# Utvidelse og formell sikkerhetsanalyse av Dynamic Presentation Generator

Aleksander Vines

Institutt for informatikk
Universitetet i Bergen
Norge


Lang Masteroppgave
Mai 2013

# Forord

Denne masteroppgaven er resultatet av kandidatens mastergrad i informatikk ved Universitet i Bergen.

Oppgaven omhandler sikkerhetsproblematikk i Dynamic Presentation Generator og undersøker muligheten for å bruke en single-sign-on via Mi side.

Kandidaten vil gjerne takke sin veileder Khalid Azim Mughal. Han har kommet med gode innspill og gjort det mulig å gjennomføre denne oppgaven. Kandidaten ønsker også å takke Ana G. Pino, Kelly Alexander Whiteley, Aleksander Waage, Øystein Lund Rolland og Morten Høiland, som alle var medstudenter på JAFU-prosjektet.

Til slutt vil kandidaten takke familie og venner for støtten gjennom hele studietiden, og spesielt Johannes Solbraa Bay og Katarzyna Lysko som har hjulpet til med å korrekturlese oppgaven.

*Aleksander Vines*
*Bergen, 1. mai 2013*

# Innhold

# Figurer

# Tabeller

# Eksempler

# Forkortelser

| | |
|---|---|
| **API:** | Application Programming Interface |
| **ASVS:** | Application Security Verification Standard |
| **CMS:** | Innholdshåndteringssystem(eng. *Content Management System*) |
| **CSRF:** | Cross Site Request Forgery |
| **DPG:** | Dynamic Presentation Generator |
| **ESAPI:** | Enterprise Security API |
| **FS:** | Felles Studentsystem |
| **JAFU:** | JAva for FjernUndervisning |
| **LMS:** | Learning Management System |
| **MIM:** | Mann-i-midten |
| **OWASP:** | The Open Web Application Security Project |
| **TLS:** | Transportlagsikkerhet (eng. *Transport Layer Security*) |
| **UiB:** | Universitetet i Bergen |
| **XML:** | Extensible Markup Language |
| **XSS:** | Cross Site Scripting |
| **ZAP:** | Zed Attack Proxy |

# 1 Introduksjon

## 1.1 Sikkerhet i perspektiv

I begynnelsen av systemutviklingens historie ble ikke sikkerheten tatt hensyn til under utviklingsprosessen i noe nevneverdig grad. Etter hvert som nettverk ble større, og kriminaliteten på nettet vokste, ble flere og flere oppmerksomme på at dette var et stort problem. Internett har gjort hele verden til et stort nettverk, og det eksisterer enormt med åpninger for angrep. Mye har blitt gjort for å sikre seg mot angrep, men det viser seg å være en meget vanskelig oppgave å lage sikre og pålitelige systemer.

Et eksempel kan være fra nettbankens verden [13]: Der innførte man opprinnelig en løsning som hadde fungert bra i en annen setting, uten å grundig vurdere hvor sikker den ville bli i den nye settingen. Ideen var å benytte et system kundene allerede var kjent med fra bankverdenen; Bankkort har en personlig PIN, og skriver man inn feil PIN tre ganger, blir kortet sperret. I 2002-2003 brukte Skandiabanken, som er en ren nettbank, en slik løsning på sine hjemmesider. Man logget inn med personnummer (som ifølge Forvaltningsloven §13 ikke er en hemmelighet) og en personlig PIN på 4 siffer. Dessverre betyr ikke det at en metode fungerer i et system at den også vil fungerer i et nytt systemet. Dette ble tydelig i Skandiabankens tilfelle, hvor de som ville prøve å angripe banken kunne gjøre det ganske lett.

Et systems design bestemmer hvordan hele systemet fungerer, og vil derfor påvirke sikkerheten i stor grad. Videre kan en endring i designet føre til store endringer i hele systemet, og det blir gjerne dyrere å endre jo senere man er i en utviklingsprosess. Derfor er det svært viktig å tenke på sikkerhet allerede i første fase av en utviklingsprosess [6]. Det å tilpasse designet best mulig til endringer ved hjelp av smidige teknikker vil også hjelpe til med å holde kostnadene for endringer nede [1].

## 1.2 Innholdshåndteringssystem

Et innholdshåndteringssystem (CMS (eng. *Content Management System*)) er et program som gjør det mulig å publisere og redigere forskjellige typer *innhold*. Hva er så *innhold*? Det kan være en artikkel, et bilde, en video, en musikkfil, et kart, eller annen data som kan presenteres til en bruker. CMS-er skal gjøre det lett å presentere innhold også for brukere som ikke kan bruke HTML eller CSS. Systemet bør tilby et brukergrensesnitt som er enkelt å bruke slik at alle kan organisere innholdet.

To vanlige CMS-er som blir brukt i dag er WordPress [53] og Jomla! [19].

## 1.3 Fjernundervisning

Bergen Webucator prosjektet ble startet i 1998 som et samarbeid mellom Institutt for Informatikk og Senter for etter og videreutdanning, ved Universistet i Bergen (UiB). Det hadde som formål å utvikle infrastruktur for å kunne tilby nettbaserte fjernundervisningskurs. Kurset I110 var det første kurset som ble tilbydt som fjernundervisningskurs. I 2000 ble navnet på prosjektet byttet til JAva FjernUndervisning(JAFU). I dag er det tre kurs som benytter systemene til JAFU: INF100-F/*Grunnkurs i programmering*, *INF101-F/Videregåande programmering* og *MAR252/Praksisperiode, lovverk og forvaltning i akvakultur*.

## 1.4 Webucator

Det var få verktøy og teknologier som eksisterte for å kjøre og administrere fjernundervisning da Bergen Webucator prosjektet startet opp. Webucator systemet ble laget som et læringsplattform (LMS). Webucator inkluderte et system for interaktive tester, et CMS og et brukerhåndteringssystem. På grunnlag av prinsippet om "separation of concerns" [14] ble Webucator delt opp. I dag eksisterer Webucator 3.0, som er et brukerhåndteringssytem, og Dynamic Presentation Generator 2.2 (DPG),

som er et CMS. SolveIt er dagens versjon av systemet for interaktive tester, men det er verken klar for bruk eller under utvikling.

DPG og Webucator 3.0 (heretter referert som Webucator), er de to systemene som er i bruk i dag. Ole Henning Vårdal har i sin masteroppgave [51] gjort en sikkerhetsanalyse av disse systemene. Flere sikkerhetshull ble oppdaget. Han foretok en evaluering av løsninger og verktøy som Enterprise Security API (ESAPI) og Spring Security. Noen forslag til forbedring av sikkerheten ble presentert, men det var ikke så mye av det som ble implementert. Høsten 2010 ble det i faget INF226/*Programvaresikkerhet*, også laget rapporter av studenter som hadde foretatt en sikkerhetsanalyse av DPG [20] [4] [2]. Her ble det funnet flere svakheter i sikkerheten til DPG.

## 1.5 The Open Web Application Security Project

The Open Web Application Security Project(OWASP) [40] er en ideell organisasjon som fokuserer på å forbedre programvaresikkerhet. De har mange prosjekter som det kan være svært nyttig å undersøke i forhold til denne oppgaven.

## 1.6 Mål for oppgaven

Sikkerhet er noe som tradisjonelt har vært tenkt på etter at en applikasjon har vært ferdig [25]. Det er også tilfelle med DPG. Derfor trengs det en grundig sikkerhetsanalyse etter en velkjent metodologi, og *Application Security Verification Standard*(ASVS) [33] er valgt til dette. Det er tidligere gjort sikkerhetsanalyser av systemene men de har ikke fulgt noen slik formell standard som ASVS tilbyr. Det er heller ikke gjort noen forbedringer i sikkerheten til DPG basert på de tidligere rapportene.

Ved Jafu-prosjektet har det også vært ønske om å koble DPG mot UiB sin brukerdatabase. Derfor er kandidaten interessert i å se på mulighetene for å integrere det med Mi side, som er UiB sin LMS løsning.

### 1.6.1 Overordnet mål

Det ene overordnede målet for denne masteroppgaven er å utrede sikkerhetsproblemene i DPG, og å evaluere en metodologi som blir benyttet til dette. Det spesifikke systemet kandidaten skal se på er DPG og metodologien som skal testes er ASVS. Sikkerheten skal også forbedres basert på de funnene som blir gjort i analysen.

Det andre overordede målet er å integrere DPG til bruk med Mi side.

### 1.6.2 Delmål

For å kunne oppnå det overordnede målet må følgende delmål være oppnådd

- Utforsk OWASP prosjekter som kan benyttes
- Utføre en systematisk kvalitetssikring/testing for sikkerhetsproblemer basert på ASVS
  - Statisk testing
  - Dynamisk testing
  - Manuell kodegjennomgang (eng. *code review*)
  - Manuell designanalyse
- Forbedre DPG arkitektur og design
  - Forbedre brukerhåndteringen
    - Analysere dagens løsning
    - Implementere en single-sign-on-løsning med Mi side
  - Forbedre sikkerhetsproblemer funnet under testingen
- Evaluere verktøy og metoder brukt

## 1.7 Tilknytting til JAFU-prosjektet

Kandidaten har valgt å skrive masteroppgave i JAFU-prosjektet etter å ha blitt tilknyttet prosjektet i diverse jobber og kurs ved Institutt for informatikk:

- Fra høsten 2008 til våren 2011 var kandidaten gruppeleder i INF100, som jobbet i tett samarbeid med JAFU.
- Høsten 2010 gjennomførte kandidaten, i forbindelse med kurset INF226 / *Programvaresikkerhet*, en statisk sikkerhetsanalyse av SolveIt, som er en av systemene til JAFU.
- Våren 2011 tok kandidaten over som driftsansvarlig ved JAFU etter Haakon Nilsen, som var ansatt ved instituttet som dataingeniør med dette som en av arbeidsoppgavene sine. Denne jobben har kandidaten siden hatt.

- Våren 2011 hadde kandidaten også et prosjekt i programmering, der testdekningen i DPG ble forbedret.
- Våren 2012 var kandidaten gruppeleder i INF100 og INF100-F, og hadde rollen å administrere fjernundervisningen ved JAFU.

## 1.8 Organisering av oppgaven

Oppgaven er delt opp i 5 kapitler, som tar for seg hvert sitt tema.

**Kapittel 2: The Open Web Application Security Project**

De forskjellige OWASP prosjektene som er blitt brukt i oppgaven er beskrevet i dette kapittelet. Det blir gitt en oversikt over hva de brukes til, hvorfor de brukes og hvor de blir brukt i oppgaven.

**Kapittel 3: Brukerhåndtering og adgangskontroll**

I dette kapittelet blir dagens brukerhåndteringsløsning presentert. Det blir presentert flere svakheter med dagens løsning, og alternative løsninger blir foreslått. En sømløs overgang til Mi side blir implementert og evaluert.

**Kapittel 4: Sikkerhetsevaluering med ASVS**

Sikkerhetsanalysen av DPG basert på ASVS blir i dette kapittelet presentert. Mange svakheter blir funnet og evaluert, og noen forbedringer blir implementert og presentert.

**Kapittel 5: Evaluering, konklusjon og videre arbeid**

Dette kapittelet tar for seg konklusjon og evaluering av målene for oppgaven. Evaluering av teknologier, metoder og hjelpemidler benyttet blir også presentert. Ideer til videre arbeid blir så foreslått.

### 1.8.1 Oversettelser av fagutrykk

Det å finne egnede begreper på norsk for engelske fagutrykk er ikke alltid like lett. Kandidaten har forsøkt å gjøre dette på best mulig måte. Men for å unngå misforståelser er det flere steder i oppgaven også angitt den engelske versjonen i parantes. I kildekoden er det benyttet engelsk til kommentering og navngivning.

# 2

# The Open Web Application Security Project

OWASP er en ideell organisasjon som fokuserer på å forbedre sikkerheten i applikasjonsprogramvare. De ønsker å synliggjøre aplikasjonssikkerhet slik at folk og organisasjoner kan ta informerte beslutninger om virkelige sikkerhetsrisikoer.

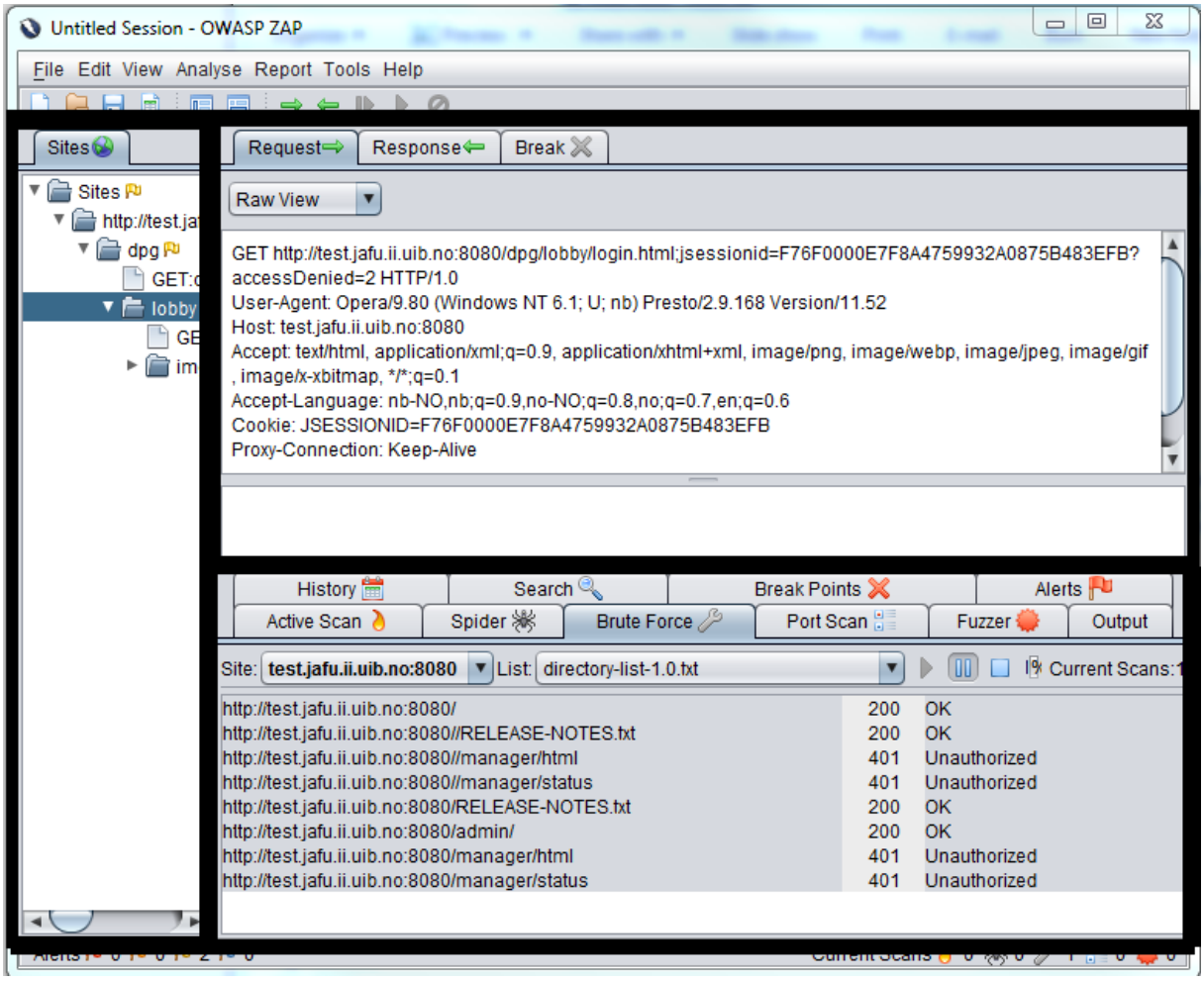
Kandidaten har utforsket forskjellige OWASP prosjekter, og noen prosjekter ble funnet som kunne være nyttige for DPG og Jafu i forhold til denne oppgaven. I dette kapittelet skal kandidaten gi en oversikt over disse prosjektene.

## 2.1 Zed Attack Proxy

Zed Attack Proxy [41] [26] (ZAP) er et verktøy for penetreringstesting [3] av webapplikasjoner. ZAP Fungerer som en avskjærende (eng. *intercepting*) proxy mellom en nettleser og webserveren. Man setter opp nettleseren til å bruke ZAP som en proxy. Da vil man kunne bruke ZAP til å lese all data som blir sent til og fra webtjeneren, og man vil kunne endre på dataene på veien. ZAP kan også sende individuelle spørringer til tjeneren på egenhånd, og mange av de automatiske funksjonene i ZAP krever at den gjør det. ZAP er laget for å være enkel å bruke; den skal kunne brukes av personer som akkurat har begynt med sikkerhetstesting. Den skal samtidig være et nyttig verktøy for erfarne sikkerhetstestere. Som alle andre OWASP prosjekter, har ZAP åpen kildekode og er gratis å bruke. ZAP baserer seg på å legge all funksjona-

litet i utvidelser (eng. *extensions*) eller tillegg (eng. *addons*), og hvem som helst kan lage det og legge til i ZAP, dermed gir det veldig gode muligheter for å skreddersy verktøyet til nøyaktig det som behøves.

Kandidaten har valgt å bruke ZAP for å assistere penetreringstestingen av DPG i kapittel 4.

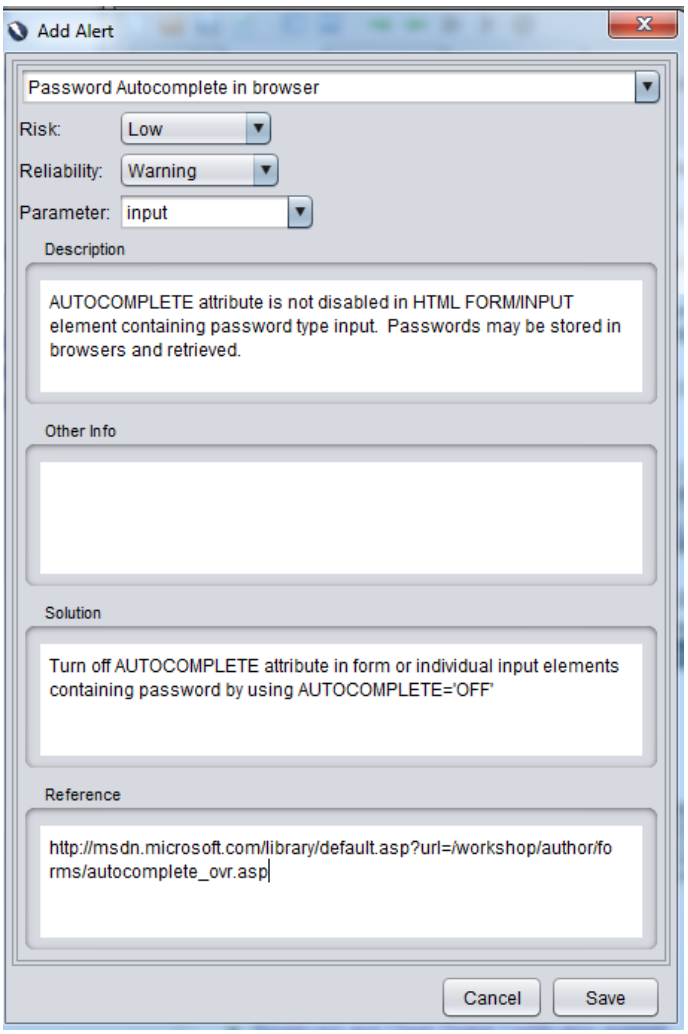



*Figur 2.1:* Skjermbilde av ZAP

I figur 2.1 vises et skjermbilde av ZAP med tre hovedrammer, som alle er markert med en sort boks. Til venstre er det en liste over nettsider og URL-er ZAP har avskjært. Øvers til høyere kan man se den spesifikke spørringen man har aktivert i den første rammen. Nederst til høyere dukker det meste av funksjonaliteten til ZAP opp. De neste avsnittene skal ta for seg hovedfunksjonaliteten i ZAP.

### 2.1.1 Alerts

Dette er en liste over potensielle sårbarheter som er oppdaget av ZAP, eller de kan være manuelt lagt til av testere. ZAP kan kategorisere sårbarheter i fem kategorier: høy, medium, lav, informative og falsk positiv. En tester kan enkelt redigere all informasjon som er lagret om en alert, se figur 2.2.




*Figur 2.2:* Redigeringsboks for en ZAP *alert*

### 2.1.2 Break Points

*Break Points* gir testeren mulighet til å stoppe spørringer eller svar før de blir videresendt av ZAP. Det er nyttig i manuell testing og det kan blant annet brukes til å teste om inndatavalidering kun skjer på klientsiden. Dette gjøres ved å redigere dataen til verdier som egentlig skal være ulovlige etter de er sent fra klienten, og dermed om serveren også validerer inndataen.

### 2.1.3 Fuzzer

Denne kan brukes til å automatisk erstatte deler av en spørring med strenger fra predefinerte lister. Et eksempel på en slik liste vises i listing 2.1 der de ti første stringene i en predefinert XSS-liste er vist.

**Listing 2.1: De ti første strengene i fuzzing-listen for å teste etter XSS**

```
1  <SCRIPT>alert('XSS');</SCRIPT>
2  '';!--"<XSS>=&{()}
3  <SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
4  <IMG SRC="javascript:alert('XSS');">
5  <IMG SRC=javascript:alert('XSS')>
6  <IMG SRC=JaVaScRiPt:alert('XSS')>
7  <IMG SRC=javascript:alert(&quot;XSS&quot;)>
8  <IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>
9  <IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
10 SRC=&#10<IMG ↩
       6;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108↩
        ;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>
```

### 2.1.4 Port Scan

Denne skanneren kan skanne en webtjener etter åpne porter. Det kan brukes for å finne mulige andre innganger inn til serveren. Det er en viktig del av testingen å analysere angrepsoverflaten til en applikasjon [31].

### 2.1.5 Brute Force

*Brute Force* er en funksjonalitet som, også her ved hjelp av predefinerte lister, kan skanne en nettside på jakt etter filer og mapper som er skjult for brukeren.

### 2.1.6 Spider

En *spider* er et verktøy som brukes for å besøke og finne alle ressursene på en nettside. Den besøker kjente URL-er på siden, og finner alle hyperlinker slik at den videre kan besøke de nye URL-ene. Dette kan være til hjelp for å utforske nettsiden raskere og å finne eventuelle skjulte linker, eller linker man har oversett.

### 2.1.7 Active Scan

Den aktive skanneren utfører en automatisk penetreringstesting av en nettside. En del kjente angrep blir forsøkt og mulige sikkerhetshull kan bli oppdaget.

## 2.2 ASVS

*Application Security Verification Standard* (ASVS) [33] er en standard for hvordan man kan måle hvor sikker en webapplikasjon er. ASVS var designet med følgende mål [33]:

- Standarden skal definere økende nivåer av verifikasjon av applikasjonssikkerhet.
- Forskjellen i dekningsgrad og rigorisme mellom nivåene bør være relativt lineær.
- Standarden skal definere funksjonelle verifikasjonskrav som følger en hvitliste (eng. *whitelist*) metode.
- Standarden skal også være uavhengig av kontrollverktøy og teknikk.

