Sikkerheitsaspekt i Dynamic Presentation Generator: Vurdering, implementasjon og strategi

Oppgåve av

Ole Henning Vårdal

Institutt for Informatikk

Universitetet i Bergen

Lang Masteroppgåve

September 2010

Forord

Denne oppgåva går gjennom, kartlegg og foreslår løysingar på sikkerheitsproblem relaterte til JAFU (Java Fjernundervisning) ved Institutt for Informatikk, Universitetet i Bergen.

Det vil først bli gitt ein introduskjon til oppgåva der bakgrunn, problemstilling og mål blir gjennomgått. Systema som skal analyserast blir introdusert med fokus på roller og ressursar. Vidare vil det vere ei analyse av problema der vurderingar vil bli gjort på truslar, sårbarheiter og risiko. Som utgangspunkt for desse vurderingane er gjennomføringa av kurset INF226 der ei analyse av JAFU vart gjort hausten 2008.

Vidare vil moglege løysingar bli gjennomgått i form av eksisterande rammeverk og verktøy, samt inkluderinga av Open Web Application Security Project sin Enterprise Security API og AnitSamy samt rammeverket Spring og Spring Security. I dette tilfellet vil gjennomgangen av løysingar på dei problema som vart avdekka gjennomgått, og ein vil vurdere korrekt bruk av verktøyet, samt kva problemer som kan oppstå ved feil bruk. Kartlegging av framtidige utviklingsaktivitetar med fokus på sikkerheit i høve til Open Security Assurance Maturity Model vil også bli vurderte som ein integrasjon til eksisterande utviklingsaktivitetar.

Eg vil gjerne nytt anledninga til å takke veilederane mine; Khalid A. Mughal og Torill Hamre for gode råd og innspel i løpet av masterstudiet mitt. Haakon Nilsen har gitt fleire gode tilbakemeldingar og vore ein verdiful kontaktperson og som har sørga for at infrastruktur og arbeidsmiljø har vore i orden.

Eg vil også takke mine medstudentar gjennom åra som vennar og studiekameratar. Tobias Rusås Olsen, Peder Skeidsvoll, Jostein Bjørge, Morten Høiland og Øystein Lund Rolland som også har jobba med JAFU-prosjektet. Til slutt vil eg gjerne takke familien min og min kjære Annie Eliassen for støtte, oppmuntringar og tolmodigheit.

Bergen, September 2010

Ole Henning Vårdal

Innhald

Figur- og illustrasjonsliste

Liste av tabellar

Liste av eksempel

Liste over forkortingar

JAFU	Java FjernUndervisning
ESAPI	Enterprise Security Application Programming Interface
SEVU	Senter for videre- og etterutdanning
OWASP	Open Web Application Security Project
OpenSAMM	Open Software Assurance Maturity Model
PCE	Presentation Content Editor
PV	Presentation Viewer
PM	Presentation Manager
DPG	Dynamic Presentation Generator
CMS	Content Management System
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
XSS	Cross-site Scripting
MD5	Message Digest 5
SQL	Structured Query Language
SSL	Secure Sockets Layer
AJAX	Asynchronous JavaScript and XML
URL	Uniform Resource Locator
LDAP	Lightweight Directory Access Protocol
MVC	Model View Controller
CSS	Cascading Style Sheets
CLASP	Comprehensive Lightweight Application Security Process
SDL	Security Development Lifecycle

1

1 Introduksjon

Denne oppgåva er skriven i tilknytning til JAFU-prosjektet (Java FjernUndervisning) ved Institutt for Informatikk, Universitetet i Bergen. Dette prosjektet vart starta i 1998, og var då kjent under eit anna namn, Bergen Webucator. Formålet med prosjektet var å tilby fjernundervisning i programmering gjennom kurs ved Universitetet i Bergen. Senter for videre- og etterutdanning (SEVU) ved Universitetet i Bergen var også involvert i prosjektet.

Det har tidlegare blitt fokusert på sikkerheit i JAFU-prosjektet gjennom blant anna kurs som tek for seg sikkerheit i vevapplikasjonar, samt utvikling av nye system. Analyse av gamle system blei gjennomført av masterstudentar som produserte dei nye applikasjonane, og inntrykk dei fekk frå denne analysa har vore viktig for dei vala som blei gjort under vidare utvikling.

Sikkerheit er ein prosess, ikkje eit produkt [55]. Og etter kvart som gamle studentar går ut av universitetet, og nye studentar tek over, har dei sikkerheitsrelaterte aktivitetane stagnert, og behovet for ei omfattande analyse av dei nye systema er nødvendig.

TODO: Skrive meir og rydde opp i introduksjonen etter kvart som oppgåva nærmar seg slutten.

1.1 Bakgrunn for JAFU

I hovudsak har INF100-F - Grunnkurs i programmering og INF101-F – Vidaregåande Programmering vore dei tilbodne kursa ved JAFU. Desse kursa følgjer samme pensum

som på campus, men undervisning skjer altså gjennom internett. Dette gjer at studentane i fjernundervisninga kan ta eksamen i lag med dei ordinære studentane som tek kurset på campus.

Ei rekke nettbaserte verktøy for publisering og handtering av kursmaterielle og anna informasjon har opp gjennom åra vorte tatt i bruk. Innhald- og brukarhandteringssystem av høg kvalitet er nødvendige for å konkurrere med andre aktørar som tilbyr liknande tenester for fjernundervisning, og statiske nettsider for formidling av store mengder informasjon er ikkje tilstrekkeleg.

I løpet av dei tolv åra JAFU har eksistert har ei rekke forskjellige system for å i møtekome dette behovet blitt utvikla. I dag består systemet i hovudsak av to forskjellige applikasjonar; *Webucator 3.0* og *DPG 2.0 (Dynamic Presentation Generator 2)*, som vart utvikla av ei rekke masterstudentar og var utgangspunktet i oppgåvene deira i 2008 [1, 2, 3, 4].

DPG 2.0 er eit innhaldshandteirngssystem basert på konseptet bak presentasjonsmønster [5]. Webucator 3.0 er eit brukarhandteringssystem produsert for å handtere brukarar i DPG 2.0 og knytte dei opp mot kurs ved fjernundervisninga.

Sikkerheit har over åra vist seg å vere ei utfordring for desse systema, og til tross for tiltak for å forbetre sikkerheitssituasjonen i DPG 2 og Webucator 3 i forhold til dei foregåande systema, har det i ettertid vist seg å vere for lite, og fokusen på moglege sikkerheitsutfordringar har ikkje vore omfattande nok, noko som kan vitne om manglande forståing for dei potensielle truslane systema står ovanfor.

Ei rekke sikkerheitsaspekt som tilgangskontroll og autentisering har vorte tatt hand om til ei viss grad, medan sikkerheitsproblem relatert til kodefeil og manglande validering og sanitering av inndata har vorte ignorerte. Forslag til vidare arbeid har heller ikkje inkludert sikkerheit som eit mogleg problemområde [1, 2, 3, 4], noko som tilseier at problema i stor grad har vorte ignorerte, eller at kunnskapen rundt desse utfordringane ikkje har vore god nok blant utviklarane.

1.2 Problemstilling

Utvikling av DPG 2 og Webucator 3 til JAFU medførte ein del renovering og nyutvikling. Ny design, ny arkitektur og ny kode medfører nye sikkerheitsutfordringar, noko som kan vere eit omfattande arbeid, spesielt dersom ein ikkje tek tak i desse problema frå byrjinga [19]. Ein del av dei utfordringane og problema som har dukka opp blir presenterte i dette delkapittelet.

1.2.1 Omfattande sikkerheitsproblem i JAFU-systema

Hausten 2008 gjennomførte studentar ved Universitetet i Bergen ei sikkerheitsanalyse av vevapplikasjonar knytta til JAFU-prosjektet [12, 13, 14, 15, 16, 17]. Ved hjelp av statisk og manuell kodeanalyse samt omfattande sikkerheitstesting vart det kartlagt ei stor mengde med feil i både DPG2 og Webucator. Dette er feil relatert til blant anna XSS (Cross Site Scripting), loggforfalsking, svak autentisering og liknande.

Denne analysa har blitt gjennomført grundigare og meir omfattande i denne oppgåva. Problema kartleggast og kategoriserast for implementasjon av moglege løysingar og integrering av ulike teknologiar.

1.2.2 Ingen spesielle løysingar på vanlege sikkerheitsproblem

Det ligg i dag få spesielle løysingar til rette for handtering av sikkerheitsproblema som tidlegare vart avdekka. Ein bør ta ibruk spesialiserte verktøy og rammeverk for å handtere dei sikkerheitsproblema som blir kartlagt i denne oppgåva. Verktøy som Spring Security er allereie i bruk, men ikkje utnytta fullstendig for å i møtekome krav om god nok autentisering og tilgangskontroll, samt sikker lagring. Andre verktøy som ESAPI (Enterprise Security API) og AntiSamy vil også bli inkludert for å handtere ein del av dei omfattande sikkerheitsproblema relatert til feil i koden som eksisterer.

Ein bør unngå å skrive desse verktøya sjølv. Dette er med utgangspunkt i kor komplisert og avansert det er å produsere slike verktøy av høg nok kvalitet. Det er også vanleg å gjere ei rekke feil i utvikling av slike verktøy, og sjølv store kommersielle firma gjer ofte feil [19]. Videotenesta YouTube blei i 2010 ramma av cross site scripting angrep på grunn av svakheiter i validering av inndata [56]. Det vil vere problematisk om ein trur at applikasjonen er sikker mot ein spesifikk type angrep, medan det i realiteten er mogleg for ein angripar å omgå eventuelle beskyttingsmekanismar.

1.2.3 Manglande sikkerheitsstrategi

Tidlegare sikkerheitsaktivitetar relaterte til JAFU-prosjektet har vist seg å ikkje vere nok. Etter kvart som sikkerheitsproblem vart avdekka i tidlegare system var eit av måla for utvikling av dei nye systema å handtere desse sikkerheitsproblema. Ingen anna sikkerheitsstrategi vart lagt til rette under utviklinga for å handtere eventuelle nye sikkerheitsproblem som kunne oppstå, og situasjonen for sikkerheitsarbeid i framtidig utvikling ved JAFU var ikkje avklart.

1.3 Mål for opppgåva

Problem utgreid i kapittel 1.2 legg grunnlaget for måla med denne oppgåva. Måla med oppgåva er med andre ord å løyse dei omtalte problemområda.

- Detaljert og omfattande kartlegging av noverande sikkerheitsproblem for å legge til rette for rask og enkel implementasjon og vurdering av moglege løysingar.
    - Vurdering av teknologien som blir tatt i bruk for analyse og vurdering av sikkerheitsproblema som eksisterer i systemet i dag.
    - Etablere riktig framgangsmåte for vurdering, testing og analyse av eksisterande og framtidige sikkerheitsproblem systemet står eller kan stå overfor.
- Implementering av verktøy for å handtere slike problem. Verktøya bør vere godt dokumenterte, og bør kunne handtere ei stor mengde forskjellige problem, spesielt med fokus på vidare bruk av slike verktøy i framtidig sikker utvikling.
    - Vurdering av kva verktøy som egnar seg best for å i møtekome dei truslane og sikkerheitsproblema som JAFU-systema, samt andre typer liknande system står ovanfor i dag.
- Ein strategi bør også leggast til rette for vidare utvikling av JAFU. Det er viktig å unngå nye (og gamle) feil etter kvart som systema veks, og ny funksjonalitet vert lagt til.
    - Etablere utgangspunktet for vidare forbedring. Altså påpeike korleis sikkerheitssituasjonen i JAFU var under utvikling av dei gamle systema, og rangere sikkerheitsrelaterte aktivitetar med utgangspunkt i OpenSAMM (Open Software Assurance Maturity Model)
    - Dette inneber ei vurdering av moglege modellar for sikker utvikling, med spesiell fokus på OpenSAMM, samt ein strategi som enkelt kan takast i bruk for å forbetre sikkerheitsprosedyrer under vidare utvikling.

1.4 Organisering av dokumentet

Kapittel 2 - Error: Reference source not found

I dette kapittelet vert det gitt ein grunnleggande introduksjon til systema som skal vurderast. Det blir gjennomgått formål, arkitektur, roller og teknologiar som inngår i desse systema.

Kapittel 3 - 3 Sikkerheitsanalyse av JAFU

I dette kapittelet blir det gjennomgått kva verktøy og metodar som har blitt nytta for analyse av DPG 2.0 og Webucator 3.0. Resultatesettet vert presentert, og det vil bli påpeikt kva delar av systema som er utsatte, kva problem dette medfører og utbreiinga av desse.

Kapittel 4 - 4 Sikkerheitsproblem – vurdering

Her blir problema presentert i større detaljegrad enn i kapittel 3, og det vil bli demonstrert angrep, samt vurdere alvorlegheita av desse problema, satt i samanheng med korleis dei framtrer i dei aktuelle delane av DPG 2 og Webucator 3.

Kapittel 5 - 5 Implementasjon av løysinger

Implementasjon av løysingane på nokre av problema som blir presentert i kapittel . Fokus er på verktøya Enterprise Security API [46] og AntiSamy[49] frå OWASP Foundation som potensielle løysingar på problema i JAFU-systema. I tillegg blir det vurdert korleis verktøy og rammeverk som allereie er i bruk kan nyttast til å handtere desse problema.

Kapittel 6 - 6 Sikkerheit og systemutvikling

Spørsmålet er korleis dei sikkerheitsproblema som blir presenterte i denne oppgåva oppstod i utgangspuntket, og korleis ein kunne ha hindra at dei oppstod. Kva strategiar og modellar under utviklinga kan takast i bruk for å luke ut alvorlege sikkerheitsproblem? Fokus vil her vere på OpenSAMM[43], og dei aktivitetane denne modellen består av.

Kapittel 7 - 7 Konklusjon og evaluering

Dette kapittelet vil innehalde ei kort oppsummering og evaluering av arbeidet som er gjennomført. Resultat og funn frå analysa, implementasjon og vurdering av sikkerheitsstrategi vil bli evaluert mot måla vart beskrive i introduksjonskapittelet. Det blir også vurdert korleis denne oppgåva har bidratt til JAFU.

2

2 Gjennomgang av DPG 2.0 og Webucator 3.0

For å forstå kvifor ein del av sikkerheitsproblema som seinare blir presentert er så alvorlege som dei er, må det sjåast i samanheng med systemet som er sårbart. Det er derfor viktig å gi ein grunnleggande gjennomgang av korleis systema er oppbygde, og til kva formål dei er konstruerte. Uten ei oversikt over korleis systema fungerer er det vanskeleg å gi ei vurdering av kor alvorlege sikkerheitsproblema er.

2.1 Beskrivelse av presentasjonsmønster

I tilfeller der ein gjerne ynskjer å gjenbruke ei nettside, men med nytt innhold har det tradisjonellt sett vore vanskeleg å gjennomføre utan å på ny skrive store delar av koden frå grunnen av. Årsaka til dette er ofte at innholdet på nettsida ein ynskjer å gjenbruke er kopla mot formateringa av nettsida.

Presentasjonsmønster er eit konsept som var lagt fram av Khalid A. Mughal, Yngve Espelid og Torill Hamre i 2003 [5]. Eit presentasjonsmønster er eit regelsett som tek for seg layout, navigasjon og datastrukturar i presentasjonar på web. Presentasjonsmønster blir bygde med utgangspunkt i ein *presentasjonsmønsterspesifikasjon*. Ein presentasjonsmønsterspesifikasjon består av fire delar.

- Side (eng. *page*)
- Utsnitt (eng. *view*)
- Entitet (eng. *entity*)
- Entitetsinstans (eng. *entity-instance*)

Ein entitet er ei liste med felt, som kan vere blant anna strengar, XHTML, datoar eller plugins. Ein entitetsinstans refererer til ein entitet. Det kan lagast fleire instansar av samme entitet. I ein entitetsinstans blir innhaldet til ein entitet definert. Eit utsnitt refererer til ein entitetsinstans, og definerer korleis informasjonen skal framvisast for brukaren ved å knytte entitetsinstansen opp mot ein transformasjon. Dette inneber at ein kan framvise ein entitetsinstans på fleire forskjellige måtar, for eksempel ved å sortere felt på andre måtar og liknande. Ei side består av ei samling med utsnitt. På ei nettside kan ei side vere til dømes seksjonen *hjelp* i ein hovudmeny som kan innebere ei rekke utsnitt som *kundestøtte*, *FAQ*, *besøksadresse*, *kontaktinfo* og liknande.

Instansar av presentasjonsmønster (presentasjonar) gjer det mogleg å bruke samme mønster for fleire presentasjonar, som så er uavhengige av kvarandre. Desse presentasjonane kan for eksempel vere fjernundervisningskurs ved Universitetet i Bergen, men er ikkje avgrensa til dette. I tillegg kan det innebere bloggar, nyheitssider og liknande.

2.2 DPG 2

Dynamic Presentation Generator 2.0 er eit web-basert innhaldshandteringssystem (eng. *Content Management System - CMS)* for JAFU. Systemet er ein ein implementasjon av idéen bak presentasjonsmønster, og blir i dag brukt til fjernundervisning ved Universitetet i Bergen.

Eit web-basert innhaldshandteringssystem er eit system som skal forenkle redigering, handtering og publisering av data og informasjon til web. Dette er i form av tekst, HTML, bilde og filmer. Dette gjer det mogleg for personar som ikkje har kompetanse til å drive med webutvikling, å publisere, redigere og handtere internettinnhald. Eksempel på andre innhaldshandteringssystem for web er Wordpress [38], Joomla [39] og Wikipedia [40].

2.2.1 Brukarroller i DPG 2.0

Tilgangen til DPG 2.0 er rollebasert. Brukarane med desse rollene blir handterte og administrerte i brukarhandteringssystemet Webucator 3. I DPG 2 er det per i dag tre forskjellige typar brukarar (roller):

- *Reader* – Dette er den rolla dei fleste brukarane i systemet vil ha. Desse brukarane har lesetilgang til ein eller fleire presentasjonar. I fjernundervisninga er brukarar med denne rolla som regel studentar.
- *Publisher* – Denne rolla har skrivetilgang til ein eller fleire presentasjonar. For brukarar med denne rolla er det mogleg å endre på innhald i presentasjonane dei er *publisher* i, men det er ikkje mogleg å endre konfigurasjonen til presentasjonane. I fjernundervisninga er dette som regel kursansvarlege eller studieassistentar.

- *Administrator* – Denne rolla har full tilgang til systemet. Alle presentasjonar er tilgjengelege, både for lesing og endring. Det er også mogleg for administratorar å endre sjølve konfigurasjonen til alle presentasjonane i systemet.

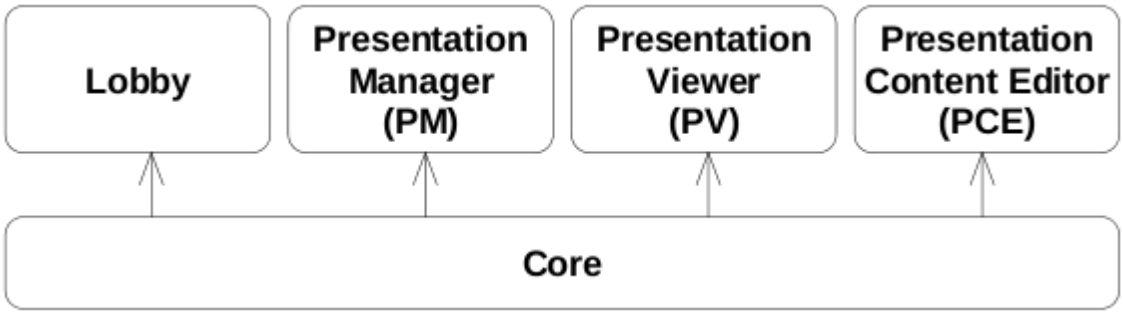
Brukarar med rolla *reader* og *publisher* har som sagt ikkje nødvendigvis tilgang til alle presentasjonane. Tilgangen er handtert gjennom brukarhandteringssystemet Webucator 3.0. Her blir brukarane knytt til instansar som presentasjonane er bygd på, og dei skal verken kunne lese eller endre andre presentasjonar. Rollene *reader* og *publisher* er knytta til ein enkelt presentasjon. Dette inneber at ein og samme brukar kan vere *publisher* i ein gitt presentasjon A, medan han kun er *reader* i ein gitt presentasjon B.

Ein presentasjon kan vere for eksempel ei nettside for eit kurs ved Universitetet i Bergen, med nyheitsside, framdriftsplan, ressursar, eksamensplan og liknande informasjon. Det treng ikkje nødvendigvis vere kurs ved UiB, men også nettaviser, bloggar og liknande.

Administratorrolla er knytt mot heile systemet, og ikkje kun til ein eller fleire presentasjonar. Administratorar kan konfigurere, opprette, endre og slette presentasjonar i systemet.

2.2.2 Delsystem i DPG 2.0

DPG 2.0 består av fem ulike delsystem: *Lobby*, *Presentation Viewer (PV)*, *Presentation Content Editor (PCE)*, *Presentation Manager (PM)*, og *Core*. Oppdelinga av dei forskjellige delsystema kan ein sjå i figur 1.



Figur 1: DPG 2.0 – Delsystem [1]

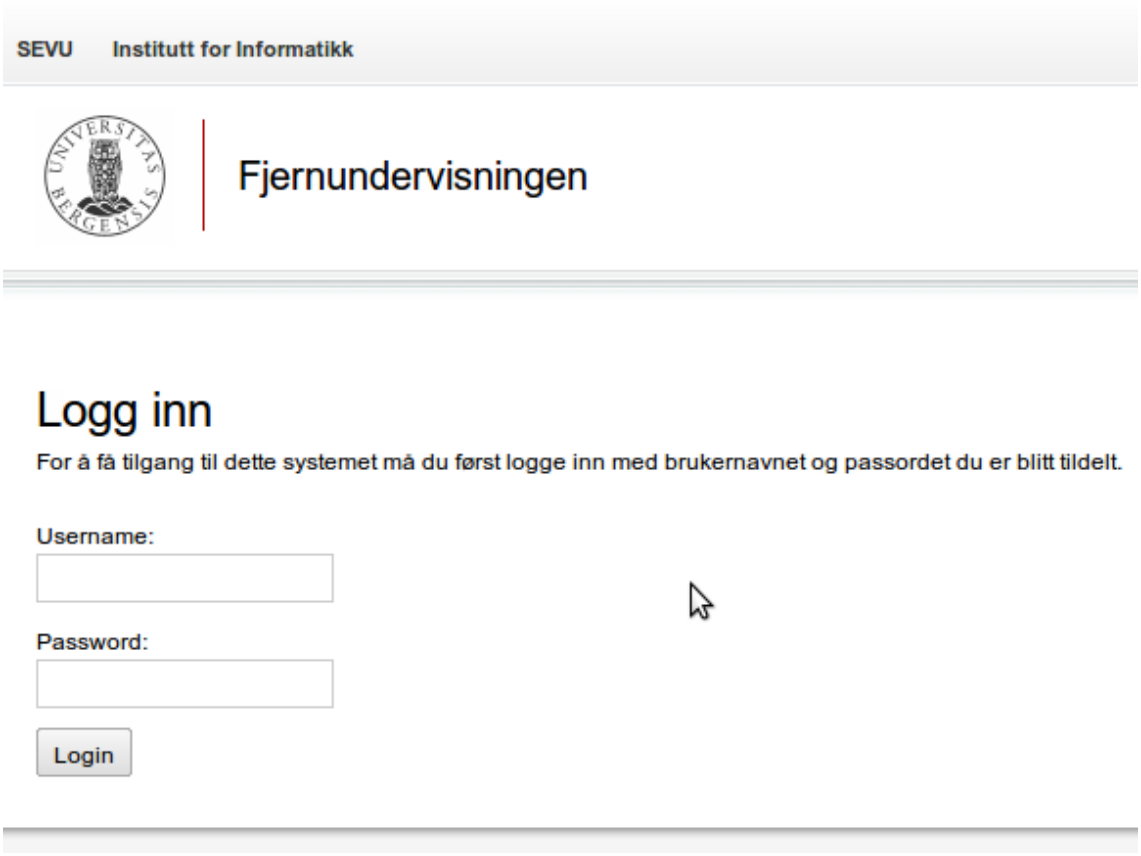
Core

Dette delsystemet inneheld all den funksjonaliteten som er felles for dei andre delsystema. Ein stor del av dette inneber persistenslaget, som vart utforma av Karianne Berg som ein del av hennar masteroppgåve våren 2008 [2].

Resten av systemet er utforma av Bjørn Christian Sebak, som ein del av hans masteroppgåve [3]. I tillegg eksisterar det ein del kode for handtering av multimedia som bilder, videosnuttar, lydfiler og så vidare. Dette er utforma av Bjørn Ove Ingaldsen som ein del av hans masteroppgåve [4].

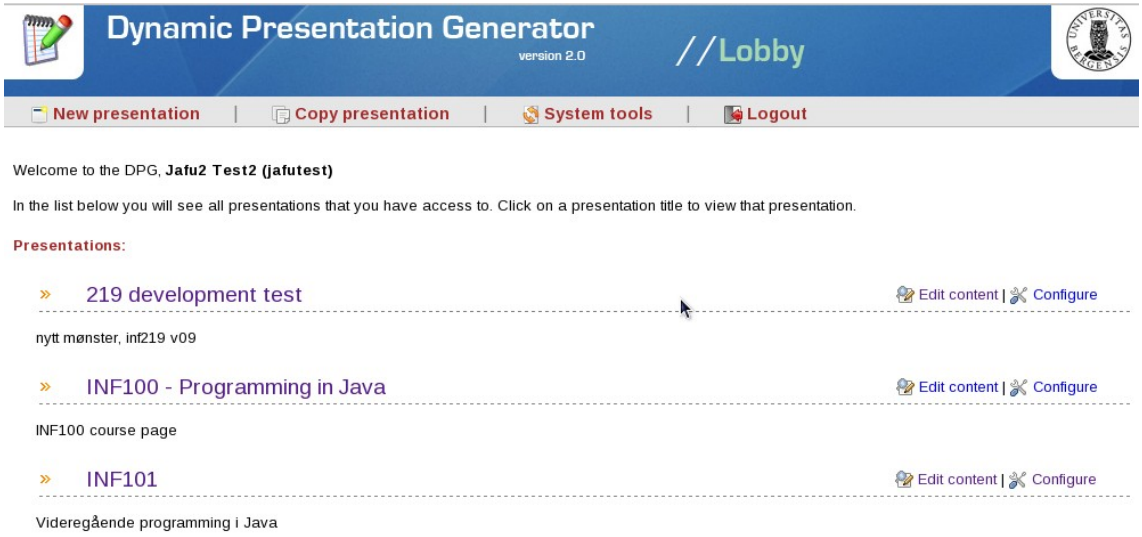
Lobby

Det minste delsystemet er inngangsportalen til DPG 2.0 (Figur 2), kalt *Lobby*. Ved bruk av Spring Security og Spring Remoting mot Webucator 3.0 blir brukarar autentiserte ved hjelp av brukarnamn og passord. og sendt vidare til oversikta over presentasjonar dei har tilgang til basert på rettigheitene den aktuelle brukaren har.



Figur 2: DPG 2.0 – Lobby

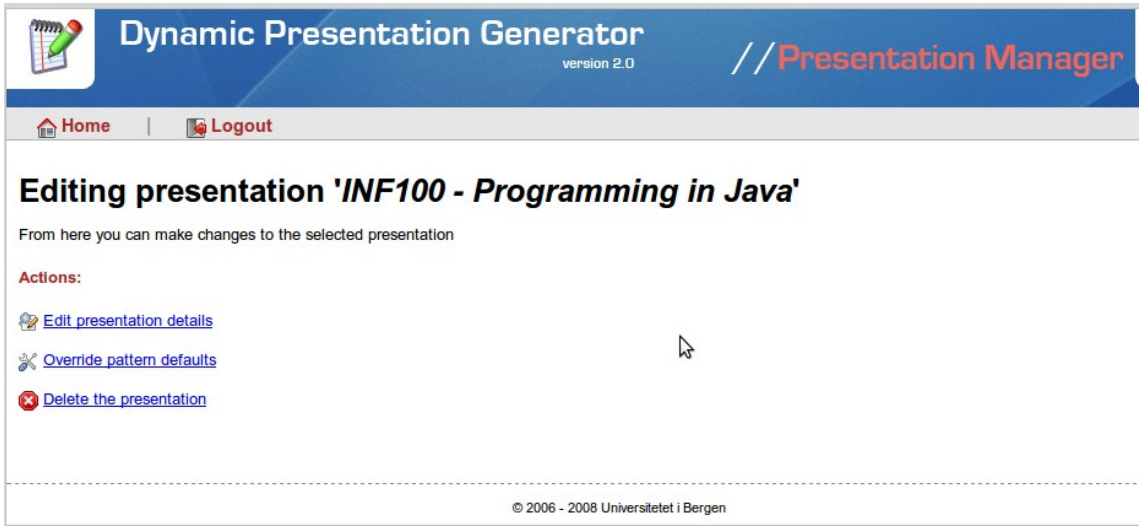
Etter innloggingа blir brukaren som sagt sendt vidare til ei oversikt over presentasjonar han eller ho har tilgang til (Figur 3). Her kjem rolle-basert tilgangskontroll inn i bildet. Informasjonen vist vil variere alt etter kva brukarar som er innlogga. Administratorar vil sjå alle presentasjonar i systemet, medan *publishers* og *readers* kun vil ha tilgang til dei aktiverte presentasjonane dei er knytt opp mot. Individuelle brukarar si tilknytning til persentasjonar er handtert av Webucator 3.



Figur 3: DPG 2.0 – Lobby – Innlogga brukar

Presentation Manager (PM)

Kun brukarar med rolla administrator har tilgang til dette delsystemet. Her kan ein opprette, kopiere, endre og fjerne presentasjonar og persentasjonsmønster. Ein kan også knytte ein presentasjon til ei gitt brukargruppe som skal ha tilgang til denne. Systemet ser vi i figur 4.



Figur 4: DPG 2.0 – Presentation Manager

Administratorar kan også overkøyre transformasjoner, sidemaler og stilark i denne delen av systemet. Med andre ord er det mogleg å definere spesielt for enkelte presentasjonar korleis transformasjonar, sidemaler og stilark skal framstå. Dette inneber at ein ynskjer å ta imot kode frå brukarane. Sidan det er tilfellet er det knytta potensielle utfordringar til sikring av dette delsystemet. Input validering blir problematisk med ein gong ein ber

om å kunne ta imot relativt rik kode, som JavaScript og HTML for fullstendige HTML-sider.

Presentation Viewer (PV)

Denne komponenten rendrer presentasjonane og viser dei til brukarane som ei nettside. Alle brukarar har tilgang til PV, så lenge dei er autentiserte, og har tilgang til den aktuelle presentasjonen. I figur 5 er for eksempel sidene *Startsiden, Fremdriftsplan, Kursinformasjon* synlege oppe i høgre hjørne, og menyen på *Startsiden* viser utsnitta *Siste meldinger* og *Meldingsarkiv.*

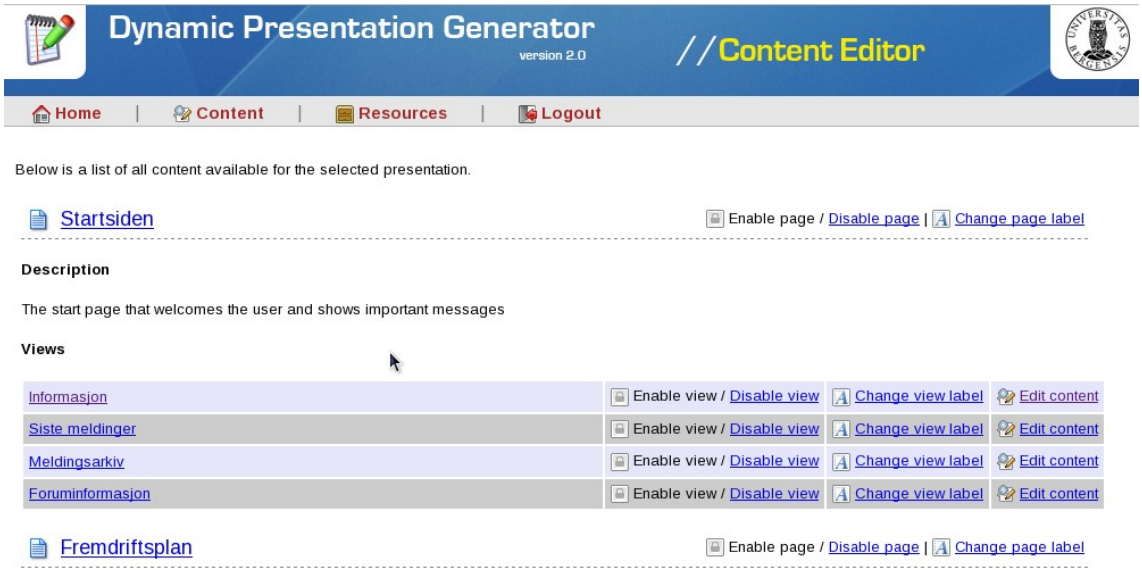
Kva informasjon som blir framvist er også avhengig av kva rolle ein brukar har. Rollene *publisher* og administrator gir blant anna tilgang til redigeringsfunksjonar (øverst på Figur 5), noko som vil føre dei vidare til *Presentation Content Editor* (PCE).

Figur 5: DPG 2.0 – Presentation Viewer

Presentation Content Editor (PCE)

Figur 6 viser Presentation Content Editor. I dette delsystemet har brukarane moglegheit til å redigere innholdet i presentasjonane. Brukarar med rolla *publisher* i presentasjonen dei ynskjer å redigere, samt administratorar i DPG 2 har tilgang til dette delsystemet.

I tillegg inneheld PCE eit ressurshandteringssystem, der det er mogleg for administratorar og publishers å opprette, laste opp, slette og flytte mapper og filer (til dømes bilete og dokument).



