NTNU
Det skapende universitet

Sikkerhetstest av "Sikker Varsling"

Øystein Sekse Øie

Master i datateknikk
Oppgaven levert: Juni 2008
Hovedveileder: Torbjørn Skramstad, IDI
Biveileder(e): Skule Johansen, Kantega

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Oppgavetekst

"Sikker Varsling" er en varslingstjeneste som arbeidstakere kan bruke for å varsle om kritikkverdige forhold på arbeidsplassen. Varslingen blir gjort over Internett og sikrer arbeidstakerne 100 % anonymitet. Denne tjenesten ble utviklet i forbindelse med nye lovregler fra 1. januar 2007, som omhandler de ansattes ytringsfrihet på arbeidsplassen.

Oppgaven går ut på å gjennomføre en sikkerhetstest av systemet "Sikker Varsling", hvor design, kode og sluttprodukt blir testet og evaluert. Under testingen vil forskjellige tester og testverktøy bli vurdert. Dette skal gi grunnlag for utvikling av en testprosedyre for "Sikker Varsling" og tilsvarende tekniske løsninger.

Oppgaven skal også inneholde en teoridel, som omhandler relevant litteratur om testing.

Oppgaven gitt: 15. januar 2008
Hovedveileder: Torbjørn Skramstad, IDI

Sammendrag

Sikkerhet blir stadig mer aktuelt innen programvareutvikling. En av hovedgrunnene til dette er at organisasjoner har blitt mer bevisst på hvilke økonomiske konsekvenser sikkerhetsfeil kan medføre. Økningen i rapporterte sårbarheter har også satt fokus på programvaresikkerhet. Den beste fremgangsmåten for å unngå sårbarheter i programvare er å tenke sikkerhet så tidlig som mulig i utviklingsprosessen. Det finnes flere teknikker og metoder som kan inkluderes i forskjellige faser av utviklingsprosessen, og de mest sentrale er beskrevet i denne rapporten.

Økt fokus på programvaresikkerhet har også ført til økt fokus så sikkerhetstesting. Dette prosjektet har gjennomført en sikkerhetstest av websystemet ”Sikker Varsling”. ”Sikker Varsling” er en varslingstjeneste for ansatte i en organisasjon, hvor de kan varsle anonymt om forhold på arbeidsplassen. Systemet skal med andre ord behandle sensitiv informasjon, og det er dermed svært viktig at det ikke inneholder sikkerhetsfeil som kan avsløre denne informasjonen.

Testingen har fulgt en risikobasert fremgangsmåte. Dette innebærer at risikoer rundt ”Sikker Varsling” har blitt identifisert og rangert, for deretter å teste de mest kritiske risikoene. Testingen har tatt i bruk forskjellige testverktøy som også har blitt vurdert i dette prosjektet. Resultatene fra testingen har avdekket en del sikkerhetsfeil, hvor enkelte har blitt vurdert som kritiske for systemet. Resultatene har også lagt grunnlaget for en testprosedyre som prosjektet mener kan brukes i fremtidig sikkerhetstesting av lignende websystemer.

Forord

Denne masteroppgaven er gjennomført av Øystein Sekse Øie vårsemesteret 2008, og er siste del av masterutdanningen innen datateknikk ved Norges Teknisk-Naturvitenskaplige Universitet (NTNU). Oppgaven er gitt av Kantega AS, og utført sammen med NTNU.

Jeg vil spesielt takke veilederen min ved Kantega AS, Skule Johansen, for verdifulle tilbakemeldinger og god støtte gjennom hele oppgaven. Jeg vil også takke veilederen min ved NTNU, Torbjørn Skramstad, for god hjelp med oppgaven. Til slutt vil jeg takke Kantega AS for lån av datamaskin og gode lunsjer.

Trondheim, 06. juni 2008

_____

Øystein Sekse Øie

Innholdsfortegnelse

Figurliste

Tabelliste

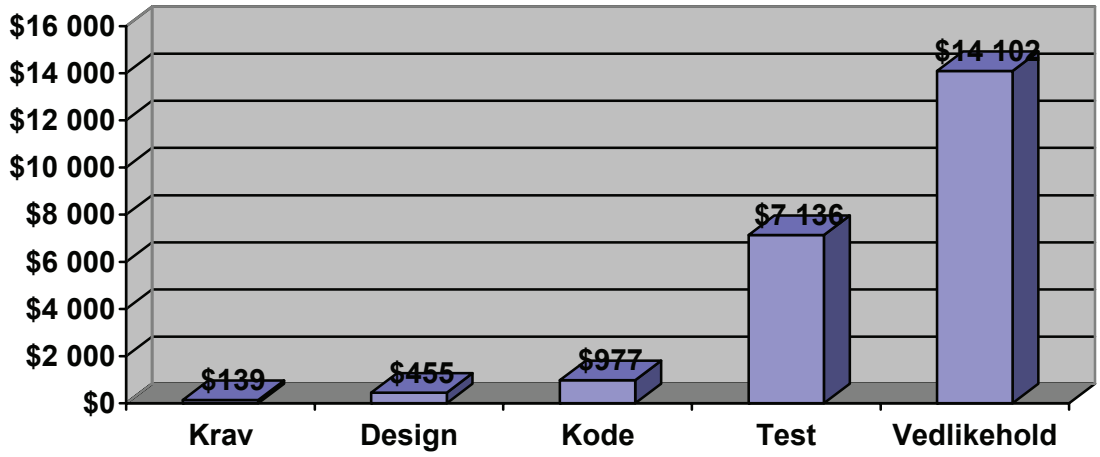
1. Introduksjon

1.1. Motivasjon

Mange forbinder sikkerhetstesting med testing av ferdig programvare. Etter at programvaren er ferdig utviklet, blir enkelte sikkerhetstester gjennomført for å sikre at produktet oppnår eventuelle sikkerhetskrav som er satt.

Motivasjonen bak denne oppgaven er å lage en testprosedyre som tar sikte på å innføre en sikkerhetstankegang gjennom hele utviklingsprosessen. En slik tankegang bør innarbeides allerede i kravspesifikasjonen, hvor det bør utarbeides egne krav for sikkerheten til programvaren. Ved å tenke og innarbeide sikkerhet gjennom hele utviklingsprosessen, øker sannsynligheten for sikrere programvareutvikling.

Sikker programvare betyr mindre sårbarheter og mindre sjanse for angrep på programvaren. Dette gir brukerne en trygghet når de bruker programvaren. Det er også til fordel for firmaet som utvikler programvaren, ved at de opparbeider seg et godt rykte, og samtidig bruker mindre tid og penger på vedlikehold av programvaren. Figur 1.1 viser hvor økonomisk fordelsmessig det er å finne feil i programvaren tidlig i utviklingsfasen. Tallene er tatt fra en studie gjort av Barry Boehm og Vic Basili[1], og viser kostnadene ved å rette opp feil i de forskjellige utviklingsfasene. Dette gjelder for programvarefeil generelt, og inkluderer derfor også sikkerhetsfeil i programvaren. Kostnadsøkningen i test- og vedlikeholdsfasen bør være et argument for å innføre sikkerhet som en del av hele utviklingsprosessen.



Figur 1.1: Kostnader ved å rette opp feil i forskjellige utviklingsfaser

Fagområdet rundt programvaresikkerhet har utviklet seg mye de siste årene, og det blir stadig mer litteratur som omhandler dette.[2] Dette har også bidratt til større oppmerksomhet rundt sikkerhetstesting, som igjen har ført til mer fokus på testmetoder og testverktøy for sikkerhetstesting av programvare. Det finnes uttalige testverktøy på Internett, og en av utfordringene bak denne oppgaven er å definere en kombinasjon av tester og testverktøy som bidrar til å utvikle en effektiv testprosedyre for webapplikasjoner.

1.2. Problemdefinisjon

Hovedfokuset i denne oppgaven er å gjennomføre en sikkerhetstest av systemet ”Sikker Varsling”. ”Sikker Varsling” er en varslingstjeneste som arbeidstakere kan bruke for å varsle om kritikkverdige forhold på arbeidsplassen. Varslingen blir gjort over internett og skal sikre arbeidstakerne 100 % anonymitet. Denne tjenesten ble utviklet i forbindelse med nye lovregler fra 1. januar 2007, som omhandler de ansattes ytringsfrihet på arbeidsplassen. Sikkerhetstesten går ut på å finne forskjellige typer tester, som på best mulig måte tester sikkerheten i systemet.

Et viktig poeng er at denne prosedyren også skal kunne brukes på tilsvarende tekniske løsninger. ”Sikker Varsling” blir dermed et utgangspunkt for hvilke tester og testverktøy som inngår i en slik testprosedyre. I prosessen med å bestemme hvilke tester og testverktøy som er mest aktuelle, er det gjennomført en risikovurdering for å finne potensielle risikoer i systemet. I tillegg til identifikasjon av risikoer, er de også blitt rangert slik at de mest kritiske risikoene har blitt vurdert først. En slik risikovurdering har også vært med på å avgrense oppgaven, ved at fokuset har vært på de delene av systemet som har høyest risiko.

Et annet poeng er at testingen i denne oppgaven blir gjort på et ferdigutviklet system. ”Sikker Varsling” er i slutten av testfasen og blir testet av blant annet Trondheim Kommune. Dette vil si at selve utviklingen av systemet er ferdig. Som nevnt i avsnittet over, er noe av motivasjonen bak oppgaven å innarbeide en sikkerhetstankegang fra begynnelsen av utviklingsprosessen. Derfor vil denne oppgaven også inkludere metoder som kan brukes i tidligere utviklingsfaser for lignende systemer. Disse metodene vil derimot bare bli teoretisk beskrevet, siden de ikke kan gjennomføres i praksis på grunn av at ”Sikker Varsling” er ferdigutviklet.

1.3. Omfang

Omfanget av sikkerhetstestingen er begrenset til en testinstans av ”Sikker Varsling”. Denne testinstansen vil naturlig nok avvike noe fra et produksjonsmiljø, men det ligger utenfor omfanget av denne oppgaven å gi en fullstendig oversikt over forskjellen mellom testmiljøet og et produksjonsmiljø. De forskjellene som er identifisert er dokumentert i avsnitt 5.1.

Sikkerhetstestingen er også begrenset til kun å gjelde for ”Sikker Varsling”. I tilfeller hvor applikasjonen tilgangsbeskyttes av andre systemer, blir sikkerhetsperspektivet et annet. Dette prosjektet har tatt utgangspunkt i ”Sikker Varsling” som et eget system, som ikke tilgangsbeskyttes av andre systemer.

1.4. Rapportstruktur

Neste kapittel gir bakgrunnsinformasjon om testing av programvare. Dette inkluderer en beskrivelse av aspekter rundt generell testing og en introduksjon til fagområdet rundt sikkerhetstesting. Kapittelet inneholder også en gjennomgang av konseptet rundt

risikostyring, hvor to standarder er brukt for å illustrere forskjellige fremgangsmåter til en slik risikostyring.

Kapittel 3 starter med en beskrivelse av systemet ”Sikker Varsling”. Deretter gir kapittelet resultatene fra risikostyringen av ”Sikker Varsling”, som innebærer en trussel- og risikoidentifikasjon. Til slutt gir kapittelet en oversikt over hvordan testprosessen i dette prosjektet er planlagt.

Første del av kapittel 4 gir en oversikt over hvordan risikoene er håndtert. Dette viser hvilke risikoer som var planlagt testet, og begrunnelse for hvorfor en del risikoer ble utelatt. Andre del av kapittelet gir en testbeskrivelse for hver av de planlagte testene. Disse testbeskrivelsene har blitt brukt som bakgrunn og referanse for selve testingen.

Kapittel 5 dokumenterer resultatene av testingen. Dette innebærer dokumentasjon av begrensninger i testmiljøet, en beskrivelse av hvordan resultatene er dokumentert, og selve resultatene av testene.

Kapittel 6 omhandler evalueringen av prosjektet. Kapittelet evaluerer valget av tester og testverktøy, samt en evaluering av testgjennomføringen og testresultatene. I tillegg gir dette kapittelet et forslag til testprosedyre for ”Sikker Varsling” og lignende systemer.

Kapittel 7 er konklusjonen av prosjektet. Dette kapittelet gir en oppsummering av arbeidet og erfaringene som prosjektet har gitt. I tillegg gir kapittelet forslag til arbeid som kan bygge videre på dette prosjektet.

Kapittel 8 inneholder bibliografien for dette prosjektet. De fleste litteraturkildene i prosjektet er dokumentert i dette kapittelet. De resterende referansene er skrevet som fotnoter. Fotnotene er referanser som ikke er så sentrale og som i hovedsak er ment som tilleggsinformasjon. Fotnotene er blitt brukt for å gjøre rapporten mer lesbar.

1.5. Definisjoner og forkortelser

CSRF – Cross Site Request Forgery
Denne typen sårbarhet går ut på at en pålogget bruker ubevisst sender en forhåndsgenerert forespørsel til en sårbar webserver. En fiendtlig forespørsel kan dermed forårsake skade for webserveren eller brukeren.

DoS – Denial of Service
Dette er et angrep som går ut på å overfylle en webserver med forespørsler slik at andre brukere ikke får tilgang til webområde.[3]

Forretningsrisikoer
Dette er risikoer som virker direkte inn på firmaet. Slike risikoer kan innvirke til finansielle tap, ødelegge firmaets rykte, brudd på kundekontrakt/lovbestemmelser, økt utviklingskostnad, osv.

Fullstendig test
En framgangsmåte der testen omfatter alle kombinasjoner av inputverdier og forbetingelser.[4]

Hash-funksjon
Dette er en funksjon som får en input med vilkårlig lengde, og produserer et *hashet* resultat med bestemt lengde. Dette er en form for kryptering, hvor samme input resulterer i samme *hashede* resultat.

IDE – Integrated Development Environment
Dette er et utviklingsmiljø som innholder all nødvendig funksjonalitet for programvareutvikling.

IDS – Intrusion Detection System
Software- eller hardwaresystemer som automatiserer prosessen med å overvåke hendelser i et datasystem eller nettverk, og analysere dem for tegn til sikkerhetsproblemer.[5]

Man-In-The-Middle angrep
Dette er et angrep hvor to parter kommuniserer gjennom en tredjepart. De to partene tror de kommuniserer med hverandre, mens i realiteten så går all kommunikasjon gjennom tredjeparten som dermed overvåker og kontrollerer all kommunikasjon mellom de to partene.

Risiko
Fare for tap av viktige verdier som følge av en uønsket hendelse. Risiko uttrykkes ved sannsynligheten for og konsekvensene av en slik hendelse.[6]

Risikokontrollerende tiltak
Tiltak som reduserer/eliminerer risikoer i et system. Det kan for eksempel være teknologiske løsninger, forsikringer, trening av ansatte, prosessforbedringer, osv.