ASVS er ment å kunne bli brukt

- av applikasjonsutviklere og produkteiere som en målestokk til hvor sikker applikasjonen er.
- av sikkerhetskontrollutviklere som en veiledning til hvilken kontroller man trenger.
- i kontrakten mot kunder for å spesifisere sikkerhetskrav til applikasjonen.

I denne oppgaven har kandidaten brukt ASVS i kapittel 4. Først som en målestokk for å måle sikkerheten til DPG, og så som en slags veiledning ved utbedring av svakheter funnet under analysen.

## 2.3 Code Review og Testing Guide

Code Review Guide [37] og Testing Guide [39] er, som navnene tilsier, guider for kodegjennomgang og sikkerhetstesting.

Code Review Guide fokuserer på mekanikken ved å gjennomgå kode på jakt etter spesifikke svakheter, og gir begrenset veiledning om hvordan arbeidet bør være strukturert og utført.

Testing Guide inneholder et sett med teknikker som kan brukes for å finne forskjellige typer sikkerhetshull.

Forskjellen på de to guidene er åpenbar: Code Review Guide er laget for å teste kildekoden bak programmet, og å analysere designet basert på det. Testing Guide er laget for å utføre testing av programmet når det kjører, også kalt dynamisk testing.

Begge guidene er i denne oppgaven brukt under ASVS analysen i kapittel 4. Code Review Guide ble brukt under statisk kodeanalyse og Testing Guide under den dynamiske analysen.

## 2.4 Cheat Sheets

Cheat Sheets [35] er et prosjekt med en kolleksjon av jukseark (eng. *cheat sheets*) over mange temaer om webapplikasjonssikkerhet. Juksearkene er skrevet av sikkerhetseksperter og har utmerkede råd og informasjon av høy kvalitet, skrevet på et format som er meget enkelt å lese. Noen temaer som har sitt jukseark, som er benyttet i denne oppgaven er: *XSS Prevention Cheat Sheet*, se avsnitt 4.3.4, *Attack Surface Analysis Cheat Sheet* og *Cross-Site Request Forgery Prevention Cheat Sheet*, se avsnitt 4.3.3.

## 2.5 ESAPI

*Enterprise Security API* (ESAPI) [34] er et bibliotek som har som mål å gjøre det enklere for utviklere å produsere sikrere applikasjoner [51]. ESAPI har funksjoalitet for å tilby autentisering, tilgangskontroll, hashing, kryptering, unnslipping(eng. *escaping*), unntakshåndtering, angrepsdetektering, logging, CSRF token og inputvalidering. De tilbyr også en del HTTP verktøy og en nettapplikasjonsbrannmur (eng. *Web Application Firewall (WAF [12])*). EASPI er tilgenglig for Java EE, .NET, ASP, PHP og Python.

I denne oppgaven blir ESAPI evaluert, i avsnitt 4.4, som et alternativt sikkerhetsbibliotek til nåværende løsning.

## 2.6 CSRFGuard

CSRFGuard [38] er et bibliotek som implementerer en variant av mønsteret for synkroniseringsoken (eng. *Synchronizer Token Pattern*) [30]. Det benyttes for å redusere risikoen for *Cross Site Request Forgery* (CSRF) angrep. Det er benyttet i avsnitt 4.3.3.

## 2.7 Topp 10

I *Topp 10* prosjektet [43] forsøker OWASP å oppsummere de ti mest kritiske sikkerhetsfeil i webapplikasjoner. De kommer med en ny versjon av prosjektet hvert tredje år, og første versjon kom i 2004. Siste utgitte versjon er for 2010, men versjonen for 2013 på god vei. Målet med dette prosjektet er å tilby et dokument som bevisstgjør sikkerhet i webapplikasjoner.

Listen for 2010 ser ut som følgende:

1. Injeksjon
2. Cross-Site Scripting (XSS)
3. Ødelagt autentiserings- og sesjonsstyring
4. Usikker direkte objektreferanse
5. Cross-Site Request Forgery (CSRF)
6. Sikkerhetsfeilkonfigurasjon (eng. *Security Misconfiguration*)
7. Usikker kryptografisk lagring
8. Manglende evne til å begrense URL-tilgang
9. Utilstrekkelig transportlagsikkerhet
10. Ubekreftede viderekoblinger og forwards

Dette prosjektet er referert til i avsnitt 3.1 for å bevisstgjøre hvor viktig det er å implementere brukerhåndtering og adgangskontroll på en robust måte.

## 2.8 AntiSamy

AntiSamy [32] er et rammeverk for å validere HTML og CSS. Det er laget for at applikasjoner skal kunne fjerne ondsinnet kode fra HTML kode som er gitt av brukere. Webapplikasjoner vil gjerne tilby muligheter for brukerene til å formatere blant annet profiler, innlegg, eller som i denne oppgaven, dataen i et CMS. Inhold som er generert av brukere kan inneholde XSS angrep [42], og det er slike angrep AntiSamy kan hjelpe til med å forsvare mot.

Det er foreslått å bruke AntiSamy i avsnitt 4.3.4, for å forbedre en XSS svakhet.

# 3

# Brukerhåndtering og adgangskontroll

JAFU har administrert fjernundervisningskursene ved Institutt for informatikk ved Universitetet i Bergen. Det som har krevd mye tid er konfigurering av forum- og innleveringsystemene, samt oppretting av nye brukere. Det har derfor vært et ønske ved JAFU å forene DPG med Mi side for å smidiggjøre bruken av systemet for sluttbrukerene. Tanken er at en slik forening vil ta i bruk forum- og innleveringssystemene til Mi side og la Mi side ta seg av brukerhåndteringen.

Det kandidaten også ønsker å oppnå med dette kapittelet er å forbedre sikkerheten tilknyttet brukerhåndteringen og adgangskontrollen i DPG. For å løse oppgaven må hele designet og arkitekturen til DPG revurderes.

## 3.1 Brukerhåndtering og adgangskontroll

Med brukerhåndtering og adgangskontroll menes det her aspektene som angår hvordan brukerdata:

- opprettes
- lagres
- endres

- hentes ut
- slettes

og hvordan:

- brukere autentiserer seg mot systemet
- systemet autoriserer brukere

På OWASP sin topp 10 liste [43] er det fem av risikoene som kan knyttes til brukerhåndtering og adgangskontroll. Disse er listet opp og enkelt forklart i tabell 3.1.

*Tabell 3.1:* Risikoene i OWASP topp 10 tilknyttet brukerhåndtering og adgangskontroll

| Rangering og Type | | Beskrivelse |
|---|---|---|
| 3 | Ødelagt Autentiserings- og Sesjonsstyring | Autentiserings- og sesjons-styringen i applikasjoner er ofte ikke implementert riktig eller sikkert nok. En angriper kan finne måter å autentisere seg som en legitim bruker ved å finne smutthull i disse metodene. |
| 4 | Usikker Direkte Objekt-referanse | En angriper kan ha muligheten til å endre en direkte objektreferanse, det kan f.eks. være en fil, mappe eller en databasenøkkel. Uten tilstrekkelig tilgangskontroll i bakgrunnen vil han da kanskje kunne få tilgang til uautorisert data. |
| 6 | Sikkerhetsfeilkonfigurasjon (eng. *Security Misconfiguration*) | Miljøet og konfigurasjon rundt en applikasjon er viktig for at sikkerheten skal være ivaretatt. Det er mange som gjør feil her, og det åpner for sikkerhetshull. |
| 7 | Usikker Kryptografisk Lagring | Det finnes mye data som er sensitiv, som passord, kredittkortnummer eller personnummer. Det bør krypteres når den lagres, også sikkerhetskopier må man huske å kryptere. Angripere kan stjele eller modifisere svakt beskyttet data. |
| 8 | Manglende Evne til å Begrense URL-tilgang | Det å skjule en link for en angriper er ikke nok til å hindre noen i å komme inn på en side. Hver gang en HTTP spørring spør etter en side, må serveren foreta en tilgangskontroll. Dersom dette er glemt, har man en feil som kan føre til at angripere kan få uautorisert tilgang. |

Halvparten av topp 10 listen til OWASP kan dermed kobles til brukerhåndtering og adgangskontroll. Dette gir et sterkt inntrykk av at det å velge riktig arkitektur, og å ha en god implementasjon av den, er svært viktig for sikkerheten i en applikasjon.

## 3.2 Adgangskontroll i DPG

I DPG er det benyttet rollebasert adgangskontroll [23] [46]. Det vil si at hver bruker har en eller flere tilknyttede rolle, og hvilken rolle de har bestemmer hva de skal ha tilgang til. Det gjør det enklere å håndtere større grupper med brukere da man ikke trenger å tildele direkte tilgang enkeltvis per bruker. Det er tre forskjellige typer roller i DPG: *reader*, *publisher* og *admin*. Disse rollene er forklart i tabell 3.2.

*Tabell 3.2:* Beskrivelse av de forskjellige rollene i DPG

| Rolle | Beskrivelse | Hvem |
|---|---|---|
| *Reader* | Har tilgang til å lese gitte presentasjoner.<br>Kan laste ned ressursene til disse presentasjonene. | Studenter og andre sluttbrukere som kun skal ha tilgang til innholdet i en presentasjon. |
| *Publisher* | Kan gjøre det en *reader* kan.<br>Kan endre innholdet i gitte presentasjon, og laste opp ressurser. | Kursansvarlig og undervisningsassistenter. Alle som skal ha tilgang til å kunne endre informasjon i en presentasjon. |
| *Admin* | Kan gjøre det en *publisher* kan.<br>Kan opprette/slette presentasjoner.<br>Kan spesialisere/endre mønsteret i presentasjoner. | Drifsansvarlig. |

Rollene *reader* og *publisher* er individuelle for hver presentasjon. *Admin*-rollen er ikke knyttet til noen spesifikk presentasjon, men har tilgang til alle presentasjonene. I tillegg er det mulig å tillate anonymisert lesetilgang til en presentasjon. Det vil si at man ikke trenger å logge på for å få *reader*-rolle i denne presentasjonen.

Det kan være interessant å utvide funksjonaliteten for en *publisher* ved å gi dem tilgang til å endre kursmønsteret i en presentasjon. Dette gjør at de selv kan modifisere hvordan dataen blir presentert, og trenger dermed ikke å gå gjennom en administrator for å gjøre dette.

Slik systemet fungerer i dag krever det en del teknisk innsikt i hvordan mønsteret

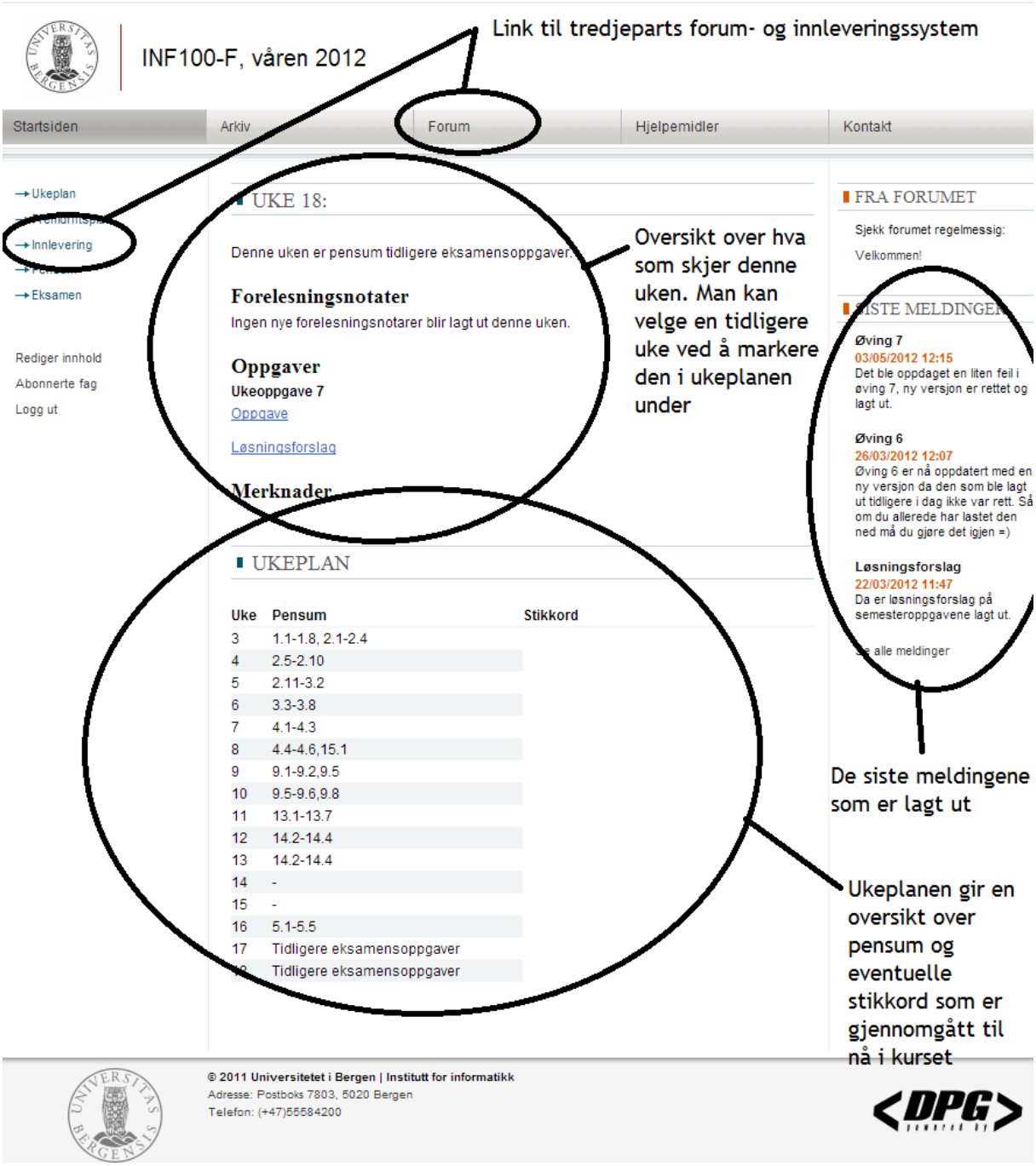
fungerer for å ikke bryte systemet når mønsteret endres. Dermed er dette en oppgave som absolutt bør kontrolleres av driftsansvarlige. Mønsteret må valideres før eventuelle endringer blir lagret, og denne valideringsfunksjonaliteten eksisterer ikke i DPG. Det vil kreve et større inngrep i systemet for å gjøre denne funksjonaliteten sikker for *publishere*. Derfor blir dette ikke gjort noe med i denne oppgaven, men det er foreslått som videre arbeid med DPG.

## 3.3 Funksjonalitet i DPG

Tabell 3.3 viser funksjonalitet for *readers* i fjernundervisningen som benytter DPG i dag. I tillegg benytter DPG to tredjepartsprogrammer, ett for innlevering av oppgaver og ett for forum. Disse er koblet sammen i en single-sign-on med DPG og linket til i presentasjonen. To skjermbilder av startsiden og arkivet til DPG, der noe funksjonalitet er kommentert og merket vises i figurene 3.1 og 3.2.

*Tabell 3.3:* Funksjonalitet for *readers* i DPG med presentasjonsmønster for fjernundervisning i JAVA

| Funksjonalitet | Beskrivelse |
|---|---|
| Meldinger | Meldinger kan bli lagt ut, og de tre siste vil vises på startsiden til en presentasjon. |
| Pensumliste | Informasjon om pensum. |
| Oversikt over hjelpemidler: litteratur, nettsteder og programvare | Informasjon om diverse hjelpemidler som kan være nyttige i kurset. |
| Fremdriftsplan | Fremdriftsplanen uke for uke, oppstilt i en tabell. Det står hva som skal leses i pensum hver uke, hvilke oppgaver man skal gjøre, og eventuelle merknader til uken. |
| Ukeplan | Viser hva som står på fremdriftsplanen for en spesifikk uke. Viser også stikkord om innholdet denne uken. |
| Arkiv for forelesningsnotater og oppgaver | Alle oppgaver og forelesningsnotater som er lagt ut så langt i kurset blir tilgjengelige i arkivet. |
| Arkiv for meldinger | Alle meldinger lagt ut i en presentasjon er tilgjengelige i arkivet. |
| Kontaktinformasjon | Side med kontaktinformasjon ved forskjellige henvendelser. |
| Eksamensinformasjon | Side med informasjon om hvor og når eksamen blir holdt. |
| Innlogging | En bruker kan logge inn. |
| Utlogging | En bruker kan logge ut. |




*Figur 3.1:* Skjermbilde av startsiden til DPG

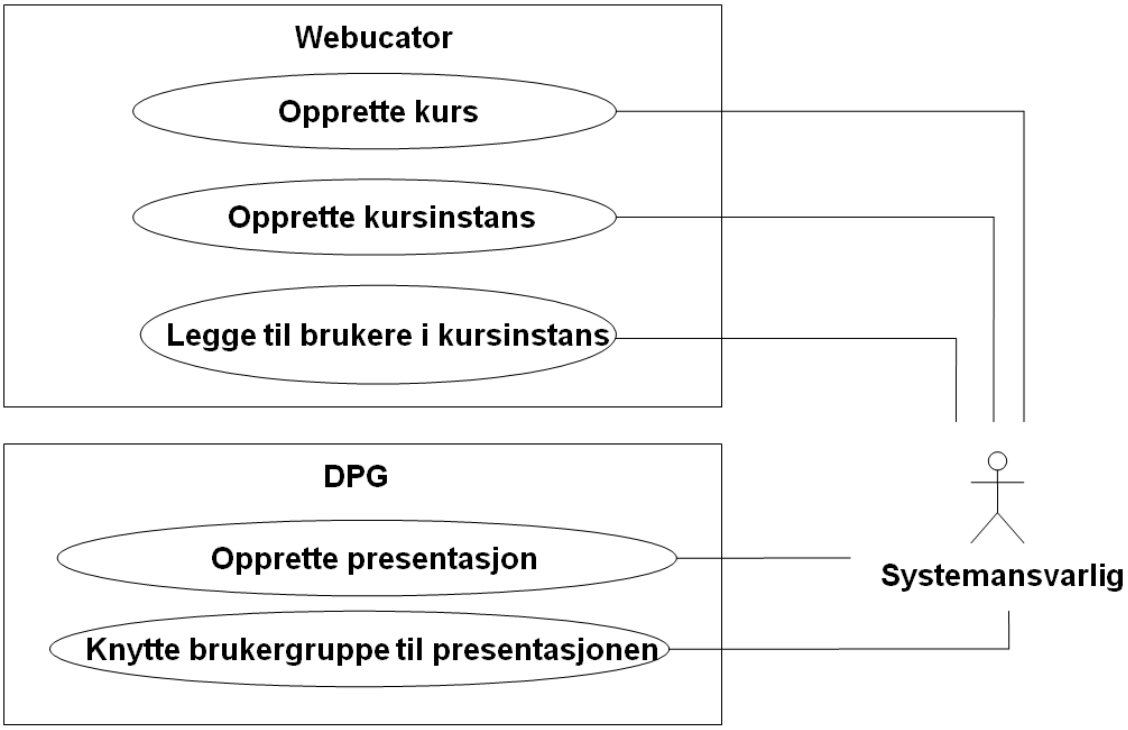

*Figur 3.2:* Utsnitt av skjermilde for arkivet til DPG

## 3.4 Dagens løsning: Webucator

I dag er det Webucator som tar hånd om all brukerhåndteringen til DPG. Alle brukere må legges inn her, enten en og en, eller flere av gangen i et Excel regneark. De forskjellige stegene som må gjennomføres er vist i figur 3.3.

Klassen `FilterSecurityInterceptor` er Spring Security [28] [47] sitt standardfilter for å foreta tilgangskontroll på en ressurs. Når dette filteret blir kalt fra filterkjeden vil det først passe på at brukeren er autentisert. Dette gjør den ved å kalle et `AuthenticationManager`-objekt og sender inn `Authentication`-objekt som tidligere i filterkjeden ble lagret i objektet `SecurityContextHolder`. Deretter vil objektet `FilterSecurityInterceptor` kontrollere om brukeren har tilgang til gitt ressurs. Dersom brukeren ikke får tilgang, vil et unntak bli kastet, og filterkjeden stoppes.

Dersom man ønsker å bruke flere forskjellige autentiseringsmekanismer, er det i Spring Security lagt opp til at man skal bruke en `ProviderManager` [28]. Figur 3.4 viser et flytdiagram for dette. En `ProviderManager` implementerer samme kontrakt som en `AuthenticationManager`. Forskjellen er at

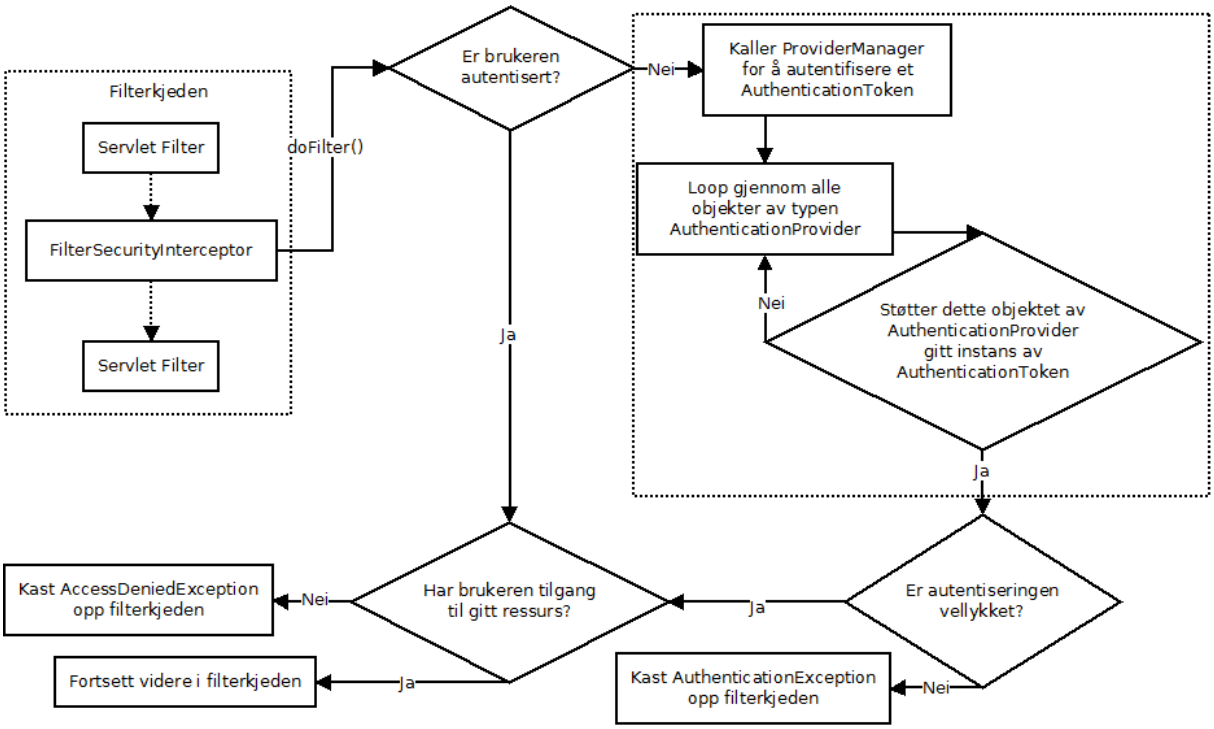



*Figur 3.3:* Koblingen mellom Webucator og DPG

`ProviderManager` er laget for å ha forskjellige `AuthenticationProvider`-objekter som hver spesifiserer en implementasjon av kontrakten `Authentication` som `AuthenticationProvider`-objektet kan autentisere. Om man vil legge til flere autentiseringsmekanismer, så vil det foregå ved å legge til flere objekter av klassen `AuthenticationProvider` i instansen til `ProviderManager`. Denne atferden til Providermanager er vist i den stiplede boksen til høyere i figur 3.4.

