang skilles det ikke på lese og skrive rettigheter, hvis en bruker har tilgang til ressursen kan han gjør alt med den.



Figur 2.4: Tilgangskontroll flyten til RBAC.

2.5.1 RBAC prinsipper og konsepter

Et prinsipp kan være en regel, og prinsippet om minst tilgang går ut på at en bruker av systemet ikke skal ha tilgang til mer enn det han har tillatelse til å gjøre. En av oppgavene til en administrator er å passe på at rollene ikke gir for mye tilgang, det er et viktig prinsipp i RBAC [36].

En plikt er en oppgave. Separasjon av plikter er separasjon av oppgaver, det vil si at en rolle skal bare ha en oppgave. Prinsippet brukes for å realisere separasjon av plikter. Prinsippet skal forhindre svindel og feil, dette oppnår en ved å spre pliktene slik at en rolle har en handling [37]

En annen viktig funksjon er rollehierarki. Et rollehierarki er når roller arver tillatelser som er gitt til en rolle som ligger høyere i hierarkiet, dette kalles tillatelseaspekter. En bruker som er tildelt en bestemt rolle kan også aktivere underoller i hierarkiet, dette kalles aktiveringaspekt av rollehierarkiet [38]

2.5.2 RBAC modellutvikling

NIST har definert fire ulike RBAC modeller, hver av dem har sine styrker og svakheter.

- $RBAC_0$ er den enkleste modellen. Den baserer seg på å ha minste privilegier og separerer plikter. Dette gjennomføres ved å gi tillatelser og den benyttes ikke på hierarki. Rollene og funksjoner blir gitt direkte til en bruker [39].

- $RBAC_1$ baserer seg på $RBAC_0$ og introduserer hierarki. Dette ble utviklet for å kunne gi en naturlig fordeling av ansvaret i en organisasjon [39].

- I $RBAC_2$ ble det introdusert begrensninger. Dette introduserer en rekke nye funksjoner i RBAC miljøet. $RBAC_2$ tar ikke i bruk hierarki, men begrensningen kan fremtvinge bruken av policyer. En policy kan være at en bruker bare kan ha en rolle og dermed forsikre oss om at det er bare en administrator i systemet. Det kan også brukes til å regulere tilgang til ressurser for å forsikre seg om at kriteriene er oppfylt [39].

- $RBAC_3$ er den mest komplette implementasjonen av RBAC-modellene. Den kombinerer både hierarki og begrensinger.

2.5.3 Fordeler og ulemper med RBAC

Fordeler med RBAC er at det er enkelt å administrere roller og rollen kan deles med flere brukere, samt gi tilgang til flere ressurser.

RBAC kan skalere hvis organisasjonens sikkerhetspolitikk følger et godt oppsett og er dokumentert, samt hvis hierarkiet er designet slik at det kan legges til flere roller som arver.

Roller kan bli brukt som klassifisering. Gjennom hierarki, rettighetsarving og begrensninger kan det argumenters for at RBAC har nivåer i sikkerhetsmodellen. Muligheten til å lage nye roller, hierarki og begrensninger kan lage skreddersydde roller.

Ulemper med RBAC er at rollene er statiske. Rollebasert tilgangskontroll tar ikke hensyn til kontekst.

Skreddersydd tilgang kan resultere i noe som kalles en rolleksplosjon, dvs. at antall roller vokser enormt raskt når rollen brukes til skreddersydd tilgang. Når antall roller økes, økes også kompleksiteten og tiden som brukes for å vedlikeholde rollene.

Når tilgangskontrollen er dårlig dokumentert er det vanskelig å skalere RBAC. Hvis det ikke finnes en klar plan / visjon i organisasjonen om hvordan sikkerhetspolitikken skal håndteres vil det hindre skalering av systemet.

Svak eller lite administrering av rollen kan medføre at brukere får roller med for mye tilgang.

2.6 Attribute Based Access Control (ABAC)

ABAC er en utvikling fra rollebasert tilgangskontroll. ABAC er en logisk tilgangskontroll hvor tilgangsavgjørelsen er basert på attributter assosiert med forespørsler, miljø, ressurser, kontekst og handlinger. I ABAC blir attributtene validert på et lokalt nivå og kan variere for hver implementasjon.

2.6.1 Attributter i ABAC

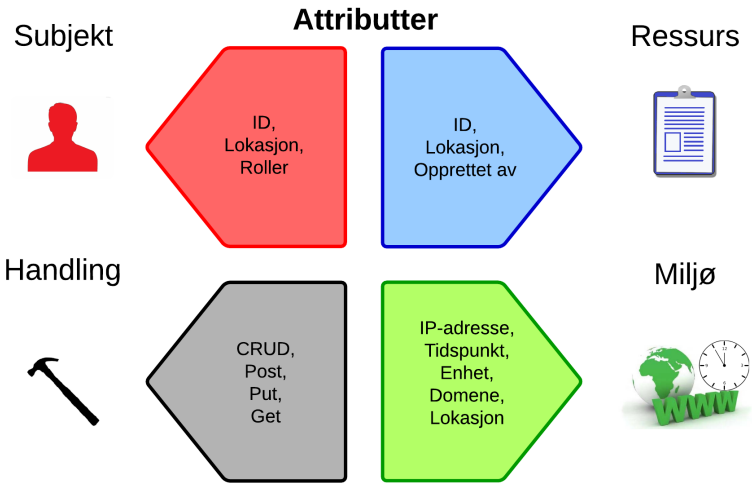
Attributtene har en verdi som brukes for å bestemme adgang til en ressurs. Attributtene er et sett med atomiske verdier, dvs. verdier som ikke kan deles opp i flere verdier. Figur 2.5 viser noen av attributtene som ABAC kan benytte seg av.

En bruker blir assosiert med et sett med attributter. Attributtene blir tildelt brukeren av en administrator, og representerer egenskaper til brukeren som f.eks. navn, roller, kjønn, jobbstilling, klarering og lignende. Denne typen attributter kategoriseres som *subjektattributter*.

Ressurser er det brukeren prøver å få tilgang til og de kan beskyttes av *ressursattributter*. Ressursen kan bli opprettet av en bruker og han kan sette føring på hvilke attributter som skal assosieres med ressursen [29].

Handlingsattributter brukes for å avgjøre hvilken handling som skal utføres. Dette gjør det enklere å skille på kritiske handlinger som å slette eller modifisere data.

ABAC kan også benytte seg av *miljøattributter*, dvs. attributter som klokkeslett, domene, lokasjon, IP-adresse o.l. Miljøattributtene brukes for å kunne ta enda mer skreddersydde avgjørelser. Attributtene kan være med på å plassere brukeren hvor han befinner seg, hvilken enhet (mobil, laptop etc.) han bruker, og når han prøver å få tilgang til ressursene.



Figur 2.5: Noen av attributtene som ABAC kan bruke[1].

2.6.2 Komponenter

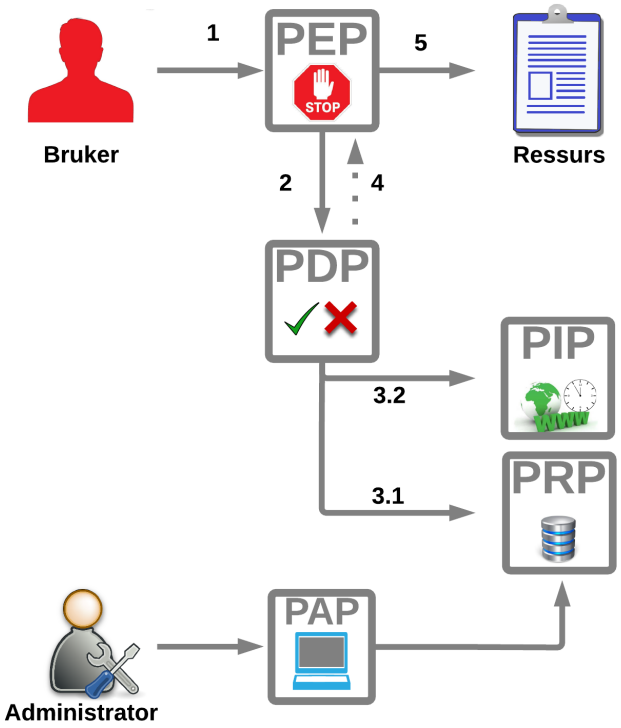
ABAC består av komponenter. Listen forklarer de ulike komponentene:

- Policy Decision Point (PDP) evaluerer aktuelle policyer og sender tilbake autorisasjonsavgjørelser.
- Policy Information Point (PIP) er kilden til attributtene.
- Policy Enforcment Point (PEP) er plassen som utfører tilgangskontroll ved å spørre PDP om en autorisasjonsavgjørelse og deretter eksekverer tilgangen i henhold til avgjørelsen.
- Policy Administation Point (PAP) brukes til å administrere policyene som skal brukes i systemet.
- Policy Repository Point (PRP) er plassen policyene lagres.

2.6.3 Tilgangsbestemmelsen

Figur 2.6 viser flyten i tilgangsbestemmelsen for ABAC. Den samme tilgangsbestemmelsen brukes også i Policy-Based Access Control (PBAC) og til dels i Risk-Adaptable Access Control (RAdAC)[40].

1. Brukeren sender inn en forespørsel om tilgang til en ressurs. PEP avgjør om brukeren har tillatelse til å aksessere objektet på bakgrunn av informasjonen den får fra Policy Decision Point (PDP).
2. PDP avgjør om brukeren har tilgang til ressursen.
3. Attributt og policy innsamling
   3.1. PDP henter policyen fra PRP.
   3.2. PIP samler inn informasjon fra informasjonskilder f.eks. databaser, klokkeslett etc.
4. PDP sender svar til PEP i form av tillat eller avslått.
5. PEP behandler svar fra PDP og gir tilgang til ressursen på bakgrunn av responsen.



Figur 2.6: XACML flyten[2].

2.6.4 Fordeler og ulemper med ABAC

ABAC sine autorisasjonsregler er intuitive i hvordan de gir tilgang til en ressurs. De er fleksible, nesten alt kan uttrykkes som en regel så lenge de nødvendige dataene er tilgjengelige. Reglene kan defineres som setninger, noe som gjør dem enkle å forstå.

Evaluering av regler skjer under kjøring, og ABAC kan dra nytte av kontekst, lokasjon og klokkeslett. Reglene trenger mindre vedlikehold og kan gi en skreddersydd tilgang.

Attributtene til ABAC trenger ikke å være styrt av autorisasjonssystemet, de kan komme fra tredjeparter f.eks. databaser.

ABAC er tilstandsløst, det vil si at en avgjørelse baseres ikke på hva som har skjedd før, men tar avgjørelser på bakgrunn av hvilke attributter som blir presentert og er tilgjengelig under kjøring.

En av de store utfordringene med ABAC er evalueringen av regler under kjøring. Når det er dårlig forbindelse eller problemer med å hente attributter fra en tredjepart er det vanskelig å gi tilgang.

Tillit i ABAC er vanskelig å oppnå når autorisasjonen brukes av flere systemer, og pålideligheten til attributtene kan være vanskelig å sjekke.

ABAC kan lede til attributteksplosjon slik som det ble nevnt med rollene. I et system med N antall attributter vil den verste kjøretiden bli eksponentiell 2^N[41].

2.6.5 Policy-Based Access Control (PBAC)

PBAC bygger på de samme prinsipper og komponenter som ABAC, men er mer policybelagt. Tanken bak PBAC er å gi systemet en uniform tilgangskontroll på tvers av organisasjonen. Den har samme skreddersydde tilgangskontroll som ABAC og bruker de samme attributtene, ressursene, miljøer, kontekster og forespørsler. Som nevnt i ABAC blir attributtene validert på et lokalt nivå. PBAC vil gi organisasjonen en større helhet og validerer attributtene for alle organisasjoner som er med i samme system. Dette vil gjelde ressurser som møter de sensitive kriteriene i organisasjonen. PBAC er som nevnt tidligere en utvikling av ABAC, men den er mer komplisert. Attributtene må vedlikeholdes på tvers av organisasjonen. Her er det viktig at organisasjonen bruker en autoritær attributtkilde slik at organisasjonene kan bruke de samme attributtene [31].

Fordelene og ulempene til PBAC arves av ABAC det vil si at PBAC har de samme fordeler og ulemper som ABAC.

2.7 Andre typer tilgangskontroller

Det finnes nyere tilgangskontroller som enda ikke har stor utbredelse, men de er ikke relevante for oppgaven. Tilgangskontroller som f.eks. Relationship Based Access Control (ReBAC)[42] og RAdAC, blir nevnt for fullstendighetens skyld.

2.8 Oppsummering av tilgangskontroller

Tilgangskontrollene ACL, MAC og DAC er eldre tilgangskontroller som ikke brukes i noe større grad i webutvikling. RBAC er den desidert mest utbredte tilgangskontrollen, og det er denne tilgangskontrollen ODK Aggregate og OpenMRS benytter seg av.

ABAC og PBAC er mer fleksible tilgangskontroller, men de er ikke veldig utbredt. Etterspørselen etter en skreddersydd tilgangskontroll er blitt større, og det klarer ikke RBAC å imøtekomme uten å endre seg mer mot ABAC og PBAC.

Fokuset i oppgaven vil ligge på ABAC og utskiftingen av RBAC i eksisterende systemer. Tabellen 2.2 oppsummerer de mest interessante forskjellene mellom tilgangskontrollene. Kandidaten sammenligner adoptering, funksjonalitet, implementasjon, vedlikehold, administrasjon og tilbakekalling.

	RBAC	**ABAC**
Adoptering	Utbredt, velkjent.	Relativt ny mekanisme, mange systemer benytter seg av ABAC lignende funksjoner.
Funksjonalitet	Imøtekommer ikke miljøvariabler, eller gjør kontekstbaserte avgjørelser.	Veldig fleksibelt, kontekstbevist og miljøbevisst. Kan også ta avgjørelser under kjøring.
Implementasjon	Basisfunksjoner av RBAC er som regel bygget inn i rammeverkene. Det er lett å implementere. Komplekse avgjørelsespunkter må kodes.	Krever implementasjon av PIP, PEP, PAP og PDP. Det finnes noe kommersielle verktøy men de er i en tidlig fase.
Vedlikehold	Krever kodeendringer.	Veldig fleksibelt, kan endre policyer under kjøring
Administrasjon	Krever lite administrering når systemet er oppe og kjører	Kan være vanskelig å administrere.
Tilbakekalling	Vanskelig å ta vekk roller når brukeren er innlogget.	Enkelt å endre tilgang til brukerne i under kjøring, pga. kallene er tilstandsløse.

Tabell 2.2: RBAC vs. ABAC[6].

3 Implementasjon av ABAC og relevante teknologier

Kapittelet belyser overveielser en utvikler må ta når det skal implementeres ABAC i tredjepartssystemer, og gir en grundig innføring i XACML sine hovedprinsipper og begreper. I tillegg vil det bli gitt en innføring i WSO_2 IS.

3.1 Krav til implementasjon av ABAC

Implementasjon av ABAC kan gjøres ved å implementere følgende tre elementer: policy-modellering, attributtmodellering og applikasjonsintegrering.

3.1.1 Policy-modellering

Policy-modellering er å definere hvilke forretningsregler som gjelder for applikasjonen. Reglene kan defineres som vanlige setninger som beskriver hvem, hva, når, hvor, hvorfor og hvordan. En regel kan f.eks. være *En doktor kan bare se pasienter som han har undersøkt i normal arbeidstid på kontoret*.

3.1.2 Attributtmodellering

Hvilke attributter som skal brukes blir definert i attributtmodelleringsfasen. Attributtene er kontekstuelle og brukes for å kunne ta avgjørelser. Figur 2.5 fra seksjon 2.6.1 i kapittel 2

viser noen av attributtene som kan brukes. Det er ikke begrensinger for hvilke attributter som kan brukes så lenge applikasjonen klarer å sende det i forespørselen.

3.1.3 Applikasjonsintegrering

Applikasjonsintegrering starter med å definere hva som skal beskyttes av ressurser. Deretter må det defineres hvor i applikasjonen autorisasjon skal plasseres. Applikasjonen kan deles inn i tre lag; presentasjonslaget, servicelaget og datalaget.

- Presentasjonslaget er laget som brukerne av systemet får se, det er der informasjonen presenteres.
- Servicelaget er laget som kommuniserer med presentasjon og datalaget. Servicelaget inneholder også logikk.
- Datalaget kommuniserer med databasen.

For å kunne plassere PDP må det kartlegges flaskehalser. Å kartlegge flaskehalser vil si at en må finne steder i koden hvor all trafikk går igjennom. Der flaskehalsene blir definert kan det settes opp en PEP som kan ta avgjørelser om brukeren har tilgang til ressurser. Flaskehalser kan oppstå i alle lag, og det plasseres PEP for hvert lag som skal beskyttes. I sikkerhetssammenheng er flaskehalser ikke negativt, det er steder i koden hvor trafikken går igjennom slik at det enklere kan autoriseres. I ytelsesammenheng er flaskehalser negativt.

For å kunne benytte egendefinerte attributter må det lages individuelle PIPer. Dette for å kunne knytte brukeren opp mot rettighetene den har i systemet slik at PDP kan ta informert avgjørelse.

Når ABAC integreres i et system finnes det valg hvordan det skal sette opp PDP. Plassering av PDP kan ha mye å si for ytelsen på autorisasjonene. Før en skal implementere må en stille seg følgende spørsmål for å få best plassering av PDP i forhold til situasjonen.

- Skal PDP kjøre på egen server?
- Kjører den på samme nettverk?
- Skal den implementeres i applikasjonen?
- Hvor langt (fysisk avstand) fra PDP er PEP og attributtkildene?

3.2 eXtensible Access Control Markup Language (XACML)

XACML ble introdusert i 2003. Den definerer en eXtensible Markup Language (XML) basert tilgangskontroll som har blitt gjort til en standard av OASIS's tekniske komite. XACML validerer attributter som brukeren sender inn, og kombinerer dem med informasjon den har tilgang til. Dette for å kunne ta avgjørelsen om brukeren har lov å se ressursene. XACML tar ikke ansvar for autentiseringen, men den er bare opptatt om brukerne har autorisasjon til å se ressursene [43].

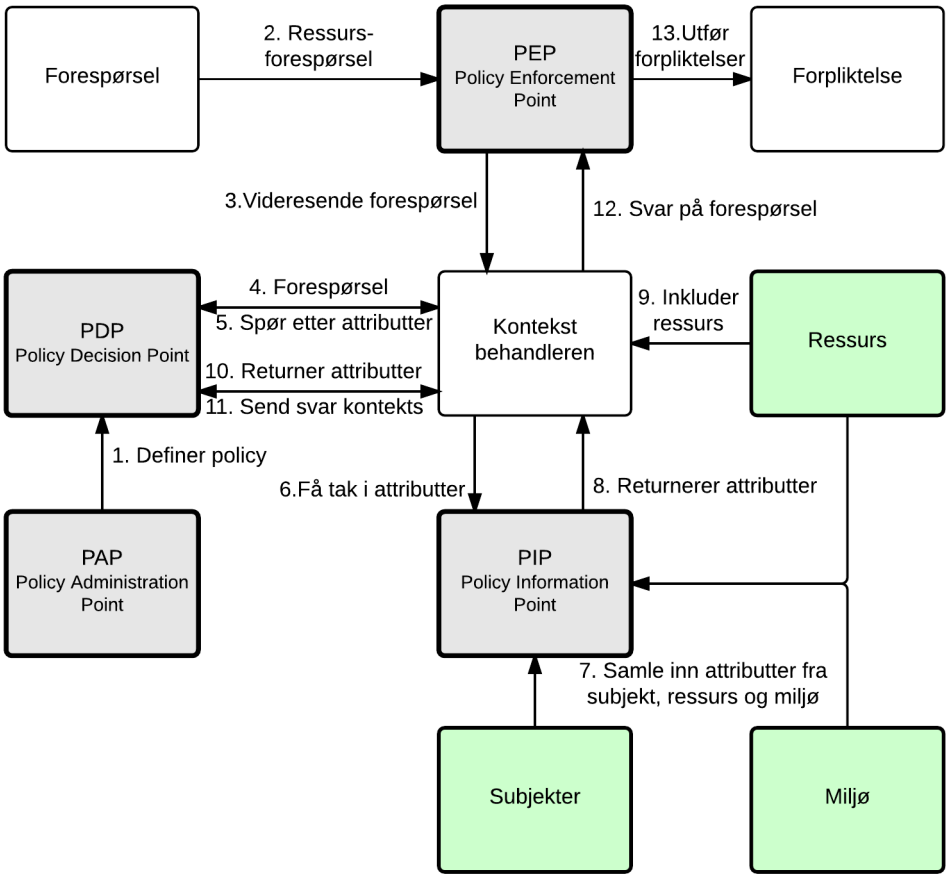
3.2.1 Hvorfor XACML?

I tradisjonelle autoriseringsmetoder er logikken hardkodet og vanskelig å endre etter at systemet er tatt i produksjon. XACML tilbyr at man løsriver autorisasjonen slik at de samme autorisasjonsregler kan brukes på flere applikasjoner.

XACML er dynamisk og den kan bytte ut autorisasjonsregler uten å endre på kildekoden til og med når applikasjonen kjører. Den tar også dynamiske avgjørelser basert på informasjon den har fått fra PIP. XACML kan også brukes til å implementere vanlig ABAC og PBAC tilgangskontroll.

Figur 3.1 viser oversikten til XACML. PAP, PDP, PIP og PEP er farget i grå og er de samme punktene som er med i ABAC. Subjekter, miljø og ressurser er farget i grønt og er attributter som systemet benytter seg av. I korte trekk går flyten slik:

Først blir en policy laget og tilgjengeliggjort. Prosessen starter med at en bruker spør om tilgang til en ressurs. Forespørselen går til PEP og blir sendt til kontekstbehandleren. Deretter går forespørselen til PDP som henter inn ekstra informasjon fra PIP. Når den har fått informasjonen den trenger tar den en avgjørelse. Deretter returner den avgjørelsen til PEP, som eventuelt må utføre forpliktelser. Brukeren får tilgang til ressursen hvis forespørselen ble godkjent.



Figur 3.1: Oversikt over XACML[3].

3.2.2 XACML policy-modell

I påfølgende seksjoner vil en XACML *policy* bli beskrevet. En *policy* er bygget på *regler* og et *policy-sett* er en måte å samle flere *policyer* på. Måten det kommer til å bli forklart på er å starte med en *regel* som inneholder *effekt*, *mål* og *tilstand*. Deretter vil en policy som inneholder *regler*, *råd* og *forpliktelser* bli forklart.

3.2.2.1 Regel: effekt, mål og tilstand

En *regel* starter med et *effekt* attributt. Eksempel 3.1 viser at regelen har en *RuleId* attributt som må være unik, og en `effekt` som kan være `Permit` eller `Deny`.

```
1 <Rule RuleId="get" Effect="Permit">
```

Eksempel 3.1: XACML `effekt`

Et *mål* er et sett med beslutningsforespørsler, som kan identifiseres av en `ressurs`, et `subjekt`, `handlinger` og `miljø`. Et *mål* er knyttet til en *regel*, *policy* eller et *policysett* og brukes til evalueringen. Eksempel 3.2 viser et utsnitt av et *mål* til en regel. Linje 1 viser en `<Target>` tagg som skal inneholde en `<AnyOf>` tagg som trenger minst et positivt treff på en `<AllOf>` tagg. En `<AnyOf>` tagg kan inneholde flere `<AllOf>` tagger, og en `<AllOf>` må inneholde en `<Match>` tagg som inneholder `<AttributeValue>` tagg og en `<AttributeDesignator>` tagg. Linje 4 viser en `<Match>` taggen som har en id som sier hvilken funksjon som skal brukes. I dette tilfellet er den en Regulært utrykk[1]. Det Regulære utrykket på linje 6 skal treffe på alle strenger som starter på `get` eller `filterEncountersByViewPermissions`. `<AttributeValue>` taggen sier hva den skal lete etter, `<AttributeDesignator>` taggen sier hvilken kategori, datatype og hvor den skal lete etter attributtene. `<AttributeValue>` og `<AttributeDesignator>` taggene kommer til å dukke opp igjen og har samme funksjon som ble beskrevet i denne seksjonen.

```
1  <Target>
2   <AnyOf>
3    <AllOf>
4     <Match MatchId="...:string-regexp-match">
5     <AttributeValue "...#string" >
6      ^(get[A-Za-z0-9][ A-Za-z0-9-]*|filterEncountersByViewPermissions)
7      </AttributeValue>
8      <AttributeDesignator  AttributeId="...:resource-id"
9      Category="...:resource" DataType="...#string"
10     MustBePresent="true"></AttributeDesignator>
11    </Match>
12   </AllOf>
13  </AnyOf>
14 </Target>
```

Eksempel 3.2: XACML *mål*.

[1]Et regulært utrykk er en streng som beskriver et sett av strenger, med et mønster som følger syntaksregler[44].

Eksempel 3.3 viser en *tilstand* (eng. condition). Det er et sannhetsutrykk som evaluerer attributter. En `<Condition>` tagg består av `<Apply>` tagg som skal inneholde en `<Function>`, `<AttributeValue>` og `<AttributeDesignator>` tagger. `<AttributeDesignator>` taggen tar imot et egendefinert attributt og ikke et XACML definert attributt. Egendefinerte attributter som lages trenger en PIP for å finne attributtene.

```
<Condition>
 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
  <Function
   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
  </Function>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
  (Clinician|Provider)</AttributeValue>
  <AttributeDesignator
  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
  AttributeId="http://openmrs.com/id/role"
  DataType="http://www.w3.org/2001/XMLSchema#string"
  MustBePresent="false"></AttributeDesignator>
 </Apply>
</Condition>
```

Eksempel 3.3: XACML *tilstand*.