SQL injeksjoner
Dette er angrep hvor input fra en angriper blir sendt som en kommando eller spørring mot databasen. Dette kan føre til utilsiktet utføring av kommandoer, eller modifisering/sletting av data.

Sårbarhet
Svakhet i et system som kan forårsake skade ved brudd på dets sikkerhetsregler.[7] Begrepet *sikkerhetsfeil* har også blitt brukt i denne sammenhengen.

Tekniske risikoer
Dette er en situasjon som går motsatt av det planlagte designet eller implementeringen av et system. Det kan for eksempel være knyttet til utviklingsprosessen, som gir for mye rom for feil i design og implementering.

Testobjekt
Komponenten eller systemet som er gjenstand for testing.[4]

Trussel
Muligheten for at en trusselkilde[1] (med vilje eller tilfeldigvis) utnytter en spesifikk sårbarhet.[8]

Utviklingsprosess
Dette begrepet er brukt som en samlende betegnelse for alle fasene i programvareutvikling.

XSS – Cross Site Scripting
Dette er en metode hvor en angriper utnytter en sårbar webserver ved å legge inn egne skript som kjører på serveren. Andre brukere av webserveren vil dermed kjøre skriptet i sine nettlesere, og dermed bli utsatt for angrep.

[1] Eksempel på trusselkilder er gitt i appendiks A.

2. Testing av programvare

Dette kapittelet omhandler grunnleggende informasjon om testing av programvare. Dette innebærer en introduksjon til sentrale modeller og begreper innenfor generell testing og innenfor sikkerhetstesting. Det er sikkerhetstesting som er hovedfokuset i dette prosjektet, men det er viktig å gi litt bakgrunnsinformasjon om generell testing, siden sentrale begreper innenfor dette fagfeltet også er sentrale innenfor sikkerhetstestingen.

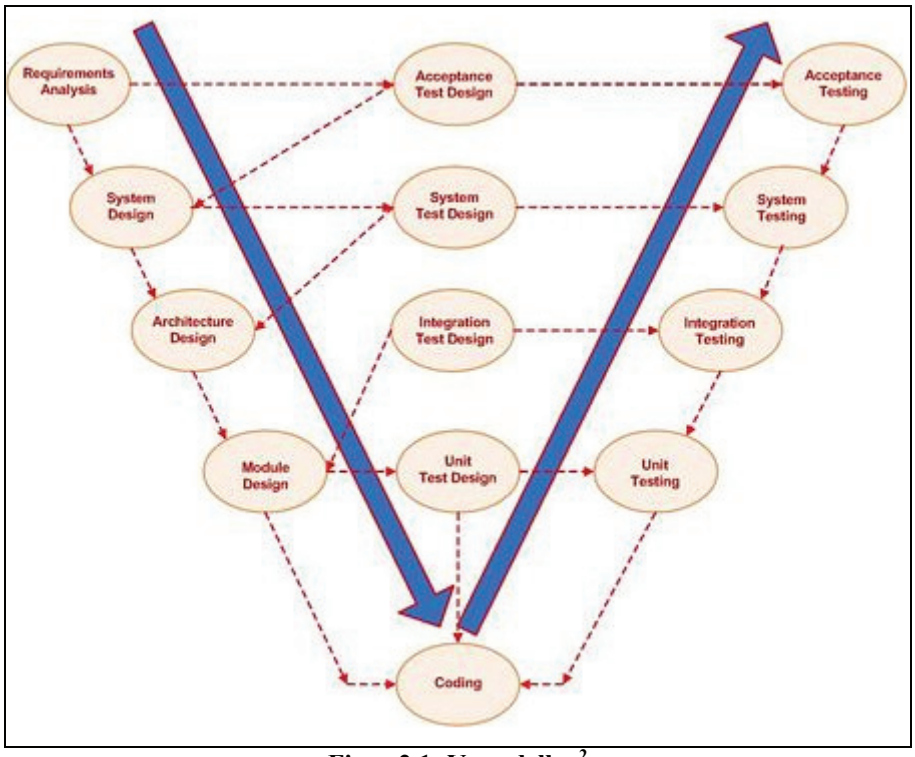
2.1. Generell testing

Testing av programvare er en viktig del av utviklingsprosessen. For å sikre at programvaren har den nødvendige funksjonaliteten i henhold til en antatt korrekt kravspesifikasjon, er det viktig at det blir gjennomført tilstekkelig testing av programvaren. Validering og verifikasjon er to sentrale begreper innenfor programvaretesting. Validering er å teste om et system, eller deler av et system, oppfyller kravene og forventningene som er satt til det. Validering går derfor ut på å teste om programmet er egnet for sitt tilsiktede bruk. Et spørsmål man kan stille seg under valideringen er; *Utvikler vi det rette systemet?* Verifikasjon derimot, er å teste om et system er komplett og korrekt utviklet i henhold til kravspesifikasjonen. Under verifiseringen kan man stille seg et annet spørsmål; *Utvikler vi systemet rett?*

I realistiske tilfeller vil det ikke være mulig å gjennomføre en fullstendig test. Derfor er det viktig å ha kunnskap rundt testing for å konstruere en effektiv testprosess for et gitt testobjekt. De neste avsnittene introduserer sentrale begreper og metoder som kan hjelpe til med dette.

2.1.1. V-modellen

En sentral modell innenfor programvaretesting er V-modellen. Hovedideen bak denne modellen er å framheve at testing og utvikling av programvare er korresponderende aktiviteter som bør betraktes som like viktige i en utviklingsprosess. Den ene siden av modellen representerer utviklingsprosessen, mens den andre siden representerer integrasjons- og testprosessen. Disse to sidene danner en V, som gir grunnlag for å vise tilhørigheten mellom forskjellige testnivåer og forskjellige utviklingssteg. Et eksempel på en slik modell er vist i Figur 2.1.



Figur 2.1: V-modellen[2]

Ut fra denne modellen kan det se ut som testing av programvaren skal starte sent i utviklingsprosessen. Dette er feil! Modellen legger opp til at *planleggingen* av de forskjellige testnivåene blir gjort i respektive utviklingsfaser. Det er kun selve *gjennomføringen* av testene som blir gjort i senere utviklingsfaser. Hvilke steg som inngår i planleggingen av en test er beskrevet i avsnitt 2.1.2, som omhandler hele testprosessen.

Hver av de fire testnivåene i denne modellen er koblet til en spesiell fase i utviklingsprosessen. Enhetstestingen er det første testnivået etter programmeringen, og er ment for å teste de enkelte enhetene/modulene/klassene i koden. I objektorientert programmering er slike tester rettet mot å teste de enkelte klassene i programkoden.

Selv om alle enhetene er testet og fungerer korrekt, kan det oppstå feil når de blir integrert til et samlet system. Derfor blir det neste testnivået integrasjonstester, hvor samhandlingen mellom enhetene blir testet. Slik testing går på den tekniske delen av systemdesignet, arkitekturen.

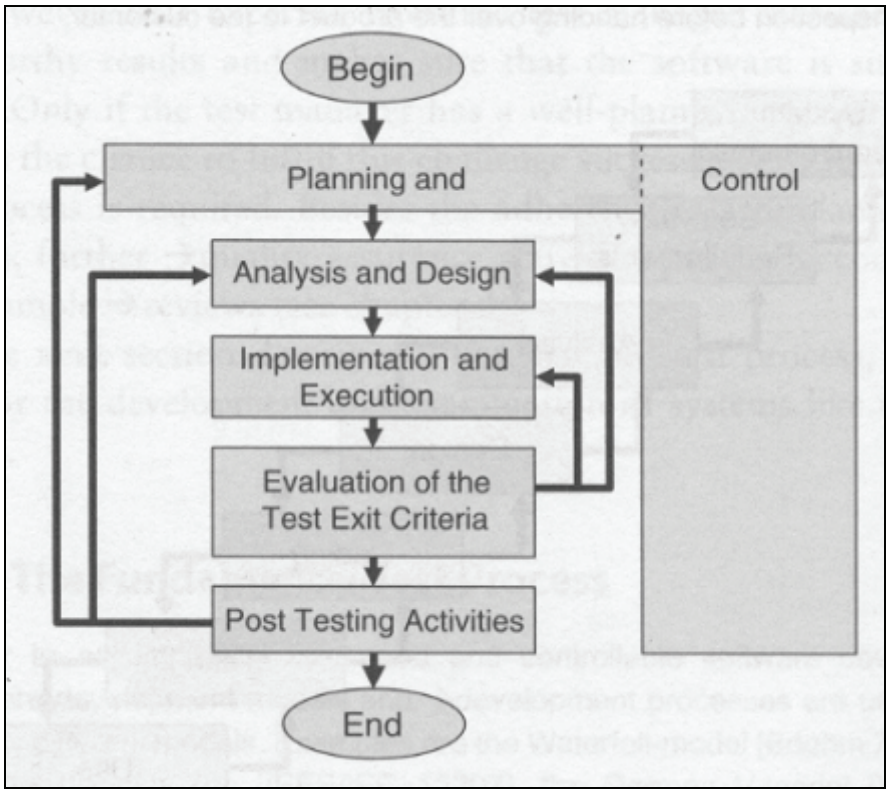
Det tredje testnivået ser mer på den funksjonelle delen av systemdesignet. Dette vil si at testingen ser på systemet fra brukerne sitt perspektiv. Slike tester bør gjennomføres i separate testmiljø, hvor testmiljøet er mest mulig likt et senere realistisk driftsmiljø. Ved å gjennomføre slike tester, vil systemet bli testet for driftsfeil som ikke er mulig å avdekke i tidligere testnivå.

Akseptansetester er det siste av testnivåene i V-modellen. En akseptansetest skiller seg fra de andre tre testene ved at kunden/brukerne blir inkludert i testingen. Slike tester tar sikte på å teste at systemet oppfyller kravene i en eventuell kontrakt, og blir akseptert av brukere og administratorer.

[2] Figur er hentet fra: http://en.wikipedia.org/wiki/V-Model_(software_development)

2.1.2. Testprosessen

I likhet med utvikling av programvare, har testing også en prosess med definerte steg som skal gjennomføres. Bøkene *Software Testing Foundations*[9] og *Software Testing Practice: Test Management*[10] definerer denne testprosessen på samme måte. Figuren de bruker er nesten identisk, bortsett fra navnene på noen av stegene. Figur 2.2 er tatt fra [9], og gir en god oversikt over en slik prosess.



Figur 2.2: Testprosessen

De enkelte stegene som inngår i denne prosessen er beskrevet under.

1. Første steg i testprosessen er planlegging og kontroll av testprosessen. Planleggingen består først og fremst av å definere omfanget av testingen. Dette inkluderer planlegging av antall personer som skal involveres, tidsforbruk, testverktøy og andre ressurser. Planleggingen bør også, ut fra en risikovurdering, prioritere hvilke tester som skal gjennomføres, samtidig som det for hver test blir definert avslutningskriterier som må oppfylles etter en utført test. Kontroll av testprosessen er som nevnt også en del av dette steget, men denne aktiviteten er egentlig kontinuerlig gjennom hele prosessen. En slik aktivitet er nødvendig for å sikre fremgang og kvalitet i prosessen.
2. Neste steg er analyse og design av testingen. Her blir testene som skal gjennomføres definert med tanke på hvordan de skal utføres. Analysen gir grunnlag for designet av testene. Avsnittene 2.1.3, 2.1.4 og 2.1.5 gir forklaring til noen av begrepene som brukes i defineringen av testene.
3. Steg nummer tre er selve utførelsen av testene. Dette innebærer at man gjennomfører testene slik som de ble bestemt i forrige steg. Et viktig poeng i denne delen av testingen er at testarbeidet blir dokumentert for å vise hvilke tester som ble gjort, hvem som utførte de og hvilket resultat testene hadde i forhold til forventet resultat.

4. Etter å ha gjennomført de oppsatte testene, skal arbeidet evalueres og dokumenteres i en testrapport. Som figuren viser kan denne evalueringen gi grunnlag for en ny testrunde hvor man går tilbake til å designe eller gjennomføre nye tester. En slik avgjørelse er basert på avslutningskriteriene definert i planleggingsfasen. I praksis er ofte tid og kostnader de reelle avslutningskriteriene.[10] Denne fasen av testprosessen skal avsluttes med en testrapport hvor testarbeidet blir oppsummert og resultatet fremlagt til prosjektledelse, ledere, kunder eller lignende.
5. Siste steg i figuren er ferdiggjøring av hele testprosessen. Dette innebærer arbeid hvor man evaluerer og dokumenterer erfaringene fra testprosessen som kan bli brukt som grunnlag i senere testing. Ifølge både [9] og [10] er dette en aktivitet som ofte blir glemt eller nedprioritert. En av grunnene til dette kan være mangel på tid, eller at man ikke ser gevinsten av dette arbeidet på kort sikt. Hvis man tenker mer langsiktig, vil man kunne se fordelen ved slike evalueringer ved at testprosessen kan forbedres i senere testsituasjoner. Både [9] og [10] bruker begrepet *"Testware conservation"* om arbeidet med å bevare design, dokumentasjon og testverktøy for bruk i samme eller lignende testsituasjoner.

Som figuren viser er ikke testprosessen en prosess hvor man jobber steg for steg. Den er mer iterativ av natur, hvor man ofte går tilbake til tidligere steg for å forbedre testingen. En slik forbedring kan også gjøres ved å se på selve testprosessen. I likhet med programvareutvikling, har også testing egne modeller/metoder for prosessforbedring. En av disse er *Test Maturity Model Integration* (TMMI)[3], som tar sikte på en stegvis forbedring av testprosessen i en organisasjon. Denne modellen har sin bakgrunn i den anerkjente modellen *Capability Maturity Model Integration* (CMMI), som er en modell for prosessforbedring. En annen modell for testprosessforbedring er *Test Process Improvement* (TPI)[4]. Denne modellen kan også hjelpe en organisasjon med å utarbeide trinnvise og målbare forbedringer i testprosessen.

2.1.3. Statisk vs. Dynamisk testing

Selve testene i en testprosess kan være statiske eller dynamiske. Statiske tester omhandler i hovedsak manuell gjennomgang og statisk analyse.[9] En manuell gjennomgang kan for eksempel være en inspeksjon av systemdokumenter, hvor målet er å finne feil og avvik fra systemspesifikasjonen, definerte standarder eller prosjektplanen. Menneskelig dyktighet er sentralt, og evnen til å tenke konstruktivt og analysere dokumentene er viktig i en slik prosess. Statisk analyse går også ut på å finne slike feil og avvik, men i en slik analyse bruker man testverktøy. Verktøyene er med på å automatisere analysearbeidet, og dermed også å effektivisere testingen. Et eksempel kan være et verktøy som analyserer programkoden, og finner potensielle programmeringsfeil eller svakheter i koden.