Webucator tilbyr en tjeneste til DPG som implementerer kontrakten `AuthenticationManager`. Denne sekvensen er vist i figur 3.5 Dette `AuthenticationManager` objektet må omskrives helt for å tilby flere autentiseringsmekanismer. Det er ikke en veldig fleksibel løsning da det strider mot åpen-lukket prinsippet (eng. *Open-Closed Principle*) [24]. Åpen-lukket prinsipper sier at en entitet (klasser, moduler, funksjoner, etc.) skal være åpen for utvidelse, men lukket for modifikasjoner. Dette prinsippet blir beskrevet av Martin [24] for å være i hjertet av objekt-orientert design og veldig viktig for en smidig (eng. *agile*) løsning.

Det er kun mulig å deklarere en global `AuthenticationManager` i Spring Security. Det virker heller ikke som om det er meningen å omgå dette ved å lage en egen implementasjon av `FilterSecurityInterceptor` som kan bruke flere forskjellige




*Figur 3.4:* Diagram for `FilterSecurityInterceptor` med `ProviderManager`

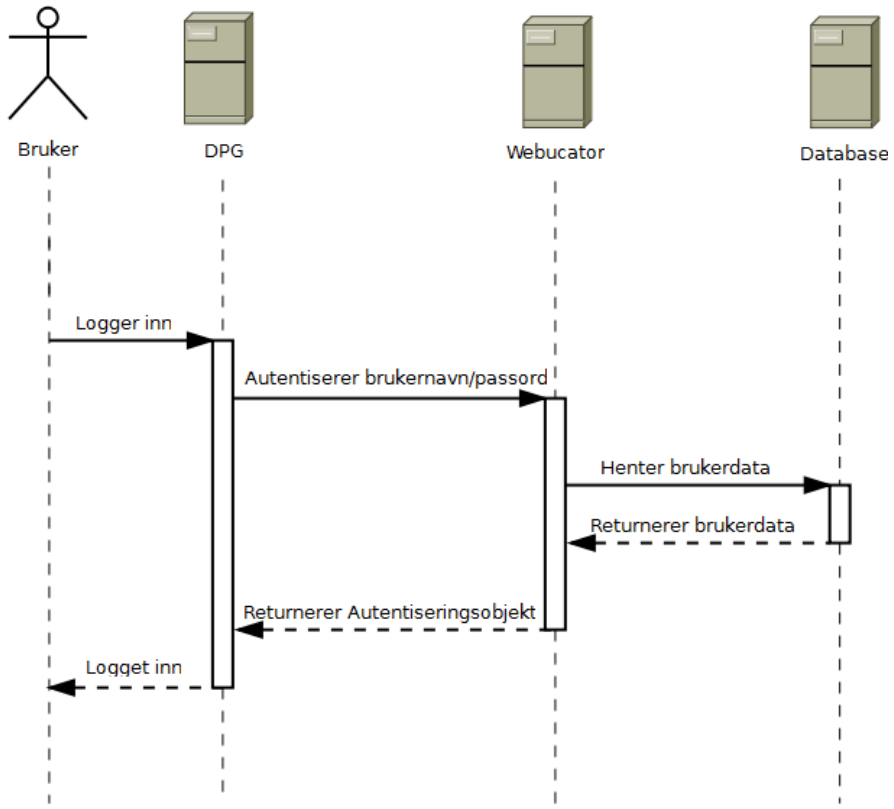
instanser av `AuthenticationManager`.

DPG er låst til denne løsningen og vil i dag ikke fungere med flere autentiseringsmekanismer ved siden av Webucator. Det ønskes en mer dynamisk løsning enn dette.

Denne løsningen krever at det lagres personopplysninger som navn, epostadresser og passord i Webucator, og slik data må beskyttes godt. Det må også være mulig å slette disse opplysningene om ønskelig [29].

Terminologien er ikke konsekvent mellom Webucator og DPG. I Webucator oppretter man kursinstanser og legger til brukere i disse. Kursinstansene har navn basert på kurskode, tittel, semester og år. I DPG referer man til dette som brukergrupper. Dette er gjort fordi Webucator kun er tenkt på som et verktøy som skal brukes med DPG og presentasjon av kursdata.

Det er en mulighet for inkonsistens av data ved at noe informasjon om presentasjoner og kurs må lagres både i DPG og i Webucator. En bedre løsning kunne her vært at det bare var nødvendig å gjøre i Webucator, og at DPG hentet denne informasjonen via en tjeneste.




*Figur 3.5:* Sekvensdiagram for innlogging via Webucator

Det går enkelt an å koble en brukergruppe fra Webucator til flere forskjellige presentasjoner i DPG, men det er ikke mulig å koble flere brukergrupper til en presentasjon. Webucator fungerer også slik at en bruker kun kan ha en rolle i systemet: enten *reader* eller *publisher* tilknyttet en spesifikk brukergruppe, eller de kan være *admin*.

I Webucator har man også en funksjonalitet for å deaktivere en kursinstans. Brukere som har status som *reader* i en deaktivert kursinstans vil ikke ha tilgang til denne. Dette er en praktisk funksjonalitet dersom man for eksempel ønsker å opprette en presentasjon for å klargjøre innholdet, men ikke enda vil at *readers* skal ha tilgang.

Webucator tilbyr autentisering som en remote service som implementerer kontrakten `AuthenticationManager` i pakken `security.authentication` fra Spring Security.

Sikkerhetshull i denne løsningen [51] som vil påvirke valget av ny løsning:

- Passordene er lagret i klartekst i databasen.

- Manglende bruk av transportlagsikkerhet (TLS(eng. *Transport Layer Security*))
- "Svak" autentisering:
  - Ingen krav til passordstyrke hos administratorer
  - Ingen begrensninger på antall innloggingsforsøk

### 3.4.1 Lagring av passord

Webucator lagrer alle passordene i klartekst. Dersom en angriper da får tilgang til denne dataen vil han få passord, samt navn og epostadresse, til alle brukerene av DPG. Om en angriper får tak i alle passord og brukernavn har han fri tilgang til å for eksempel logge inn som en administrator og slette en presentasjon. Det er dessuten mange som benytter seg av samme passord mange steder, og det åpner også da opp for at angriperen kan få tilgang i helt andre systemer. Negativ publisitet er også en stor skadefaktor som kan oppstå om personopplysninger som dette lekker ut.

### 3.4.2 Transportlagsikkerhet

Kommunikasjonen over netteverket foregår uten noen form for kryptering. Dermed kan all data som sendes leses i klartekst av noen som lytter til nettverket. Det å lytte til ett nettverk er heller ikke spesielt vanskelig i dag da det finnes mange forskjellige verktøy for dette. En angriper kan da få tak i for eksempel sesjons-ID, brukernavn og passord. Dette åpner også opp for mann-i-midten (MIM) angrep. Et MIM angrep foregår ved at en angriper fungerer som et mellomledd mellom en sluttbruker og DPG. For både DPG og sluttbrukeren virker det tilsynelatende da som om de snakker med den andre parten. En ting en angriper da kan gjøre er å endre på informasjonen som er sendt, slik som innholdet i en presentasjon.

### 3.4.3 Autentisering

DPG har ingen krav til passordstyrke hos administratorer. Kombinert med det at det ikke er noen begrensninger på innloggingsforsøk, gjør dette at det kan være veldig lett for en angriper å gjette seg frem til riktig passord. Vårdal [51] fant, på tyve minutter, fire administratorpassord ved å bruke en enkel metode der man forsøker å logge inn med en liste av potensielle passord.

Både brukernavn og brukerpassordene som automatisk genereres av Webucator er bygget opp etter en meget enkel struktur. Brukernavnet består av første bokstav i fornavet, deretter de to første bokstavene i etternavnet etterfult av et tresiffret

tall som starter på 001 og inkrementeres dersom en bruker allerede eksisterer med dette brukernavnet. For eksempel kan brukernavnet for Ola Normann være `ono001`. Passordene er også på en enkel form [51]. Det er $10^3 + 6^3 + 14^2 = 42336000$ mulige kombinasjoner av mulige genererte passord. Dette er ikke et stort tall om man tar i betraktning regnekraften i dagens datamaskiner. Det tar ikke lang tid å finne riktig kode om man har et ubegrenset antall innloggingsforsøk.

### 3.4.4 Single-sign-on for forum og innlevering

I klassen `CustomAuthenticationProcesingFilter` var det lagt inn en liten hack som lagrer sesjons-ID-ene i systemegenskaper (eng. *system properties*). Denne vises i listing 3.1. Den ble lagt til for å ha en single-sign-on fra DPG til de to tredjepartsprogrammene for innlevering av oppgaver og forum. Det ble også laget en hack i hver av de to for å tilpasse seg denne løsningen. Det ble også passet på at når en bruker logger ut, eller en sesjon på annen måte invalideres, så blir denne sesjons-ID-en slettet fra systemegenskapene.

**Listing 3.1: Hack for single-sign-on mot forum og innleveringssystem**

```
1  StringBuilder buf = new StringBuilder();
2  //legger brukernavnet i bufferen
3  buf.append(principal.toString());
4  //itererer over alle presentasjonene brukeren har tilgang til
5  //legger presentasjons-ID-en til hver av dem i en buffer
6  for (String presId : auth.getPresentations().keySet()) {
7      buf.append(':').append(presId);
8  }
9  //legger sesjons-ID-en inn som en systemvariabel med verdien satt til ↩
       bufferen
10 System.setProperty("dpg2.session."+request.getSession().getId(),buf.↩
       toString());
```

Om denne løsningen lager et sikkerhetshull er usikkert. Det avhenger for eksempel av de andre applikasjonene som kjører på samme *Java virtual machine*, da de alle har tilgang til de samme systemegenskapene. En bedre løsning kunne vært å benytte Spring-Security-rammeverket som tilbyr støtte for flere single-sign-on løsninger [28].

Et annet problem med denne løsningen var at avbildingen (eng. *mapping*)) fra brukergrupper i Webucator til forskjellige forum i forum-løsningen måtte legges inn manuelt i et xml-dokument, som var veldig upraktisk, og krevde tilgang til ID-en til hver gruppe og forum.

## 3.5 Alternative løsninger

For å forbedre dagens system og å gjøre det best mulig, er det vurdert to mulige løsninger. Den første er å endre hvordan Webucator blir brukt, og den andre er å lage en ny løsning med sømløs overgang til Mi side.

### 3.5.1 Endre Webucator

DPG er hovedsystemet til JAFU, og det burde ikke være avhengig av Webucator for å kjøre. Webucator kunne tilby en tjeneste som DPG har muligheten til å benytte seg av. Dersom `AuthenticationManager`'en hadde eksistere i DPG og en `AuthenticationService` [28] vært tilbudt fra Webucator ville det være en bedre løsning med tanke på fleksibilitet.

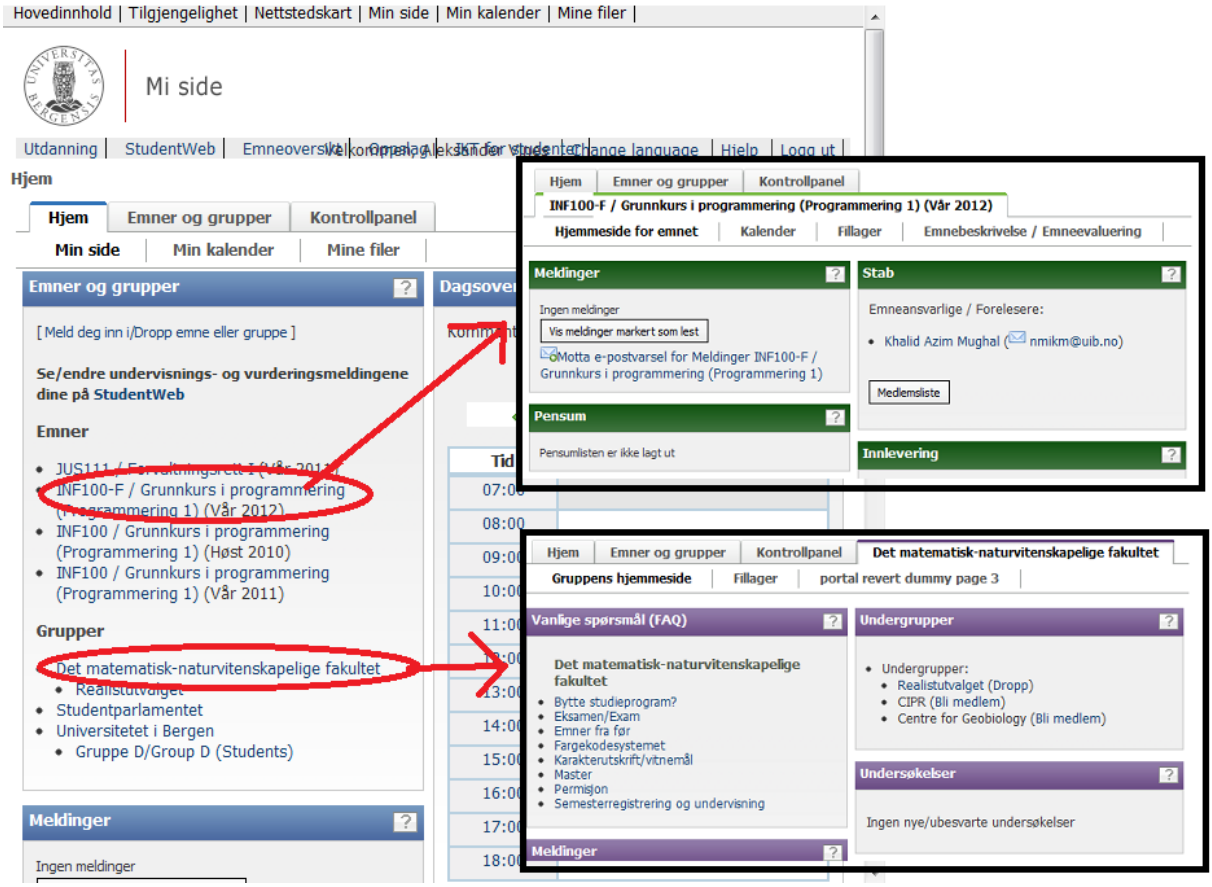
Webucator burde også bruke samme terminologi som DPG for å unngå misforståelser. Det vil si at man kunne opprette brukergrupper istedenfor kursinstanser i Webucator. Løsningen, foreslått i avsnitt 3.4, tilknyttet mulig inkonsistens av data kunne også vært gjennomført.

### 3.5.2 Mi side

Mi side [17] er et .LRN [22] / OpenACS (eng. *Open Architecture Community System*) [7] nettsted. .LRN Consortium, som står bak .LRN, er en frivillig organisasjon som fokuserer å lage og tilby støtte til nettbaserte læringssystemer. OpenACS består av et sett med verktøy og et rammeverk for å lage skalerbare, fellesskapsorienterte nettapplikasjoner. .LRN er bygget på OpenACS.

På UiB benyttes Mi side som et intranett for studenter. Mi side har tre hovednivåer, personlig side, emneside og gruppeside, som alle vises en del av i figur 3.6. Her er det den personlige siden som er i bakgrunnen. Den er unik for hver student og samler informasjon om alle emner og grupper som man er medlem av. Den øverste fremhevede boksen er dit man kommer om man går inn på et spesifikt emne, i dette tilfelle INF100-F for våren 2012. Den nederste boksen viser en spesifikk gruppe, her gruppen for det matematisk-naturvitenskapelige fakultet.

Basisfunksjonaliteten til Mi side vises i tabell 3.4.




*Figur 3.6:* De tre hovednivåene til Mi side

### Sømløs overgang

Mi side tilbyr en sømløs overgang fra Mi side til andre systemer. I dag benytter Kark [16] seg av denne sømløse overgangen. Den fungerer slik at en lenke til det eksterne systemet blir lagt ut på hjemmesiden til et kurs på Mi side. Lenken inneholder dynamiske elementer som sesjons-ID, emnekode, emnenavn, periode, undervisningsaktivitetskode(*uaktkode*), navn på undergruppe og siden(URL) hvor lenken kommer ifra. Basert på informasjonen må det eksterne systemet sende en forespørsel tilbake til Mi side for å autentisere brukeren basert på sesjons-ID-en. Dersom sesjons-iden er for en gyldig sesjon, vil et xml-dokument returneres, ellers returneres bare en tom linje.

Returdataen man får fra spørringen tilbake til Mi side avhenger av hvilke argumenter man sender inn. En oversikt over de tre forskjellige kombinasjonene av argumenter

*Tabell 3.4:* Basisfunksjonalitet i Mi side

| Funksjonalitet | Beskrivelse |
|---|---|
| Grupper | Brukes for å dele studentene inn i mindre grupper. |
| Meldingstjeneste | Brukes for å gi meldinger til studenter. |
| Fillager | Viser alle filer som er lagt ut om alle emner og grupper som man er medlem av. |
| E-post | Brukes til å sende E-post. |
| Litteraturlister | Inneholder pensumliste for et fag. |
| Kalender | Inneholder kalenderopplysninger fra alle de gruppene og emnene man tilhører. Det er også mulig å legge til personlige kalenderinnlegg. |
| Innlevering | Funksjon for å levere inn oppgaver. |
| Forum | Diskusjonsforum som kan legges til på en gruppe eller fag. Alle medlemmer av gruppen eller faget vil da ha mulighet til å delta i diskusjon. |

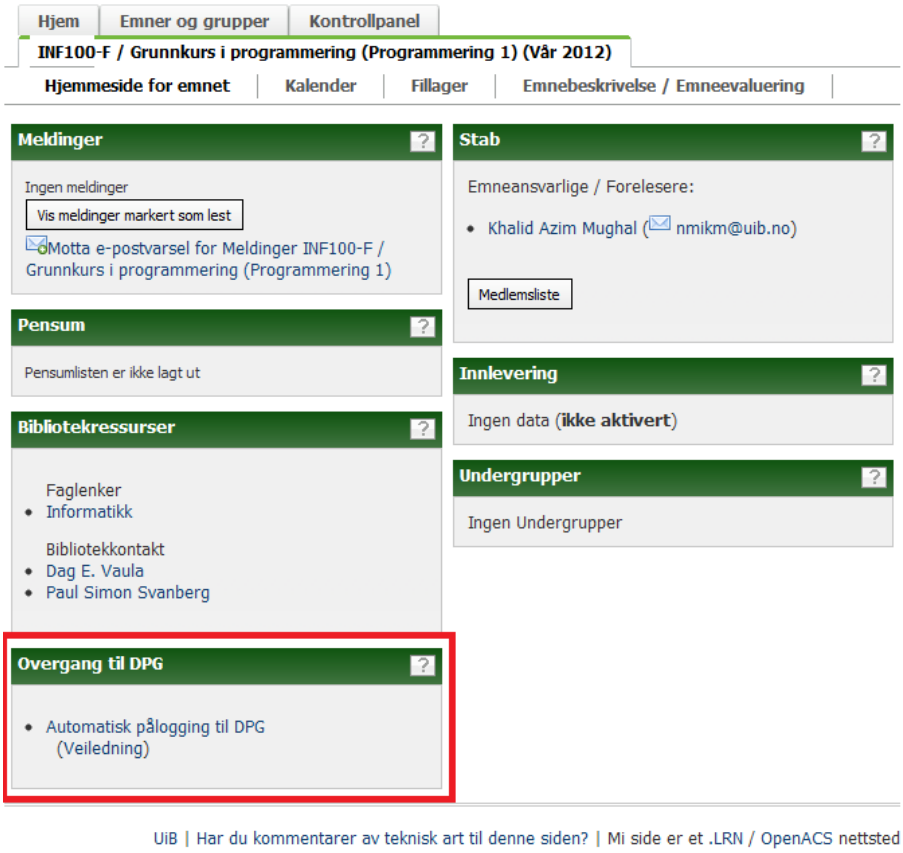
vises i tabell 3.5.

*Tabell 3.5:* Oversikt over hvilke returdata man finner i xml-dokumentet for et tilbakekall til Mi side. ID referer til sesjons-ID-en

| Returdata | Argument | | |
|---|---|---|---|
| | ID | ID & emnekode & periode | ID & emnekode & periode & uaktkode |
| brukernavn | Ja | Ja | Ja |
| brukertype(student, ansatt,ekstern) | Ja | Ja | Ja |
| navn | Ja | Ja | Ja |
| status(X eller tomt) | Ja | Ja (på emnet) | Ja (på emnet) |
| admin(0 eller 1) | Nei | Ja (på emnet) | Ja (på undergruppe) |
| role | Nei | Ja (på emnet) | Ja (på undergruppe) |
| fak_inst(4 siffer) | Nei | Ja (på emnet) | Ja (på emnet) |

Den største praktiske fordelen med en slik løsning vil kanskje være at administratorer av fjernundervisningen ikke trenger å tenke på å lage egne brukere for studentene i DPG. Studentene blir automatisk lagt inn i Felles Studentsystem(FS) og vil dermed automatisk få tilgang til DPG via Mi side. FS er UiB's studieadministrative system [50] og Mi side henter sin brukerdata derfra.

Å lage en slik løsning vil også utvide funksjonaliteten i Mi side. Det er mulig at flere kurs vil ønske å benytte seg av DPG for det er mulig å skreddersy DPG og lage




*Figur 3.7:* Mi side med overgang til DPG

funksjonalitet som trengs til de spesifikke kursene.

### Sikkerhetssvakhet i den sømløse overgangen

Den sømløse overgangen baserer seg på en sesjons-ID bestående av et heltall med 9 sifre. Det betyr at det er en milliard forskjellige mulige sesjons-ID-er. Mi side krever ingen sikkerhetsfunksjoner, av systemer som benytter den sømløse overgangen, som vil prøve å begrense antall innloggingsforsøk. Dette medfører da at det eksisterer en sikkerhetssvakhet, hvor det kan være mulig å gjenomføre et brute-force angrep som tester alle tall fra null til en milliard. Dermed skal alle sesjons-ID-er som er gyldige i det de blir forsøkt bli oppdaget. Det vil kanskje også være mulig å spre angrepene slik at 500 millioner siffer testes på KARK og 500 millioner på DPG for å spre belastningen på disse to serverene litt. Det kan videre også være mulig å distribuere angrepene slik at de kommer fra forskjellige maskiner og dermed blir vanskeligere å

oppdage.

Det er kun en del av sesjons-ID-en som blir benyttet i overgangen, så denne svakheten gjør ikke direkte at man får tilgang til en brukers sesjon på Mi side, men man vil få tilgang til alle systemer som benytter seg av den sømløse overgangen.