Figur 6: DPG 2.0 – Presentation Content Editor

2.2.3 Teknologiar som er i bruk

Utgangspunkta for val av teknologiar til *DPG 2* var at systemet skulle kunne køyre på instituttet sine webserverar. I tillegg var det viktig at forskjellige system avhengige av kvarandre, som *Webucator 3* og *DPG 2* skulle ha relativt lik oppbygging. Det vart også lagt vekt på at systema skulle bruke teknologi basert på open og fri kildekode [1].

Ei rekke forskjellige teknologiar er tatt i bruk i DPG 2. Sjølve koden er skrive i *Java 6* [6]. Systemet blir bygd med *Ant* [9] og køyrer på ein *Apache Tomcat* [7] tjener som JSP- og Servlet Container. I bunn står ein *PostgreSQL 8* [8] databaseserver. Som applikasjonsrammeverk har valet falt på *Spring Framework* [10]. *Spring Security*, ein del av *Spring Framework* er sentral i koblinga mot Webucator 3 for autentisering av brukarar. Dette blir gjort i ved bruk av *Remoting*-funksjonane i Spring.

2.3 Webucator 3

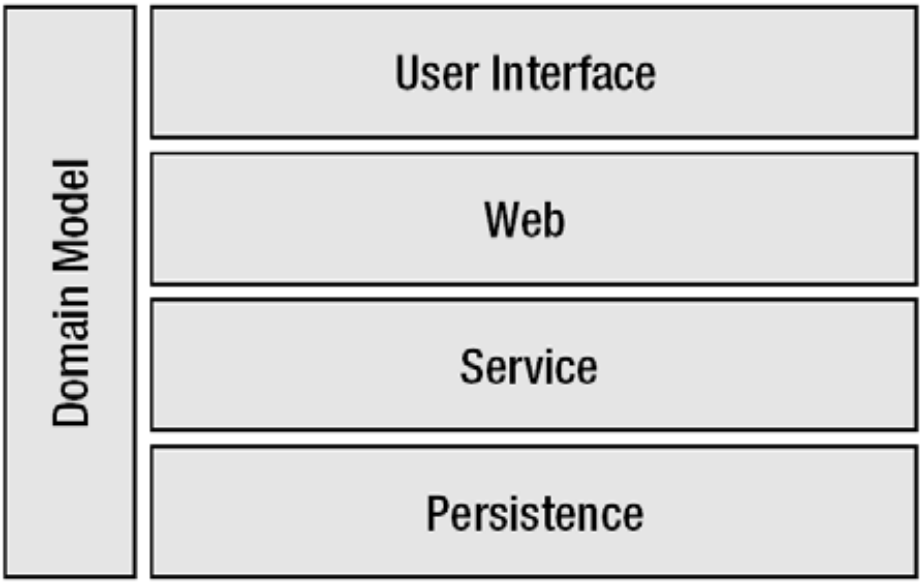
I eit system der ei rekke forskjellige brukarrollar er definerte, med behov for tilgangskontroll knytta til desse brukarane, var det også behov for eit eige system dedikert for å handtere brukarane og knytte dei opp mot roller og presentasjonar. Dette var utgangspunktet for utviklinga av Webucator 3. Webucator 3 vart opprinneleg utvikla av Kristian Skønberg Løvik som ein del av masteroppgåva han skreiv i 2008 [1]. Webucator 3 er eit sjølvstendig system, men har ein del innverknad på andre eventuelle system, som *DPG 2* gjennom handtering av brukarane og tilknytinga dei har til presentasjonar (tilgangskontroll).

2.3.1 Roller i Webucator 3.0

Webucator 3 har kun ei rolle for eigne brukarar. Dette er administrator rolla. Ein administrator i Webucator 3 er ikkje det samme som ein administrator i DPG 2. Desse to er heilt seperate, og Webucators eigne brukarar har ikkje tilgang til DPG 2. Med andre ord; brukarane som Webucator 3 handterer er i bruk av andre system, og ikkje i Webucator.

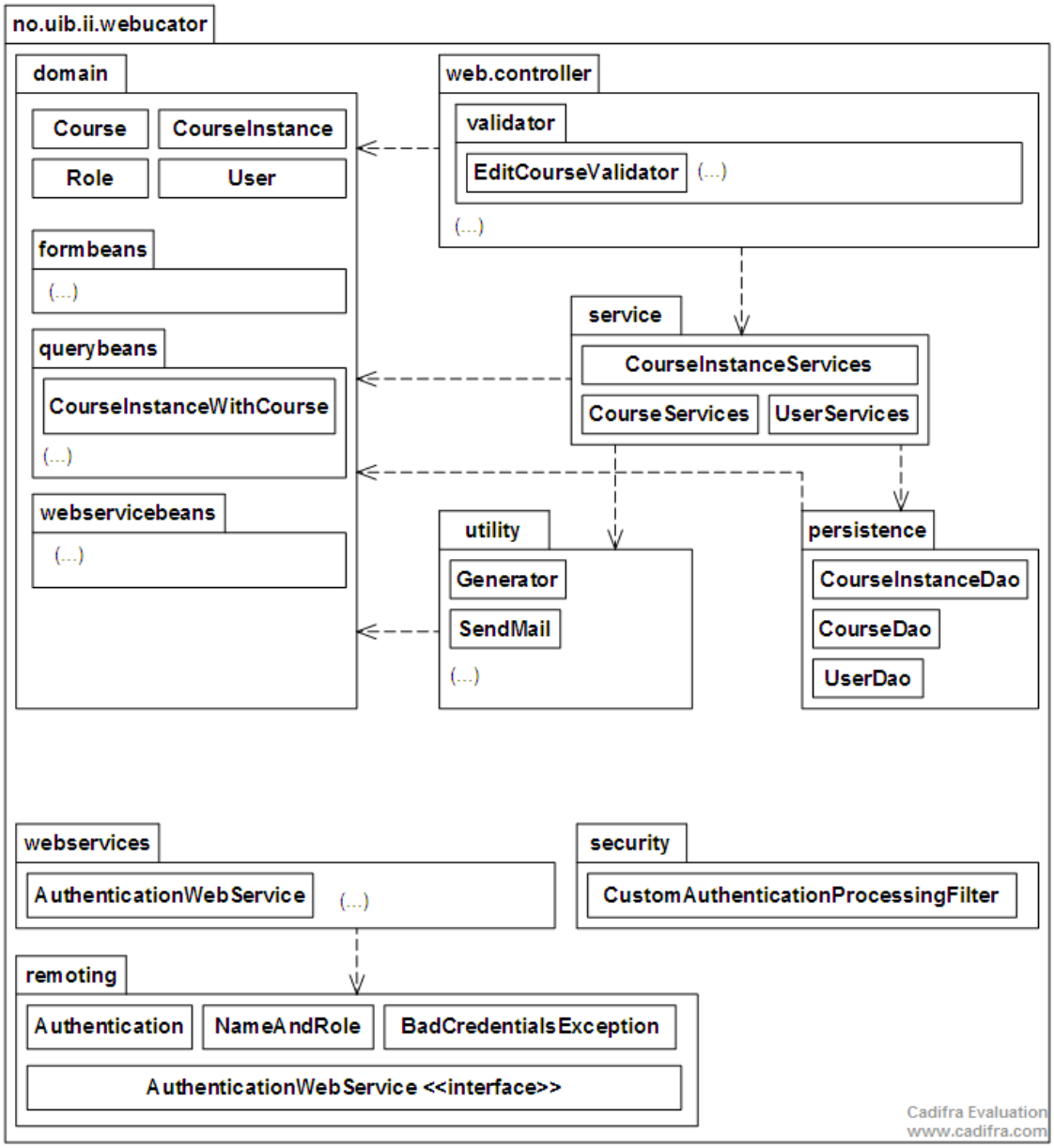
2.3.2 Struktur og arkitektur for Webucator 3.0

Webucator 3 har ein model-view-controller struktur. Med bruk av Spring Framework kan ein oppnå ein slik struktur med fem lag for abstraksjon (Figur 7)



Figur 7: Spring MVC [32]

Denne modellen er reflektert i pakkestrukturen til Webucator 3 (Figur 8) med unntak av brukargrensesnittet, som er i form av JSP-sider, og dei tre pakkane `security`, `remoting`, `utility` og `webservices`. Når vi ser på pakkane i Webucator 3 ser vi korleis dei ulike pakkane passar inn i Spring MVC.



Figur 8: Webucator 3.0 – Pakkestruktur [1]

- `domain`-pakken inneheld applikasjonen sine domeneobjekt. Dette er altså kurs, kursinstanser, roller og brukarar.
- `web`-pakken inneheld kontrollane til systemet, altså prosessering og validering av data.
- `service`-pakken tilbyr tenester og grensesnitt for ein del av kontrollørane til applikasjonen. Til dømes kan dette innebere databasekall og prosesseringer, slik at kontrollørane i `web`-laget kan ta i bruk slike funksjonar med enkle metodekall.
- `persistence`-pakken tek i bruk JDBC for kommunikasjon med databasen.

I tillegg ser vi tre pakkar som ikkje passar i Spring MVC modellen. Dette er `utility`, `security`, `webservices` og `remoting`.

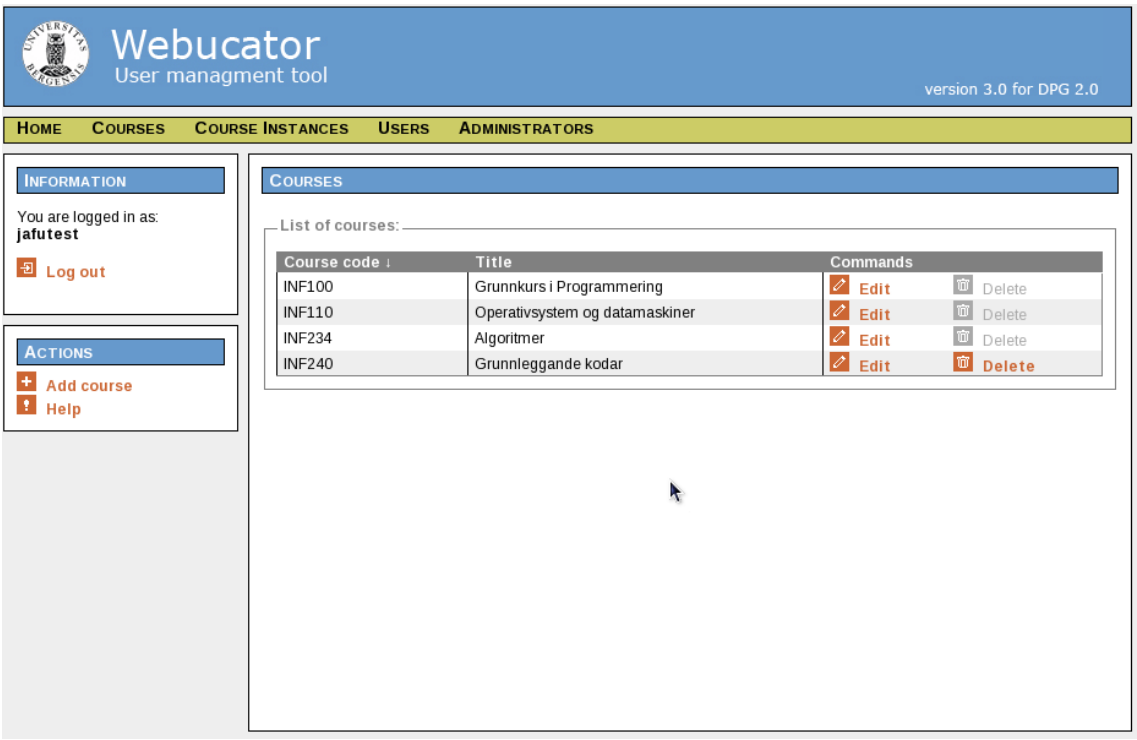
- `utility` pakken inneheld klasser for generering av brukarnamn og passord, grensesnitt for å sende e-post fra systemet med meir.
- `security` tek seg autentiseringsgrensesnittet som DPG 2 vil ta i bruk.
- `webservices` og `remoting` handterer kommunikasjonen med DPG 2.

2.3.3 Gjennomgang av funksjonalitet

Hovudoppgåva til Webucator 3 er å administrere kurs, kursinstansar og brukarar, og så tilby denne informasjonen til andre system som kan ha nytte av den, i dette tilfellet DPG 2 [1]. På denne måten kan brukarane, kursa og kursinstansane nyttast i fjernundervisningssammenheng.

Kurs

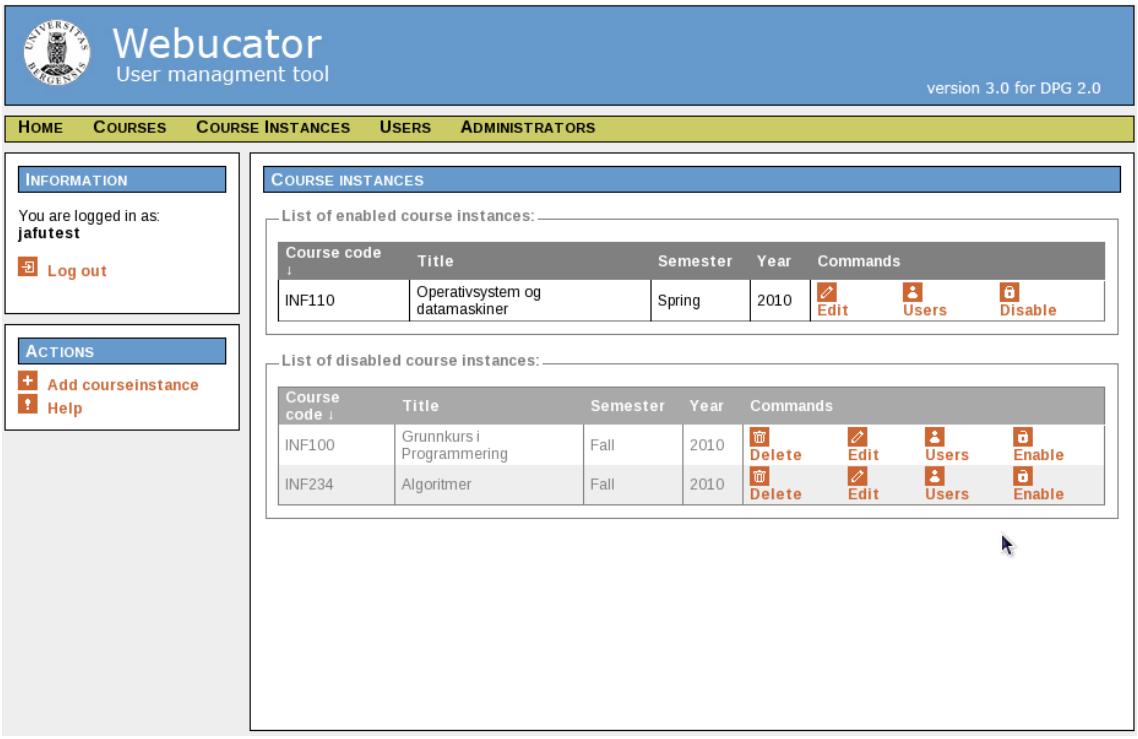
Kurssidene (figur 9) tilbyr ei oversikt over dei fjerundervisningskursa som er lagt inn i systemet, samt moglegheit for å redigere desse. Dette inneber å legge til nye, endre og slette eksisterande kurs. Eit kurs er representert med ein kurskode og eit namn, som til dømes *INF100 – Grunnkurs i programmering*.



Figur 9: Webucator 3.0 – Kursoversikt

Kursinstansar

Figur 10 viser instansar av fjernundervisningskurs. Her blir det spesifisert informasjon om kva semester (vår/haust) og år (for eksempel 2009). Det er mot disse instansane at brukarane er knytta. Kvar enkelt kursinstans har ei liste over brukarar som er tilknytta den aktuelle instansen, og kva rettigheiter (rolle) desse brukarane har. Dette er korleis *readers* og *publishers* blir knytta opp mot *presentasjonar*.



Figur 10: Webucator 3.0 – Kursinstansar

Brukarar i ein kursinstans

I oversikta over brukarar til ein kursinstans har ein moglegheit til å legge til, endre og fjerne brukarar. Dersom ein legg til ein brukar, inneber dette å fylle inn fornamn, etternamn og e-postadresse, samt kva rolle brukaren skal ha i kursinstansen (*reader/publisher)*. Det blir så oppretta eit brukarnamn. Eit passord blir vidare generert, og brukarnamnet og passordet blir sendt til e-postadressa som er oppgitt. Dette er informasjonen brukaren skal nytte når han eller ho skal logge inn i DPG 2.

Det er dessutan mogleg å importere ei *Excel*-fil som inneheld ei oversikt over kva brukarar som skal inkluderast i systemet. Det vil deretter bli automatisk generert brukarnamn og passord for desse brukarane når dei blir importerte i systemet.

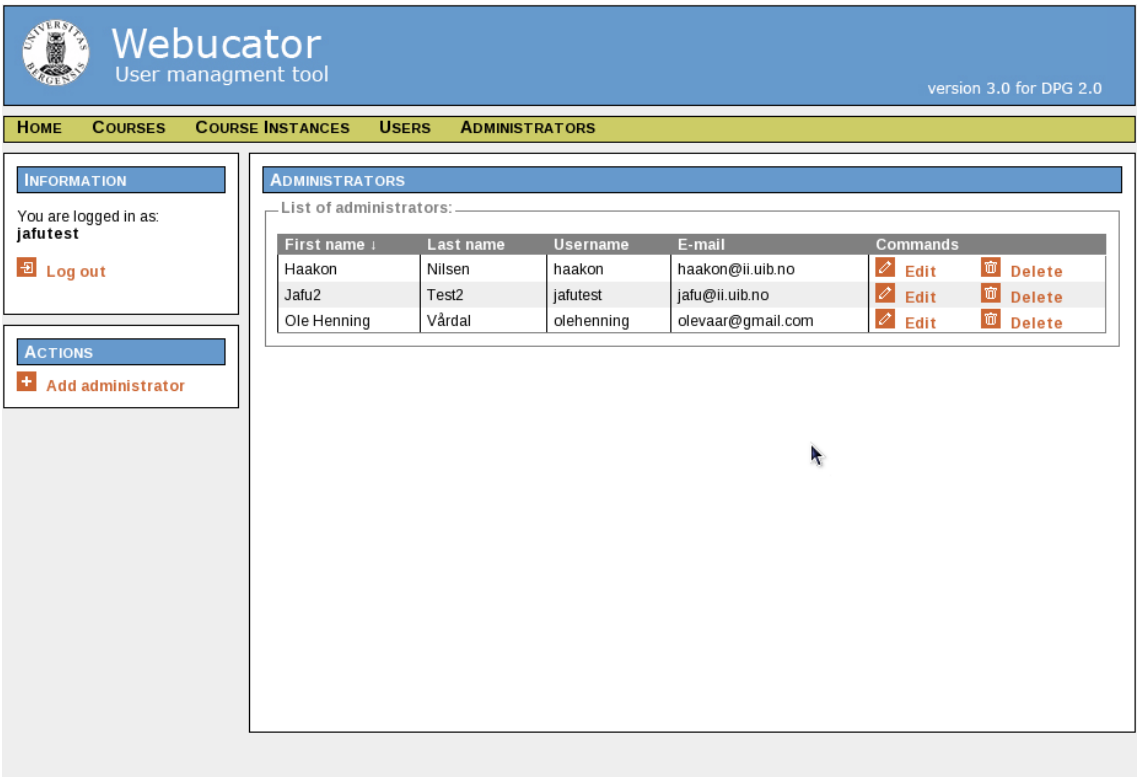
Ofte er det behov for at ein brukar kan vere publisher i fleire presentasjonar. Ein kan derfor importere *publishers* frå andre kursinstansar, slik at ein slepp å opprette desse på nytt for kvar kursinstans.

Andre brukarar

Elles har systemet ei oversikt over brukarar som ikkje er tilknytta nokon kursinstansar. Denne oversikta gir moglegheit for å fjerne desse brukarane slik at det ikkje går ut over ytelsen til resten av applikasjonen.

Administratorar

Under denne delen av systemet (figur 11) er det mogleg å legge til, endre og slette administratorar i DPG 2. Når ein veljer å legge til ein ny administrator, fyller ein inn detaljar som fornamn, etternamn, e-post adresse, brukarnamn og passord i eit skjema. Passordet vel ein sjølv, og det er ingen krav til styrke på passordet her. Dette blir diskutert nærmare i kapittel 4.



Figur 11: Webucator 3.0 – Administratoroversikt

2.3.4 Teknologiar i Webucator 3.0

Utgangspunktet for val av teknologiar i Webucator 3.0 var å sikre at systemet skulle kunne køyre på instituttet sine vevtenarar, samt at applikasjonen si oppbygging og

struktur ikkje skulle vere for ulik dei andre systema i JAFU. I tillegg var det viktig at teknologiane systemet skulle ta i bruk var basert på open kildekode.

I bunn køyrer instituttet Webucator 3.0 på ein *PostgreSQL8* [8] databaseserver. Webserveren køyrer *Apache Tomcat* [7] som JSP- og Servlet Container. Sjølve applikasjonen er skriven i Sun Java 6 [6] med Ant [9] som applikasjonsbygger. I likheit med DPG 2 er applikasjonsrammeverket Spring Framework [10] sentralt. Som sagt blir Spring Security og Spring Remoting nytta for at andre system som DPG 2.0 skal kunne gjennomføre autentisering mot Webucator 3.0.

3

3 Sikkerheitsanalyse av JAFU

Dette kapittelet presenterer sikkerheitsanalysa av JAFU-systema. Dette inneber både statisk analyse, kontroll av kildekode, og manuell sikkerheitstesting (penetreringstesting) av systemet. Funna vil bli presentert meir detaljert i kapittel 4 med fokus på korleis problema utartar seg i dei aktuelle systema.

3.1 Bakgrunn for analyse

Hausten 2008 vart det, som ein del av kurset INF226 ved Universitetet i Bergen, gjennomført ei omfattande analyse av DPG2 og Webucator 3 av studentane involverte i kurset [12, 13, 14, 15, 16, 17]. Hensikta var å avdekke sikkerheitsproblem i DPG 2 og Webucator 3. I tillegg skulle det vurderast løysingar, og studentane skulle få ei forståing av kvifor slike problem oppstår, korleis ein kan handtere dei, og korleis ein kan hindre sårbarheiter som dette i å oppstå i framtida. I hovudfokus var sikkerheitsproblem i vevapplikasjonar relatert til feil og avgrensningar i programkode [26]. Ei rekke nye sikkerheitsproblem med varierande grad av alvorlegheit vart avdekka under analysa.

Kurset tok for seg applikasjonssikkerheit og studentane skulle i løpet av hausten gjennomføre statisk og manuell kodeanalyse, generell sikkerheitstesting, samt evaluere ei rekke sikkerheitsmønster mot systema som vart testa. Bøkene *Secure Programming with Static Analysis* [19] og *Security Patterns – Integrating Security and Systems Engineering* [21] var pensum.

Analysene som blei gjennomført i 2008 danna grunnlaget til denne oppgåva. Analysene tok i hovudsak for seg resultatsettet returnert av Fortify SCAS. Ein del av problemområda vart belyste, men fokusen var avgrensa, og bredde og utbreiinga av problema var ikkje evaluert godt nok. I tillegg la ein del av studentane for mykje tillit til

det Fortify rapporterte, og det blei ikkje gjort gode nok vurderingar av potensielle falske positiver (tilfeller der problem blir rapportert når dei ikkje eksisterer). Resultatsettet frå desse opppgåvene blei på nytt evaluert, og det vart gjort ei meir detaljert manuell analyse av kildekode og dokumentasjon. Det blei vidare gjennomført ei meir omfattande penetreringstesting av applikasjonane. Resultata frå denne nye analysa blir presentert i dette kapittelet.

Fordi sikkerheitstestinga vart påbegynt *etter* at studentane involverte i sjølve utviklinga av DPG 2.0 og Webucator 3.0 hadde slutta ved universitetet, var det naturleg at vurderinga av systemet tok i bruk tre metodar for sikkerheitsanalysene:

- Gjennomgang og vurdering av dokumentasjonen utarbeidd av tidlegare studentar.
- Gjennomgang og vurdering av kildekode (med bruk av statisk analyse).
- Penetreringstesting med fokus på potensielle sikkerheitsproblem avdekka i dei to forrige punkta.

Penetreringstesting er ein metode for å evaluerer sikkerheita til eit system, ein applikasjon eller eit nettverk gjennom å simulere angrep [20]. Penetreringstesting er ofte effektivt når ein gjennomfører ei sikkerheitsanalyse av eit nettverk eller operativsystem. Dette er på grunn av at størsteparten av arbeidet går i å finne og utnytte kjende sikkerheitsfeil og sårbarheiter i teknologiane som er i bruk. Mot ein applikasjon er det derimot litt annaleis, sidan arbeidet går ut på å finne og utnytte nye sikkerheitsproblem og sårbarheiter i applikasjonen. Gjennom ei komplett vurdering av kildekode og dokumentasjon kan ein oppnå dette meir heilheitleg [20].

> *«If you fail a penetration test you know you have a very bad problem indeed. If you pass a penetration test, you do not know that you don't have a very bad problem.»*
>
> -- Gary McGraw

Fokusen har derfor vore på å finne og utnytte feil som ein mistenker eksisterer basert på gjennomgangen og vurderinga av kildekode og dokumentasjon, såkalla *focused penetration testing* [20].

Penetreringstesting er ein mykje brukt metode i dag, men eignar seg ikkje spesielt godt som den primære metoden for sikkerheitstesting av system som er under utvikling. Slik testing blir gjerne gjennomført seint i utviklinga sidan det er behov for eit køyrbart og relativt fullstendig system for å få resultat [20]. Dette inneber blant anna at kostnadane knytta til tetting av sikkerheitshol ein finn gjennom penetreringstesting gjerne er større enn om alternative metodar, som trusselanalyse, vurdering av design, utarbeiding av sikkerheitskrav, kodevurdering og liknande hadde vorte tatt i bruk tidlegare i utviklinga [20][19].

I kapittel 6 blir det gitt ein gjennomgang og ei vurdering av sikkerheitsrelaterte aktivitetar som bør vere ein del av utviklingsprosessen, og korleis desse kunne ha vorte brukt for å unngå sikkerheitsproblema DPG 2 og Webucator 3 står ovanfor i dag, med fokus på korleis ein kan nytte OpenSAMM og aktivitetane som er vektlagte i den modellen for å tilpasse sin eigen utviklingsprosess.

Som sagt har det blitt gjennomført statisk og manuel kodeanalyse av DPG 2.0 og Webucator 3.0. Til tross for at mange argumenterer for at kodeanalyse er tidkrevande og dyrt, er det i følge OWASP Code Review Guide [30] den raskaste og mest effektive måten å evaluere sikkerheitssituasjonen til vevapplikasjonar.

Statisk analyse er analyse av programvare som er gjennomført uten å faktisk køyre ein ferdigbygd versjon av applikasjonen ein analyserer. Statisk analyse er som regel gjennomført på kildekode. Enkelte verktøy som *FindBugs* [24] analyserer *bytekode* i staden for kildekode. Statisk analyse blir vanlegvis gjennomført automatisk med verktøy som går gjennom og samanliknar koden med mønster og reglar som representerer feil og sikkerheitsproblem. Hensikta er å avdekke *bugs*, feil i koden og sårbar kode [19]. Statisk analyseverktøy for sikkerheit skiller seg dessutan frå meir generelle verktøy for statisk analyse ved at dei under analysa opererer med eit sett med reglar som inneber stor bredde i eventuelle treff. Det er betre å rapportere om problem som ikkje eksisterer, enn å risikere at problem eksisterer i kildekoden som ikkje blir belyste [19]. Problem som blir rapporterte, men som i realiteten ikkje er problem bli kalt *falske positiver*. Problem som ikkje blir rapporterte, men som eksisterer blir kalt *falske negativer.*

Sjølv om verktøy som automatisk leiter fram sikkerheitsproblem i koden er nyttige, er det viktig å ikkje legge all tillit til slike verktøy. Statisk analyse har mykje til felles med stavekontroll i, til dømes, kontorprogram. Slike verktøy har problem med å sjå konteksten i koden, på same måte som stavekontroll kan ha problem med å finne feil i grammatikk og setningsoppbygging [19]. Statisk analyse bør derfor helst nyttast som eit ledd i ein større og meir omfattande prosess for manuell analyse av kode.

Verktøyet som har blitt brukt i den statiske analysa av DPG 2 og Webucator 3 er *Fortify Static Analysis Tool* [23]. Verktøyet er produsert av *Fortify Software* og er laga spesielt for å avdekke sikkerheitsproblem i kildekoden.

3.2 Påviste sikkerheitsproblem i JAFU

Før resultata blir presenterte vil det bli gitt ei oversikt over dei typane sikkerheitsproblem som har blitt påvist i DPG 2.0 og Webucator 3.0. Det er mogleg at framtidig analyse vil kunne påvise fleire sikkerheitsproblem enn dei som er ein del av denne oppgåva.

Cross Site Scripting

XSS (Cross-Site Scripting) er eit utbredt sikkerheitsproblem på Internett i dag. I følge [39] kan så mange som 90% av alle vevapplikasjonar ha problem med cross site scripting. *OWASP Top Ten*-prosjektet tilbyr ei oversikt over dei mest kritiske sikkerheitsproblema som rammer vevapplikasjonar i dag [22]. OWASP gir ut ein ny rapport kvart tredje år med ei slik oversikt. I rapporten som var publisert i april 2010 blei cross site scripting rangert som den nest mest alvorlege sikkerheitsfeilen, rett bak innsprøytingsangrep (f.eks. Innsprøyting av SQL og LDAP) [22]. Den forrige rapporten, som kom i 2007 vurderte den gangen XSS som det mest alvorlege sikkerheitsproblemet vevapplikasjonar står ovanfor [22], hovudsakeleg på grunn av utbreiinga av slike angrep og sårbarheiter.

XSS angrep gjer at ein ondsinna brukar kan stjele sensitiv informasjon frå offeret, som informasjonskapslar, skjemainnhald og liknande [18, 19]. I tillegg vil slike angrep gi moglegheit for å køyre kode lokalt på maskina til offeret, endre utsjånad på sida og omdirigere brukaren til skadelege nettsider. Det er mogleg å nytte XSS som ledd i phishing-angrep [39]. XSS-angrep har til og med blitt brukt for å spre ormar i vevapplikasjonar [19].

XSS angrep eksisterer i hovudsak i to former, *ikkje-persistente* eller *reflekterte* og *persistente* angrep [37]. Det fins også mindre vanlege DOM-baserte angrep [39]. Her tek vi for oss dei to mest vanlege.

Ikkje-persistente eller reflekterte angrep (også kalt Type 1 [40]) skjer når applikasjonen tek imot inndata frå ein brukar, og gjengir denne inndataen direkte som ein del av den neste sida som blir generert. Angriparar tek i bruk *social engineering* for å få legitime brukarar til å besøke ein URL som inneheld skadeleg HTML og JavaScriptkode. Offeret sin nettlesar vil vise fram HTML-dokumentet og køyre scriptet inkludert av angriparen. For å hindre denne typen angrep bør brukarar vere varsomme med kva lenker dei trykker på. Dette er ikkje alltid like lett, sidan ein angriper kan skjule innhaldet i adressa gjennom blant anna å *enkode* det skadelege innhaldet, eller ta i bruk verktøy som TinyURL for å pakke ein URL inn i ein annan.

Persistente angrep (også kjend som Type 2 [40]) skjer når applikasjonen tek imot data og dette, typisk i ein database eller i filsystemet. Dette blir så henta fram igjen kvar gong ein brukar ber om å få innhald frå sida. Med andre ord blir potensiell skadeleg kode gjengitt kvar gong sida blir rendra [19][40]. For å kunne gjennomføre slike angrep bør ein brukar kunne gi data til applikasjonen som skal lagrast.

Loggforfalsking

Applikasjonar brukar ofte loggar for å lagre ei historie av hendingar og transaksjonar

slik at dei seinare kan lesast for evaluering, statistikksamling og *debugging*. Loggforfalsking oppstår når data blir gjenngitt uvalidert frå applikasjonen til loggen [19]. Dette medfører at brukarar kan manipulere innhald i loggane [19]. På denne måten kan hendelsesforløp enten bli forfalska eller forstyrra slik at det ikkje er mogleg å få eit klart bilde av kva som har skjedd.

Tenestenekting

Tenestenekting er å forstyrre tilgjengelegheita til ein applikasjon [41]. Dette medfører at legitime brukarar ikkje får gjennomført oppgåver mot systemet heilheitleg, og ikkje får tak i ressursar og tenester dei treng. Feil som fører til at applikasjonen blir utilgjengeleg, og som er mogleg å tvinge fram kvalifiserer som tenestenekting. Ein annan vanleg type tenestenektingsangrep som rammar mange system på verdensveven er når ein angripar sender ei stor mengde med spørringar til tenaren og gjer det umogeleg for legitime spørringar å få svar.

Katalogtraversering

Fleire vevapplikasjonar i dag er avhengige av lokale ressursar som bilete, filer, script og liknande. Katalogtraversering inneber at brukarane manipulerer parameteret som peiker på den lokale ressursen, og på den måten beveger seg oppover i katalogstrukturen slik at ressursar som ikkje skal vere tilgjengelege, blir det [42].

Svak autentisering

Autentiseringsproblem handlar om tilfelle der identifikasjon og verifisering (innanfor ei viss grad av tillit) av at brukaren er den han utgir seg for å vere ikkje er god nok. Dette kan innebere tilfelle av brukarnamn og passord som autentisering der passorda er så svake at ein angriper kan gjette dei, og kan prøve å autentisere fleire gongar uten at det blir sperra. Dette er rangert som det tredje mest alvorlege problemet som rammar vevapplikasjonar i dag i følge [22].

Usikker lagring

Sensitiv data som passord og personlig informasjon må lagrast sikkert. Usikker lagring medfører at eksponert data gir viktig informasjon til ein angripar. Dette inneber informasjon som er lagra med usikker kryptering eller i klartekst. Dette problemet er rangert som det sjuande mest alvorlege problemet i vevapplikasjonar i følge [22].