En komplett *regel* består av en *effekt*, et *mål* og en *tilstand*. Eksempel 3.4 viser et utsnitt hvordan *effekt*, *mål* og *tilstand* blir plassert i en *regel*. En *regel* kan ha *råd* og *forpliktelse*, men det ikke påkrevd. *Råd* og *forpliktelser* vil bli forklart i påfølgende seksjon.

```
<Rule RuleId="get" Effect="Permit">
 <Target> ... </Target>
 <Condition> ... </Condition>
</Rule>
```

Eksempel 3.4: XACML *regel*.

3.2.2.2 Policy-sett: policy, regler, råd og forpliktelser

Et *råd* er ekstra informasjon i en *regel*, *policy* og et *policy-sett*, som blir sent til et PEP. Det er opp til PEP om rådet skal bli behandlet. Eksempel 3.5 viser hvordan et råd kan være utformet. Linje 2 i koden sier at rådet bare blir kjørt hvis en regel får `Permit` resultat.

```
<AdviceExpressions>
 <AdviceExpression AdviceId="app-enable" AppliesTo="Permit">
  <AttributeAssignmentExpression AttributeId="app-enable">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
   AppEnable</AttributeValue>
  </AttributeAssignmentExpression>
  </AdviceExpression>
</AdviceExpressions>
```

Eksempel 3.5: XACML *råd*[7].

Forpliktelser (eng. obligations) er en operasjon spesifisert i en *regel*, *policy* og et *policysett* som skal behandles av PEP i samhandling med autorisasjonsavgjørelsen. En *forpliktelse* kan være at en bruker prøver å få tilgang til et dokument. Responsen kan da være å sende en mail til eieren av dokumentet og si at brukeren har lest dokumentet. Eksempel 3.6 viser at et en *forpliktelse* inneholder flere `<AttributeAssigment>` tagger, en `<AttributeAssigment>` tagg inneholder en `<AttributeSelector>` tagg. `<AttributeAssigment>` taggen sier hva som skal gjøres eller bli utført, og `<AttributeSelector>` taggen henter informasjonen som den trenger. *Råd* og *forpliktelse* er valgfritt, *råd* kan ignoreres av PEP, men det kan ikke *forpliktelser*. *Forpliktelser* blir ikke mye brukt, dette er fordi det er ressurskrevende og vil kunne forsinke avgjørelsesprosessen.

```
<Obligation ObligationId="...:obligation:email" FulfillOn="Permit">
 <AttributeAssignment
 AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:mailto"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeSelector RequestContextPath="/email/path"
  DataType="http://www.w3.org/2001/XMLSchema#string"/>
 </AttributeAssignment> <AttributeAssignment
  AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
  DataType="http://www.w3.org/2001/XMLSchema#string">
   Your record has been accessed by:
  </AttributeAssignment>  <AttributeAssignment
  AttributeId="urn:oasis:names:tc:xacml:2.0:example:attribute:text"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <SubjectAttributeDesignator
  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
  DataType="http://www.w3.org/2001/XMLSchema#string"/>
  </AttributeAssignment>
</Obligation>
```

Eksempel 3.6: XACML *forpliktelse*[8].

En *policy* er en kombinasjon av alle elementer som er blitt beskrevet. Eksempel 3.7 viser på linje 1 at `<policy>` taggen inneholder informasjon om *policyen*, informasjonen sier hvilket skjema den skal bruke for å validere hvilken XACML versjon som brukes. *Policyen* er også identifisert med en unik `id`, `versjonsnummer` og en `kombinasjonsalgoritme`. Linje 5 i eksempelet viser at den inneholder et *mål*, *målet* er likt som *målet* beskrevet i seksjon 3.2.2.1. En *policy* kan inneholde en eller flere *regler*. Dette er den samme *reglen* som ble beskrevet i seksjon 3.2.2.1. En *policy* kan også inneholde *råd* og *forpliktelser*, men det er som nevnt valgfritt og blir ikke mye brukt.

```
1  <Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
2  xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"  PolicyId="Openmrs"
3  RuleCombiningAlgId=
4      "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
5  Version="1.0">
6   <Target>...</Target>
7   <Rule RuleId="get" Effect="Permit">...</Rule>
8   <Rule RuleId="view" Effect="Permit">...</Rule>
9   <Rule RuleId="last-rule" Effect="Deny">...</Rule>
10  <AdviceExpressions>...</AdviceExpressions>
11  <Obligation ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
12  FulfillOn="Permit">...</Obligation>
13 </Policy>
```

Eksempel 3.7: XACML *policy*.

Flere *policyer* kan kombineres i et *policy-sett*. Eksempel 3.8 er en kombinasjon av alt som har blitt vist til nå.

```
1  <PolicySet  PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
2  Version="1.0" PolicyCombiningAlgId=
3   "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
4   <Description> Example policy set. </Description>
5   <Target>...</Target>
6   <PolicyIdReference> urn:oasis:names:tc:xacml:3.0:example:policyid:3
7   </PolicyIdReference>
8   <Policy PolicyId="xx0">...</Policy>
9   <Policy PolicyId="xx1">...</Policy>
10  <Policy PolicyId="xx3">...</Policy>
11  <Policy PolicyId="xx4">...</Policy>
12  <AdviceExpressions>...</AdviceExpressions>
13  <Obligation ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
14  FulfillOn="Permit">...</Obligation>
15 </PolicySet>
```

Eksempel 3.8: XACML *policy-sett*.

Policy-sett og *policy* benytter seg av kombinasjonsalgoritmer. Algoritmene beskriver strategier for hvordan policyen eller regelen skal kunne godkjenne *forespørsler*. Eksempel 3.8 viser på linje 1-3 at den bruker en `policy-kombinasjonsalgoritme`, og Eksempel 3.7 bruker en `regelkombinasjonsalgoritme`[3].

3.2.3 XACML forespørsel og respons

En *forespørsel* er bygget opp på attributter. Det er attributtene som skal kunne passe til en *policy*. XACML *forespørselen* kan se ut som Eksempel 3.9. En `<Request>` tagg består av flere `<Attributes>` tagger som inneholder `<Attribute>` og `<AttributeValue>` tagger. `<Attributes>` taggen sier hvilken kategori de tilhører; *ressurs*, *subjekt*, *handling* eller *miljø*. `<Attribute>` taggen sier hvilken id som det er og `<AttributeValue>` taggen er verdien en PEP sender inn, det vil si attributter fra en bruker. *Handlingen* som skal utføres er `Get Patients`, *subjekt* iden er `1`. *Miljøet* er `openmrs.com` og *ressursen* er `patient`.

```
1  <Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
2  CombinedDecision="false" ReturnPolicyIdList="false">
3  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
4   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
5    IncludeInResult="false">
6   <AttributeValue DataType="...#string">Get Patients</AttributeValue>
7   </Attribute> </Attributes>
8  <Attributes
9   Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
10 <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
11 IncludeInResult="false">
12 <AttributeValue DataType="...#string">1</AttributeValue>
13 </Attribute> </Attributes>
14 <Attributes
15     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
16 <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:environment:environment-id"
17  IncludeInResult="false">
18  <AttributeValue DataType="...#string">openmrs.com</AttributeValue>
19  </Attribute></Attributes>
20 <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
21  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
22  IncludeInResult="false">
23  <AttributeValue DataType="...#string">patient</AttributeValue>
24  </Attribute>
25  </Attributes>
26 </Request>
```

Eksempel 3.9: XACML *forespørsel*.

En *respons* kommer etter at *forespørselen* har vært igjennom XACML prosessen ref. Figur 3.1. Det er svaret som skal behandles av PEP. Eksempel 3.10 viser hvordan en godtatt respons kan se ut og linje 3 viser avgjørelsen.

```
1 <Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
2  <Result>
3  <Decision>Permit</Decision>
4   <Status><StatusCode Value="...:status:ok"/></Status>
5  </Result>
6 </Response>
```

Eksempel 3.10: XACML *respons*.

3.3 Avgjørelsen om hvilken PDP som skal benyttes

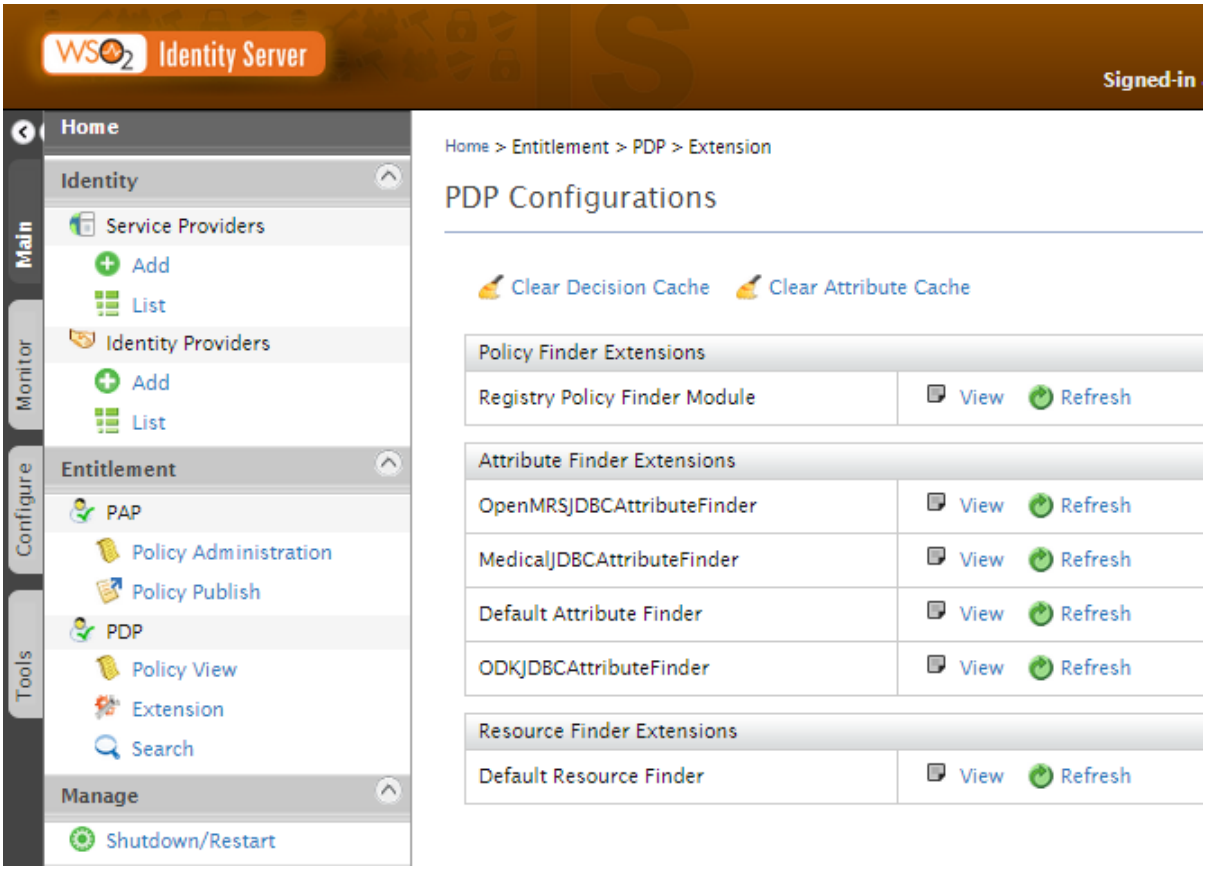
For å kunne avgjøre hvilken PDP som passet best til prosjektet måtte følgende krav oppfylles. Den skal kjøre på egen instans og være et åpent kildekodeprosjekt. Det må også være mulighet for å kunne lage PIP som kan integreres eller plugges inn i serveren. Den må også fungere som et PAP som kan validere og teste XACML 3.0 policyer. Kravene er satt med bakgrunn i kandidatens erfaring og forståelse av hva som vil skille et godt PDP fra et annet. Flere prosjekter ble vurdert slik som Holistic Enterprise-Ready Application Security Architecture Framework (HERAS^{AF})[45], SunXACML[46], Enterprise Java XACML[47]. Dette er prosjekter som ikke innfridde kravene og valget falt dermed på WSO_2 IS som oppfylte kravene.

3.3.1 WSO_2 IS

WSO_2 IS er en åpen kildekodeserver som kan brukes som et PDP. Serveren støtter den nyeste versjon av XACML 3.0, og det er mulighet for å lage egne PIP. Den er godt dokumentert og kommer med gode eksempler. WSO_2 IS miljøet er aktivt og jobber med å forbedre løsningen. Den har en rask kommunikasjon mellom PDP og PEP med Apache Thrift [48] som er en binær kommunikasjonsprotokoll. WSO_2 IS støtter *caching* av XACML forespørselen noe som øker ytelsen. Den har også muligheten for å kunne validere og teste XACML policyer og forespørsler. WSO_2 IS gir funksjonalitet som sikker kommunikasjon via Hypertext Transfer Protocol Secure (HTTPS) og grensesnitt som kan brukes til å konfigurere XACML.

3.3.2 Funksjonaliteten til WSO_2 IS i sammenheng med XACML

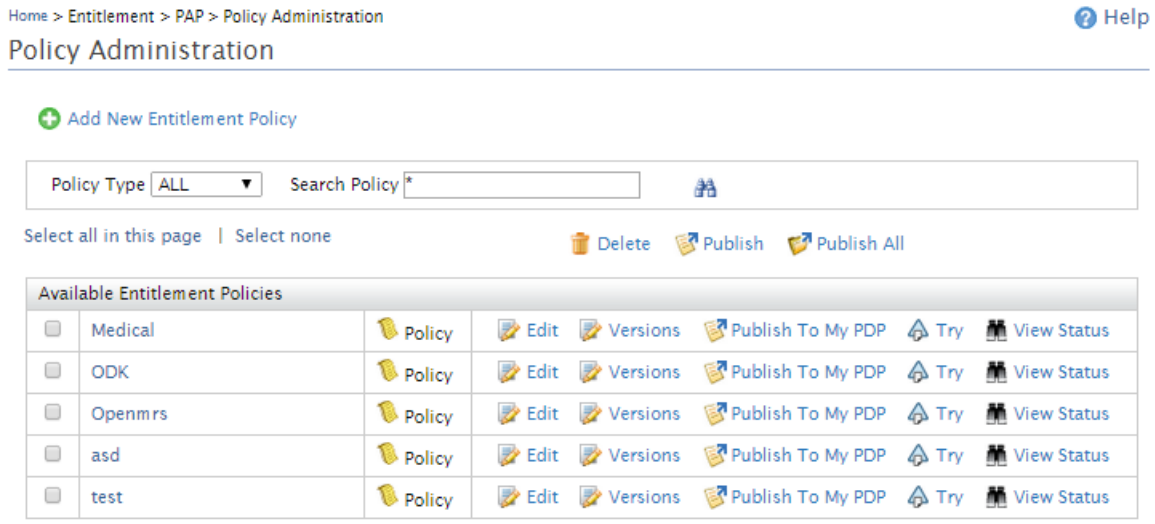
WSO_2 IS vil fungere som PDP, PIP,PRP og PAP. Figur 3.2 viser hvordan `Attribute Finder Extensions` også kjent som PIP inneholder OpenMRS, ODK Aggregate sine PIP. `View` knappen viser hvilke attributter som er registrert i utvidelsen. Attributtene som er registrert kan en policy bruke for å finne mer informasjon om brukeren.



Figur 3.2: PDP konfigurasjon med PIP som attributtfinner.

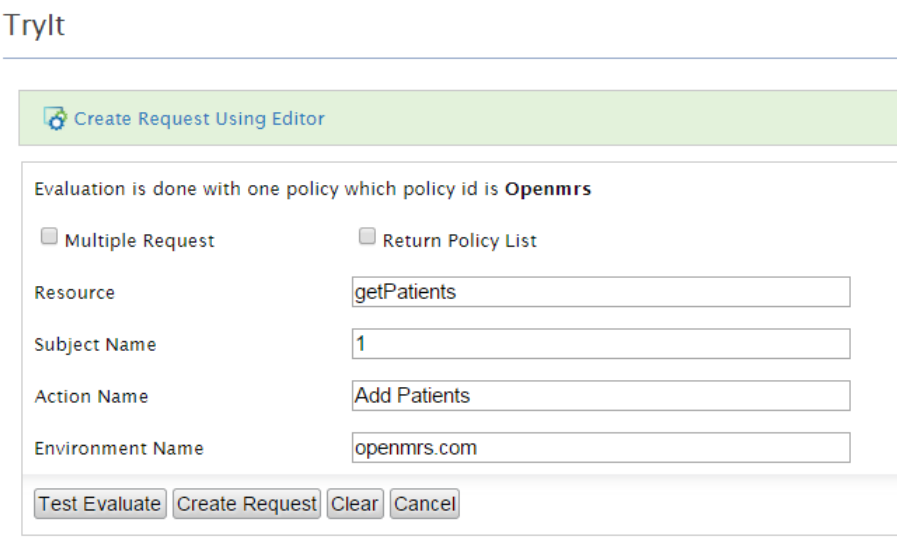
Figur 3.3 viser en oversikt over XACML policyer som den kan bruke.

- `Policy` lenken viser hele policyen.
- `Edit` lenken gjør det mulig å endre policyen.
- `Versions` viser ulike versjoner av policyen hvis det finnes.
- `Publish My PDP` lenken gjør policyen aktiv.
- `Try` lenken gjør det mulig å teste policyen.
- `View Status` lenken viser status på policyen, om den er aktiv eller ikke.



Figur 3.3: PAP med policyer.

Når en bruker trykker på `Try` lenken fra Figur 3.3 blir han sendt til en `TryIt` funksjon som gir en grafisk representasjon av en forespørsel slik som vist på Figur 3.4. Det som blir vist på figuren er det samme som Eksempel 3.9.

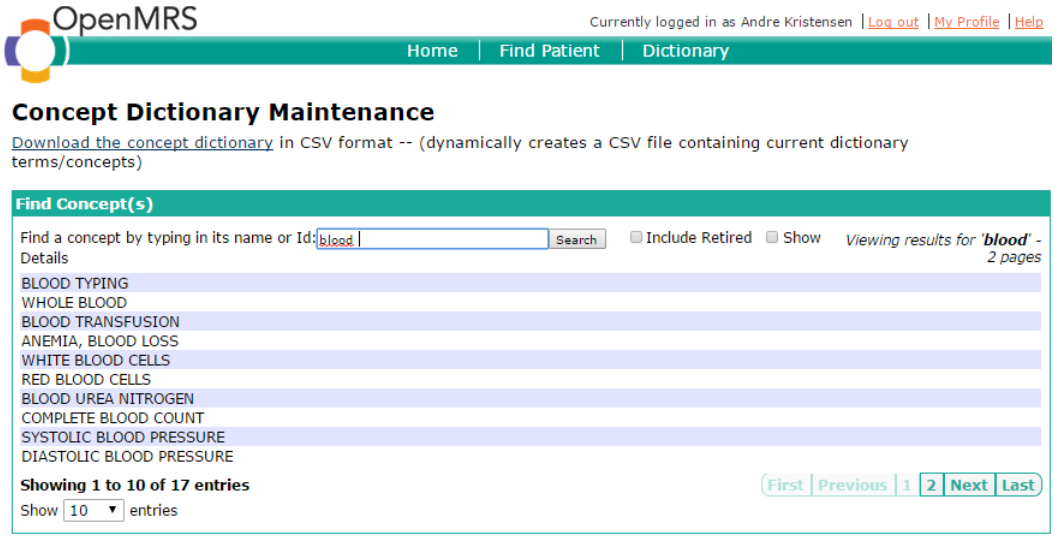


Figur 3.4: `TryIt` for testing av policyer.

WSO_2 IS serveren ble brukt for å validere autorisasjonsavgjørelser som ble sendt fra klientene. Serveren ble også brukt for å validere XACML policyer. I tillegg ble den brukt til å kommunisere med databasene via egendefinerte PIP som ble lagt til som utvidelser til serveren.

4 Implementasjon av ABAC i OpenMRS

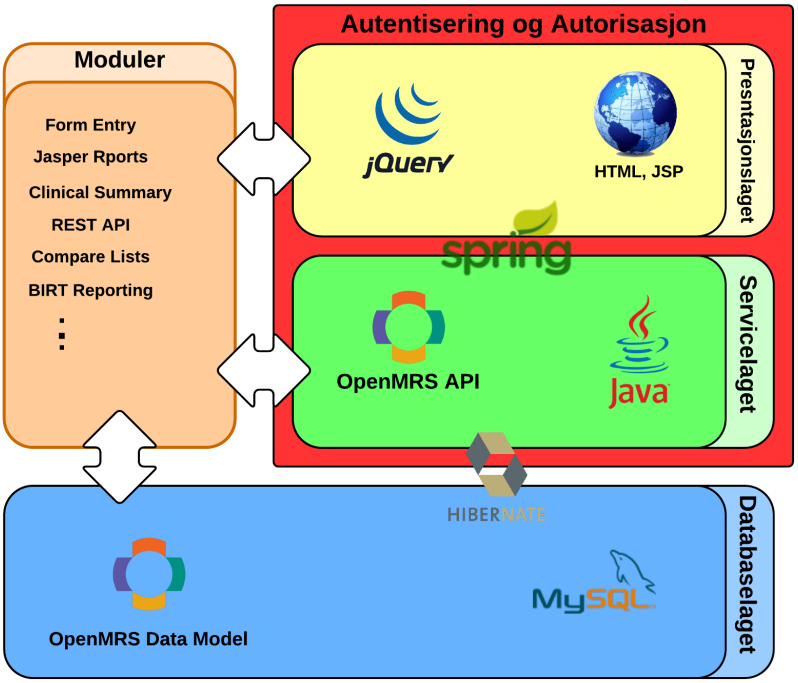
OpenMRS er et samfunnsdrevet, ikke for profitt samarbeidsprosjekt ledet av Regnestrief instituttet[20]. Det ble opprettet i 2004 for å bedre informasjonsflyten av helsedata i utviklingsland. OpenMRS samfunnet jobber for å utvikle et åpent elektronisk helsesystem som primært brukes til: pasient informasjon (all informasjon i fra besøk til prøver etc.), kartlegging av sykdommer og felles ordliste (eng. dictonary) som forklarer sykehuskonsepter som f.eks. blodtrykk, graviditet, malaria, vannkopper, (slik som Figur 4.1 viser). Systemet brukes blant annet i Etiopia, Tanzania, Nigeria osv[49], mens forskningen finner hovedsakelig sted i Europa og U.S.A.[49].



Figur 4.1: Konseptordliste i OpenMRS

4.1 Tilgangskontroll i OpenMRS

OpenMRS arkitekturen er lagdelt. Figur 4.2 viser at den er delt inn i tre lag. Et presentasjonslag, et servicelag og et databaselag. Prosjektet benytter seg av moduler som kobler seg opp til lagene slik at de kan dra nytte av funksjonaliteten lagene gir.



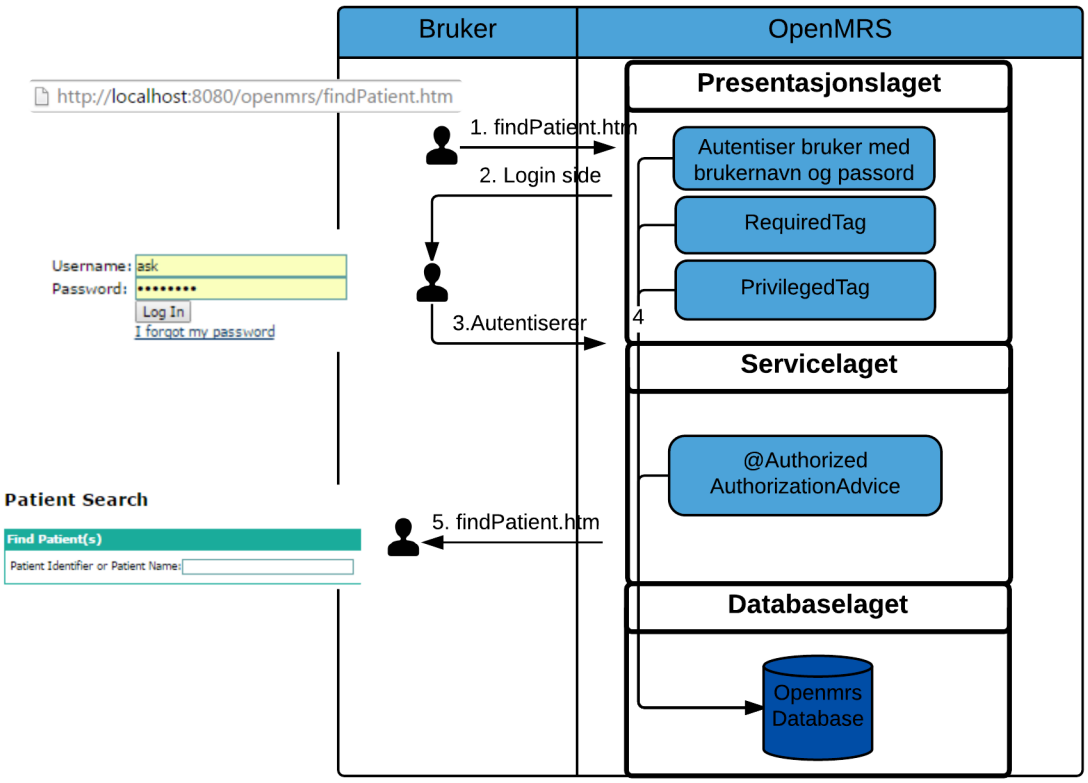
Figur 4.2: Oversikten til OpenMRS Systemet [4].

OpenMRS bruker rollebasert tilgangskontroll i systemet, og alle handlinger er styrt av privilegier. Applikasjonen bruker tagger og annotasjoner for å gi tilgang til ressurser. Autorisasjonsavgjørelsen er delt inn i to lag, presentasjonslaget og servicelaget.