Studieadministrativ avdeling [48], som er systemeier for Mi side, og IT-avdelingen [15], som har ansvar for driften av Mi side, ble informert om denne svakheten. Etter kort tid ble en løsning presentert og implementert. Løsningen besto av å inkludere en enveis-hash, laget av sesjons-ID-en, en hemmelighet (salt [21]) og dagens dato, med sesjons-ID-en. Med det endres sesjons-ID-en seg for eksempel fra `528338401` til `528338401:211369CEE588A699521431FA3B0FBA0EDDB744B1`. Dersom en angriper ikke klarer å oppdage nøyaktig hvordan denne hashen blir generert, er det nå tilnærmet umulig å benytte seg av et slikt brute-force angrep.

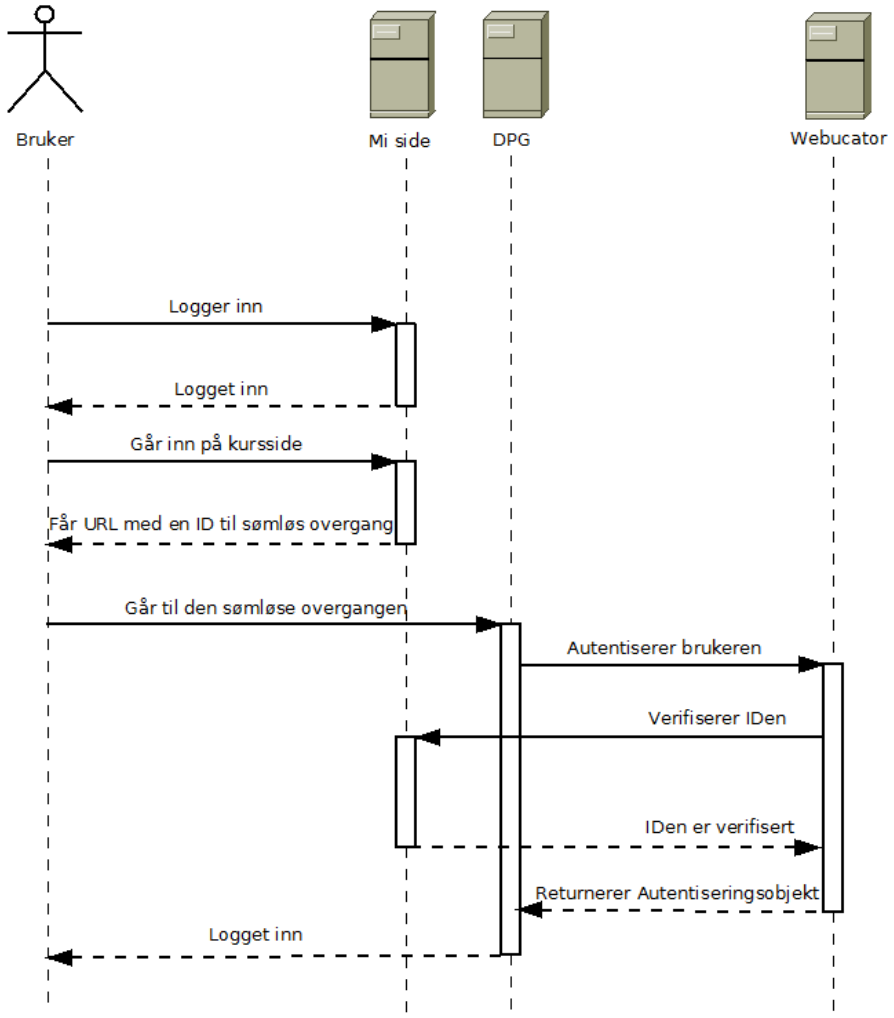
Det kan være problemer ved å benytte seg av den nye løsningen rundt midnatt. Hashen baserer seg på dagens dato, og vil endres når det er en ny dag. Dersom en bruker da prøver å logge inn rett før midnatt, kan sesjons-ID-en han prøver å logge inn med ha ugyldig hash i det tilbakekallet skjer mot Mi side fordi datoen har endret seg siden hashen ble generert.

## 3.6 Koblingen til Mi side

Det var to forskjellige løsninger for arkitektur som ble vurdert i forhold til den sømløse overgangen fra Mi side. Det første alternativet var å beholde Webucator for å foreta autentiseringen og gi den funksjonalitet for å foreta et tilbakekall til Mi side for å autentisere en bruker fra en sesjons-ID. Denne løsningen skisseres i figur 3.8. Det andre alternativet var å ikke bruke Webucator ved denne påloggingen, men å legge denne funksjonaliteten inn i DPG. Et sekvensdiagram som beskriver denne løsningen vises i figur 3.9. Det sistnevnte alternativet ble valgt for å prøve å løse opp den sterke koblingen mellom DPG og Webucator.

### 3.6.1 Data fra Webucator vs Mi side

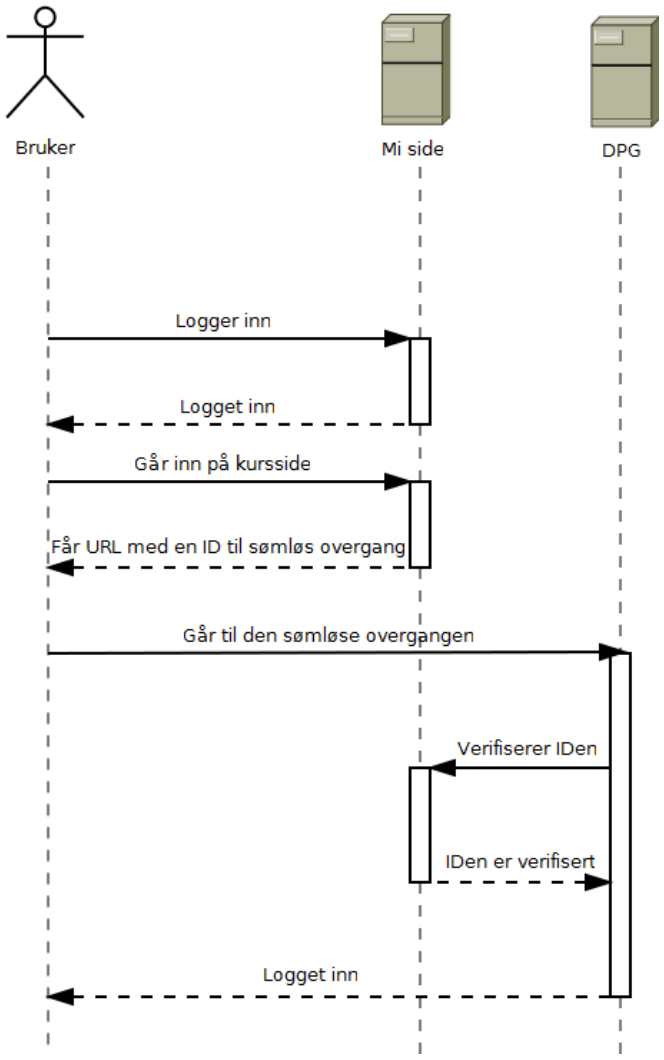
Webucator ga en brukergruppe-ID, som adgangskontrollen i DPG baserte seg på. Mi side ga kurskode og periode. Dermed måtte det endres fra å sjekke at brukergruppene stemte overens til å sjekke at presentasjons-ID-ene, som er basert på kurskode og periode, stemte. Denne endringen ble gjort i sikkerhetsfilterene i pakken `core.security.filters` som avgjorde tilgangen til hver enkelt presentasjon.




*Figur 3.8:* Sekvensdiagram for sømløs overgang via Webucator

### 3.6.2 Funksjonalitet for forum og innlevering

Den enkleste løsningen på problemene forklart i avsnitt 3.4.4 var å fjerne hacken. I og med at det nå ønskes å bruke forumet og innleveringssystemet til Mi side det ingen bruk for å ha en slik hack. Dersom det i fremtiden ønskes å implementere noe lignende igjen, så er det en god idé å benytte seg av et eksisterende rammeverk istedenfor en slik hack. All kode og konfigurasjon som hørte til koblingen mellom forum- og innleveringssystemene og DPG ble fjernet.




*Figur 3.9:* Sekvensdiagram for sømløs overgang

### 3.6.3 Standard inloggings/utloggings URL

DPG brukte standard URL for inlogging og utlogging, for innlogging er det /spring_security_login. Det anbefales [28] å bytte denne URLen for å skjule at det er Spring Security som brukes som implementasjon for diverse sikkerhetsfunksjonaliteter. Dette kan gjøre det vanskeligere for en angriper å finne sikkerhetshull dersom det skulle bli oppdaget svakheter i Spring Security rammeverket. Selv om sikkerhet ved obskuritet ikke egentlig øker sikkerheten i en applikasjon, kan det gjøre det vanskeligere for standariserte hackingverktøy å finne sikkerhetshullene.

Denne enkle standardverdien måtte endres to steder i en xml-fil og i fem forskjellige jsp-filer. Det kunne her vært en bedre løsning å lage en variabel som kunne benyttes alle disse stedene slik at man unngår mulige feil. Dette er ingen direkte sikkerhetssvakhet og prioriteres ikke å løses.

### 3.6.4 Sikkerhetskonfigurasjonen

I konfigurasjonsfilen til Spring Security, *applicationContext-Securtiy.xml*, ble det funnet beans som ikke ble brukt, beans som var helt like (med forskjellige navn), og beans som brukte foreldete (eng. *deprecated*) klasser. Det var også filtere i filterkjedene som ikke hadde noen effekt. Dette tydet på at de som har konfigurert dette kanskje ikke visste helt hva de gjorde, og det åpner for muligheter for sikkerhetshull .

`HttpSessionContextIntegrationFilter` er foreldet i Spring Security 3, så den ble byttet ut med `SecurityContextPersistenceFilter`, som er dens superklasse og i praksis fungerer helt likt for DPG. Klasser som er foreldet blir ikke nødvendigvis oppdatert lengre. Dersom det blir oppdaget sikkerhetssvakheter i dem kan systemet være utsatt for kjente sikkerhetshull selv om man tilsynelatende har nyeste versjoner av rammeverkene. Det kan også være at noe er foreldet fordi det er oppdaget en sikkerhetssvakhet med denne løsningen. Man bør derfor kun bruke klasser som ikke er foreldet.

I listing 3.2 vises filterkjeden som knyttes til URL på formen `/pce/**`. Filterene `logoutFilter` og `authenticationProcessingFilter` vil ikke ha noen effekt i denne kjeden. `logoutFilter` vil kun kjøre dersom URLen er */j_spring_security_logout* da denne standardverdien ikke blir endret i DPG. På samme måte vil `authenticationProcessingFilter` kun prosessere spørringer der URLen er *"/j_spring_security_check"*.

Disse filterene benytter seg av flere klasser som er foreldet, for eksempel klassene `HttpSessionContextIntegrationFilter` i pakken `httpSessionContextIntegrationFilter` og `AuthenticationProcessingFilterEntryPoint` i pakken `exceptionTranslationFilter`.

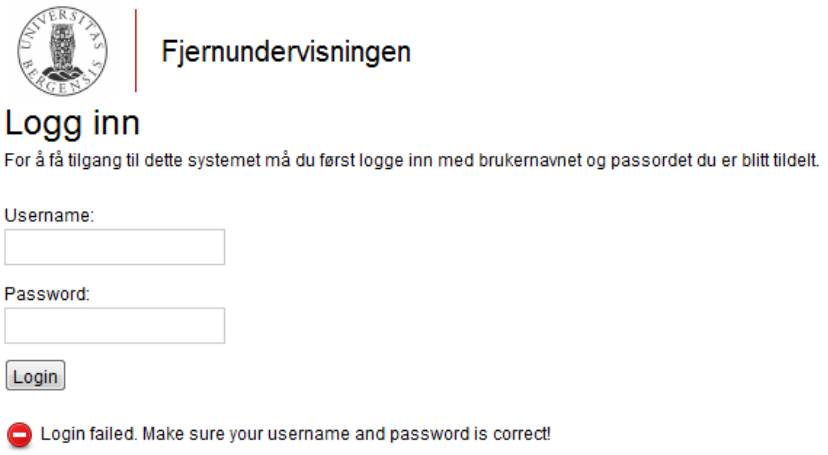
Filterene `filterSecurityInterceptor` og `filterInvocationInterceptor` er helt identiske og brukes litt om hverandre. Beanen `authenticationProcessingFilterEntryPoint` blir ikke brukt, men ser ut til å være rester fra en eldre konfigurasjon.

**Listing 3.2: Filterkjeden til pce**

```
1 <sec:filter-chain pattern="/pce/**" filters="
2 httpSessionContextIntegrationFilter, logoutFilter,
3 authenticationProcessingFilter, securityContextHolderAwareRequestFilter,
    ,
4 exceptionTranslationFilter, filterInvocationInterceptor,
    customPCEAuthFilter"/>
```

### 3.6.5 Feilmeldinger

Dersom det skjedde en feil i autentiseringen eller autoriseringen ble brukerene sendt tilbake til login-siden. En slik feil oppstår om en bruker oppgir ugyldig logininformasjon eller at han prøvde å gå til ressurser han ikke hadde tilgang til. Siden inneholdt en feilmelding og det er vist i figur 3.10.




*Figur 3.10:* Tidligere feilmelding i DPG

Da brukere nå skal bruke den sømløse overgangen mot Mi side gir det lite mening for dem å få opp en loginside om noe skulle gå galt. Dermed ønskes det å gi en mer brukervennlig feilmelding.

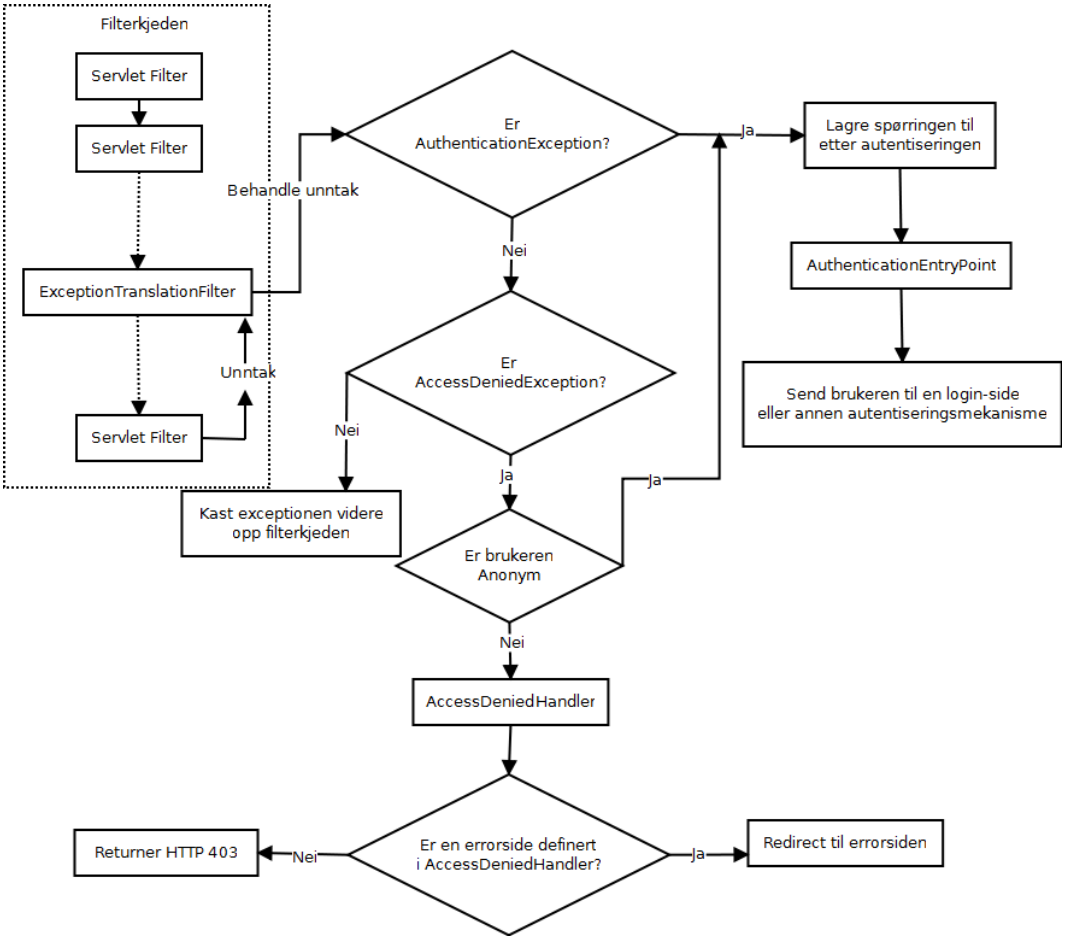
Spring Security har to unntak som kan kastes, det er `AuthenticationException` og `AccessDeniedException`.

En `AuthenticationException` kastes dersom det skjer en feil under autentiseringen, slik at en bruker ikke blir autentisert. Det mest vanlige vil her være at brukeren oppgir feil brukerdata, som passord eller brukernavn, men det kan også for eksempel være at systemet ikke kan koble til en database eller tjeneste som er

nødvendig for å autentisere brukeren.

`AccessDeniedException` kastes av `AccessDecisionManager` dersom en bruker ikke har nødvendige rettigheter for tilgang til en ressurs.

Filteret `ExceptionTranslationFilter` tar seg av disse to unntakene. Dette vises i figur 3.11. Dersom det er en `AccessDeniedException` og brukeren ikke er anonym blir han redirected til en feilside, dersom det er definert i `AccessDeniedHandler`, ellers returneres `HTTP 403 Forbudt` feil. Dersom det verken er `AuthenticationException` eller `AccessDeniedException` blir exceptionen kastet videre opp i filterkjeden. Ellers blir brukeren sendt til en autentiseringsmekanisme definert i `AuthenticationEntryPoint`.




*Figur 3.11:* Diagram for unntakshåndtering i `ExceptionTranslationFilter`

Til den sømløse overgangen vises siden tilknyttet `AuthenticationEntryPoint` i figur 3.12, og feilsiden i `AccessDeniedHandler` vises i figur 3.13.


*Figur 3.12:* Innloggingsside i `AuthenticationEntryPoint` for Mi side


*Figur 3.13:* Feilsiden i `AccessDeniedHandler`

### 3.6.6 Login for administratorer

Slik de tre rollene i DPG fungerer er det en direkte tilknyttning mellom reader/-publisher og spesifikke presentasjoner. Administratorer er ikke tilknyttet noen presentasjoner, men har tilgang til hele systemet og alle presentasjoner. Den sømløse overgangen baserer seg på at brukere skal få tilgang til spesifikke presentasjoner, basert på hvilket kurs de velger å logge inn fra på Mi side. Dermed blir det uklart for hvordan man skal håndtere administratorer med tanke på den sømløse overgangen. Fra den sømløse overgangen kan brukere ha følgende roller: *student*, *administrativ kontaktperson*, *undervisningsassistent*, *administrator*, *emneansvarlig* eller *sensor*.

Tre løsninger er vurdert på dette problemet:

- Gi alle administratorer og emneansvarlige på kursene i Mi side administratorstatus i DPG. Med denne løsningen vil man ikke trenge noe annet enn Mi side

for å kunne logge inn alle slags brukere. Dette er også en veldig enkel løsning som ikke krever mye koding. På den negative siden så blir det veldig mange som unødvendig får rollen som *admin*, men det er kun noen få som faktisk trenger det. Administratorer er også mente for å være globale i DPG, mens alle rollene i Mi side er spesifikke for hvert kurs og periode.

- Gi alle administratorer og emneansvarlige på kursene i Mi side administratorstatus i DPG og endre på DPG slik at administratorer også er knyttet til spesifikke kurs. Da kan for eksempel en ny presentasjon bli opprettet når en administrator første gang bruker koblingen fra Mi side. Her behøver man også kun koblingen mot Mi side for å kunne autentisere alle typer brukere. Imidlertid vil man trenge store endringer i DPG for at det skal fungere, og man vil endre på meningen av det å være en administrator i DPG.
- Ingen får administratorrettigheter fra Mi side. Man behøver da en alternativ autentiseringsmekanisme for å kunne logge på som administrator. I dette tilfellet er tanken å fortsatt bruke Webucator ved siden av Mi side. Man beholder da rollene i DPG slik det er i dag. Løsningen krever litt koding, og en del konfigurering av Spring Security. Dersom et sikkerhetshull skulle bli oppdaget i Mi side, vil det føre til mindre risiko for DPG.

### 3.6.7 Inkapsling av funksjonalitet for den sømløs overgangen

For å holde forskjellige komponenter mest mulig adskilt fra hverandre, og for å få en veldig fleksibel løsning er det valgt å separere autentiseringslogikken mest mulig fra DPG. Dermed er den sømløse overgangen som blir brukt enkel å bytte ut. Det man trenger da er noe som implementerer samme grensesnitt og å konfigurere dette i DPG.

## 3.7 Etter produksjonssetting

Våren 2013 ble den sømløse overgangen fra Mi side tatt i bruk. Det var fagene INF100-F og INF101-F som skulle bruke Mi side og DPG.

### 3.7.1 Innloggingsproblemer

Kort tid etter semesterstart meldte noen studenter at de ikke fikk logget inn i DPG. Når de trykket på den sømløse overgangen, fikk de bare opp feilsiden vist i figur 3.12.

Problemet måtte løses raskt og feilsøkingen ble startet ved å analysere loggfilen. Det ble funnet at klassen `MiSideCallbackImpl`, som gjennomfører tilbakekallet til Mi side og tolker XML-filen som returneres, kaster et unntak. Feilmeldingen i unntaket ble logget, og var "*Invalid byte 2 of 3-byte UTF-8 sequence*". Det ble da klart at det hadde noe med enkodingen (eng. *encoding*) å gjøre. Kandidaten merket seg også at det var de studentene som hadde de norske særtegnene `[æ, ø, å]` i navnene som hadde problemer med å logge inn.

Tilbakekallet ble så gjennomført manuelt og XML-filen som ble sendt tilbake ble analysert. Det ble klart at enkodingen her var i `ISO-8859-1` format. Det var ingen deklarasjon av dette i XML-dokumentet. Da antar `SAX`, som er rammeverket benyttet for å tolke XML, at det er i `UTF-8` format. Deklarasjonen som burde vært i starten av dokumentet er vist i listing 3.3.