Usikker kommunikasjon

Manglande bruk av TLS (Transport Layer Security) ved hjelp av HTTPS kan medføre at sensitiv data i overføring mellom to partar blir avlytta av ein angriper. I tillegg medfører det at partane ikkje kan vere sikker på kven dei kommuniserer med, noko som legg til rette for *mann-i-midten* angrep. Manglande bruk av TLS er rangert som det niande mest alvorlege sikkerheitsproblemet i vevapplikasjonar i følge [22]

Usikker generering av sensitiv data

Når sensitive data som passord og liknande skal genererast av ein applikasjon er det viktig at det ikkje er mogleg for uvedkommande å forutsjå eller finne ut av kva data som blir generert. Dette inneber forutsigbar struktur, bruk av statistiske slumptallsgeneratorar og liknande.

Informasjonslekkasje

Informasjonslekkasje oppstår når applikasjonen gir frå seg data om interne prosedyrer og metodar som helst ikkje skal bli eksponerte for brukaren. Dette gjer at angriparar kan samle informasjon om systemet og planlegge korleis eventuelle angrep skal gjennomførast. Dette problemet blei kun påvist i DPG 2.0, og der er hovudsakeleg der dette ville vere eit problem sidan systemet skal vere tilgjengeleg for fleire brukarar, og ikkje kun administratorar. Likevel er tilfella enkle å handtere gjennom å fjerne avslørande HTML-kommentarar og stoppe Tomcat frå å gi *stack trace* til brukaren. Problemet vil derfor ikkje bli diskutert i særleg stor grad vidare.

3.3 Analyse av DPG 2

Statisk analyse av DPG 2 rapporterte om 254 sikkerheitsproblem. Fortify deler inn sikkerheitsproblem i tre forskjellige kategoriar. *Hot* er dei mest kritiske sikkerheitsproblema. Dette er sårbarheiter som cross site scripting og injiseringsangrep (SQL, LDAP og liknande). *Warning* tek for seg middels alvorlege problem som informasjonslekkasje, forfalsking av loggar, ressursar som ikkje blir frigitte, manglande sjekk for nullverdiar og liknande. *Non-critical info* tek for seg mindre alvorlege problem som feil i unntakshandtering (for breie catch og throw klausular m.m.), ubrukt kode og liknande.

Fortifys rangering av problem tek ikkje for seg applikasjonslogikk og kontekst når problema blir klassifiserte. Det har derfor blitt gjort ei manuell vurdering basert på den type resultat verktøyet returnerte. Under denne prosessen blir *falske positiver* og *falske negativer* luka ut. Falske positiver er når verktøyet rapporterar om problem som ikkje eksisterer. Falske negativer på si side er meir alvorlege. Dette er problem som eksisterer, men som verktøyet ikkje klarte å påpeike.

Fortify rapporterte også om ei stor mengde med problem som ikkje er utnyttbare på nokon måte, som derfor er mindre alvorlege, og vil ikkje bli vurderte her. Dette er problem som ikkje-serialiserbare objekt i sesjon og feil metode for samanlikning av `String`-objekt (til dømes bruk av == i staden for `equals`), som er dårleg praksis, men som ikkje inneber spesielt store problem i systema. .

Fortify påviste blant anna tilfelle av cross-site scripting og loggforfalsking. Dette ga eit

godt utgangspunkt for vidare analyse. Blant anna kan ein, ut i frå dei foreløpige resultata, tenke seg til at lite eller ingen form for validering og sanitering av inndata, samt enkoding av utdata har blitt gjennomført. Dette kan altså bety at andre feil relatert til dette kan eksistere i applikasjonen.

Når det gjeld problema som har blitt avdekka og kor alvorlege dei er, så er dette dessutan avhengig av korleis dei kan utnyttast, samt kven som gjennomfører angrep. Det vil seie om angrepsoverflata til applikasjonen tillet uvedkommande å manipulere inndata på ein slik måte at problema blir utnytta, samt kva ein faktisk kan oppnå i denne samanhengen.

3.3.1 Core

Det har kun blitt påvist problem knytta til loggforfalsking og katalogtraversering i delsystemet Core. Desse tilfella kan vi sjå i tabell 1. Ein del av problema knytta til ressurshandteringssystemet i PCE kan seiast å vere ein del av `persistence`-pakken i Core, men problemet er introdusert via PCE, og blir vidare presentert seinare.

Filnamn	**Funksjon**	**Inndata**	**Utnyttbart**	**Info**
`AbstractDpg Authentication Filter.java`	Vurdere om ein brukar har tilgang til ein presentasjon.	Presentasjons-ID	Ja	Loggforfalsking
`XmlTransformer.java`	Transformasjon av XML-dokument.	Entitetssti	Ja	Loggforfalsking
`JcrResourceDao.java`	Persistens-funksjonar knytta til ressurs-handtering (PCE)	Entitetssti	Ja	Loggforfalsking
`PresentationResource Controller.java`	Hente filer knytta til presentasjonar	Katalogsti	Ja	Katalogtraversering

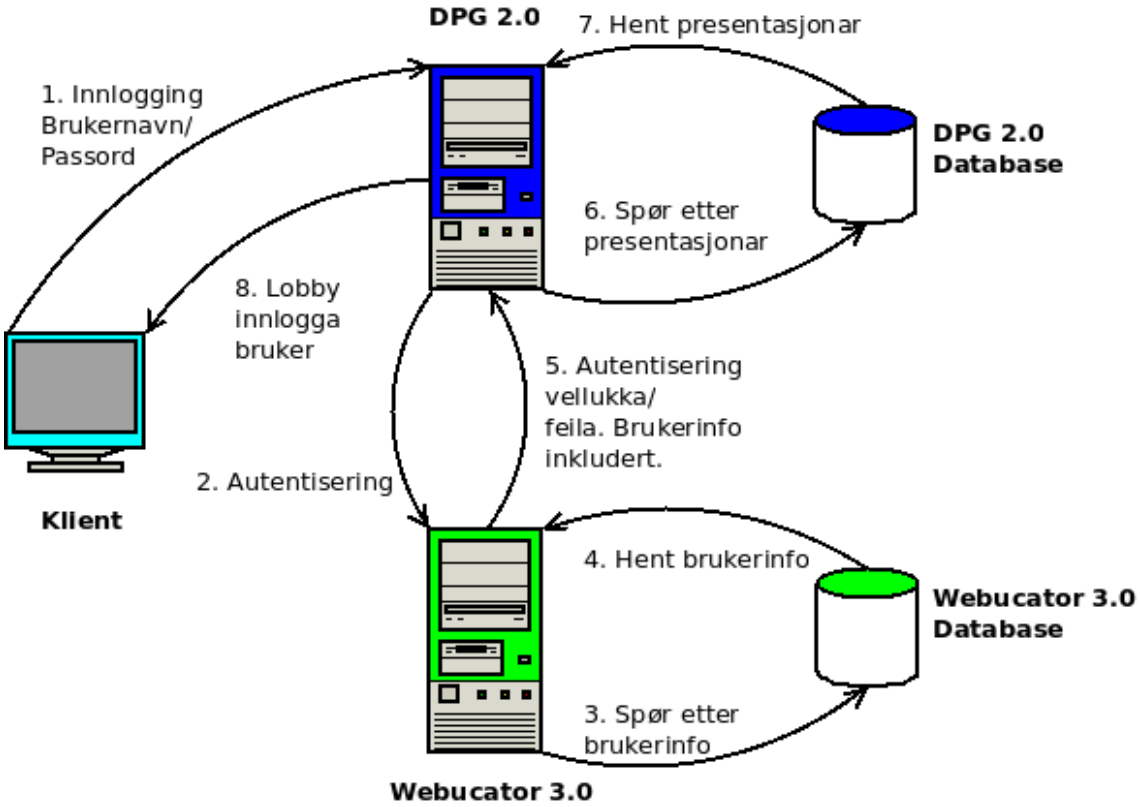
Tabell 1: DPG 2.0 – Sikkerheitsproblem i Core

3.3.2 Lobby

XSS er gjennomførbart mot delsystemet Lobby. Problemet her er input som blir mottatt frå Webucator 3. Dette består i hovudsak av fornamn og etternamn knytta til den aktuelle innlogga brukaren. Via to metodar for å gi input til Webucator 3 er det mogleg å legge til script som blir køyrd når brukaren har logga inn, sidan namnet til brukaren blir

presentert i lobbyen.

I figur 12 er innloggingа illustrert. Først sender ein klient sin innloggingsdata til DPG 2.0. Dette inneber brukarnamn og passord. DPG 2.0 på si side kontaktar Webucator 3.0, som verifiserer brukarnamnet og passordet. Dersom brukaren eksisterer og passordet er korrekt, svarer Webucator 3 med informasjon om fornamn, etternamn, brukarnamn, kva presentasjonar brukaren tilhøyrer og kva rolla til brukaren er i desse presentasjonane. Her er det fornamn, og etternamn som kan innehalde skadeleg kode i form av for eksempel JavaScript.



Figur 12: DPG 2.0 - Lobby - Innlogging

Dette problemet blei ikkje belyst av Fortify, og kan reknast som ein falsk negativ. Dette viser kor viktig det er å analysere begge systema samstundes, sidan dei er så avhengige av kvarandre. Samtidig viser det svakheiter knytta til statisk analyse og mangelen på ei god vurdering av konteksten til data, samt viktigheita med å gjennomføre analyse manuelt i tillegg. Problemet blir vidare diskutert i kapittel 4.

Delsystemet har dessutan manglar i den form av at det ikkje eksisterer nokon avgrensningar for antall innloggingsforsøk. Dette gjer at autentiseringa blir langt svakare enn ein skulle ønske. Lett tilgjengelege brukarnamn, ofte med forutsigbar struktur, samt svake passord, som blir diskutert seinare, produserer svakheiter i som ikkje burde vere der.

3.3.3 Presentation Content Editor

Som tidlegare nemnt handterer dette delsystemet redigering av presentasjonar i DPG 2. Dette delsystemet inneheld ein stor del av dei sårbarheitene som vart avdekka. Spesielt er det her ein del av vektorane for XSS-angrep er i DPG 2.

Tabell 2 inneheld ei oversikt over dei viktigaste problema. Enkelte av problema i tabellen er ikkje utnyttbare i den versjonen av systemet som vart testa, men det var viktig å inkludere dei i oversikta. Dette er på grunn av at etter kvart som systemet utviklare seg i framtida er det mogleg at desse problema blir aktive, og kan då vere attraktive for angrep.

Filnamn	Funksjon	Inndata	Utnyttbart	Info
`listContent.jsp`	Gir ei liste over innhaldet som skal endrast	Status Message	Ja	XSS (reflektert)
		Presentasjons-ID	Nei	Potensiell XSS (reflektert)
`listResources.jsp`	Gir ei liste over ressursar som skal endrast	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Katalogsti	Ja	Katalogtraversering
		Status Message	Ja	XSS (reflektert)
`editContentForm.jsp/ EditContentSubmit Controller.java`	Redigering av tillegg og endring av innhold i ein presentasjon.	HTML-skjema.	Ja	XSS (persistent) Loggforfalsking
		Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Side-ID	Nei	Potensiell XSS (reflektert)
		Utsnitt-ID	Nei	Potensiell XSS (reflektert)
`editViewLabel.jsp/ EditViewLabelForm Controller.java`	Redigering av endring av namn på utsnitt *(views)*	HTML-skjema.	Ja	XSS (persistent)
		Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Side-ID	Nei	Potensiell XSS (reflektert)
		Utsnitt-ID	Nei	Potensiell XSS (reflektert)
`editPageLabel.jsp/ EditPageLabelForm Controller.java`	Redigering av endring av namn på side *(page)*	HTML-skjema.	Ja	XSS (persistent)
		Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Side-ID	Nei	Potensiell XSS (reflektert)
`upload ResourceFileForm.jsp/ UploadResourceFile Controller.java`	Verktøy for opplasting av filer til ein presentasjon	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Katalogsti	Ja	Katalogtraversering
`newResource FolderForm.jsp`	Oppretting av katalogar til ein presentasjon	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		Katalogsti	Ja	Katalogtraversering
`viewDetails.jsp`	Oversikt over avsnitt *(views)* som skal redigerast.	Utsnitt-ID	Nei	Potensiell XSS (reflektert)
		Presentasjons-ID	Nei	Potensiell XSS (reflektert)
`PresentationContent EditorService Impl.java`	Funksjonar knytta til redigering av view, page og presentasjon	presentationId	Nei	Potensiell Loggforfalsking
		Utsnitt-ID	Ja	Loggforfalsking
		Side-ID	Ja	Loggforfalsking

Tabell 2: DPG 2.0 – Sikkerheitsproblem i PCE

Dei problema som har HTML-skjema som punkt for inndata blei ikkje belyst av verktøyet for statisk analyse. Dette betyr med andre ord at dei er å rekne som falske negativer. Dette er på grunn av at Fortify har problem med å evaluere korleis skjemainnhaldet blir brukt i applikasjonen.

3.3.4 Presentation Viewer

Dette delsystemet er sårbart for persistente XSS-angrep. Angrepsoverflata i dette tilfellet er i Presentation Content Editor og Presentation Manager, altså er det ikkje mogleg å gi skadelege script til systemet via Presentation Viewer. Likevel blir slike script som er introduserte via dei andre systema, reflekterte som ein del av presentasjonane som er framvist i Presentation Viewer.

Tabell 3 viser sikkerheitsproblema som er til stade med angrepsoverflate i Presentation Viewer. Alle desse problema er knytta til forfalsking av loggar. Merk at parameteret for entitetshandlinger ikkje har nokon funksjon i DPG 2.0 i dag.

Filnamn	**Funksjon**	**Inndata**	**Utnyttbart**	**Info**
`Presentation Controller.java`	Framvising av presentasjon til brukaren	Presentasjons-ID	Ja	Loggforfalsking
		Side-ID	Ja	Loggforfalsking
		Utsnitt-ID	Ja	Loggforfalsking
		Entitets-handling	Ja	Loggforfalsking
		Entitetssti	Ja	Loggforfalsking
`PresentationViewer ServiceImpl.java`	Framvising av presentasjon til brukaren	Entitetssti	Ja	Loggforfalsking

Tabell 3: DPG 2.0 – Sikkerheitsproblem i PV

3.3.5 Presentation Manager

Det vart påvist XSS-problem også i dette delssystemet. Problema er her like lette å utnytte for nokon med tilgang, som i ein del av dei tilfella i PCE, men trusselen er langt lavare sidan kun administratorar skal ha tilgang til dette delsystemet. Avhengig av andre utnytta sårbarheiter er det framleis mogleg å oppnå tilgang, enten ved overtaking av andre sesjonar eller brot på autentisering, derfor må problemet likevel vurderast i like stor grad som andre.

Her er XSS-problema knytta til funksjonen *Override Master Template*. I dette tilfellet tek applikasjonen imot HTML kode frå ein administrator for å endre på utsjånad og struktur til ein gitt presentasjon. Sidan det her skal vere mogleg for ein administrator å gi rik HTML-kode der legitim JavaScript kode også kan vere inkludert er det ikkje

mogleg å hindre brot på sikkerheit her utan å fjerne nødvendig funksjonalitet frå administratorar. Ein må prioritere arbeid med å redusere problemet, blant anna gjennom å passe på at uvedkommande ikkje skal ha tilgang til PM-systemet. Dette blir diskutert nærmare i kapittel 4 og 5.

Filnamn	**Funksjon**	**Inndata**	**Utnyttbart**	**Info**
`deletePresentation Form.jsp`	Sletting av presentasjonar.	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
`editPresentation DetailsForm.jsp/ EditPresentation DetailsForm Controller.java`	Endring av informasjon knytta til ein presentasjon.	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
`newPresentation Form.jsp/ NewPresentationForm Controller.java`	Oppretting av nye presentasjonar	HTML-skjema	Ja	XSS (persistent)
`newPresentationFrom CopyForm.jsp/ NewPresentationFrom CopyForm Controller.java`	Kopiering av presentasjonar	HTML-skjema	Ja	XSS (persistent)
`editPresentation DetailsSuccess.jsp`	Tilbakemelding om endring.	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
`overrideMaster TemplateForm.jsp/ OverrideMaster Template Controller.java`	Endring av layout til ein presentasjon.	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
`overridePage TemplateForm.jsp/ OverridePage Template Controller.java`	Endring av layout til ein presentasjon.	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
`overrideMaster Transformation Form.jsp/ OverrideMaster Transformation FormController.java`	Endring av transformasjonar	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
`overridePattern DefaultsOverview.jsp`	Oversikt over redigerings-moglegheiter	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
`overrideStylesheet Form.jsp/ OverrideStylesheet FormController.java`	Endring av CSS stylesheets	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
`OverrideView Transformation Form.jsp/ OverrideView Transformation FormController.java`	Enring av transformasjonar	Presentasjons-ID	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)

Tabell 4: DPG 2.0 – Sikkerheitsproblem i PM

3.4 Analyse av Webucator 3

Sikkerheit i brukarhandteringssystem er absolutt kritisk. Sidan slike system har moglegheit til å støtte ei rekke andre applikasjonar vil moglege sikkerheitsproblem i eit slikt system ha innverknad på alle andre eventuelle system. Per i dag er det kun *DPG 2* som er knytt opp mot brukarane som Webucator 3 handterer.

I likheit med DPG 2.0 avdekka Fortify ei rekke sikkerheitsproblem, og på samme måte som i DPG 2.0 er det tydeleg at lite eller ingen validering og sanitering av inndata er gjennomført. Ein del av dokumentasjonen ga dessutan bekymringar i forhold til generering og lagring av brukardata, og ei rekke problem blei påvist på det området.

Dei alvorlegaste problema i Webucator 3.0 var å finne i pakkane utility, web, persistence, samt i brukargrensesnittet i form av JSP-filer. Fortify rapporterte også om ei rekke advarslar, men dei er ikkje å ansjå som spesielt alvorlege, og vil ikkje bli diskutert i nokon særleg detaljgrad. Dette er problem som usjekka returverdiar, forvirrande namngiving av felt og metodar i klasser og liknande.

3.4.1 Pakkane Remoting og Webservices

Som sagt handterer denne delen av Webucator 3.0 kommunikasjon med DPG 2.0. Mangel på kryptering her vil føre til at ein angripar kan *sniffe* data som brukarnamn og passord, samt utføre *mann-i-midten* angrep. I tillegg er det her ein bør handtere eventuelle forbedringar av autentiseringa for DPG 2.0.

3.4.2 Security-pakken

I likheit med DPG 2.0 er det ingen grenser for antall innloggingsforsøk i Webucator 3.0. Passord for Webucator 3.0 brukarar blir krypterte og lagt til manuellt. Dermed eksisterer det ingen reglar for passordstyrke, noko som kan føre til potensielle brute-force angrep.

3.4.3 Web-pakken

Tabell 5 viser ei oversikt over dei XSS-problema som blir lagra i databasen. Desse problema kjem inn via HTML-skjema og innhaldet blir ikkje validert før lagring. Desse problema er også lista under brukegrensesnittet.

Her er det persistent XSS som er problemet. Inndata blir lagra uvalidert i PostgreSQL databasen til Webucator 3.0, og kan seinare bli henta ut og brukt i både Webucator 3.0 og DPG 2.0

Filnamn	Funksjon	Inndata	Utnyttbart	Info
`ProcessExcel Controller.java`	Oppretting av brukarar frå excell-dokument	Felt i Excel-fil	Ja	XSS (persistent)
`AddAdministrator Controller.java`	Oppretting av administrator	HTML-skjema	Ja	XSS (persistent)
`EditAdministrator Controller.java`	Endring av brukardata for administrator	HTML-skjema	Ja	XSS (persistent)
`AddUser Controller.java`	Oppretting av brukar	HTML-skjema	Ja	XSS (persistent)
`EditUser Controller.Java`	Endring av brukardata	HTML-skjema	Ja	XSS (persistent)

Tabell 5: Webucator 3.0 – Sikkerheitsproblem i web-pakken

3.4.4 Service-pakken

Problemet relatert til skadeleg kode som kjem inn via ei excel-fil kan handterast i service. Sjølve fila kjem inn via `web`-pakken, og brukargrensesnittet, men brukardata blir henta ut i pakkane `service` og `utility`, og må derfor nevnast her (tabell 6).

Filnamn	Funksjon	Inndata	Utnyttbart	Info
`UserServices Class.java`	Oppretting av brukarar frå excell-dokument	Felt i Excel-fil	Ja	Persistent XSS

Tabell 6: Webucator 3.0 – Sikkerheitsproblem i service-pakken

3.4.5 Utility-pakken

Tabell 7 viser dei viktigaste problema knytta til utility pakken. Dette gjeld i hovudsak generering av brukarnamn og passord, samt innhenting av data frå ei excel-fil.

Usikker generering av passord blei belyst av Fortify, men vart rangert som *non-critical info*. Dette er standardrangeringa for usikker generering av tilfeldig data. Problemet er derimot meir alvorleg enn det som kjem fram i Fortify rapporten.

Filnamn	Funksjon	Inndata	Utnyttbart	Info
`Generator.Java`	Genererer brukarnamn og passord	Fornamn, etternamn.	N/A	Forutsigbar passordstruktur på tilfeldig genererte passord.
				Forutsigbar struktur på brukarnamn
				Usikker slumptallsgenerator for generering av passord.
`GetUsersFromExcel Class.java`	Oppretting av brukarar frå excell-dokument	Felt i Excel-fil	Ja	Persistent XSS

Tabell 7: Webucator 3.0 – Sikkerheitsproblem i utility-pakken

3.4.6 Persistence-pakken

Eit problem som derimot var til stades i persistence-pakken var mangelen på sikring av sensitive data (tabell 8). Dette inneber at til dømes passord til både brukarar og administratorar blir lagra i klartekst. Slik data bør krypterast før det blir lagra. Dette inneber også endringar i `webservices`-pakken.

Filnamn	Funksjon	Inndata	Utnyttbart	Info
`UserDaoJdbc.java`	Lagring av brukardata	Fornamn, etternamn, passord, brukarnamn, e-post, roller	N/A	Usikker lagring av sensitiv data. Passord blir lagra i klartekst for alle brukarane i DPG 2.0

Tabell 8: Webucator 3.0 – Sikkerheitsproblem i persistence-pakken

3.4.7 Brukargrensesnitt - JSP-filer

Webucator 3.0 er sårbart for eit stort antall XSS-angrep, og spesielt her reflekterte XSS-angrep. Tabell 9 viser ei oversikt over dei aktuelle filene. Dei persistente problema her vart også belyste i kapittel 3.4.3.

Filnamn	Funksjon	Inndata	Utnyttbart	Info
addUserForm.jsp	Oppretting av brukar	HTML-skjema	Ja	XSS (persistent)
		Kursinstans-ID	Ja	XSS (reflektert)
addUserSuccess.jsp	Tilbakemelding etter oppretting av brukar	Kursinstans-ID	Ja	XSS (reflektert)
		HTML-skjema	Ja	XSS (reflektert)
addAdministrator Form.jsp	Oppretting av administratorar	HTML-skjema	Ja	XSS (persistent)
addAdministrator Success.jsp	Tilbakemelding etter oppretting av administrator	HTML-skjema	Ja	XSS (reflektert)
editAdministrator Form.jsp	Redigering av administrator	Brukarnamn	Nei	Potensiell XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
editAdministrator PasswordForm.jsp	Redigering av administrator (passord)	Brukarnamn	Nei	Potensiell XSS (reflektert)
editAdministrator PasswordSuccess.jsp	Tilbakemelding , passordendring	Brukarnamn	Ja	XSS (reflektert)
editCourseInstance Form.jsp	Redigering av kursinstansar	Kursinstans-ID (chosenId)	Ja	XSS (reflektert)
editUserForm.jsp	Redigering av brukardata	Kursinstans-ID	Ja	XSS (reflektert)
		HTML-skjema	Ja	XSS (persistent)
editUserSuccess.jsp	Tilbakemelding etter redigering av brukar			
editAdministrator Success.jsp	Tilbakemelding etter endring av administrator	HTML-skjema i	Ja	XSS (persistent)
helpAddUser.jsp	Hjelp-side for oppretting av brukarar	Kursinstans-ID	Ja	XSS (reflektert)
helpListUsers ForCourse.jsp	Hjelp-side for brukarliste	Kursinstans-ID	Ja	XSS (reflektert)
importPublishers Success.jsp	Tilbakemelding etter import.	Kursinstans-ID	Ja	XSS (reflektert)
listPublishers.jsp	Liste av publishers.	Kursinstans-ID	Nei	Potensiell XSS (reflektert)
listUsersFor Course.jsp	Liste av brukarar knytta til ein kursinstans.	Kursinstans-ID	Nei	Potensiell XSS (reflektert)
		Error Message	Ja	XSS (reflektert)
		Success Message	Ja	XSS (reflektert)
uploadUsersForm.jsp	Opplasting av excell-fil med brukardata	Kursinstans-ID	Ja	XSS (reflektert)
		Excell-fil	Ja	XSS (persistent)

Tabell 9: Webucator 3.0 – Sikkerheitsproblem i jsp-filene

3.5 Oppsummering

Mengda problem påviste i DPG 2.0 og Webucator 3.0 var større enn først antatt. I rapportane som var skrive av studentane som tok INF226 hausten 2008 var fokusen ofte på dei rå resulta frå Fortify, og ikkje nok arbeid var nedlagt i ei vurdering av utbreiinga til desse problema. Angrepsoverflata var ikkje kartlagt skikkeleg, noko som gjorde at det ville vere vanskeleg å bruke rapportane som utgangspunkt for å implementere sikkerheitskontrollar som skal handtere problema. Spesielt er dette tydeleg med hensyn til persistent XSS, då Fortify ikkje klarte å påvise kvar skadeleg data blei introdusert til systemet.

Det gjenstår framleis ei vurdering av problema, og korleis dei framtrer i applikasjonane, samt kor alvorlege problema er. Ei slik vurdering blir gitt i neste kapittel, og det blir også demonstrert korleis ein kan utnytte problema som har blitt påvist til no.

4

4 Sikkerheitsproblem – vurdering

Dette kapittelet tek nærmare for seg dei problema som vart oppdaga i sikkerheitsanalyser og sett dei i samanheng med *JAFU*-systema. Kvifor er desse problema så alvorlege, og korleis kan ein angriper utnytte dei? Det vil bli gitt ein demonstrasjon av dei viktigaste problema, og ein gjennomgang av angrepa som kan utførast. I tillegg vil alvorlegheita av problema bli vurdert nærmare med utgangspunkt i arkitekturen, brukarrollene og mekanismane til JAFU-systema.

Vurderinga er her basert på den vurderingsmetoden som har blitt brukt i [22] (figur 22). Denne metoden er igjen basert på *OWASP Risk Rating Methodology* i [20].

Threat Agent	Attack Vector	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact
?	Easy	Widespread	Easy	Severe	?
	Average	Common	Average	Moderate	
	Difficult	Uncommon	Difficult	Minor	

Figur 13: Metode for vurdering av sikkerheitsproblem [22]

Basert på denne vurderinga blir problema også rangert. For dette formålet blir skalaen i tabell 10 nytta. Denne er basert på Red Hat Common Vulnerability Scoring System [58] og tilpassa vevapplikasjonane i JAFU. Fokusen her vil ikkje vere på spesifikke sårbarheiter, men heller på generelle problem.

Rangering	***Beskrivelse***
Kritisk	Denne rangeringa gjeld for problem som er trivielle å utnytte og er gjennomførbare av ein uautenisert angriper uten brukarinteraksjon. Dette inneber problem som har store konsekvensar for tilgjenge, integritet og konfidensialitet av systemressursar.
Høg	Dette gjeld problem som er lette å utnytte og som har innverknad på konfidensialitet, integritet og tilgjenge av systemressursar.
Medium	Dette gjeld problem som er utfordrande å utnytte og som har mindre innverknad på konfidensialitet, integritet og tilgjenge av systemressursar.
Lav	Dette er alle andre sikkerheitsproblem. Det inneber sårbarheiter der utnytting er usannsynleg og utfordrande, samt inneber minimale konsekvensar for systemet.

Tabell 10: Skala for rangering av sårbarheiter

4.1 Cross Site Scripting i DPG 2.0 og Webucator 3.0

Eit av dei mest utbredte problema i både DPG 2.0 og Webucator 3.0 er XSS. Denne typen angrep står for brorparten av sårbarheitene i applikasjonane, og er eit resultat av manglande validering og sanitering av inndata, samt manglande enkoding av utdata.

Som tidlegare nemnt er det mogleg å utnytte XSS for å stjele informasjonskapslar. Veven er i utgangspunktet tilstandslaus, og konseptet med ein innloggingssesjon eksisterer ikkje. Nettlesaren til ein brukar sender ein forespurnad om å hente ei nettside, og tenaren svarer. Etter dette gløymer teneren at den aktuelle klienten har henta nettsida [42]. Dette er heilt greit for statiske nettsider, slik formålet med verdensveven var i utgangspunktet. Men etter kvart som nye teknologiar og dynamiske nettsider dukka opp, oppstod også behovet for at tenaren skulle kunne huske klientane sine [42].

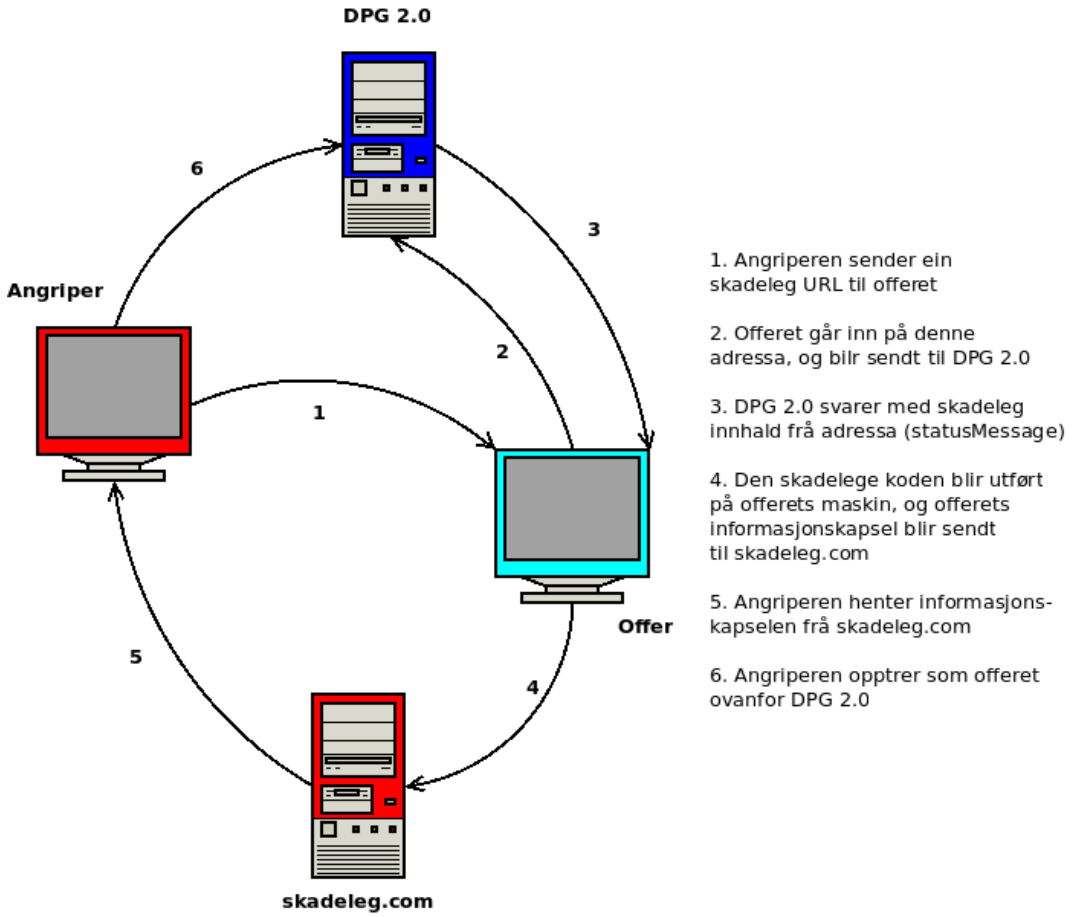
Informasjonskapslar er ein metode for å handtere dette problemet. Konseptet blei opprinneleg utvikla av Netscape [42]. Informasjonskapslar fungerer ved at klienten sender ei spørring om ei nettside, så kan tenaren generere ein informasjonskapsel og legge denne ved svaret som tenaren gir til klienten [42]. Dette er typisk ei fil eller ein streng. Nettlesaren til klienten lagrar så denne informasjonen lokalt, og for å oppretthalde tilstanden mellom klient og tenar kan denne informasjonskapselen nyttast for at klienten kan vise til at han er innlogga.

Dette inneber at ein angripar som har fått tak i nokon andre sin informasjonskapsel kan opptre som den aktuelle brukaren overfor vevapplikasjonen [33]. Når ein ser på korleis brukarrollene er strukturerte blir det fort klart at XSS i DPG 2 er svært alvorleg.

Konsekvensane av eit eventuellt angrep er at ein kan få tilgang til presentasjonar og delar av systemet som ein normalt ikkje skal ha.

- Angriparar uten tilgang til systemet kan oppnå tilgang ved å gjennomføre angrep som utnytter reflekterte XSS sårbarheiter.
- Brukarar med avgrensa tilgang kan overta innloggingane til brukarar med meir tilgang. Til dømes Publisher/Reader => Administrator.

Figur 14 viser korleis reflektert cross-site scripting kan utnyttast for kapring av innloggingssesjonar til ordinære brukarar.