4.1.1 Sikkerhetsflyten i OpenMRS

Autentisering skjer bare i presentasjonslaget, mens autorisasjonen skjer i presentasjonslaget og servicelaget. Presentasjonslaget er laget som kommuniserer med klientene. Servicelaget er et bindeledd mellom presentasjonslaget og databaselaget. Figur 4.3 viser hvordan en bruker får tilgang til en ressurs, stegene under forklarer autorisasjonsflyten.

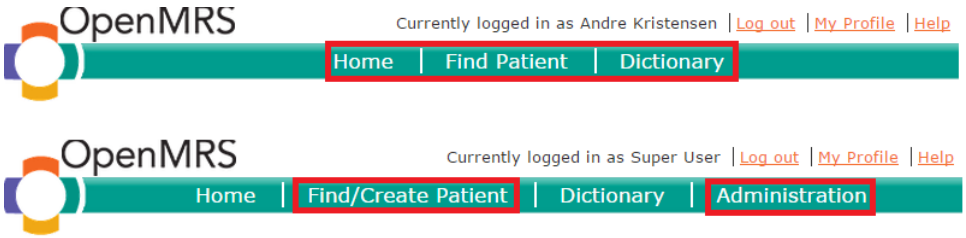
1. Først spør brukeren om tilgang til `findPatient.htm` siden.
2. Brukeren blir sendt til innloggingsiden.
3. Brukeren autentiseres med brukernavn og passord.
4. Serveren finner ut om det er et kall fra presentasjonslaget, eller om det kommer fra service laget. Den leter i databasen og finner ut om brukeren har tilgang til `findPatient.htm` siden.
5. Brukeren får tilgang til ressursen som er knyttet til `findPatient.htm` siden.



Figur 4.3: Sikkerhetsflyten i OpenMRS.

4.1.2 Presentasjonslaget

Autorisasjonen i presentasjonslaget begrenser hva en bruker kan se av linker og hvilke sider han kan navigere seg til. Dette gir bedre brukervennlighet fordi brukeren bare kan utføre handlinger han har rett til. Da slipper han å prøve seg fram for å finne ut hva han har rettigheter til. I OpenMRS kan brukerne i systemet få ulik tilgang. Figur 4.4 viser hvordan en bruker får tilgang til `Home | Find Patient | Dictionary`, men en administrator får tilgang til å navigere seg til flere sider og kan utføre flere handlinger. Dette blir realisert ved å bruke tagger i selve Java Server Pages (JSP) koden som generer siden ved å bruke JSP Standard Tag Library (JSTL).



Figur 4.4: En innlogget bruker og administrator i OpenMRS.

4.1.2.1 Tag for navigeringslinker: <openmrs:hasPrivilege>

Taggen `<openmrs:hasPrivilege>` brukes til å avgjøre hvilke navigeringslinker som skal bli med til siden når den lastes inn. Eksempel 4.1 krever at brukeren har `View Navigation Menu` privilegiet [9]. Taggen er koblet til en klassen `PrivilegeTag` som utfører logikk når taggen blir validert. Figur 4.4 viser at to ulike brukere har ikke tilgang til de samme lenkene.

```
1 <openmrs:hasPrivilege privilege="View Navigation Menu">
```

Eksempel 4.1: Taggen `<openmrs:hasPrivilege>`.

4.1.2.2 Tag for tillatelse til å navigere seg til sider: <openmrs:require>

Taggen `<openmrs:require>` brukes for å sjekke om brukeren har de rette privilegier for å vise siden. Eksempel 4.2 vil kreve at brukeren har `Manage Concept Classes` privilegiet. Hvis ikke blir brukeren sent til innloggingsiden og deretter sendt videre til

sitt opprinnelige mål [9]. Her blir taggen styrt av klassen `RequiredTag` som validerer dataen fra `<openmrs:require>` taggen.

```
<openmrs:require
    privilege="Manage Concept Classes"
    otherwise="/login.htm"
    redirect="/admin/concepts/conceptClass.form" />
```

Eksempel 4.2: Taggen `<openmrs:require>`.

4.1.3 Servicelaget

Servicelaget er laget som kommuniserer med presentasjonslaget og databaselaget. Autorisasjonen må være til stede også i dette laget slik at en angriper ikke bare kan bruke en lenke til å få tilgang til ressurser han ikke har tilgang til. Et eksempel er at en bruker logger seg på som vanlig, men kan komme til administrator siden ved å bruke en stjålet lenke eller bare prøve en tilfeldig lenke.

I OpenMRS blir ressursene i servicelaget beskyttet med annotasjoner istedenfor tagger. En annotasjon er metadata og brukes til å beskrive andre data. Annotasjoner kan prosesseres før selve koden kjøres. Den brukes for eksempel til å definere krav som skal oppfylles for at koden kan eksekveres. I prosjektet definerer det hvilke privilegier den innloggede brukeren må ha for å kjøre en spesifikk metode.

```
//EX 1: Requires that the user has one of the privileges.
@Authorized ({"View Users", "Add User"})
public void getUsersByName(String name);

//EX 2: Requires that the user has all the privileges.
@Authorized (value = {"Add Users", "Edit Users"}, requireAll=true)
public void getUsersByName(String name);

//EX 3: Requires that the user is authenticated.
@Authorized ()
public void getUsersByName(String name);
```

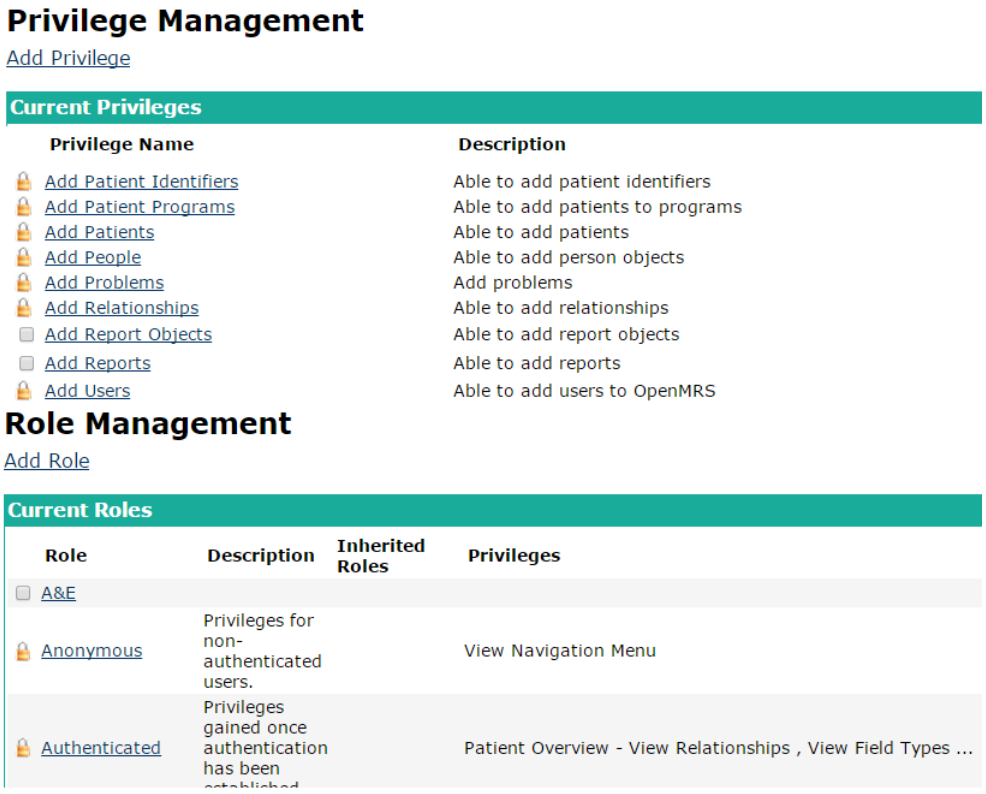
Eksempel 4.3: Annotasjonseksempel av en metode [9].

APIet er der alle bakgrunnsjobber blir utført. Den bruker `@Authorized` annotasjonen for å bestemme tilgang til metodenivå. Eksempel 4.3 viser hvordan annotasjoner er koblet til metoder og hvordan de definerer privilegiene som metodene krever for å kjøre. Annotasjonen prosesseres med refleksjon. Refleksjon er muligheten for et program å inspisere,

endre struktur og oppførsel på koden under kjøring. I OpenMRS brukes refleksjon til å finne metoder med `@Authorized` annotasjonen og kjører `AuthorizationAdvice` klassen for å se om brukeren som prøver å få tilgang har lov til å kjøre metoden.

4.1.4 Roller og Privilegier

Rollene og privilegiene blir redigert i en administratorside, se Figur 4.5. De bruker privilegier for å vise, hente og redigere data. Privilegier blir knyttet opp mot roller slik at rollene kan bli tildelt brukerne. Rollene kan arve fra andre roller og vil dermed få de samme privilegier. Hvis en rolle får nye privilegier, vil den som arver rollen også få de nye privilegiene.



Figur 4.5: Utsnitt fra administratorsidene `roller` og `privilegier`.

4.2 Utfordringer med OpenMRS

For å få bedre innsikt i OpenMRS må en se på hvilke utfordringer applikasjonen har med tanke på tilgangskontroll. Privilegiene er statiske og roller er dynamiske. Dette medfører at en vil få ulike implementasjoner av OpenMRS for hver instans som blir opprettet. Hver organisasjon må sette opp rolleprivilegiumkombinasjonen selv.

OpenMRS sitt konsept med roller som er knyttet til privilegier (handlinger) skalerer ikke. Det er fordi hvis man skal lage en ny rolle må man opprette rollen og legge til privilegiene selv. Etter en stund vil man ende opp med en rolleeksplosjon (beskrevet i seksjon 2.5.3) som blir vanskelig å håndtere.

Et annet problem med den nåværende tilgangskontrollen er at det er bare fire forhåndsdefinert roller og nesten 200 privilegier som skal fordeles. Eksempel 4.3 viser at privilegiene blir hardkodet, og dersom nye privilegier skal kunne brukes må applikasjonen rekompileres og privilegiene må knyttes til ressurser. Dette er veldig ressurskrevende.

Dynamiske roller krever mye av en administrator. Kun et fåtall av rollene er forhåndsdefinerte og resten må administrator lage selv og knytte dem til privilegier. Dette er en oppgave som er utsatt for å gjøre feil. Rollene kan arve fra andre roller. Dette kan medføre at det blir gitt privilegier til en rolle, og andre roller som arver fra denne rollen får privilegier som de ikke skulle ha.

Et eksempel på lite fleksibilitet er at med OpenMRS sin nåværende løsning er det ikke mulig å skille pasienter på behandlingsavdeling. Når en pasient blir behandlet blir det registrert behandlingsavdeling, men den blir ikke brukt videre for å skille tilgang. Dette er fordi en administrator ikke kan lage nye privilegier som kobles opp til behandlingsavdelingen.

Et eksempel av funksjonalitet som ikke kan implementeres med nåværende tilgangskontroll er noe sluttbrukeren ønsker seg. Nedefor er et sitat fra [50] oversatt fra engelsk.

OpenMRS ønsker muligheten til å begrense tilgang til pasienter og besøk med lokasjon. Dette dekker to brukerhistorier, (a) Flere installasjoner på samme plass. (b) Lokasjoner inni enheten slik at spesielle personvernskrav møtes, for eksempel psykiatri og kjønnssykdoms klinikk

4.2.1 Utfordringer med å implementere ABAC i OpenMRS

Implementasjon i et stort prosjekt byr på utfordringer. I prosjektet ble følgende utfordringer testet:

- Hvordan kan man beholde eksisterende autorisasjon med å bytte fra rollebasert til attributtbasert tilgangskontroll?
- Har prosjektet samlet autorisasjonsavgjørelsene på noen steder i applikasjonen eller er den spredd over hele koden?
- Hvordan kan man få ut privilegiene?
- Hvordan kan man få løst rolleprivilegiumproblemet?

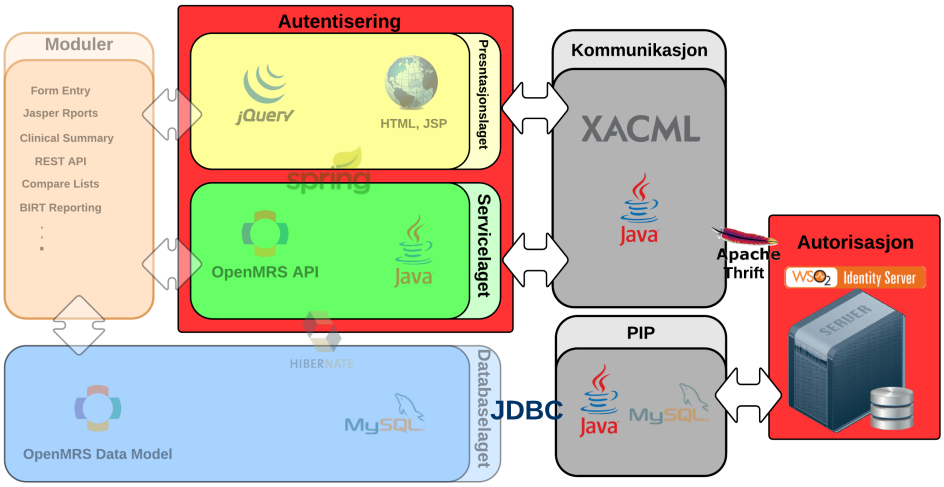
4.3 Implementasjon av ABAC

For å kunne implementere ABAC må det defineres flaskehalser hvor autorisasjonen skal plasseres (ref. seksjon 3.1.3). OpenMRS har tre definerte flaskehalser, en for alle API hendelser, en som definerer hva brukeren kan se av grensesnittet, og en som gir tilgang til hvilke sider det kan navigeres til.

En viktig avgjørelse som ble tatt var å beholde privilegiene. Dette var på grunn av at privilegiene er knyttet opp til alle metoder i systemet. Rollene og privilegiene ble gjort om til attributter og privilegiene blir brukt som handlinger i XACML forespørsler.

Figur 4.6 viser implementasjonen av løsningen. I den nye løsningen er det et klart skille mellom autentisering og autorisasjon. To nye moduler er lagt til for å generere og håndtere ABAC forespørsler. Den ene tar seg av kommunikasjonen og generering av forespørsler mellom OpenMRS og WSO_2 IS og bruker Apache Thrift for dette formålet. Den andre implementerer en PIP som lar WSO_2 IS serveren få tilgang til OpenMRS sin database gjennom Java Database Connectivity (JDBC) og henter den nødvendige informasjonen for å ta en avgjørelse.

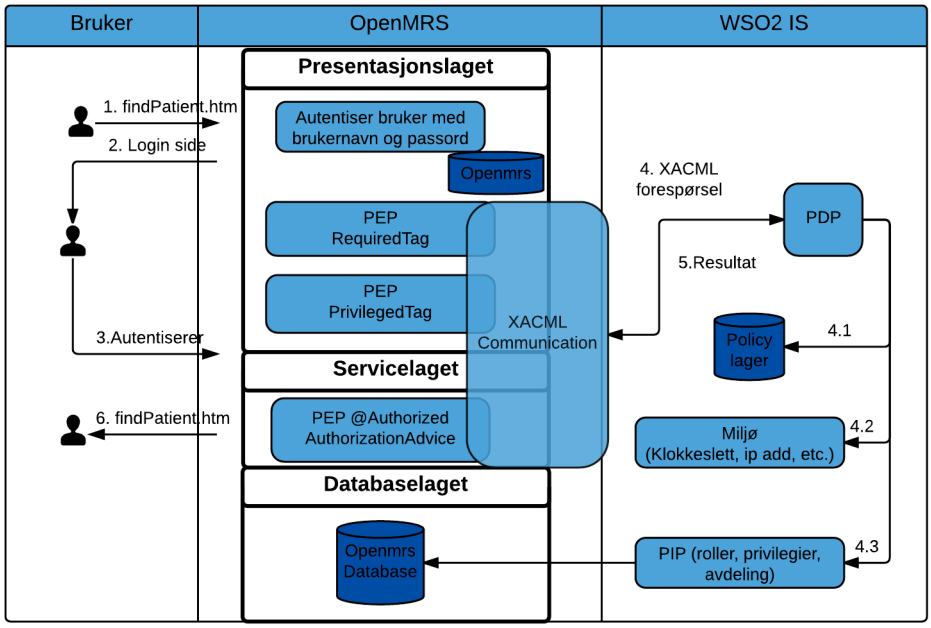
Andre nødvendige endringer som ble gjort er beskrevet i mer detalj i påfølgende delseksjoner.



Figur 4.6: Oversikten til OpenMRS systemet med ABAC.

4.3.1 Autentisering og autorisasjonsflyten til ABAC løsningen

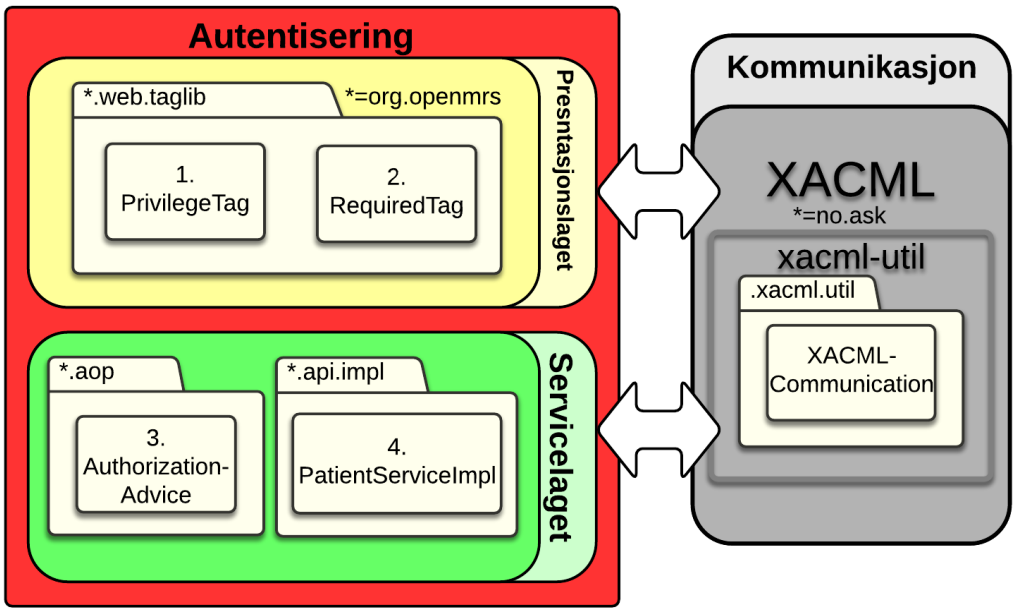
Den nye autorisasjonsflyten til ABAC løsningen benytter seg av den eksisterende autentiseringsmåten. Original løsningen 4.3 viser at `RequiredTag`, `PrivilegeTag` og `AuthorizationAdvice` klassene hadde direkte tilgang til databasen. Denne tilgangen er nå fjernet og eksisterende autorisasjonsgrunnlag RBAC er erstattet med ABAC. Det vil si at slik Figur 4.7 viser er klassene gjort om til PEP. WSO_2 IS server er lagt til som et PDP og kommuniserer med PEP. Serveren har også tilgang til OpenMRS databasen slik at den kan ta informerte avgjørelser.



Figur 4.7: Kommunikasjonen mellom OpenMRS og WSO_2 IS server.

4.3.2 Plassering av PEP

På bakgrunn av informasjon om flaskehalser kunne det defineres hvor PEP skulle plassers. Figur 4.8 viser en oversikt over OpenMRS prosjektet og to av modulene, `API` og `web`. XACML modulen brukes for kommunikasjon med WSO_2 IS. `PrivilegeTag(1)`, `RequiredTag(2)` og `AuthorizationAdvice(3)` klassene viser hvor eksisterende autorisasjonsavgjørelse var plassert. RBAC ble byttet ut med ABAC og klassene benytter seg nå av `XACMLCommunication` klassen for å avgjøre om brukeren får tilgang. `PatientServiceImpl(4)` klassen hadde ikke autorisasjon på pasienter. Her ble det også lagt til et PEP for å filtrere pasienter på lokasjon.



Figur 4.8: Oversikt over implementasjonene i OpenMRS.

4.3.2.1 XACML kommunikasjonsklassen

`XACMLCommunication` klassen tar seg av oppkobling til WSO_2 IS serveren. Klassen lager også forespørsler slik som ble vist i Eksempel 3.9 fra kapittel 3. Eksempel 4.4 viser hvordan metoden `getDecisionResults` fungerer. Metoden har følgende parametere: `subjektId`, `handlinger`, `miljø` og `ressurs`. Den lager en forespørsel som blir sendt til WSO_2 IS. Metoden får også svar fra serveren og sender svaret tilbake til brukeren. Svaret kan være `Permit, Deny, Indeterminate og NotApplicable`. Klassen lager forespørsler ut ifra variablene som blir sendt inn til metoden og hvis noen av dem er `null` blir de ikke med i forespørselen.

```
public class XACMLCommunication {
private PEPAgent pepAgent = null;
// Setup of communication in constructor
public List<String> getDecisionResults(String subjectId,
 Collection<String> actions, String environment, String resource)
 throws PEPAgentException, XMLStreamException, NullPointerException {
 if (pepAgent == null) {
  throw new NullPointerException("PEPAgent not initialized");
 }
 RequestDTO requestDTO = new RequestDTO();
 if (actions != null) {
  for (String privelege : actions) {
   requestDTO.setAttributeDTOs(ATTRIBUTE_CATEGORY_ACTION,
    request(requestDTO, privelege, ID_ACTION, DATA_TYPE_STRING));
  }
 }
 if (null != subjectId) {
  requestDTO.setAttributeDTOs(ATTRIBUTE_CATEGORY_ACCESS_SUBJECT,
   request(requestDTO, subjectId, ID_SUBJECT, DATA_TYPE_STRING));
 }
 if (null != environment) {
  requestDTO.setAttributeDTOs(ATTRIBUTE_CATEGORY_ENVIRONMENT,
   request(requestDTO, environment, ID_ENVIRONMENT, DATA_TYPE_STRING));
 }
 if (null != resource) {
  requestDTO.setAttributeDTOs(ATTRIBUTE_CATEGORY_RESOURCE,
   request(requestDTO, resource, ID_RESOURCE, DATA_TYPE_STRING));
 }
 // decision method returns Permit, Deny, Indeterminate and NotApplicable
  return decision(pepAgent.getDecision(requestDTO));
 }
}
```

Eksempel 4.4: Utsnitt av `XACMLCommunication` klassen.

4.3.2.2 Webmodulen (Presentasjonslaget)

I `webmodulen` ble det avdekket to flaskehalser, `PrivilegeTag` og `RequiredTag` klassene. Dette er klasser som kjøres når taggen i seksjon 4.1.2.2 og 4.1.2.1 valideres. Avgjørelsesgrunnlaget til klassene er blitt gjort om til et PDP for autentiserte brukere. Dette vil si at klassene bruker også `XACMLCommunication` klassen for kommunikasjon til PDP jf. Figur 4.8.

4.3.2.3 API-modulen (Servicelaget)

I `API-modulen` blir `AutorizationAdvice` klassen endret fra å bruke rollebasert tilgang til å bli et PEP som bruker `XACMLCommuncation` klassen for å kommunisere med PDP som kjører på WSO_2 IS serveren. Det ble implementert PEP i pasienttjenesten på metoden `getPatient()` for å vise hvordan PEP kan implementeres i flere lag og gi et skreddersydd autorisasjonsgrunnlag. Ved å implementere det i metoden kan det nå skilles på hvor pasienten fikk behandling, og dermed kreve at brukeren som prøver å få tilgang må ha tilgang til lokasjonen. Eksempel 4.5 linje 8-10 viser at det sendes en XACML forespørsel før pasienten hentes. Responsen fra serveren avgjør om brukeren får tilgang til ressursen.

```
1  @Transactional(readOnly = true)
2  public Patient getPatient(Integer patientId) throws APIException {
3   UserContext userContext = Context.getUserContext();
4   if (userContext.getAuthenticatedUser() != null) {
5    try {
6     ArrayList<String> actions = new ArrayList<String>();
7     actions.add(PrivilegeConstants.GET_PATIENTS.toString());
8     List<String> decisonResults =
9      xacml.getDecisionResults(userContext.getAuthenticatedUser()
10     .getId().toString(),actions, "openmrs.com", patientId.toString());
11    if (!decisonResults.isEmpty() &&
12     decisonResults.get(0).equals(XACMLCommunication.RESULT_PERMIT)) {
13     return dao.getPatient(patientId);
14    }
15   } catch (Exception e) {
16    e.printStackTrace();
17   }
18  }
19  return null;
20 }
```

Eksempel 4.5: XACML implementasjon i `PatientServiceImpl` klassen.

4.3.3 Policy Information Point

Hvis en XACML policy skal kunne benytte egendefinerte attributter må det lages en applikasjon som registrerer og leter etter attributtdata. WSO_2 IS benytter seg av en klasse som arver fra `AbstractPIPAttributeFinder` klassen. Det er en abstrakt klasse som har definerte metoder for å finne attributtverdier. Klassen kobler seg opp til databasen til OpenMRS for å finne attributtene den trenger. Applikasjonen blir lagt til i WSO_2 IS biblioteket som en komponent. Siden klassen arver fra `AbstractPIPAttributeFinder`

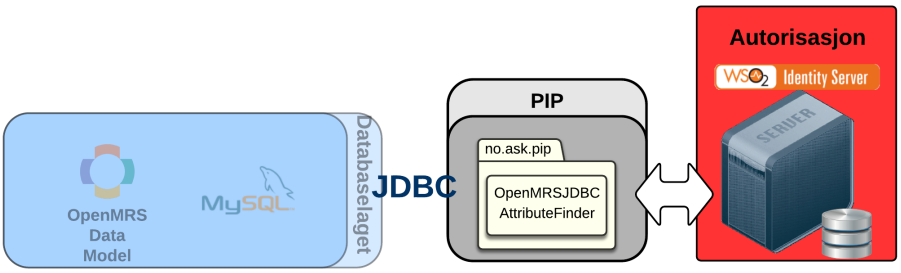
klassen vet WSO_2 IS hvilke metoder den kan bruke.