**Listing 3.3: Deklarasjon av XML-versjon og enkoding**

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
```

Systemet ble testet grundig før det ble produksjonssatt, men å teste for enkodingfeil ble ikke tenkt på. Under integrasjonstestingen var ingen brukere med `æ, ø, å` i navnene tilgjengelig da den eneste brukeren som kunne bli benyttet i Mi side sitt testingmiljø var kandidatens egen.

Både studieadministrativ avdeling, og IT-avdelingen ble informert om denne systemfeilen. Det ble også anbefalt at deklarasjonen ble lagt på XML-dokumentet når det ble opprettet i Mi side sine systemer. Ettersom systemet også brukes til Kark, vil dette imidlertid ikke bli implementert før etter at Kark har testet at systemet deres vil fungerer med en slik endring. Det kan derfor skje tidligst sommeren 2013.

Derfor måtte kandidaten implementere en løsning slik at studentene fikk logget inn. Det ble gjort ved å legge legge til deklarasjonen vist i listing 3.3 i XML-dokumentet før det ble tolket av DPG. Dette har fungert fint og ingen brukere har siden meldt om problemer med innloggingen.

### 3.7.2 Brukerevaluering

Det ble foretatt en liten brukerundersøkelse av den nye løsningen etter snaue fire måneder i bruk. Det ble spurt spørsmål som hvor ofte studentene var inne på DPG, om dataene var presentert fornuftig, hvilken funksjonalitet de var kjent med, hvilke de brukte og hvilken som de følte var unyttig. Hvor ofte de er logget inn på DPG kan også bli sjekket ved å manuelt analysere loggen til DPG, men det er det ingen enkle

verktøy tilgjengelig for å gjennomføre. Det ble også spurt om studentene generelt følte DPG var en nyttig ressurs og om de følte at det var enkelt eller praktisk å bruke med Mi side.

Alle de forespurte har svart at de var inne på DPG to eller tre ganger i uka. De fleste mener at dataene er delvis fornuftig representert. Det er imidlertid rom for forbedring i forhold til hvordan man viser endringer siden sist gang man logget inn, siden studentene klager på at dette kan være vanskelig å oppdage.

Det er også verdt å merke seg at studentene stort sett ikke ser ut til å være kjent med annen funksjonalitet enn det som er direkte på den siden de kommer inn på når de logger inn. Det tyder kanskje på at de ikke er interessert i å utforske hva som finnes på siden. Det virker ikke som om nåværende struktur gjør slik informasjon tilstrekkelig tilgjengelig.

Generelt er det også enighet om at DPG ikke har vært en nyttig ressurs for fagene, og at Mi side kunne vært benyttet til alt. Det kan være på grunn av det foregående punktet; at studentene ikke er kjent med funksjonaliteten i DPG. Dette er med andre ord en utfordring for utviklingen av nye presentasjonsmønstere.

## 3.8 Evaluering

Ved å ta i bruk Mi side som autentiseringsmekanisme for DPG trenger ikke JAFU å lagre informasjon om studentene i egne systemer. Dette gjør at kun er brukerdataen til administratorer som fortsatt håndteres av Webucator. Administratorer er gjerne IT-personellet som drifter DPG, og det har antagelig mye mindre skadeeffekt om bare denne brukerdataen skulle bli kompromittert. Dette kan da sees på som en løsning med mindre risiko, og dermed burde den være bedre sett i et sikkerhetsperspektiv. En annen effekt av denne løsningen er at funksjonaliteten til Mi side er utvidet. DPG kan nå tilbys som et CMS, og ved å lage nye kursmønstere og implementere nye plugins kan man tilby mye ny funksjonalitet til Mi side.

I tabell 3.6 vises det hvordan den nye funksjonaliteten i en modifisert versjon av presentasjonsmønsteret brukt i INF100-F og INF101-F og den sømløse overgangen til Mi side er fordelt. Ingen bakgrunnsfarge viser funksjonalitet som ikke skal benyttes og de med bakgrunnsfarge er funksjonalitet som skal benyttes. Når det gjelder litteraturlisten i Mi side mot pensumlisten i DPG så er det et krav fra Studieadministrativ Avdeling at dette blir benyttet på Mi side. Det er ikke så lett å finne frem til denne på Mi side, så det var et ønske om å også ha den tilgjengelig i DPG, men det er da mulighet for inkonsistens av data ved å måtte lagre det to steder. En god forbedring ville vært om presentasjonsmønsteret var satt opp til å automatisk hente

*Tabell 3.6:* Funksjonaliteter med den sømløse overgangen og modifisert presentasjonsmønster for INF100-F og INF101-F

| Mi side | DPG |
|---|---|
| Grupper | |
| Meldingstjeneste | Meldinger |
| Fillager | Arkiv (Forelesningsnotater/oppgaver) |
| E-post | |
| Litteraturlister | Pensumliste |
| Kalender | |
| Selvlaget rute: Kursinfo | Oversikt over hjelpemidler (Litteratur/-Nettsteder/Programvare)<br>Kontaktinfo<br>Eksamensinfo |
| User tracking | |
| Selvlaget rute: fremdriftsplan | Fremdriftsplan og Ukeplan |
| Innlevering | Innlevering 3rd party |
| Forum | Forum 3rd party |
| Innlogging | Innlogging |
| Utlogging | Utlogging |

pensumlisten ut fra Mi side.

Den nye arkitekturen gjør koblingen mellom DPG og Webucator løsere, og den tilbyr en enkel måte å fjerne eller legge til en autentiseringsmekanisme. DPG er ikke bare tenkt til å kunne være brukt som et LMS slik Webucator er. Det å også kunne tilby DPG som en annen type CMS uten å kun være avhengig av å bruke vårt eget brukerhåndteringssystem vil øke nytteverdien til DPG.

For å få denne koblingen til å fungere har ikke Webucator blitt endret. I og med at det ble gjort en endring i filterene som avgjør tilgang på presentasjonsnivå, se avsnitt 3.6.1, så fungerer ikke dataen man får fra Webucator på presentasjonsnivå. Å logge inn som administrator fungerer fordi de uansett har tilgang til alle presentasjoner. Dette er noe som må endres i Webucator dersom det skal brukes for å håndtere *reader*'e og *publisher*'e igjen.

Feilen ved bruk av feil tegnsett under tolkinga av XML-fila fra tilbakekallet til Mi side viser at testingen av systemet ikke var godt nok gjennomført. Det er vanskelig å tenke på alle mulige situasjoner som kan føre til at ting går galt. Det er første gang kandidaten har fått en slik feilmelding, men å sjekke tegnsett, og å ikke anta for mye om inndata fra andre systemer er noen erfaringer å ta med seg.

# 4

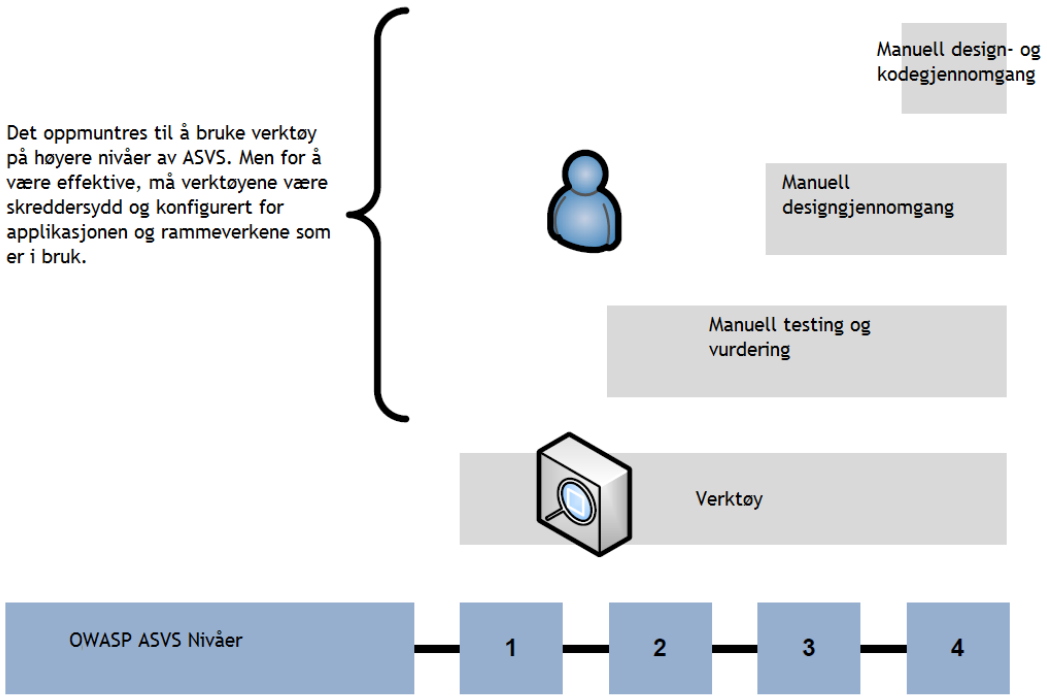
# Sikkerhetsevaluering med ASVS

Dette kapittelet tar for seg hvordan man kan måle sikkerheten i en applikasjon. Det skal, med utgangspunkt i OWASP Application Security Verification Standard(ASVS) [33], utføres en sikkerhetsevaluering av DPG. Dette kapittelet skal også se på hvordan sikkerheten kan forbedres i DPG basert på funnene i analysen. Det ønskes også å bidra til at gjennomføringen av en slik kvalitetssikring blir lettere i fremtiden.

## 4.1 OWASP ASVS

ASVS er en standard utviklet av OWASP. Hovedmålet med prosjektet er å normalisere intervallet i dekning og nivå av strenghet tilgjengelig i markedet når det gjelder å utføre sikkerhetsverifisering av en webapplikasjon med en kommersielt gjennomførbar åpen standard.

ASVS har fire nivåer, som vist i figur 4.1. Hvert nivå innfører nye krav som må verifiseres, samt at man i større grad må være forsikret om at hvert krav er verifisert. For eksempel trenger man på nivå 1 kun å kjøre automatiserte verktøy, og manuelt verifisere resultatene fra disse, for at et krav skal bli verifisert. Riktignok må dette verktøyet være i stand til å kunne avdekke vanlige sårbarheter knyttet til det spesifikket kravet. På nivå 4 må man utføre tester, manuelt gjennomgå all koden og utføre en designanalyse for å verifisere at et krav er verifisert.




*Figur 4.1:* OWASP ASVS Nivåer

Omfanget av verifiseringen øker også med nivået. På nivå 1 inkluderes kun den koden som er utviklet eller modifisert under utviklingen av applikasjonen. På nivå 4 er det all kode som er assosiert med applikasjonen, inkludert rammeverk, biblioteker, kjøretidsmiljøer, utviklingsverktøy, byggeverktøy og distribueringsverktøy. Det man kan ekskludere på nivå 4 er kode for platform programvare, applikasjonsserver, databaseserver, virtuel maskin eller operativsystem som har mottatt en betydelig mengde offentlig ettersyn.

## 4.2 Gjennomføring av en ASVS analyse på DPG

I denne ASVS analysen velges det å se på alle kravene som må verifiseres på nivå 4 og å bruke alt fra automatiserte verktøy til manuell design- og kodeanalyse. For å ikke gjøre omfanget av oppgaven for stor velges det å ikke inkludere annen kode enn det som er utviklet eller modifisert under utviklingen av applikasjonen når analysen gjennomføres. Det vil si at det ikke blir foretatt analyse av tredjeparts biblioteker og rammeverk som DPG benytter seg av.

Da ASVS rapporten skulle være på norsk var det hensiktsmessig å oversette alle kravene til norsk.

Gjennomføringen av verifiseringen ble gjennomført på følgende måte

1. Introduksjonen til rapporten blir laget, der kreves det følgende
   - Applikasjonen som skal testes blir identifisert.
   - ASVS nivå, som man ønsker å oppnå, blir identifisert.
   - Forretningsrisikoer for DPG blir identifisert, se avsnitt 4.2.1.
   - Restriksjoner for å redusere omfanget av oppgaven blir introdusert: det blir definert at at kode som ikke er utviklet ved JAFU skal antas som sikker.
2. En manuell analyse av sikkerhetsarkitekturen blir gjennomført, se avsnitt 4.2.2.
3. Statiske og dynamiske analyseverktøy blir kjørt for å automatisk finne sikkerhetshull i DPG.
4. Funnene fra verktøyene blir koblet til de respektive ASVS kravene de eventuelt vil føre til at ikke blir verifisert.
5. Kravene ble forsøkt verifisert en og en.
   - OWASP Testing Guide og Code Review Guide ble benyttet for å assistere testingen, se avsnitt 4.2.3.
   - Der det var hensiktsmessig ble dynamiske verktøy (ZAP) brukt ved manuell testing av applikasjonen.

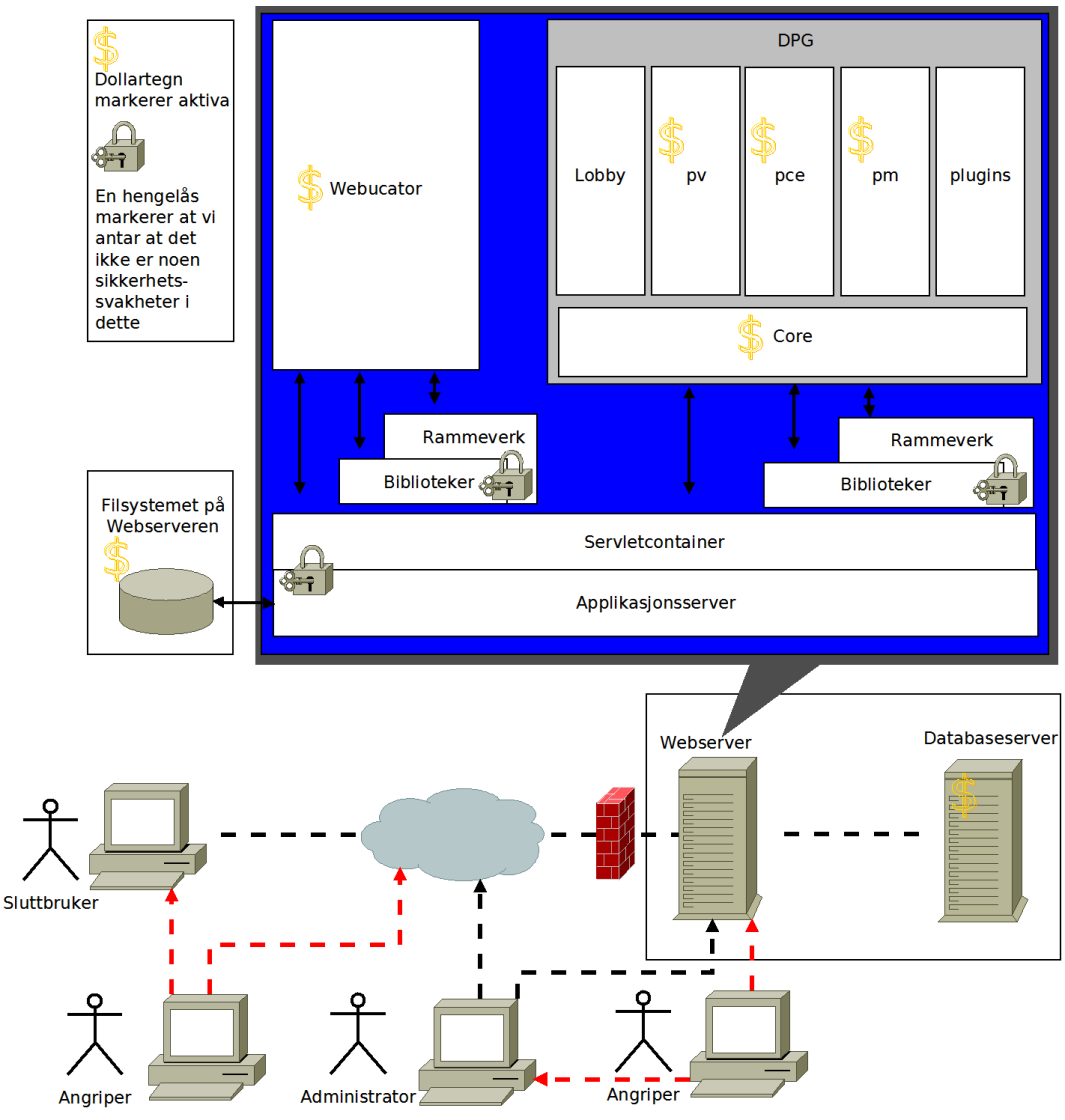
### 4.2.1 Forretningsrisikoer

Et av kravene til rapporten er at de viktigste forretningsrisikoene blir identifisert. For DPG, som i dag blir benyttet som CMS for fjernundervisning i noen kurs ved UiB, er følgende forretningsrisikoer identifisert:

- Systemet blir utilgjengelig for brukere. Det kan for eksempel være på grunn av programmeringsfeil, tjenester som er nede (som Webucator) eller tjenestenekt-angrep.
- Sensitiv brukerdata som navn og passord lekkes ut.
- Innholdet blir modifisert, eller slettet, av uautoriserte brukere eller ved en feil.
- Brukere får tilgang til innhold de egentlig ikke er autorisert til å kunne se. Det kan for eksempel være løsningsforslaget på en oppgave som blir tilgjengelig for tidlig.

### 4.2.2 Gjennomgang av sikkerhetsarkitektur

I figur 4.2 vises en høynivåsoversikt av arkitekturen i DPG. Applikasjonskomponentene er her identifisert, noe som oppfyller første krav i ASVS. Stien spørringer fra en sluttbruker kan ta er også vist i denne figuren (det er et krav fra nivå 3). Denne høynivåoversikten kan verifikatoren lage dersom applikasjonsutvikleren ikke tilbyr dette.

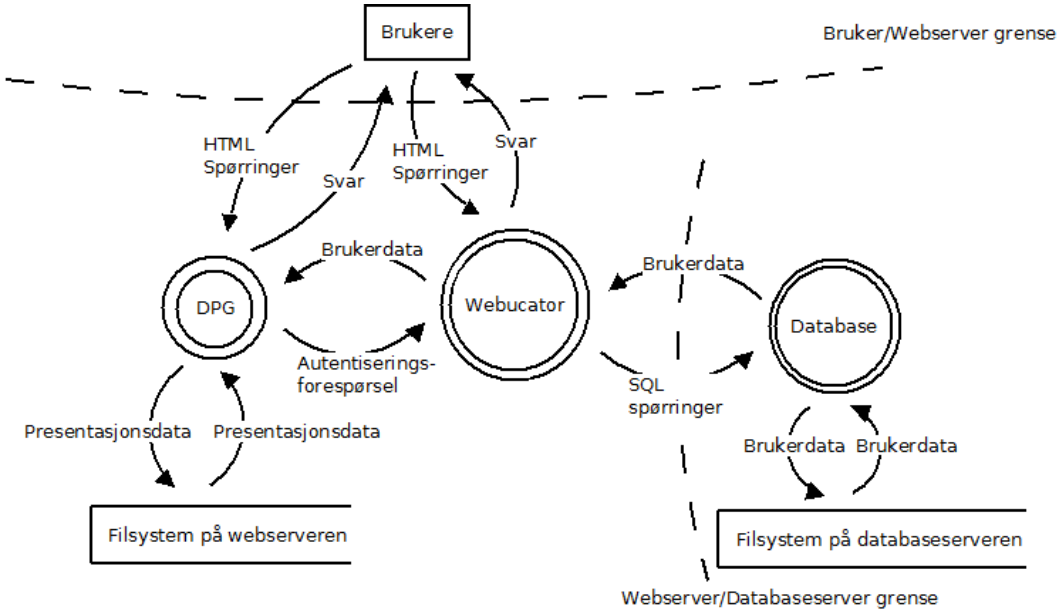



*Figur 4.2:* DPG Arktitekturen

### Trusselmodellering

På nivå tre og fire krever ASVS at trusselmodellering (eng. *threat modeling*) er gitt. Det er ingen nøyaktig krav til hva trusselmodelleringen skal innebære, men de definerer trusselmodellering som en teknikk som består av å supplere dokumentasjonen av sikkerhetsarkitekturen med å identifisere:

- trusselagenter (eng. *threat agents*) - Trusselagenter er definert i Vårdal sin masteroppgave [51].
- sikkerhetssoner - Sikkerhetssoner er definert med de stipplede linjene i figur 4.3.
- sikkerhetskontroller - Sikkerhetskontroller er definert i Løvik sin masteroppgave [23].
- viktig teknisk aktiva - All aktiva er identifisert i figur 4.2. Teknisk aktive er noen av modulene til DPG (*pv*, *pce*, *pm* og *core*) og Webucator.
- viktig forretningsaktiva - All aktiva er identifisert i figur 4.2. Forretningsaktiva er databaseserveren og filsystemet på webserveren.




*Figur 4.3:* Dataflyten i DPG

### 4.2.3 Testing og Code Review guidene

OWASP Testing Guide [39] og Code Review Guide [37] ble benyttet for å assistere i den manuelle testingen. For eksempel beskriver testingguiden en test som tester for logut- og nettlesercache-håndtering. Testen har kodenavnet `OWASP-AT-007`, AT står her for autentiseringstesting. Det er hele 5 av kravene i ASVS som helt eller delvis kan bli verifisert ved å gjennomføre den, disse er listet opp i tabell 4.1.

*Tabell 4.1:* Krav som blir testet av OWASP-AT-007

| Nummer | Beskrivelse |
|---|---|
| 2.11 | Verifiser at etter en administrativt konfigurerbar periode utløper autentiseringsakkreditiver. |
| 3.2 | Verifiser at sesjonen blir ugyldiggjort når en bruker logger ut. |
| 3.3 | Verifiser at sesjoner blir ugyldiggjort etter en spesifisert periode uten aktivitet. |
| 3.5 | Verifiser at alle sider som krever autentisering for å få tilgang har utloggingslenker. |
| 3.9 | Verifiser at sesjons-ID-en endres eller slettes ved utlogging. |

### 4.2.4 Sikkerhetspolicy

Det er to av ASVS sine krav, se tabell 4.2, som direkte krever at en sikkerhetspolicy er definert i prosjektet. Ved JAFU er det ingen eksplisitt policy som utviklere må følge og disse kravene feiler automatisk på grunn av denne mangelen.

*Tabell 4.2:* Krav som krever en sikkerhetspolicy

| Nummer | Beskrivelse |
|---|---|
| 7.9 | Verifiser at det er en eksplisitt policy for hvordan kryptografiske nøkler blir forvaltet (for eksempel generert, distribuert, tilbakekalt, utløpt). Verifiser at denne policyen håndheves riktig. |
| 9.2 | Verifiser at listen med sensitiv data som behandles av programmet blir identifisert, og at det er en eksplisitt policy for hvordan tilgang til disse dataene skal kontrolleres, og når disse dataene må være kryptert (både ved lagring og transport). Verifiser at denne policyen håndheves riktig. |

### 4.2.5 Analyseverktøy

Å bruke verktøy for å utføre en grundig analyse av applikasjonen er anbefalt. På ASVS nivå 1 er det tilstrekkelig for å godkjenne ett krav å kjøre et verktøy som kan teste applikasjonen for den spesifikke svakheten. På nivå 1 krever også ASVS at en avbilding fra verktøyets funksjoner til kravene i ASVS er gitt for å forsikre om at alle kravene blir testet.

Analyseverktøyene kan deles inn i to hovedgrupper, dynamiske og statiske verktøy. Det er også laget et verktøy som skal hjelpe til med å analysere resultatene fra de forskjellige verktøyene. De neste avsnittene skal diskutere bruken og erfaringer fra disse verktøyene.

#### Dynamiske verktøy

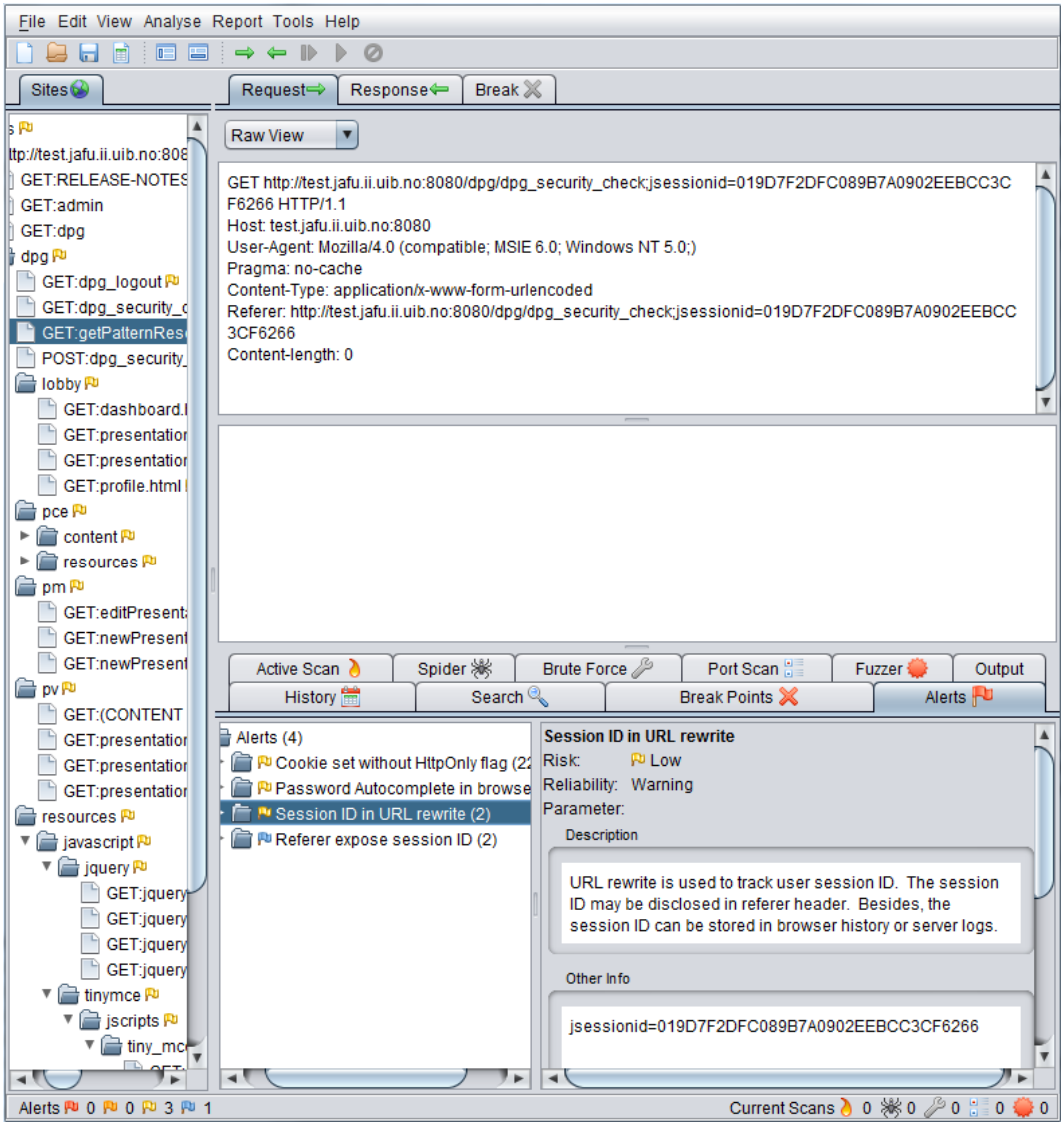
Dynamiske verktøy tester applikasjonen mens den kjører. En testserver som kjører en instans av Apache Tomcat [9] med DPG og Webucator er satt opp. Det er også en testdatabase med brukerdata og testpresentasjoner tilgjengelig for DPG. Dette er gjort for at testserveren skal oppføre seg tilnærmet likt produksjonsserveren DPG er beregnet for å kjøre på.

#### Zed Attack Proxy

OWASP Zed Attack Proxy (ZAP) [41] er et verktøy til bruk ved penetreringstester av webapplikasjoner.

ZAP er meget enkelt å komme igang med. Fremgangsmåten som ble benyttet for å komme igang med å finne sårbarheter:

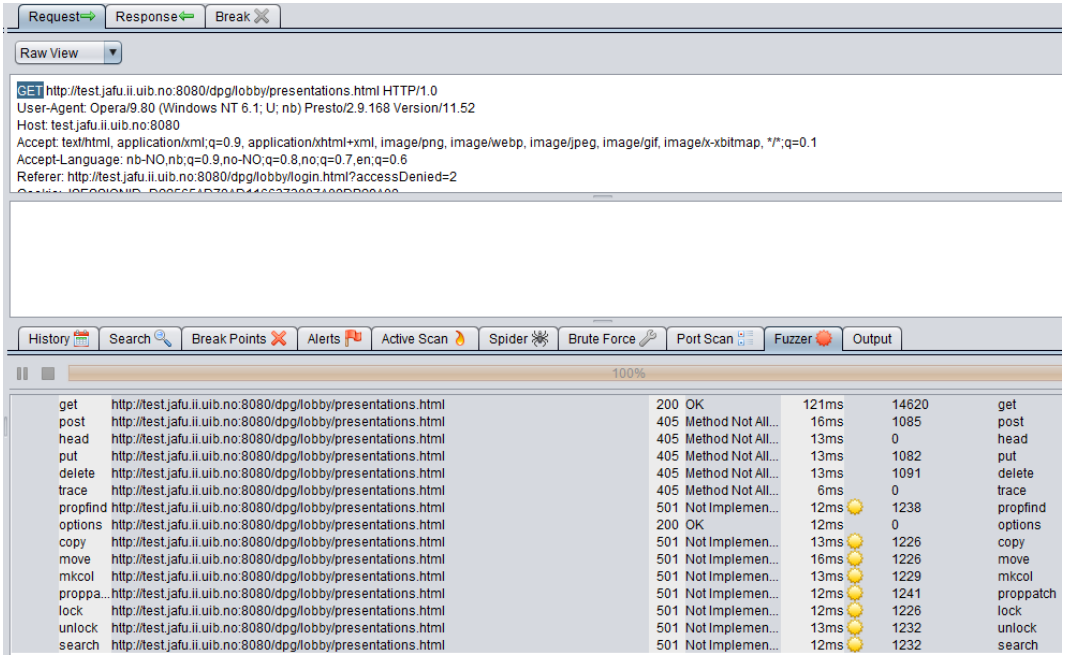
1. Installerte ZAP. ZAP kan installeres på flere måter. Den enkleste måten var å benytte en Windows installer.
2. Satt opp nettleseren til å benytte ZAP som en proxy.
3. Brukte nettleseren til å manuelt utforske all funksjonalitet på siden.
4. Brukte edderkopp(eng. *spider*) funksjonen til å finne URLer og funksjoner som ikke ble oppdaget under den manuelle utforskningen.
5. Brukte *Brute Force* skanneren for å lete etter filer eller mapper som ikke er referert til via noen URL i applikasjonen.




*Figur 4.4:* Skjermbilde av ZAP

6. Kjørte den aktive skanneren for å finne svakheter. De fire forskjellige mulige sårbarhetene avdekket av skanneren vises i rammen nederst på figur 4.4.

ZAP har også to andre funksjoner som ble brukt under den manuelle penetreringstestingen etter sårbarheter.



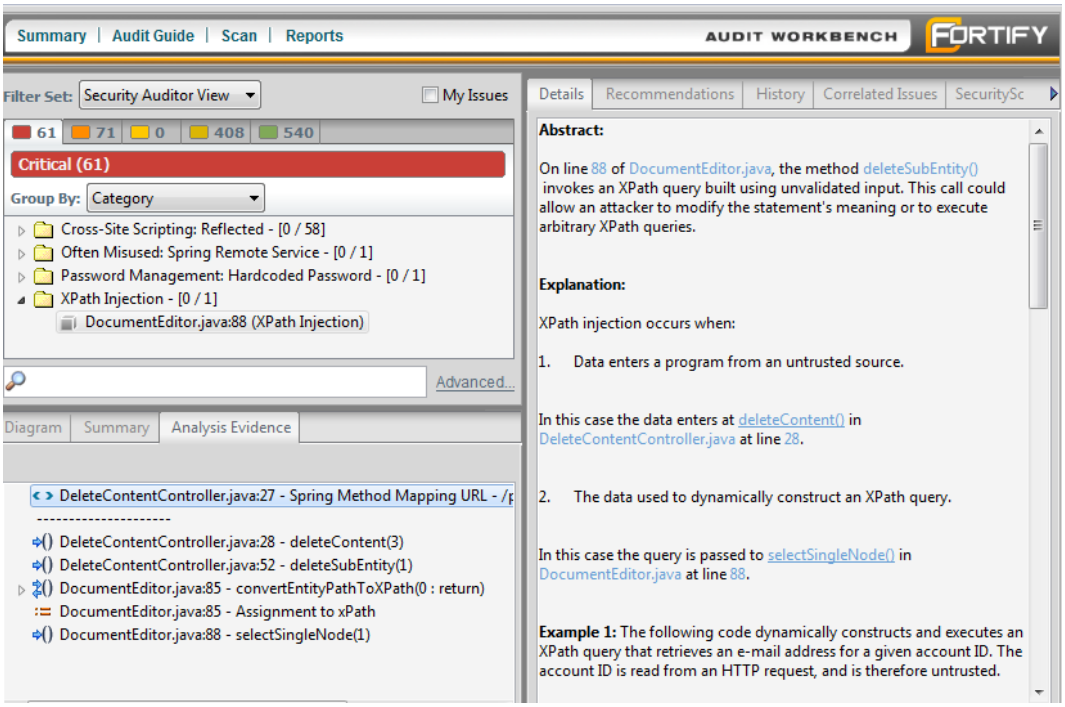

*Figur 4.5:* Skjermbilde av *Fuzzer* verktøyet i bruk

*Break Points* Man kan stoppe en forespørsel fra nettleseren, eller et svar fra applikasjonen, og endre på det før det sendes videre. Dette kan brukes til å for eksempel unnvike valideringen på klientsiden ved å endre på dataene etter at klienten har sent dem.

*Fuzzer* Denne funksjonen kan brukes til å sende store mengder ugyldig eller uventet data til applikasjonen. Fuzzeren kommer med mange forskjellige predefinerte datasett som sett for å teste etter SQL-injeksjoner, XML-injeksjoner eller XSS sårbarheter. Et eksempel på bruken av dette er vist i figur 4.5. Her blir det forsøkt å teste alle mulige HTTP metoder på en gitt URL. Spørringen vises øverst i bildet, her er GET markert slik at fuzzeren vet at det er det ordet som skal erstattes. Fuzzeren prøver så alle HTTP-metodene den har i listen sin.

### Statiske verktøy

Statiske verktøy tester applikasjonen ved å analysere kildekoden uten å eksekvere den. De to programmene som er benyttet er Fortify [5] og Findbugs [45].




*Figur 4.6:* Fortify SCA

### Fortify Static Code Analyzer

I figur 4.6 vises et skjermbilde av Fortify etter at en analyse har blitt kjørt på DPG. Fortify har veldig mange forskjellige alternativer for å gruppere sårbarheter. Den overliggende grupperingen er på risiko og har fire grader: lav, medium, høy og kritisk. I figur 4.6 ser man at det er de kritiske sårbarhetene som er i fokus, og disse er så gruppert etter kategori.

Fortify fant i alt 540 mulige sårbarheter i DPG, de kategoriene med flest funn er vist i tabell 4.3. Ut ifra denne listen kan man se at mange av funnene ikke direkte er interessante for en ASVS analyse, men de påpeker mulige programmeringsfeil. Dette gjelder da nummer 1, 6 og 7 i denne listen, som gjør opp for 178 av de totalt 540 sårbarhetene som er funnet.

### Findbugs

FindBugs [45] ble også benyttet som statisk analyseverktøy. FindBugs er i hovedsak ment for å finne programmeringsfeil, og har ikke mange mønstere for å oppdage sik-

*Tabell 4.3:* Oppsummering av de svakhetene Fortify fant flest av

| Nummer | Kategori | Antall |
|---|---|---|
| 1 | Dead Code: Unused Field | 123 |
| 2 | Cross-Site Scripting: Reflected | 58 |
| 3 | System Information Leak: HTML Comment in JSP | 53 |
| 4 | System Information Leak | 44 |
| 5 | Cross-Site Request Forgery | 33 |
| 6 | Unreleased Resource: Streams | 28 |
| 7 | Missing Check against Null | 27 |
| 8 | Log Forging (debug) | 25 |
| 9 | Trust Boundary Violation | 19 |
| 10 | Log Forging | 18 |
| 11 | Poor Error Handling: Overly Broad Throws | 13 |
| 12 | Path Manipulation | 11 |
| 13 | JavaScript Hijacking: Vulnerable Framework | 10 |
| 14 | Poor Logging Practice: Logger Not Declared Static Final | 10 |

kerhetssvakheter som er relevante for en ASVS rapport. Det ble heller ikke rapportert om noen mulige sårbarheter som var relevante for rapporten.

### Avbildningsverktøy

Resultatene fra de forskjellige analyseverktøyene var upraktisk kategorisert i forhold til at man prøver å gjennomføre en ASVS analyse. En løsning som kunne løse dette problemet var å lage en applikasjon som avbildet resultatene fra verktøyene til de forskjellige ASVS kravene. Resultatene fra de tre verktøyene som ble brukt var mulig å hente ut i XML format. Det ble så laget en applikasjon som tolket disse dataene slik at man kan analysere alle funnene sammen, og lettere kunne verifisere kravene i ASVS.

Et eksempel på slike avbildinger kan vises i listing 4.1. En avbilding definerers ved å spesifisere en sårbarhet fra et spesifikt verktøy og knytte det opp mot et krav i ASVS. I dette tilfellet er det to sårbarheter som er avbildet fra Fortify og ZAP, disse er delvis vist i henholdsvis listingene 4.2 og 4.3, som avbildes til krav 8.10 og 9.1 i ASVS. En sårbarhet blir identifisert fra to strenger; en som forteller hvilket verktøy den kommer fra, og en som spesifiserer nøyaktig hvilken sårbarhetsgruppe det er. Fra Fortify velges det både `Type` og `Subtype`, se listing 4.2, for å spesifisere en sårbarhet, fra ZAP er det `alert` elementet, se listing 4.3, som blir brukt.

Dersom man har en komplett avbilding fra de verktøyene man bruker til ASVS vil

det bli mye enklere å foreta en analyse. Om man for eksempel ønsker å verifisere krav 8.10, kan man spørre etter alle resultater fra alle verktøy som omhandler dette kravet. Det vil gjøre det mye enklere å filtrere vekk andre resultater ifra verktøyene enn de som man spesifikt er interessert i for det kravet man ønsker å verifisere.

**Listing 4.1: Avbildinger fra sårbarheter i Fortify og ZAP til ASVS**