Dynamiske tester er tester som i motsetning til statiske tester, kjører kode for å teste hvordan et testobjekt fungerer. Dette blir gjort ved å gi input til testobjektet, for deretter å observere utfallet av inputen. Det er to hovedteknikker innen dynamisk testing; "black box" og "white box" testing. Disse to, samt en kombinasjon av disse teknikkene, er forklart i avsnitt 2.1.5.

[3] Hjemmeside: http://www.tmmifoundation.org/
[4] Hjemmeside: http://www.sogeti.no/templates/Sogeti_LokalStartsida____2225.aspx

2.1.4. Testtyper

En vanlig oppdeling av testtyper er inndelingen i funksjonelle og ikke-funksjonelle tester. En kort beskrivelse som viser forskjellen mellom disse to testtypene er gitt under.

Funksjonelle tester
Funksjonelle tester baserer seg på å verifisere at et testobjekt gir korrekt output for gitte input. Slike tester baserer seg på de funksjonelle kravene som fastsetter *hva* testobjektet skal gjøre.

Ikke-funksjonelle tester
Slike tester er ment å teste de ikke-funksjonelle kravene som er satt til testobjektet. Ikke-funksjonelle krav sier noe om *hvor bra* et system eller en del av et system skal fungere. Hvor bra et system oppfyller slike krav gjenspeiler seg ofte i hvordan brukerne oppfatter systemet.

Ifølge [9] er det i tillegg to andre testtyper som også skal være med i oppdelingen. Den ene er testing av programvarestrukturen, og den andre er testing relatert til forandringer. Disse to testtypene er kort forklart under.

Testing av programvarestruktur
Slike tester tar sikte på å teste strukturen i et testobjekt. Dette kan for eksempel være testing av koden eller arkitekturen i testobjektet. Teknikken som brukes for slike tester er ”white box” testing, som er beskrevet i avsnitt 2.1.5.

Testing relatert til forandringer
Denne typen tester er ment å oppdage feil i programvaren etter at forandringer er gjort. Slike forandringer kan være feilrettinger, oppdateringer, oppgraderinger, etc. Slike tester er kalt regresjonstester, og skal sikre at ingen nye feil har blitt introdusert etter at forandringene er gjort.

2.1.5. Testteknikker

Som nevnt i avsnitt 2.1.3, er dynamiske tester enten ”black-box”, ”white-box” eller en kombinasjon av disse. Hovedforskjellen mellom ”black-box” og ”white-box”, er tilgangen til systemspesifikasjoner som design og kildekode. En ”black-box” test ser på systemet som en boks som tar imot input og genererer output. De som gjennomfører slike tester vet ikke hvordan systemet genererer outputen. Det eneste som kan testes er at systemet gir den forventede outputen til gitte input.

I gjennomføringen av ”white-box” testing har man tilgang til systemspesifikasjonene og kan derfor gjennomføre testing av systemet på bakgrunn av denne informasjonen. Tilgang til for eksempel designløsningene kan ofte være til hjelp med å avdekke hvor systemet mest sannsynlig vil feile. På denne måten kan man begrense omfanget av testingen ved å konsentrere seg om mindre deler av systemet. Dette kan også være negativ konsekvens ifølge boken *Software Testing*[11]. Ved å for eksempel ha informasjon om koden, kan det være lett å overse feil ved at man tilpasser testene til hvordan koden kjører.

En tredje testteknikk, som er en kombinasjon av de to forrige, er ”gray-box” testing. Denne teknikken bruker ”black-box” testing på deler av systemet, samtidig som man har tilgang til noe av systemspesifikasjonene, og dermed kan gjennomføre enkelte ”white-box” tester. Et

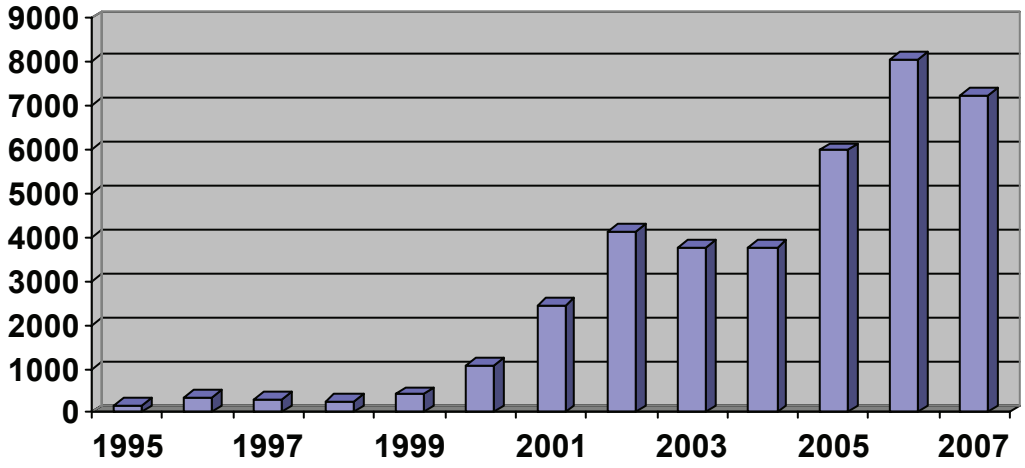
godt eksempel på en slik testteknikk er når man tester websider. Her har alle tilgang til HTML koden, mens selve programmeringen bak websiden er skjult.

2.2. Sikkerhetstesting

De siste årene har det blitt større oppmerksomhet rundt sikkerhet i programvare, noe som også har gitt sikkerhetstesting større oppmerksomhet. Sikkerhetstesting går ut på å teste programvare for å avdekke eventuelle sårbarheter. Sårbarheter finnes i mange forskjellige varianter, og en kreativ angriper finner stadig nye, innovative måter å utnytte disse sårbarhetene. Derfor er det viktig at testere og utviklere har kunnskap om programvaresikkerhet for å forhindre slike sårbarheter.

2.2.1. Økning i sårbarheter

En av grunnene til økt fokus på programvaresikkerhet er økningen av antall sårbarheter. Figur 2.3 viser hvordan antall rapporterte sårbarheter har økt siden 1995. Fra å være nede i 171 rapporterte tilfeller i 1995, var antallet 7 236 i 2007.[12]



Figur 2.3: Rapporterte sårbarheter til CERT/CC[5]

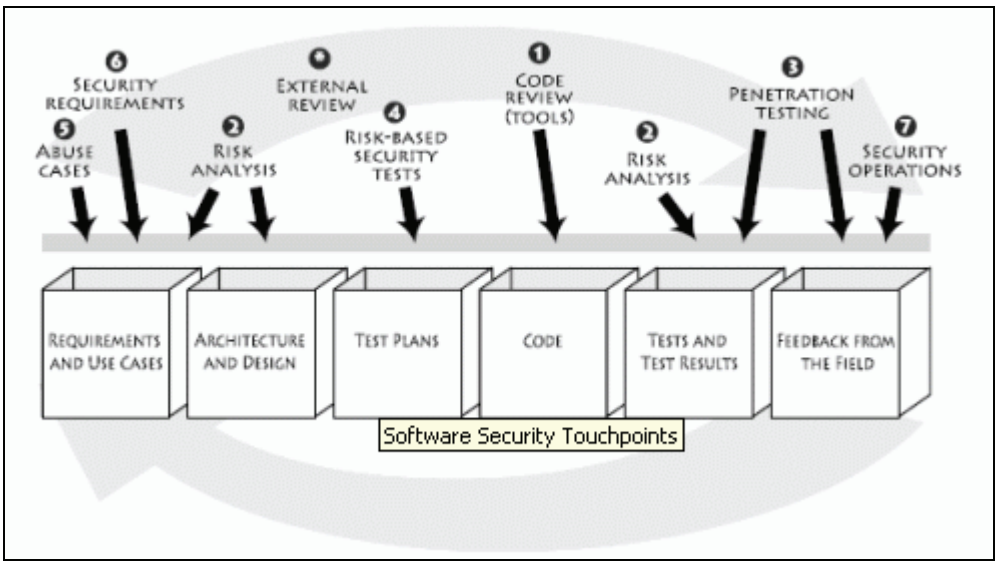
Det er mange årsaker til denne økningen, men Gary McGraw trekker frem tre hovedgrunner til hvorfor dette er et stadig økende problem[2]:

- Tilkoblinger – den økende tilkoblingen av datamaskiner gjennom Internett har gjort det lettere å foreta angrep.
- Utvidelser – den andre faktoren som spiller inn er at dagens systemer er mer utvidbare, noe som vil si at datamaskinene lettere kan oppdateres for å forbedre funksjonaliteten. Dette gjør det vanskeligere å forhindre sårbarheter som følge av slike oppdateringer.
- Kompleksitet – den siste hovedfaktoren er den økende kompleksiteten i informasjonssystemer, og spesielt i programvare. Kompleksiteten gjør det vanskeligere å unngå at sårbarheter oppstår.

[5] Hjemmeside: http://www.cert.org/certcc.html

2.2.2. Sikkerhet i utviklingsprosessen

For å redusere sårbarheter i programvare er det viktig at sikkerhet ikke er en ”aktivitet” som utføres i slutten av en utviklingsprosess. Mange er av den oppfatning at programvaresikkerhet oppnås ved hjelp av brannmurer og kryptering av kommunikasjon. McGraw[2] legger stor vekt på å tenke sikkerhet fra begynnelsen av utviklingsprosessen. I boken sin foreslår han sju sikkerhetsmetoder som kan inkluderes i forskjellige faser av utviklingsprosessen. Figur 2.4 gir en oversikt over hvor i utviklingsprosessen disse forskjellige metodene bør iverksettes. Noen av de mest sentrale metodene er beskrevet i avsnitt 2.2.4.



Figur 2.4: Sikkerhetsmetoder i utviklingsprosessen[2]

Hovedargumentet for å introdusere en sikkerhetstankegang fra starten av et utviklingsprosjekt, er at sårbarheter blir identifisert på et tidligere stadium. Dette resulterer i lavere kostnader til retting av sikkerhetsfeil, noe som Figur 1.1 viser. En annen viktig grunn er at 50 % av alle sikkerhetsproblemer kommer fra feil i designet.[2] Dette betyr at man kan luke ut halvparten av alle sårbarhetene ved å teste krav og design av et system.

2.2.3. Prinsipper for sikkerhetstesting

I testguiden fra *Open Web Application Security Project* (OWASP)[13] og manualen fra *Institute for Security and Open Methodologies* (ISECOM)[14], er det satt opp viktige testprinsipper som bør tas hensyn til under sikkerhetstestingen. Noen av disse er:

Det finnes ingen ”magisk” løsning
Det kan være fristende å tro at testverktøy vil finne alle sikkerhetsfeil i applikasjonen. På samme måte vil det være fristende å stole på at brannmuren blokkerer alle ulovlige forespørsler til applikasjonen. Dette er ikke korrekt, sikkerhet er en prosess og ikke et produkt.

Test tidlig og test ofte
Det er viktig å begynne testingen i en tidlig fase for å oppnå en sikker applikasjon på en effektiv og kostnadsbesparende måte. Figur 2.4 gir eksempler på sikkerhetsoperasjoner som man kan innføre i forskjellige deler av utviklingsprosessen.

Tenk som en angriper
For å gjennomføre en vellykket sikkerhetstest er det viktig at man tenker nytt og utradisjonelt. Et viktig poeng er at man klarer å sette seg inn i en angripers posisjon, for å få testet hvordan applikasjonen motstår eventuelle angrep. Dette krever kunnskap om hvilke angrep som kan utføres. Bøkene *How to Break Web Software*[3] og *Innocent Code*[15], gir mange gode eksempler på forskjellige slike angrep.

Bruk riktig testverktøy
Bruk av testverktøy er et viktig supplement i testarbeidet. Det er viktig å vite hvilke testverktøy som gir det beste resultatet for en gitt applikasjon. En introduksjon til testverktøy for sikkerhetstesting er gitt i avsnitt 2.2.5.

Vær detaljorientert
En god sikkerhetstest forutsetter at man er nøye med detaljene. Hvis man gjennomfører en grov og lite detaljert sikkerhetstest, kan dette gi en falsk trygghet angående sikkerheten i applikasjonen.

Bruk kildekode
Dersom kildekoden er tilgjengelig, er det en fordel om den blir brukt under sikkerhetstestingen. Bruk av kildekode vil gi et bedre bilde av sikkerheten, og avduke sikkerhetsfeil som ikke er mulig å finne med ”black box” testing.

Lag en effektiv testprosedyre
I mange prosjekter blir testing, og spesielt sikkerhetstesting, nedprioritert i budsjettet. Dette fører til at testingen får lite tid og ressurser til å bli gjennomført. For å øke testbudsjettet er det viktig at testprosedyren er effektiv, samtidig som den er grundig nok til å vise seg som en verdifull aktivitet.

Dokumenter resultatet skikkelig
Hvordan testresultatet blir oppfattet er avhengig av hvordan det presenteres. Som med alle faser i en utviklingsprosess, er det viktig at testresultatene blir grundig dokumentert.

2.2.4. Metoder i sikkerhetstesting

Det finnes forskjellige metoder i sikkerhetstesting, som alle har målsetting om å forbedre programvaresikkerheten. Under er det beskrevet en del sentrale testmetoder som kan introduseres i forskjellige faser i utviklingsprosessen. Rekkefølgen de er gjengitt i, er et forslag til når de bør introduseres i forhold til hverandre og i forhold til utviklingsprosessen.

Abuse case
Abuse case er den engelske betegnelsen på en metode som setter opp hypoteser på hva som kan gå galt i systemet. Det er spesielt to aktiviteter som står sentralt i utformingen av abuse case. Den første er utviklingen av anti-krav, som beskriver hva systemet ikke skal gjøre. Det andre er utformingen av en angrepsmodell, hvor en liste med kjente angrep blir gjennomgått

for å vurdere hvilke angrep som er gjeldende for det gitte systemet. Metoden beskriver med andre ord hvordan systemet skal fungere under angrep.

Risikostyring
Risikostyring er en testmetode som bør gjennomføres gjennom hele utviklingsprosessen. Sikkerhet er risikostyring, og siden risikostyring er så sentralt er det diskutert mer i detalj i avsnitt 2.3.

Manuell kontroll og gjennomgang
Denne metoden er tatt fra testguiden til OWASP[13], og går ut på å teste sikkerheten rundt mennesker, policys og prosesser i organisasjonen. På denne måten vil det komme frem om sikkerheten er forstått av menneskene, dokumentert i policyene og ivaretatt i prosessene. Metoden er i hovedsak gjort ved gjennomgang av dokumenter og intervju av designere og eiere av systemet. En slik type testing er tidskrevende og forutsetter dyktighet og kunnskap hos de som gjennomfører testingen. Dette betyr at metoden er ressurskrevende, men den store fordelen er at teknikken kan innføres tidlig i utviklingsprosessen, noe som øker sjansen for at sikkerhetsfeil blir funnet eller unngått i en tidlig fase. Metoden inkluderer også muligheten for inspeksjon av arkitekturen for å finne potensielle designfeil i systemet.