4.3.3.1 Konfigurasjonsfiler i WSO_2 IS

For at applikasjonen skal kunne finne databasekilden som sier hvilken database den skal koble seg til lages det en konfigurasjon som blir koblet opp mot et Java Naming and Directory Interface (JNDI). Tilkoblingen blir definert i `master-datasources.xml` fil til WSO_2 IS serveren. I `entitlement.properties` filen setter man opp navnet til applikasjonen med pakkedefinisjon og klassenavn slik at WSO_2 IS vet at det er et PIP. Her kan man også sette opp variabler som applikasjonen kan bruke.

4.3.3.2 PIP attributter

Når applikasjonen starter legger den til attributtene som er blitt definert. For OpenMRS er det rolle, privilegier og avdeling. Attributtene som PIP kan bruke som søkekriterier gjenspeiler attributtene som blir sendt som XACML forespørsel fra klienten (subjekt-id, handling, ressurs-id og miljø). Figur 4.9 viser hvordan PIP kommuniserer med OpenMRS sin database for å få tilgang til rollene, privilegiene og avdelingsinformasjonen til brukeren som spør om tilgang.



Figur 4.9: Utvidelsen som ble lagt til i WSO_2 IS serveren.

4.3.3.3 OpenMRSJDBCAttributeFinder klassen

Klassen `OpenMRSJDBCAttributeFinder` brukes til å definere attributter som kan brukes i policyer. Denne måten å lage attributter på er WSO_2 IS spesifikk. Eksempel 4.6 viser hvordan det kan gjøres:

- Linje 1 klassen `OpenMRSJDBCAttributeFinder` utvider `AbstractPIPAttributeFinder`. Klassen har definerte metoder som WSO_2 IS serveren kan benytte seg av.
- Linje 2-4 definerer hvilke attributter som skal brukes. Attributtene blir registrert i en `init-metode` ved oppstart.
- Linje 7 er metoden som henter attributtene. Metoden tar imot følgende identifikatorparametere: `subjekt, ressurs, handling, miljø, attributt , og utgiver`.
- Linje 10 viser at `attributtId` er et av de egendefinerte attributtene som blir sendt inn `PRIVILEGE_ID`. Attributtet avgjør hvilken SQL-spørring som skal lages, og hvilke ekstra parametere den trenger for å fullføre SQL-spørringen. Spørringen er av klassen `PreparedStatement` som tar imot en parametrisert spørring, og setter inn attributtene slik at de ikke kan bli *SQL Injected*. Dette er den største sikkerhetsfeilen ifølge OWASP[14].
- Linje 21-24 er det samme som linje 10 bare med andre attributtId(er): `role` og `location`.

```
public class OpenMRSJDBCAttributeFinder extends AbstractPIPAttributeFinder {
private static final String PRIVILEGE_ID = "http://openmrs.com/id/privilege";
private static final String ROLE_ID = "http://openmrs.com/id/role";
private static final String LOCATION_ID = "http://openmrs.com/id/location";
// init method, moduleName and SupportedAttributes
@Override
public Set<String> getAttributeValues(String subjectId, String resourceId,
 String actionId, String environmentId, String attributeId,
 String issuer) throws Exception {
  if (PRIVILEGE_ID.equals(attributeId)) {
   prepStmt = connection.prepareStatement(
   "SELECT privilege FROM openmrs.role_privilege WHERE privilege = ? "
   + "AND role IN (SELECT role FROM openmrs.user_role WHERE user_id = ?)");
   prepStmt.setString(1, actionId);
   prepStmt.setString(2, subjectId);
   resultSet = prepStmt.executeQuery();
   while (resultSet.next()) {
    returnAttributes.add(resultSet.getString(1));
   }
  logger.info(PRIVILEGE_ID +" " + returnAttributes.toString());
  } else if (ROLE_ID.equals(attributeId)) {  // Selects a users roles
  } else if (LOCATION_ID.equals(attributeId)) {
  // select location name if the user have access to the patients location
  }} // catch block and finally closing resultSet, prepStat and connection
 return returnAttributes;
 }}
```

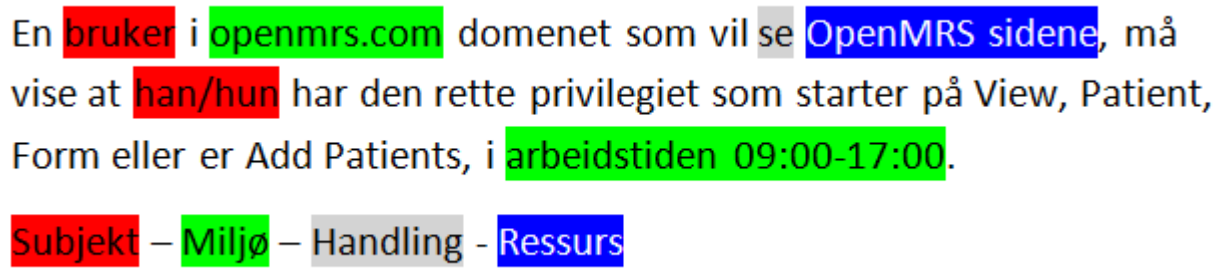
Eksempel 4.6: `OpenMRSJDBCAttributeFinder` klassen.

4.3.4 Oppgavene til PDP

Alle forespørsler som sendes fra klienten blir sendt til PDP for en avgjørelse. PDP prøver å finne den rette policyen for den mottatte forespørselen. Når den har mottatt forespørslene blir den evaluert og sendt tilbake til PEP med en avgjørelse om den er godkjent eller avslått. Hvis PDP trenger ytterlige informasjon kan den bruke PIP som leter etter informasjon om de egendefinerte attributtene. I sammenheng med OpenMRS (ref. Figur 4.7), viser den at PDP kan få forespørsler fra tre PEP plasser. To fra presentasjonslaget som er forespørsler fra `RequiredTag` og `PrivilegeTag` klassene, og en forespørsel fra servicelaget fra `@Authorized` annotasjonen som `AuthorizationAdvice` klassen utfører.

4.3.4.1 XACML policyer

For å kunne ta avgjørelser trenger PDP en policy, det vil si en XML fil med regler som sier hva som er lov. En policy har en overordnet avgjørelsesalgoritme. I OpenMRS er den satt til første aktuelle (eng. first applicable) regel. Det vil si at første regel som slår inn vil godkjenne forespørselen. Hvis ikke noen regler slår til vil den siste regelen nekte adgang, regelen er satt til å avslå autorisasjonsavgjørelsen. Figur 4.10 viser hvordan en XACML regel defineres, med en setning som sier hva som er lov. Attributtene som blir definert i figuren er fargekodet for å gjøre dem mer synlig.



Figur 4.10: Viser hvordan en regel kan knyttes opp til de ulike XACML attributtene.

Fra kapittel 3 seksjon 3.2.2.1 ble en policy beskrevet i detalj. Eksempel 4.7 viser hvordan regelen fra Figur 4.10 ser ut som en XACML policy:

- Linje 2-4 viser en beskrivelse av regelen. Dette er valgfritt men gjør det enklere for administratoren å se hva regelen gjør.
- Linje 5-16 definerer en `<Target>` tagg som sier hva som må være tilstede i forespørselen for at denne regelen skal bli evaluert. I dette eksemplet trenger den en ressurs som starter på `view`.
- Linje 17-36 viser en `<Condition>` tagg.
- Linje 18-26 er første *tilstand* som må passe, dvs. at et *subjekt* har attributtene som passer til det *regulære utrykket*. I eksemplet må attributtene være privilegier som brukeren har starte på `View, Patient, From` eller være `Add Patients`.
- Linje 27-34 er *tilstand* nummer to som må tilfredsstilles, *tilstanden* tilsier at klokkeslettet til forespørselen må være mellom `09:00 og 17:00`. Klokkeslettet blir testet mot tiden til serveren.

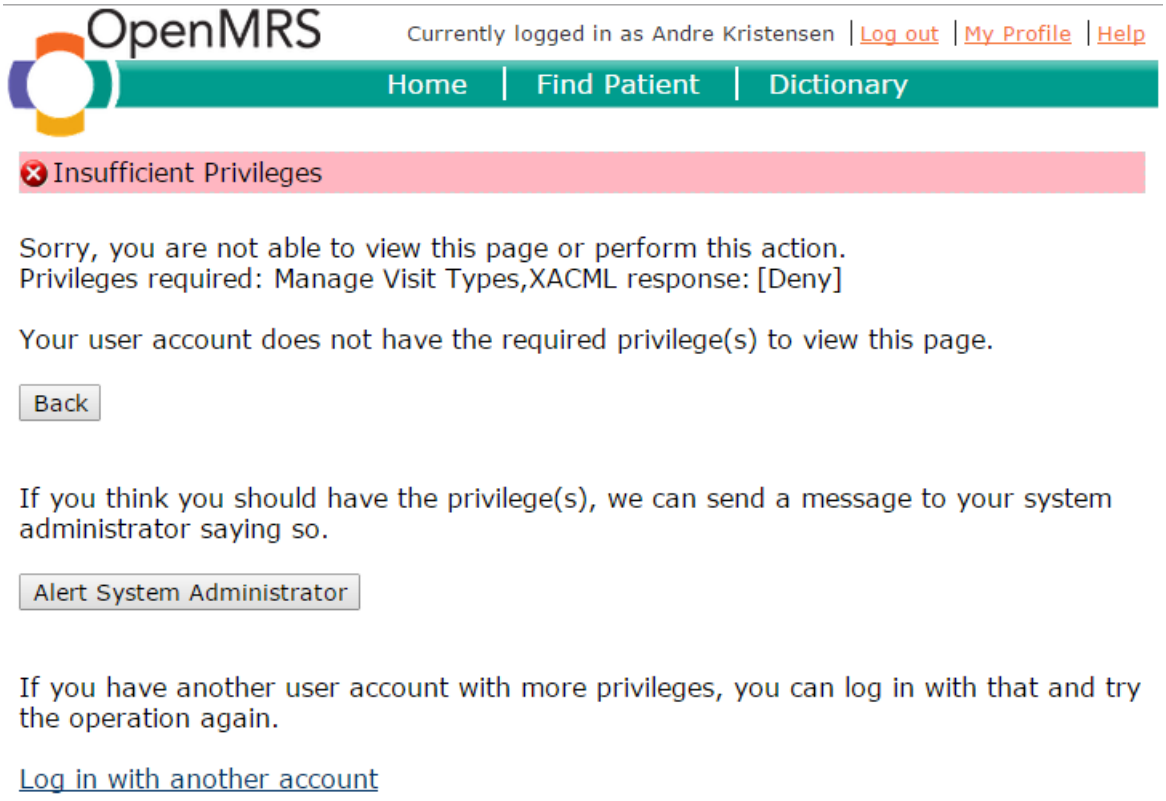
Dette er regelen for å se navigasjons lenker som Figur 4.4 viser i starten av kapittelet.

```
<Rule RuleId="view" Effect="Permit">
<Description> This is the rule for all the links and navigation. The target
 is view and the users need to have access to a privilege starting with View,
 Patient, Form or Add Patients in office hours. </Description>
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
     <AttributeValue DataType="...#string">^view</AttributeValue>
     <AttributeDesignator AttributeId="...:resource:resource-id"
     Category="...:attribute-category:resource"
     DataType="...#string" MustBePresent="true"></AttributeDesignator>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
    <Function FunctionId="...:string-regexp-match"></Function>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
     ^((View|Patient|Form)[ A-Za-z0-9-][ A-Za-z0-9-]*|Add Patients)
    </AttributeValue>
    <AttributeDesignator Category="...:subject-category:access-subject"
     AttributeId="http://openmrs.com/id/privilege"
     DataType="...#string" MustBePresent="false"></AttributeDesignator></Apply>
   <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:time-in-range">
    <AttributeValue DataType="...#time">09:00:00</AttributeValue>
    <AttributeValue DataType="...#time">17:00:00</AttributeValue>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
     <AttributeDesignator MustBePresent="false" DataType="...#time"
     AttributeId="...:environment:current-time"
     Category="..:attribute-category:environment" />
    </Apply>
   </Apply>
 </Condition>
</Rule>
```

Eksempel 4.7: Utsnitt av en XACML policy.

4.4 Sammenligning av RBAC og ABAC løsningen i OpenMRS

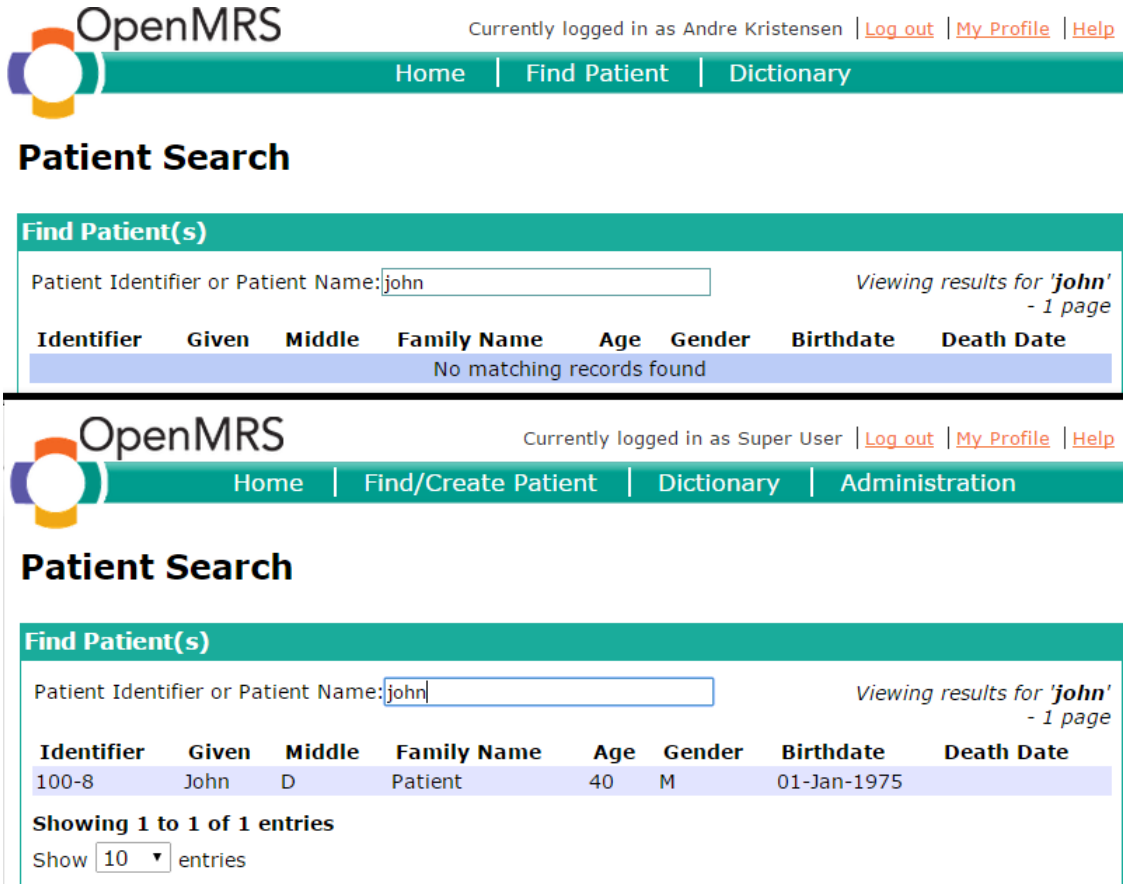
I løsningen byttet kandidaten ut rollebasert autorisasjon for autentiserte brukere med attributtbasert tilgangskontroll. Dette gir et større spillerom for fleksibilitet i avgjørelsesprosessen. En fordel er at det nå benyttes et eksternt PDP som gir muligheten til å sentralisere tilgangskontrollen slik at eksterne applikasjoner kan bruke de samme policyene. Det er mulig å endre regler under kjøring uten at brukeren må autentisere seg på nytt. En kan også ta med miljøvariabler i avgjørelsesprosessen, for eksempel klokkeslett. Figur 4.11 viser hvordan det ser ut når et API kall ikke får adgang, til ressursen: `XACML response: [Deny]`.



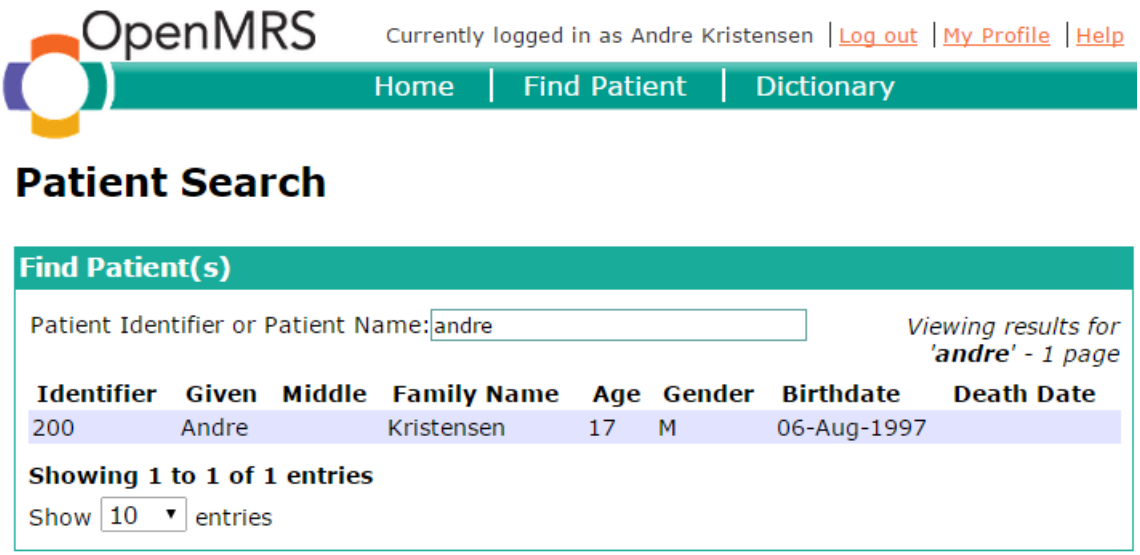
Figur 4.11: Viser hva som skjer når et API kall blir nektet.

4.4.1 Pasientsøk med behandlingsavdeling

En stor fordel er at det nå går an å skille på behandlingsavdeling til pasienten. Det vil si at en bruker som prøver å få tilgang til en pasient må ha tilgang til behandlingsavdelingene til pasienten. Figur 4.12 viser først en autentisert bruker som prøver å søke på `john` og får ingen treff. Rett under ser man en innlogget administrator som søker på det samme og får pasientinformasjonen. Den autentiserte brukeren har ikke tilgang til pasientens behandlingsavdeling, men en administrator har tilgang til alle. Figur 4.13 viser at den samme autentisert brukeren søker på pasient `andre` og finner pasienten, det vil si at brukeren har tilgang til behandlingsavdelingen til pasienten. Dette understøtter den første hypotesen: med ABAC kan pasienter skilles på behandlingsavdeling.



Figur 4.12: En vanlig bruker og en administrator søker på den samme pasient.



Figur 4.13: En bruker søker på en pasient med samme behandlingsavdeling.

4.4.2 Eksisterende rolleprivilegiumløsning

OpenMRS sin originale sikkerhetsløsning var vanskelig å løsrive seg fra. Den eksisterende rolleprivilegiumløsningen er for tett integrert i systemet og kan fortsatt gi rolleeksplosjon som beskrevet som en ulempe fra seksjon 2.5.3. Hypotese nummer to ble dermed ikke understøttet, siden ABAC ikke kunne løse rolleprivilegiumproblemet.

Det er mange like privilegier med tilknytning til ulike domener. Eks: Pasient, Skjema, osv. En løsning på privilegiene og rollekoblingene kunne vært å generalisere privilegiene til generelle handlinger slik som *View, Add, Assign, Delete, Edit, Manage, Purge, Remove, Update, Upload*. Dette er en finere utgave av *Create Read Update Delete (CRUD)*. Med denne fremgangsmåten kunne problemet bli løst, men det ville krevd mye endringer i den fundamentale koden, og det kan resultere i et svakere sikkerhetsnivå. Endringer på denne skalaen i prosjekter som har blitt utviklet i over ti år kan medføre ringvirkninger og uventet konsekvenser for andre deler av prosjektet. Her er det mange kodelukter[1] (eng. code smells), frykt for å gjøre endringer som kan få uante konsekvenser.

Løsningen som ble benyttet til slutt var å bruke de eksisterende privilegiene som handlingsattributter for XACML forespørslene.

[1] Kodelukter vil bli beskrevet i kapittel 6 seksjon 6.3.1

4.5 Evaluering av utfordringer med implementasjonen

Implementasjonen av ABAC har gitt OpenMRS mer fleksibilitet enn den hadde med eksisterende løsning. Avgjørelsesgrunnlaget til OpenMRS har økt betraktelig og gir systemet mulighet til å ta enda mer skreddersydde avgjørelser om brukeren får tilgang til ressursen.

- Hvordan kan man beholde eksisterende autorisasjon med å bytte fra rollebasert til attributtbasert tilgangskontroll?
  - For å beholde eksisterende autorisasjonsnivå ble ABAC plassert der autorisasjonen var. På grunn av at flaskehalsene i applikasjonen er plassert på strategiske steder, var det enkelt å bytte autorisasjonsgrunnlaget og beholde eksisterende autorisasjonsnivå.
- Har prosjekt samlet autorisasjonsavgjørelsen på noen steder i applikasjonen eller er den spredd over hele koden?
  - Autorisasjonsavgjørelsen til prosjektet var relativt samlet. Autorisasjonen var fordelt på tre klasser: `PrivilegeTag`, `RequiredTag` og `AuthorizationAdvice`. Det ble implementert autorisasjon på `PatientServiceImpl` klassen slik at det nå går an å kreve at brukeren har tilgang til pasienten sin behandlingsavdeling.
- Hvordan kan man få ut privilegiene?
  - Privilegiene til OpenMRS er for tett integrert i systemet. Ved å fjerne dem vil systemet trenge et redesign. På dette grunnlaget ble privilegiene beholdt.
- Hvordan kan man få løst rolleprivilegiumproblemet?
  - Problemet med 200 privilegier kunne ikke løses trivielt. Med ABAC implementasjon ble ikke privilegiumproblemet løst. Det ble laget en løsning som benyttet seg av privilegiene som handlingsattributter.

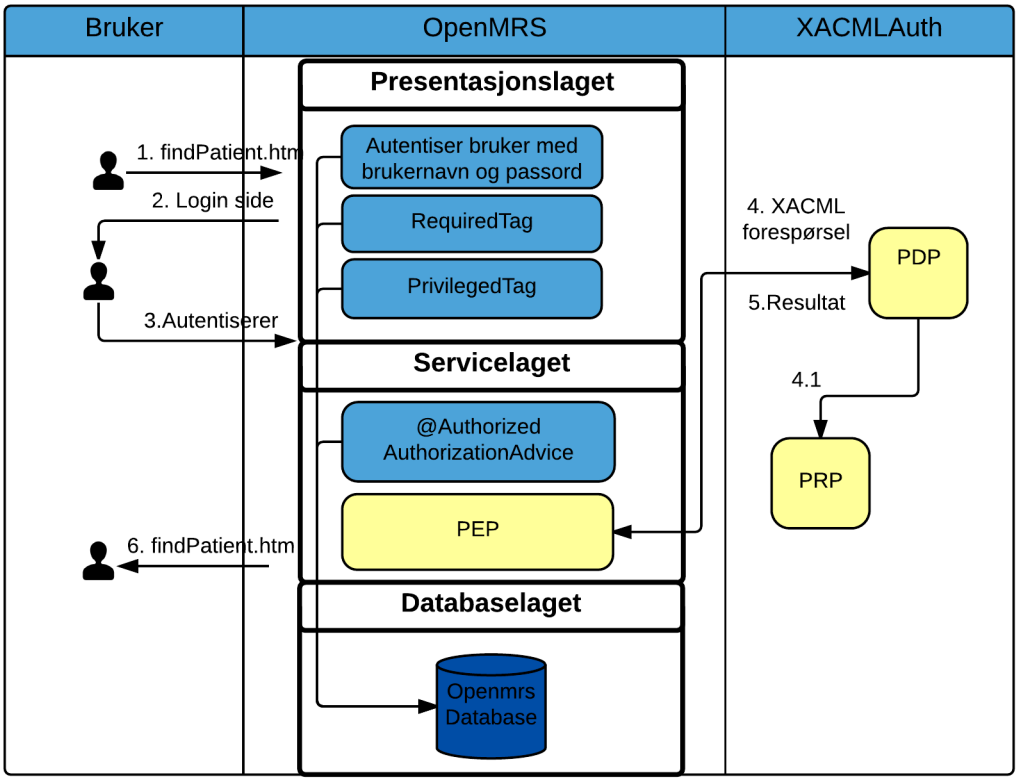
4.6 Relatert arbeid: Policy Based Electronic Medical Record System

I denne seksjonen vil det bli beskrevet en implementasjon av OpenMRS som bruker Policy-Based Access Control (beskrevet i seksjon 2.6.5). Arbeidet er en masteroppgave

med tittel *Policy Based Electronic Medical Record System* skrevet av Ranaweera[30]. Seksjonen beskriver autentisering, autorisasjonsflyten, vurdering og en sammenligning mellom ABAC og PBAC versjonen.