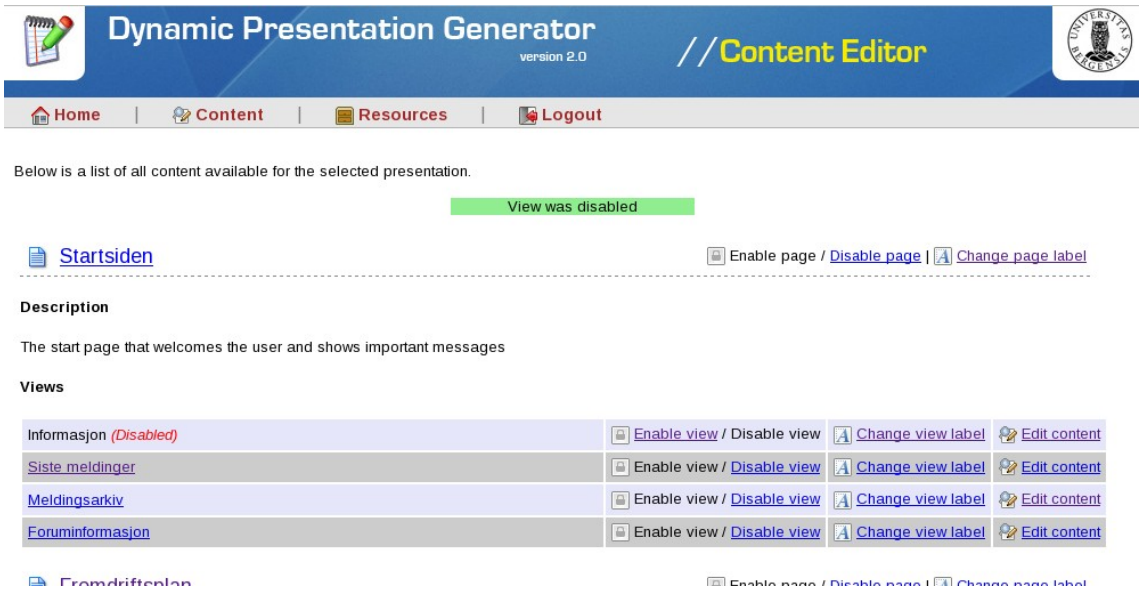
Figur 14: Reflektert XSS angrep for tyveri av informasjonskapslar

På grunn av korleis tilgangskontrollen i systema er satt opp framtrer cross-site scripting på forskjellige måtar i DPG 2.0 og Webucator 3.0, og vil derfor her vurderast kvar for seg basert på type (reflektert/persistent).

Reflektert XSS i DPG 2.0

Delsystemet PCE er sårbart for reflekterte XSS angrep. Dette inneber manipulering av

parameteret `statusMessage` for sidene `listContent.jsp` og `listResources.jsp`. Ein angriper kan altså konstruere ein URL der parameteret er gitt med skadeleg kode. Denne adressa kan så sendast til eit offer, og når offeret så går til adressa blir den skadelege koden køyrt.



Figur 15: DPG 2.0 - Status Message i listContent.jsp

Formålet med parameteret `statusMessage` er å gi tilbakemeldingar til brukarane etter eventuelle endringar. Til dømes når ein publisher gjer endringar i ein presentasjon som å koble ut eit utsnitt eller ei side, vil `statusMessage` bli satt for å gi tilbake melding om suksess eller svikt (Figur 15).

Problemet er at parameteret ikkje blir validert eller enkoda før det blir presentert til brukaren, noko som gjer at det kan inneholde skadeleg kode. Vi kan sjå korleis parameteret blir reflektert i eksempel 1.

```
<% if (request.getParameter("statusMessage") != null) { %>
    <center>
      <span class="statusMessage">${param.statusMessage}</span>
    </center>
<% } %>
```

Eksempel 1: DPG 2.0 - Reflektert XSS - Sårbar kode

Dette kan utnyttast ved at ein vondlynt angriper konstruerer ein URL som inneheld skadeleg kode, og får ein legitim brukar med tilgang til PCE til å gå til denne adressa. Det er også mogleg å skjule innhaldet i adressa ved å bruke verktøy som TinyURL [37], eller ved å enkode den skadelege delen av adressa. På grunn av at det ikkje er nokon tilgang nødvendig for å gjennomføre denne typen angrep, kan det utførast av kven som helst. Alt ein angriper treng er kjennskap til det sårbare parameteret, samt kjennskap til

brukarar med tilgang til PCE, enten dette inneber ein publisher eller administrator.

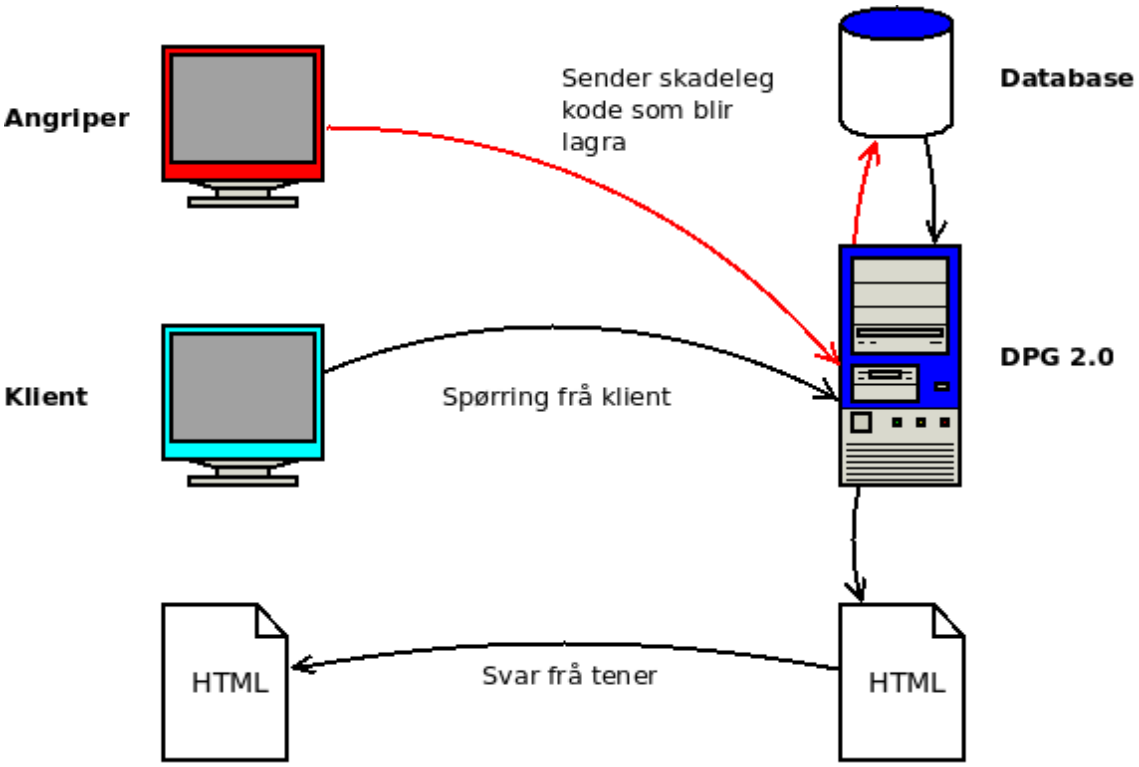
Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FLEIRE	Utnytting LETT	Utbreiing VANLEG	Oppdaging MEDIUM	Innvikning MEDIUM	Innvikning ALVORLEG

Tabell 11: Alvorlegheit - Reflektert XSS i DPG 2.0

På grunn av behovet for brukarinteraksjon og at sårbarheiten er vanskeleg å oppdage for uautentiserte angriparar, får desse problema rangeringa ***høg***. Eventuell utnytting kan medføre alvorlege brudd på konfidensialitet og integritet av systemressursar. Tabell 11 viser vurderinga for reflektert XSS i DPG 2.0.

Persistent XSS i DPG 2.0

Persistent XSS i DPG 2.0 er gjennomførbart mot delsystema PCE og PM. Figur 16 viser korleis eit slikt angrep vil kunne fungere.



Figur 16: Persistent XSS i DPG 2.0

For å gjennomføre persistente XSS angrep i DPG 2.0 må angriparen enten ha privilegier

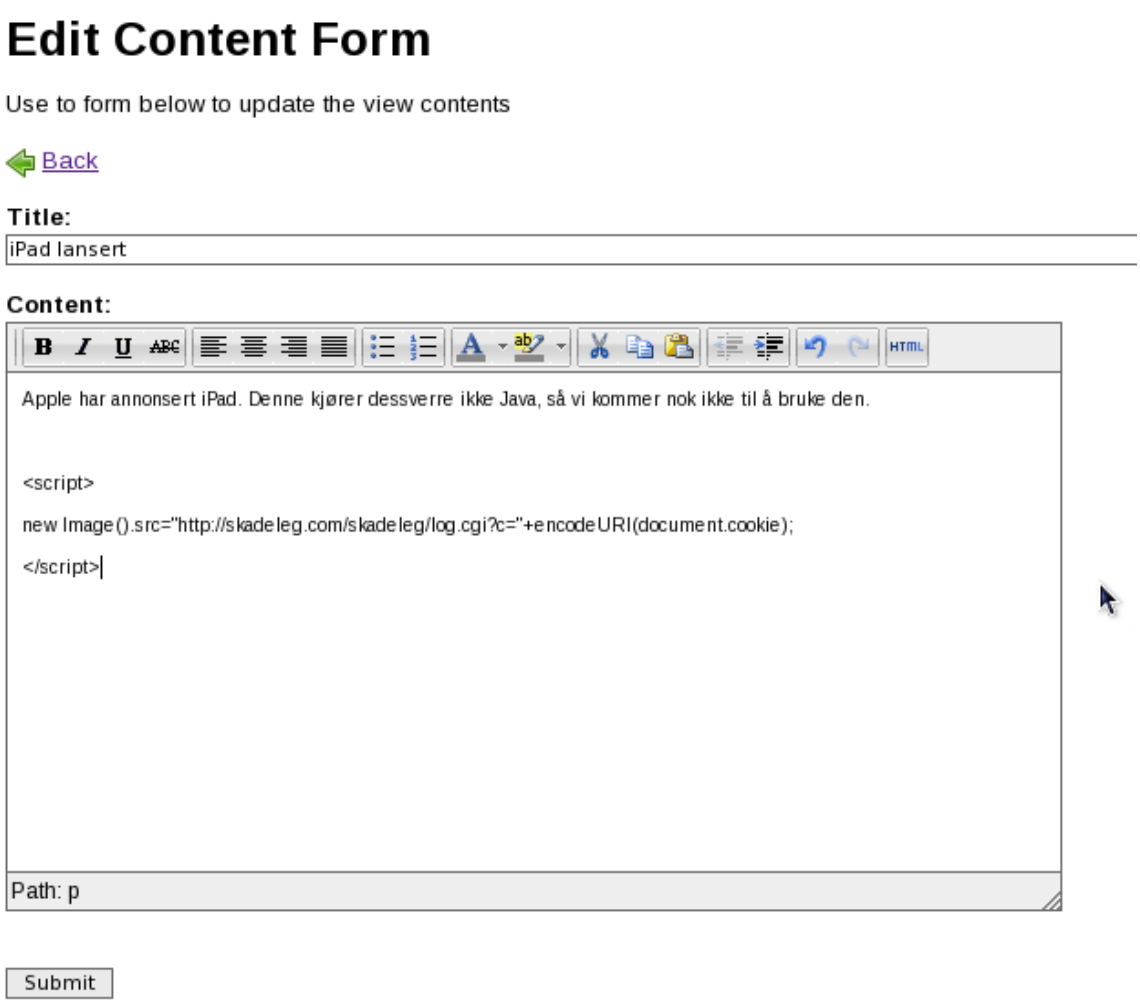
til ein administrator eller publisher. Dette inneber at problemet mest sannsynleg blir utnytta av legitime brukarar, men sett at tilgangskontrollen til PCE eller PM har feila blir problemet utnyttbart for alle.

For PCE er problemet relatert til følgjande funksjonar:

- Edit Content Form
- Change Page Label
- Change View Label

For å kunne nytte nokon av funksjonane over må ein vere publisher for presentasjonen ein gjennomfører endringar på, eller administrator. Denne tilgangen kan ein oppnå gjennom angrep på autentiseringa eller sesjonshandteringa til systemet. Likevel bør DPG 2.0 innehalde fleire publishers enn administratorar, avhengig av antall presentasjonar i systemet, då det er ei rolle med mindre tilgangsmoglegheiter og som ikkje er knytta til heile systemet, men heller til presentasjonar. Dette betyr at sannsynlegheita for interne angrep er større mot Presentation Content Editor.

Utnytting av desse sårbarheitene vil tillate ein angriper å køyre skadelege script mot ei større gruppe brukarar enn det som er tilfellet med reflekterte angrep. Dette er fordi den skadelege koden blir framvist både i PCE og i PV. Altså vil dette angrepet kunne påvirke både publishers, readers og administratorar. Eit eksempel kan vi sjå i figur 17.



Figur 17: Persistent XSS i DPG 2.0, Presentation Content Editor

Inndata til Edit Content Form består av enkel HTML kode frå ein publisher slik at det er mogleg å formatere innhaldet i entitetar for å gjere lesinga enklare. Altså må denne funksjonen ha moglegheit til å ta imot HTML-kode. Change Page Label og Change View Label bør kun ta imot enkle strengar, sidan dette er titler på sider og utsnitt.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting LETT	Utbreiing HØG	Oppdaging LETT	Innvikning MEDIUM	Innvikning ALVORLEG

Tabell 12: Alvorlegheit – Persistent XSS i DPG 2.0 (PCE)

Tabell 12 viser vurderinga for persistent XSS i DPG 2.0 (PCE). For å kunne utnytte persistent XSS i PCE må ein brukar vere autentisert som enten administrator eller publisher. Denne typen problem krev brukarinteraksjon, men innverknaden på konfidensialitet og integritet av systemressursar er relativt alvorleg. Derfor får desse

problema rangeringa ***høg***.

For PM er problemet relatert til følgande funksjonar:

- Setting og endring av namn på presentasjonar
- Setting og endring av beskrivelse på presentasjonar
- Overstyring av sidemalar (HTML)
- Overstyring av transformasjonar (XSLT)
- Overstyring av stilark (CSS)

Persistente XSS angrep mot dette delsystemet er kun gjennomførbart av administratorar. Likevel er dette alvorlege problem då eventuelle sikkerheitsbrot vil kunne påvirke alle brukarane i heile systemet.

Problemet med overstyring av sidemalar, transformasjonar og stilark er vanskeleg å handtere, fordi ein forventer å kunne ta imot rik HTML, XSLT og CSS-kode. Dette inneber blant anna at sidemalane skal kunne innehalde JavaScript som JQuery og liknande, noko som gjer tradisjonell validering og sanitering av inndata til ei utfordring i dette tilfellet. Avgrensing av moglege endringar ein administrator kan gjennomføre i denne delen av systemet vil medføre at nødvendige arbeidsoppgåver blir umoglege. Eit argument for å bevare denne funksjonaliteten i si noverande form er at ein administrator skal ha moglegheit til å gjennomføre denne type endringar.

For å sikre delsystemet mot angrep, samstundes som ein lar administratorar gjennomføre arbeidsoppgåvene sine, bør autentisering og sesjonshandtering sikrast for å forhindre at uvedkommande får tilgang til dette kritiske delsystemet. I tilfeller der ein ikkje treng å ta imot HTML, XSLT og CSS-kode frå administratorane bør ein gjennomføre validering av inndata.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FÅ	Utnytting LETT	Utbreiing HØG	Oppdaging LETT	Innvikning MEDIUM	Innvikning LAV

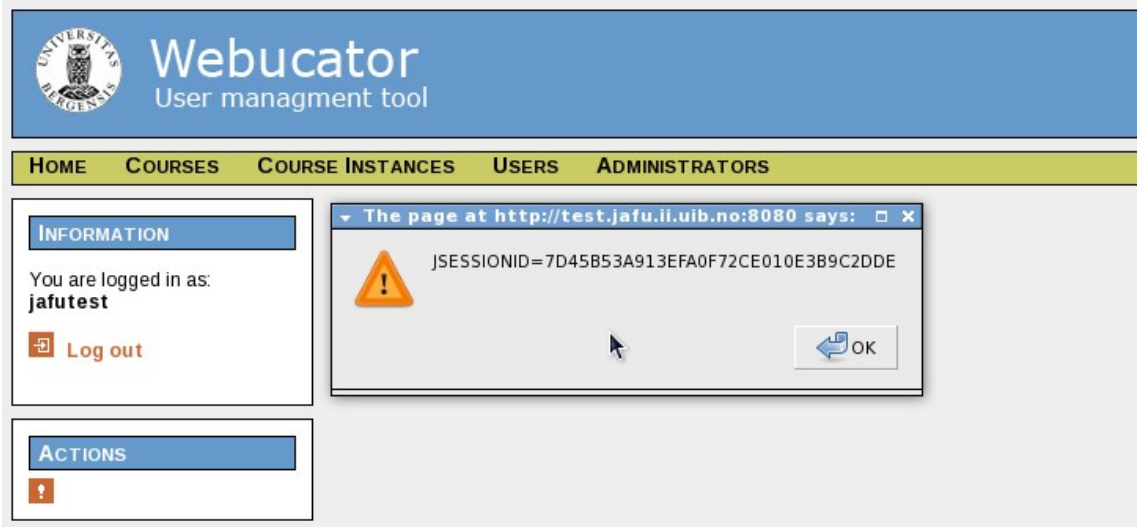
Tabell 13: Alvorlegheit – Persistent XSS i DPG 2.0 (PM)

Tabell 13 viser vurderinga av persistent XSS i DPG 2.0 (PM) Sidan eit mindretall av brukarane i DPG 2.0 skal vere administratorar, altså brukarar som allereie har full tilgang til heile systemet blir dette problemet vurdert til ***medium***.

Reflektert XSS i Webucator 3.0

Reflektert XSS blei og påvist i Webucator 3.0. Alvorlegheita her er mindre enn i DPG 2.0, på grunn av at systemet ikkje er tilgjengeleg for andre enn administratorar. Kunnskap om sårbarheiten er derfor lite truleg for eksterne angriparar. Likevel kan ein ikkje vere garantert at kjennskap til sårbarheitene ikkje er uoppnåeleg, og problemet må handterast.

Eit eksempel på reflektert XSS kan vi sjå i figur 18. I dette tilfellet har parameteret `courseInstanceId` blitt manipulert for sida `addUserForm.jsp`. Her blir eit enkelt script brukt som køyrer funksjonen `alert(document.cookie)`.



Figur 18: Reflektert XSS i Webucator 3.0.

På grunn av at Webucator 3.0 ikkje vil vere tilgjengeleg for angriparar er utnytting av denne sårbarheiten lite truleg. Reflektert XSS i Webucator 3.0 krev brukarinteraksjon og er utfordrande å gjennomføre. Eventuelle angrep vil derimot kunne ha innvirknad på konfidensialitet og integritet for ressursar i Webucator 3.0. Tabell 14 viser vurderinga for reflektert XSS i Webucator 3.0 Med utgangspunkt i desse faktorane blir dette problemet vurdert til **medium**.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting LETT	Utbreiing HØG	Oppdaging MEDIUM	Innvikning MEDIUM	Innvikning LAV

Tabell 14: Alvorlegheit – Reflektert XSS i Webucator 3.0

Persistent XSS i Webucator 3.0

Persistent XSS i Webucator 3.0 eksisterer i to former. Først og fremst er det mogleg å gi XSS til applikasjonen gjennom HTML-skjema i Webucator 3.0. For å utnytte denne sårbarheiten må ein angriper ha administratortilgang til Webucator 3.0. Desse problema vil derfor vere vanskelege å utnytte, og utgjer liten trussel frå eksterne angriparar.

Eit større problem er XSS via eit Excel-skjema som inneheld brukardata som fornamn, etternamn og e-post. Dette Excel-skjema blir levert til JAFU frå SEVU og inneheld informasjonen om studentar i fjernundervisninga. Sett at ein angriper sender eit manipulert Excel-skjema til JAFU og får det til å sjå ut som om det kjem frå SEVU vil eit angrep kunne bli gjennomført uten å ha tilgang til Webucator 3.0.

Dette inneber likevel ein del problem.

- Først og fremst må angriparen ha kjenskap til sårbarheiten. Dette er lite truleg, sidan systemet i utgangspunktet er utilgjengeleg for uvedkommande.
- Ein må også ha kunnskap om enkelte avgrensingar i data som kan gis til Webucator 3.0. Databasen har ei avgrensing på 45 teikn per datafelt, noko som gjer at script og namn må ta opp svært lite plass.
- Angriparen må også ha kjennskap til strukturen i Excel-skjemaet.

Persistent XSS kan likevel gjennomførast av tilsette i JAFU, og må derfor handterast med input validering og output encoding. Persistente XSS-angrep mot Webucator 3.0 vil vere svært alvorlege då dette vil påvirke alle brukarane i system som er tilknytta Webucator 3.0.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FÅ	Utnytting MEDIUM	Utbreiing HØG	Oppdaging MEDIUM	Innvikning MEDIUM	Innvikning MEDIUM

Tabell 15: Alvorlegheit – Persistent XSS i Webucator 3.0

Persistent XSS i Webucator 3.0 er overførbart til DPG 2.0. Data om kurs, kursinstansar og brukarar blir sendt til DPG 2.0 og presentert der. På denne måten vil eventuelle angrep mot Webucator 3.0 også kunne bli gjennomført mot DPG 2.0. Sjølv om persistent XSS mot Webucator 3.0 har krav til kjennskap om sårbare problem i systemet er det ikkje nødvendig med tilgang til sjølve systemet dersom ein til dømes nyttar ei manipulert Excel-fil. Innvirknaden er stor, sidan eit slikt angrep påvirker både DPG 2.0 og Webucator 3.0. Vurderinga ser vi i tabell 15. Problemet blir med utgangspunkt i dette rangert som ***medium.***

4.2 Loggforfalsking i DPG 2.0

DPG 2.0 er sårbar for loggforfalsking i alle delsystema med unntak av Lobby. I JAFU blir det ikkje brukt nokre verktøy for automatisk evaluering av loggane i dag. Loggforfalskingsangrep mot DPG 2.0 kan derfor vere samansatt av to typar:

- Angrep hvis formål er å forstyrre hendelsesforløp ved å gjere loggen vanskeleg å lese.
- Angrep hvis formål er å gi inntrykk av forskjellig hendelsesforløp frå det som faktisk har skjedd.

Den første typen angrep kan for eksempel innebere å sende store mengder med tilfeldig data i sårbare parameter til DPG 2.0. Den andre typen angrep krev derimot kjennskap til korleis reelle beskjedar i loggen kan sjå ut, og vil vere vanskelegare å gjennomføre.

Sett for eksempel at ein angriper ynskjer å få det til å sjå ut som om at han har logga ut, slik at eventuelle handlingar ikkje blir tilknytta han, vil han kunne manipulere informasjon som blir gitt til DPG 2.0 på følgande måte:

```
http://test.jafu.ii.uib.no:8080/dpg2.1/pv/presentation.html?
pid=inf101h08'%0aINFO:+User+'badguy'+logged+off.%0a
```

Dette vil bli gjengitt i loggen. Karakteren %0a referer til eit linjebrot, noko som lar den resterande delen av strengen komme på ei ny linje på følgande måte:

```
DEBUG [http-8080-1] (AbstractDpgAuthenticationFilter.java:49) –
Presentation id is set to 'inf101h08'
INFO: User 'badguy' logged off.
```

Tabell 16 viser vurderinga for loggforfalsking i DPG 2.0. Sannsynlegheit for angrep er relativt lav, og på grunn av manglande aktivitetslogging (i dag blir hovudsakeleg loggane nytta for debugging) er det lite ein angriper kan oppnå. Angriparane må også ha kjennskap til korleis loggane er strukturererte, noko som gjer at sannsynlegheita for å oppdage dette problemet er relativt lav. Derfor blir dette problemet vurdert med rangeringa ***lav***.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting LETT	Utbreiing HØG	Oppdaging LAV	Innvikning LAV	Innvikning LAV

Tabell 16: Alvorlegheit – Loggforfalsking i DPG 2.0

Mangelfull logging er eit problem i seg sjølv, og vil bli diskutert nærmare i kapittel

7.4.2.

4.3 Tenestenekting mot DPG 2.0

Tenestenekting i DPG 2.0 er gjennomførbart grunna ein feil i korleis applikasjonen handterer presentasjonar i delsystemet Lobby. Når ein brukar er innlogga vil han få ei oversikt over dei presentasjonane han har tilgang til. Dette inneber blant anna at DPG 2.0 hentar fram spesifikasjonen for kvar presentasjon. Denne spesifikasjonen ligg i

```
dpg/presentations/presentasjonsid
```

Dersom applikasjonen kjem over ei mappe som ikkje har den nødvendige spesifikasjonen vil den kaste eit unntak i staden for å hente lista over presentasjonar. Dette vil gjere systemet utilgjengeleg. Problemet kan utnyttast av ein angriper gjennom delsystemet PCE. Ressurshandteringa for presentasjonar tillet administratorar og publishers å opprette nye mapper. Applikasjonen tek imot eit mappenamn, og opprettar så denne mappa. Til dømes dersom nokon ynskjer å opprette mappa *Løysingsforslag* blir denne oppretta som

```
dpg/presentations/presentasjonsid/resources/Løysingsforslag
```

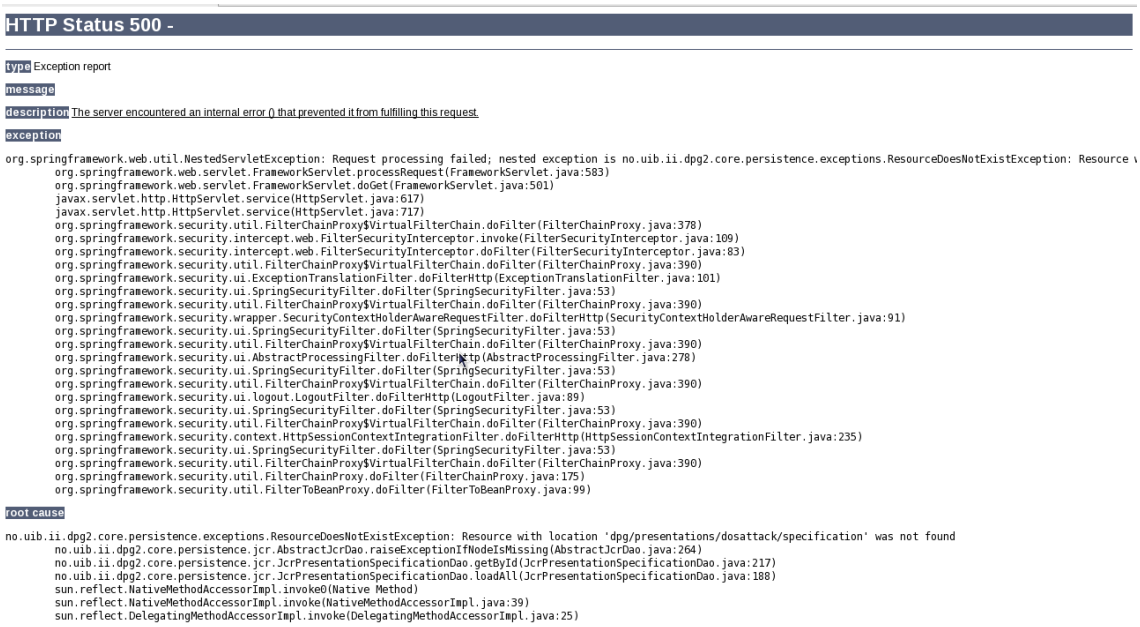
Ved å legge til kommandoar for å traversere i katalogstrukturen i namnet på mappa som skal opprettast vil den aktuelle mappa blir lagt lenger oppe. Eit slikt tilfelle ser vi i figur 19.

Figur 19: Tenestenekting ved katalogtraversering i DPG 2.0

Dette vil medføre at mappa blir lagt i

```
dpg/presentations/dosattack
```

Denne mappa blir så tolka som ein presentasjon av DPG 2.0. Sidan mappa ikkje inneheld nokon spesifikasjon klarer ikkje lobbyen å produsere ei liste av persentasjonar. I staden blir eit unntak kasta (figur 20), og oversikta over presentasjonar blir utilgjengeleg.



Figur 20: DPG 2.0 - Tenestenektingsangrep

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting LETT	Utbreiing LAV	Oppdaging MEDIUM	Innvikning MEDIUM	Innvikning MEDIUM

Tabell 17: Alvorlegheit – Tenestenekting ved katalogtraversering i DPG 2.0

For å utnytte denne feilen er det nødvendig for angriparen å ha tilgang som publisher eller administrator, sidan readers ikkje har lov til å opprette nye mapper. Like fullt er det mogleg å oppnå slik tilgang ved å utnytte andre feil i systemet, som til dømes reflektert XSS. Dette problemet blir vurdert til ***medium***. Vurderinga for dette problemet ser vi i tabell 17.

4.4 Brot på tilgangskontroll i DPG 2.0

Eit anna resultat av katalogtraverseringsproblemet er at det blir mogleg å omgå tilgangskontrollen som er knytt til presentasjonar. Utnytting av katalogtraversering i ressurshandteirngssystemet legg til rette for at publishers kan få tilgang til filer og mapper i presentasjonar dei ikkje skal ha tilgang til. For eksempel kan *publisher A* ha tilgang til presentasjon *444*, men han ynskjer å sjå kva filer som tilhøyrer presentasjon *555*.

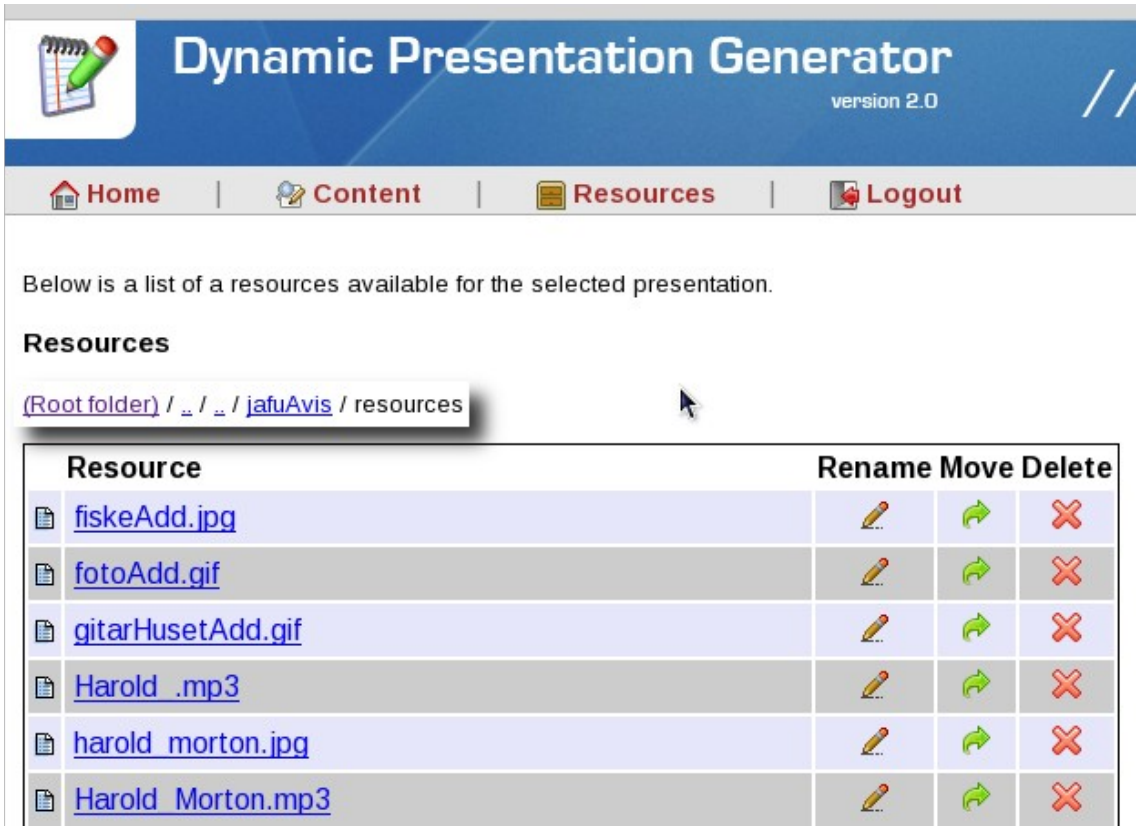
Readers kan også få tilgang til ressursar som tilhøyrer andre persentasjonar, men vil ikkje ha moglegheit til å gjere endringar.

Ved å manipulere *path* (eksempel 2) parameteret i ressurshandteringa knytt til sin eigen presentasjon (*444*) kan *publisher A* få ei oversikt over alle filene og mappene i presentasjon *555*.

```
http://test.jafu.ii.uib.no:8080/dpg2/pce/resources/listResources.html
?pid=444&path=../../555/resources
```

Eksempel 2: DPG 2.0 - Katalogtraversering i path-parameteret

I figur 21 ser vi korleis ein publisher i presentasjonen *inf100h08* har fått opp alle filene i presentasjonen *jafuAvis*. Her er det mogleg å endre namn, slette, flytte og laste opp nye filer i ein presentasjon brukaren ellers ikkje skal ha tilgang til.



Figur 21: DPG 2.0 - PCE - Brot på tilgangskontroll

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting LETT	Utbreiing VANLEG	Oppdaging LETT	Innvikning MEDIUM	Innvikning MEDIUM

Tabell 18: Alvorlegheit – Brot på tilgangskontroll ved katalogtraversering i DPG 2.0

Her er det mogleg å ikkje berre sjå, og laste ned filene til andre presentasjonar, men også endre namn, slette og flytte desse, samt opprette nye mapper dersom ein har rolla *publisher*. Der er ingen krav om brukarinteraksjon, og slike angrep kan føre til utilgjengelege ressursar og brot på konfidensialitet og integritet knytta til tilgangskontrollen for presentasjonar. Kun filer i ressusrhandteringssystemet er påvirka, og ingen *entitetar.* Med utgangspunkt i dette blir dette problemet vurdert til kategorien ***medium***. Vurderinga er i tabell 18.

4.5 Problem knytta til passord

Kombinasjon av brukarnamn og passord er ein svært vanleg autentiseringsmetode i bruk idag. Slik autentisering har ei rekke fordelar. Det er lett å implementere. Det er ikkje dyrt (krev ikkje sofistikert maskinvare), og det er enkelt å bruke. Dei fleste brukarane er kjende og komfortable med slike metodar for autentisering. Likevel er det og knytta ulemper til denne typen autentisering. Passord kan ofte vere lette å gjette, dei er sårbare under overføring, og brukarane gløymer ofte passord og skriv dei ned for å huske dei [35].