Risikobasert sikkerhetstesting
Denne testmetoden er komponentbasert, noe som vil si at testingen fokuserer på å teste hver komponent i systemet isolert. Metoden har likhetstrekk med penetrasjonstesting, men mens penetrasjonstester er opptatt med hvordan systemet fungerer i det miljøet den er satt opp i, er denne testmetoden mer opptatt av å teste sikkerheten til hver enkel komponent. Navnet på denne testmetoden tilsier at den baserer seg på risikostyringen rundt systemet. I tillegg baserer den seg på tidligere utviklede abuse case og andre funksjonelle sikkerhetskrav.

Kodegjennomgang
En ting alle programvarer har til felles er kildekode. Dermed blir gjennomgang og analyse av kode et viktig element i sikkerhetstestingen. Kodegjennomgang kan enten utføres automatisk ved hjelp av testverktøy, eller det kan gjøres manuelt ved at testere selv går gjennom koden. En manuell gjennomgang er mer tidkrevende, men samtidig kan en slik gjennomgang unngå den store ulempen med et automatisk testverktøy; mange falske positiver. En falsk positiv er når testverktøyet rapporterer om sikkerhetsfeil i applikasjonen som i realiteten ikke eksisterer. I tillegg til å unngå testverktøy som genererer for mange falske positiver, peker McGraw[2] på to andre karakteristikker som bør unngås i et slikt testverktøy; dårlig integrasjon med Integrated Development Environment (IDE), og kun support for programmeringsspråket C.

Penetrasjonstest
Mange organisasjoner bruker penetrasjonstester som den eneste sikkerhetstestingen av et system[2]. Penetrasjonstester er ment å teste et ferdigutviklet system i det miljøet det skal operere i. Dette vil si at slike tester som regel blir utført i en sen fase av utviklingsprosessen. Som vist i Figur 1.1, er dette en kostbar måte å avdekke feil i programvaren på. På den annen side, er penetrasjonstester viktige for å teste hvordan applikasjonen er konfigurert i det oppsatte miljøet. OWASP[13] legger vekt på at slike tester ikke krever det samme kunnskapsnivået som andre testnivåer. Dette er bare delvis korrekt, siden en god penetrasjonstest krever at testerne er i stand til å tenke som en angriper. Dette krever kunnskap om angrep og sårbarheter som systemet kan være utsatt for. Det finnes en del litteratur på dette området. Som nevnt dekker bøkene[3] og [15] mange slike angrep mot webapplikasjoner.

2.2.5. Testverktøy

Testverktøy hjelper i mange sammenhenger med å effektivisere sikkerhetstestingen. Ved riktig bruk av verktøyene kan enkle oppgaver bli automatisert, noe som ofte sparer tid for testerne. Det finnes mange tilgjengelige testverktøy, og et problem er å finne de rette testverktøyene til gitte tester. *Insecure.Org*[6] har gitt ut en liste over populære testverktøy, hvor mange av dem er gratis. Listen er utarbeidet på bakgrunn av flere tusen brukere, og deres favorittverktøy. Som nevnt i avsnitt 1.1, er en av oppgavene i dette prosjektet å finne passende testverktøy for de forskjellige testene. I valget av forskjellige testverktøy har *Insecure.Org* blitt brukt, samt resultatene fra enkelte søk på Internett. Testbeskrivelsene i avsnitt 4.2 gir en oversikt over hvilke testverktøy som er brukt, og Appendiks A gir en kort beskrivelse av verktøyene og erfaringene fra bruken av dem.

2.3. Risikostyring

Risikostyring er en viktig aktivitet både i generell testing og i sikkerhetstesting. Ifølge [9] består en risikostyring av tre hovedaktiviteter:

- Bedømme (og revurdere jevnlig) hva som kan gå galt (risikoer).
- Prioritere de identifiserte risikoene.
- Iverksette risikokontrollerende tiltak for å redusere/eliminere risikoene.

Alle disse aktivitetene krever en god del erfaring for å oppnå en mest mulig realistisk risikostyring. En god bedømmelse og prioritering av risikoer er ofte avhengig av erfaring fra lignende prosjekter. I prioriteringen blir risikoer definert som produktet av sannsynlighet og konsekvens. Sannsynligheten sier noe om hvor ofte en gitt risiko vil skje, mens konsekvensen sier noe om hvor alvorlig det er hvis en risiko inntreffer. Det varierer hvordan man graderer sannsynlighet og konsekvens. Et eksempel på en slik gradering er gitt i Appendiks B.

Som vist i Figur 2.4, bør risikostyringen innføres fra starten av utviklingsprosessen for å avdekke risikoer på et tidlig stadium. Deretter bør risikostyringen være en kontinuerlig prosess som sørger for at risikobilde til enhver tid er oppdatert. Det finnes mange standarder for risikostyring, som alle definerer visse steg i arbeidet med å identifisere, evaluere og redusere risikoer forbundet med et gitt system. De to neste avsnittene presenterer to forskjellige slike standarder. Grunnen til at disse to standardene ble valgt, var at begge fokuserer på risikostyring i forbindelse med sikkerhetstesting. Dermed blir slike risikostyringer brukt til å identifisere og rangere sikkerhetsrisikoer.

2.3.1. Risikostyringsstandard fra NIST

En standard som ofte blir referert er *NIST* sin guide for risikostyring[8]. Guiden gir en oversikt over oppgaver som må gjennomføres i en risikostyring. Risikostyringen er todelt, hvor første del er en identifikasjonsprosess og andre del er en reduksjonsprosess. Identifikasjonsprosessen består av å kartlegge mulige risikoer og rangere de ut fra

[6] Hjemmesiden til Insecure.Org er http://insecure.org/, og listen med testverktøy finnes på http://sectools.org/.

sannsynlighet og konsekvens. Stegene under illustrerer prosessen bak en slik risikovurdering av et system.

1. Få oversikt over systemet gjennom spørreskjema, intervju og gjennomgang av systemdokumenter.
2. Lage en oversikt over potensielle trusselkilder som er relevant for systemet. For hver trusselkilde bør det spesifiseres hvilke motiver og konkrete trusselhandlinger som kan forekomme.
3. Lage en oversikt over potensielle sårbarheter i systemet. Slike sårbarheter kan avdekkes ved hjelp av tidligere kjente sårbarheter[7], sårbarhetsskannere, og penetrasjonstester.
4. Få oversikt over eksisterende risikokontrollerende tiltak i systemet som blir brukt for å redusere sannsynligheten for utnyttelse av sårbarheter.
5. Bestemme sannsynligheten for at en potensiell sårbarhet kan utnyttes innenfor omfanget av tidligere oppdagede trusselkilder. Det er spesielt tre faktorer som bør bestemme sannsynligheten for en sårbarhet: motivasjonen og dyktigheten til potensielle trusler, egenskapene til selve sårbarheten og effektiviteten til eksisterende risikokontrollerende tiltak.
6. Bestemme konsekvensen ved utnyttelse av en sårbarhet. Dette kan bestemmes ved å se på hvilke negative konsekvenser en slik situasjon har for konfidensialiteten, integriteten og tilgjengeligheten til et system. Både sannsynligheten og konsekvensen til en sårbarhet må defineres kvalitativt eller kvantitativt. Forskjellen mellom disse to metodene er forklart i Appendiks B.
7. Bestemme risikonivået for hver sårbarhet. Til dette trengs en risikomatrise som definerer risikonivået som et produkt av sannsynlighet og konsekvens. Avhengig av hvor detaljert man vil være, varierer størrelsen på risikomatrisen. Eksempler på en slik risikomatrise er gitt i Appendiks B.
8. Gi anbefalinger til nye risikokontrollerende tiltak som kan bidra til å redusere risikoene.
9. Utarbeide en rapport som dokumenterer arbeidet med risikostyringen.

Reduksjonsprosessen er også en stegvis prosess hvor målet er å innføre risikokontrollerende tiltak som kan redusere/eliminere risikoer som blir ansett som uakseptable. Reduksjonsprosessen inkluderer sju steg som til sammen gjennomfører reduksjonen av utvalgte risikoer. Stegene er kort beskrevet under.

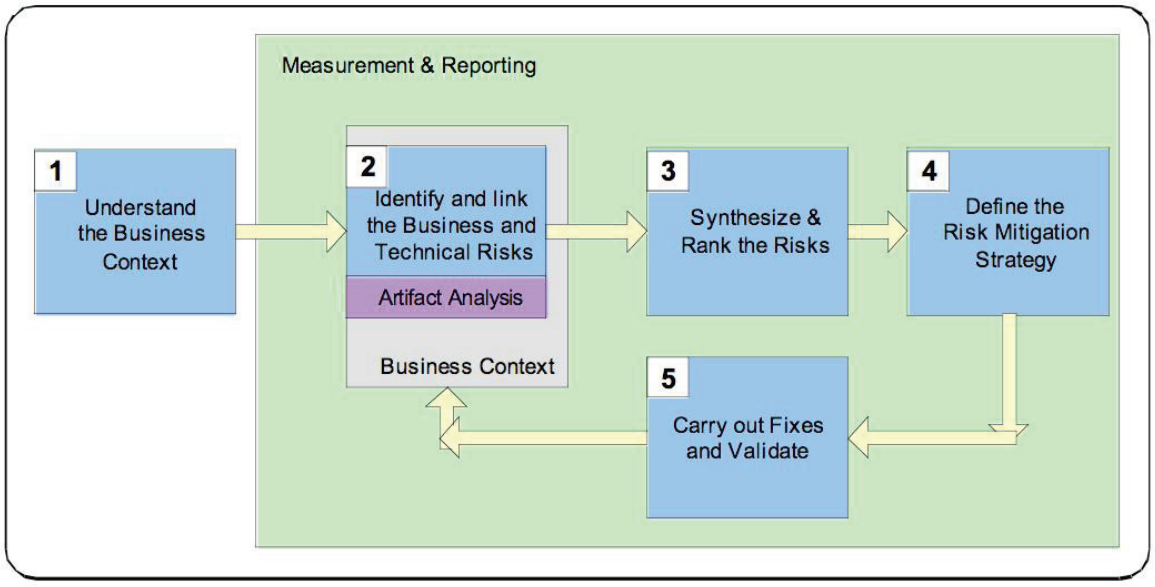
1. Allokere ressurser slik at risikoene med høyest risikonivå blir redusert først. Et risikonivå som er høyere enn en definert grense, tilsier at risikokontrollerende tiltak bør iverksettes umiddelbart.
2. Evaluere anbefalingene til risikokontrollerende tiltak fra risikovurderingen. Målet er å komme frem til en liste med slike tiltak som på best mulig måte reduserer den gitte risikoen.
3. Lage en lønnsomhetsanalyse over de foreslåtte risikokontrollerende tiltakene. Denne analysen vil gi grunnlag for å avgjøre hvilke tiltak som er mest lønnsomme å iverksette.
4. Velge risikokontrollerende tiltak ut fra lønnsomhetsanalysen.
5. Utpeke ansvarlige personer med den nødvendige kunnskapen og erfaringen til å implementere de valgte tiltakene.
6. Opprette en plan for implementeringen. Denne planen bør inneholde risikoer og de anbefalte risikokontrollerende tiltakene fra risikovurderingen, samt resultatene fra denne reduksjonsprosessen.

[7] Websidene http://nvd.nist.gov/ og http://www.securityfocus.com/ er eksempler som inneholder sårbarhetsdatabaser hvor man kan søke etter tidligere rapporterte sårbarheter.

7. Lage en oversikt over eventuelle gjenværende risikoer etter at tiltakene er implementert. Dette gir en oversikt over hvor mye risikoene ble redusert.

2.3.2. Risikostyringsstandard fra Cigital

En annen risikostyringsstandard kommer fra *Cigital*[8], og inkluderer også organisatoriske risikoer. Istedenfor bare å se på risikoer fra et systemperspektiv, blir de tekniske risikoene i et system funnet på bakgrunn av forretningsrisikoer. Figur 2.5 gir en oversikt over hvilke steg som inngår i *Cigitals* risikostyring.



Figur 2.5: Stegene i Cigitals risikostyring[2]

Som figuren viser, er denne prosessen delt opp i fem steg som hver bidrar til en fullstendig risikovurdering av et system sett fra en forretningssammenheng. Stegene er forklart under.

1. Det første steget blir gjennomført for å forstå organisasjonens forretningsmål, og derifra vurdere hvilke programvarerisikoer som er mest alvorlige for organisasjonen.
2. Det andre steget er sentralt i denne standarden. Her identifiseres de tekniske risikoene som finnes i systemet, samt forretningsrisikoene som truer identifiserte forretningsmål. Det som gjør dette steget så sentralt, er at de tekniske risikoene som avdekkes må kobles til forretningsmålene gjennom gitte forretningsrisikoer.
3. Neste steg er å analysere og rangere de tekniske risikoene. Analysen skal resultere i at hver enkelt risiko får en sannsynlighet og en konsekvens. Sannsynligheten sier noe om hvor stor sjanse det er for at en risiko inntreffer på en slik måte at det negativt påvirker organisasjonen. Konsekvensen sier noe om *hvor* negativ en slik risiko vil være. Rangeringen blir gjort ut fra risikonivået, som bestemmes ved hjelp av en risikomatrise. Mer om skalaer for sannsynlighet og konsekvens, samt et eksempel på en risikomatrise er gitt i Appendiks B.

[8] Hjemmeside: http://www.cigital.com/

4. Rangeringen i forrige steg gir grunnlag for neste steg, som er å utarbeide hvilke risikokontrollerende tiltak som på best mulig måte kan redusere eller eliminere risikoene. I tillegg bør også dette steget inkludere en strategi for hvordan man kan validere at risikoene er redusert/eliminert.
5. Siste steget i denne prosessen er selve implementeringen og valideringen av de løsningene som ble foreslått i forrige steg.

2.3.3. Sammenligning av risikostandardene

Den viktigste forskjellen på disse standardene ligger i identifikasjonsprosessen. Cigital inkluderer et organisatorisk perspektiv når risikoer blir identifisert. Dette vil ofte være en fordel siden man kobler tekniske risikoer opp mot organisatoriske forretningsmål. På denne måten vil man se hvilke risikoer som er mest alvorlige for organisasjonen, og ikke bare for systemet. En slik risikostyring er et krav i standarden ISO 17799[16] (fra juli 2007 omdøpt til ISO 27001), som omhandler styringssystemer for informasjonssikkerhet. Her blir det lagt vekt på at risikostyringen skal se på risikoer som omfatter hele organisasjonen, og ikke bare risikoer rundt spesifikke informasjonssystemer.