4.6.1 Autentisering og autorisasjonsflyten til PBAC løsningen

Løsningen benytter seg av OpenMRS sin eksisterende autorisering og autorisering fra PDP serveren. Figur 4.14 viser hvordan den følger den gamle autentiseringen slik som Figur 4.3 beskriver i kapittel 4. Det som skiller seg ut er at det er lagt til et PEP i servicelaget som kommuniserer med en ekstern `XACMLAuth` server etter at den har kjørt den gamle autoriseringen.



Figur 4.14: PBAC løsningen.

4.6.2 Vurdering av PBAC løsning

Løsningen auto genererer policyer men hvis det skal legges til ekstra avgjørelsesgrunnlag må det gjøres manuelt. Fokuset til løsningen er tre komplekse sikkerhetskrav fra [30], oversatt fra engelsk.

1. *En doktor kan se pasientinformasjon bare mellom 08:00 og 16:00.*
2. *En doktor skal ikke kunne se en pasient sin informasjon mer en n ganger.*
3. *En doktor som prøver å se HIV infiserte pasienter må ha et autorisasjons-token.*

PBAC løsningen benytter seg av eksisterende avgjørelsesgrunnlag og legger til XACML avgjørelse på toppen. Det fungerer som en begrensing på avgjørelsesgrunnlaget. Å begrense antall ganger et dokument leses har potensiale til å skape problemer og har ikke noe praktisk formål. Oppgaven utnytter ikke XACML sitt fulle potensial, den benytter seg ikke av ressursene i OpenMRS slik som pasienter. I praksis er det muligheter for å bruke miljøattributter på toppen av eksisterende tilgangsmekanisme.

4.6.3 Teknisk sammenligning av OpenMRS ABAC og PBAC løsning

Tabell 4.1 viser oversikten over hva som er hovedforskjellene mellom den tidligere masteroppgaven som implementerte PBAC og denne oppgaven som implementerer ABAC. Oversikten viser at det er brukt ulike teknologier. I PBAC versjonen av OpenMRS ble det tatt et valg å beholde eksisterende autorisasjon og legge til XACML. I ABAC versjonen ble det tatt en avgjørelse om å bytte ut eksisterende autorisasjonsløsning og bare benytte seg av XACML for autentiserte brukere.

[2] Åpen kildekode, separat prosjekt som WSO_2 IS bruker. Det kan også brukes for å lage egen PDP[51].

OpenMRS	**PBAC**	**ABAC**
RBAC	Ja	Nei
PDP - PEP kommunikasjon	SOAP + WSDL	Apache Thrift
Policy Decision Point	Egen	WSO_2 IS
Statisk policy	Ja	Nei
Egendefinerte attributter	Nei	Ja
XACML - Versjon	2.0	3.0
XACML - Motor	$HERAS^{AF}$	$Balana^2$
XACML - Miljø	Ja	Ja
XACML - Handlinger	Ja	Ja
XACML - Ressurser	Nei	Ja
XACML - Subjekt	Ja	Ja

Tabell 4.1: OpenMRS sammenligning av ABAC og PBAC.

4.6.4 Forskjeller mellom ABAC og PBAC implementasjon

Hovedforskjellen mellom ABAC og PBAC løsningene er at i PBAC løsningen stoler systemet på brukeren og alt det den sender inn til PDP. Det blir ikke tatt noe sjekk om brukeren som sender attributtene har de rette attributtene. I ABAC løsningen blir alle attributter validert av XACML policyen. Det vil si at PIP sjekker om brukeren har attributtene som blir sendt inn. PBAC løsningen bruker den gamle autorisasjonsmekanismen og XACML policy til å autorisere brukeren for å få tilgang til ressursene.

4.6.5 ABAC og PBAC XACML-utnyttelse

Følgende forskjeller kan sammenlignes:

- XACML Miljø: PBAC versjonen bruker klokkeslett, ABAC versjonen bruker klokkeslett og domene.
- XACML Handlinger: PBAC versjonen bruker egendefinert *CRUD* løsning, ABAC bruker privilegiene som ressursen trenger og sjekker om brukerne har tilgang til ressursen.
- XACML Ressurser: PBAC benytter seg ikke av ressurser. ABAC bruker metodenavn som ressurs og pasientens behandlingsavdeling. Det vil si at brukeren som vil se pasienten må ha autorisasjon til lokasjonen for å se pasienten.
- XACML Subjekt: PBAC sender inn rollen til brukeren, ABAC sender inn bruker-id

slik at den kan brukes til å sjekke om brukeren har rollene, privilegiene og lokasjonen som kreves for å få tilgang til ressursen.

4.6.6 Oppsummering av OpenMRS med PBAC

Oppgaven løser ikke rolleprivilegiumproblemet, den bygger på eksisterende løsning. Løsningen gir mulighet for å bruke miljøvariabler som klokkeslett. XACML implementasjonen setter en begrensning på eksisterende tilgangskontroll. Systemet har full tillit til alle attributter som blir sendt inn, det forekommer ikke noe sjekk om brukeren har de rettigheter den spør etter.

5

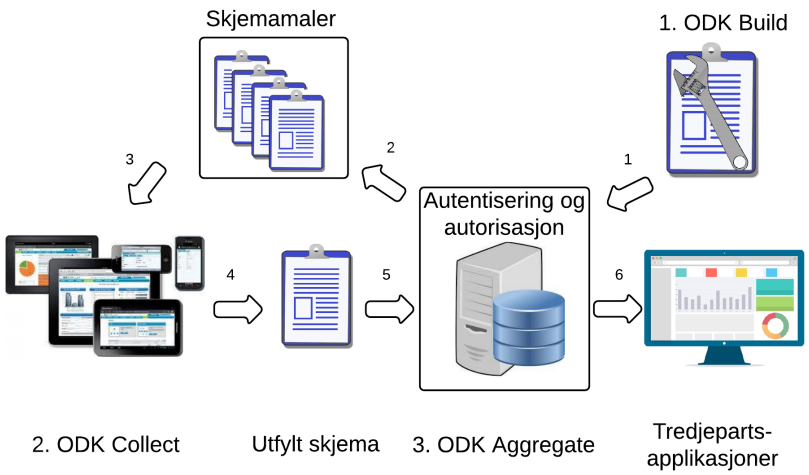
Implementasjon av ABAC i ODK Aggregate

ODK Aggregate startet i 2008 som et sabbatsårprosjekt sponset av Google [16]. Det er designet for å være en generisk datainnsamlingsplass der tredjepartsapplikasjoner kan lagre dataene de samler inn fra mobile applikasjoner. Det ble utviklet spesielt for å støtte datainnsamling i utviklingsland med svak/ikke eksisterende infrastruktur, og for at de skal de kunne samle inn data på en kostnadseffektiv måte.

ODK kan brukes for å kartlegge sykdommer, naturkatastrofer osv. Figur 5.1 viser hvordan dataene kan plasseres i Google Maps for å gi en bedre forståelse hvor dataene ble innsamlet og hvordan det så ut på plassen. ODK brukes blant annet av Academic Model for Providing Access to Healthcare (AMPATH)[52] i Kenya. Prosjektet har som mål å kartlegge og teste befolkningen for HIV og AIDS.

Figur 5.1: ODK datainnsamling visualisert i Google Maps [5].

ODK Aggregate er en av tre komponenter i ODK økosystemet. Flyten til økosystemet blir vist med Figur 5.2. Systemet virker slik at ODK Build lager skjemamaler som blir opplastet til ODK Aggregate. Skjemamalene blir deretter benyttet av ODK Collect som er en Android applikasjon. Applikasjonen brukes til å utføre undersøkelser og sende dem til ODK Aggregate. ODK Aggregate fungerer som en oppbevaringsplass for dataene som er innsamlet. Tredjepartsapplikasjoner kan dermed benytte seg av dataene og visualisere dem i grafer samt bruke dem som beslutningsgrunnlag.



Figur 5.2: Oversikten til ODK økosystemet.

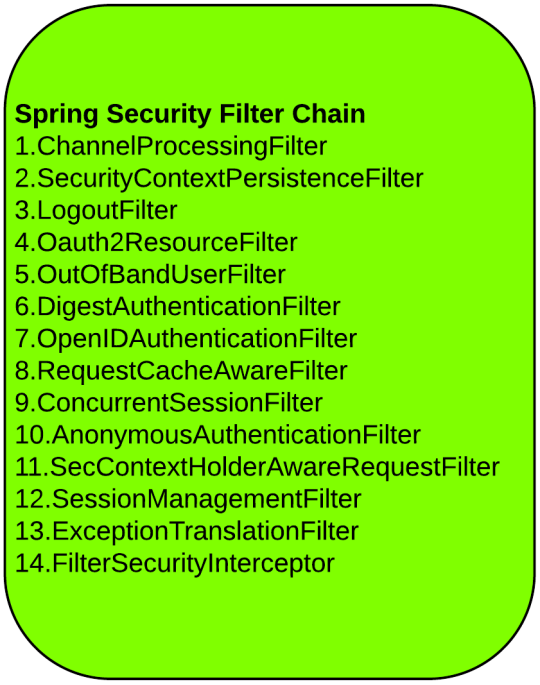
5.1 Tilgangskontrollen i ODK Aggregate

ODK Aggregate benytter seg av rollebasert tilgangskontroll. Den er implementert med Spring Security som rammeverk og sikkerheten blir lagt til Uniform Resource Locator (URL)-nivå. Det vil si at ressursene er beskyttet i presentasjonslaget. Figur 5.3 viser at URLen har fragment i seg, dvs. alt som er etter # er et fragment til en URL. Innholdet i hver side er også beskyttet ved å bruke autorisasjon på kodenivå i presentasjonslaget.

Figur 5.3: `Aggregate.html` siden.

5.1.0.1 URL-nivå beskyttelse med Spring Security

Spring Security støtter et stort spekter med autentiseringsmekanismer og rollebasert autorisasjon. Dette gjør det til det mest foretrukne valget for Java Platform Enterprise Edition (J2EE) utvikling. Spring Security Filter Chain (SSFC) er basert på standard `javax.servlet.Filter` klassen. Det er klasser som hver forespørsel må igjennom før den får tilgang til ressursen. Rekkefølgen på filtrene er viktig siden de bruker informasjon fra de andre filtrene. Figur 5.4 viser hvordan ODK Aggregate har laget filterkjedet sitt.



Figur 5.4: Oversikten ODK Aggregate sitt Spring Security Filter Chain.

Spring Security benytter seg også av en konfigurasjonsfil. Eksempel 5.1 viser hvordan URLene får hardkodede roller knyttet opp til seg. Linje 1 viser at `/aggregateui/preferenceservice` URLen trenger rollen `ROLE USER`. Denne løsningen gir lite rom for fleksibilitet og låser applikasjonen til et sett med roller. I sammenheng med SSFC og Figur 5.4 er det filter nummer 14 som beskytter applikasjonen for tilgang til URLene som konfigurasjonsfilen definerer.

```
<intercept-url pattern="/aggregateui/preferenceservice"
               access="hasRole('ROLE_USER')" />
<intercept-url pattern="/aggregateui/securityservice"
               access="hasRole('ROLE_USER') or
               hasRole('ROLE_SITE_ACCESS_ADMIN')" />
<intercept-url pattern="/aggregateui/filterservice"
               access="hasRole('ROLE_DATA_VIEWER')" />
```

Eksempel 5.1: Utsnitt fra `applicationContext-security.xml` filen.

5.1.0.2 Kodenivåbeskyttelse

ODK Aggregate benytter seg av beskyttelse på kodenivå. Det vil si at det blir sjekket for autorisasjon i koden. Eksempel 5.2 viser et av stedene hvor det blir benyttet Kodenivå-beskyttelse. Metoden sjekker om brukeren er autorisert for en gitt fane i applikasjonen. Koden viser at rollene er hardkodet og kan ikke endres.

```
... GrantedAuthorityName => GAN
private boolean authorizedForTab(Tabs tab) {
 switch (tab) {
  case SUBMISSIONS:
   return userInfo.getGrantedAuthorities().contains(GAN.ROLE_DATA_VIEWER);
  case MANAGEMENT:
   return userInfo.getGrantedAuthorities().contains(GAN.ROLE_DATA_OWNER);
  case ADMIN:
   return userInfo.getGrantedAuthorities().contains(GAN.ROLE_SITE_ACCESS_ADMIN);
  case ODKTABLES:
   return userInfo.getGrantedAuthorities().contains(GAN.ROLE_SYNCHRONIZE_TABLES)
   ||  userInfo.getGrantedAuthorities().contains(GAN.ROLE_ADMINISTER_TABLES);
  default:
  return false;
 }
}
...
```

Eksempel 5.2: Utsnitt fra `AggregateUI.java` klassen.

5.1.1 Sikkerhetsflyten i ODK Aggregate

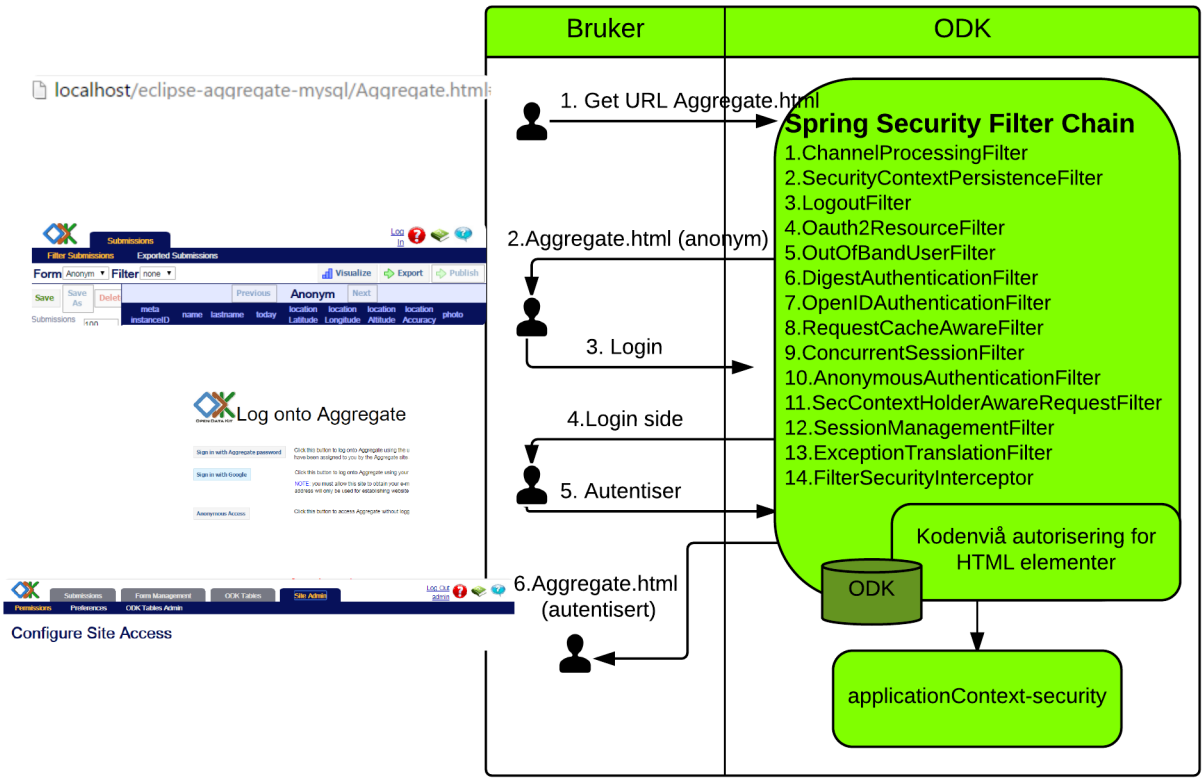
Applikasjonene benytter seg av to autentiseringsmekanismer, basisautentisering (Basic Authentication) og OpenID (OID) som bruker en Google konto for autentisering. Basisautentisering er en klartekst autentiseringsmekanisme som sender brukernavn og passord i headeren med hver forespørsel. Headeren kan f.eks. se ut slik `Authorization: Basic`

`YWRtaW46YWRtaW4=`, der brukernavn og passord ikke er kryptert men enkodet med Base64 koding. I klartekst ser brukernavnet og passordet slik: `admin:admin`. Autentiseringsmekanismen er ikke sikker, men med HTTPS blir kommunikasjonen kryptert.

OpenID er en åpen standard og en desentralisert protokoll som lar brukeren autentisere seg på vegne av noen kjente tjenester som Google, Yahoo, Flicker, WordPress osv.[53]. For ODK Aggregate vil det si at de bruker Google og lar brukeren autentisere seg med en Gmail konto. Når brukeren er blitt autentisert får ODK Aggregate tilgang til et autentisert token fra Google som de kan bruke for å verifisere autentiseringen.

For at brukeren skal få tilgang til en ressurs må den igjennom noen steg. Figur 5.5 viser hvordan sikkerhetsflyten i applikasjonene foregår.

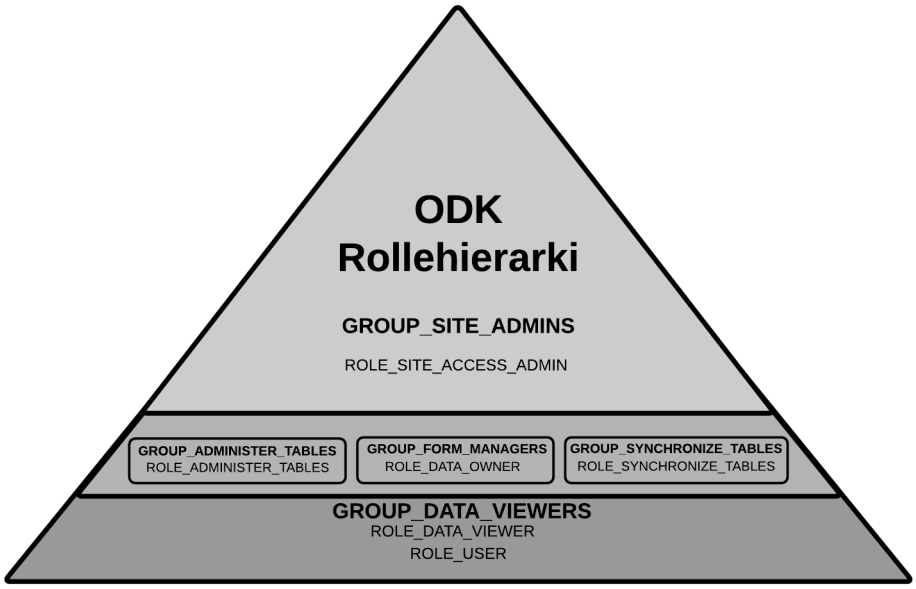
1. En bruker spør om `Aggregate.html` siden. Forespørselen blir autorisert på URL og kodenivå.
2. Applikasjonen finner ut at brukeren er anonym og får tilsendt `Aggregate.html` siden som en anonym bruker.
3. Bruker velger å logge seg inn ved å klikke på innloggings lenken.
4. Brukeren blir sendt til innloggingsiden der vedkommende må autentisere seg med brukernavn og passord.
5. Forespørselen blir autorisert på URL og kodenivå.
6. Returnerer `Aggregate.html` siden med innholdet som brukeren er autorisert til å se.



Figur 5.5: Sikkerhetsflyten i ODK Aggregate.

5.1.2 Autorisasjonen i ODK Aggregate

ODK Aggregate benytter seg av rollebasert tilgangskontroll med rollehierarki. Dette er $RBAC_1$ som ble gjennomgått i seksjon 2.5.2. Rollene er hardkodet, det vil si at de kan ikke endres etter applikasjonen er oppe og kjører. Figur 5.6 viser rollehierarkiet til ODK Aggregate. Rollene er delt inn i 5 grupper. Gruppene som ligger under arver de gruppene som ligger over. Eks: `GROUP FORM MANAGER` rollegruppen arver `GROUP DATA VIEWERS` rollegruppen, men ikke de andre rollegruppen som ligger ved siden eller under i hierarkiet.



Figur 5.6: Oversikten over ODK Aggregate sitt rollehierarki.

5.2 Utfordringer med nåværende løsning

ODK Aggregate er et klassisk rollebasert prosjekt der rollene er statiske. Rollene tilhører flere ressurser og gir implisitte handlinger til dem. Tilgangen til ressursene styres av en konfigurasjonsfil som Spring bruker. Filen spesifiserer hvilke URLer og roller som den trenger (beskrevet i seksjon 5.1.0.1). Brukere av systemet kan se alle skjemaene, og det finnes ikke en måte å filtrere dem på. Eksempel 5.3 viser på linje 8 og 15 at de har planer om å implementere autorisasjon, men de har ikke kommet så langt. Dette kan knyttes opp til OWASP *Missing Function Level Access Control (MFLAC)* som sier at det ikke er nok med bare autentisering, men hver forespørsel må bli autorisert i flere lag i applikasjonen slik at brukere ikke kan bare teste seg fram for å finne ubeskyttede ressurser.

```
1  private static IForm getForm(String topLevelAuri, CallingContext cc)
2  throws ODKOverQuotaException, ODKEntityNotFoundException, ODKDatastoreException{
3   List<IForm> forms = internalGetForms(topLevelAuri, cc);
4   if ( forms.isEmpty() )
5    throw new ODKEntityNotFoundException
6      ("Could not retrieve form uri: " + topLevelAuri);
7   IForm f = forms.get(0);
8   // TODO: check authorization?
9   return f;
10 }
11 public static final List<IForm>
12  getForms(boolean checkAuthorization, CallingContext cc)
13  throws ODKOverQuotaException, ODKDatastoreException {
14  List<IForm> forms = internalGetForms(null, cc);
15  // TODO: check authorization
16  return forms;
17 }
```

Eksempel 5.3: Mangel på autorisasjonssjekk av skjemaer.

ODK Aggregate bruker URLer med fragmenter. Figur 5.3 fra seksjon 5.3 viser et eksempel på hvordan det ser ut. Fragmenter gjør det vanskelig å få tak i hele URLen og kan dermed ikke spesifisere ressursen helt til det fineste nivået[54].

Kodenivåautorisasjonen fungerer som autorisasjonen for komponentene til websiden, det vil si at den skiller ut hva brukeren kan se når siden lastes inn. Koden utvikles i Java og kompileres med Google Web Toolkit (GWT)[55] til JavaScript[56]. Problemet med GWT er at det ikke kan kompilere nye biblioteker til JavaScript, bare forhåndsdefinerte biblioteker [57].

5.2.1 Utfordringer med å implementere ABAC i ODK Aggregate

Implementeringen av ABAC i ODK Aggregate har de samme utfordringene som ble nevnt for OpenMRS (beskrevet i seksjon 4.2.1).

- Hvordan kan man beholde eksisterende autorisasjon med å bytte fra rollebasert til attributtbasert tilgangskontroll?
- Har de samlet autorisasjonen på noen plasser eller er den spredd utover hele koden?
- Hvordan kan man løsrive seg fra hardkodede roller i koden?

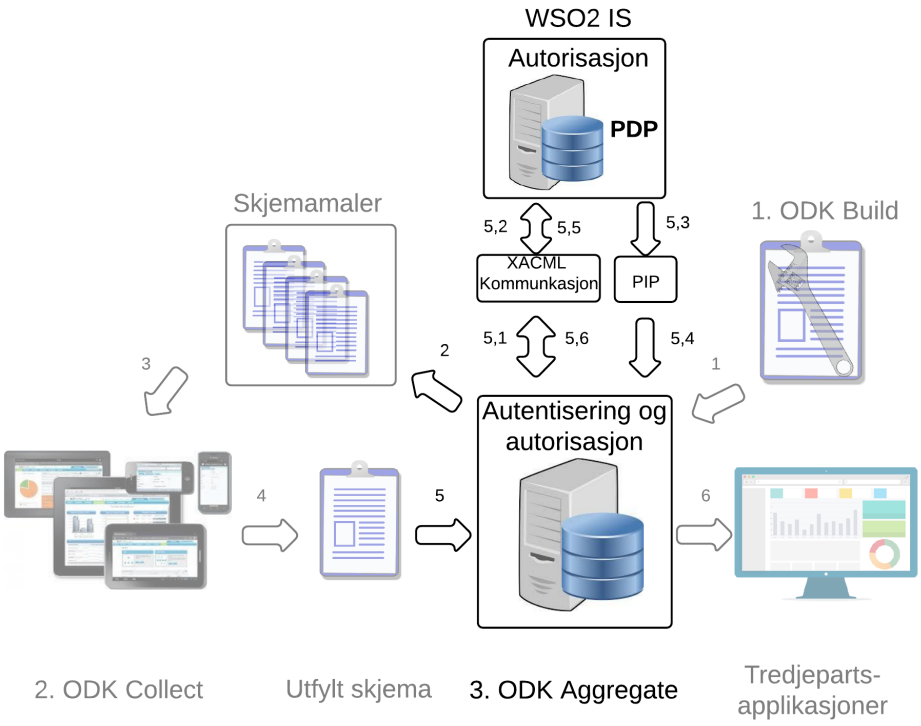
5.2.2 Svakheter med ODK Aggregate applikasjonen

Rapporten *Analyse av ODK Aggregate* [58] er en rapport som analyserer sikkerhetshull i ODK Aggregate. Rapporten påpeker at basisautentisering er svak og det er dårlig passordbehandling av intern bruker. Passordene blir *hashet* med Message-Digest algorithm 5 (MD5), og ikke saltet[1]. MD5 regnes som en ikke sikker *hash-algoritme*, dette fordi den har påvist kollisjon. Det vil si at et ord som blir hashet kan få samme *hash* som et annet ord. Det avdekkes flere kritiske feil som kan knyttes til OWASP topp 10 listen[14]: *Cross Site Scripting (XSS)*, *Unvalidetet Redirects and Forwards*, og *Missing Function Level Access Control*. Fokuset for oppgaven er autorisasjon og vil dermed legge vekt på *MFLAC*. Resten av sikkerhetshullene avdekket i rapporten vil ikke være i fokuset til oppgaven.