```
<avbilding>
  <verktoy>Fortify</verktoy>
  <saarbarhet>
    System Information Leak:Overly Broad SQL Logging
  </saarbarhet>
  <krav>8.10</krav>
</avbilding>
<avbilding>
  <verktoy>ZAP</verktoy>
  <saarbarhet>
    Password Autocomplete in browser
  </saarbarhet>
  <krav>9.1</krav>
</avbilding>
```

**Listing 4.2: XML Data om en sårbarhet fra en Fortify rapport**

```
<Vulnerability>
  <ClassInfo>
    ...
    <Type>System Information Leak</Type>
    <Subtype>Overly Broad SQL Logging</Subtype>
    ...
  </ClassInfo>
  ...
<Vulnerability>
```

**Listing 4.3: XML Data om en sårbarhet fra en ZAP rapport**

```
<alertitem>
    ...
    <alert>Password Autocomplete in browser</alert>
    ...
</alertitem>
```

### 4.2.6 Resultat av analysen

ASVS har sine krav delt inn i 14 forskjellige kategorier. Resultatene blir oppsummert i tabell 4.4 og i de påfølgende avsnittene vil dette bli forklart i større detalj.

*Tabell 4.4:* Oppsummering av resultatene

| Krav-gruppe | Oppsummering |
|---|---|
| Dokumentasjon | Dokumentasjonen til DPG er akseptabel og ok. |
| Autentisering | Det er mange sikkerhetssvakheter her , spesielt tilknyttet selve passordet. |
| Sesjonshåndtering | Disse kravene virker det ikke som om utviklerene har tenkt på, noen har blitt håndtert med forholdsvis standard konfigurering av Tomcat. |
| Tilgangskontroll | Alle disse kravene ble verifisert. |
| Inndatavalidering | Langt ifra all inndata er validert, noe som må rettes opp i for at alle disse kravene skal bli godkjent. |
| Utdata-enkoding/ unnslipping | Veldig lite utdata er tilstrekkelig kontrollert. |
| Kryptografi | Ingen kryptografiske funksjoner eller moduler er benyttet i DPG. |
| Feilhåndtering og logging | Dette er ikke foretatt i henhold til ASVS sine krav. |
| Databeskyttelse | Her er det mange mangler, og lite som blir godkjent. Virker ikke som om dette er vurdert under utviklingen. |
| Kommunikasjons-sikkerhet | TLS er ikke brukt i DPG. |
| HTTP-sikkerhet | Ingen av kravene i denne kategorien ble verifisert. |
| Sikkerhets-konfigurasjon | Virker greit med Spring Security. Ingen mangler ble funnet. |
| Ondsinnet kode | Ingen ondsinnet kode ble funnet, men det kan snike seg inn via bakveier. |
| Intern sikkerhet | Ingen direkte sikkerhetshull oppdaget, men det er vanskelig å verifisere kravene. |

#### Dokumentasjon

Kravene for dokumentasjonen til applikasjonen er stort sett oppfyllt. Kravet om dokumentasjon av høynivås-arkitektur kan verifikatoren selv lage, resten må være gitt i prosjektet. Det som mangler er at tredjepartskomponenter er definert i form

av forretningsfunksjonene og/eller sikkerhetsfunksjonene som de tilbyr.

### Autentisering

Her er det få av kravene som blir godkjent. Noen sikkerhetssvakheter som ble funnet ved verifisering av disse kravene: Autocomplete er ikke slått av i passordfelt, ingen maksimum antall påloggingsforsøk som forhindrer bruteforce av passordet, ingen krav til passordstyrke, ingen krav til re-autentisering før sensitive operasjoner som å slette en presentasjon, autentiseringsakkreditivene utløper aldri, ingen autentiseringsbeslutninger blir logget, kontopassordene blir verken saltet eller hashet før lagring.

### Sesjonshåndtering

De fleste kravene her har nok ikke utviklerene tenkt på engang da det er Tomcat manageren som tar seg av det meste som har med sesjonshåndtering å gjøre, og kun standard konfigurering av denne er brukt. Det medfører at det er flere av kravene som ikke blir godkjent. Dette gjelder blant annet

- ugyldiggjøring av sesjons-ID-en etter en administrativt-konfigurerbar maksimal periode uavhengig av aktivitet
- URL-omskrivning av sesjoninformasjonskapsler
- endring av sesjons-ID-en ved innlogging og re-autentisering.

### Tilgangskontroll

I denne gruppen er ingen sikkerhetssvakheter oppdaget. Når det gjelder sikkerhet i DPG virker det som om det er her hovedfokuset har vært.

### Inndatavalidering

Noe inndatavalidering er foretatt i DPG, men kravene avdekker flere muligheter for sikkerhetshull. Det kreves at flere retningslinjer følges, blandt annet at et positivt valideringsmønster er definert og brukes på all inndata, at et tegnsett er spesifisert for alle inndatakilder, at alle valideringsfeil er logget og at all inndata er normalisert for alle nedstrømsdekodere eller -tolker før validering.

### Utdata-enkoding/unnslipping

Her ble det funnet flere mangler fordi utdata stort sett ikke er kontrollert. Det virker som om man antar at all data som er lagt inn i mønster og presentasjoner er til å stole på, og at man ikke trenger å kontrollere utdata. Her er det mange mulige sikkerhetshull.

### Kryptografi

Ingen kryptografiske funksjoner eller moduler er benyttet i DPG. All data som sendes, mottas og lagres er i klartekst. Dette er helt klart en sikkerhetsrisiko og medfører at kravene i denne kategorien ikke blir verifisert. Det kreves blandt annet sikker lagring av passord, med hashing og bruk av salt, TLS og en eksplisitt policy for hvordan kryptografiske nøkler bli forvaltet.

### Feilhåndtering og logging

Både logging og feilhåndtering er ikke behandlet slik ASVS krever. Det kan være fordi utviklerene ikke har hatt noen standard å forholde seg til, og dermed ikke har tenkt på disse tingene.

### Databeskyttelse

Nok ett område som ikke virker vurdert av utviklerene. Sensitiv data er ikke definert i DPG, og data som navn og passord, som helt klart burde være sensitiv er ikke behandlet slik ASVS krever.

### Kommunikasjonssikkerhet

TLS er ikke brukt i DPG. Dermed feiler alle krav i denne kategorien.

### HTTP-sikkerhet

Ingen krav ble verifisert i denne kategorien heller. Nok en gang er dette noe som ikke er påtenkt fra utviklere, da de stort sett har tenkt på funksjonalitet og ikke sikkerhetskonfigurasjon. Enkle ting som at HTTPOnly og sikker-flagget q(eng. *secure*

*flag*) er brukt på informasjonskapsler er meget enkelt å fikse. Andre ting som burde vært tenkt på er å bruke et CSRF-token, kun tillate et definert sett av HTTP-forespørselmetoder og å passe på at tegnsett er spesifisert i `HTTP response`.

#### Sikkerhetskonfigurasjon

Kravene i denne kategorien er ganske enkle å verifisere, da man antar at rammeverkene benyttet for dette er sikre. Ingen mangler ble her funnet.

#### Ondsinnet kode

Ingen ondsinnet kode ble funnet, men integriteten av tolket kode er ikke bekreftet ved hjelp av sjekksummer eller hasher, noe som kan føre til at ondsinnet kode kan snike seg inn.

#### Intern sikkerhet

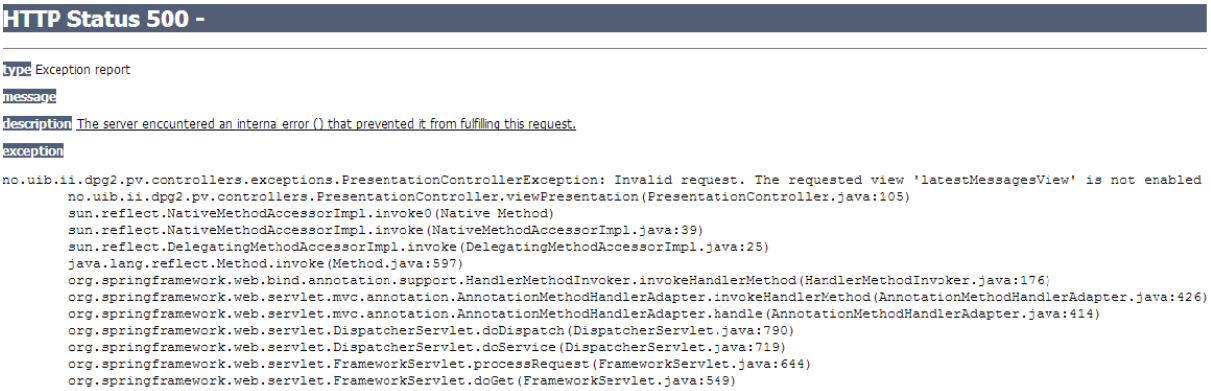
Her ble ingen direkte sikkerhetshull oppdaget, men å verifisere disse kravene er i utgangspunktet ikke lett. Likevel kan to av tre krav i denne kategorien verifiseres, i og med det er antatt at tredjeparts-kode som blir benyttet er sikker. Det siste kravet krever at sikkerhetskontrollgrensesnittene er "enkle nok" å bruke. Dette blir en subjektiv tolkning av den enkelte verifikatoren.

## 4.3 Utbedring av svakheter

Under analysen ble det funnet en del potensielle sårbarheter. Dette avsnittet skal ta nærmere for seg disse problemene, samt det som gjør dem til svakheter og hvordan de kan utnyttes. Deretter presenteres løsninger på disse svakhetene.

### 4.3.1 Feilbehandling

Dersom et internt unntak blir kastet i programmet, hender det at feilen ikke blir fanget og at et stacktrace blir skrevet ut til skjermen, en slik feilmelding vises i figur 4.7. Dette er dårlig praksis og en sikkerhetssvakhet fordi det kan gi en angriper informasjon om hvordan applikasjonen fungerer og dermed kanskje gjøre det lettere å finne svakheter.




*Figur 4.7:* Feilmelding brukeren kan få om ett unntak kastes av applikasjonen

Dersom systemet blir gjort opensource blir dette en vesentlig mindre sikkerhetsrisiko siden all kildekoden og alle detaljer være tilgjengelig for en angriper. Eventuelle detajler om den interne strukturen til applikasjonen vil en angriper i så fall ikke nødvendigvis trenge. Men det er fortsatt ansett som dårlig koding å vise unntak med stacktracer til brukeren.

Det er flere mulige løsninger på dette problemet. Det enkleste er å spesifisere en generell feilside i distribusjonsdeskriptoren(eng. *deployment descriptor*), `web.xml` ved en av metodene vist i listing 4.4. De to første elementene av `error-page` vil fange alle unntak av typen `Exception` eller `Throwable`. Den siste vil fange opp spesifikke HTML-statuskoder, i dette tilfellet `500`, som er for intern serverfeil.

**Listing 4.4: Konfigurering av standard feilside i web.xml**