En annen forskjell er detaljnivået i standardene. NIST sin risikostandard er mer detaljert og legger flere føringer i hvordan fremgangsmåten skal være. Dette kan være en fordel for mennesker som ikke har mye erfaring med risikostyring. For disse kan en detaljert fremgangsmåte være til stor hjelp i gjennomføringen av risikostyringen. For mennesker med mer erfaring rundt risikostyring, kan standarden fungere som en sjekkliste for å sikre at arbeidet er skikkelig utført.

Både risikostandarden fra NIST og fra Cigital inneholder en identifikasjons- og reduksjonsprosess. NIST sin standard har en mer tydelig deling av prosessene, mens Cigital definerer risikostyringen som en mer helhetlig prosess, hvor identifikasjon og reduksjon av risikoer er samlet i en felles prosess.

2.3.4. Valg av risikostandard

På grunn av Cigital sitt organisatoriske perspektiv, ligger denne standarden utenfor omfanget av dette prosjektet. Risikostandarden fra NIST har dermed blitt brukt som bakgrunn for risikostyringen. Arbeidet med risikostyringen er dokumentert i avsnitt 3.2, og inneholder identifikasjonsprosessen fra risikostandarden. Reduksjonsprosessen er ikke blitt gjennomført, siden hovedfokuset har vert å identifisere risikoene rundt systemet ”Sikker Varsling”.

3. Prosjektutdyping

Dette kapittelet dokumenterer det arbeidet som ble gjort for å oppnå en effektiv testprosedyre for ”Sikker Varsling”. Dokumentasjonen inkluderer en oversikt over selve applikasjonen, en beskrivelse av den tilpassede risikostyringen som er gjort, og en oversikt over hvordan testprosessen er tilpasset omfanget av prosjektet.

3.1. ”Sikker Varsling”

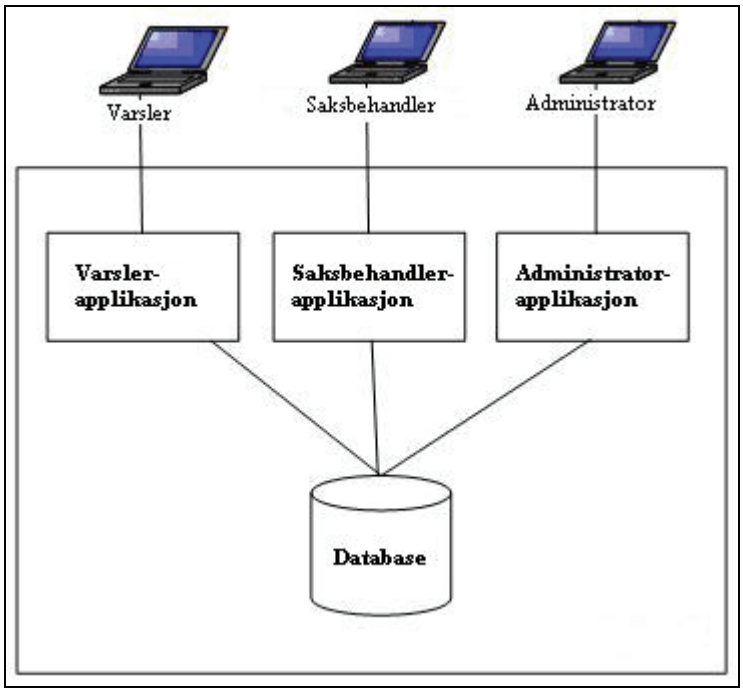
”Sikker Varsling” er en varslingstjeneste for arbeidstakere på arbeidsplassen. Denne tjenesten er utviklet av *Kantega AS* som en webløsning som kjører over https. Tjenesten er utviklet for å redusere terskelen for varsling av problemer på arbeidsplassen. For å bidra til dette skal ”Sikker Varsling” sikre arbeidstakerne 100 % anonymitet ved varsling. Systemet inneholder også et saksbehandlingssystem som holder orden på rapporterte saker og som gir organisasjonen mulighet til å følge opp tidligere rapporterte saker.

”Sikker Varsling” har tre forskjellige brukerroller som hver har sine rettigheter i systemet. Disse tre rollene blir alle definert som brukere av systemet, men i tillegg har de spesifikke rollenavn ut ifra hvilke rettigheter de har. De forskjellige brukerrollene er vist i Tabell 3.1.

Rolle	**Beskrivelse**
Varsler	Rolle som kun har rettigheter til å varsle en sak eller følge opp sine egne, tidligere rapporterte saker. Ved varsling av en sak kan varsleren være anonym eller registrere seg.
Saksbehandler	Rolle som har rettigheter til å behandle saker som er registrert gjennom tjenesten.
Administrator	Rolle som har rettigheter til å administrere andre brukere og kategorier som inneholder registrerte saker.

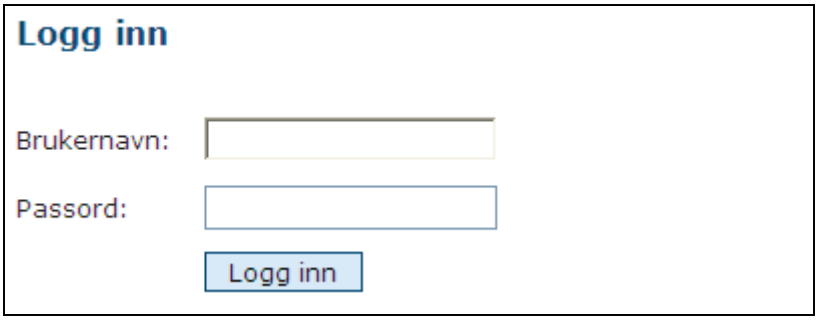
Tabell 3.1: Definisjoner av roller

Systemet er delt inn slik at hver brukerrolle har en egen applikasjon mot databasen. Dette er for å skille mellom hvilke funksjoner hver bruker kan utføre. For å illustrere hvordan ”Sikker Varsling” er bygd opp, viser Figur 3.1 en enkel oversikt over systemet.



Figur 3.1: Systemoversikt av "Sikker Varsling"

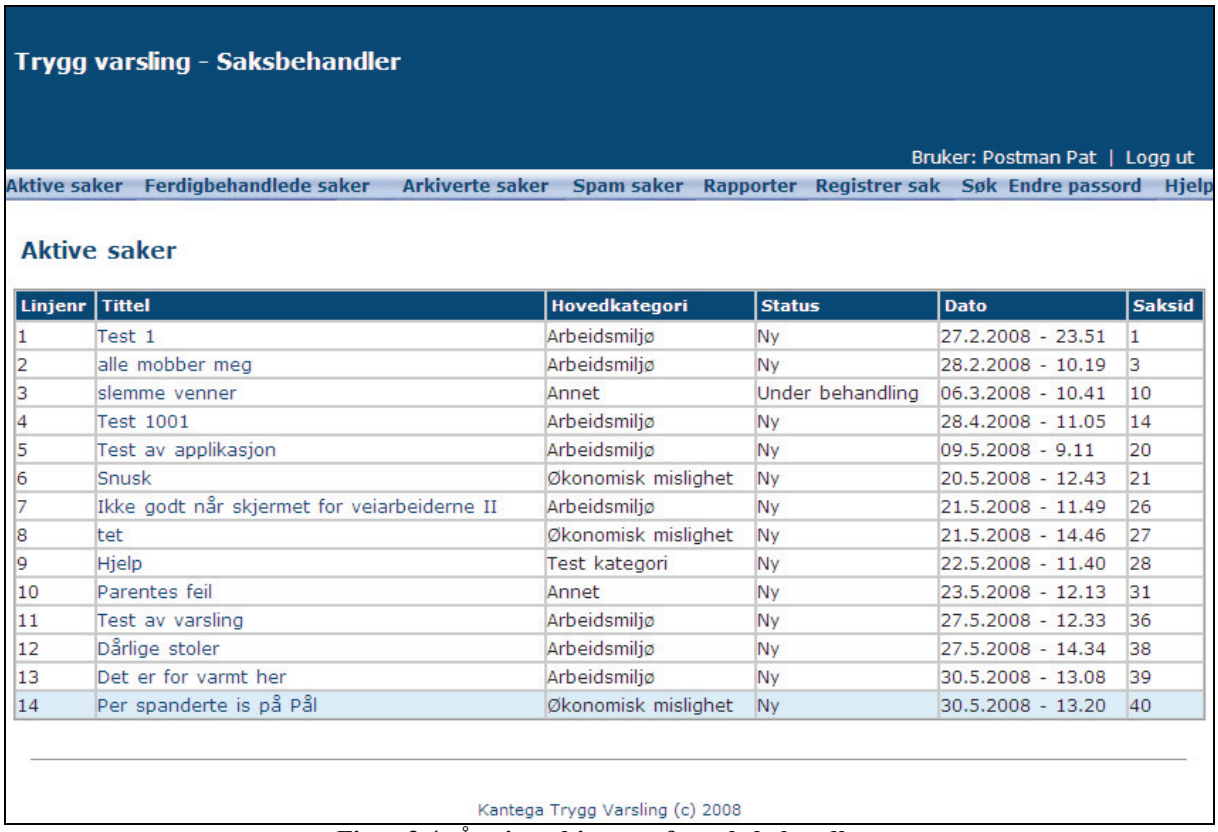
Både saksbehandler og administrator må logge seg inn med brukernavn og passord for å få tilgang til sine respektive sider. Innloggingen skjer ved hjelp av et brukernavn og passord, som vist i Figur 3.2.



Figur 3.2: Innlogging for administratorer og saksbehandlere

Ved en vellykket innlogging får hver av brukerrollene opp sine respektive sider. Disse sidene inneholder funksjonalitet som vist i Figur 3.3 og Figur 3.4. Hver enkel funksjonalitet er ikke beskrevet i detalj her, men heller beskrevet ved behov i testresultatene i kapittel 5.

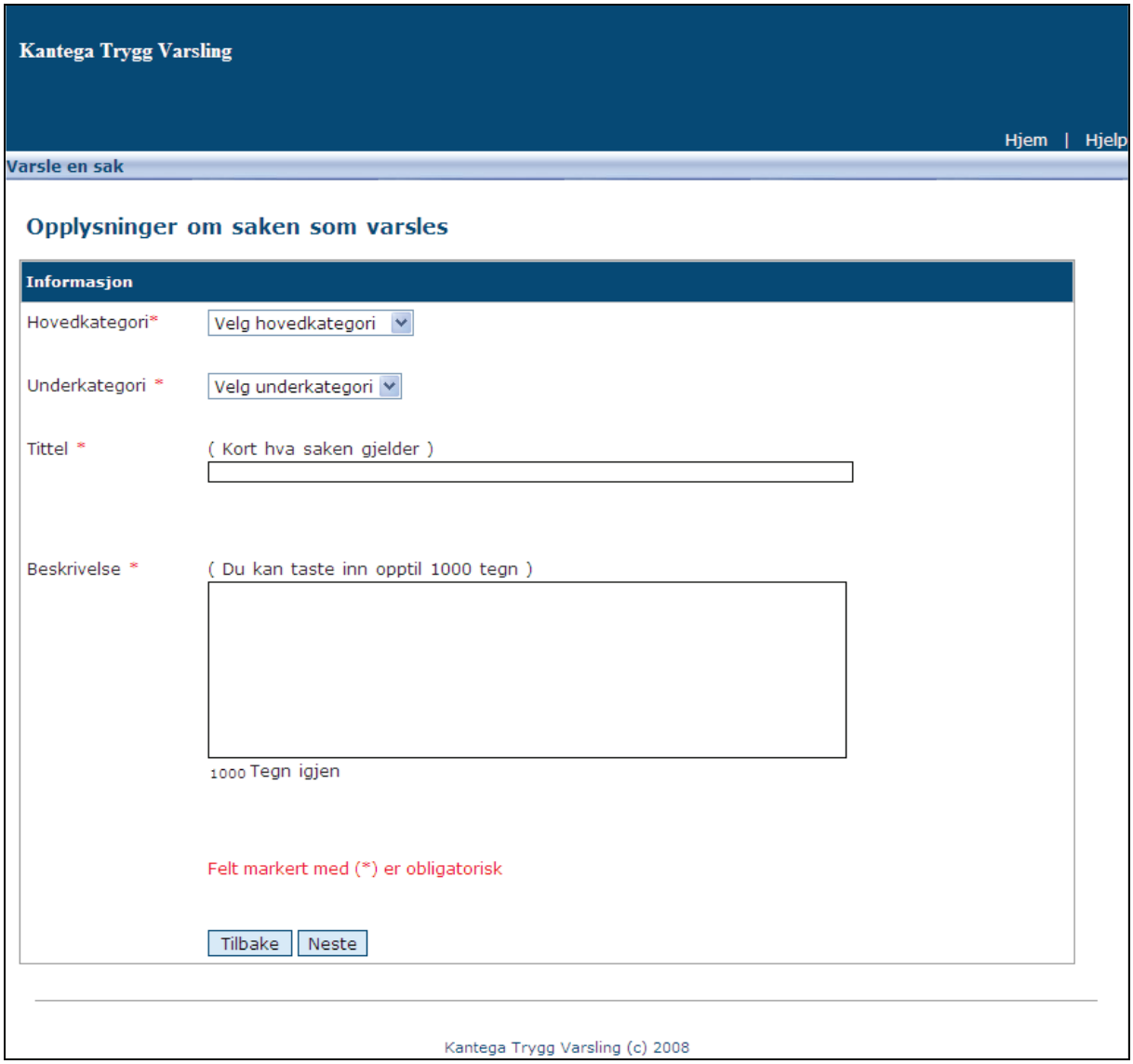
Figur 3.3: Åpningsskjermen for administratorer



Linjenr	Tittel	Hovedkategori	Status	Dato	Saksid
1	Test 1	Arbeidsmiljø	Ny	27.2.2008 - 23.51	1
2	alle mobber meg	Arbeidsmiljø	Ny	28.2.2008 - 10.19	3
3	slemme venner	Annet	Under behandling	06.3.2008 - 10.41	10
4	Test 1001	Arbeidsmiljø	Ny	28.4.2008 - 11.05	14
5	Test av applikasjon	Arbeidsmiljø	Ny	09.5.2008 - 9.11	20
6	Snusk	Økonomisk mislighet	Ny	20.5.2008 - 12.43	21
7	Ikke godt når skjermet for veiarbeiderne II	Arbeidsmiljø	Ny	21.5.2008 - 11.49	26
8	tet	Økonomisk mislighet	Ny	21.5.2008 - 14.46	27
9	Hjelp	Test kategori	Ny	22.5.2008 - 11.40	28
10	Parentes feil	Annet	Ny	23.5.2008 - 12.13	31
11	Test av varsling	Arbeidsmiljø	Ny	27.5.2008 - 12.33	36
12	Dårlige stoler	Arbeidsmiljø	Ny	27.5.2008 - 14.34	38
13	Det er for varmt her	Arbeidsmiljø	Ny	30.5.2008 - 13.08	39
14	Per spanderte is på Pål	Økonomisk mislighet	Ny	30.5.2008 - 13.20	40

Figur 3.4: Åpningsskjermen for saksbehandlere

Når en varsler velger å rapportere en sak, kan varsleren velge om saken skal rapporteres anonymt eller ikke. Dette er en sentral funksjonalitet som ”Sikker Varsling” tilbyr. Ved å velge å være anonym skal ingen kunne finne ut hvem som rapporterte saken. Ved oppfølging av en sak trenger varsleren å vite en 12-sifret PIN kode som genereres av ”Sikker Varsling” ved oppretting av saken. Figur 3.5 viser hvilke felter som må fylles inn ved varsling av en sak.