5.3 ODK Aggregate implementasjon av ABAC

Figur 5.7 viser oversikt over implementasjonen av ABAC i ODK Aggregate. På samme måte som OpenMRS vil det bli brukt en server som fungerer som PDP. For å kunne kommunisere med serveren kan `XACMLCommunication` klassen som OpenMRS bruker brukes til å kommunisere med PDP. Det blir også laget et PIP for å få tilgang til ODK Aggregate sin database.

[1] Salt er tilfeldige tall/bokstaver/tegn som legges til, f.eks. et passord får et tilfeldig «ord» konkatenert før det hashes slik at det ikke kan gjette hashen.

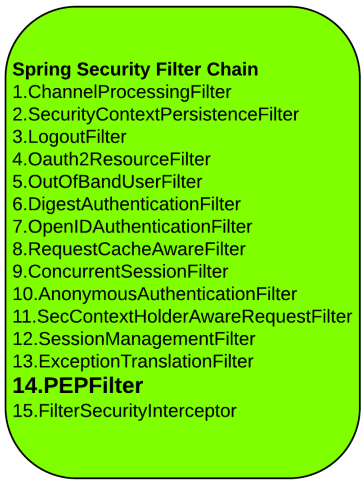


Figur 5.7: ODK Aggregate systemet med ekstern PDP server.

5.3.1 Flaskehalser i ODK Aggregate

Siden Spring Security ble brukt var det ikke vanskelig å identifisere flaskehalsene i ODK Aggregate. Trafikken blir tvunget igjennom filterkjedet, og derfor virket det tilstrekkelig å endre det for å bytte ut statiske roller med dynamiske. Imidlertid etter en grundigere undersøkelse av applikasjonen ble det avduket at i tillegg til URL-nivå beskyttelse var autorisasjon spredt rundt i koden på metodene. Dette skapte problemer siden det ikke er noe definert flaskehals for autorisasjonen på kodenivå, og det gikk ikke å bytte de statiske rollene med dynamiske roller for kodenivået.

I flaskehalsen på URL-nivå ble det laget en `PEPFilter` klasse. Filteret ble plassert slik som i Figur 5.8: rett før `FilterSecuirytInterceptor` klassen som kan utføre benektelsen hvis en forespørsel fra `PEPFilter` klassen ikke blir godkjent.



Figur 5.8: Oversikten ODK Aggregate sitt Spring Security Filter Chain utvidet med PEP-Filter.

Flaskehalsene på kodenivå var vanskeligere å kartlegge. Applikasjonene hadde en flaskehals for hvilke faner som skal vises. Her ble to implementasjoner prøvd men ingen virket som tiltenkt.

- Først ble det prøvd å implementere direkte i koden som blir kompilert til JavaScript med GWT, men dette virket ikke på grunn av tredjepartsbiblioteker ikke kan kompileres fra Java til JavaScript (beskrevet i seksjon 5.2).

- Det andre som ble prøvd var å bruke GWT-Remote Procedure Calls (RPC) med asynkrone kall. GWT-RPC blir kompilert til Asynchronous JavaScript and XML (AJAX)[59]. Denne måten virket lovende og kommunikasjonen mellom ODK Aggregate og WSO_2 IS fungerte. Metoden `authorizedForTab` fra Eksempel 5.2 skal gi tilbake `true/false` om en bruker får tilgang til fanen. Det var en detalj som stoppet det hele. Metoden trenger å være rask og da fungerer ikke asynkron kommunikasjon.

  Problemet kan ikke løses med nåværende løsning da GWT ikke støtter synkrone kall[60]. Google begrunner dette med at JavaScript kjører i en tråd og synkroniserte kall kan dermed blokkere hele applikasjonen.

5.3.2 Spring Security konfigurasjonsfil

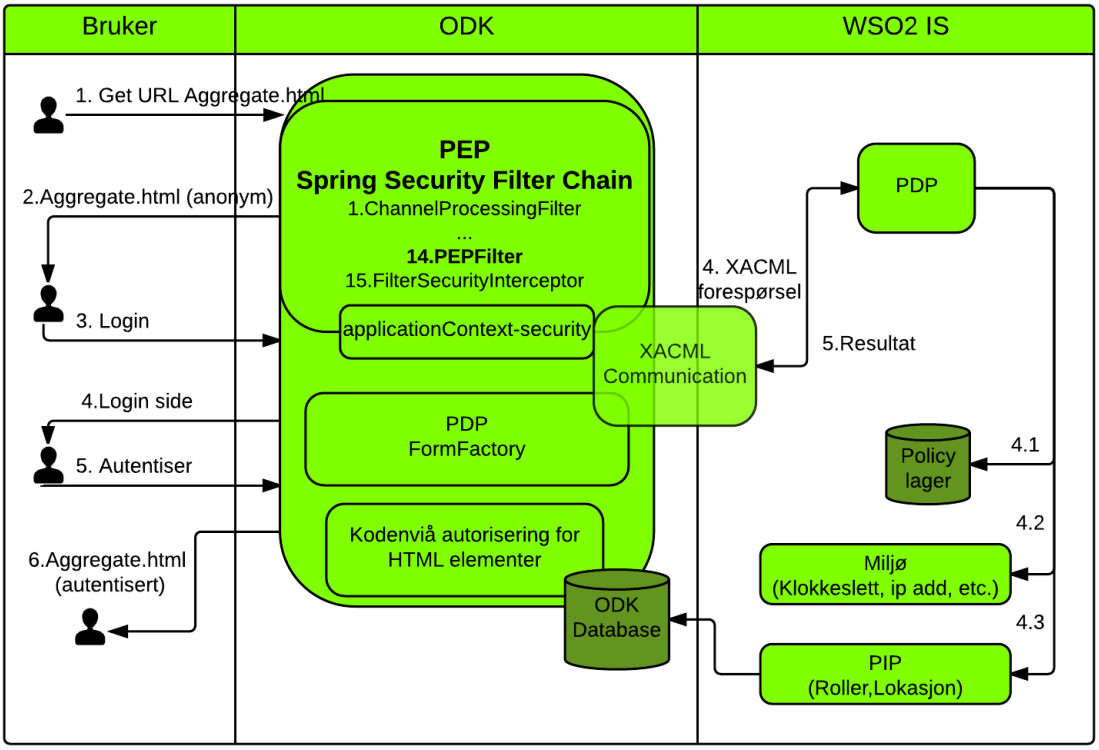
Fra seksjon 5.1.0.1 ble den gamle konfigurasjonsfilen forklart. Autorisasjonsløsningen var statisk og rollene hardkodet. Med ABAC implementert i systemet ble konfigurasjonsfilen endret til å kreve autentisert bruker istedenfor roller slik som Eksempel 5.4 viser. Autorisasjonsgrunnlaget har blitt flyttet til WSO_2 IS og gir ODK Aggregate en større fleksibilitet enn det var før. Eksempel 5.4 viser at noen av URLene krever tilgang anonym eller autentisert bruker. Fra Eksempel 5.1 trenger `/aggregateui/preferenceservice` URLen, rollen `USER`. En anonym bruker har rollen `USER` og det vil si fra den gamle autorisering vil en anonym bruker se det samme som en innlogget bruker.

```
1 <intercept-url pattern="/aggregateui/preferenceservice"
2                access="isAnonymous() or isAuthenticated()" />
3 <intercept-url pattern="/aggregateui/securityservice"
4                access="isAnonymous() or isAuthenticated() " />
5 <intercept-url pattern="/aggregateui/filterservice"
6                access="isAnonymous() or isAuthenticated()" />
```

Eksempel 5.4: Utsnitt fra oppdatert `applicationContext-security.xml` fil.

5.3.3 Autentisering og autorisasjon med ABAC

Autentiseringen til ODK Aggregate ble ikke endret og fungerer som før. Figur 5.9 viser at i ODK Aggregate er det lagt til to PEP. En for alle URLer og en for skjemaer. Konfigurasjonsfilene ble endret slik som beskrevet i seksjon 5.3.2. WSO_2 IS ble satt som PDP. Det er den samme instansen som OpenMRS bruker. Her ble det også laget et PIP som har tilgang til databasen til ODK Aggregate.

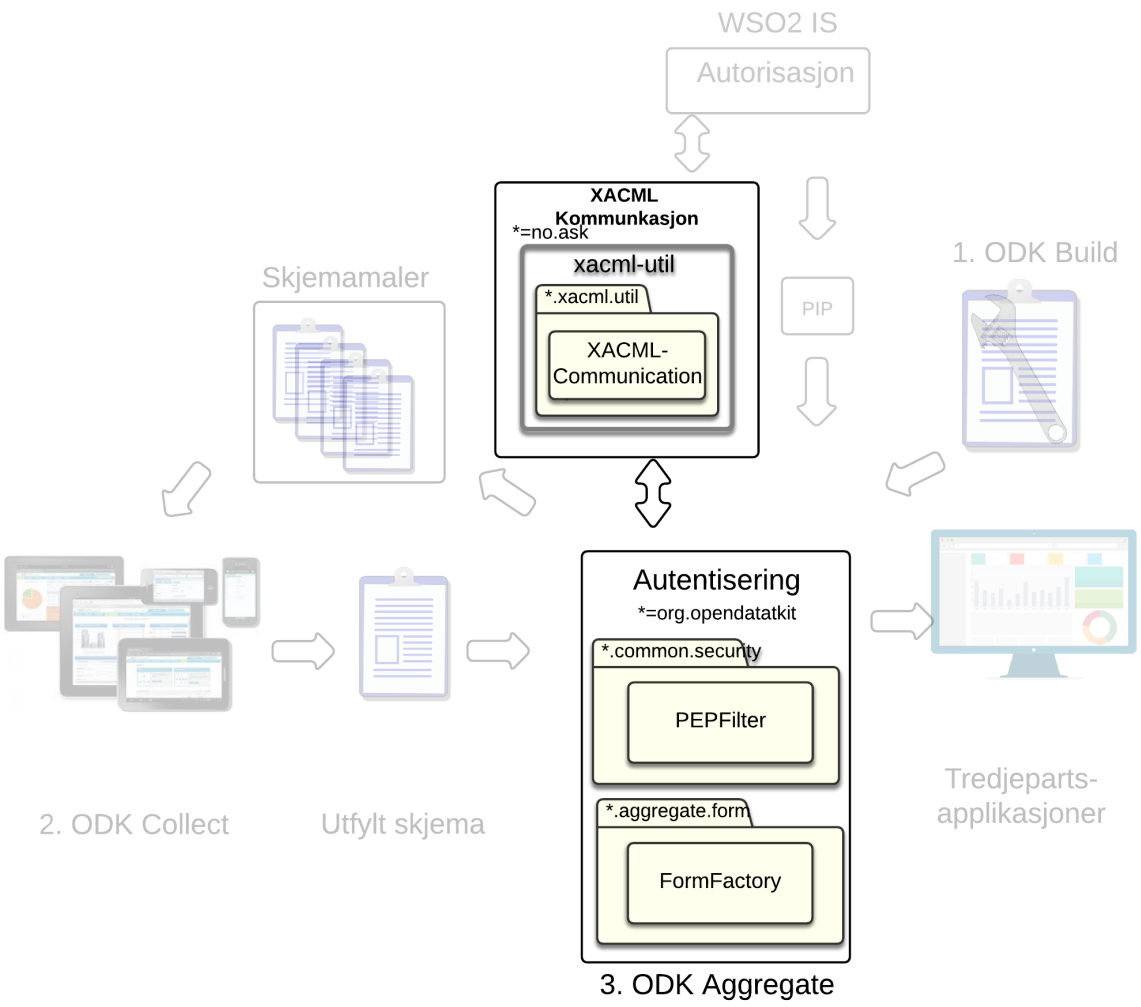


Figur 5.9: Kommunikasjonen mellom ODK Aggregate og WSO_2 IS.

5.3.4 Plassering av PEP, PIP og PDP

Plassering av PEP ble gjort to plasser. Figur 5.10 viser at det ble laget en `PEPFilter` klasse i sikkerhetspakken, og PEP i `FormFactory` klassen. Planen var å plassere PEP for autorisasjonen på fanene til websiden men det var ikke mulig å gjennomføre på grunn av begrensinger i GWT. `XACMLCommunication` klassen er den samme som brukes for OpenMRS. Klassen brukes for å kommuniserer med WSO_2 IS og ODK Aggregate applikasjonen.

`PEPFilter` klassen blir kjørt hver gang en forespørsel fra serveren kjøres. Den brukes som et PEP og behandler resultatet fra PDP. Figur 5.8 illustrerer hvor filteret ble plassert i filterkjedet. Hvis en forespørsel får `Deny` sender metoden respons `401` som er uautorisert.



Figur 5.10: Oversikt over implementasjonen av ABAC i ODK Aggregate

ODK Aggregate er sentret rundt skjemaer. Dagens løsning gir brukerne mulighet til å se alle skjemaene som er tilgjengelig. Ved å innføre et PEP i `FormFactory` klassen og lage en koblingstabell til databasen vil en kunne filtrere skjemaer. Reglene for å filtrere skjemaer lages som XACML policyer. Dette gir en dynamisk skjemabegrensing.

Eksempel 5.5 viser utsnitt av klassen `FormFactory`. Fra seksjon 5.2 ble det nevnt at metodene `getForm()` og `getForms()` manglet autorisasjon. Implementasjonen av ABAC har gitt autorisasjon til metodene. Metoden `getAllowedForms()` (linje 16) tar imot en

liste med skjemaer og sjekker om brukeren har tilgang til dem. Dette gjøres ved å spørre PDP. Dette vil også beskytte koden for *MFLAC* som er en av OWASP sine sikkerhetsfeil.

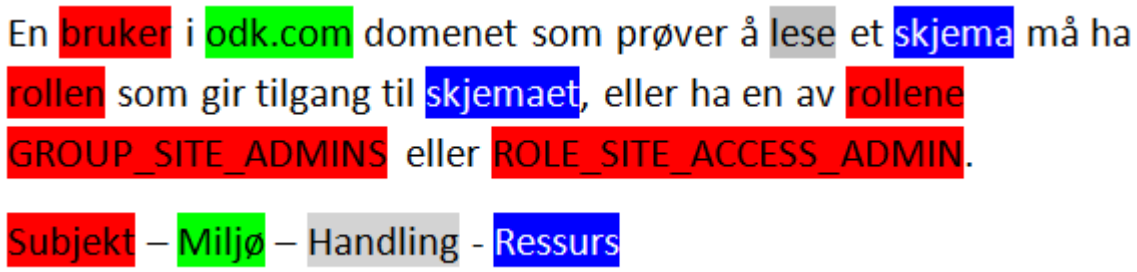
PIP og PDP implementasjonene er veldig lik implementasjonene fra OpenMRS. Forskjellene er at PIP kobler seg til ODK Aggregate sin database og bruker attributtene `role` og `form`. PDP implementasjonen er den samme som brukes i OpenMRS, men det er ikke den samme policyen.

```
1  private static IForm getForm(String topLevelAuri, CallingContext cc) throws
2  ODKOverQuotaException, ODKEntityNotFoundException, ODKDatastoreException {
3   List<IForm> forms = getAllowedForms(cc,internalGetForms(topLevelAuri, cc));
4   if (forms.isEmpty())
5   throw new ODKEntityNotFoundException
6    ("Could not retrieve form uri: " + topLevelAuri);
7   return forms.get(0);
8  }
9  public static final List<IForm>
10  getForms(boolean checkAuthorization, CallingContext cc)
11 throws ODKOverQuotaException, ODKDatastoreException {
12  return getAllowedForms(cc, internalGetForms(null, cc));
13 }
14 //Finds the available forms and checks if the user has access
15 //to it, and returns the available forms.
16 private static List<IForm>
17  getAllowedForms(CallingContext cc, List<IForm> tempForms) {
18  List<IForm> forms = new ArrayList<IForm>();
19  for (IForm form : tempForms) {
20   String environment = "odk.com";
21   List<String> decisonResults;
22   try {
23    ArrayList<String> actions = new ArrayList<String>();
24    actions.add("Read");
25    decisonResults =
26    FormServiceImpl.xacml.getDecisionResults
27    (cc.getCurrentUser().getUriUser(), actions, environment, form.getFormId());
28
29    if (!decisonResults.isEmpty() &&
30     decisonResults.get(0).equals(XACMLCommunication.RESULT_PERMIT)) {
31         forms.add(form);
32     logger.debug(cc.getCurrentUser().getUriUser() + " -- " + form.getFormId());
33   }} catch (Exception e) {   e.printStackTrace();  }
34  }
35  return forms;
36 }
```

Eksempel 5.5: Implementasjon av ABAC i ODK Aggregate.

5.3.5 XACML policy

Policyene som ble laget er basert på tilgangen slik den var. En fordel er at rollene ikke er direkte knyttet til URLer lengre. Det vil si at en kan skifte URL rolle kombinasjon i XACML policyen på serveren. Det ble lagt til nye regler i policyen for å imøtekomme autorisasjon på skjemaer. Figur 5.11 viser hvordan en bruker kan få begrenset tilgang til et skjema. Regelen krever at brukeren har rollen som tilhører skjemaet vedkommende sender inn.



Figur 5.11: Hvordan en XACML regel kan knyttes opp til attributtene.

Eksempel 5.6 viser hvordan regelen beskrevet i Figur 5.11 ser ut som et XACML policy. Policyen er bygget opp likt som policyen beskrevet i seksjon 4.3.4.1.

- Linje 2-5 starter med en valgfri beskrivelse av regelen.
- Linje 6-19 definerer målet som er `Read`.
- Linje 20-47 sier at en av to tilstander må godkjennes.
- Linje 22-33 Den første tilstand er om brukeren har tilgang til skjemaet som sendes inn. Et regulært utrykk beskriver hva brukeren må ha av attributter. For dette eksempelet er det eksplisitt definert hva skjemaet heter. Hvis applikasjonene skulle brukes i produksjon ville skjemaene fulgt et prefiks slik at en ikke trenger å legge til eksplisitte skjemaer for hver gang et nytt skjema blir lagt til.
- Linje 33-46 Hvis ikke den første *tilstanden* blir validert blir neste *tilstand* testet om brukerne har en av administratorrollene.

```
<Rule RuleId="Forms_get" Effect="Permit">
 <Description>
 A user who tries to read a form needs to have access to the form or
 have the role GROUP_SITE_ADMINS|ROLE_SITE_ACCESS_ADMIN
 </Description>
 <Target>
  <AnyOf>
   <AllOf>
    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
     <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">Read</AttributeValue>
     <AttributeDesignator MustBePresent="false"
     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
     DataType="http://www.w3.org/2001/XMLSchema#string"></AttributeDesignator>
    </Match>
   </AllOf>
  </AnyOf>
 </Target>
 <Condition>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
    <Function
     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
    </Function>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    ^(nurse|patient|doctor|anonym|birds)</AttributeValue>
    <AttributeDesignator
    Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="http://odk.com/id/form"
    DataType="http://www.w3.org/2001/XMLSchema#string"
    MustBePresent="false"></AttributeDesignator>
   </Apply>
   <Apply
   FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
    <Function
     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
    </Function>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    (GROUP_SITE_ADMINS|ROLE_SITE_ACCESS_ADMIN)</AttributeValue>
    <AttributeDesignator
    Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
    AttributeId="http://odk.com/id/role"
    DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false">
    </AttributeDesignator>
   </Apply>  </Apply>
 </Condition>
</Rule>
```

Eksempel 5.6: Utsnitt av en XACML policy fra ODK Aggregate.

5.4 Sammenligning av RBAC og ABAC løsningen i ODK Aggregate

ODK Aggregate original løsning er statisk og gir ikke mulighet for å spesifisere eller endre på rollene. Med den nye løsningen er rollene gjort om til attributter, og det er mulig å endre hvilke roller man trenger for å få tilgang til URLene. Løsningen kan også dra nytte av ABAC sin fleksibilitet ved å benytte miljøvariabler som for eksempel klokkeslett. Attributtene og autorisasjonsgrunnlaget kan endres under kjøring uten at brukeren trenger å autentisere seg på nytt. Dette bekrefter delvis hypotese nummer tre, ABAC gir mer dynamisk tilgangskontroll for URL-nivå autorisasjon, men ikke for kodenivåautorisasjon.

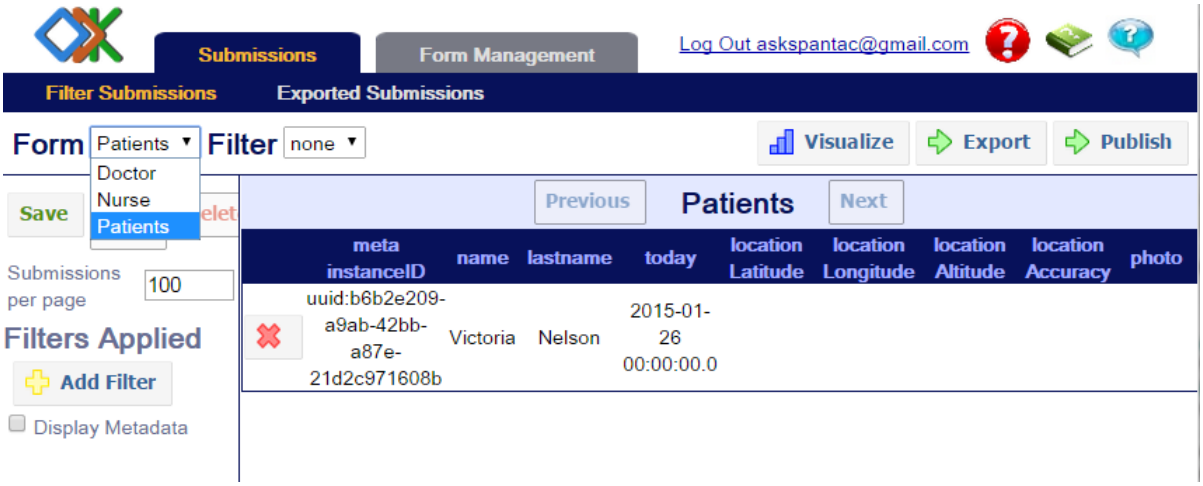
Skjemaene i ODK Aggregate hadde ingen begrensinger noe som gjorde at anonyme brukere kunne få tilgang til alle skjemaene slik som Figur 5.12 viser.

Figur 5.12: En anonym bruker kan se alle skjemaene i ODK Aggregate før ABAC ble implementert.

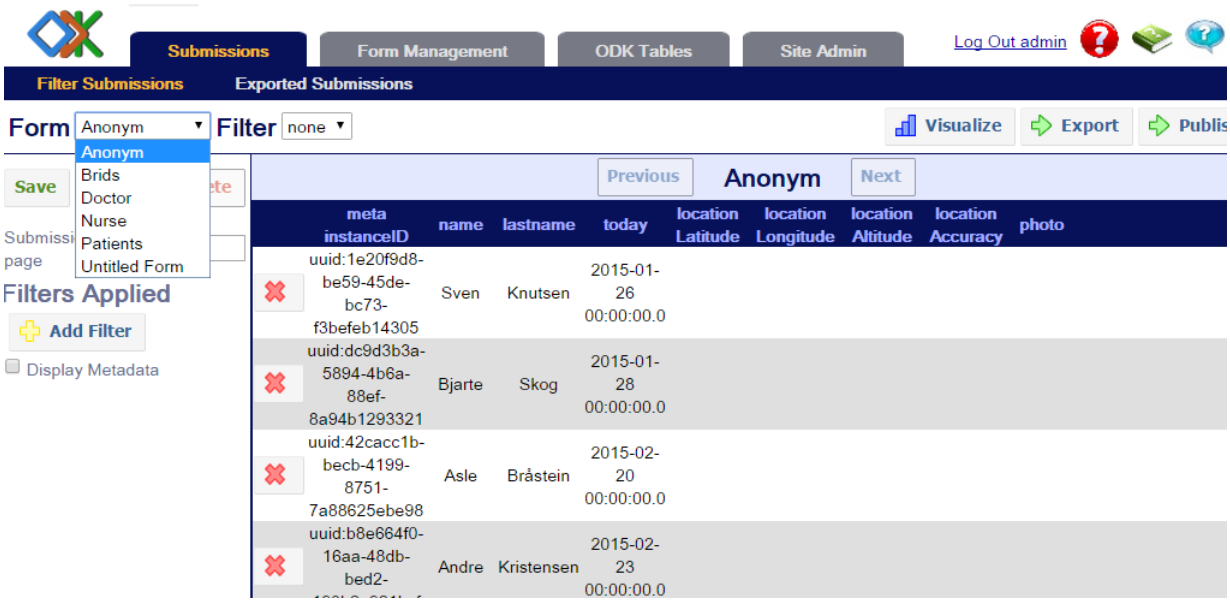
Med den nye løsningen ble det laget en policy som fungerer som en kobling mellom roller og skjemaer. Nå kan det sorteres på hvem som kan se å behandle skjemaene. Figur 5.13 viser at etter ABAC ble implementert kan ikke anonyme brukere se alle skjemaene.

Figur 5.13: En anonym bruker kan se utvalgt skjema i ODK Aggregate.

En autentisert bruker har tilgang til flere skjemaer. Figur 5.14 viser at en autentisert bruker med grupperollen `Group Form Managers` har tilgang til `nurse`, `doctor` og `patients` skjemaene. En administrator får full tilgang til alle skjemaene slik som Figur 5.15 viser. Dette bekrefter hypotese nummer fire; med ABAC er det mulig å filtrere skjemaene med rolleattributter.



Figur 5.14: En innlogget bruker kan se utvalgte skjemaer i ODK Aggregate.



Figur 5.15: En administrator kan se alle skjemaene i ODK Aggregate.

5.5 Evaluering av utfordringer med implementasjonen

Utfordringene i applikasjonene ble delvis løst, men på grunn av begrensinger i GWT ble ikke alt løst.