```
1  <error-page>
2      <exception-type>java.lang.Exception</exception-type>
3      <location>/error.html</location>
4  </error-page>
5
6  <error-page>
7      <exception-type>java.lang.Throwable</exception-type>
8      <location>/error.html</location>
9  </error-page>
10
11 <error-page>
12     <error-code>500</error-code>
13     <location>/error.html</location>
14 </error-page>
```

En annen metode vil være å lage et egendefinert filter som implementerer `javax.servlet.Filter`, som dermed kan puttes inn først i filterkjeden til Spring Security. Det er flere fordeler ved å lage et eget filter:

- Man kan videresende brukeren til forskjellige sider, ikke bare basert på hvilken type unntak som kastes eller HTTP-statuskoden.
- Man kan få større kontroll på loggingen som utføres når unntaket kastes.

### 4.3.2 Autofullfør

Autofullfør (eng. *autocomplete*) er en funksjon som kan fortelle nettleseren om det skal være mulig å lagre dataen som er skrevet inn i dette feltet/skjemaet eller ikke. Dersom autofullfør ikke er slått av, kan en bruker for eksempel få muligheten til å lagre passord/brukernavn. Disse passordene blir så lagret på harddisken av nettleseren, og en angriper kan få tilgang til disse passordene via brukerens datamaskin.

ASVS krever at dette er slått av på passordfelt og for annen sensitiv data. For å slå det av trenger man bare å legge til `autocomplete="off"` i enten `form`, eller `input`-elementet, dette er vist i listing 4.5. Her kan man velge å enten å gjøre det på linje 2, eller linje 5 og 9.

**Listing 4.5: Hvordan man kan slå av autofullfør**

```
1  <form name="loginForm" action="<c:url value='/j_login'/>" method="post"
2  autocomplete="off">
3    <p>
4      <b>Username:</b><br/>
5      <input type="text" name="j_username" value=""/ autocomplete="off">
6    </p>
7    <p>
8      <b>Password:</b><br/>
9      <input type="password" name="j_password" value=""/ autocomplete="↩
           off">
10   </p>
11   <p>
12     <input type="submit" name="Login" value="Login"/>
13   </p>
14 </form>
```

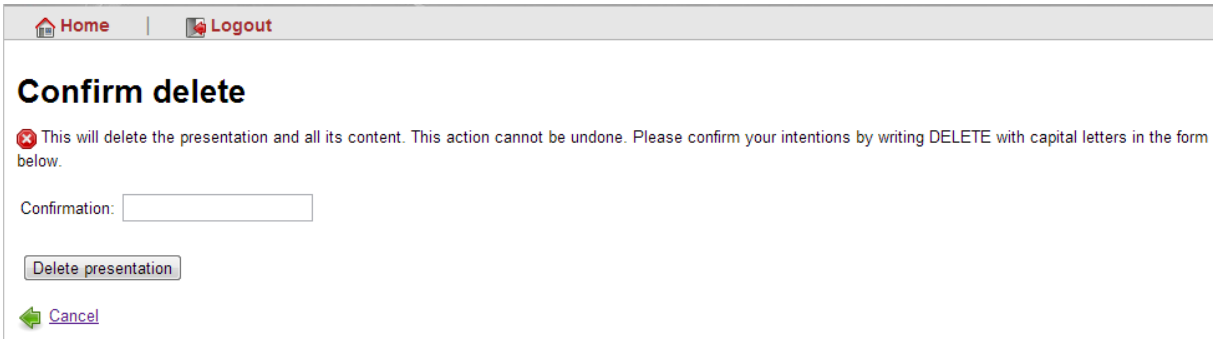
### 4.3.3 Cross Site Request Forgery

CSRF er et vanlig [43] sikkerhetsproblem. ASVS har et eget krav, se tabell 4.5, som krever at en mekanisme som forhindrer CSRF er på plass. Det beskriver i korte trekk mønsteret for synkroniseringstoken, som er forklart og anbefalt i *Cross-Site Request Forgery Prevention Cheat Sheet* [36]. Dette kravet er ikke godkjent for DPG.

*Tabell 4.5:* Kravet for å forsvare mot CSRF

| Nummer | Beskrivelse |
|---|---|
| 11.7 | Verifiser at programmet generer et sterkt tilfeldig token som en del av alle lenker og skjemaer knyttet til transaksjoner eller tilgang til sensitiv data, og at programmet bekrefter tilstedeværelsen av dette tokenet med korrekt verdi for den aktuelle brukeren ved behandling av disse spørringene. |

Et eksempel på en mulig utnyttelse av denne svakheten i DPG er å få en presentasjon slettet. For å enkelt forklare dette angrepet så kan en angriper "lure" en innlogget administrator til å trykke på en lenke som sender beskjed til DPG om å slette en gitt presentasjon. Hvordan det er ment at en presentasjon skal slettes vises i figur 4.8. Kun en innlogget administrator har adgang til denne siden. For at man ikke skal slette presentasjoner ved et uhell må man først skrive inn ordet `DELETE` i tekstboksen og så trykke på knappen.




*Figur 4.8:* Hvordan siden for å slette en presentasjon ser ut

I det eksempelet som her vises vil hele skjemaet være fylt ut av angriperen, og alle detaljer er skjult for en bruker. Koden for dette skjemaet er vist i listing 4.6. Her referer `action`-egenskapen til applikasjonen som skal angripes, og den inkluderer ID-en til presentasjonen som skal slettes. Det eneste som vil vises av dette skjemaet er en knapp der det står "`Klikk her`". Dersom denne knappen trykkes på av en administrator som er innlogget i DPG vil denne presentasjonen bli slettet.

**Listing 4.6: Skjema som sletter en presentasjon**

```
1 <form id="command" action="http://test.jafu.ii.uib.no:8080/dpg/pm/↩
    deletePresentationConfirmationForm.html?pid=deleteme" method="post">
2   <input id="presentationId" name="presentationId" type="hidden" value=↩
      "deleteme">
3   <input id="confirmation" name="confirmation" type="hidden" value="↩
      DELETE">
4   <input type="submit" value="Klikk her">
5 </form>
```

Løsningen som ASVS krever er å bruke et sterkt tilfeldig token som en del av lenken, og at applikasjonen verifiserer at den er med ved alle spørringer. Dette krever at angriperen må gjette seg til riktig token. I og med at dette tokenet er påkrevd å være sterkt og tilfeldig, så skal det være sikkert.

OWASP CSRFGuard prosjektet [38] er laget for nettopp dette. Dette prosjektet tilyr en måte å legge til CSRF-tokener og å validere dem i JavaEE. Den bruker et filter som kontrollerer enhver request før den blir sendt videre til Spring Security sin filterkjede. Ved standard settings generer den en sterk tilfeldig token hver gang en bruker får en ny sesjons-ID. Man kan så inkludere et Javascript-snutt på hver side for at tokenet skal bli lagt til i lenker og skjemaer.

Dette førte f.eks. til at et element med navn `OWASP_CSRFTOKEN` og verdi `3TW3-BIND-4QO9-XSUR-W2GE-RINM-97ID-GUUY` ble lagt til alle lenker på en side som inkluderte dette skriptet. Alle spørringer til sider som da skal være beskyttet blir kontrollert med at de inneholder dette tokenet. Hvis spørringen inneholder tokenet blir spørringen videresendt til Spring Security sin filterkjede. Dersom tokenet ikke blir validert som gyldig for den aktuelle sesjonen blir brukeren videresendt til en standard feilside.

Det er en negativ konsekvens av å bruke et slikt token direkte i URL og det er at bokmerker og eksterne lenker til beskyttede sider ikke vil virke slik man ønsker. I dette eksempelet kunne man brukt tokenet i `form`-elementet(i listing 4.6) da det ville gi beskyttelse mot uønsket sletting av presentasjoner. Det ville ikke beskyttet mot angrep som baserer seg på å hente data der man ikke trenger å bruke et HTML-skjema for å sende inn data.

### 4.3.4 Cross Site Scripting

Cross Site Scripting(XSS) er den mest utbredte sikkerhetsfeilen i webapplikasjoner [42]. Store firmaer og nettjenester som McAfee, Amazon, PayPal og eBay har alle vært (og kanskje fortsatt er) åpne for XSS angrep [54]. Offentlige tjenester i mange land har også vist seg å ha denne typen feil [27].

XSS angrep forekommer når en angriper bruker en webapplikasjon for å sende ondsinnet kode til en bruker. Et script kan da se ut for en bruker, eller rettere sagt brukerens nettleser, til å komme fra en legitim server. Og scriptet blir så kjørt på brukerens maskin. Dermed kan en angriper f.eks. kapre (eng. *hijack*) brukerøkter (eng. *user sessions*), endre den visuelle visningen av nettsiden, omdirigere brukere eller kapre nettleseren til brukeren ved hjelp av skadeprogrammer (eng. *malware*) [42].

ASVS kravet vist i tabell 4.6 er spesifikt ment for å forebygge blant annet en XSS sårbarhet.

*Tabell 4.6:* Kravet for å forsvare mot XSS

| Nummer | Beskrivelse |
|---|---|
| 6.1 | Verifiser at all uklarert data som er utmatet til HTML (inkludert HTML elementer, HTML attributter, javascript dataverdier, CSS blokker, og URI attributter) er ordentlig unnsluppet for den aktuelle konteksten. |

I figur 4.9 vises vinduet som lar `publishers` legge inn en melding til brukere av DPG. Dersom meldingen vist her blir lagt ut, vil resultatet være at en beskjed vil dukke opp i et forgrunnsvindu når en bruker besøker siden som skal vise meldingen, som vist i figur 4.10. Det en angriper kan gjøre for å kapre brukerøkten er å sende denne informasjonen til seg selv, da det er denne sesjons-ID-en som blir brukt for å identifisere en bruker som allerede er logget inn.

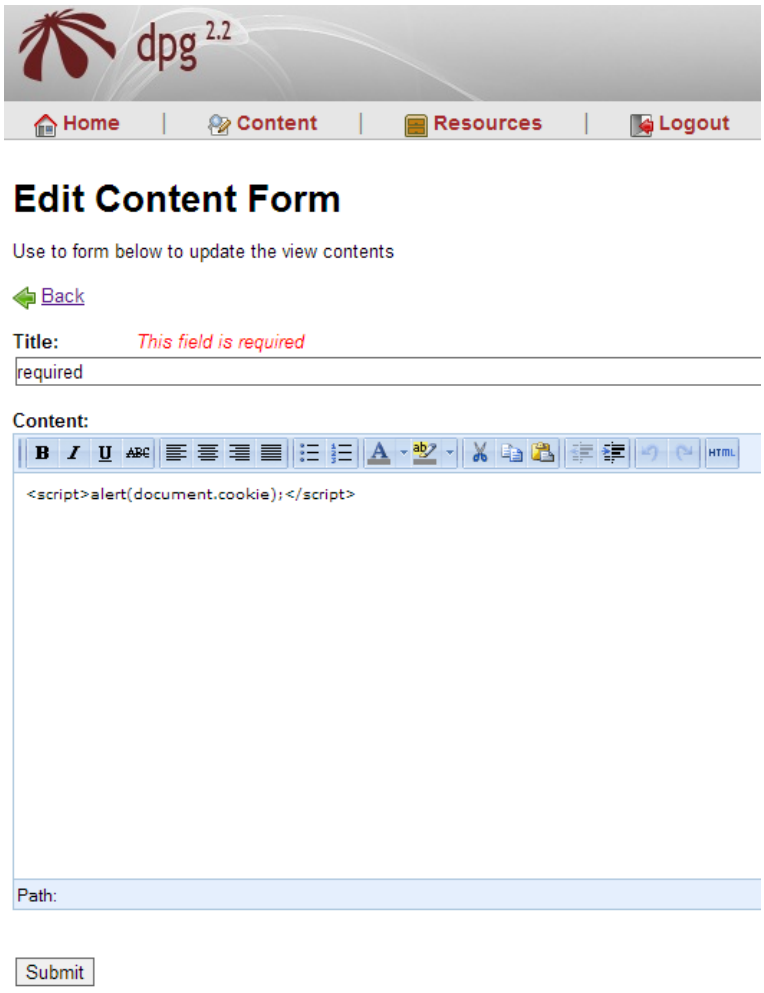
I DPG er det presentasjonsmønsteret som bestemmer hvor en bruker kan legge inn data. Dermed vil mønsteret ha en stor grad av ansvaret for om det foreligger en utnyttbar sårbarhet. I listing 4.7 vises det hvordan entiteten som blir brukt for å åpne for den påviste XSS svakheten blir definert. I listing 4.8 vises hvordan mønsteret spesifiserer at dataene i `messageEntity` skal bli brukt. På linje 5 vises det at den som har laget mønsteret har valgt å slå av unnslippingen. Dette er gjort for at tekstformateringen, som man ser som funksjonalitet i figur 4.9 skal være mulig, antagelig uten viten om at dette skapte en XSS sårbarhet.

**Listing 4.7: Her defineres message entiteten i mønsteret**

```
1 <entity id="messageEntity">
2   <field type="string" required="true">title</field>
3   <field type="xhtml">content</field>
4 </entity>
```

**Listing 4.8: Her brukes message entiteten i en transformasjon**

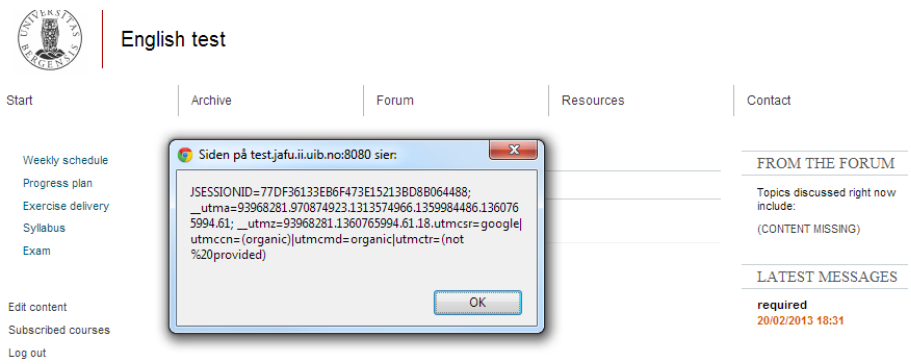
```
1 <li>
2   <h4>
3     <xsl:value-of select="title"/>
4   </h4>
5   <xsl:value-of select="content" disable-output-escaping="yes" />
6 </li>
```




*Figur 4.9:* Her kan et XSS angrep legges inn

Man trenger også forskjellig grad av unslipping avhengig av hvor i HTML koden man skal bruke dataen. OWASP sin *XSS Prevention Cheat Sheet* [44] gir en god oversikt over regler for dette. En god løsning kan være å spesifisere i mønsteret nøyaktig hvordan den spesifikke dataen skal unnslippes basert på hvor den skal benyttes. Det kan så blitt håndhevet når dataen kommer inn i systemet, slik at dataen blir unnslippet med en gang den blir lagret. Det ville kreve at den som lager mønsteret vet, eller har klare rettningslinjer for, hvordan dataen skal unnslippes.

En enklere løsning, men som ikke gir like mye frihet til mønsterene, ville være å bruke OWASP AntiSamy [32] på all xhtml inndata. AntiSamy er et rammeverk som kan brukes for å fjerne ondsinnet kode fra inndata, men beholde lovlig kode. Man




*Figur 4.10:* Dette er resultatet av testen etter XSS sårbarhet som ble lagt inn i figur 4.9

kan fortelle AntiSamy at `HTML`-elementene som editoren viser funksjonalitet for å benytte i figur 4.9 legger til er lovlige, og alt annet er ulovlig. Dermed vil brukeren kunne formatere teksten, men ikke kunne legge inn skadelig kode som, for eksempel, Javascript snutter.

### 4.3.5 Forfalskning av loggen

En angriper kan i teorien forfalske og skrive det han vil inn i loggen ved hjelp av manipulering og enkoding av parametere i URL-en. Det kan for eksempel gjøres ved å endre `page`-variabelen. Et utsnitt av en slik manipulert URL er vist i figur 4.11. Det som kommer i loggen i dette eksempelet er vist i figur 4.12.

```
.../presentation.html?pid=test&page=thispage\%0A\%0AJeg\%20forfalsker\%20loggen!\%0A
```

*Figur 4.11:* Manipulering av URL-en kan forfalske loggen

```
 WARN [http-8080-11] (Presentation.java:124) - No page state found for page '