Styrken knytt til passord er avhengig av tre faktorar. Lengde, kompleksitet og uforutsigbarheit. Passord bør vere lange nok til at det er mogleg med eit stort antall kombinasjonar. Dei bør vere komplekse i at dei inneheld mange forskjellige typar teikn. I tillegg bør dei vere uforutsigbare for å unngå at nokon kan gjette dei.

Passorda til brukarane i DPG 2.0 oppstår på to forskjellige måtar. For administratorar blir passorda satt manuellt, medan for readers og publishers blir passorda generert automatisk.

4.5.1 Usikker generering av brukarpassord

Nøkkelen for å sikre tilfeldig generering av sensitiv data er å unngå forutsigbarheit. Det er viktig å passe på at uvedkommande ikkje kan forutsjå kva data som blir generert, då dette drastisk svekkar sikkerheita knytt til data som er generert.

Passord for vanlege readers og publishers er laga tilfeldig, som vi ser i eksempel 3. Alle desse passorda er konstruerte med utgangspunkt ein spesiell struktur. Alle passord er 8 teikn lange, og dei tre første teikna i passordet er alltid tall. Deretter er det annakvar vokalar og konsonantar. Altså, er følgande struktur brukt:

X X X Y Z Y Z Y

Der X er tall, Y er vokalar og Z er konsonantar. I tillegg er ikkje alle konsonantane i alfabetet i bruk i passordgenereringa. På grunn av den svake og forutsigbare strukturen

til passorda skal det relativt få passord til før algoritma genererer samme passordet to eller fleire gongar. Etter kun 10 000 passord har algoritma generert samme streng fleire gongar.

```
public String generatePassword() {
    Random r = new Random();
    String[] vowels = { "a", "e", "i", "o", "u", "y" };
    String[] consonants = {"b", "d", "f", "g", "j", "k", "l",
        "m", "n", "p", "r", "s", "t", "v" };
    String password = "";

    /* Password starts with 3 numbers */
    for (int j = 0; j < 3; j++) {
        password += "" + r.nextInt(10);
    }

    /*
     * Then continues with 5 letters,
     * with shifting vowels/consonants for easier reading
     */
    for (int j = 0; j < 5; j++) {
    String nextLetter = null;
        if (j % 2 == 0)
            nextLetter = vowels[r.nextInt(vowels.length)];
        else
            nextLetter =
                consonants[r.nextInt(consonants.length)];
            password += "" + nextLetter;
    }
    return password;
}
```

Eksempel 3: Webucator 3.0 – Usikker passordgenerering

Dette medfører at sjølv om passordet blir lagra sikkert (det vil seie med einvegskryptering og salting, som vil bli diskutert nærmare seinare), så kan ein avdekke passorda som er krypterte på svært kort tid. Ved å køyre metoden i eksempel 3, vil ein på kort tid finne tilbake til det passordet som blei generert i utgangspunktet. I eit forsøk vart hundre passord genererte og enkoda med *SHA-1* (Secure Hash Algorithm). Ved å køyre metoden i eksempel 3 fleire gongar vart krypterte passord avdekka med ei snitt-tid på 112 sekund.

Det interessante med dette er begrunninga for strukturen som er gitt som kommentarar i koden.

> *«Then continues with 5 letters with shifting vowels/consonants for easier reading.»*

I dette tilfellet er passordet allereie generert automatisk, og leseligheita til passordet er

allereie lav. Dersom formålet med å auke leseligheita til passordet er å redusere sannsynlegheita for at brukarane skriv det ned, er det då merkeleg at brukarane ikkje får velje passordet sjølve. Her bør altså ikkje leseligheita ha presedens over styrken, sidan passordet allereie er vanskeleg å huske.

For generering av tilfeldige passord, er det også viktig at slumptallsgeneratoren er uforutsigbar. Det er ikkje tilfellet i Webucator 3. I staden for å bruke den sikre slumptallsgeneratoren i Java, `java.security.SecureRandom`, så har valet falt på `java.util.Random`. Dette er eit risikabelt val. Standardkonstruktøren til java.util.Random tar i bruk systemtid i millisekund som utgangspunkt for slumptallsgenerering [67]. Ved å synkronisere lokal tid med tida på tenaren er det derfor mogleg å redusere det moglege antallet med verdiar som er nytta som utgangspunkt for slumptallsgenerasjonen [34].

Som sagt er det her eit betre alternativ å ta ibruk java.security.SecureRandom. Standard for java.security.SecureRandom er å ta utgangspunkt i operativsystemets eigne sikre funksjonar for eit tilfeldig utgangspunkt eller *seed* [34]. I Linux er dette til dømes /dev/urandom.

Moglege angriparar for dette problemet inneber brukarar i DPG 2.0 eller administratorar i Webucator 3.0. Vurderinga knytta til dette er presentert i tabell 19.

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FÅ	Utnytting VANSKELEG	Utbreiing LAV	Oppdaging VANSKELEG	Innvikning ALVORLEG	Innvikning ALVORLEG

Tabell 19: Alvorlegheit – Svak passordgenerering i Webucator 3.0

Desse angrepa har likevel ein del avgrensningar. Først og fremst må algoritma vere kjent for ein angriper for å kunne utnytte dette. For det andre må strukturen vere kjent, til dømes kva konsonantar som inngår i passordet. Ein del av denne informasjonen kan ein angriper finne med ei stor nok mengde med passord. Med utgangspunkt i dette blir alvorlegheita i dette tilfellet vurdert til ***medium***.

Sjølv om sannsynlegheita for et angrep her er lav bør problema handterast, og sterkare passord må genererast. Strukturen bør forkastast, og rekkefølge, samt antall av vokalar, konsonantar og tall bør vere tilfeldig nok til at passorda blir uforutsigbare. I kapittel 5 blir ESAPIs funksjon for automatisk generering av passord vurdert som eit alternativ.

4.5.2 Manglande reglar for passordstyrke

I motsetning til vanlege brukarar har administratorar moglegheit til å velje sine eigne

passord. Dette kan vere både positivt og negativt. Administratorane har moglegheit til å velje passord som er vanskelege å gjette samt det vil vere enklare for dei å huske passord dei sjølv har valt. Administratorane kan altså velje så sterke passord dei ynskjer så lenge passorda er samansatt av 45 teikn eller mindre, som er avgrensninga satt i databasen.

Men Webucator 3.0 har ingen reglar for minimal styrke på passorda til DPG 2.0 administratorar. Med andre ord kan eit passord vere minimum eit teikn langt, og det er ingen krav til kva teikn som inngår i passordet.

- Det eksisterer ikkje krav om variasjon av store og små bokstavar i passorda.
- Det eksisterer ikkje krav om minimum eller maximum lengde på passorda ein vel utover avgrensningar i databasen.
- Det eksisterer ikkje krav om at tall skal vere inkluderte i passordet.
- Det eksisterer ikkje krav om at spesielle teikn (!#%& etc.) skal vere inkludert i passordet.

Alvorlegheita her blir satt opp mot autentisering i kapittel 4.7.

4.6 Usikker kommunikasjon og lagring

Her blir det presentert manglar relatert til usikker lagring av sensitiv data, og kommunikasjon som ikkje er kryptert.

4.6.1 Usikker lagring av passord

Mange system har behov for å lagre sensitiv data. Dette er også tilfellet med Webucator 3.0. Systemet er bygd for å handtere brukarane til DPG 2.0, og må derfor handtere data som brukarnamn, passord, fornamn, etternamn, e-post adresser og liknande. Viktigheita med å lagre sensitiv informasjon sikkert er sjølvforklarande. Til tross for dette lagrar ikkje Webucator 3.0 sensitiv data på ein trygg og forsvarleg måte.

Alt som blir lagt i Webucator 3.0 sin SQL database er i klartekst. Dette inneber at dersom ein angripar skulle få tilgang til denne databasen vil han finne passord til alle brukarane, både readers og publishers (eksempel 4), samt administratorar. Fullt namn på alle brukarane, og e-post adressa dei er registrerte med blir også eksponerte. Det er derfor viktig å sikre sensitiv data. Spesielt passorda har prioritet her, sidan dei kan nyttast for å få tilgang til DPG 2.0. Til dette formålet er det vanleg å nytte einvegskryptering eller *hashing*.

```
webucator=# select * from users;
id | username |  passwd  | firstname | lastname | email        | enabled
---+----------+----------+-----------+----------+--------------+--------
18 | oha001   | 452atoky | Ole       | Hansen   | abc@def.ghi1 | t
19 | fni001   | 087yrefe | Fred      | Nilsen   | abc@def.ghi2 | t
20 | kol001   | 418opena | Kristine  | Olsen    | abc@def.ghi3 | t
21 | ono001   | 863egepe | Ola       | Nordmann | abc@def.ghi4 | t
22 | kku001   | 067ydome | Kurt      | Kurtson  | abc@def.ghi5 | t
23 | gbu001   | 675ovovu | George    | Bush     | abc@def.ghi6 | t
(6 rows)
```

Eksempel 4: Tabell over brukarar

Administratorar i Webucator 3.0 (altså rolla som er unik for Webucator 3.0) blir lagra på tryggare vis. Brukardata blir her lagra i ei XML-fil (eksempel) og passord blir hasha med *SHA-1* (Secure Hash Algorithm 1). Dette kan virke sikkert, men det er fullstendig avhengig av lengda og kompleksiteten til det aktuelle passordet [36]. Bruk av førehandsproduserte oppslagstabellar kan ofte finne klarteksten av korte strengar. Såkalla *rainbow tables* er fritt tilgjengelege på internett for ei rekke algoritmar som blant anna *SHA-1*, *MD5* (Message Digest 5), og *NTLM* (NT LAN Manager) [36].

For å gjere lagringa av passorda så sikkert som mogleg bør ein ikkje berre ta ibruk einvegskryptering av passorda, men også *salte* dei. Dette inneber å appendere ein tilfeldig verdi til strengen som skal hashast, noko som gjer at kompleksiteten og lengda på strengen aukar. Saltet blir så lagra og gjenbruk for kvar gong ein skal verifisere strengen ved hashing [31].

Tabell 20 viser vurderinga av dette problemet. Angriparar i dette tilfellet kan vere alt frå brukarar i systemet til profesjonelle kriminelle. Webucator 3.0 er ikkje sårbart for SQL innsprøyting, noko som gjer det vanskeleg for ein angriper å få tilgang til ukryptert data. Uten den nødvendige tilgangen er det vanskeleg for ein angriper å vite at data er lagra usikkert. Rangeringa for dette problemet er med utgangspunkt i dette ***medium*.**

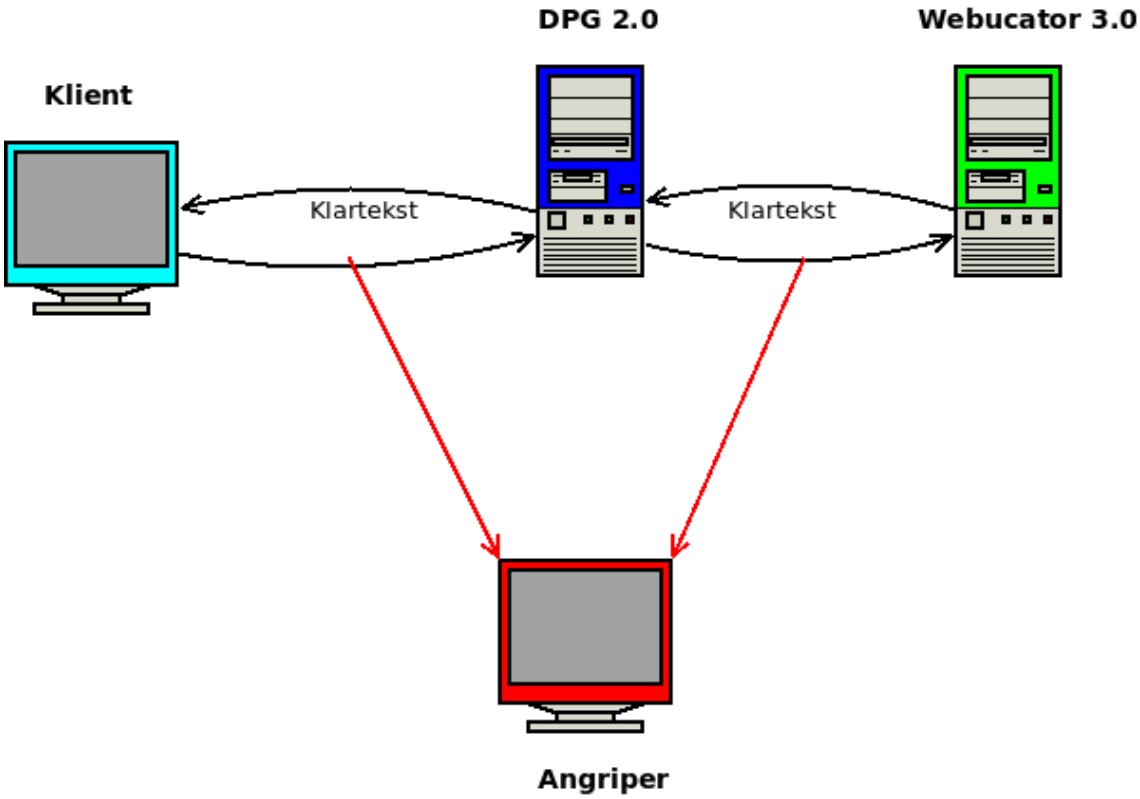
Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FÅ	Utnytting VANSKELEG	Utbreiing HØG	Oppdaging VANSKELEG	Innvikning ALVORLEG	Innvikning ALVORLEG

Tabell 20: Alvorlegheit – Usikker lagring i Webucator 3.0

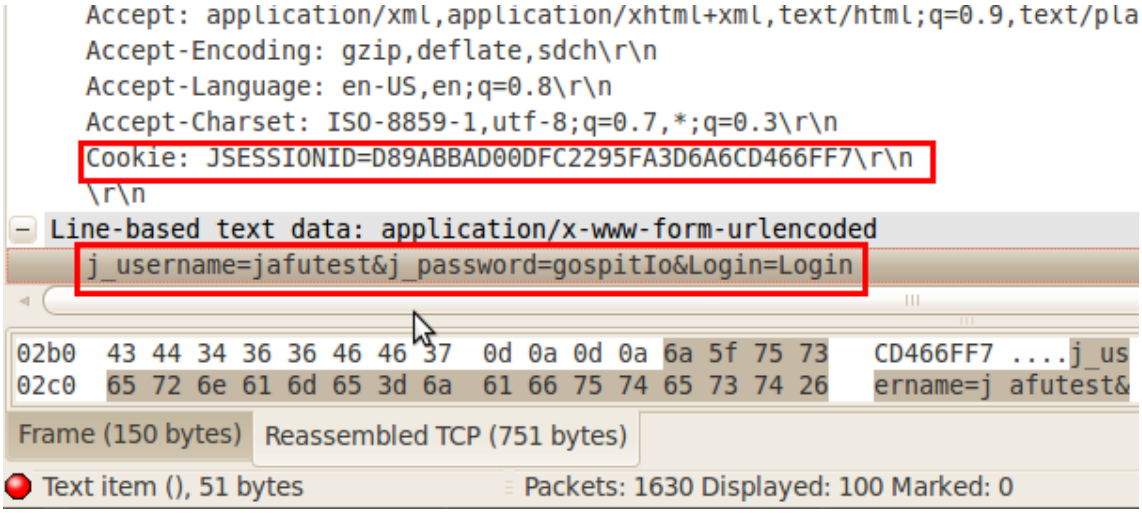
Dette gjer at oppslagstabellar blir ineffektive, sidan det ville ta alt for lang tid å bygge tabellar som er store nok til å inkludere salt. Salting blir vidare diskutert i kapittel 5.

4.6.2 Usikre kommunikasjonskanalar

For innlogging blir ikkje HTTPS nytta. Dette medfører at all innloggingsinformasjon både mellom klient og DPG 2.0, samt mellom DPG 2.0 og Webucator 3.0 skjer i klartekst. Dette er problematisk, fordi det opner for avlytting av informasjon som brukarnamn og passord i det slik informasjon blir sendt over nettverket (figur 22). Figur 23 viser passord og informasjonskapsel i klartekst slik det kjem fram i verktøyet Wireshark [48].

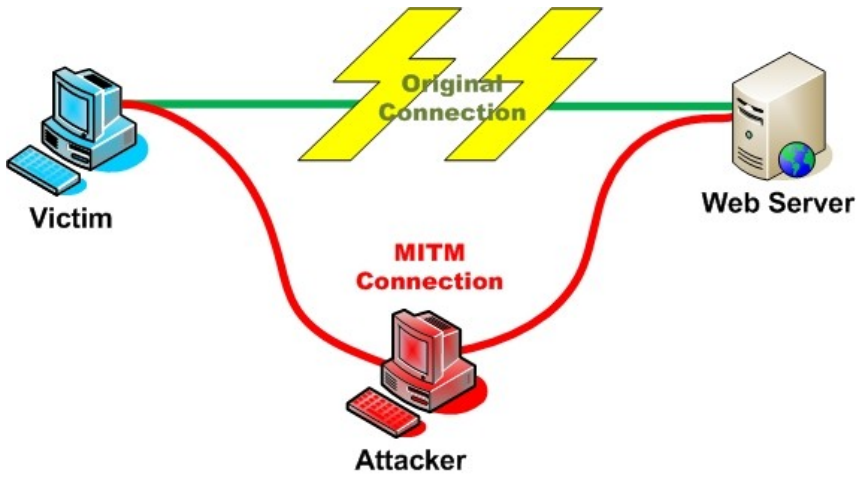


Figur 22: Avlytting av Klient, DPG 2.0 og Webucator 3.0



Figur 23: Sensitiv data avlest med Wireshark

I tillegg opnar usikre kommunikasjonskanalar såkalla *mann-i-midten* angrep (figur 24). I eit slikt tilfelle vil ein angriper utgi seg for å vere ein av partane i kommunikasjonen og på den måten lese, endre og legge til informasjon i overføringa [52]. Dersom ein til dømes gjennomfører eit MITM angrep der ein plasserer seg mellom DPG 2.0 og Webucator 3.0 kan ein fortelle DPG 2.0 at alle innloggingane er korrekte, og at brukaren som forsøker å logge inn alltid er administrator, uavhengig av om denne brukaren faktisk eksisterer eller ikkje.



Figur 24: Mann-i-midten angrep [52]

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar MIDDELS	Utnytting MEDIUM	Utbreiing HØG	Oppdaging LETT	Innvikning MIDDELS	Innvikning ALVORLEG

Tabell 21: Alvorlegheit – Usikre kommunikasjonskanalar i JAFU-systema

Avlytting av trafikk er ikkje spesielt vanskeleg å gjennomføre, og har store konsekvensar for integritet og konfidensialitet i JAFU-systema. Mann-i-midten angrep er litt vanskelegare å gjennomføre, men kan til gjengjeld utførast uten at ordinære brukarar er involverte (mellom DPG 2.0 og Webucator 3.0). Dette problemet får derfor rangeringa ***høg*** til ***kritisk***. Tabell 21 viser vurderinga tilknytta dette problemet i JAFU-systema.

4.7 Autentisering

På grunn av problema relatert til administratorpassord kan det vere relativt enkelt å gjennomføre brute-force autentisering mot DPG 2.0 og Webucator 3.0. Svake passord kan medføre at gjentatte forsøk på innlogging blir vellukka, sidan moglegheitene for å oppdage ein lovleg kombinasjon av brukarnamn og passord er større [33].

Applikasjonane har heller ingen avgrensningar for antall innloggingar det er mogleg å gjennomføre for kvar konto på kort tid. Gjennomføring av brute-force angrep er svært enkelt. Det eksisterer fleire verktøy for å gjennomføre slike angrep, og dei er heller ikkje kompliserte å skrive sjølv.

I eit forsøk på denne typen angrep mot DPG 2.0 vart verktøyet THC Hydra[47] nytta. På rundt 20 minutt, og med ei avgrensa liste med potensielle passord (ordbok-basert angrep), blei passorda til fleire av brukarane oppdaga. Sjølv om testen var avgrensa i omfang ga den raskt resultat (eksempel 5).

```
[8080][www-form] host: 129.177.16.93 login: brsseb password: 123
[8080][www-form] host: 129.177.16.93 login: bjornovei password: pass
[8080][www-form] host: 129.177.16.93 login: torill password: pass
[8080][www-form] host: 129.177.16.93 login: lol password: lol
```

Eksempel 5: Resultat frå brute-force angrep

Trussel-agentar	Angreps-vektorar	Svakheiter		Teknisk innverknad	Forretnings-innverknad
Angriparar FLEIRE	Utnytting MEDIUM	Utbreiing HØG	Oppdaging MIDDELS	Innvikning MIDDELS	Innvikning HØG

Tabell 22: Alvorlegheit – Svak autentisering

Riktignok var angrepet gjennomført mot ein testversjon av systemet der brukarane ikkje nødvendigvis ser behovet for sterke passord. Men systemet bør ikkje legge til rette for at slike angrep skal kunne skje. Problem knytta til autentisering blir derfor vurdert som svært alvorlege, og får rangeringa ***høg***.

4.8 Oppsummering

I dette kapittelet har det blitt presentert ei vurdering av dei problema som har vorte avdekka under sikkerheitsanalysa av DPG 2.0 og Webucator 3.0. Basert på sannsynlegheit for utnytting, og innvirknad på ressursar i systema (integritet, konfidensialitet, tilgjenge) har det blitt gjort ei vurdering av kor alvorlege desse problema er.

Dei mest alvorlege problema er reflektert og persistent XSS i DPG 2.0, svak autentisering og usikre kommunikasjonskanalar. Dette er på grunn av at dei er lette å utnytte og oppdage. I tillegg kan utnytting av desse problema ha store konsekvensar for brukarar og innehavarar av systema.

Middels alvorlege problem inneber tenestenekting og eksponerte ressursar gjennom katalogtraversering, persistent og reflektert XSS i Webucator 3.0 og svak passordgenerering. For desse er enten innverknaden på systema og brukarane litt lavare enn ved til dømes angrep på autentiseringa, eller at utnytting er mindre sannsynleg.

Loggforfalsking er eit problem, men er ikkje vurdert som spesiellt alvorleg. Potensielle angriparar er brukarar i systemet, og kun eit fåtall av dei veit korleis loggane ser ut. I tillegg er det ikkje nytta noko aktivitetslogging i dag.

Det er viktig å påpeike at sjølv om angrepa virkar usannsynlege er det ikkje umogeleg at dei skjer. Sjølv om eksterne angriparar ikkje har dei nødvendige tilgangane for å utnytte problema er det mogleg, og ikkje usannsynleg, at angriparane kan vere interne brukarar [32]

5

5 Implementasjon av løysinger

Dette kapittelet gir ein gjennomgang av løysingar på dei sikkerheitsproblema som hittil er presenterte. Fokusen er tredelt. Først vil verktøyet ESAPI bli vurdert og nytta for å i møtekome krav om sanitering, validering og enkoding av inn- og utdata. ESAPI blir også nytta for å forbetre passordgenerering og verifisering. Den andre delen vil ta for seg Spring Security og kva endringar som gjennomførast for å i møtekome behov for sikker lagring. Autentiseringa vil bli utvida til å inkludere låsing av brukarkontoar etter eit visst antall feila innloggingsforsøk. Til slutt vil det bli vurdert korleis ein kan konfigurere applikasjonane for å forbetre sikkerheita, blant anna gjennom sikring av kommunikasjonskanalane.

5.1 OWASP ESAPI

OWASP (Open Web Application Security Project) er eit internasjonalt prosjekt som har som formål å forbetre sikkerheit og påliteligheit i applikasjonar. Dette blir gjort gjennom å samle informasjon om vanlege sikkkerheitsproblem og utfordringar, samt tilby verktøy og rammeverk for å i møtekome desse problema.

ESAPI (Enterprise Security API) er eit verktøy med open og fri kjeldekode der målet er å gjere det enklare for utviklarar å produsere sikre applikasjonar. Verktøyet er spesielt utvikla for at det skal vere lett å utvide sikkerheitskontrollar i eksisterande applikasjonar for å i møtekome vanlege sikkerheitsbehov som autentisering, tilgangskontroll, kryptering, validering av inndata, enkoding med meir. Verktøyet er tilgjengeleg for Java EE, .NET, ASP, PHP, Coldfusion, Python med meir.

5.1.1 Validering av inndata

Validering av inndata inneber å kontrollere data som kjem frå eksterne kilder mot eit sett med reglar som blant anna definerer korleis data skal vere strukturert, kva moglege teikn inndata kan bestå av og kor lang gitt inndata kan vere. Manglande validering av inndata er ei av årsakene for ei rekke av dei problema som til no har blitt påvist i DPG 2.0 og Webucator 3.0, som blant anna loggforfalsking, cross-site scripting og katalogtraversering. Andre vanlege problem som ofte oppstår som eit resultat av manglande validering av inndata er SQL innsprøyting [29] og buffer overflyt sårbarheiter [19].

Validering av inndata bør gjennomførast med *whitelisting*. Dette betyr at reglane som data blir validert mot fortel korleis data skal vere strukturert, altså kva data som er ynskjeleg [19]. Dette er i motsetning til *blacklisting*. Blacklisting inneber reglar som definerer kva data som *ikkje* er ynskjeleg. Problemet med ein slik framgangsmåte er at settet med data som ikkje er ynskjeleg er uendelig mykje større enn settet med data som er ynskjeleg, og det er vanskeleg å kunne svareliste alle typer skadeleg data [19]. Ein angriper kan utnytte dette ved å enkode inndata på ein slik måte at gitt data ikkje er svartelista, og på den måten omgå valideringa. Ved å kun ta imot data som følgjer den strukturen ein forventar, vil all anna data bli forkasta uansett, og ein treng derfor ikkje å nytte blacklisting.

Validering av inndata i DPG 2.0

Ved å validere namn på sider, utnitt og presentasjonar, samt beskrivelse av presentasjonar kan ein unngå persistent cross-site scripting. I tillegg til at desse felta bør validerast vil dei også bli enkoda når dei blir presenterte for brukarane. Tabell 23 inneheld ei oversikt over dei valideringsreglane som gjeld for denne type inndata. Maksimum lengde i desse tilfella er satt med hensyn til framvising i nettlesaren, av reint estetiske årsaker.

Inndata	**Valideringsreglar**
Namn på sider	Alfanumerisk med mellomrom og punktum. Maksimum lengde på 15 teikn.
Namn på utsnitt	
Namn på presentasjonar	
Beskrivelse av presentasjonar	Alfanumerisk med mellomrom og punktum. Maksimum lengde på 60 teikn.

Tabell 23: Valideringsreglar for namn og beskrivelsar i DPG 2.0

Katalogtraversering i ressurshandteringssystemet til DPG 2.0 er knytt til tenestenekting og brot på tilgangskontrollen i systemet. DPG 2.0 manglar også reglar for filtyper og

maksimum størrelse. For å imøte komme dette blir filer mapper og katalogstiar validerte (tabell 24). Lengde på filnavn blir avgrensa til 20 teikn for at dei skal vere oversiktelege i ressurshandteringssystemet. I tillegg blir maksimum størrelse satt til 100 megabytes. DPG 2.0 er eit innhaldshandteringssystem, og kan dermed ha ei rekke formål. Avhengig av desse formåla kan det vere nødvendig å redusere eller auke godkjent filstørrelse. Lengda på parameter for katalogsti er satt til 65 teikn. Dette er med utgangspunkt i eit mappehierarki på tre mapper. Igjen kan det vere nødvendig å endre dette avhengig av korleis ein nytte DPG 2.0.

Inndata	**Valideringsreglar**
Filer	Alfanumerisk med bindestrek, punktum og understrek. Maksimum lengde på 20 teikn. Avgrensing av lovlege filtypar. Maksimum størrelse på 300 megabytes (som satt av tidlegare utviklarar i `applicationContext-servlet.xml`.)
Mappenamn	Alfanumerisk med understrek. Maksimum lengde på 20 teikn.
Katalogsti	Alfanumerisk med bindestrek, understrek og skråstrek (forover). Maksimum lengde på 65 teikn.

Tabell 24: Valideringsreglar for ressurshandtering i DPG 2.0

For å unngå forfalsking av loggar må også parameter for presentasjonar validerast. Dette inneber ID på presentasjonar, utsnitt og sider. I tillegg var det påvist loggforfalsking gjennom parametera for entitetssti og entiteshandling. Ein entitetssti kan vere temmeleg lang, og maksimum lengde blir satt til 100 på grunn av dette. Entitetshandling kan enten vere *add*, *edit* eller *delete*. Med utgangspunkt i dette blir et sett med valideringsreglar definert (tabell 25).

PresentasjonsID	Alfanumerisk med understrek. Maksimum lengde 15 (identisk lengde med presentasjonsnamn).
UtsnittID	Alfanumerisk. Maksimum lengde 15.
SideID	Alfanumerisk. Maksimum lengde 15.
Entitetssti	Alfanumerisk med understrek. Maksimum lengde på 100 teikn.
Entitetshandling	Mellom a og z. Kun små bokstavar, maksimum lengde på 6.

Tabell 25: Valideringsreglar for presentasjonsparameter i DPG 2.0

ESAPI gjennomfører validering mot to forskjellige konfigurasjonsfiler. Dette er `validation.properties` (eksempel 6) og `ESAPI.properties` (eksempel 7).

```
# Validation-rules for whitelisting input in Webucator 3.0
# ---------------
Validator.EntityPath=^[\\p{Alnum}\\_]{0,1024}$
Validator.Label=^[\\p{Alnum}\\p{Space}]{0,1024}$
Validator.PresID=^[\\p{Alnum}\\_]{0,1024}$
Validator.ViewAndPageID=^[\\p{Alnum}]{0,1024}$
Validator.Description=^[.\\p{Alnum}\\p{Space}]{0,1024}$
Validator.Filename=^[a-zA-Z0-9.\\-_]{0,255}$
Validator.Directory=^[a-zA-Z0-9\\_]{0,255}$
Validator.Path=^[a-zA-Z0-9\\_\\/]{0,255}$
```

Eksempel 6: validation.properties i DPG 2.0

```
# File upload configuration
HttpUtilities.ApprovedUploadExtensions=.xls, .pdf, .odt, .zip,
.rar, .flv, .png, .jpg, .gif, .doc, .swf, .java, .cpp
HttpUtilities.MaxUploadFileBytes=314572800
```

Eksempel 7: Utrag frå ESAPI.properties i DPG 2.0

I tillegg til dei typane data som til no har blitt presentert kan nye typer data for entitetar bli introduserte gjennom nye presentasjonsmønster. Den enklaste metoden for å handtere dette i dag, er ved sanitering med AntiSamy. Men avhengig av moglege typer data som blir introdusert i nye presentasjonsmønster, kan det vere nødvendig med meir spesifikke valideringsreglar.

Validering av inndata i Webucator 3.0

For oppretting av kurs og kursinstansar er det viktig å verifisere at gitt inndata stemmer overens med det ein forventer. Dette er for å unngå moglege angrep som cross-site scripting og SQL innsprøyting.

Inndata	**Valideringsreglar**
Kurskode	Alfanumerisk. Maksimum lengde på 10.
Kurstittel	Alfanumerisk med mellomrom. Maksimum lengde på 45.
Semester	Spring eller Fall
År	Frå 2008 og ti år framover i tid frå noverande år.

Tabell 26: Valideringsreglar for kurs og kursinstansar i Webucator 3.0

Oppretting av brukarar i Webucator 3.0 inneber data som fornamn, etternamn, brukarnamn og e-post. Dette er data som må validerast for å unngå sikkerheitsbrot. I tillegg blir passordstyrke validert.

Inndata	**Valideringsreglar**
Fornamn	Alfabetet med mellomrom og bindestrek. Maksimum lengde på 45.
Etternamn	
Brukarnamn	Alfanumerisk. Maksimum lengde på 10
E-post	Må følge vanleg struktur på e-post adresser.
Passord	To oppgitte passord må vere identiske (for å sikre seg mot at brukaren har skrive feil og ikkje vil få tilgang). Styrke blir verifisert med ESAPI.

Tabell 27: Valideringsreglar for brukardata i Webucator 3.0

Reglane som er forklarte i tabell 26 og 27 kan vi sjå i eksempel 8. År, semester og passord blir validerte i koden. Resten av reglane for validering av strengar blir definerte i `validation.properties`.

```
# Validation-rules for whitelisting input in Webucator 3.0
# ---------------
# Note that the Course-rule applies both for course codes and course
# titles.
Validator.Username=^[\\p{Alnum}]{0,1024}$
Validator.Name=^[A-ZÆØÅa-zæøå\\p{Space}\\-]{0,1024}$
Validator.SafeString=^[.\\p{Alnum}\\p{Space}]{0,1024}$
Validator.Course=^[\\p{Alnum}\\p{Space}]{0,1024}$
Validator.Email=^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\\.[a-zA-Z]{2,4}$
```

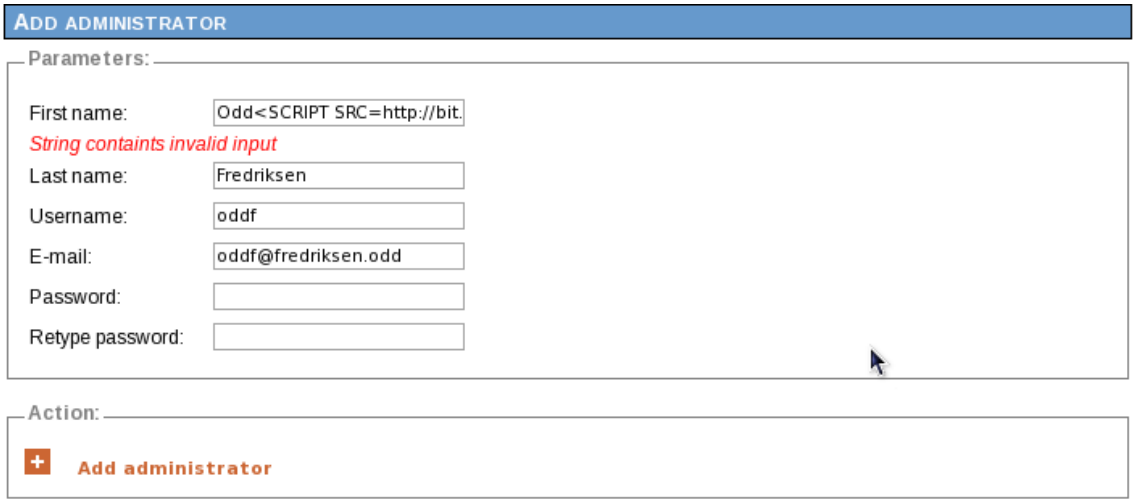
Eksempel 8: validation.properties i Webucator 3.0

Eksempel 9 viser valideringsreglar for filopplasting i `ESAPI.properties`. Opplasta filer bør kun vere Excel-filer, og maksimum størrelse er 10000 bytes.

```
# File upload configuration
HttpUtilities.ApprovedUploadExtensions=.xls
HttpUtilities.MaxUploadFileBytes=10000
```

Eksempel 9: Validering av filer i Webucator 3.0

Figur 25 viser korleis validering kan blokkere skadeleg kode i HTML-skjema for å opprette administratorar. Fornamnet i dette eksempelet inneheld script som ikkje er ynskjeleg.