- Hvordan kan man beholde eksisterende autorisasjon med å bytte fra rollebasert til attributtbasert tilgangskontroll?
  - For å beholde eksisterende autorisasjonsgrunnlag ble XACML policyen modellert til å være lik som autorisasjonen var for URLene.
- Har de samlet autorisasjonen på noen plasser eller er den spredd utover hele koden?
  - Autorisasjonsavgjørelsen er spredt. Spring Security Filter Chain er en definert plass, men for alle komponentene til siden som knapper og faner er autorisasjonen spredt rundt i koden.
- Hvordan kan man løsrive seg fra hardkodede roller i koden?
  - Autorisasjonen som er sikret med Spring Security er enkel å bytte ut med dynamiske roller, men autorisasjonene som var spredt rundt i koden var ikke

mulig å endre på. Dette var på grunn av begrensinger med GWT som ikke kan bruke hvilke som helst biblioteker på klientsiden, og heller ikke kunne bruke synkrone kall[61].

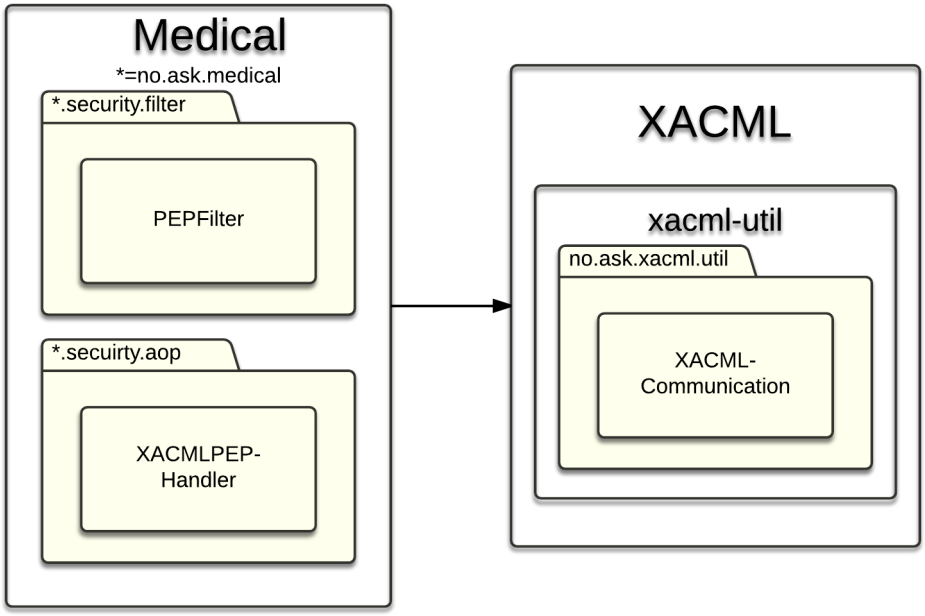
6 Proof of Concept: Medical prosjekt

6.1 Motivasjon for å lage en PoC løsning

Prosjektet ble laget på bakgrunn av begrensninger i implementasjonene av ABAC i OpenMRS og ODK Aggregate. OpenMRS har begrensninger med rolleprivilegiumstrukturen. ODK Aggregate har begrensinger med fragmenter i URL, måten de henter skjema på og at autorisasjonen er spredt rundt i koden. Erfaringer tatt fra implementasjonen ble inspirasjonen til å forsøke å lage en bedre ABAC løsning. Prosjektet fikk navnet Medical.

6.2 Utvikling av Medical

Hovedkomponentene i Medical prosjektet er Spring- Model View Controller (MVC), Security og Java Persistence API (JPA). Prosjektet ble utviklet med Domain Driven Design (DDD)[62] og MVC[63] arkitekturen. Figur 6.1 viser en oversikt over hvor PEP ble plassert i Medical prosjektet. Strukturen er veldig lik som implementasjonen til ODK Aggregate. `PEPFilter` klassen ble plassert i filterkjedet på samme plass som ODK Aggregate. Dette fordi det er den optimale plassen å plassere førstelagsbeskyttelse på. Det som skiller seg ut er `XACMLPEPHandler` klassen som er et aspekt[64]. Den tilhører ikke en klasse men alle metoder som blir annotert med `@PEP(action)` annotasjon.



Figur 6.1: Oversikt over implementasjon av PEP.

6.2.1 Valg av autorisasjons metode med Spring Rammeverket

I artikkelen *Attribute Based Access Control for APIs in Spring Security* [25] brukes Spring Security med Spring Expression Language (SpEL) for attributtbasert tilgangskontroll på metodenivå. Denne måten ble vurdert å bruke på Medical prosjektet, men den hardkoder hvilke tilganger den gir til metodene slik som i OpenMRS og ODK Aggregate. Ved å ikke bruke XACML kan ikke applikasjonen dra nytte av tilstandsløs og dynamisk autorisasjon. På denne bakgrunn ble ikke SpEL valgt.

6.2.2 PEPFilter

Kode Eksempel 6.1 er et utsnitt av `PEPFilter` klassen som implementerer `javax.servlet.Filter`. Klassen blir plassert i filterkjedet før `ExceptionTranslationFilter`. Filteret samler inn `brukernavn`, `handling`, `miljø` (domenet) og `ressurs`, før den sender det til WSO_2 IS serveren. Handlingene til filteret kan være `GET` eller `POST`. Ressursen er URLen som det blir spurt om. Påfølgende steg forklarer de viktigste elementene i eksemplet:

- Linje 6 viser at metoden henter ressursen som er URLen som blir sendt inn.
- Linje 7 får tak i den autentiserte brukeren.
- Linje 12 legger til handlingen. Handlingen er metoden fra forespørselen, dvs. at det kan være en `GET` eller `POST` forespørsel.
- Linje 14 viser metoden som gjør en XACML forespørsel.
- Linje 17-23 avgjør om brukeren har tilgang ut i fra resultatet fra XACML forespørselen. Hvis resultatet er `Denied` vil metoden sende en feilmelding med koden `401` ikke-autentisert.
- Linje 27 sender brukeren videre til neste filter.

```
1  @Override
2  public void doFilter(ServletRequest request, ServletResponse
3  response, FilterChain chain) throws IOException, ServletException {
4   FilterInvocation filterInvocation =
5    new FilterInvocation(request, response, chain);
6   String requestUrl = filterInvocation.getRequestUrl();
7   Authentication auth =
8    SecurityContextHolder.getContext().getAuthentication();
9   List<String> decisonResults;
10  try {
11  ArrayList<String> actions = new ArrayList<String>();
12  actions.add(filterInvocation.getRequest().getMethod());
13
14  decisonResults =
15   xacml.getDecisionResults(auth.getName(), actions, environment, requestUrl);
16
17  if (!decisonResults.isEmpty() &&
18   !decisonResults.get(0).equals(XACMLCommunication.RESULT_PERMIT)) {
19   log.info(requestUrl + " " + decisonResults);
20   ((HttpServletResponse) response)
21   .sendError(HttpServletResponse.SC_UNAUTHORIZED, decisonResults.isEmpty() ?
22    "empty" : decisonResults.get(0) + " For url " + requestUrl);
23  }
24  } catch (NullPointerException | PEPAgentException | XMLStreamException e) {
25   e.printStackTrace();
26  }
27  chain.doFilter(request, response);
28 }
```

Eksempel 6.1: `doFilter` metoden til `PEPFilter` klassen.

6.2.3 Spring Security Java konfigurasjon

I den nyeste versjonen av Spring Security har de endret konfigurasjonsfilen fra å være XML til å bli Java. Dette har medført at den er blitt enklere å konfigurere, og den har fått en bedre struktur. Eksempel 6.2 viser hvordan den nye konfigurasjonen kan se ut. Listen forklarer hva klassen og metoden gjør:

- Linje 1-5 Er annotasjoner til klassen. Den sier at klassen aktiverer websikkerhet, hvilke pakker den skal lete etter Aspekt-klasser, og hvor den skal hente konfigurasjonsfilen.
- Linje 8 sier at *Cross-Site Request Forgery (CSRF)* er slått av. Man vil ha den på i større applikasjoner, men for dette prosjektet er den ikke i bruk.
- Linje 9-10 sier hvilke URLer som skal beskyttes. I denne applikasjonen skal alle forespørsler være autentiserte.
- Linje 11 sier hvilken autentisering den skal bruke. Siden det er en PoC applikasjonen og autentisering ikke er i fokus blir standard konfigurasjonen brukt.
- Linje 12 sier hvilke ekstra filter som skal legges til. I vår tilfelle er det PEP Filteret som blir lagt til.
- Linje 13 sier at `jsessionid` blir slettet når brukeren logger seg ut.

```
1  @Configuration
2  @EnableWebSecurity
3  @ComponentScan(basePackages = { "no.ask.medical.secuirty.aop"})
4  @EnableAspectJAutoProxy
5  @PropertySource(value = "classpath:/application.properties")
6  public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
7   protected void configure(HttpSecurity http) throws Exception {
8    http.csrf().disable()
9        .authorizeRequests()
10       .anyRequest().authenticated()
11       .and().httpBasic()
12       .and().addFilterAfter(pepFilter(), ExceptionTranslationFilter.class)
13       .logout().deleteCookies("jsessionid");
14  }
15 }
```

Eksempel 6.2: Spring Security Java konfigurasjon.

6.2.4 @PEP(action) annotasjon

Eksempel 6.3 viser hvordan `PEP`-annotasjonen kan brukes på metoder. Annotasjonen inneholder hvilken handling den skal utføre. I motsetning til OpenMRS, kan annotasjonen ikke ha flere handlinger. Dette er fordi den skal være i tråd med Single Responsibility Principle (SRP)[65] som sier at en metode skal kun gjøre en ting. Eksemplet er et utsnitt av klassen `PatientService`. Listen forklarer hva som vises i klassen:

- Linje 1 sier at klassen er en komponent, dvs. at klassen kan finnes av Spring rammeverket for å bli brukt i andre klasser.
- Linje 5-8 fra eksempelet viser hvordan annotasjonen kan brukes med å gi den en handling. Handlingen er `READ` og metoden leverer ut en itererbar liste med pasienter.
- Linje 10-13 metoden leverer ut en pasient med en gitt identifikator.

```
1  @Component
2  public class PatientService {
3  ...
4
5   @PEP(action = PEPActions.READ)
6   public Iterable<Patient> readAllPatients() {
7    return repo.findAll();
8   }
9
10  @PEP(action = PEPActions.READ)
11  public Patient readPatient(Long id) {
12   return repo.findOne(id);
13  }
14  ...
15 }
```

Eksempel 6.3: Utsnitt av klassen `PatientService` som bruker `@PEP(action)` annotasjoner.

6.2.5 XACMLPEPHandler klassen

`XACMLPEPHandler` er en klasse som er annotert med `@Aspect` annotasjon. Den har en metode som Eksempel 6.4 viser. Metoden kjører før eksekvering av alle metoder som er annotert med `@PEP`. Koden henter argumenter fra metoden som er annotert. Linje

14 viser at den sender en XACML forespørsel. Hvis forespørselen ikke blir godkjent så kaster klassen et `PEPExectpion`, og metoden som ble kjørt blir stoppet og brukeren får ikke tilgang til ressursen.

```
@Before("execution(* *(..)) && @annotation(pep)")
public void pep(JoinPoint jp, PEP pep) throws Throwable {
 Object[] args = jp.getArgs();
 CodeSignature signature = (CodeSignature) jp.getSignature();
 String[] parameterNames = signature.getParameterNames();

 Authentication auth = SecurityContextHolder.getContext().getAuthentication();
 ArrayList<String> actions = new ArrayList<String>();
 List<String> decisonResults = new ArrayList<String>();
 actions.add(pep.action());
 String resource =  (parameterNames.length > 0
  && parameterNames[0].contains("id"))
  ? args[0] + "" : jp.getSignature().toShortString();
 decisonResults =
  xacml.getDecisionResults(auth.getName(), actions, environment, resource);

 if (!decisonResults.isEmpty()  &&
  !decisonResults.get(0).equals(XACMLCommunication.RESULT_PERMIT)) {
   throw new PEPException("XACML respons is " + decisonResults.toString());
 }
}
```

Eksempel 6.4: `pep` metoden til `XACMLPEPHandler` klassen.

6.2.6 Sikkerhetsflyten i Medical

Sikkerhetsflyten i Medical er vist på Figur 6.2. Utformingen er en kombinasjon av ODK Aggregate og OpenMRS.

1. En bruker spør etter ressursen som ligger bak URLen `medical/patient/id/1`
2. Forespørselen går gjennom Spring Security Filter Chain og finner ut at det er en anonym bruker. Filteret sender brukeren til innlogging siden.
3. Brukeren autentiserer seg.
4. Deretter går forespørselen igjennom `PEPFilter` klassen for å få tilgang til siden. Hvis metoden som blir kalt fra servicelaget har annotasjonen `@PEP` vil `XACMLPEPHandler` klassen bli kjørt, og det vil bli sendt en ny XACML forespørsel. PDP behandler forespørselen fra `PEPFilter` og `XACMLPEPHandler` klassen.

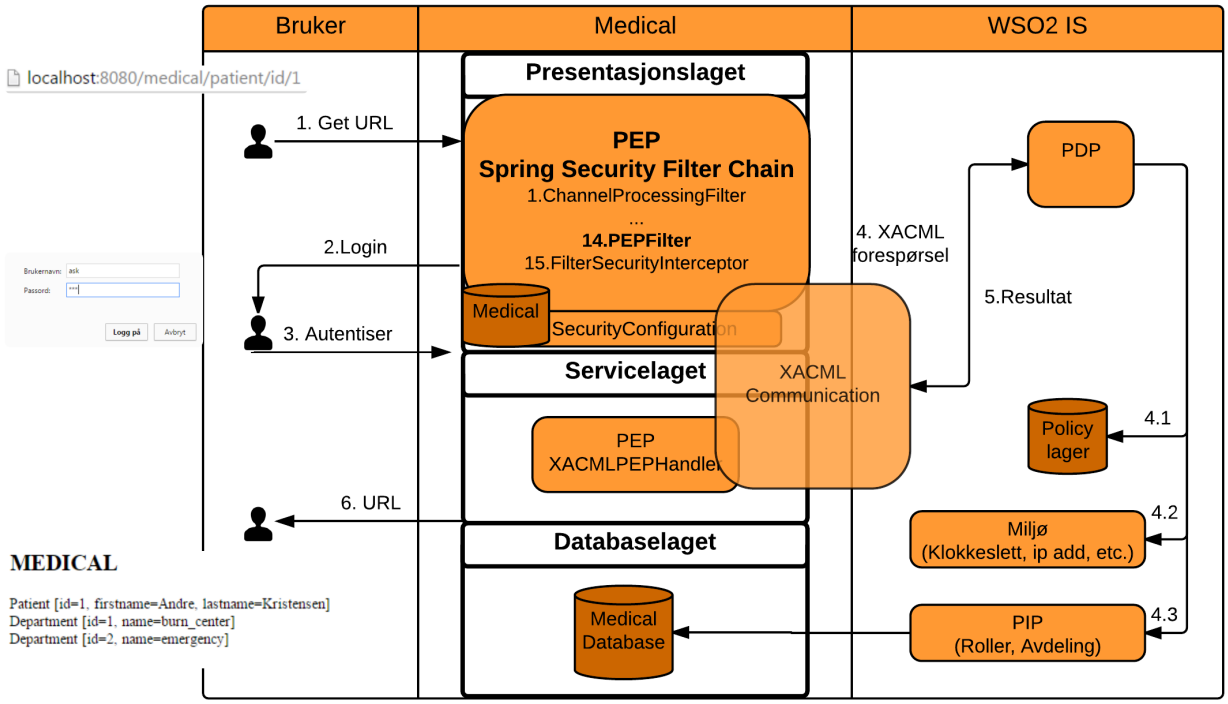
4.1. PDP henter den rette policyen.

4.2. Eventuelle miljøvariabler blir hentet.

4.3. PIP blir kjørt hvis policyen trenger å finne rolle eller avdeling. PIP finner brukerens roller og avdelinger og sender dem tilbake til PDP.

5. PDP Sender svaret tilbake til PEP som utfører korrekt handling i henhold til respons.

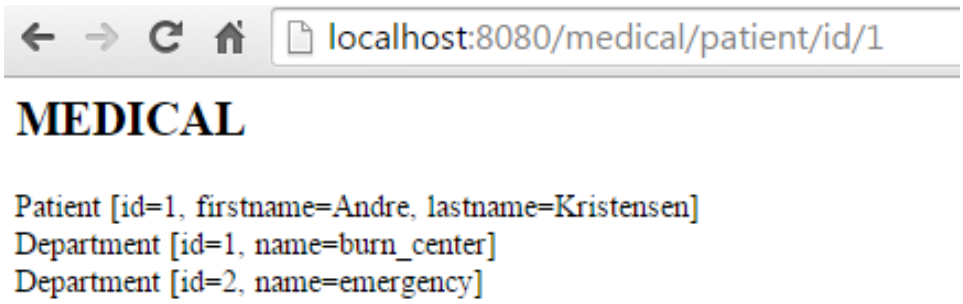
6. Brukeren får tilbake siden med pasientdata.



Figur 6.2: Sikkerhetsflyten i Medical.

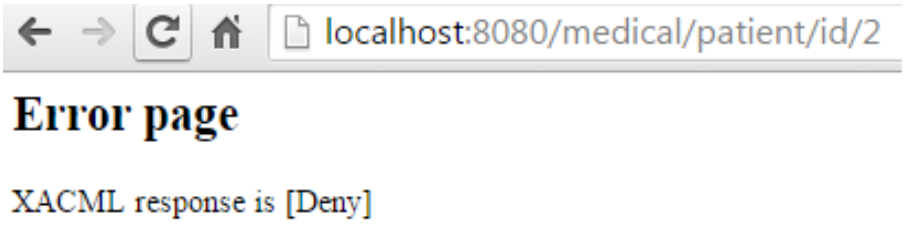
6.2.7 Funksjonaliteten i Medical applikasjonen

I Medical applikasjonene kan en bruker søke på pasient med `id`. Figur 6.3 viser hva som skjer når en XACML forespørsel blir tillatt. URLen som blir brukt representerer ressursen som brukeren skal ha tak i. En pasient blir vist med hvilke avdelinger vedkommende har blitt behandlet på, hvis brukeren har tilgang til avdelingen.



Figur 6.3: En pasient med tilhørighet i to avdelinger.

Figur 6.4 viser hva som skjer når en XACML forespørsel blir nektet. Brukeren blir sendt til en feilside og får vite hvilken XACML respons vedkommende fikk.



Figur 6.4: En bruker prøver å få tilgang til en pasient utenfor sine avdelinger.

6.3 Evaluering av Medical prosjektet

Implementering av ABAC fra starten er en bedre fremgangsmåte fordi autorisasjonen har kommet med i designfasen, og den har blitt bedre integrert i prosjektet.

For å oppsummere så bruker Medical prosjektet det beste fra ODK Aggregate og OpenMRS. Fra ODK Aggregate blir det benyttet Spring Security Filter Chain, og det lages et

`PEPFilter` som avskjærer alle forespørsler. Fra OpenMRS blir det benyttet annotasjoner som i Medical er `@PEP`, og den benytter seg av generisk *CRUD* handlinger. Dette er en forbedring fra OpenMRS sin rolleprivilegiumstruktur. Det er også mulighet for å skille på lokasjon til pasientene slik som det ble implementert i OpenMRS.

Hypotesene som kan bekreftes av Medical prosjektet er følgende:

1. Ved implementasjon av ABAC i OpenMRS vil en kunne skille pasienter på behandlingsavdeling.
    - Med ABAC i Medical prosjektet kan en filtrere pasienter på behandlingsavdeling.
2. Ved implementasjon av ABAC i OpenMRS vil en kunne frigjøre rolleprivilegiumproblemet.
    - Rolleprivilegiumproblemet som OpenMRS ble unngått ved å benytte *CRUD* handlinger istedenfor 200 handlingene som OpenMRS benytter.
3. Ved implementasjon av ABAC i ODK Aggregate vil en kunne få dynamisk tilgangskontroll.
    - Her ble det benyttet Spring Security for URL-nivå beskyttelse og annotasjoner med aspekter for metodenivå.
4. Ved implementasjon av ABAC i ODK Aggregate vil en kunne begrense tilgang til skjemaene med rolle-attributter.
    - Den fjerde hypotesen er ikke gyldig for Medical prosjektet.

6.3.1 Designprinsipper

Kodelukter er symptomer i kildekoden som indikerer et dypere problem. I følge Fowler[66] er en kodelukt en indikasjon på overflaten som svarer til et dypere problem. Utrykket ble innført av Kent Beck[66]. En annen måte å se på kodelukter er med hensyn til designprinsipper. Kodelukter er strukturer som indikerer brudd på grunnleggende designprinsipper som negativt påvirker kvaliteten til koden[67]. Kodelukter er ikke tekniske feil, og de hindrer ikke funksjonaliteten til programmet. I stedet viser det svakheter i designet som kan bremse videreutvikling og øker risikoen for feil i fremtiden. Kodelukter kan være:

- Klipp og lim løsninger kan forekomme, og endringer i koden medfører at det må også endres der koden ble klippet fra. Utviklere følger ikke Don't repeat yourself (DRY) prinsippet som skal forhindre duplikat av kode. Ved å bruke annotasjoner slipper man å skrive de samme sikkerhetssjekkene for metodene som skal beskyttes.
- Lange metoder, store klasser og for mange parametere i metoden er indikatorer på at koden har kodelukter, dette strider også med Keep It Simple Stupid (KISS) prinsippet[68].
- Tett integrasjon med andre klasser (*high coupling*) medfører at koden er vanskelig å vedlikeholde. Endringer i en klasse har konsekvenser for andre klasser på grunn av avhengigheten[69].
- En annen kodelukt er at metoder gjør for mye (*high cohesion*), og strider med Single Responsibility Principle[69].

Kodelukter kan være små problemer som ikke er så farlige når de er isolert, men kombinasjonen av flere av dem skaper problemer. Det er derfor viktig å luke dem ut tidlig. Med ABAC, annotasjoner og filterkjede er man på god vei til å forhindre kodelukter.

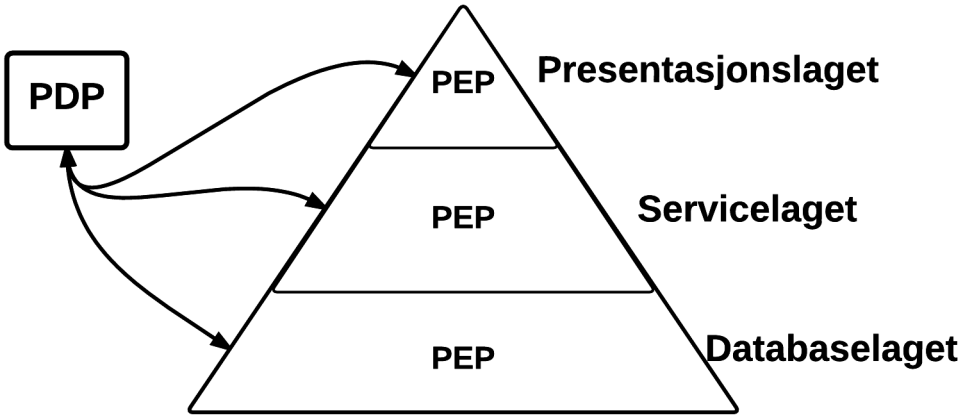
6.4 Sikkerhetsaspekter med løsningene

OpenMRS, ODK Aggregate og Medical benytter seg av sikker kommunikasjon mellom PDP og PEP via HTTPS. Bruk av ABAC i et prosjekt kan hjelpe med å forebygge sikkerhetsrisikoer som OWASP topp 10 listen[14] nevner. Følgende sikkerhetsrisikoer omhandler autorisasjon direkte og indirekte:

- *A4 Insecure Direct Object References*. Direkte tilgang til objekter som blir eksponert til brukeren uten å gjøre en sjekk om brukeren har lov til å utføre handlingene. Ved å benytte ABAC kan en sette opp skallbeskyttelse slik at ressurser man prøver å bruke blir stoppet av PEP.
- *A5 Security Misconfiguration*. Konfigurering av sikkerhet er viktig og standardinnstillingene er ikke gode nok. Den største utbredte sikkerhetsløsningen er RBAC og det finnes mange rammeverk som støtter denne løsningen. Dette kan medføre at sikkerheten kan bli feil konfigurert pga. en kan klippe og lime sikkerheten inn i applikasjonen. Med ABAC krever det bedre ferdigheter, kunnskap og tid for å implementere i en applikasjon. Det finnes ikke ferdige løsninger som kan brukes og

man kan ikke klippe og lime inn oppsett. Argumentet er at ved å bruke mer tid og tanke på oppsett av sikkerheten vil en få mindre feil og dermed få en bedre autorisasjon.

- *A7 Missing Function Level Access Control*. De fleste applikasjoner kjører med autorisasjon i presentasjonslaget. Med ABAC kan en enkelt implementere autorisasjon i alle lagene som det kreves. Figur 6.5 viser at det kan plasseres PEP i alle lagene og de kan kommunisere med samme PDP server.



Figur 6.5: Illustrerer hvor autorisasjon med ABAC kan plasseres i lagene.

7 Erfaringer, videre arbeid og konklusjon

7.1 Erfaringer

I denne seksjonen vil kandidaten gå gjennom erfaringene han sitter igjen med etter å ha jobbet med masteroppgaven.

7.1.1 Åpne kildekodeprosjekter

Kandidaten erfarer at åpne kildekodeprosjekter har varierende kvalitet på kode og dokumentasjon. Dette på grunn av at det er mange utviklere med ulik kompetanse, noe som gjenspeiler kvaliteten på prosjektene.