Figur 3.5: Felter som må fylles inn ved rapportering av sak

Saksbehandlingssystemet er bygd opp som en trestruktur med hovedkategorier og underkategorier som kan inneholde saker rapportert av ansatte. En hovedkategori kan inneholde flere underkategorier. For å gjøre forandringer som oppretting, endring eller sletting av kategorier, må brukeren være logget inn som Administrator.

Alle saker har en status som gir informasjon angående tilstanden til sakene. Dette gir brukerne informasjon om hvor langt en rapportert sak har kommet i rapporteringsprosessen. De forskjellige statusene en sak kan ha er vist i Tabell 3.2 under.

Status	Beskrivelse
Ny	Dette sier at saken nettopp er rapportert og ligger klar for behandling
Under behandling	Dette vil si at saken blir behandlet av en saksbehandler
Behandlet	Denne statusen blir satt på saker som er ferdig behandlet og klare for arkivering. Slike saker kan ikke endres
Falsk	Dette er saker som blir betraktet som tull/ugyldige/uekte
Arkivert	Dette er saker som er ferdig behandlet og blitt arkivert av en saksbehandler

Tabell 3.2: Definisjoner av saksstatus

3.2. Risikostyring av "Sikker Varsling"

Dette avsnittet tar for seg risikostyringen som ble gjort rundt systemet "Sikker Varsling". Som nevnt i avsnitt 2.3.4, ble risikostandarden fra NIST valgt som bakgrunn for risikostyringen. De neste avsnittene beskriver hvordan risikostyringen ble gjennomført, og resultatene av dette arbeidet.

3.2.1. Oversikt over systemet

Det første steget i risikostyringen, hvor målet er å få oversikt over systemet, har blitt gjennomført på egenhånd. Dette har i hovedsak vært gjennom analyse av tilgjengelige systemdokumenter. Denne gjennomgangen er grunnlaget for beskrivelsen av "Sikker Varsling" i avsnitt 3.1.

3.2.2. Trusselidentifikasjon

Trusselidentifikasjonen har i hovedsak blitt gjort på egenhånd. I tillegg har veilederne bidratt med innspill og kommentarer for å forbedre resultatet. For å begrense omfanget, har det kun blitt fokusert på menneskelige trusselkilder. Miljøbestemte trusselkilder kunne også blitt vurdert, siden trusler som for eksempel langvarig strømbrudd kan være relevant. Naturtrusler har blitt ansett som usannsynlige, og har derfor blitt utelatt i denne vurderingen. De to menneskelige truslene som er blitt ansett som relevant for dette prosjektet er vist i Tabell 3.3. Tabellen viser også motivasjonsfaktorer og konkrete handlinger for hver av disse truslene.

Trusler	Motivasjon	Trusselhandlinger
Hacker	• Informasjon • Anseelse • Ødeleggelse	• Uautorisert tilgang • Denial of Service • Innsetting av virus
Insider (blant brukere eller utviklere)	• Informasjon • Hevn • Forandre/slette saker • Utilsiktede feil	• Uautorisert tilgang • Utpressing • Modifisering av data • Sletting av data • Misbruk av informasjon • Salg av informasjon • Programmeringsfeil • Feil bruk av systemet

Tabell 3.3: Oversikt over trusselkilder

3.2.3. Risikoidentifikasjon

I dette steget av risikostyringen har det blitt gjennomført en liten brainstorming for å komme opp med risikoer rundt systemet. Denne prosessen har stort sett foregått på samme måte som ved trusselidentifikasjonen, hvor et sett med risikoer er blitt identifisert på egenhånd, for deretter å ha blitt vurdert og forbedret av veilederne til prosjektet. Prosessen har fokusert på å finne ikke-tekniske og tekniske risikoer. Hver identifisert risiko er blitt nummerert for å gjøre det lettere å referere til hver enkelt risiko.

Ikke-tekniske risikoer
I arbeidet med å finne ikke-tekniske risikoer har det blitt tatt utgangspunkt i seks forskjellige aktører; *Varsleren*, *Saksbehandleren*, *Administratoren*, *Kjøperen av systemet*, *Eieren av systemet*, og *Driftsoperatøren*. De tre første aktørene representerer forskjellige brukere av systemet, som vist i Tabell 3.1. Hos disse har risikoene blitt delt inn i de tre sikkerhetsprinsippene; *konfidensialitet*, *integritet* og *tilgjengelighet*. Under gis en oversikt over de identifiserte risikoene, med eksempler på årsaker som kan realisere risikoene tilknyttet disse brukerne.

Varsler

R-1 Konfidensialitet
- Uvedkommende leser hva *varsler* har skrevet
  - Gjennom kjennskap til PIN kode
  - Ved bruk av ukryptert kommunikasjon
  - Papirutskrifter makuleres ikke
  - Saksbehandler misbruker sin myndighet

R-2 Integritet
- Uvedkommende forandrer eller sletter det *varsler* har skrevet
  - Kan følge opp sak ved kjennskap til PIN koden
  - Saksbehandler misbruker sin myndighet

R-3 Tilgjengelighet
- Systemet er nede, så *varsler* får ikke varslet
  - På grunn av Denial of Service (DoS) angrep
  - På grunn av dårlige vedlikeholdsrutiner
- Varsler mister (glemmer) PIN kode
- Problemer med internettforbindelsen eller andre kommunikasjonsmedium

Saksbehandler

R-4 Konfidensialitet
- Uvedkommende får tak i personlige opplysninger om *saksbehandler*
- Uvedkommende får *saksbehandlers* oversikt over registrerte saker
  - Gjennom kjennskap eller tyveri av brukernavn/passord
  - Ved manglende autorisasjon av en bruker

R-5 Integritet
- Uvedkommende forandrer/sletter opplysninger om *saksbehandler*
  - Administrator misbruker sine rettigheter
- Uvedkommende forandrer status eller sletter saker
  - Gjennom kjennskap eller tyveri av brukernavn/passord

R-6 Tilgjengelighet
- Systemet er nede, så *saksbehandler* kan ikke behandle saker
  - På grunn av (DoS) angrep
  - På grunn av dårlige vedlikeholdsrutiner
- *Saksbehandler* mister (glemmer) brukernavn/passord
- Administrator sperrer *saksbehandlers* tilgang til systemet
- Problemer med internettforbindelsen eller andre kommunikasjonsmedium

Administrator

R-7 Konfidensialitet
- Uvedkommende får tak i personlige opplysninger om *administrator*
- Uvedkommende får tak i informasjon om andre *administratorer//saksbehandlere*
  - Gjennom kjennskap eller tyveri av brukernavn/passord

R-8 Integritet
- Uvedkommende kan forandre eller slette brukernavn/passord til andre brukere
- Uvedkommende kan forandre eller slette hovedkategorier/underkategorier
  - Gjennom kjennskap eller tyveri av brukernavn/passord

R-9 Tilgjengelighet
- Systemet er nede, så *administrator* kan ikke administrere systemet
  - På grunn av DoS angrep
  - På grunn av dårlige vedlikeholdsrutiner
- Mistet (glemt) brukernavn/passord
- Problemer med internettforbindelsen eller andre kommunikasjonsmedium

De tre siste aktørene er ikke direkte brukere av systemet, men heller interessenter som er knyttet til systemet på andre måter. For hver av disse har det blitt identifisert forskjellige risikoer som er knyttet til systemet.

Kjøpere av systemet

R-10 Velger upålitelige og ukompetente administratorer/saksbehandlere

R-11 Manglende rutiner for tilbakemelding av problemer/feil med systemet

R-12 Manglende informasjon om systemet til ansatte

Eier av systemet (Kantega AS)

R-13 Ansatte lager ”bakdør” enten bevisst eller ubevisst
R-14 Unødvendig mange som har tilgang til systemdokumentasjon
R-15 Dårlige vedlikeholdsrutiner ved oppdatering/feilretting
R-16 Dårlig kunnskap rundt programvaresikkerhet

Driftoperatør

R-17 Dårlige vedlikeholdsrutiner
- Systemet er nede i lang tid
- Ikke oppdatert med tanke på nye sårbarheter

R-18 Dårlige sikkerhetsmekanismer for blokkering av innbruddsforsøk
- Ingen *Intrusion Detection System* (IDS) i driftsmiljøet

R-19 Dårlig logging av innbruddsforsøk
- Ingen logging av mange mislykkede innloggingsforsøk

Tekniske risikoer

Identifikasjon av de tekniske risikoene inkluderer flere innfallsvinkler. Først har OWASP sin topp 10 liste[9] over sårbarheter blitt vurdert for å se om disse utgjør en risiko for ”Sikker Varsling”. Dette er vurdert i listen under, hvor begrepene er beskrevet og vurdert i henhold til ”Sikker Varsling”.

R-20 Cross Site Scripting (XSS)
- XSS er beskrevet i avsnitt 1.5.
  - ”Sikker Varsling” tillatter input fra brukerne, og kan dermed være sårbar for XSS.

R-21 Injeksjonsfeil
- Dette gjelder hovedsakelig SQL injeksjoner, som er beskrevet i avsnitt 1.5.
  - ”Sikker Varsling” tar imot input fra brukerne som lagres i en database, og kan dermed være sårbar for SQL injeksjoner.

R-22 Ondsinnet filutføring
- Dette går ut på at angripere kan legge ved filer i applikasjonen som utfører utilsiktede operasjoner.
  - Vedlegg av filer er en funksjon som kan aktiveres, noe som kan gjøre dette mulig i ”Sikker Varsling”.

R-23 Usikre referanser til indre objekter
- Dette oppstår når det eksisterer direkte referanser til indre objekter som for eksempel filer, databaser eller andre koder. Dette kan en angriper forandre og dermed få adgang til slike objekter uten autorisasjon.
  - Et eksempel er at PIN koden for anonymt varsel blir direkte referert i kildekoden.

R-24 Cross Site Request Forgery (CSRF)
- CSRF er beskrevet i avsnitt 1.5.
  - Siden XSS angrep er mulig, er også CSRF et mulig angrep som må vurderes.

[9] Hjemmeside: http://www.owasp.org/index.php/Top_10_2007

R-25 Dårlig feilhåndtering
- Dette retter seg mot hvordan webapplikasjonen håndterer eventuelle feilmeldinger. En dårlig håndtering kan føre til at angripere får verdifull informasjon om applikasjonen, eller dets brukere.
  - Brukerinformasjon, varslingsinformasjon, feil i innlogging, og feil i registrering av sak er alle eksempler på informasjon og feilhåndtering som er relevant i ”Sikker Varsling”.

R-26 Dårlig sesjonshåndtering
- Dette oppstår når innloggingsdetaljer og annen informasjon under innlogget sesjon ikke er godt nok beskyttet mot angrep.
  - ”Sikker Varsling” genererer en sesjonsidentifikator for hver gang en bruker kobler opp mot systemet. Dette gjør denne sårbarheten relevant for ”Sikker Varsling”.

R-27 Usikker lagring
- Dette kan oppstå på grunn av at lagret informasjon er dårlig kryptert.
  - Dette er ansett som mindre relevant siden all lagret informasjon blir kryptert i databasen.

R-28 Usikker kommunikasjon
- Dette kan oppstå på grunn av for dårlig kryptert kommunikasjon mellom applikasjonen og brukeren.
  - Dette er ansett som mindre relevant siden all kommunikasjon skjer over https protokollen.

R-29 Feil i URL-tilgang
- Dette går ut på at applikasjonen hindrer uautoriserte brukere å nå websider ved å skjule linker eller URL-er. En slik fremgangsmåte gjør at angripere kan skrive inn en URL direkte i nettleseren, og dermed få tilgang til ”skjulte” sider.
  - ”Sikker Varsling” inkluderer mange URL-er, slik at angrep på en slik sårbarhet kan være relevant.

De nevnte sårbarhetene representerer, ifølge OWASP, de mest vanlige sårbarhetene for webapplikasjoner. Det finnes også andre sårbarheter og angrep som kan være relevante for slike applikasjoner, og noen av dem er nevnt under.

R-30 Denial of Service (DoS)
- DoS er beskrevet i avsnitt 1.5.
  - ”Sikker Varsling” er webbasert og dermed blir DoS angrep relevant for systemet.

R-31 Tvinge frem svak kryptering
- Dette er en situasjon hvor en angriper kan velge hvor sterk kryptering det skal være på kommunikasjonen mellom en bruker og en webapplikasjon. Hvis en angriper klarer å velge en svak kryptering, blir det lettere å dekryptere nettverkstrafikken.
  - Kan være relevant for ”Sikker Varsling” hvis tidligere versjoner av SSL brukes.

R-32 Informasjon i kildekoden
- En slik sårbarhet oppstår ved at utviklerne skriver kommentarer eller annen informasjon i kildekoden. Dermed kan angripere direkte lese denne informasjonen, og eventuelt bruke den i arbeidet med å angripe en webapplikasjon. Html-kode er spesielt sårbar for slike kommentarer.
  - Som alle webapplikasjoner, er html-koden for enkelte sider i ”Sikker Varsling” tilgjengelig for alle. Dermed blir eventuelle kommentarer synlige for angripere.

R-33 Sårbarheter i tredjeparts programvare
- En webapplikasjon kan utsettes for sårbarheter ved at man integrerer tredjeparts programvare som en del av applikasjonen. Eksempler på slik tredjeparts programvare er webserver, database, brannmur, IDS, etc.
  - ”Sikker Varsling” bruker tredjeparts programvare i webserveren og i databasen, og er derfor utsatt for sårbarheter introdusert av disse.