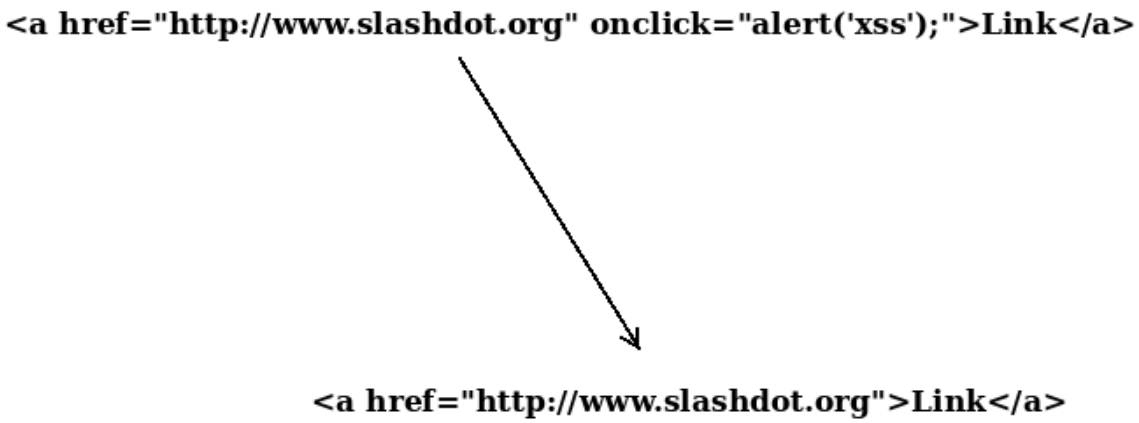
Figur 25: Eksempel - Validering av input i Webucator 3.0

5.1.2 Sanitering av inndata

Gitt at ein ynskjer å ta imot HTML og CSS kode frå brukarane, slik at dei sjølve kan definere utsjånad og formatering på data dei sender til applikasjonen, korleis kan ein sikre seg mot at dei ikkje sender skadeleg kode? Tradisjonell validering av inndata vil i dette tilfellet ikkje vere egna, fordi ein vil måtte tillate teikn som <, > og ' for at brukarane skal kunne gi HTML til applikasjonen.

Sanitering av data inneber å rense inndata for skadeleg kode. Medan validering vil forkaste alle data som ikkje stemmer med eit gitt regelsett vil sanitering forsøke å fjerne dei delane av gitt data som er å ansjå som skadelege, og beholde eit trygt subsett av gitt inndata (figur 28).

Det eksisterer fleire forskjellige metodar for å passe på at brukarar kan formatere inndata for at det skal bli enklare å lese. Ein metode er å enkode all inndata, og kun dekode kjende HTML-tags basert på ei liste med gode verdiar (*whitelisting)* [59]. Ein kan og utvikle eigen *markup* for å la brukarar formatere inndata [59]. Eksempel på dette er BBCode [59, 60]. Dette er ofte ikkje ynskjeleg då det inneber at brukarane må lære seg eit nytt språk, samt at det ikkje er like rikt som HTML [59].



Figur 26: Eksempel - Sanitering av inndata med AntiSamy

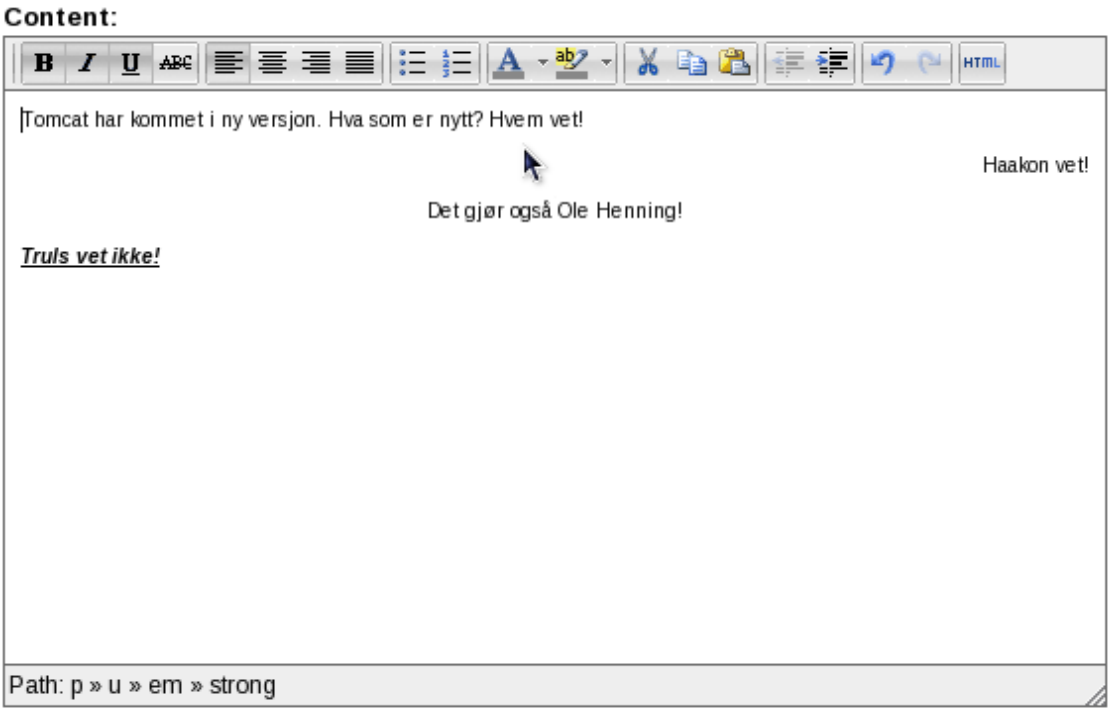
For sanitering av data i DPG 2.0 kan ein bruke AntiSamy [49]. AntiSamy er eit XSS filter som tillet brukarar å sende inn HTML og CSS, men vil blokkere skadeleg innhald. Tradisjonelt sett har organisasjonar utvikla eigne løysingar for sanitering av HTML og CSS. Utvikling av slike verktøy er svært utfordrande. På mange måtar er det ikkje ulikt det å skrive eigne krypteringsmekanismar, og alvorlege feil blir gjort. Sjølv større selskap som MySpace og YouTube har hatt problem med metodar dei sjølv har produsert.

Sanitering i staden for validering blir ofte sett på som usikkert og risikabelt. Ei anbefaling i [19] er å forkaste all inndata når validering feiler, og ikkje gjere forsøk på å rense gitt inndata for dårlege verdiar. Forskjellen her er at AntiSamy nyttar seg av whitelisting mot ei konfigurasjonsfil, og vil forkaste eller enkode alt som ikkje stemmer med det konfigurasjonsfila seier er tillatt *etter* at gitt HTML har blitt parsa. På denne måten blir kvar HTML tag og kvar CSS kodesnutt validert for seg sjølv [49].

I oktober 2005 opplevde nettopp MySpace at deira eigne filter svikta, og ormen Samy (derav namnet AntiSamy) spredte seg gjennom applikasjonen ved hjelp av cross-site scripting og cross-site request forgery. Over ein million brukarar blei påvirka på under eit døgn. Problemet med filteret til MySpace var at dei nytta ei svarteliste over skadelege taggar og strengar [19].

Sanitering av inndata i DPG 2.0

Dei fleste entitetar i DPG 2.0 blir redigerte med TinyMCE [51]. TinyMCE er eit verktøy skrive i JavaScript som tilbyr moglegheit for å formatere tekst med HTML og CSS (figur 27). OWASP har gitt ut ei konfigurasjonsfil som kan nyttast med AntiSamy for å sanitere data uten å redusere funksjonaliteten i TinyMCE. Mindre endringar i denne fila må til for at den skal kunne nyttast med den versjonen av TinyMCE som blir nytta i DPG 2.0. Det må blant anna leggast til CSS-reglar for oppstilling av tekst (*venstre, høgre, midstilt, justified*).



Figur 27: TinyMCE i DPG 2.0

I det ein publisher eller administrator vel å trykke på submitt blir AntiSamy nytta for å verifisere at innhaldet stemmer overens med det som er forventa i hendhold til konfigurasjonsfila til AntiSamy.

5.1.3 Enkoding av utdata

Enkoding av utdata eller *escaping* er ein teknikk for å forsikre seg om at data som skal nyttast i ein gitt kontekst blir brukt kun som data, og ikkje som kode som skal køyre i den relevante parseren [51]. Dette blir som regel gjennomført ved å nytte spesielle teikn og sekvensar av teikn for å markere at gitt data ikkje blir køyrd som kode i ein spesifikk kontekst, som HTML, SQL, JavaScript og liknande [51]. ESAPI tilbyr enkoding for ei rekke kontekstar. I DPG 2.0 og Webucator 3.0 er det følgande kontekstar som er relevante:

- HTML Element
- HTML Entity
- JavaScript
- HTML URL parameter

Tabell 28 viser ei oversikt over dei ulike kontekstane, samt enkodinga som inngår i kvart tilfelle.

HTML Element	`String safe = ESAPI.encoder().encodeForHTML(String unsafe);`

`<body>... **ESCAPE HER**...</body>`
`<div>...**ESCAPE HER**...</div>`
Og andre ordinære HTML element.	

HTML Entity	`String safe = ESAPI.encoder().encodeForHTMLAttribute(String unsafe);`

`<div attr=...**ESCAPE HER**...>innhald</div>`
`<div attr='...**ESCAPE HER**...'>innhald</div>`
`<div attr="...**ESCAPE HER**...">innhald</div>`	

JavaScript	`String safe = ESAPI.encoder().encodeForJavaScript(String unsafe);`

`<script>alert('...**ESCAPE HER**...')</script>`
`<script>x='...**ESCAPE HER**...'</script>`
`<div onmouseover="x='...**ESCAPE HER**...'"></div>`	

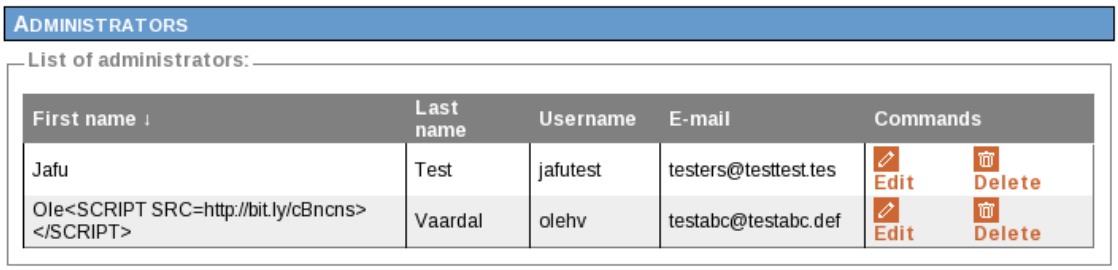
HTML URL parameter	`String safe = ESAPI.encoder().encodeForURL(String unsafe);`
`<a href="...**ESCAPE HER**...">lenke</a>`	

Tabell 28: Oversikt over enkoding i JAFU-systema

Enkoding av utdata skjer på samme måte i både DPG 2.0 og i Webucator 3.0. Det blir gjennomført med JSP scriptlets. Det blir gjort på denne staden i applikasjonen fordi samme data blir ofte gjengitt i fleire kontekstar. Ved å enkode i scriptlets er det mogleg å definere korrekt enkoding for riktig kontekst på riktig punkt i applikasjonen. Eksempel 10 viser korleis administratordata blir enkoda i `listAdministrators.jsp` i Webucator 3.0. Figur 28 viser korleis enkoda data ser ut i nettlesaren.

```
<%
List<User> l = (List<User>) request.getAttribute("admins");
    for (User u: l) {
    %>
    <tr>
        <td><%=ESAPI.encoder().encodeForHTML(u.getFirstName())%></td>
        <td><%=ESAPI.encoder().encodeForHTML(u.getLastName())%></td>
        <td><%=ESAPI.encoder().encodeForHTML(u.getUsername())%></td>
        <td><%=ESAPI.encoder().encodeForHTML(u.getEmail())%></td>
        <td>
            <img src="<c:url value='/graphics/icons/edit.gif' />"
            class="iconlink"/>
            <a href="editAdministrator.html?user=
            <=ESAPI.encoder().encodeForURL(u.getUsername())%>"
            title="Click here to edit this administrator">
            Edit</a>
        </td>
        <td class="button">
            <img src="<c:url value='/graphics/icons/trash.gif' />"
            class="iconlink"/>
            <a href="deleteAdministratorSuccess.html?username=
            <%=ESAPI.encoder().encodeForURL(u.getUsername())%>"
            title="Click here to delete this administrator"
            onclick="return confirm('Are you sure you want to
            delete
            <%=ESAPI.encoder().encodeForJavaScript(u.getFirstName())
            %>
            <%=ESAPI.encoder().encodeForJavaScript(u.getLastName())
            %>?')">Delete</a>
        </td>
    </tr>
    <%
    }
%>
```

Eksempel 10: Enkoding av utdata i listAdministrators.jsp i Webucator 3.0



Figur 28: Enkoding av utdata i Webucator 3.0

5.1.4 Sikker passordgenerering og verifisering

ESAPI tilbyr innebygde funksjonar for sikker generering av passord. På den samme måten som den opprinnelege funksjonen i Webucator 3.0 for passordgenerering har ESAPIs passord ein bestemt struktur.

X X X X Y Z Z Z

Der *X* er bokstavar i alfabetet mellom a og z. Desse bokstavane kan både vere store og små, og det er ingen krav om vekslande vokalar og konsonantar. *Y* er eit spesialteikn. Dette kan til dømes vere !,* og $ med meir. *Z* er tall mellom 0 og 9. Sjølv om dette inneber ein relativt kjend struktur er framleis mengda med moglege passord såpass stor at den sjeldan vil generere samme passord to gongar slik tilfellet var med den opprinnelege funksjonen.

I tillegg nyttar den ESAPIs `Randomizer`. Standard for `Randomizer` er `java.security.SecureRandom` med ei algoritme for slumptallsgenerering spesifisert i `ESAPI.properties`. Standardfila spesifiserer *Secure Hash Algorithm* som algoritme for dette formålet. Dette gjer at angrep på slumptallsgeneratoren for å finne passord vil vere langt vanskelegare. Skulle det vere behov for endring av algoritme i framtida vil dette kunne gjennomførast ved å redigere `ESAPI.properties`.

```
/**
 * Generate a strong password that is not similar to the
 * specified old password.
 *
 * @param oldPassword the password to be compared to the new
 * password for similarity
 * @return a new strong password that is dissimilar to the
 * specified old password
 */
private String generateStrongPassword(String oldPassword) {
    Randomizer r = ESAPI.randomizer();
    int letters = r.getRandomInteger(4, 6);// inclusive,exclusive
    int digits = 7 - letters;
    String passLetters = r.getRandomString(letters,
      EncoderConstants.CHAR_PASSWORD_LETTERS);
    String passDigits = r.getRandomString(digits,
      EncoderConstants.CHAR_PASSWORD_DIGITS);
    String passSpecial = r.getRandomString(1,
      EncoderConstants.CHAR_PASSWORD_SPECIALS);
    String newPassword = passLetters + passSpecial + passDigits;
    if (StringUtilities.getLevenshteinDistance(oldPassword,
      newPassword) > 5) {
        return newPassword;
    }
    return generateStrongPassword(oldPassword);
}
```

Eksempel 11: ESAPI.authenticator().generateStrongPassword(String oldPassword)[57]

Verifisering av passord er også mogleg i ESAPI. Denne metoden går gjennom passordet og vurderar forekomst av store og små bokstavar, tall og spesielle teikn. Dette blir så satt opp mot lengda på passordet for å produsere ein verdi som svarer til styrken på passordet. Passord med ein verdi mindre enn 16 blir av ESAPI vurderte som svake. Basert på denne funksjonen kan til dømes eit passord på 8 teikn ha fire forskjellige verdiar som representerer styrken på passordet (tabell 29).

8	Ein type teikn inngår i passordet
16	To typar teikn inngår i passordet
24	Tre typar teikn inngår i passordet
32	Fire typar teikn inngår i passordet

Tabell 29: Eksempel - ESAPIs rangering av passord som er 8 teikn lange

Desse funksjonane i ESAPI imøtekjem behovet for sterkare passord for brukarane i DPG 2.0. Automatisk genererte passord er ikkje lenger forutsigbare, og alle passord blir verifiserte for å forsikre seg om god nok styrke.

5.2 Spring Security

Spring Security begynte som ACEGI i 2003. For versjon 2.0 av Spring vart ACEGI inkludert i rammeverket som Spring Security. Det er eit sikkerheitsrammeverk for autentisering og tilgangskontroll i Java og .NET.

Hovusakeleg kan Spring Security nyttast for å passe på at data som passord blir lagra sikkert og ikkje vil vere tilgjengeleg dersom databasen blir eksponert.

5.2.1 Sikker lagring

Webucator 3.0 ville opprinneleg samanlikne passorda sendt frå brukaren med passordet i databasen. For dette formålet var det nødvendig at passordet i databasen var lagra i klartekst. For å utbetre dette har valet falt på å kryptere passorda. Spring Security tilbyr ei rekke moglegheiter for å gjennomføre dette (tabell 26).

Passord encoder	**Formål**
`encoding.Md5PasswordEncoder`	Hasher passordet med Message Digest (MD5).
`encoding.PlaintextPasswordEncoder`	Gjennomfører ingen endringar og returnerer passordet i klartekst.
`encoding.ShaPasswordEncoder`	Hasher passordet med Secure Hash Algorithm (SHA).
`ldap.authenticator.LdapShaPasswordEncoder`	Hasher passordet med LDAP SHA og salta SHA (SSHA)

Tabell 30: Spring Security - Metodar for passordkryptering

Valget her fell på bruk av *SHA-1*. Dette gjer det mogleg å lagre passorda langt sikrare i databasen. Før ei eventuell samanlikning av passord sendt av brukaren må også dette passordet krypterast. Dersom det krypterte passordet frå brukaren stemmer med det krypterte passordet i databasen vil det bety at passordet er det samme.

Webucator 3.0 sine eigne administratorar blir allereie lagra sikkert med SHA-1. Likevel sikrer ikkje dette passorda mot angrep som kan gjennomførast lokalt. Brute-forcing av passord er langt raskare når ein slepp å vente på svar frå tenaren mellom kvart forsøk. I tillegg er det, som tidlegare nemntt, mogleg å bruke oppslagstabellar med førehandsgenererte verdiar for å finne passord. For å motvirke dette kan ein nytte salting av passorda. Dette gjeld både for brukarane i DPG 2.0 og administratorane i Webucator 3.0.

Salting inneber å legge til ein ekstra verdi (eit salt) før kryptering av passordet, noko som bidrar til at det effektive passordet blir mykje lenger [31]. Ein kan nytte to ulike

typer salt. Salt som gjeld for heile systemet, og salt som reflekterer unike detaljer med brukaren. Reflekterte salt vil vere det ynskjelege for DPG 2.0. På denne måten må ein angriper evaluere hashen til kvar enkelt brukar med eit unikt salt, noko som er langt meir tidkrevande.

Det er viktig at saltet ikkje endrar seg. Verdien som utgjer saltet må vere fast fordi dersom denne verdien endrar seg vil det bety at det ikkje blir samsvar mellom den krypterte utgåva av passordet som kjem frå brukaren, og den krypterte utgåva av passordet som ligg i databasen. Dette vil medføre at applikasjonen blir utilgjengeleg for brukaren.

For readers og publishers er det her nok å nytte brukarnamn som salt sidan det ikkje vil endre seg. For administratorar er det derimot litt annaleis, sidan dei har moglegheit til å endre brukarnamn. For å i møtekome dette blir det for kvar administrator også generert ein unik streng som vil fungere som salt. Til dette kan vi nytte ESAPIs passordgenerator som er beskrive i kapittel 5.1.4.

Kombinasjonen av sterke passord og sikker lagring med einvegskryptering og salting fører til at det blir langt vanskelegare å avdekke sensitiv data dersom databasen til Webucator 3.0 blir eksponert.

5.3 Generelle endringar

I tillegg til sikkerheitsfunksjonar i Spring Security og i verktøy frå OWASP må det også enkelte andre endringar til i DPG 2.0 og Webucator 3.0.

5.3.1 Autentisering

For å i møtekome behovet for betre autentisering var det nødvendig med betre passordgenerering og verifisering. I tillegg er det ein nyttig framgangsmåte å avgrense antall innloggingsforsøk for kvar brukar [18].

For å kunne avgrense antall innloggingsforsøk er det nødvendig med ein teller som blir inkrementert kvar gong eit innloggingsforøk feiler. Med utgangspunkt i dette måtte `User.java` endrast for å inkludere ein slik teljar. Denne teljaren må også vidareførast til databasen.

For kvart feila innloggingsforsøk der brukarnamnet er korrekt blir tellaren attempts i `User.java` inkrementert. Etter dette blir oppføringa til den aktuelle brukaren oppdatert i databasen for å inkludere denne telleren. Vidare innloggingsforsøk vil deretter hente ut teljaren og sjekke den mot ei øvre grense. Når teljaren når denne grensa vil det bli kasta eit unntak og brukaren vil ikkje kunne logge inn før tellaren er nullstilt.

Dette vil stanse eventuelle brute-force angrep. På grunn av krav om passordstyrke er det lite truleg at passorda blir oppdaga innan 10 forsøk, og eit angrep vil ikkje bli vellukka etter at den aktuelle kontoen er låst.

Tilbakemeldinga til brukaren bør vere generinsk i dette tilfellet. Det vil seie at tilbakemeldinga ikkje bør endre seg etter at kontoen blir låst, men heller innehalda informasjon om at fleire mislukka forsøk på innlogging vil føre til låst konto frå byrjinga. Dersom feilmeldinga som blir presentert til brukaren endrar seg etter ti forsøk for å fortelle om ein låst konto kan angriparar nytte denne informasjonen til å samle gyldige brukarkontoar for vidare angrep.

5.3.2 Sikker kommunikasjon

For å sikre DPG 2.0 og Webucator 3.0 mot avlytting og mann-i-midten angrep kan ein nytte HTTPS for å sikre kommunikasjonskanalane. HTTPS er ein kombinasjon av Hypertext Transfer Protocol og Transport Layer Security for å kryptere komminkasjon og sikre identifikasjon av tenaren.

Det er to metodar for å bruke TLS. På ei side kan ein bruke Spring Security for dette formålet. Dette vil derimot medføre ein del kode- og konfigurasjonsendringar. Eit betre alternativ er derfor å aktivere TLS på Apache Tomcat [7]. Dette er enklare og raskare, og krev ingen kodeendringar.

Sikring mot mann-i-midten angrep krev to ting. Det første er eit sertifikat som er signert av ein tredjepart som både klient og tenar stoler på. Det andre er at brukarane er klare over at sertifikatet skal vere signert, og ikkje godtek usignerte utgåver. Dersom brukarane likevel godtek eit usignert sertifikat frå tenaren, vil det framleis vere mogleg å gjennomføre mann-i-midten angrep [52].

5.3.3 Presentation Manager

Eit av problemområda i DPG 2.0 var Presentation Manager. Sikkerheita til dette systemet er fullstendig avhengig av at tilgangskontrollen ikkje feiler og at ingen administratorar blir utsatte for angrep som medfører innbrot via deira brukarkontoar.

For å fullstendig separere tilgangen til dette delsystemet frå Presentation Content Editor og Presentation Viewer, bør dette delsystemet skillast ut i frå DPG 2.0 og opprettast som ein eigen applikasjon.

Det eksisterer ingen enkle måtar å sikre at administratorar skal kunne gjennomføre arbeidsoppgåvene sine i dette delsystemet samstundes som ein vil passe på at dei ikkje skal kunne misbruke denne funksjonaliteten.

5.4 Oppsummering

Problem knytt til loggforfalsking og katalogtraversering (tenestenekting og brot på tilgangskontroll) blir handtert med validering i ESAPI. XSS blir handtert med validering og sanitering av inndata, samt enkoding av utdata. For dette formålet er ESAPI og AntiSamy gode verktøy. Ein kunne ha løyst desse problema med Spring Security, men validering med ESAPI er lettare å implementere, og vil oppdage forsøk på å omgå valideringa ved enkoding gjennom *canonicalization*.

Problem med usikker lagring blir løyst ved at alle passord blir *hasha*, altså einvegskrypterte med SHA-1. For å gjere dette blir Spring Security nytta. Her blir også kvart passord salta med brukernamnet for *readers* og *publishers*. For administratorar blir det generert ein unik verdi som blir nytta for salt. Til dette formålet blir ESAPIs passordgenerator nytta.

ESAPIs passordgenerator blir dessutan nytta for å generere sterkare passord som vil vere svært vanskelege å forutsjå for ein eventuell angripar. For administratorar som skal ha moglegheit til å velje passord sjølve blir desse verifisert med ESAPI for å passe på at dei vel sterke nok passord.

Autentiseringa har blitt forsterka blant anna gjennom betre passord, samt eit avgrensa antall innloggingsforsøk for kvar bruker. Avgrensing av innloggingsforsøk for brukarane er ein enkel måte å stanse *brute-force*-angrep, men det opnar for tenestenekting. Derfor vil det og vere nødvendig å handtere dette problemet på nettverksnivået, gjennom blant anna brannmurar som vil kunne blokkere IP-adresser som forsøker eit større antall innloggingar.

For sikker kommunikasjon er det nødvendig med ei rekonfigurering av Apache Tomcat, og eit gyldig sertifikat. Brukarane må få informasjon om dette og få beskjed om å kun stole på nettsida når sertifikatet blir godkjent av nettlesaren.

6

6 Sikkerheit og systemutvikling

Dette kapittelet tek for seg sikkerheit som ein del av utviklings- og organisasjonsprosessen til JAFU-systema, og korleis enkelte manglar i denne prosessen kan føre til sikkerheitsproblem i ein ferdigutvikla vevapplikasjon.

Ynskje om forbedring av sikkerheita i JAFU var eit av utgangspunkta for å bygge nye system [1, 2, 3]. Det blei gjennom analyse av dei tidlegare systema avdekka alvorlege sikkerheitsproblem, og nye rammeverk og teknologiar vart tatt i bruk for å handtere desse, og forbetre sikkerheitssituasjonen til systema.

> *«Any person can invent a security system so clever that he or she can't imagine a way of breaking it.»*
>
> -- *Cory Doctorow*

Sitatet over er kjend som *Schneiers lov.* Konseptet vart lagt fram av Cory Doctorow under ein Microsoft konferanse om *Digital Rights Management*, og refererer til Bruce Scheiners bok *Applied Cryptography* [11]. Sjølv om utgangspunktet for sitatet omhandlar sikkerheit knytt til kryptografi, har det stor relevans i forhold til applikasjonssikkerheit.

Når ein utvikler eit nytt system med sikkerheit i tankane er det lett å ta for seg sikkerheit som funksjonar eller nye delar av systemet som blir lagt til i form av passord, kryptografi og tilgangskontroll. Det er ofte lett å ignorere korleis programkode og andre funksjonar kan misbrukast og utnyttast på måtar ein ellers ikkje hadde sett for seg. Ein

programmerar vil som regel jobbe med utgangspunkt i korleis programmet skal fungere, og ignorere korleis det ikkje er ynskjeleg at programmet skal fungere [19].

Dei nye systema tok for seg dei sikkerheitsproblema allereie avdekka i dei gamle systema, og jobba for at desse ikkje skulle oppstå i dei nye systema. Det blei derimot ikkje gjort vurderingar av nye problem som kunne oppstå som eit resultat av at heilt nye system vart utvikla, og utover spesifikasjon og implementasjon av teknologiar for å handtere autentisering og tilgangskontroll, vart få sikkerheitsrelaterte aktivitetar gjennomførte som ein del av utviklingsprosessen, og i JAFU som organisasjon.

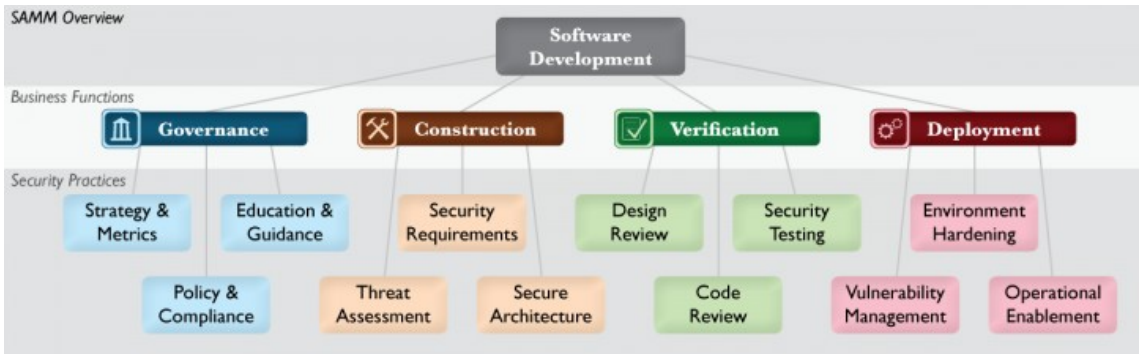
Eit resultat av dette var at vevapplikasjonane inneheld omfattande sikkerheitsfeil, som til no har blitt presenterte. Dette kapittelet tek for seg sikkerheitsaktivitetar som bør vere ein del av ein utviklings- og organisasjonsprosess.

6.1 Introduksjon til OpenSAMM

OpenSAMM står for Open Software Assurance Maturity Model. Det er eit ope rammeverk som skal hjelpe organisasjonar med å formulere og implementere strategiar for sikker programvareutvikling skreddarsydd for dei truslane og risikoane organisasjonen står overfor [43]. Dette inneber

- Evaluering av eksisterande praksis rundt programvaresikkerheit.
- Bygging av eit avansert program for utvikling av sikker programvare.
- Demonstrere konkrete forbetringar i utviklignsprogrammet med fokus på sikkerheit.
- Definere og måle sikkerheitsrelaterte aktivitetar i ein organisasjon.

Modellen er inndelt i fire hovudkategoriar der kvar kategori har tre underkategoriar (Figur 29). Kvar underkategori har tre forskjellige grader og eit nullpunkt. Nullpunktet relfekterer den situasjonen der ingenting enno har blitt gjennomført for den aktuelle underkategorien [43].



Figur 29: Oversikt over OpenSAMM [43]

Dei fire kategoriane *Governance*, *Construction*, *Verification* og *Deployment*. Dette er såkalla *business functions*. Ein kvar organisasjon som driv med programvareutvikling vil bruke desse fire kategoriane til ei viss grad [43].

Governance handlar om prosessar og aktivitetar relatert til korleis ein organisasjon administrerer overordna aktivitetar i programvareutvikling. Construction inneber prosessar og aktivitetar relatert til korleis ein organisasjon definerer mål og produserer programvare i utviklingsprosjekt. Dette inneber design, kravspesifikasjon, spesifisering av arkitektur og implementasjon. Verification er korleis ein organisasjon sjekkar og testar gjenstander som blir produsert under utviklinga. Deployment omhandlar utgiving av programvare som har blitt produsert, samt vedlikehald. Underkategoriane i OpenSAMM er beskrive i tabell 31.

Governance		
Strategy & Metrics involverer den generelle strategiske retninga ein organisasjon ynskjer å ta, samt ei evaluering av eksisterande sikkerheitspraksis i organisasjonen.	*Policy & Compliance* involverer strategiar for samsvar med sikkerheitsstandardar og lovverk, samt rammeverk for verifisering av dette.	*Education & Guidance* involverer utdanning og kompetansebygging innanfor sikkerheit blant personell som skal jobbe med systemet.
Construction		
Threat Assessment involverer identifikasjon av truslar og potensielle angrep mot organisasjonens programvare.	*Security Requirements* involverer inkludering av sikkerheitskrav under kravspesifisering som ein del av utviklingsprosessen.	*Security Architecture* handlar om forsterking av sikkerheit i designprosessen, samt inkludering av teknologiar og rammeverk.
Verification		
Design Review er inspeksjon av designgjenstander for verifisering av gode nok sikkerheitsmekanismer og samsvar med organisasjonens forventningar til sikkerheit.	*Code Review* inneber evaluering av kildekode for å oppdage sårbarheiter, samt formilding og reparasjon, for å etablere praksis rundt sikker programmering.	*Security Testing* involverer testing av programvare i kjøretid for å finne sårbarheiter og sette ein minstestandard for release.
Deployment		
Vulnerability Management er etablering av ein konsisten prosess for å handtere interne og eksterne sårbarheitsrapportar.	*Environment Hardening* er implementasjon av kontrollar for operasjonsmiljøet til programvare med sikte på å forbetre sikkerheit i applikasjonar som er i bruk.	*Operational Enablement* involverer identifikasjon og innhenting av relevant informasjon for å konfigurere, bygge og køyre programvare.