Jeg forfalsker loggen!
'
```

*Figur 4.12:* Utsnitt av forfalsket logg

Ved å manipulere loggen kan en angriper skjule sine spor og gjøre det vanskelig å oppdage eller analysere et angrep.

For å forhindre logforfalskning er validering av inndata en god forebyggende metode. Dette kan for eksempel forhindre at en angriper bruker en eller annen form for

linjeskift i sitt angrep, noe som gjør at hver linje i loggen starter på en "legitim" måte. En annen metode er å enkode all utdata til loggen.

### 4.3.6 Utloggingslenke på alle sider som krever autentisering

Et krav, se tabell 4.7, er at alle sider som krever autentisering skal inneholde en utloggingslenke. Slik DPG benyttes i dag så gjør alle sider det. Men dette avhenger av mønsteret på presentasjonene. Disse mønstrene kan lett endres, også mens applikasjonen kjører. Feilkonfigurering av DPG, eller mønstrene DPG benytter, kan føre til at noen sider ikke har en utloggingslenke.

*Tabell 4.7:* Kravet om utloggingslenke

| Nummer | Beskrivelse |
|---|---|
| 3.5 | Verifiser at alle sider som krever autentisering for å få tilgang har utloggingslenker. |

For å løse dette kan det kreve endringer i hvordan mønsteret skal valideres. En mulighet vil være å lage en validator som tester at alle sider mønsteret kan produsere vil inkludere en innloggingslenke. En annen mulig metode kan være at når en side blir generert av DPG, men før den returneres til en bruker, vil det sjekkes at den inneholder en utloggingslenke, ellers vil den bli lagt til for eksempel nederst på siden.

## 4.4 Sikkerhetsrammeverk

Spring Security er det eneste rammeverket som tilbyr sikkerhetsfunksjonalitet i DPG i dag. Det eksisterer mange rammeverk som kan tilby forskjellig funksjonalitet, og det ønskes å se på hvilken fordeler man kan oppnå med andre rammeverk.

Fire rammeverk ble sammenlignet: ESAPI, Spring Security, Apache Shiro [11], Apache Commons Validator [10]. En oversikt over funksjonalitetene til de forskjellige rammeverkene vises i tabell 4.8.

OWASP ESAPI er et rammeverk som tilbyr en del forskjellige sikkerhetskontroller. Det tilbys en god del funksjonalitet utenom det som er vist i tabell 4.8, disse er listet opp i avsnitt 2.5. ESAPI tilbyr mye funksjonalitet som kan forbedre sikkerheten i DPG slik at man kan oppnå et ASVS nivå.

Mange av utviklerene bak Shiro bruker Shiro sammen med Spring og det skal være enkelt å integrere med Spring [11]. Shiro skal ha all funksjonaliteten som Spring

*Tabell 4.8:* Funksjonalitene i ESAPI, Spring Security, Apache Shiro, Apache Commons Validator

| Fuksjonalitet | ESAPI | SS | Shiro | CV |
|---|---|---|---|---|
| Autentisering | ☑ | ☑ | ☑ | ☐ |
| Tilgangskontroll | ☑ | ☑ | ☑ | ☐ |
| Hashfunksjoner | ☑ | ☑ | ☑ | ☐ |
| Kryptering | ☑ | ☐ | ☑ | ☐ |
| Inndatavalidering | ☑ | ☐ | ☐ | ☑ |

Security tilbyr i DPG, og i tillegg en krpyteringsfunksjon. Det er ingen store fordeler med å bytte rammeverk av den grunn, og å bruke både Spring Security og Shiro virker overflødig.

## 4.5 Konklusjon

I dette kapittelet har det blitt beskrevet hvordan en ASVS analyse har blitt utført på DPG. Resultatet av analysen er oppsummert og det er påvist mange mangler i DPG. DPG hadde ikke oppnådd noen ASVS nivå, da det var krav som ikke ble tilfredsstilt allerede på nivå 1.

Ved å gjennomføre en ASVS analyse har man fått en systematisk oversikt over sikkerheten i DPG. Tidligere analyser har avdekket en rekke svakheter, men nå finnes det en bedre organisert analyse som man kan jobbe ut ifra. En del potensielle svakheter ved sikkerheten som ikke er vurdert tidligere vil man nå være oppmerksom på.

Alle kravene i ASVS er kanskje ikke like nødvendig å verifisere for å fortsatt ha en sikker applikasjon. Dette kan for eksempel gjelde noen krav til en sentral implementasjon av autentisering, tilgangskontroll og logging, se tabell 4.9. Dette er sett på som god design som gjør applikasjonen mer vedlikeholdsvennlig, og er derfor anbefalt. Det er også krav til loggingen og dokumentasjonen av systemet som ikke direkte øker sikkerheten. ASVS kunne vært delt opp i flere nivåer, eller subnivåer, som reflekterte at ikke alle kravene direkte øker sikkerheten.

Det å utføre en ASVS analyse på nivå 4 var en veldig omfattende jobb, selv om omfanget ble gjort mindre ved å kun velge å se på kildekoden til DPG og Webucator. DPG er ikke et enormt prosjekt, men det benytter seg av over 100 forskjellige tredjepartsbiblioteker. Det å utføre en nivå 4 analyse på alle disse hadde vært et svært stort prosjekt. Restriksjonen gjorde at denne analysen uansett ikke kunne oppnå noe høyere offisielt ASVS nivå enn nivå 1.

*Tabell 4.9:* Krav som ikke direkte er nødvendige for en sikker applikasjon

| Nummer | Beskrivelse |
|---|---|
| 2.5 | Verifiser at alle autentiseringskontroller (inkludert biblioteker som kaller eksterne autentiseringstjenester) har en sentralisert implementasjon. |
| 4.12 | Verifiser at det er en sentralisert mekanisme (inkludert biblioteker som kaller eksterne autoriseringstjenester) for å beskytte tilgang til hver type beskyttet ressurs. |
| 8.9 | Verifiser at det er en enkel implementasjon for logging som er brukt av programmet. |

Dersom et krav feiler, er det greit nok å forklare hvorfor ved å referere til kildekoden eller arkitekturen, eller å komme med konkret eksempel på hvordan det kan utnyttes. Derimot kan det å verifisere et krav, eller komme med tilstrekkelig bevis på at et krav er verifisert, være en veldig vanskelig oppgave. ASVS sier at det, på over nivå 1, skal være "*en begrunnelse for avgjørelsen(et argument for kompletthet og korekthet, med spesifikt bevis)*". OWASP har ingen dokumentasjon, eller eksempler, på hvordan man kan gjøre dette. Det fører til at ASVS kan være vanskelig å gjennomføre for en som ikke er en erfaren sikkerhetsverifikator.

For å utføre en ASVS analyse på nivå 1 kreves det at verifikatoren kjører automatiserte verktøy og er i stand til å verifisere resultatene fra disse. Dersom verktøyene ikke dekker alle kravene, ville analysen måtte blitt utvidet med manuell verifisering av de resterende kravene. Å utføre en slik analyse kan passe for en som ikke har noe erfaring i å gjennomføre sikkerhetsanalyser. Verifikatoren burde likevel være en dyktig systemutvikler og å ha basiskunnskap om sikkerhetsmomentene ASVS krever: autentisering, sesjonsstyring, tilgangskontroll, inndatavalidering, utdata-enkoding/unnslipping, unntakshåndtering, TLS og HTTP-sikkerhet.

ASVS nivå 4 er ment for "*kritiske applikasjoner som beskytter liv og sikkerhet, kritisk infrastruktur eller forsvarsfunksjoner*" [33]. Det er dermed absolutt ikke nødvendig å forsøke å oppnå dette for et LMS som DPG. Nivå 2 er ment for "*applikasjoner som håndterer personlige transaksjoner, utfører forretningstransaksjoner, prosesserer kredittkortinformasjon, eller prosesserer personlig identifiserbar informasjon*" [33]. DPG kan muligens gå under denne beskrivelsen med tanke på to ting:

- Navn på studenter kan lekkes ut, men alle som har tilgang til Mi side vil uansett kunne se denne listen.
- Dersom løsningsforslaget for oppgaver blir lekket ut for tidlig vil det kunne påvirke karaktersetting i faget.

Det er tydelig at det ikke har blitt fokusert stort på sikkerhet under utviklingen av DPG, men heller på at det skal være autentisering og tilgangskontroll. Dette er i hovedsak oppnådd ved å bruke Spring Security og å konfigurere og skreddersy det for DPG. ASVS burde brukes videre som en guide til utviklere ved Jafu. Det burde tilstrebes å oppnå ASVS nivå 1.

En god måte å gjennomføre dette ved JAFU kan være å innføre automatisk sikkerhetstesting i Jenkins [18]. Jenkins er et verktøy som blir brukt ved JAFU til kontinuerlig integrasjon (eng. *Continuous Integration*) [8] av DPG. Jenkins kjører i dag automatiske enhetstester av DPG når noe blir oppdatert i svn [49]. Den kan også bli satt opp med for eksempel Fortify og ZAP. Det vil også være en mulighet å modifisere avbildningsverktøy fra avsnitt 4.2.5 for at Jenkins automatisk kan markere hvilke krav i ASVS som potensielt feiler. Det må da også være mulig for utviklere å markere falske positiver for Jenkins.

I dette kapittelet ble også noen sikkerhetssvakheter utbedret: CSRF, feilbehandling og autofullfør. Det ble også foreslått hvordan man kan utbedre XSS, loggforfalskning og kravet om utloggingslenke. Videre utbedringer for å oppnå ønsket ASVS nivå blir foreslått som videre arbeid i avsnitt 5.3.

# 5

# Evaluering, konklusjon og videre arbeid

## 5.1 Evalureing av mål

I avsnitt 1.6 ble målet for oppgaven definert, og derav noen delmål som kandidaten ønsket å oppnå. Disse var som følger:

1. Utforsk OWASP prosjekter som kan benyttes
2. Utføre en systematisk kvalitetssikring/testing for sikkerhetsproblemer basert på ASVS
    - Statisk testing
    - Dynamisk testing
    - Manuell kodegjennomgang
    - Manuell designanalyse
3. Forbedre DPG arkitektur og design
    - Forbedre brukerhåndteringen
        - Analysere dagens løsning
        - Implementere en single-sign-on-løsning med Mi side
    - Forbedre sikkerhetsproblemer funnet under testingen

4. Evaluere verktøy og metoder brukt

Det første delmålet var å utforske OWASP prosjekter som kunne være nyttige i arbeidet. En rekke prosjekter som har blitt brukt i arbeidet blir identifisert og beskrevet i kapittel 2. Det er også foreslått hvordan ASVS kan bli benyttet videre ved JAFU-prosjektet for å opparbeide en systemutviklingsprosess som skaper sikrere programvare.

Det andre delmålet var å utføre en analyse basert på ASVS. En grundig analyse av DPG, basert på ASVS, har blitt utført og er beskrevet i kapittel 4.

Det tredje delmålet var å forbedre DPG arkitektur og design. Dette var et todelt delmål. For det første var det ønsker ved JAFU å forbedre brukerhåndteringen. Kandidaten har utvidet DPG med en sømløs overgang til Mi side, som dermed lar DPG benytte seg av UiB sin brukerhåndtering. For det andre ønsket kandidaten å forbedre sikkerhetsproblemer funnet under ASVS analysen. Det ble avdekket en rekke problemer, og noen av disse ble løst. Problemene med feilbehandlingen, bruken av autofullfør for sensitiv data og mangelen på bruken av et CSRF-token ble implementert. Det ble også foreslått løsninger på problemene vedrørende XSS, forfalskning av loggen og kravet om utloggingslenke på alle sider som krever autentisering.

Det siste delmålet var å evaluere verktøy og metoder brukt i oppgaven. Dette ble gjort til en viss grad i kapitlene 3 og 4, og en mer systematisk evaluering ble gjort i avsnitt 5.2.

Kandidaten føler med dette at målet for oppgaven er oppnådd.

## 5.2 Vurdering av teknologier, rammeverk og hjelpemidler

Arbeidet i denne oppgaven kan deles inn i to deler:

- Systemutvikling
  - Implementasjonen av sømløs overgang til Mi side i kapittel 3.
  - Utbedring av sikkerhetsproblemer i DPG i kapittel 4.
- Sikkerhetsanalyse
  - Analysen utført i kapittel 4.

Det er derfor valgt å gruppere verktøyene etter hvilken av disse to prosessene de ble brukt under. En tabell som oppsummerer styrker og svakheter slik kandidaten har erfart det i den konteksten de her er brukt, ble utarbeidet for hver gruppe. Dette er henholdsvis tabellene 5.1 og 5.2. Noen verktøy forekommer i begge gruppene, og det er fordi de har blitt brukt på forskjellige måter i hver del.

*Tabell 5.1:* Under systemutviklingsprosessen

| Navn | Styrker | Svakheter |
|---|---|---|
| Maven | Forenkler byggingen av systemet. | Et bibliotek som tidligere var brukt var ikke lenger tilgjengelig i *repositoriet* og måtte lastes ned og legges til manuelt. |
| Eclipse | Enkelt å utrulle (eng. *deploy*) og teste applikasjonen på en Tomcat server. Har mange plugins, for eksempel Maven og svn. | Kræsjet noen få ganger. |
| Tomcat | Kan publisere en del endringer uten å måtte restarte applikasjonen. | Bruker noen minutter på å starte opp med DPG. |
| Cheat Sheets | Mye nyttig informasjon som er enkelt og konkret forklart. Lett å følge ved implementering. | Ikke alle juksearkene er ferdige, for eksempel det for tilgangskontroll, som har statusen *pågående arbeid*. |
| ASVS | Gir en god pekepin på sikkerhetsaspekter man burde tenke på. | Burde inn allerde i kravspesifikasjon. Ikke nødvendigvis egnet å lappe på et system basert på ASVS, i hvertfall i henhold til å ønske å oppnå nivå 4, grunnet krav som ikke direkte fører til at programvaren blir sikrere, men som er ansett som god design. |
| Spring Security | Enkelt å utvide med flere autentiseringsmekanikker. | Noe omfattende konfigureringsfiler. Det speilet seg igjen i at noen ting var duplisert, og flere ting var ikke i bruk lenger. Skulle gjerne hatt mer funksjonalitet, som kryptering og inndatavalidering. |

*Tabell 5.2:* Under sikkerhetsanalysen

| Navn | Styrker | Svakheter |
|---|---|---|
| Fortify | Oppdaget en god del sikkerhetshull. Viser tydelig hvilken sti inndata tar ved mulige svakheter. | Kommersielt verktøy. Det tok litt tid å få en ut *utdannings-lisens*. Kan ikke kategorisere etter ASVS krav. |
| Findbugs | Meget enkelt å sette opp og kjøre på DPG. | Har bare noen få mønstere for å finne sikkerhetssvakheter. |
| ZAP | Enkelt å sette opp og bruke for å komme i gang med penetreringstesting. | Listene over de predefinerte strengene til *Fuzzer*-verktøyet burde vært tilgjengelig på en enkel måte. |
| Eclipse | Forenkler den manuelle kodegjennomgangen. Lettere å følge stier inndata tar i applikasjonen. Plugin for Fortify og Findbugs gjør det enklere å analysere resultater. | Ingen svakheter erfart. |
| Code Review Guide | Følger den samme organiseringen som ASVS. Gode generelle retningslinjer med spesifikke instruksjoner for enkelte programmeringsspråk, som Java, .NET, PHP, og Classic ASP. Forklarer mulige angrep, og hvordan de bør mitigeres. | Ingen svakheter erfart. |
| Testing Guide | Inneholdt gode forklaringer med oversikt over hva som må testes. Organiseringen er veldig lik ASVS og Code Review Guide | Gir hint om hvordan man kan teste, men de er veldig løse og krever mye erfaring for å få god gjennomføring - litt motsatt av Code Review Guide, som var mer rett på sak mhp. hva man skulle lete etter. |
| ASVS | Systematisk oversikt med veldig spesifikke krav til sikkerheten. | Noen krav er ikke nødvendig for å ha et sikkert system, men er mer gode anbefalinger. Å oppnå noe mer enn nivå 2 virker unødvendig for et system som DPG. |

## 5.3 Videre arbeid

Under arbeidet gjennomført i denne oppgaven har det dukket opp en del idéer til videre arbeid som enten kan forbedre DPG eller gjøre det enklere å gjennomføre en ASVS analyse. Her blir de mest relevante av disse presentert.

### 5.3.1 Forbedre loggingen

All loggingen i dag blir loggført direkte i `Catalina.out` filen fra Tomcat. Å gå gjennom loggen er en meget tidkrevende prosess og det beste verktøyet man har er *grep* [52]. På nivå 2 krever ASVS at et verktøy for å analysere loggene er tilgjengelig. For å enklere kunne foreta feilsøking anbefales det at loggingen gjennomføres på en mer systematisk måte.

### 5.3.2 Ny versjon av Webucator

Koblingen som er implementert til Mi side har endret hvordan DPG gir tillatelse til *readere* og *publishere*. For å bruke Webucator til å tilby disse rollene i DPG må denne endringen også implementeres i Webucator.

### 5.3.3 Videre forbedringer av sikkerhetssvakheter i DPG

Mange sikkerhetshull er fortsatt til stede i DPG, og ASVS feiler på mange punkter. Noen direkte forslag til løsninger av svakheter er foreslått, men løsninger må implementeres og det burde tilstrebes å oppnå ASVS nivå 1.

### 5.3.4 TLS i DPG

I dag er det ingenting som forhindrer mann-i-midten angrep. En løsning som vil forhindre dette burde være på plass.

### 5.3.5 Validering av mønsteret

Dersom det er en feil i utformingen av et presentasjonsmønster, vil en systemfeil oppstå i DPG. Mønsterene er innviklede og store, og ikke lett å ha full oversikt over. Det er en funksjonalitet i DPG som lar administratorer endre mønsteret direkte fra

*presentation manager*(pm), men den har ingen valideringsregler for mønsteret. Det gjør det til en stor risiko å benytte denne funksjonen om man ikke har full oversikt over hvordan mønsteret skal lages.

Dersom man i *presentation content editor*(pce) deaktiverer en side som er satt som standard i et *utsnitt*, vil det føre til at et unntak kastes når *utsnitt*-et skal vises.

Mangelen på validering av mønsteret i *pm* og *pce* skaper muligheter for feil i systemet, og bør derfor utbedres. Det å ha en validator kan også senke terskelen for å opprette nye mønstere, og dermed kan det bli enklere å benytte DPG til nye formål.

### 5.3.6 Ny funksjonalitet for bruk med Mi side

Fra brukerevalueringen til den sømløse overgangen kom det frem at det hadde vært en fordel om endringer fra sist innlogging kom tydelig frem. Det krever endringer i mønsteret, og også ny funksjonalitet i DPG.

Det hadde også vært en god ide om DPG kunne hentet pensumet direkte fra Mi side. Pensumet er offentlig tilgjengelig for alle kurs, og å legge inn en slik funksjonalitet krever muligens kun endringer i presentasjonsmønsteret.

### 5.3.7 Kontinuerlig integrasjon med sikkerhetstesting

I avsnitt 4.5 ble det anbefalt å integrere automatisk sikkerhetstesting i den kontinuerlige integrasjonen av DPG. Det innebærer at Jenkins kjører verktøy som Fortify og ZAP automatisk. Det ble også anbefalt å bruke avbildingsverktøyet fra avsnitt 4.2.5 slik at resultater direkte kan avbildes til ASVS krav og vises i Jenkins.

For å dekke flere av kravene til ASVS nivå 1 må også flere analyseverktøy identifiseres og taes i bruk.

### 5.3.8 Fortify - lage en kategori som kan sortere på ASVS krav

I avsnitt 4.2.5 ble det nevnet at Fortify kan gruppere sårbarheter på veldig mange forskjellige måter. Det hadde vært praktisk å direkte kunne sortere etter ASVS krav og nivå for å foreta en ASVS analyse. Da Fortfiy er et kommersielt verktøy kan det hende dette ikke er mulig å gjennomføre for en bruker.

## 5.4 Oppsummering

### 5.4.1 Personlige erfaringer

I løpet av arbeidet med denne oppgaven har kandidaten opparbeidet en grundigere forståelse for sikkerhetsproblemer man står ovenfor som systemutvikler.

Det å forsøke å gjennomføre en ASVS nivå 4 verifisering første gang man prøver ut ASVS er ikke å anbefale. For å oppnå en verifisering på nivå 4 burde man ha benyttet ASVS i hele utviklingsprosessen, helt fra kravspesifikasjon, gjennom systemutviklingen og til testingen.

Kandidaten har jobbet med kjente sikkerhetsrammeverk som Spring Security og OWASP CSRFGuard. Det er implementert sikkerhetsrelaterte funksjoner, som autentisering og tilgangskontroll, og sikkerhetssvakheter er utbedret i DPG.

Kandidaten har også erfart konsekvensene av det å ikke utføre tilstrekkelig med integrasjonstesting, da en systemfeil ikke ble oppdaget før den førte til at flere brukere ikke fikk logget inn.

Det å ha kjennskap til en del sikkerhetsproblemer og å kunne benytte verktøy for å oppdage og utbedre disse er nyttig for en utvikler i dag. Kandidaten føler dermed at han har vokst i utviklerrollen ved å ha gjennomført denne oppgaven.

### 5.4.2 Nyttig videre

Analysen av DPG har gitt et godt utgangspunkt for å øke sikkerheten i DPG. Noen problemer er utbedret, men en del gjenstår om man vil oppnå et ASVS nivå. Avbildingsverktøyet fra analyseresultat til ASVS-krav vil forhåpentligvis være en nyttig brikke i både det å foreta en ASVS analyse og under utviklingsprosessen for å automatisere sikkerhetstestingen.

Den sømløse overgangen som ble utviklet i forbindelse med denne oppgaven er allerede i bruk. Innloggingsproblemene ble raskt løst og har siden fungert uten feil. Kandidaten kan også konkludere med at det har vært vellyekket med hensyn til tilbakemeldingene fra studentene. Dette åpner for mulighet til å utvide bruken av DPG til flere fag ved UiB.

# Bibliografi


[1] Scott Ambler. Examining the agile cost of change curve. http://www.agilemodeling.com/essays/costOfChange.htm.

[2] Lars Risa Anders Sandven. Dynamic presentation generator, 2010. INF226 report.

[3] G. McGraw B. Arkin, S. Stender. Software penetration testing. *Security Privacy, IEEE*, 2005.

[4] Madiha Mir Borghild Vikøyr. The dpg system, 2010. INF226 report.

[5] Jacob West Brian Chess. *Secure Programming With Static Analysis*. Addison Wesley, 2007.

[6] Dino Dai Zobi Elfriede Dustin Chris Wysopal, Lucas Nelson. *The Art of Software Security Testing: Identifying Software Security Flaws*. Addison Wesley, 2006.

[7] OpenACS community. Openacs. http://openacs.org/.

[8] Paul M. Duvall, Steve Matyas, and Andrew Glover. *Continuous Integration:Improving Software Quality and Reducing Risk*. Addison Wesley, 2007.

[9] Apache Software Foundation. Apache tomcat. http://tomcat.apache.org/.

[10] The Apache Software Foundation. Apache commons validator. http://commons.apache.org/proper/commons-validator/.

[11] The Apache Software Foundation. Apache shiro. http://shiro.apache.org/.

[12] Namit Gupta and Abakash Saikia. Web application firewall. Technical report, Department of Computer Science and Engineering, Indian Institute Of Technology, Kanpur, 2007.

[13] K.J Hole, V Moen, and T Tjostheim. Case study: online banking security. *Security and Privacy, IEEE*, 2006.

[14] Walter Hürsch and Cristina Lope. Separation of concerns. Technical report, College of Computer Science, Northeastern University, 1995.

[15] Universitetet i Bergen. It-avdelingen. http://www.uib.no/it.

[16] Universitetet i Bergen. Kark. http://www.hist.uib.no/.

[17] Universitetet i Bergen. Mi side. http://miside.uib.no/.

[18] Jenkins. Jenkins. http://jenkins-ci.org/.

[19] Joomla! Joomla! - the dynamic portal engine and content management system. http://www.joomla.org, 2013.

[20] Aleksander Vatle Waage Kelly Alexander Teigland Whiteley. Statisk analyse av dpg 2.1, 2010. INF226 report.

[21] RSA Laboratories. Rsa laboratories' frequently asked questions about today's cryptography, version 4.1. Technical report, RSA Security Inc., 2000.

[22] .LRN. .lrn - learn, research, network. http://dotlrn.org.

[23] Kristian Skønberg Løvik. Webucator 3.0 - Brukerhåndtering og aksesskontroll for DPG 2.0. Master's thesis, Universitet i Bergen, 2008.

[24] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[25] Gary McGraw. *Software Security Building Security In*. Addison-Wesley, 2006.

[26] Russ McRee. Owasp zap – zed attack proxy. *ISSA Journal, November 2011*, 2011.

[27] Vebjørn Moen, Andre N. Klingsheim, Kent Inge Fagerland Simonsen, and Kjell Jørgen Hole. Vulnerabilities in e-governments. http://www.nowires.org/Papers-PDF/ICGeS_egov.pdf.

[28] Peter Mularien. *Spring Security 3*. Packt, 2010.

[29] Justis og beredskapsdepartementet. Lov om behandling av personopplysninger. http://www.lovdata.no/all/nl-20000414-031.html.

[30] Core J2EE Patterns. Presentation tier design considerations. http://www.corej2eepatterns.com/Design/PresoDesign.htm.

[31] The Open Web Application Security Project. Attack surface analysis cheat sheet. https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet.

[32] The Open Web Application Security Project. Category:owasp antisamy project. https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project.

[33] The Open Web Application Security Project. Category:owasp application security verification standard project. https://www.owasp.org/index.php/ASVS.

[34] The Open Web Application Security Project. Category:owasp enterprise security api. https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.

[35] The Open Web Application Security Project. Cheat sheets. https://www.owasp.org/index.php/Cheat_Sheets.

[36] The Open Web Application Security Project. Cross-site request forgery (csrf) prevention cheat sheet. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.

[37] The Open Web Application Security Project. Owasp code review project. https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project.

[38] The Open Web Application Security Project. Owasp csrfguard project. https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project.

[39] The Open Web Application Security Project. Owasp testing project. https://www.owasp.org/index.php/Category:OWASP_Testing_Project.

[40] The Open Web Application Security Project. Owasp the open web application security project. https://www.owasp.org/.

[41] The Open Web Application Security Project. Owasp zed attack proxy project. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[42] The Open Web Application Security Project. Top 10 2010-a2-cross-site scripting (xss). https://www.owasp.org/index.php/Top_10_2010-A2-Cross-Site_Scripting_(XSS).

[43] The Open Web Application Security Project. Top 10 2010-main. http://www.owasp.org/index.php/Top_10_2010-Main.

[44] The Open Web Application Security Project. Xss (cross site scripting) prevention cheat sheet. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

[45] Bill Pugh and Andrey Loskutov. Findbugs™ - find bugs in java programs. http://findbugs.sourceforge.net/.

[46] Ravi S. Sandhu and Pierangela Samarati. Access control: Principles and practice. *lEEE Communication Magazine*, 1994.

[47] SpringSource. Spring security. http://static.springsource.org/spring-security/site/.

[48] Universitetet i Bergen Studieadministrativ avdeling. Studieadministrativ avdeling. http://www.uib.no/ua/.

[49] Tigris.org. Subversion. http://subversion.tigris.org/.

[50] Universitetet i Bergen Utdanningsavdelingen. Felles studentsystem (fs). https://wikihost.uib.no/uawiki/index.php/Felles_studentsystem_(FS).

[51] Ole Henning Vårdal. Sikkerheitsaspekt i Dynamic Presentation Generator: Vurdering, implementasjon og strategi. Master's thesis, Universitetet i Bergen, 2010.

[52] The free encyclopedia Wikipedia. grep. https://en.wikipedia.org/wiki/Grep.

[53] Wordpress. Wordpress > blog tool and publishing platform. http://wordpress.org/, 2013.

[54] XSSed. Xssed | cross site scripting (xss) attacks information and archive. http://www.xssed.com/.