R-34 Sårbarheter i nettlesere
- Dette sikter også mot sårbarheter i tredjeparts programvare, men skiller seg fra risikoen over ved at brukerne selv introduserer slike sårbarheter. Manglende oppdateringer av nettlesere fra brukerne sin side, kan være en risiko for en webapplikasjon.
  - Dette kan være en risiko for ”Sikker Varsling” ved at brukere ikke har oppdatert nettleserne sine, og på den måten introduserer sårbarheter.

3.2.4. Risikorangering

En rangering av de identifiserte risikoene er viktig for å fokusere sikkerhetstestingen på de mest kritiske delene av ”Sikker Varsling”. Rangeringen har tatt utgangspunkt i at risiko er et produkt av sannsynlighet og konsekvens, og begrepene som er brukt er tatt fra risikomatrisen i Appendiks B.3. Tabell 3.4 viser rangeringen av risikoene, og numrene referer til nummereringen i avsnitt 3.2.3.

Risiko	Risikobeskrivelse	Sannsynlighet	Konsekvens	Risikonivå
R-1	Konfidensialitet varsler	Medium	Stor	Veldig høy
R-4	Konfidensialitet saksbehandler	Medium	Stor	Veldig høy
R-8	Integritet administrator	Medium	Stor	Veldig høy
R-10	Upålitelige/ukompetente admin./saksbehandler	Medium	Stor	Veldig høy
R-20	Cross Site Scripting	Medium	Stor	Veldig høy
R-21	SQL injeksjoner	Medium	Stor	Veldig høy
R-26	Dårlig sesjonshåndtering	Medium	Stor	Veldig høy
R-34	Sårbarheter i nettlesere	Høy	Moderat	Veldig høy
R-2	Integritet varsler	Medium	Moderat	Høy
R-3	Tilgjengelighet varsler	Høy	Minimal	Høy
R-5	Integritet saksbehandler	Medium	Moderat	Høy
R-6	Tilgjengelighet saksbehandler	Høy	Minimal	Høy
R-7	Konfidensialitet administrator	Medium	Moderat	Høy
R-9	Tilgjengelighet administrator	Høy	Minimal	Høy
R-13	Ansatte lager ”bakdør”	Medium	Moderat	Høy
R-15	Dårlige vedlikeholdsrutiner (eier)	Medium	Moderat	Høy
R-17	Dårlige vedlikeholdsrutiner (driftsoperatør)	Medium	Moderat	Høy
R-24	Cross Site Request Forgery	Medium	Moderat	Høy
R-25	Dårlig feilhåndtering	Medium	Moderat	Høy
R-29	Feil i URL-tilgang	Medium	Moderat	Høy
R-31	Tvinge frem svak kryptering	Lav	Stor	Høy
R-32	Informasjon i kildekoden	Medium	Moderat	Høy
R-33	Sårbarheter i tredjeparts programvare	Medium	Moderat	Høy
R-11	Manglende tilbakemeldingsrutiner	Medium	Minimal	Medium
R-12	Manglede informasjon om systemet til ansatte	Medium	Minimal	Medium
R-14	Mange har tilgang til systemdokumenter	Medium	Minimal	Medium
R-16	Dårlig kunnskap rundt programvaresikkerhet	Medium	Minimal	Medium
R-18	Dårlige sikkerhetsmekanismer	Lav	Minimal	Medium
R-19	Dårlig logging	Lav	Minimal	Medium
R-22	Ondsinnet filutføring	Lav	Moderat	Medium
R-23	Usikre referanser til indre objekter	Lav	Moderat	Medium
R-27	Usikker lagring	Usannsynlig	Stor	Medium
R-28	Usikker kommunikasjon	Usannsynlig	Stor	Medium
R-30	Denial of Service	Medium	Minimal	Medium

Tabell 3.4: Risikorangering

På grunn av manglende erfaring fra liknende prosjekter, har de fleste risikoene fått sannsynlighet *Medium*. Prosessen med å gradere sannsynlighetene og konsekvensene for hver enkelt risiko har i hovedsak blitt basert på egne antagelser. I tillegg har veilederne til prosjektet bidratt med innspill for å gjøre rangeringen mest mulig realistisk.

Håndteringen av risikoene er gjort ut fra denne rangeringen. Dette er gjort i kapittel 4, som beskriver hvordan testutføringen er planlagt.

3.2.5. Risikokontrollerende tiltak

Det er ikke blitt gjennomført en analyse for å sjekke hvilke risikokontrollerende tiltak som eksisterer fra før. Hovedgrunnen til dette er at en slik identifikasjonsprosess er tidkrevende, og avviker fra formålet med prosjektet. Formålet har vært å finne en effektiv testprosedyre bestående av forskjellige tester og tilhørende testverktøy. Eksisterende risikokontrollerende tiltak som er identifisert under selve testingen er blitt dokumentert i kapittel 5, som omhandler resultatene fra testingen.

3.3. Testprosessen

Testprosessen i dette prosjektet har i grove trekk fulgt testprosessen fra avsnitt 2.1.2. Prosessen fra avsnitt 2.1.2 er mer detaljert og setter krav til de forskjellige stegene som skal gjennomføres. Dette prosjektet har prøvd å følge disse stegene, men på grunn av tidsbegrensningen har stegene blitt forenklet og tilpasset prosjektomfanget. Selv om testprosessen fra avsnitt 2.1.2 gjelder for testing generelt, kan den også brukes i sikkerhetstestingen av ”Sikker Varsling”.

Planleggings- og kontrollfasen har i hovedsak bestått av risikostyringen i avsnitt 3.2, og utviklingen av en fremdriftsplan. Fremdriftsplanen er vist i Appendiks C, og viser en ukentlig målsetning for arbeidsoppgavene i prosjektet. Planen viser også målsetningen for fremdriften til rapportskrivingen. Kontrolldelen av prosjektet har blitt forenklet til å sjekke fremdriften opp mot denne planen. Dette er også dokumentert i planen.
Analyse- og designsteget er gjennomført på bakgrunn av risikostyringen i avsnitt 3.2. Her har forskjellige tester blitt beskrevet med tanke på omfang, testmetode, testverktøy og forventet resultat. Dette er dokumentert i kapittel 4.

Testgjennomførings- og evalueringsfasen har bestått av gjennomføringen og dokumentasjonen av testingen. Kapittel 5 dokumenterer resultatene fra testene. I denne fasen har det blitt lagt mindre vekt på avslutningskriterier for hver enkel test. Slike kriterier skal egentlig utarbeides i planleggings- og kontrollfasen, men på grunn av vanskeligheten med å definere avslutningskriterier for sikkerhetstester, har dette prosjektet valgt å bruke tidsbegrensning som hovedkriterium. Som fremdriftsplanen i Appendiks C viser, har hver testmetode fått en tidsbegrensning på en uke.

Den siste fasen av testprosessen er dekket av de to siste kapitlene, *Evaluering* og *Konklusjon*. Evalueringen legger grunnlaget for testprosedyren, og bedømmer hvilke testmetoder og testverktøy som bør brukes i fremtidige tester. Konklusjonen gir et sammendrag av arbeidet og erfaringene i prosjektet.

4. Testutføring

Dette kapittelet beskriver planleggingen av testene. Dette innebærer en beskrivelse av hvilke tester som er prioritert. Prioriteringen har brukt risikorangeringen i avsnitt 3.2.4 som grunnlag. Deretter gir kapittelet en overordnet beskrivelse av omfang, testmetode, testverktøy og forventet resultat for hver av de prioriterte testene.

4.1. Risikohåndtering

Sikkerhetstestingen av "Sikker Varsling" er basert på risikorangeringen i avsnitt 3.2.4. Risikoene med høyest risikonivå har blitt prioritert først. Tidsbegrensningen i dette prosjektet har begrenset antallet tester. Den planlagte tidsbruken på testutføringen er vist i fremdriftsplanen i Appendiks C. Tabell 4.1 gir en oversikt over de planlagte testene.

Test ID	Hovedrisiko	Tilleggsrisiko	Kommentar
T-1	R-1	R-4, R-8	Konfidensialiteten til en varsler er avslørt hvis konfidensialiteten til en saksbehandler eller administrator er avslørt.
T-2	R-20	R-24	Hvis "Sikker Varsling" er sårbar for XSS angrep, vil det også være sårbar for CSRF angrep.
T-3	R-21		
T-4	R-26		
T-5	R-34		
T-6	R-3	R-30	Tilgjengeligheten for en varsler kan ødelegges ved et DoS angrep
T-7	R-25		
T-8	R-29		

Tabell 4.1: Plan for testing av risikoer

Forklaring av tabell
Kolonnen *Test ID* gir en unik test ID for hver test som er utført. Denne blir også brukt som referanse senere i rapporten.
Kolonnen *Hovedrisiko* viser hvilken risiko som ligger til grunn for testingen, mens kolonnen *Tilleggsrisiko* viser eventuelle risikoer som kan kobles til hovedrisikoen. Ta for eksempel risiko R-1 som omhandler konfidensialiteten til en varsler. Denne konfidensialiteten kan også bli avslørt ved at en angriper avslører konfidensialiteten til en saksbehandler (R-4) eller integriteten til en administrator (R-8). Ved avsløring av konfidensialiteten til en saksbehandler får man oversikt over alle registrerte saker i systemet, og ved å kompromittere integriteten til en administrator kan man opprette en ny saksbehandler. Deretter kan man logge inn som denne saksbehandleren og dermed avsløre konfidensialiteten til varslere.

Som Tabell 4.1 viser, er det en del risikoer som ikke er blitt testet. Grunnen til dette er at noen av disse risikoene ikke lar seg teste, eller at tester for lignende risikoer er blitt gjennomført tidligere. Tabell 4.2 gir en oversikt over hvilke risikoer som ikke er testet, og en forklaring til hver av disse.

Risiko	Forklaring
R-10	Denne risikoen omhandler valg av saksbehandlere og administratorer. Risikoen har fått risikonivå *Veldig høy*, men testing av denne ligger utenfor omfanget til denne oppgaven. For å redusere denne risikoen, anbefales det at det settes opp retningslinjer som saksbehandlere og administratorer må følge. I tillegg kan det være fordelsmessig å kurse disse brukerne, slik at de er klar over hva de kan gjøre i systemet.
R-2	Denne risikoen omhandler integriteten til varslere. Grunnen til at denne risikoen ikke er blitt testet er at test T-1 testet sikkerheten til PIN koden. Testing av PIN koden er også sentralt for R-2, og for å unngå dobbeltarbeid har dette prosjektet valgt å bruke tid på å teste andre risikoer.
R-5	Denne risikoen omhandler integriteten til saksbehandlere. Grunnen til at denne risikoen ikke er blitt testet er at test T-1 testet sikkerheten bak innloggingen til saksbehandlere. Siden dette også er sentralt for R-5, har dette prosjektet valgt å fokusere på andre risikoer.
R-6	Denne risikoen omhandler tilgjengeligheten til ”Sikker Varsling” for saksbehandlere. Grunnen til at denne risikoen ikke er blitt testet, er at hovedtrusselen, DoS angrep, har blitt testet i test T-6. Med hensyn til tidsbruken, har denne risikoen ikke blitt testet.
R-7	Denne risikoen omhandler konfidensialiteten til administratorer. Grunnen til at denne risikoen ikke er blitt testet er at innloggingen til administratorer, som er hovedfokuset for denne risikoen, er blitt testet i test T-1.
R-9	Denne risikoen omhandler tilgjengeligheten til ”Sikker Varsling” for administratorer. Grunnen til at denne risikoen ikke er blitt testet, er den samme som for risiko R-6.
R-13	Denne risikoen omhandler ansatte som lager bevisste eller ubevisste ”bakdører” til testobjektet. For å teste dette er det viktig å inkludere utviklerne. Siden dette ikke var mulig, har ikke denne risikoen blitt testet.
R-15	Denne risikoen omhandler dårlige vedlikeholdsrutiner hos eieren av systemet (Kantega AS). Testing av denne risikoen ligger utenfor omfanget til dette prosjektet. Det anbefales uansett å teste en slik risiko i senere sikkerhetstesting av ”Sikker Varsling”.
R-17	Denne risikoen omhandler dårlige vedlikeholdsrutiner hos driftsoperatøren. I likhet med risiko R-15, ligger testing av denne risikoen utenfor omfanget av prosjektet. Det anbefales at en mer omfattende sikkerhetstest inkluderer en undersøkelse av vedlikeholdsrutinene for å redusere en slik risiko.

Tabell 4.2: Oversikt over risikoer som ikke er testet

De risikoene som ble rangert lavere enn risiko R-29 i Tabell 3.4, har ikke blitt testet på grunn av tidsbegrensningen i prosjektet. Dette er en av hovedgrunnene til at en risikostyring ble foretatt; for å fokusere testingen på de mest kritiske risikoene.

4.2. Testbeskrivelse

Dette avsnittet beskriver hva som er blitt testet, hvilke testmetoder og testverktøy som er vurdert, og hvilke resultat som ble forventet. Avsnittet er delt opp i åtte underavsnitt, en for hver av testene i Tabell 4.1.

4.2.1. Test T-1: Konfidensialitet varsler

Omfang
Konfidensialiteten til en varsler er i hovedsak beskyttet av en PIN kode. Testing av denne PIN koden har derfor vært sentralt i denne testingen. I tillegg er konfidensialiteten til saksbehandlere og integriteten til administratorer viktige i denne testen. Disse er beskyttet av et brukernavn og passord, og testing av sikkerheten rundt innloggingen har derfor også vært viktig i denne testen.

Testmetode
Det er i hovedsak blitt gjennomført manuelle tester for å finne svakheter i både PIN koden, brukernavn og passord. I tillegg har det blitt brukt testverktøy for å assistere den manuelle testingen.

Testverktøy
- Cain & Abel – Dette testverktøyet er beskrevet i appendiks A.1.

Forventet resultat
Et forventet resultat var at konfidensialiteten til varslere ikke kunne bli avslørt. Dette er ikke mulig å bevise, men formålet med testingen var å gi grunnlag for å øke sannsynligheten for at det ikke er mulig.

4.2.2. Test T-2: Cross Site Scripting (XSS)

Omfang
Denne testen har sjekket om input fra brukeren blir skikkelig validert på serversiden, for å unngå XSS sårbarheter. Testen har konsentrert seg om XSS, og ikke fokusert på Cross Site Request Forgery (CSRF) som var en tilleggsrisiko for denne testen.

Testmetode
Testmetoden har i hovedsak vært dynamisk, hvor både en manuell og automatisk fremgangsmåte har blitt brukt. Den manuelle delen har i hovedsak vært testing av inputfelter i ”Sikker Varsling”, mens den automatiske testingen har brukt testverktøy for å avdekke potensielle sårbarheter som ikke ble funnet under den manuelle testingen.

Testverktøy
- Paros – Dette testverktøyet er beskrevet i appendiks A.2.
- Acunetix WVS – Dette testverktøyet er beskrevet i appendiks A.3.