Tabell 31: Beskrivelse - OpenSAMM

OpenSAMM legg til rette for at ein kan måle og sette poeng til sikkerheitspraksisen i ein organisasjon. Som sagt har kvar av underkategoriane tre grader av gjennomføring, og eit nullpunkt der ingen ting blir gjort. For kvart nivå som blir gjennomført kan ein sette eit poeng. Summen av alle nivå i alle underkategoriar reflekterer målinga for sikkerheitspraksisen til ein organisasjon. Dette gjer det mogleg å gjere ei vurdering av sikkerheitspraksisen, samt sjå klar framgang etter kvart som organisasjonen byggjer ein strategi og modell som stemmer med den organisatoriske strukturen [43].

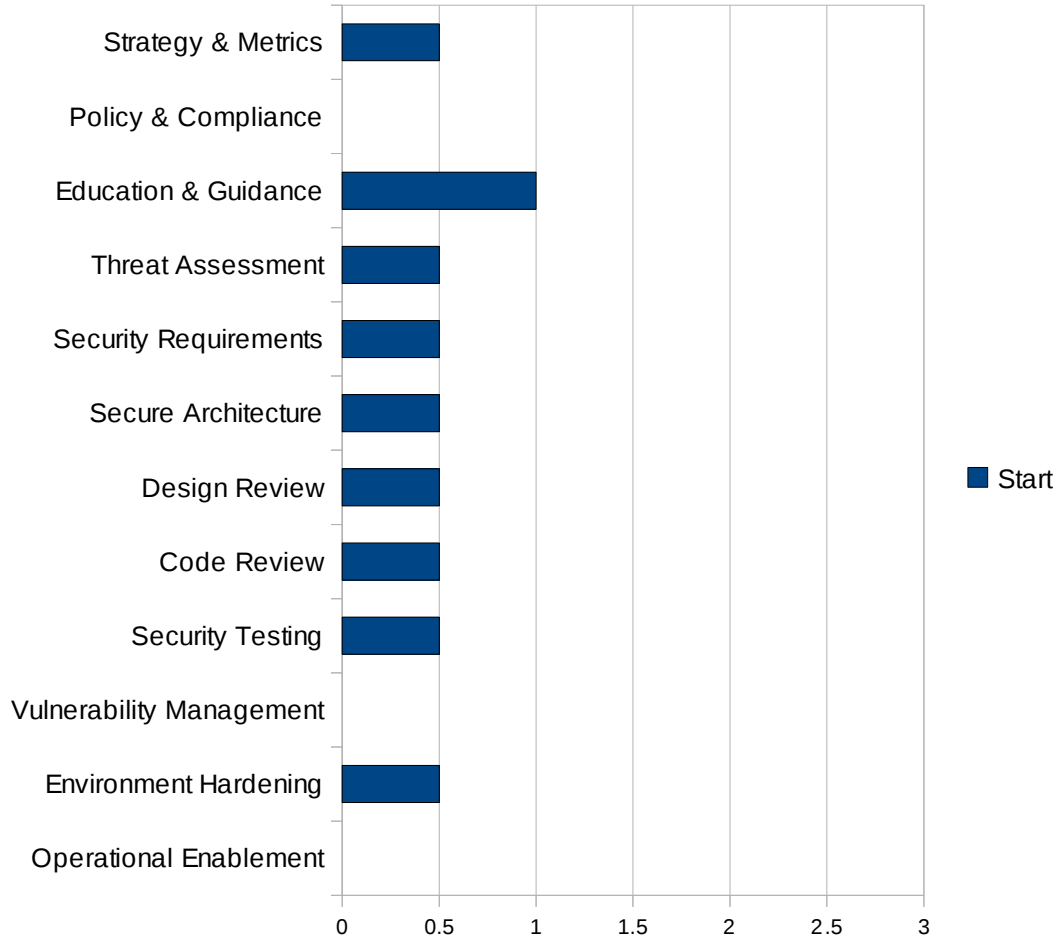
6.2 OpenSAMM i JAFU

Ein grunn til at det her er lurt å nytte OpenSAMM for å bygge ein modell for JAFU er at OpenSAMM gjer det mogleg å sjå eventuelle forbetringar under planlegging og gjennomføring av sikkerheitsaktivitetar. Dette kapittelet vil gå gjennom OpenSAMM og bygge ein modell for JAFU. OpenSAMM inneheld også ei stor breidde i sikkerheitsaktivitetar med ulike nivå for modenheit. Dette gjer det mogleg for ein organisasjon å skreddersy modellen for dei forholda som er til stades i organisasjonen.

Det eksisterer alternativ for integrere sikkerheitsaktivitetar i utviklingsprosessen. Andre modellar for dette er til dømes OWASP CLASP (Comprehensive Lightweight Application Security Process) [63], Microsoft SDL (Security Development Lifecycle) [62] og Cigital Touchpoints [61]. Desse modellane har likevel nokre svakheiter.

- CLASP
  - Definerer ei rekke sikkerheitsaktivitetar ein organisasjon bør integrere i utviklingsprosessen sin.
  - Gir ingen prioriteringar som kan hjelpe ein organisasjon i å finne ut kvar det er lurt å begynne arbeidet [64].
- Microsoft SDL
  - Omfattande modell utvikla av Microsoft for å i møtekome behova dei hadde for sikker utvikling.
  - Ofte for tung for mindre organisasjonar [64].
- Touchpoints
  - Gir ei oversikt over utviklingsprosessen, og definerer aktivitetar for kvar del.
  - For høgnivå. Gir ikkje nok detaljar som kan hjelpe organisasjonar som ikkje er kjende med konsepta [64].

For å skape eit bilde av sikkerheittsituasjonen i JAFU kan vi nytte OpenSAMM for å rangere dei ulike kategoriane i modellen opp mot JAFU og dei sikkerheitsrelaterte aktivitetane som blei gjennomført under utviklinga av DPG 2.0 og Webucator 3.0. Sidan OpenSAMM inneheld 12 kategoriar med tre nivå for kvar kategori og to mål for kvart nivå, kan vi gjere målingane basert på i kva grad desse måla blir oppnådd. Ei oversikt over utgangspunktet av sikkerheitsaktivitetar kan vi sjå i figur 30.



Figur 30: Utgangspunkt – Måling av OpenSAMM mot JAFU

Trussel- og risikovurdering for systema har altså vore mangelfull. Den opprinnelege vurderinga som vart gjort under utviklinga av DPG 2.0 og Webucator 3.0 var kun gjort med utgangspunkt i sårbarheiter som vart funne i eldre system som vart fullstendig erstatta [1, 2, 3]. Ingen spesifikke metodar for vurdering av potensielle truslar inngjekk i dette. Denne mangelfulle vurderinga av truslane i systemet har satt sitt preg på kva sikkerheitskrav som var satt for utvikling av nye system, kva potensielle problem ein fokuserte på, samt kva rammeverk og teknologiar som har blitt tatt i bruk for å i møtekome eventuelle sikkerheitsproblem.

Sikkerheitskrava til det nye systemet var basert på den vurderinga masterstudentane hadde av truslane mot systemet, og manglande standardar og retningslinjer for sikkerheit i vevapplikasjonar. Dette innebar altså kun løysingar for tilgangskontroll og autentisering av brukarar. Men det vart og påvist problem som for eksempel tenestenekting, og det var ikkje utarbeida krav om å i møtekome samme type problem for nye system.

Ei rekke av dei problema som vart påviste i analysa som er presentert i denne oppgåva

er trivielle å utnytte, samt relativt lette å oppdage. Fleire av studentane som arbeida med utviklinga av systemet, hadde deltatt i undervisning med fokus på sikkerheit i vevapplikasjonar [1, 2]. Dei burde derfor hatt nødvendig kompetanse for å oppdage ein del av desse problema gjennom grunnleggande gjennomgang av design og kode, samt gjennomføring av sikkerheitstesting. Grunnen til at dette ikkje blei gjennomført er usikker, men manglande integrering av sikkerheitsaktivitetar i arbeidsprosessen til studenta, samt manglande prioritering av desse kan ha bidratt til desse manglane.

Med utgangspunkt i den opprinnelege rangeringa kan ein altså konkludere med at til tross for at prosjektmedlemmar blir utdanna innanfor sikkerheit og har den kompetansen nødvendig for å finne og reparere sikkerheitsproblem i applikasjonane dei utvikler eksisterer det ingen formelle prosedyrer for å gjennomføre dette.

Dette medfører følgande punkt som trengs i JAFU:

- Behov for overordna sikkerheitsstrategi og risikovurdering for JAFU
- Auka kompetanse blant prosjektmedlemmar innanfor sikkerheit.
- Bedre og meir omfattande vurdering av truslar systema kan stå overfor
- Breiare og meir omfattande sikkerheitskrav som handterer fleire moglege truslar i JAFU-systema.
- Utbedra vurdering av rammeverk og teknologiar som kan nyttast for å i møtekome nye sikkerheitskrav.
- Grunnleggande sikkerheitsvurdering av systemdesign og kildekode før systema blir tatt i bruk.
- Grunnleggande sikkerheitstesting av dei mest kritiske modulane i systema før dei blir tatt i bruk.
- Prosedyrer for handtering av nye sikkerheitsproblem i system som er i bruk.

JAFU som ein organisasjon er relativt unik i forhold til andre organisasjonar. Dei fleste utviklarane prosjektet er studentar, som gjerne jobbar individuellt og i små lag. Nøkkelutfordringane for JAFU kan oppsummerast som følgande punkt:

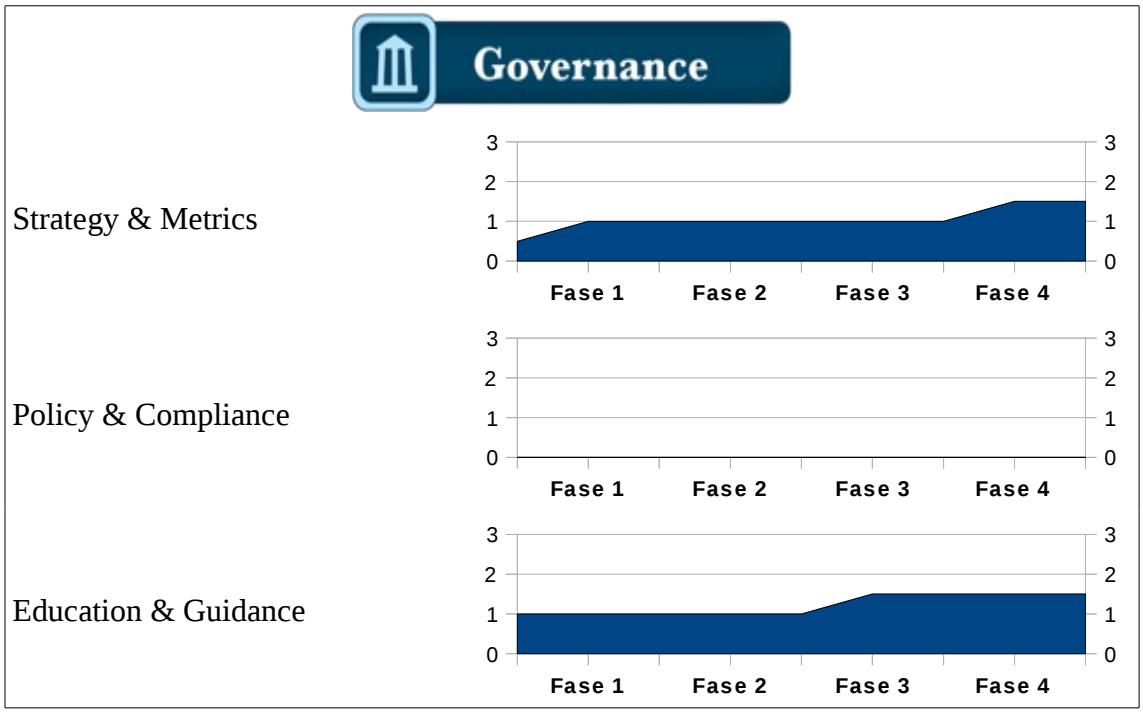
- Regelmessig utbytting av prosjektmedlemmar kvart/annakvart år. Dette er på grunn av at nye studentar kjem til prosjektet, og gamle studentar går ut av universitetet etter fullført oppgåve og grad. Nye prosjektmedlemmar har derfor avgrensa med erfaring.
- Prosjekta er ofte individuelle eller med små grupper utviklarar. Studentar har avgrensa med tid og ressursar til å fokusere på avanserte sikkerheitsaktivitetar.
- Stor bredde av forskjellige teknologiar blir nytta i organisasjonen, då studentar fokuserer på forskjellige problemstillingar for sine respektive oppgåver.
- Manglande dokumentasjon av truslar og risiko i organisasjonen vitner om at

studentane har lite forståing for desse utfordringane når dei går inn i JAFU.

6.2.1 Vegkart for utbedring av sikkerheitsaktivitetar

For utbedring av sikkerheitsaktivitetane i JAFU har det blitt satt opp følgande vegkart med utgangspunkt i sikkerheitsaktivitetane i OpenSAMM. Det vil bli demonstrert korleis OpenSAMM kan hjelpe med ei forbedring av den generelle sikkerheitssituasjonen i JAFU over fire fasar, eller iterasjonar på tre månadar. Desse fasane blei definert for at prosjektmedlemmane skal få tid til å sette seg inn i aktivitetane, og samstundes ha tid til å gjere dei andre arbeidsoppgåvene sine. På grunn av regelmessig utskifting av prosjektmedlemmar i JAFU, samt kva fokus mange av dei har i forhold til eigne prosjekt vil det ikkje vere spesielt ynskjeleg å bruke mykje tid og ressursar på for avanserte aktivitetar. Studentane arbeider ofte individuellt, og vil ha fleire forskjellige roller (testing, design, implementasjon o.l.), og ein må få tid til slike aktivitetar i tillegg til at ein skal kunne gjere andre arbeidsoppgåver.

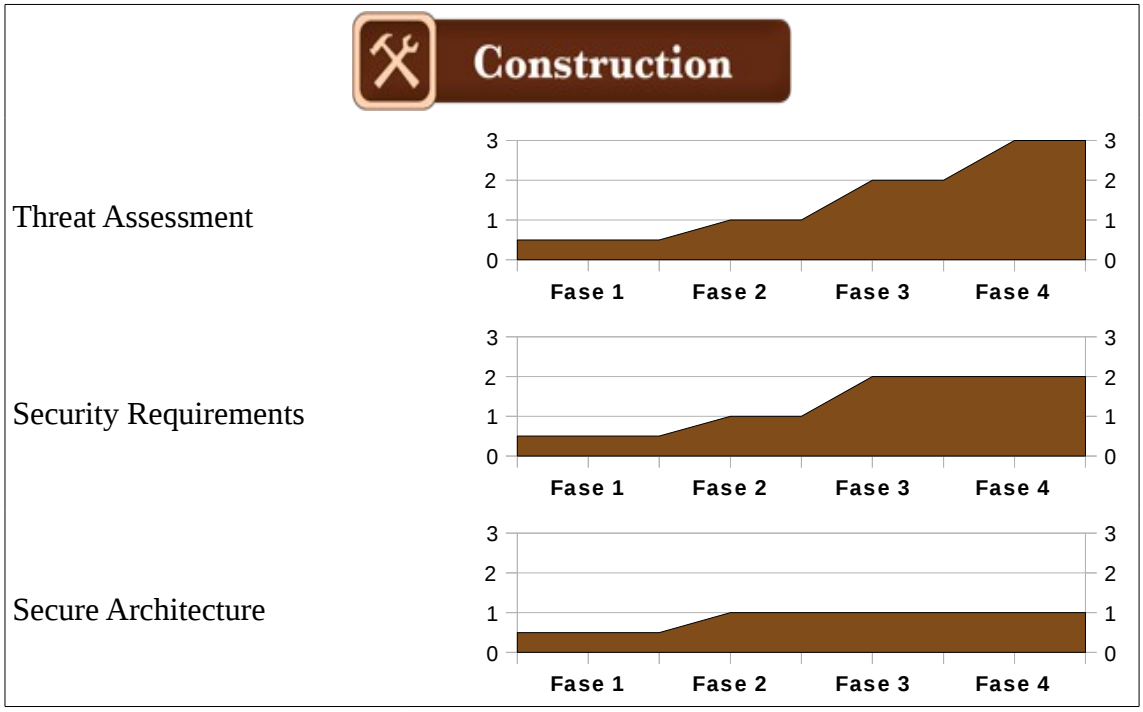
Figur 31 viser den planlagte forbedringa av administrative sikkerheitsaktivitetar i JAFU. Her er det i hovudsak strategi for sikkerheit i JAFU som skal utbedrast saman med utdanning av prosjektmedlemmar, for å bygge den nødvendige kompetansen til å kunne utvikle systema på ein sikker og konsistent måte.



Figur 31: Governance - Vegkart for dei fire neste fasane

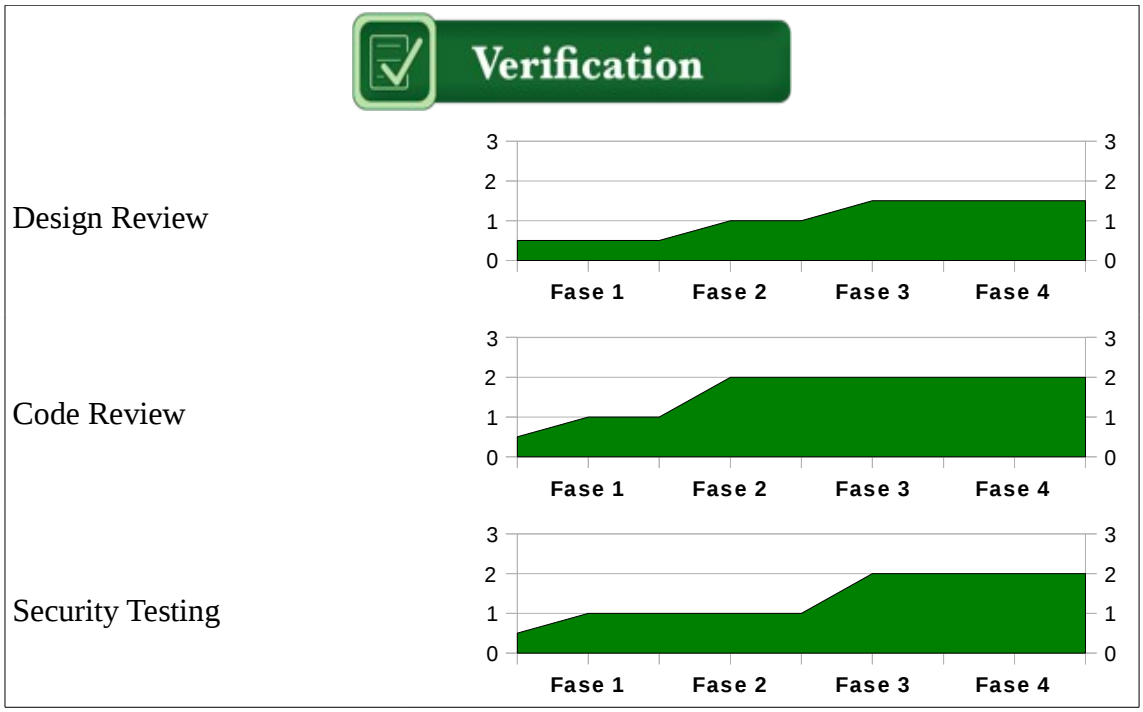
Figur 32 viser utbedring av sikkerheitsaktivitetane som inngår i sjølve konstruksjonen av systema. Det eksisterer i dag eit visst bilde av moglege truslar mot JAFU-systema. Med utgangspunkt i dette blei det spesifisert enkelte krav og teknologiar for å i

møtekome desse truslane [1, 2, 3]. Likevel er ikkje dette omfattande nok til at ein kan ha god nok bredde i desse sikkerheitsaktivitetane. Innan utgangen av fase 2 skal det derfor vere på plass grunnleggande prosessar for å handtere desse problema. Trusselvurdering og spesifikasjon av sikkerheitskrav bør vidare utbedrast etter dette.



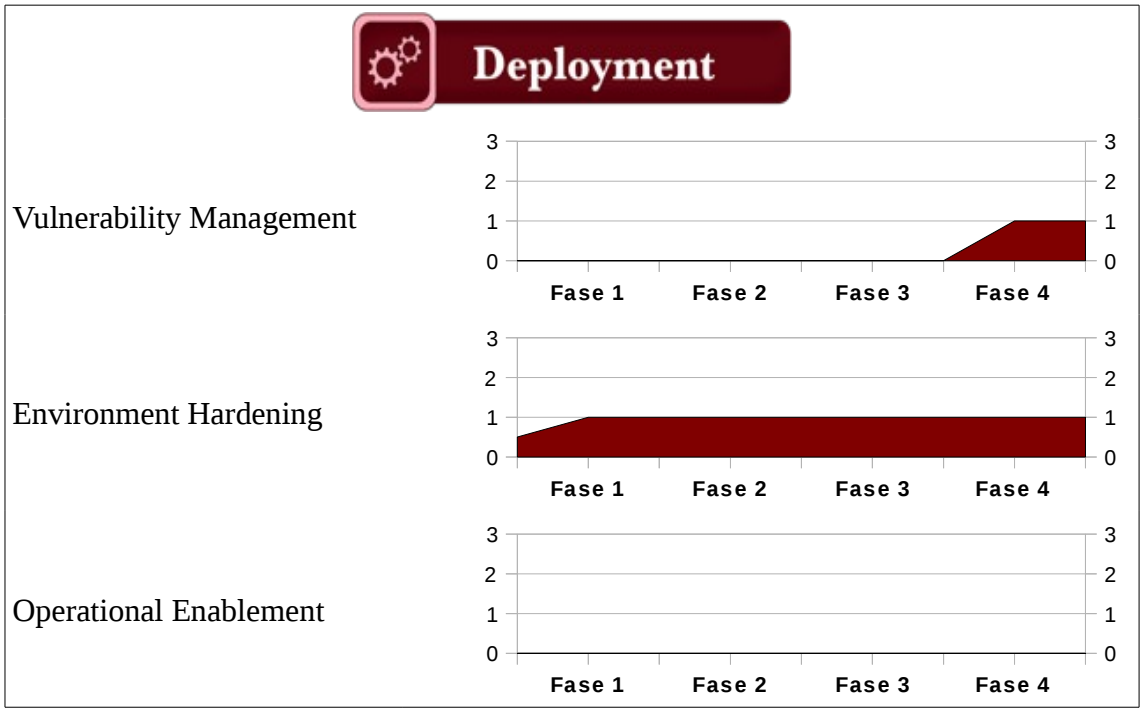
Figur 32: Construction - Vegkart for dei fire neste fasane

Verifikasjon (figur 33) inneber å passe på at design og implementasjon er av høg nok kvalitet til at ein kan ta ibruk systema på ein sikker og ansvarsfull måte. Målet er å nå modenheitsnivå 2 på *Code Review* og *Security Testing*, samt 1+ (1,5) på *Design Review*.



Figur 33: Verification - Vegkart for dei fire neste fasane

Sidan JAFU sjølv vedlikeheld og køyrer systema som er i bruk må ein ha på plass grunnleggande prosedyrer for å passe på at operasjonsmiljøet er i orden, samt at ein kan handtere eventuelle sikkerheitsproblem i versjonar av systema som er i bruk. Plan for forbedring av aktivitetar i denne sammenhen er i figur 34.



Figur 34: Deployment - Vegkart for dei fire neste fasane

6.2.2 Fase 1: (0 – 3 mnd) – Utdanning og planlegging

Når nye studentar går inn i JAFU-prosjektet vil mange av dei ha manglande kompetanse og erfaring innanfor programvaresikkerheit. I tillegg vil mange av dei ha lite eller ingen kjennskap til systema, konsepta og teknologiane dei skal jobbe med i organisasjonen.

Fokus i første fase vil derfor vere på å utdanne nye og gamle studentar innanfor sikkerheit og gjere dei kjende med sikkerheitsutfordringane i organisasjonen. Dei må også få kjennskap til systema, noko som ein kan oppnå gjennom vurdering og evaluering av masteroppgåver, kildekode og grunnleggande sikkerheitstesting av JAFU-systema og liknande system.

Nye studentar i JAFU har moglegheit til utdanning innanfor sikkerheit gjennom kurs ved Universitetet i Bergen, som INF226 – Programvaresikkerheit [25] og INF143 – Tryggleik i Distribuerte System [28]. For studentar som ikkje har moglegheit til å ta desse kursa kan det leggast opp spesialpensum for å bygge kompetanse innanfor sikkerheit.

I tillegg bør ein gjennomføre kodeanalyse basert på eksisterande sikkerheitskrav, og vurdere i kor stor grad koden tilfredsstiller desse krava, samt påpeike eventuelle problem som må handterast. OWASP Code Review Guide er eit godt utgangspunkt for dette arbeidet [30]

Sikkerheitstesting av systema vil innebere spesifisikasjon av kva som skal testast, samt grunnlegande penetreringstesting av applikasjonane. Ei anbefaling her kan vere å ta i bruk OWASP Testing Guide [20].

Environment hardening i denne fasen vil innebere å spesifisiere nødvendig programvare i ein køyrbar applikasjon. Dette inneber operativsystem, applikasjonsserver, database, nødvendige bibliotek og teknologiar [43]. Desse bør også vedlikehaldast gjennom å ta i bruk viktige sikkerheitsoppdateringar og liknande.

Strategy & Metrics

- Vurdering av overordna risikoprofil for JAFU
- Bygging og eventuell endring av program for sikkerheitsaktivitetar i organisasjonen.

Education & Guidance

- Teknisk utdanning av prosjektmedlemmar innanfor applikasjonssikkerheit
- Utgreie og vedlikehalde tekniske retningslinjer

Code Review

- Produksjon av ei liste med dei viktigaste potensielle sikkerheitsproblema JAFU står overfor, basert på eksisterande sikkerheitskrav.
- Vurdering av kildekode i høgrisikomodular basert på den produserte lista med problem ein kan stå ovanfor.

Security Testing

- Utarbeiding av *test cases* basert på eksisterande sikkerheitskrav.
- Penetreringstesting av applikasjonane i JAFU.

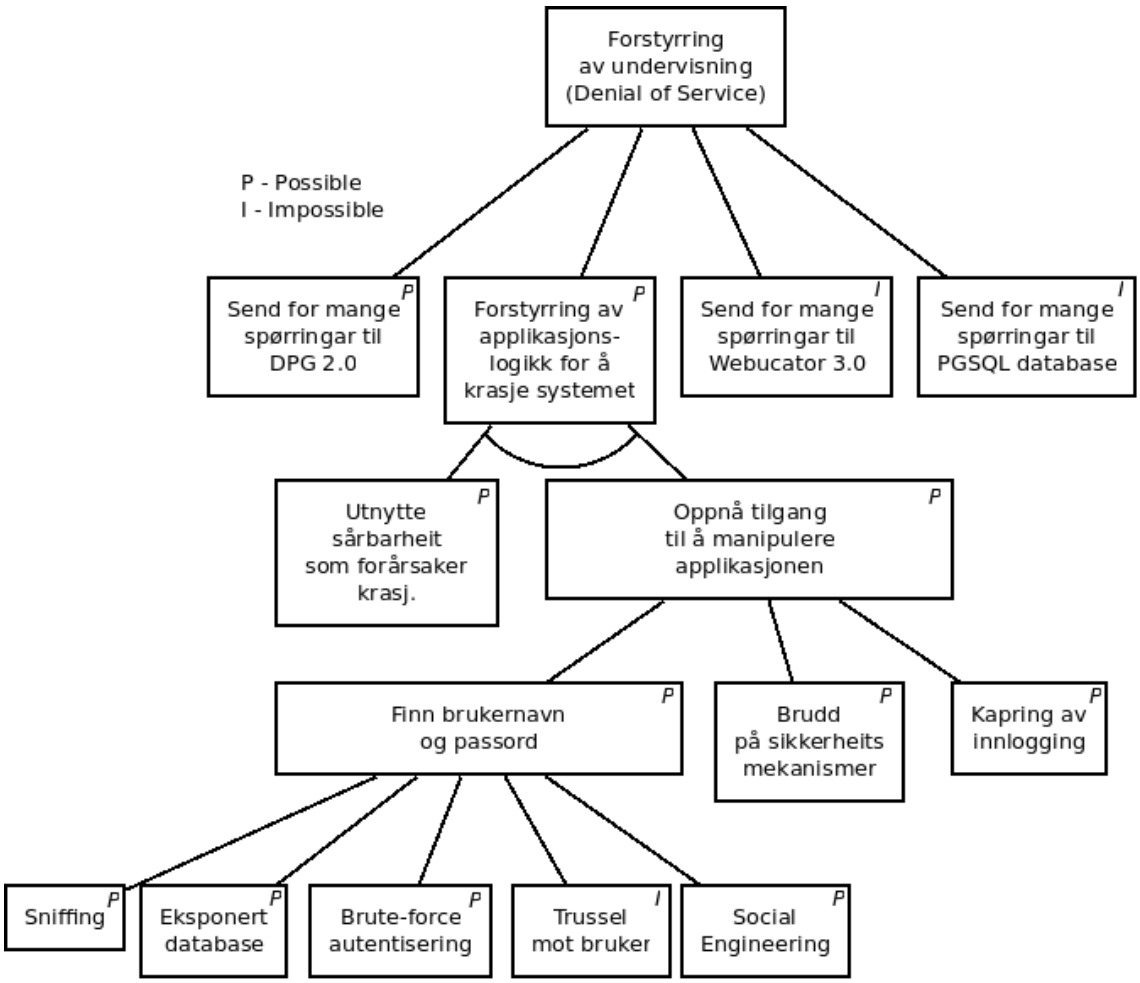
Environment Hardening

- Spesifikasjon av operasjonsmiljø
- Identifikasjon og installasjon av kritiske sikkerheitsoppgraderingar

6.2.3 Fase 2: (3 – 6 mnd) Design og analyse

I denne fasa inngår vurdering av truslar, spesifikasjon av sikkerheitskrav, etablering av teknologiar og rammeverk for ein sikker arkitektur, vurdering av design og vidare kodeanalyse.

Trusselvurdering har fram til no vore svært mangelfull. Enkle trusselmodellar for applikasjonane i JAFU er nødvendig. I følge [43] er angrepstre (en. *Attack trees)* eit godt utgangspunkt for dette. Konseptet med angrepstre blei først lagt fram av Bruce Schneier i Dr. Dobbs Journal i 1999 [38]. Formålet er å etablere kva moglege truslar eit system står overfor. Eit angrepstre inneheld ein rot-node, som respresenterer målet med eit eventuellt angrep. Ut i frå denne rot-noden er greiner som representerer moglege framgangsmåtar for å oppnå målet [38]. Figur 35 er eit eksempel på eit angrepstre.



Figur 35: Eksempel - Angrepstre

Potensielle angriparar bør også vurderast, enten dette inneber interne truslar (ansatte, prosjektmedlemmar o.l.) eller eksterne truslar (kriminelle, skadeleg programvare o.l.) [43].

Basert på den funksjonaliteten ein er ute etter i systema bør ein også etablere sikkerheitskrav for at gitt funksjonalitet er trygg. Desse krava bør samlast i lag med dei funksjonelle krava i ein formell kravspesifikasjon. Ein bør også vurdere kva forventningar desse krava vil ha med utgangspunkt i sikkerheit [43].

Design review i denne fasa går ut på å identifisere angrepsoverflata til JAFU-systema. Dette er punkt der applikasjonen tek imot data. Dette kan innebere HTTP-spørringar, informasjonskapslar, HTML-skjema, databasespørringar, konfigurasjonsfiler og liknande.

Kodeanalyse bør vidare utvidast i denne fasen for å inkludere verktøy for automatisert kodeanalyse. Dette kan vere verktøy som Fortify [23] og Graudit [44]. I tillegg bør dette

inkluderast i utviklingsprosessen. Dette kan gjennomførast ved at verktøya blir køyrde kvar gong ein bygger eit prosjekt eller ved jamne tidsintervall basert på kode i SVN brønnen [43].

Threat Assessment

- Bygging og vedlikehald av applikasjonsspesifikke trusselmodellar.
- Identifikasjon av moglege angriparar

Security Requirements

- Produser sikkerheitskrav ut i frå ynskja funksjonalitet i JAFU-systema.
- Utarbeiding av sikkerheitskrav frå eksterne retningslinjer og standarder

Secure Architecture

- Opprett ei liste med anbefalte rammeverk og teknologiar som skal nyttast i JAFU
- Ta i bruk forskjellige prinsipp for sikkerheit i design

Design Review

- Identfisier angrepsoverflata til JAFU-systema
- Analyser design mot kjende sikkerheitskrav

Code Review

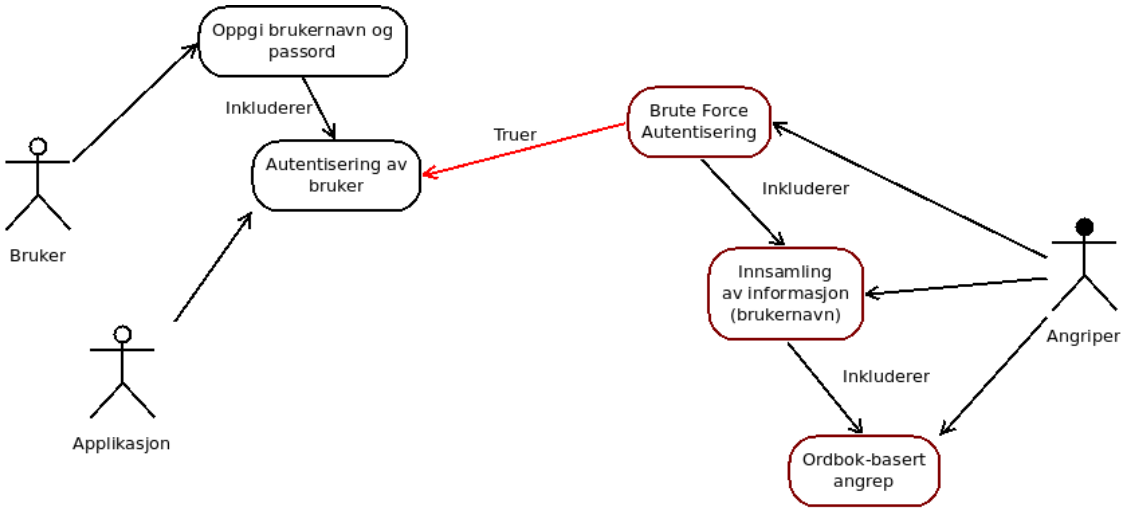
- Bruk verktøy for automatisert kodeanalyse
- Integrer kodeanalyse inn i utviklingsprosessen

6.2.4 Fase 3: (6 – 9 mnd) Forbetring av konstruksjon og verifikasjon

Denne fasa vil forbetre trusselanalyse og spesifikasjon av sikkerheitskrav, samt forbetre verifiseringsprosessen. Her vil det også forbedrast utdanning av prosjektmedlemmar basert på dei rollene dei vil ha i JAFU, og det arbeidet dei skal gjere.