Kandidaten har også observert at sikkerhet ikke er i fokus i åpne kildekodeprosjekter. Prosjektene mangler oversikt og dokumentasjon av sikkerheten. Dette gjelder en generell oversikt over hvilke rammeverk som er brukt og hvilke metoder de har brukt for å implementere sikkerheten, samt i hvilke lag i applikasjonen sikkerheten er plassert. Det oppleves heller ikke som sikkerhet har vært en del av designet. Prosjektene hadde vært mye bedre med en klar oversikt over ressursene som applikasjonen beskytter og hvilken autorisasjon man trenger for å få tilgang til dem.

Prosjekter uten god oversikt over sikkerheten gir en falsk trygget, og man får noe som på fagspråket blir kalt *security by obscurity*[70]. Det vil si at sikkerheten baserer seg på at det ikke er oversiktlig å se hvordan ressursene er beskyttet.

Kandidaten har også erfart at åpne kildekodeprosjekter, og for så vidt andre prosjekter som er utviklet over flere år, ikke oppdaterer bibliotekene til siste versjon. Dette er bekymringsverdig aspekt som medfører at prosjektene kan være sårbare for sikkerhetshull som er blitt oppdatert i nyere versjoner av bibliotekene.

For at integrasjon i eksisterende prosjekter skal gå bra, er det en del ting som må på plass for implementasjonen. Oppsett av utviklingsmiljøet krever en del arbeid, og hvert prosjekt har sin måte å sette det opp på. Å implementere i et tredjepartssystem krever god forståelse av programmering og en må kunne sette seg inn i hva utviklerne har tenkt.

7.1.1.1 OpenMRS

OpenMRS prosjektet har en aktiv wiki-side som brukes for å hjelpe utviklere til å komme i gang med å implementere i prosjektet. På wiki-siden er det en seksjon som omhandler sikkerhet og tilgangskontroll. Seksjonen forklarer hva de ønsker av funksjonalitet og hva de har prøvd av sikkerhetskontroller. Den sier ikke noe om den tekniske løsningen.

OpenMRS tilbyr en utviklerguide som gir en steg for steg løsning hvordan det skal settes opp. Oppsett av utviklingsmiljø til OpenMRS var tidkrevende, men på grunn av god dokumentasjon gikk det bra. Siden OpenMRS er et Maven prosjekt var det enkelt å legge til nye biblioteker.

Bruken av kjente rammeverk er viktig for at nye utviklere skal kunne ta i bruk prosjektet. OpenMRS benytter seg av en egendefinert sikkerhetsløsning spredt over to moduler noe som gjorde at det krevde en del arbeid å kartlegge flaskehalsene i OpenMRS.

7.1.1.2 ODK Aggregate

ODK Aggregate har ikke noe aktiv wiki-side, men de har en epostliste en kan melde seg opp på for å få informasjon og stille spørsmål. Prosjektet tilbyr en tekstfil som gir steg for steg oppsett av ODK Aggregate. Den gir også mulighet for å sette det opp med Google App Engine (GAE) eller Tomcat6 med MySQL eller PostgreSQL som databasemotor. MySQL versjonen av ODK Aggregate bruker ikke Maven. Dette gjorde det vanskeligere å legge til biblioteker og holde dem oppdatert. ODK Aggregate benytter seg av kjente rammeverk og sikkerhetsløsninger. Dette gjorde det enkelt å modifisere eksisterende løsning.

Kandidaten gjorde en viktig erfaring under utvikling av ODK Aggregate, og det er hvilke begrensinger valg av rammeverk har å si for applikasjonen. Tett integrering med bestemte rammeverk skaper problemer for videreutvikling når rammeverket møter sin funksjonsgrense. Dvs. når problemer oppstår og rammeverket ikke klarer å løse problemet. Det er oftest ikke mulig å bytte rammeverk fordi resten av applikasjonen er avhengig av det.

7.1.1.3 WSO_2 IS

WSO_2 IS er et omfattende system med mye funksjonalitet. Prosjektet er godt dokumentert og det er en god blogg som hjelper deg med å lage utvidelser som PIP. WSO_2 IS har et aktivt miljø, og det er enkelt å komme i dialog med utviklerne i prosjektet. Dette erfarte kandidaten da han fant en feil med miljøvariablene i WSO_2 IS. Etter et innlegg på stackoverflow.com og en mail til utvikleren fikk kandidaten en oppdatering som kunne legges til for å fikse problemet. Kandidaten har ikke erfart noe lignende support før og fikk en positiv opplevelse med prosessen.

Kandidaten fikk også erfare hvor viktig logging av kode er for debugging. PIP som ble utviklet til WSO_2 IS ble kompilert og lagt til som en utvidelse. For at kandidaten kunne se om koden virket var logging i PIP klassen essensielt.

7.1.2 ABAC og XACML

Før kandidaten startet på oppgaven hadde han ingen tidligere erfaringer med ABAC eller XACML. Konseptet til ABAC er relativt enkelt å forstå teoretisk, men i praksis er det vanskeligere. Artikler som nevner ABAC får det til å virke enkelt, og det er ikke noe problem å ta det i bruk i prosjekter. Slik er det ikke i virkeligheten. Dette er fordi det er mange variabler som må konfigureres. F.eks. avhengig av hvilket PDP man benytter, finnes det forskjellige måter å kommunisere med applikasjonen på. Man må bestemme hvordan policyer skal behandles og vedlikeholdes. Hvor i koden er det lurt å plassere PEP? Må en lage et PIP for å få attributtene som en ønsker, eller er det rammeverket som kan lage dem? Slike variabler blir ofte ikke diskutert i artikler som er publisert. Diskusjonene går ofte ut på hvor smidig tilgangskontrollen kan bli ved å benytte ABAC. Artiklene mangler ofte en teknisk vurdering hvor fordeler og ulemper med ulike valg blir validert opp mot hverandre.

Kandidatens erfaring er at det er høy oppstartsbarriere med XACML. Det er vanskelig

å skrive policyer samt å forstå hvordan alle komponentene henger sammen. XACML krever tid og forståelse for at det skal kunne være effektivt. Men det blir enklere når man har kommet seg over oppstartsbarrieren og forstår gangen i XACML. Videreutvikling av kode etter at en bruker XACML og ABAC er eklere for utviklere. Utviklere trenger ikke å tenke på at koden må tilgrises med autorisasjon.

For at XACML skal fungere optimalt trenger den en aktiv administrator som følger med på om det skjer endringer i omverden som medfører endret trusselbilde, og dermed endringer på policyer.

Det finnes verktøy for å skrive XACML policyer. WSO_2 IS har et brukergrensesnitt for å skrive enkle policyer. Skal det skrives mer avanserte regler må det kodes. Axiomatic har et verktøy som heter ALAF[71] som er en utvidelse til Eclipse. ALFA er et eget språk som kompilerer til XACML policyer. Erfaringen med ALFA er at den kompilerer til en Axiomatic-variant av XACML, og en må bytte prefikser og attributter for å kompilere på andre XACML motorer.

7.1.3 Andre teknologier og rammeverk brukt i oppgaven

I prosjektet ble det brukt ulike utviklingsverktøy og rammeverk. I denne seksjonen oppsummerer kandidaten erfaringer med de viktigste verktøyene for oppgaven.

Kandidaten fikk erfare hvordan en skal bruke annotasjoner sammen med aspekter. OpenMRS prosjektet viser hvordan det kan gjøres med egendefinerte annotasjoner. På bakgrunn av erfaringene som ble gjort med annotasjoner i OpenMRS ble det også brukt i Medical prosjektet.

Erfaringene som ble gjort med GWT er negative. Kandidaten mener det et tungvint å skrive Java kode for så å kompilere det til JavaScript. Denne måten skaper unødvendige begrensninger og det er enklere å skrive JavaScript koden selv. Kandidaten fikk også erfare hvilke konsekvenser valg av et rammeverk kan ha, dette med tanke på asynkron kommunikasjon og hvilke begrensninger det ga.

Kandidaten har jobbet en del med Spring rammeverket før. Under utviklingen av Medical prosjektet fikk han erfare hvor bra rammeverket er. F.eks. *CRUD* operasjoner til databasen: før måtte en skrive alle metoder for å gjøre *CRUD* handlinger, men med siste versjon av Spring JPA rammeverket kan en klasse bare utvides (extends) med en klasse som heter `CrudRepository<T,ID extends Serializable>` og man trenger ikke å

skrive SQL-spørringer.

Lucidchart ble brukt til å lage de fleste figurene i masteroppgaven. Det er meget enkelt å lage figurer, flyt og handlingsdiagrammer med dette programmet.

Kandidaten hadde lite erfaring med LaTeX før prosjektet, men siden det følger en syntaks og kandidaten har bakgrunn i programmering, var det enkelt å ta i bruk. LaTeX har et meget godt referansesystem. Kandidaten har benyttet Microsoft Word sitt referansesystemet før. Etter kandidatens vurdering er LaTeX sitt referansesystem i en annen klasse. Utvikleren av LaTeX har virkelig forstått hvordan referansene skal gjøres enkelt.

En annen erfaring med Latex er at det krever mye arbeid med plassering av bilder. Her har Word en bedre løsning, ved at man kan styre plassering og størrelsen med musen, man slipper da ringvirkninger med at et bilde skyver resten av teksten og bildene i samme kapittel.

Metodikken som ble brukt til utvikling av Medical prosjektet er: SRP, DDD og MVC. SRP prinsippet hjelper med å forenkle metoder. Det ble også brukt DDD sammen med MVC arkitekturen. Arkitekturen gir en fin struktur over prosjekt og gjør det enklere for utviklere å utvide prosjektet. Kandidaten har god erfaring med utviklingsmetodikker og har erfart hvor bra det er å bruke kjente metodikker for at utviklingen skal gå smertefritt.

Kandidaten brukte og andre verktøy under utviklingen. Eclipse ble brukt til selve utviklingen, Maven, Subversion (SVN) og Git ble brukt som utvidelser til Eclipse. Maven ble brukt til å bygge prosjektene og holde styr på bibliotekene. SVN og GitHub ble brukt til versjonskontroll på koden og masteroppgaven. Verktøyene virket som tiltenkt.

Kandidaten har strevd med å finne norske ord på noen av de engelske begrepene som er brukt.

7.2 Videre arbeid

Siden prosjektet er et forskningsprosjekt som skal teste gjennomførbarhet av ABAC i elektroniske helsesystemer, vil en videreføring være å teste OpenMRS og ODK Aggregate i et produksjonsmiljø. Ytelsestesting av PEP - PDP kommunikasjonen med mange brukere vil være et naturlig steg videre.

For å teste ytelsen til PDP og avgjøre hvor i arkitekturen den skal plasseres må det lages

ulike implementasjonsscenarioer:

- En implementasjon der PDP er integrert i applikasjonen. Det vil si at applikasjonene ikke trenger å kommunisere over nettverket.
- En implementasjon der PDP kjører på en egen server på samme nettverk.
- En implementasjon der PDP kjører på en egen server på et annet nettverk.

Målet med å implementere PDP på denne måten er å se hvor stor forskjell plasseringen av PDP har å si for ytelsen.

OpenMRS og ODK Aggregate prosjektene trenger en måte å administrere attributter på. For at attributtene skal være uavhengige av applikasjonen, og administreringen skal være gjenbrukbar, kan en videre forskning være at attributtadministreringen lages som en tredjepartsapplikasjon som kobler seg til databasene den trenger.

ODK Aggregate prosjektet ble bare testet med Eclipse og MySQL. En naturlig videreføring er å teste med Google App Engine. Det må testes hvordan ODK Aggregate fungerer når det kjører på serveren og bruker Google Data Store i sammenheng med PIP.

Ved å legge til flere regler kan man gjøre policyene enda strengere. For eksempel kan man tenke seg at personell på et sykehus kan få tilgang til informasjon fra en avdeling bare hvis han er fysisk til stede og i tillegg har de riktige rolleattributtene. I noen tilfeller kan det være viktig at informasjonen ikke kan aksesseres fra andre fysiske steder. En videre utvikling vil da være å begrense informasjon for å forlate fysiske lokasjoner og kreve tilstedeværelse med ABAC:

- En måte dette kan gjøres på er å benytte fysisk lokasjon ved bruk av GPS. Ressurser (pasient informasjon) er da knyttet til en lokasjon i sykehuset, og når en bruker prøver å få tilgang utenfor lokasjonen vil han ikke få tilgang til informasjonen.
- En annen måte er å benytte seg av nettverket til sykehuset, det vil si at en kan bruke tilgangspunkter (Access Points) fra det trådløse nettverket. Hvis ikke brukeren er tilkoblet det rette tilgangspunktet vil han ikke få informasjonen han ønsker. Dette kan legges til som et ekstra attributt som skal valideres, og det vil da medføre en bedre utnyttelse av ABAC.

Denne type informasjonsavgrensing kan også brukes på tjenester som skal verne sensitiv

data. Det er få restriksjoner på hva som kan brukes av attributter for å begrense tilgang til informasjon.

7.3 Konklusjon

Hovedmålet med masteroppgaven var å teste gjennomførbarheten til ABAC, samt se om den gav større fleksibilitet og en mer dynamisk tilgangskontroll i eksisterende systemer. Resultatene viser at implementasjonen i OpenMRS har gjort dette. Kandidaten har funnet bevis på en mer fleksibel og dynamisk tilgangskontroll etter implementering av ABAC.

7.3.1 Hypoteser

Hypotese 1: Ved implementasjon av ABAC i OpenMRS vil en kunne skille pasienter på behandlingsavdeling.

Kandidaten lykkes i å filtrere på brukere og hvilke avdelinger de kan få tilgang til. Dette bekrefter den første hypotesen.

Hypotese 2: Ved implementasjon av ABAC i OpenMRS vil en kunne frigjøre rolleprivilegiumproblemet.

Implementasjonen av ABAC i OpenMRS klarte ikke å frigjøre seg fra eksisterende rolleprivilegiumstruktur. Resultatene viste at man ikke kunne frigjøre rolleprivilegiumproblemet uten å måtte starte på nytt med autorisasjonen. Hypotese to er dermed ikke understøttet.

Hypotese 3: Ved implementasjon av ABAC i ODK Aggregate vil en kunne få dynamisk tilgangskontroll.

Denne hypotesen er delvis bekreftet. ODK Aggregate hadde opprinnelig statiske roller men fikk dynamiske roller for URLer, men ikke for kodenivå. Resultatet viser at man ikke kunne inkludere nye biblioteker på kodenivå siden Java-koden blir kompilert til JavaScript med GWT og at asynkrone kall med GWT-RPC ikke var tilstrekkelig.

Hypotese 4: Ved implementasjon av ABAC i ODK Aggregate vil en kunne begrense tilgang til skjemaene med rolle-attributter.

Implementasjon av ABAC i ODK Aggregate bekreftet hypotese fire. Før kunne alle se skjemaene men etter implementasjonen er tilgangen begrenset med rolle-attributter.

7.3.2 Medical prosjektet

Medical prosjektet kombinerte erfaringer fra OpenMRS og ODK Aggregate. Dette gav en god innsikt i hva som måtte til for å få bedre utnyttelse av ABAC. Prosjektet beviser hvor dynamisk ABAC kan være sammen med Spring Security rammeverket, annotasjoner og aspekter.

Med Medical prosjektet fikk en bekreftet følgende hypoteser:

1. Filtrering av pasienter på behandlingsavdeling.
2. Rolleprivilegiumproblemet til OpenMRS ble unngått med å bruke *CRUD* handlinger på metodene.
3. Dynamisk tilgangskontroll ble løst med å bruke Spring Security på URL-nivå og aspekter med annotasjon på metodenivå.

Den fjerde og siste hypotesen om tilgang til skjemaene med rolleattributter er ikke gyldig for Medical applikasjonen.

Implementasjonene avdekket hovedsakelig at ABAC i eksisterende systemer ikke fungerer optimalt. Dette skyldes at prosjektene er utviklet for en bestemt autorisasjonsmetode. For å få bedre utnyttelse av ABAC kreves det at autorisasjonen kommer med fra starten av planleggingsfasen.

ABAC har bevist at den kan gi fleksibel og dynamisk tilgangskontroll ved å dra nytte av attributter fra brukerne, ressursene, miljøet og handlinger. Muligheten til å endre autorisasjonsregler under kjøring er en stor forbedring i alle prosjektene hvor ABAC ble implementert.

Referanser

[1] Axiomatics. Xacml for developers - updates, new tools, & patterns for the eager. http://www.slideshare.net/DavidBrossard/xacml-for-the-developers?next_slideshow=10. Sist besøkt 28.05.2015.

[2] Axiomatic. Getting started with atribute based access control. Technical report, Axiomatic, 2012.

[3] OASIS. extensible access control markup language (xacml) version 3.0, 2014. Sist besøkt 29.01.2015.

[4] Burke Mamlin. Technical overview. https://wiki.openmrs.org/display/docs/Technical+Overview, 2014. Sist besøkt 25.02.2015.

[5] Google. Google earth outreach. http://www.google.com/earth/outreach/tutorials/odk_visualize.html. Sist besøkt 17.03.2015.

[6] August Detlefsen Jim Manico. *Iron-Clad Java Building Secure Web Applications*. McGraw-Hill Education, Columbus, OH 43218, 2014.

[7] WSO2 IS. Use xacml advice elements to generate detail decisions. | soa security. http://xacmlinfo.org/2015/01/14/use-xacml-advice-elements-to-generate-detail-decisions/. Sist besøkt 16.05.2015.

[8] Virigina tech. Authorization xacml. http://courses.cs.vt.edu/cs5204/fall08-kafura/Presentations/Authorization-XACML.pdf. Sist besøkt 16.05.2015.

[9] OpenMRS. Access control in openmrs. https://wiki.openmrs.org/display/docs/Access+Control+in+OpenMRS, 2014. Sist besøkt 29.01.2015.

[10] Dafydd Stuttard and Marcus Pinto. *The web application hacker's handbook: discovering and exploiting security flaws*. John Wiley & Sons, 2007.

[11] Cory Janssen. What is access control? - definition from techopedia. http://www.techopedia.com/definition/5831/access-control. Sist besøkt 15.04.2015.

[12] Antonio J Jara, Miguel A Zamora, and Antonio FG Skarmeta. An architecture based on internet of things to support mobility and security in medical environments. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1–5. IEEE, 2010.

[13] Rolf H Weber. Internet of things–new security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30, 2010.

[14] OWASP. Top 10 application security flaws (2013). https://www.owasp.org/index.php/Top_10_2013-Top_10, 2013. Sist besøkt 19.05.2015.

[15] Open Source Initiative. The open source definition (annotated) | open source initiative. http://opensource.org/osd-annotated. Sist besøkt 07.04.2015.

[16] ODK. About. http://opendatakit.org/about/, 2014. Sist besøkt 29.01.2015.

[17] muZima. muzima documentation. https://github.com/muzima/documentation. Sist besøkt 26.03.2015.

[18] DHIS. Dhis 2 overview | dhis 2. https://www.dhis2.org/overview. Sist besøkt 26.03.2015.

[19] OpenClinica. Clinical trial software product editions | openclinica. https://www.openclinica.com/product-editions-overview. Sist besøkt 26.03.2015.

[20] OpenMRS. About openmrs | openmrs. http://openmrs.org/about/. Sist besøkt 26.03.2015.

[21] Vito M Logrillo. Health data issues for primary health care delivery systems in developing countries. 1989.

[22] Mrinmoy Barua, Xiaohui Liang, Rongxing Lu, and Xuemin Shen. Espac: Enabling security and patient-centric access control for ehealth in cloud computing. *International Journal of Security and Networks*, 6(2):67–76, 2011.

[23] David Scott and Richard Sharp. Abstracting application-level web security. In *Proceedings of the 11th international conference on World Wide Web*, pages 396–407. ACM, 2002.

[24] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[25] Alessandro Armando, Roberto Carbone, Eyasu Getahun Chekole, and Silvio Ranise. Attribute based access control for apis in spring security. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 85–88. ACM, 2014.

[26] Spring. Expression-based access control. `http://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html`. Sist besøkt 15.04.2015.

[27] Sonu Verma, Manjeet Singh, and Suresh Kumar. Comparative analysis of role base and attribute base access control model in semantic web. *International Journal of Computer Applications*, 46, 2012.

[28] Ed Coyne and Timothy R Weil. Abac and rbac: Scalable, flexible, and auditable access management. *IT Professional*, 15(3):0014–16, 2013.

[29] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Data and applications security and privacy XXVI*, pages 41–55. Springer, 2012.

[30] D P L K Ranaweera. Policy based electronic medical record system.

[31] NIST. A survey of access control models. `http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf`, 08 2009. Sist besøkt 29.01.2015.

[32] Sylvia Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the second ACM workshop on Role-based access control*, pages 31–40. ACM, 1997.

[33] Ninghui Li. Discretionary access control. *Encyclopedia of Cryptography and Security*, pages 353–356, 2011.

[34] Techterms. Trojan horse definition. `http://techterms.com/definition/trojanhorse`. Sist besøkt 25.03.2015.

[35] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[36] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[37] Reinhardt A. Botha and Jan H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[38] Liang Chen and Jason Crampton. Inter-domain role mapping and least privilege. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 157–162. ACM, 2007.

[39] Hazen A. Weber. Role-Based Access Control: The NIST Solution. Technical report, SANS InstituteInfoSec Reading Room, 08 2003.

[40] Robert W. McGraw. Risk-Adaptable Access Control (RAdAC). Technical report, NIST, 07 2009.

[41] The Dot Net Factory. Best Practices in Enterprise Authorization:The RBAC/ABAC Hybrid Approach. Technical report, The Dot Net Factory, 09 2013.

[42] Philip WL Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 191–202. ACM, 2011.

[43] OASIS. extensible access control markup language (xacml) version 3.0. Technical report, OASIS, 01 2013.

[44] Microsoft. Regular expression language - quick reference. `https://msdn.microsoft.com/en-us/library/az24scfc%28v=vs.110%29.aspx`. Sist besøkt 07.04.2015.

[45] HERAS-AF. Heras-af homepage - xacml. `http://www.herasaf.org/heras-af-xacml.html`. Sist besøkt 31.03.2015.

[46] Sun. Sun's xacml implementation. `http://sunxacml.sourceforge.net/`. Sist besøkt 31.03.2015.

[47] enterprise-java-xacml - enterprise java xacml implementation - google project hosting. `https://code.google.com/p/enterprise-java-xacml/`. Sist besøkt 31.03.2015.

[48] Apache. About. `http://thrift.apache.org/about`, 2014. Sist besøkt 16.02.2015.

[49] OpenMRS. Openmrs atlas. `https://atlas.openmrs.org/`, 2014. Sist besøkt 29.01.2015.

[50] Roger Friedman. Security and access control. https://wiki.openmrs.org/display/docs/Security+and+Access+Control, 2013. Sist besøkt 18.02.2015.

[51] XACMLinfo. Getting start with balana | soa security. http://xacmlinfo.org/2012/12/18/getting-start-with-balana/. Sist besøkt 08.05.2015.

[52] AMPATH-Kenya. Ampath-kenya home |ampath-kenya. http://www.ampathkenya.org/. Sist besøkt 30.03.2015.

[53] OpenID. Get an openid. http://openid.net/get-an-openid/. Sist besøkt 17.03.2015.

[54] HttpWatch. 6 things you should know about fragment urls. http://blog.httpwatch.com/2011/03/01/6-things-you-should-know-about-fragment-urls/, 2011. Sist besøkt 29.01.2015.

[55] Google. Gwt overview. http://www.gwtproject.org/overview.html. Sist besøkt 14.04.2015.

[56] Mozilla. About javascript - javascript | mdn. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. Sist besøkt 14.04.2015.

[57] Google. Gwt ref jre emulation. http://www.gwtproject.org/doc/latest/RefJreEmulation.html. Sist besøkt 16.04.2015.

[58] Morten Holst og André Kristensen. Analyse av open data kit aggregate.

[59] W3schools. Ajax introduction. http://www.w3schools.com/ajax/ajax_intro.asp. Sist besøkt 16.04.2015.

[60] Google. Gwt faq - server. http://www.gwtproject.org/doc/latest/FAQ_Server.html. Sist besøkt 17.04.2015.

[61] Google. Gwt dev guide coding basics compatibility. http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsCompatibility.html. Sist besøkt 14.04.2015.

[62] Abel Avram. *Domain-driven design Quickly*. Lulu. com, 2007.

[63] James O. Coplien Trygve Reenskaug. The dci architecture: A new vision of object-oriented programming. http://www.artima.com/articles/dci_vision.html, 03 2009. Sist besøkt 27.05.2015.

[64] Spring. Aspect oriented programming with spring. http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html. Sist besøkt 23.04.2015.

[65] Objectmentor. The single responsibility principle. http://www.objectmentor.com/resources/articles/srp.pdf. Sist besøkt 27.05.2015.

[66] Martin Fowler. Codesmell. http://martinfowler.com/bliki/CodeSmell.html, 02 2006. Sist besøkt 24.04.2015.

[67] Girish Suryanarayana, Tushar Sharma, and Ganesh Samarthyam. *Refactoring for software design smells*. Elsevier Science, 2014.

[68] Techopedia. What is keep it simple stupid principle (kiss principle)? - definition from techopedia. http://www.techopedia.com/definition/20262/keep-it-simple-stupid-principle-kiss-principle. Sist besøkt 27.05.2015.

[69] Martin Micah Martin C. Robert. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, 2006.

[70] Jesper M. Johansson and Roger Grimes. Is security by obscurity a valid approach? https://technet.microsoft.com/en-us/magazine/2008.06.obscurity.aspx, 06 2008. Sist besøkt 28.05.2015.

[71] Axiomatics. Axiomatics language for authorization (alfa). http://axiomatics.com/solutions/products/authorization-for-applications/developer-tools-and-apis/192-axiomatics-language-for-authorization-alfa.html. Sist besøkt 30.03.2015.