For vidare utdanning kan dette innebere spesifikk utdanning og anbefalt litteratur for den spesifikke rolla prosjektmedlemmar har i JAFU. Dette er igjen avhengig av kva arbeid dei skal utføre som ein del av sine respektive masteroppgåver eller generelle oppgåver i JAFU.

Utviding av trusselvurdering kan vere å ta i bruk abuse cases [43]. Dette er motsetninga til use cases. I staden for å kun vise ynskjelege funksjonar i systema vil diagramma i tillegg vise moglege framgangsmåtar for angrep. Eksempel på abuse case kan vi sjå i figur 36.



Figur 36: Abuse case - eksempel

I tillegg bør det tas i bruk eit system for å rangere truslane basert på alvorlegheit. Eit rangeringsysten kan til dømes vere ei høg-medium-lav rangering av truslane. Forskjellige typer system kan nyttast, men i følge [43] bør det ikkje inkludere meir enn fem kategoriar. Til samanlikning anbefaler [21] ei rangering basert på sannsynlegheit med ein skala som går fra 1 til 6. Uavhengig av kva system som blir brukt bør data i frå denne aktiviteten nyttast i utviklingsprosessen for å prioritere tiltak for motvirking av truslane.

Sikkerheitskrav bør utvidast med ei matrise som viser tilgangskontroll. Ressursar i systemet bør vere på ein akse, og relevante roller bør vere på den andre. Denne skal så fyllast ut for å vise kva roller som skal ha tilgang til kva ressursar. Her er det og viktig å etablere kva moglegheiter rollene har til å lese, endre, oppdatere og slette ressursar i systemet [43]. Vidare bør sikkerheitskrav utvidast med kunnskap om risikoprofilen til JAFU. I tillegg bør ein dra inn kunnskap frå aktivitetar som design review, kjende truslar, kodeanalyser og sikkerheitstesting [43].

Design review bør forbedrast i form av ei formell evaluering av sikkerheitsmekanismer som tilgangskontroll, autentisering, inndatavalidering og i kor stor grad desse mekanismane kan i møtekome kjende sikkerheitskrav [43]. Dette kan til dømes gjerast felles for studentar som arbeider med dei aktuelle mekanismane.

For forbedring av sikkerheitstesting kan ein sette opp automatiserte testar, for eksempel unit testing og ved bruk av verktøy for automatisk penetreringstesting som Googles Skipfish [45]. Resultat frå slike verktøy må naturlegvis evaluerast manuellt, for

eksempel ved å gjennomføre *proof-of-concept* angrep for å luke ut falske positiver.

Education & Guidance

- Gjennomfør rolle-spesifikk kompetansebygging innanfor sikkerheit.

Threat Assessment

- Sett opp abuse-case modellar for JAFU-systema
- Benytt eit system for å vektlegge truslar mot JAFU-systema

Security Requirements

- Sett opp matriser for tilgangskontroll til ressursar og funksjonar i JAFU-systema.
- Spesifiser sikkerheitskrav basert på kjend risiko.

Design Review

- Inspeksjon av sikkerheitsmekanismer for grad av imøtekomming av sikkerheitstkrav.

Security Testing

- Bruk av automatiserte verktøy for sikkerheitstesting
- Integrasjon av sikkerheitstesting i utviklingsprosessen

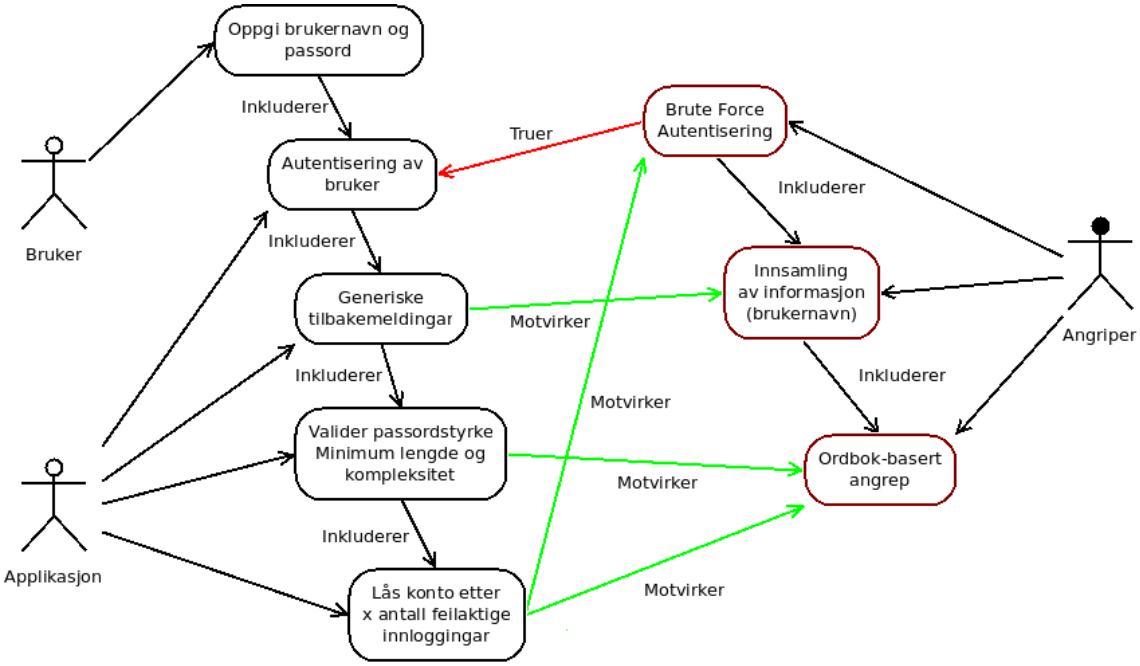
6.2.5 Fase 4: 9 – 12 månadar

Denne fasa tek for seg forbedring av strategi, trusselvurdering, samt inkludering av prosedyrer for å handtere sikkerheitsproblem i system som er i bruk.

Utbedring av strategi inneber her å klassifisere data og applikasjonar basert på risiko. Dette inneber ei vurdering frå høg til medium til lav. [21] har også interessante rangeringar av risiko for ressursar i systema basert på truslar og sårbarheiter i systema. Gitt at ein har rangert truslar, sårbarheiter og ressursar med numeriske verdiar kan risiko for ein gitt ressurs uttrykkast med følgande funksjon:

$$Risiko(A) = SUM[Trussel * Sårbarheit](A) * Ressursverdi(A)$$

Trusselvurderinga vil også her utvidast til at modellane inkluderer truslar frå tredjeparts-komponentar som er i bruk. For kvar komponent bør ein lage profilar for angriparar basert på potensiell sikkerheitsbrot relatert til desse komponentane. Ein bør også utvide modellane til å inkludere mottiltak for truslane som er identifiserte [43]. Døme for abuse cases er gitt i figur 37.



Figur 37: Abuse cases med mottiltak – eksempel

Vidare bør det også etablerast konaktpersonar for sikkerheitsproblem i applikasjonar som er i bruk. Dette vil vere personar som også har moglegheit til å sette i gong mottiltak for å motvirke eventuelle sikkerheitsproblem [43].

Strategy & Metrics

- Klassifikasjon av data og applikasjonar basert på risiko

Threat Assessment

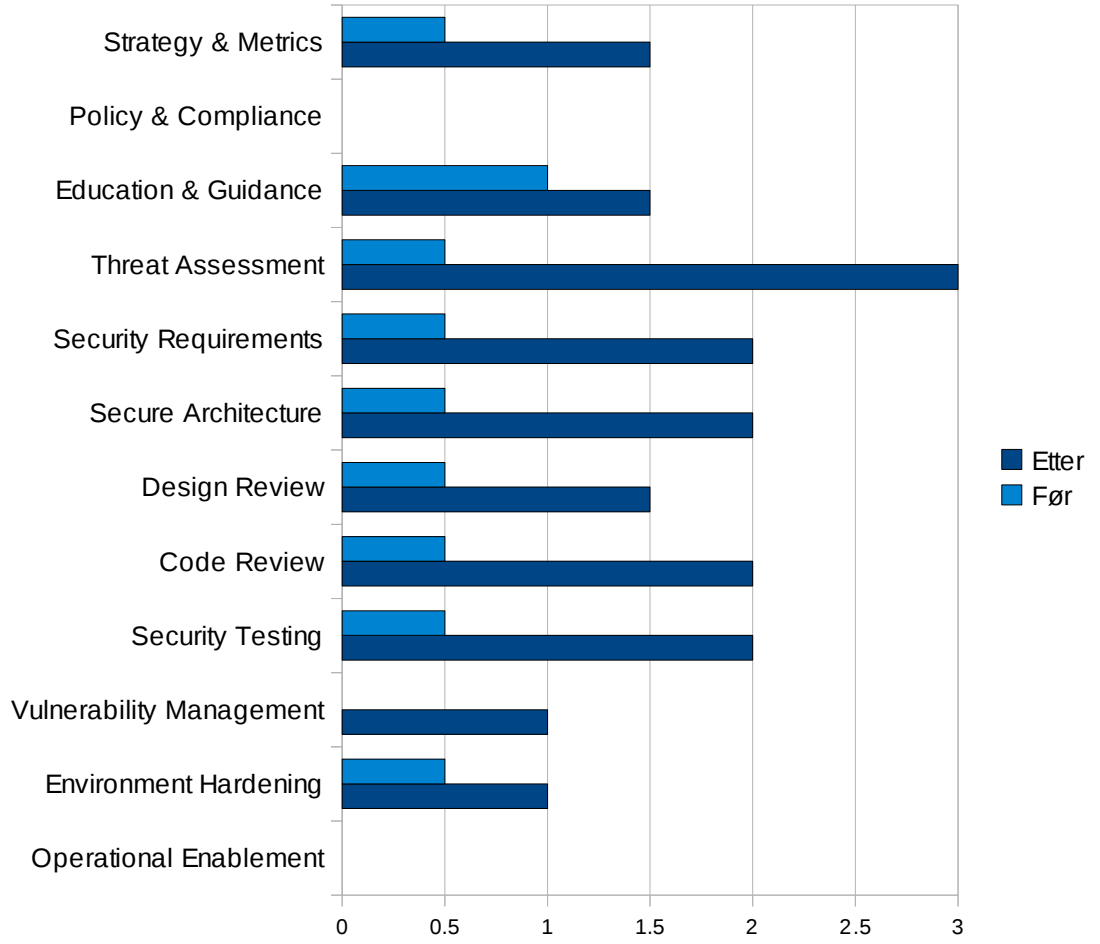
- Evaluer risiko frå tredjeparts-komponentar i JAFU-systema
- Utvid eksisterande modellar for trusselvurdering med mot-tiltak.

Vulnerability Management

- Kontaktpunkt for sikkerheitsproblem
- Informelt responsteam/person

6.2.6 Planlagt resultat

Figur 38 viser det planlagte resultatet for utbedring av sikkerheitsaktivitetane i JAFU med OpenSAMM. Dette er kun det planlagte resultatet. Den reelle vurderinga etter utbedringa er avhengig av i kor stor grad planen har blitt fulgt.



Figur 38: Planlagt resultat - OpenSAMM i JAFU

Etter kvart som ein arbeider med forbetring av sikkerheitsaktivitetane i JAFU er det ikkje usannsynleg at organisasjonen endrar seg på ein slik måte at det blir nødvendig å endre på planen for utbedringa.

Som ein ser ut i frå figur 38 bør det forekomme ei betydelig forbedring av sikkerheitssituasjonen i JAFU og dei tilknytta systema i organisasjonen. Dei mest kritiske område for forbetring basert på påviste sikkerheitsproblem i dei tidlegare systema vil vere under konstruksjon og verifikasjon. Uavhengig av i kor stor grad denne planen blir tatt ibruk bør ein forbetre prosedyrene rundt trusselvurdering, sikkerheitskrav, design review, kodeanalyse og sikkerheitstesting som ein integrert del av utviklingsprosessen.

Desse aktivitetane bør integrerast i utviklingsprosessen i JAFU, både for vidareutvikling av DPG 2.0, og for utvikling av nye system som innleveringssystem, redigeringsverktøy for presentasjonsmønster og liknande.

7

7 Konklusjon og evaluering

7.1 Oppsummering

Denne oppgåva har evaluert sikkerheita til systema i JAFU-prosjektet ved Universitetet i Bergen. I løpet av analysa vart det påvist ei rekke omfattande sikkerheitsproblem i systema som blant anna cross-site scripting, svak autentisering, tenestenekting, brot på tilgangskontroll, usikker lagring og usikker generering av sensitiv data.

For å løyse desse problema var det ynskjeleg å nytte rammeverk og teknologiar utvikla spesielt for dette formålet i motsetning til å lage løysingane sjølv. Dette var på grunn av at det er svært utfordrande og komplisert å skrive slike verktøy. Det er mykje som kan bli gjort feil, og eventuell testing av eigne løysingar vil vere avgrensa. Spring og OWASP tilbyr verktøy for å i møtekome ein del av sikkerheitsbehova på ein enkel måte. Verktøya dei tilbyr har også vore gjennom omfattande testing av ekspertar frå heile verda. Løysinga på ein del av problema innebar i hovudsak validering og sanitering av inndata, samt enkoding av utdata.

I tillegg til vurdering og løysing av sikkerheitsproblema som var påviste i systema vart det også gjort ei overordna vurdering av sikkerheitssituasjonen i JAFU. Til dette formålet blei OpenSAMM nytta, og eit bilde vart skapt for den opprinnelege sikkerheitssituasjonen. I tillegg vart det lagt opp ein strategi for korleis ein kan forbetre sikkerheitssituasjonen i JAFU. Dette innebar ei rekke sikkerheitsrelaterte aktivitetar som skulle gjennomførast og forbedrast over fire periodar på tre månadar kvar. Det har blitt demonstrert korleis ein slik strategi kan forbetre sikkerheitssituasjonen i JAFU med utgangspunkt i målingar ved hjelp av OpenSAMM.

7.2 Evaluering av teknologiar og rammeverk

Forskjellige teknologiar og rammeverk har blitt evaluerte og nytta for å i møtekome problema JAFU står overfor. Her blir det gjort ei vurdering av desse verktøya, teknologiane og rammeverka.

7.2.1 ESAPI

ESAPI har vist seg å vere eit svært nyttig verktøy. Mogleg bruk av verktøyet er langt breiare enn dei funksjonane som har blitt nytta i DPG 2.0 og Webucator 3.0. Verktøyet er lett å ta i bruk og har stor bredde i moglege funksjonar. ESAPI blir nytta av svært mange organisasjonar, så eventuelle svakheiter vil vere svært problematiske.

Dokumentasjonen i ESAPI er til dels manglande. Mykje av det er uoversikteleg, og ein del dokumentasjon for forskjellige teknologiar (som .NET og Java EE) er tilgjengeleg på samme plass. Det eksisterer også få brukarveiledningar for korrekt bruk av ESAPI. Dette er hovudsakeleg på grunn av at det ville vere vanskeleg å produsere generiske veiledningar for applikasjonar som vil vere veldig variable i oppbygging og funksjonalitet.

> *I think the decision about which security controls cannot or should not be used is requires quite a bit more context than is possible generically. Can you find documentation on the web about where deadbolt locks cannot or should not be used? We'll try to give people reasonable guidance about when and how to use the controls in ESAPI.*
>
> *- Jeff Williams, OWASP [53]*

For utviklarar som skal gjere ei vurdering for kva verktøy som passar deira behov er det viktig å få god innsikt i korleis verktøya fungerer og i kva grad dei er egna til å handtere problema ein organisasjon står overfor. Denne vurderinga blir utfordrande med manglande dokumentasjon.

Feil bruk av ESAPI kan dessutan medføre sikkerheitsproblem. I 2009 vart det påvist problem med data som blir enkoda for JavaScript [54]. Når ein enkoder data som skal inn i JavaScript funksjonar som document.write og eval blir dette dekoda når JavaScript skal bruke gitte data. På denne måten blir sikkerheitskontrollen omgått og enkodinga har inga effekt.

Dette er ikkje eit problem med ESAPI, men heller eit problem om korleis feil bruk kan føre med seg nedsett sikkerheit. Det er ingen endringar i ESAPI som kan stanse JavaScript i å dekode gitt data, så løysinga vil vere å dokumentere problemet godt nok og gi utviklarane som skal bruke ESAPI kunnskap om korrekt bruk [54]. I lang tid var det svært lite dokumentasjon om dette problemet på OWASP sine nettsider. Den

dokumentasjonen som er tilgjengeleg i dag gir heller ikkje ei komplett oversikt over korleis problemet utartar seg og korleis ein kan unngå slike feller.

7.2.2 AntiSamy

AntiSamy var nok eit verktøy frå OWASP som vart nytta her. Det var svært nyttig å kunne ta i bruk eit verktøy for avansert filtrering av HTML og CSS, som i tillegg var modent, godt dokumentert og med fleire moglegheiter for tilpassing til den type data ein ynskjer å ta imot i DPG 2.0.

7.2.3 Spring Security

Spring Security er svært nyttig for autentisering og tilgangskontroll. Ein fordel med bruk av Spring Security i forhold til, til dømes, ESAPI er at ein har moglegheit til å knytte det opp mot ei større samling med ulike teknologiar som SQL datbasar, LDAP, OpenID, X.509-sertifikat med meir [10]. Så avhengig av korleis behov for brukarhandtering endrar seg i framtida vil sannsynlegvis Spring Security kunne nyttast for autentisering og tilgangskontroll.

Spring Security har derimot ingen spesifikke mekanismar for å handtere vanlege problem i vevapplikasjonar som XSS og SQL innsprøyting. Ein skulle gjerne sett at det ville vere mogleg å samle både autentisering, tilgangskontroll, validering og sanitering av inndata samt enkoding av utdata under eit og samme rammeverk som kan integrerast med Spring og nytte seg av *dependency injection* [10].

7.2.4 Fortify

Bruk av Fortify forenkla i stor grad evalueringsprosessen. Verktøyet er enkelt å bruke og gjer det enkelt å evaluere eventuelle manglar i koden. Likevel bør ein ikkje ha for mykje tillit til automatisk kodeanalyse, spesielt innanfor sikkerheit.

Eit problem er at verktøyet produserer store mengder med falske positiver. Det belyser sikkerheitsproblem som ikkje er reelle, og dette kan medføre at kodeanalyse blir tidkrevande. Statisk kodeanalyse med fokus på sikkerheit må likevel ha ein relativt lav terskel for rapportering av problem [19]. Det er langt meir alvorleg om problem eksisterar som ikkje blir belyste av verktøyet.

Blant anna vart problem som persistent XSS i både DPG 2.0 og Webucator 3.0 ikkje belyst av Fortify. Verktøyet hadde problemer med å følge data frå HTML-skjema, via kildekode og til databasen. Verktøyet klarer ikkje sjå den fulle konteksten av korleis data blir behandla.

Dette gjer det problematisk dersom ein sett for mykje tillit til det resultatsettet Fortify returnerer. Det vil vere interessant å sjå korleis nyare versjonar av verktøyet fungerer og om det er i stand til å finne ein del av dei problema som i denne analysa ikkje var belyst

av verktøyet.

7.3 Utfordringar

Gjennomgang av kode og dokumentasjon som andre har skrive etter at desse personane har forlatt prosjektet. Ofte kjem det ikkje klart fram i koden og dokumentasjone kva denne personen tenkte når arbeidet blei gjort, og kva han eller ho ville oppnå. Mykje av koden var udokumentert og det vart lagt mykje arbeid i å finne ut nøyaktig korleis delar av applikasjonane fungerer.

Evaluering av riktig framgangsmåte for løysing av sikkerheitsproblem er også utfordrande. Ofte eksisterer det fleire forskjellige metodar for å i møtekomme sikkerheitsproblema, og riktig val er avhengig av ei rekke faktorar som varierer frå applikasjon til applikasjon, delsystem til delsystem og funksjon til funksjon.

7.4 Forslag til vidare arbeid

Det er framleis mykje arbeid igjen på sikkerheitsdelen i JAFU-systema. Etter kvart som systema utviklar seg vidare vil nye sikkerheitsbehov dukke opp. Her blir det no presentert forslag til framtidig arbeid med DPG 2.0 og Webucator 3.0

7.4.1 Vurdering av sikkerhet for Ajax

Peder Skeidsvoll implementerte AJAX (Asynchronous JavaScript and XML) som ein del av si masteroppgåve [65]. AJAX inneber ein del nye sikkerheitsutfordringar. [66] presenterer til dømes 10 vanlege sikkerheitsproblem i AJAX.

AJAX inneber nye metodar for utvikling og presentering av informasjon til brukarane, noko som gjer at eksisterande utviklingspraksis og sikkerheitstesting må utvidast og endrast for å i møtekome nye sikkerheitsbehov. Det kan vere interessant å gjere ei vurdering av dei potensielle sikkerheitsproblema AJAX fører med seg til DPG.

7.4.2 Utvida logging

Det er behov for betre logging i JAFU. Applikasjonane som er i bruk må utvide logginga for å inkludere fleire hendingar, samt strukturere desse på ein måte som effektiviserer ein eventuell gjennomgang av loggane.

Logging i dag er med fokus på debugging. Det er altså lite aktivitetslogging i bruk i applikasjonane, noko som gjer det vanskeleg å skape seg eit godt bilde av hendelsesforløp i applikasjonane med utgangspunkt i kva brukarane foretar seg.

Ein bør implementere betre logging av sikkerheitsrelevant informasjon, som blant anna kva IP-adresse brukarane logger inn frå, kva brukarnamn som er knytta til kvar enkelt hending i loggen, samt gi advarslar om moglege forsøk på angrep (til dømes når

validator feiler eller kontoar blir låste etter eit gitt antall innloggingsforsøk). ESAPI kan bidra med strukturert logging av sikkerheitshendingar.

DPG 2.0 og Webucator 3.0 loggar i dag også mykje informasjon med hensikt på debugging. Dette er informasjon som ikkje bør finne veien vidare i produksjonssystem, då den som regel ikkje er spesielt relevant for situasjonen [19].

7.4.3 Nytt brukarhandteringssystem

Framtida til Webucator 3.0 som brukarhandteringssystem er usikker. Det er mogleg at valet vil falle på å nytte Universitetet i Bergen sin brukardatabase, og gjennomføre innloggingar mot deira tenarar.

Likevel må brukarane som skal ha tilgang til for eksempel DPG verte knytt opp mot dei spesifikke rollene dei skal ha. Av brukarane til universitetet bør det også utarbeidast mekanismar for at kun eit subsett skal ha tilgang til DPG 2.0, høvesvis dei som er tilknytta fjernundervisninga.

Alt dette inneber ei ombygging av autentiseringa i DPG 2.0. Spring Security kan vidare nyttast og kan koplast opp mot blant anna LDAP som blir brukt av Universitetet i Bergen i dag.

7.4.4 Vidare vurdering av sikkerheitsstrategiar

Etter kvart som JAFU endrar seg er det ikkje usannsynleg at sikkerheitsbehova til organisasjonen endrar seg. Nye studentar med varierande kompetanse trer til i prosjektet, og vidareutvikling medfører nye sikkerheitsutfordringar gjennom nye teknologiar som AJAX. I løpet av denne prosessen må sikkerheitsstrategiane til JAFU revurderast, og nye strategiar må utarbeidast for å i møtekome nye sikkerheitsbehov.

7.4.5 Vidare vurdering av sikkerheitssituasjonen i JAFU-systema

Etter kvart som systema utviklar seg og endrar seg er det ikkje usannsynleg at nye problem blir introduserte, og gamle problem som til no ikkje er oppdaga dukkar opp. Det er derfor nødvendig med vidare vurderingar av sikkerheitssituasjonen i applikasjonane til JAFU. Fleire vurderingar av samme system kan gi varierande resultat. Problem som ein person overser kan bli oppdaga av andre.

Bibliografi

[1] Kristian Skønberg Løvik *Webucator 3.0 Brukerhåndtering og aksesskontroll for DPG 2.0.* Masteroppgåve ved Universitetet i Bergen, August 2008

[2] Karianne Berg *Persistensproblematikk i Dynamic Presentation Generator* Masteroppgåve ved Universitetet i Bergen, Juli 2008

[3] Bjørn Christian Sebak *Dynamic Presentation Generator 2.0. Utvikling av ny dynamisk presentasjonsgenerator og presentasjonsmønsterspesifikasjon* Masteroppgåve ved Universitetet i Bergen, August 2008

[4] Bjørn Ove Ingvaldsen *Multimedia i Dynamisk Presentasjons Generator 2.0* Masteroppgåve ved Universitetet i Bergen, September 2008

[5] Khalid Mughal, Torill Hamre, Ynge Espelid *Composing Web Presentations using Presentation Patterns* Universitetet i Bergen 2006, Rapport nr. 331

[6] Sun Microsystems Java 6 Enterprise Edition

http://java.sun.com/javaee/

Sist besøkt mai 2010

[7] Apache Software Foundation, Apache Tomcat

http://tomcat.apache.org/

Sist besøkt mai 2010

[8] PostgreSQL 8

http://www.postgresql.org/

Sist besøkt mai 2010

[9] Apache Software Foundation, Apache Ant

http://ant.apache.org/

Sist besøkt mai 2010

[10] SpringSource, Spring Framework

http://www.springsource.org/

Mai 2010

[11] Bruce Schneier *Applied Cryptography: Protocols, Algorithms, and Source Code in C* Wiley, Oktober 1996

[12] Lars Hopland Nestås, Tobias Rusås Olsen *Dynamic Presentation Generator - Part II: Static Analysis* Rapport i INF226 ved Universitetet i Bergen, Haust 2008

[13] Eziz Annagurban, Trond Gjertsen *INF226 Project – Part 2* Rapport i INF226 ved Universitetet i Bergen, Haust 2008

[14] Eirik Ottesen, Peder Skeidsvoll *DPG 2.0 Prosjekt i INF226* Rapport i INF226 ved Universitetet i Bergen, Haust 2008

[15] Nazar Annagurban, Christian Pedersen *Static Analysis – System: Webucator 3.0* Rappport i INF226 ved Universitetet i Bergen, Haust 2008

[16] Ole Henning Vårdal, Geir Inge Imsland *Part II: Static Analysis of Dynamic Presentation Generator 2.0* Rappport i INF226 ved Universitetet i Bergen, Haust 2008

[17] Bjørn Petter Nilsen, Alfred Lysebraate *Sikkerhetsanalyse av Webucator 3.0* Rappport i INF226 ved Universitetet i Bergen, Haust 2008

[18] Dafydd Stuttart, Marcus Pinto *Web Application Hacker's Handbook* Wiley, Oktober 2007

[19] Brian Chess, Jacob West *Secure Programming with Static Analysis* Addison Wesley, 2007

[20] *OWASP Testing Guide v3.0*

http://www.owasp.org/index.php/Category:OWASP_Testing_Project

OWASP Foundation, 2008, Sist besøkt juli 2010

[21] Marcus Schumacher, Eduardo Fernandez-Buglioni, Duanye Hybertson, Frank Buschmann, Peter Sommerland *Security Patterns – Integrating Security and Systems Engineering* Wiley, 2006

[22] *OWASP Top Ten. The Ten Most Critical Web Application Security Risks*

http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Foundation, April 2010, Sist besøkt august 2010

[23] Fortify Software, Fortify Static Analysis Tool

http://www.fortify.com

Sist besøkt mai 2010

[24] University of Maryland, FindBugs

http://findbugs.sourceforge.net

Sist besøkt mai 2010

[25] Universitetet i Bergen, Programvaresikkerhet

http://www.uib.no/emne/INF226

Sist besøkt mai 2010

[26] *OWASP CLASP*

http://www.owasp.org/index.php/CLASP

OWASP Foundation, Mai 2010

[27] Wade Trappe, Lawrence Washington *Introduction to Cryptography with Coding Theory* Pearson Prentice Hall, 2006

[28] Universitetet i Bergen, Tryggleik i distribuerte system

http://www.uib.no/emne/INF143

Sist besøkt august 2010

[29] Ramez Elmasri, Shamkant B. Navathe *Fundamentals of Database Systems* Addison Wesley, 2007

[30] *OWASP Code Review Guide v1.1*

http://www.owasp.org/index.php/Code_review

OWASP Foundation, November 2008, Sist besøkt juli 2010

[31] Craig Walls *Spring in Action* Manning, 2008

[32] Nitesh Dhanjani, Brett Hardin, Billy Rios *Hacking – The Next Generation* O'Reilly, August 2009

[33] Paco Hope, Ben Walther *Web Security Testing Cookbook* O'Reilly, 2008

[34] Jan P. Monsch *Ruining Security with java.util.Random* Ipsplosion.com 2006

[35] Bruce Schneier *Passwords are not broken, but how we choose them sure is* The Guardian, 2008

[36] RainbowCrack Project

http://project-rainbowcrack.com/

Sist besøkt juli 2010

[37] TinyURL

http://www.tinyurl.com

Sist besøkt august 2010

[38] Bruce Schneier - *Attack Trees*

http://www.drdobbs.com/184411129

Dr. Dobbs Journal, Desember 1999, Sist besøkt August 2010

[39] *OWASP Encoding Project*

http://www.owasp.org/index.php/Category:OWASP_Encoding_Project

Sist besøkt august 2010

[40] Adam Kiezun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst *Automatic Creation of SQL Injection and Cross-Site Scripting Attacks* ICSE'09, Mai 2009

[41] *Denial of Service*

http://www.owasp.org/index.php/Denial_of_Service

OWASP Foundation, Mars 2010, sist besøkt august 2010

[42] *Path Traversal Attack*

http://www.owasp.org/index.php/Category:Path_Traversal_Attack

OWASP Foundation, sist besøkt august 2010

[43] Pravir Chandra med fleire – *Open Software Assurance Maturity Model*

http://www.opensamm.org/about/

[44] Graudit
http://www.justanotherhacker.com/projects/graudit.html
Sist besøkt august 2010

[45] Google Skipfish
http://code.google.com/p/skipfish/
Sist besøkt august 2010

[46] ESAPI
http://www.owasp.org/index.php/ESAPI
Sist besøkt august 2010

[47] THC Hydra
http://freeworld.thc.org/thc-hydra/
Sist besøkt august 2010

[48] Wireshark
http://www.wireshark.org/
Sist besøkt august 2010

[49] AntiSamy
http://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project
OWASP Foundation, sist besøkt august 2010

[50] TinyMCE
http://tinymce.moxiecode.com/
Sist besøkt august 2010

[51] *XSS (Cross Site Scripting) Prevention Cheat Sheet*
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
OWASP Foundation, Sist besøkt august 2010

[52] Man-in-the-middle attack
http://www.owasp.org/index.php/Man-in-the-middle_attack
OWASP Foundation, Sist besøkt august 2010

[53] ESAPI Mailing List
https://lists.owasp.org/pipermail/esapi-user/2010-January/000033.html
Sist besøkt august 2010

[54] *Bypassing OWASP ESAPI XSS Protection inside Javascript*
http://securethoughts.com/2009/08/bypassing-owasp-esapi-xss-protection-inside-javascript/

SecureThoughts.com, Sist besøkt august 2010

[55] Bruce Schneier *Computer Security: Will We Ever Learn?*

http://www.schneier.com/crypto-gram-0005.html#1

Information Security, April 2000, Sist besøkt august 2010

[56] Michael Oreld *Google erkjenner Youtube-hack*

http://www.idg.no/computerworld/article172264.ece

Computerworld, Juli 2010, Sist besøkt august 2010

[57] ESAPI kildekode

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/?r=983

Sist besøkt august 2010

[58] Red Hat Common Vulnerability Scoring System

http://www.redhat.com/security/updates/cvss/

Sist besøkt august 2010

[59] BBCode Wikipedia artikkel

http://en.wikipedia.org/wiki/BBCode

Sist besøkt august 2010

[60] Jason Li *Validation Rich User Content: Using OWASP AntiSamy* Presentasjon ved AppSec India, August 2008

https://www.owasp.org/images/9/9d/AppSecIN08-ValidatingRichUserContent.ppt

Sist besøkt, august 2010

[61] Cigital Touchpoints

http://www.cigital.com/training/touchpoints/

Sist besøkt august 2010

[62] Microsoft SDL

http://www.microsoft.com/security/sdl/

Sist besøkt august 2010

[63] OWASP CLASP

http://www.owasp.org/index.php/Category:OWASP_CLASP_Project

Sist besøkt august 2010

[64] Pravir Chandra *Software Assurance Maturity Model* Presentasjon, OWASP MSP 2009

http://vimeo.com/6495398

Vimeo.com, Sist besøkt august 2010

[65] Peder Lång Skeidsvoll *Støtte for rike klienter i Dynamic Presentation Generator* Masteroppgåve ved Universitetet i Bergen, Juni 2010

[66] Shreeraj Shah *Top 10 Ajax Security Holes and Driving Factors*

http://www.net-security.org/article.php?id=956

Net-security.org, November 2006. Sist besøkt august 2010

[67] Oracle Java 6 JavaDoc

http://download-llnw.oracle.com/javase/6/docs/api/java/util/Random.html

Sist besøkt august 2010