Forventet resultat
Det forventede resultatet var at ”Sikker Varling” ikke er sårbar for XSS. Resultatet av testingen har blitt dokumentert med tanke på hvordan testobjektet har håndtert problemet med XSS.

4.2.3. Test T-3: Injeksjonsfeil

Omfang
Selv om risiko R-21 dekker injeksjonsfeil generelt, begrenset denne testen seg til SQL injeksjoner. SQL injeksjoner er en risiko for “Sikker Varsling” siden systemet tar imot input fra brukeren som blir lagret i en database. Slike inputfelt var relevante for denne testen.

Testmetode
Til å begynne ble det brukt automatiske testverktøy for å skaffe oversikt over potensielle injeksjonsproblemer i “Sikker Varsling”. Deretter ble det gjennomført en manuell test for å se om eventuelle rapporterte problemer virkelig var sårbare for SQL injeksjoner.

Testverktøy
- Paros – Dette testverktøyet er beskrevet i appendiks A.2.
- OWASP SQLiX v1.0 – Dette testverktøyet er beskrevet i appendiks A.4.
- SQLBrute – Dette testverktøyet er beskrevet i appendiks A.5.

Forventet resultat
Resultatet var forventet å vise at ingen inputfelt er sårbar for SQL injeksjoner. Dokumenteringen av resultatet skal vise hvordan SQL injeksjoner er unngått.

4.2.4. Test T-4: Sesjonshåndtering

Omfang
Denne testen baserte seg på mye av det samme som i test T-1. T-1 testet sentrale autentiseringsmekanismer som brukernavn, passord og PIN kode. Denne testen fokuserte på en annen autentiseringsmekanisme; sesjonshåndteringen i ”Sikker Varsling”.

Testmetode
Denne testen brukte både en statisk og dynamisk testmetode. Den statiske metoden analyserte kildekoden for svakheter i sesjonshåndteringen, mens den dynamiske metoden brukte testverktøy for å verifisere eventuelle svakheter.

Testverktøy
- Paros – Dette testverktøyet er beskrevet i appendiks A.2.

Forventet resultatet
Et forventet resultat fra denne testen var å øke sannsynligheten for at sesjonshåndteringen er sikker, slik at det er vanskelig for andre å stjele en annens sesjon.

4.2.5. Test T-5: Sårbarheter i nettlesere

Omfang
Omfanget til denne testen er stort, og for å begrense testen fokuserte den i hovedsak på to nettlesere; Internet Explorer og Mozilla Firefox. Forskjellige versjoner av nettleserne ble brukt for å teste hvordan ”Sikker Varsling” håndterer eldre nettleserversjoner.

Testmetode
Denne testen kombinerte en statisk og dynamisk testmetode. Den statiske delen av testen bestod av å finne rapporterte sårbarheter i nettleserne fra forskjellige sårbarhetsdatabaser som nevnt i 2.3.1. Den dynamiske delen gikk ut på å teste eldre versjoner av nettleserne mot ”Sikker Varsling”.

Testverktøy
Det ble ikke brukt noen testverktøy i denne testen.

Forventet resultat
Her ble det forventet at ”Sikker Varsling” stiller krav eller anbefalinger til brukerens nettleser. Dette medfører at ”Sikker Varsling” setter krav/anbefalinger til at nettlesere er oppdatert, og gir brukeren tilbakemelding dersom dette ikke er tilfelle.

4.2.6. Test T-6: Denial of Service (DoS)

Omfang
Denne testen baserte seg på tilgjengeligheten til varslere, og gikk ut på å teste om varslerapplikasjonen var sårbar for DoS angrep.

Testmetode
Testingen baserte seg på automatiske testverktøy som gjennomførte automatiske DoS angrep mot “Sikker Varsling”.

Testverktøy
- DoSHTTP 2.0 – Dette testverktøyet er beskrevet i appendiks A.6.
- Microsoft Web Application Stress Tool – Dette testverktøyet er beskrevet i appendiks A.7.

Forventet resultat
Siden det ikke er satt konkrete krav til antall forespørsler, ble det vanskelig å sette spesifikke mål på det forventede resultatet. Det forventede resultatet ble dermed at “Sikker Varsling” må klare å håndtere mange forespørsler samtidig uten at systemet feiler.

4.2.7. Test T-7: Håndtering av feilmeldinger

Omfang
Denne testen prøvde å fremprovosere feil i “Sikker Varsling” for å teste hvordan systemet gav feilmeldinger tilbake til brukeren. Feil er i hovedsak blitt fremprovosert i inputfeltene som ved feil i innloggingen, feil i registrering av sak, og feil i URL-en.

Testmetode
Fremprovoseringen av feil ble gjort manuelt mot “Sikker Varsling”.

Testverktøy
- Paros – Dette testverktøyet er beskrevet i appendiks A.2.

Forventet resultat
Det forventede resultatet var at testen ikke avdekket feilmeldinger som gir en angriper unødvendig informasjon. Slike eventuelle feilmeldinger er dokumentert og forslag til forbedringer er gitt.

4.2.8. Test T-8: URL håndtering

Omfang
Denne testen testet om “Sikker Varsling” tillot uautorisert tilgang ved kun å skrive inn ”skjulte” URL-er. For å teste dette var det viktig å skaffe seg en oversikt over alle URL-er i systemet. Alle disse er blitt testet for et slikt angrep.

Testmetode
Testingen var dynamisk ved at URL-er ble skrevet inn manuelt og testet for uautorisert tilgang.

Testverktøy
Det ble ikke brukt noen testverktøy i denne testen.

Forventet resultat
Resultatet var forventet å vise at ingen URL-er i “Sikker Varsling” tillater uautorisert tilgang.

5. Testresultater

Dette kapittelet er i hovedsak en dokumentasjon av resultatene fra testene. Resultatene er gitt i avsnitt 5.3, og følger samme rekkefølge som testbeskrivelsene i avsnitt 4.2. I tillegg gir dette kapittelet en oversikt over avvik mellom test- og produksjonsmiljø, samt en forklaring til hvordan resultatene er dokumentert.

5.1. Avvik

Den største forskjellen på testmiljøet og produksjonsmiljøet var at testmiljøet ikke brukte kryptert kommunikasjon (https). På grunn av andre pågående tester mot testmiljøet, var det ikke mulig for dette prosjektet å få konfigurert testmiljøet.

Andre avvik mellom test- og produksjonsmiljøet har vært vanskelig å forutse i dette prosjektet. Dette skyldes i hovedsak at ”Sikker Varsling” ikke er i drift, noe som gjør det vanskelig å forutsi hvilke konfigurasjoner produksjonsmiljøet vil ha.

5.2. Testdokumentasjon

Testdokumentasjonen er i hovedsak blitt gjort tekstlig med enkelte figurer for å gi en bedre illustrasjon av resultatene. Resultatdokumentasjonen av hver test følger samme rekkefølge som underavsnittene i avsnitt 4.2, hvor testene er beskrevet.

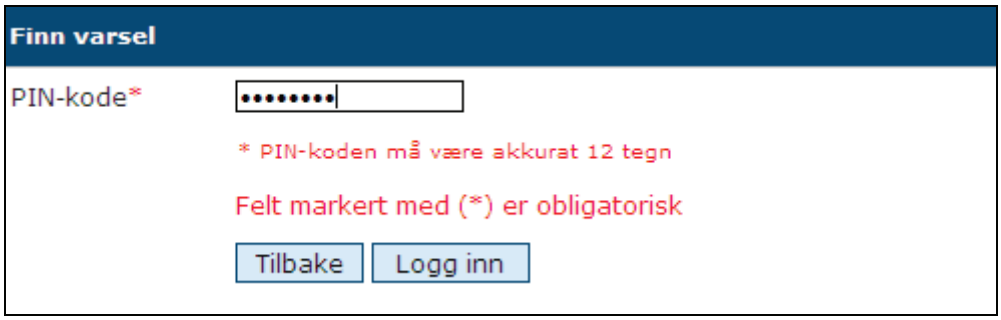
Hvert resultatavsnitt er avsluttet med en resultatsammenligning av forventet og oppnådd resultat. Hver test sitt forventede resultatet er gitt i testbeskrivelsen i avsnitt 4.2. Det skal merkes at disse forventede resultatene er ideelle resultat, og derfor vanskelige å oppnå.

5.3. Resultater

Dette avsnittet er delt inn i flere underavsnitt som hver beskriver resultatene fra testene i Tabell 4.1. Hvilke testverktøy som er brukt i hver test, er beskrevet i avsnitt 4.2. Navnene på testverktøyene er uthevet i kursiv, og beskrivelse av hvert enkelt testverktøy er gitt i Appendiks A.

5.3.1. Resultater T-1: Konfidensialitet varsler

”Sikker Varsling” gir noen feilmeldinger som gir en angriper unødvendig informasjon. De to identifiserte feilmeldingene som er funnet er vist i Figur 5.1 og Figur 5.2.



Figur 5.1: Feilmelding ved for kort PIN kode

Figur 5.2: Feilmelding ved inntasting av ulovlige tegn

Feilmeldingene omhandler PIN koden som ”Sikker Varsling” genererer for hver sak som blir varslet. Kjennskap til denne koden gir dermed full adgang til opplysningene rundt saken. De to identifiserte feilmeldingene gir en angriper informasjon om at PIN koden er 12-sifret, og kan ikke inneholde mellomrom eller spesialtegn. En bedre fremgangsmåte vil være å gi en mer generell feilmelding om at innloggingen mislyktes. Dette gir en angriper mindre informasjon, som gjør det vanskeligere å foreta angrep på PIN koden.

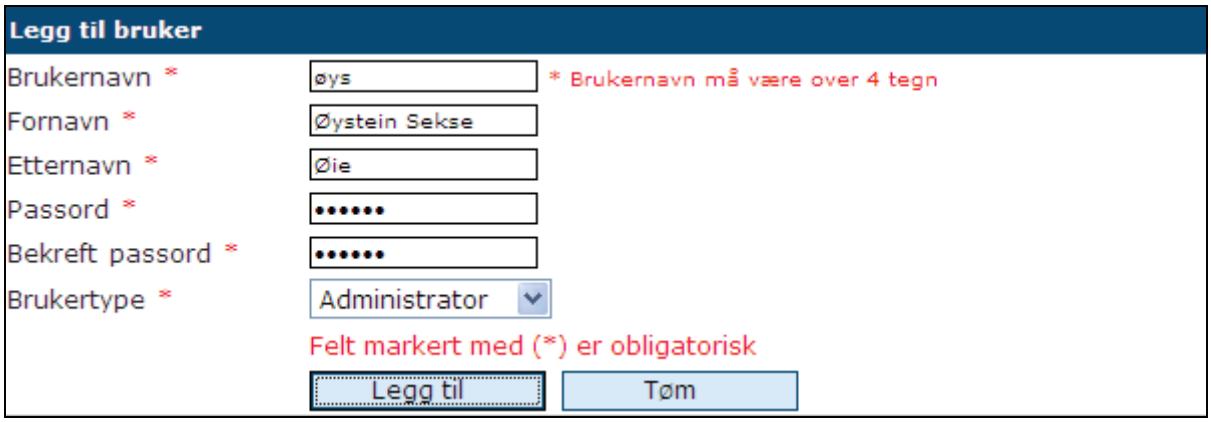
En annen svakhet ved denne innloggingen er at man kan prøve å logge inn et ubegrenset antall ganger. Systemet gir kun en feilmelding om at PIN koden ikke var riktig, og ber brukeren kontrollere koden. På denne måten kan en angriper uavbrutt prøve forskjellige koder, uten at systemet gjør noen mottiltak.

Alle svakhetene som er nevnt over fremstår ikke som store trusler. Dette er på grunn av at PIN koden inneholder hele 12 tegn. Hvert tegn kan bestå av store eller små bokstaver, samt tall. Dette gir 64 mulige varianter for hvert tegn i PIN koden. Antall kombinasjoner blir dermed:

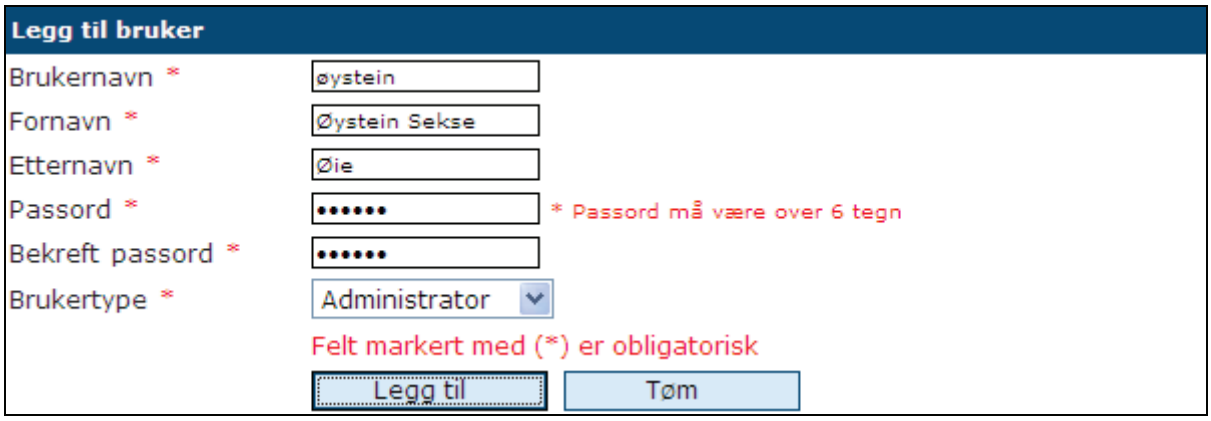
$$64^{12} = 4\ 722\ 366\ 482\ 869\ 645\ 213\ 696$$

Hvis en regner med at en datamaskin klarer å gjennomføre en million kombinasjoner pr sekund, vil en ”brute force” fremgangsmåte, hvor alle mulige kombinasjoner blir gjennomgått, ta ca. 150 millioner år. En 12-sifret PIN kode gir med andre ord en meget god sikkerhet for varsleren.

Denne testen dekker også testing av brukernavn og passord til saksbehandlere og administratorer. Det er administratorer som oppretter nye brukere, og dermed bestemmer de også brukernavn og passord til disse nye brukerne. Dette kan være en svakhet i systemet hvis ikke administratorer er opplært til å forstå viktigheten av sikre brukernavn og passord. Her bør med andre ord ”Sikker Varsling” hjelpe til ved å sette krav til brukernavn og passord. Under testingen av dette ble det funnet noen svakheter i disse kravene. Kravene som settes av systemet fremmer ikke oppretting av sikre brukernavn og passord, som Figur 5.3 og Figur 5.4 viser.



Figur 5.3: Krav til brukernavn

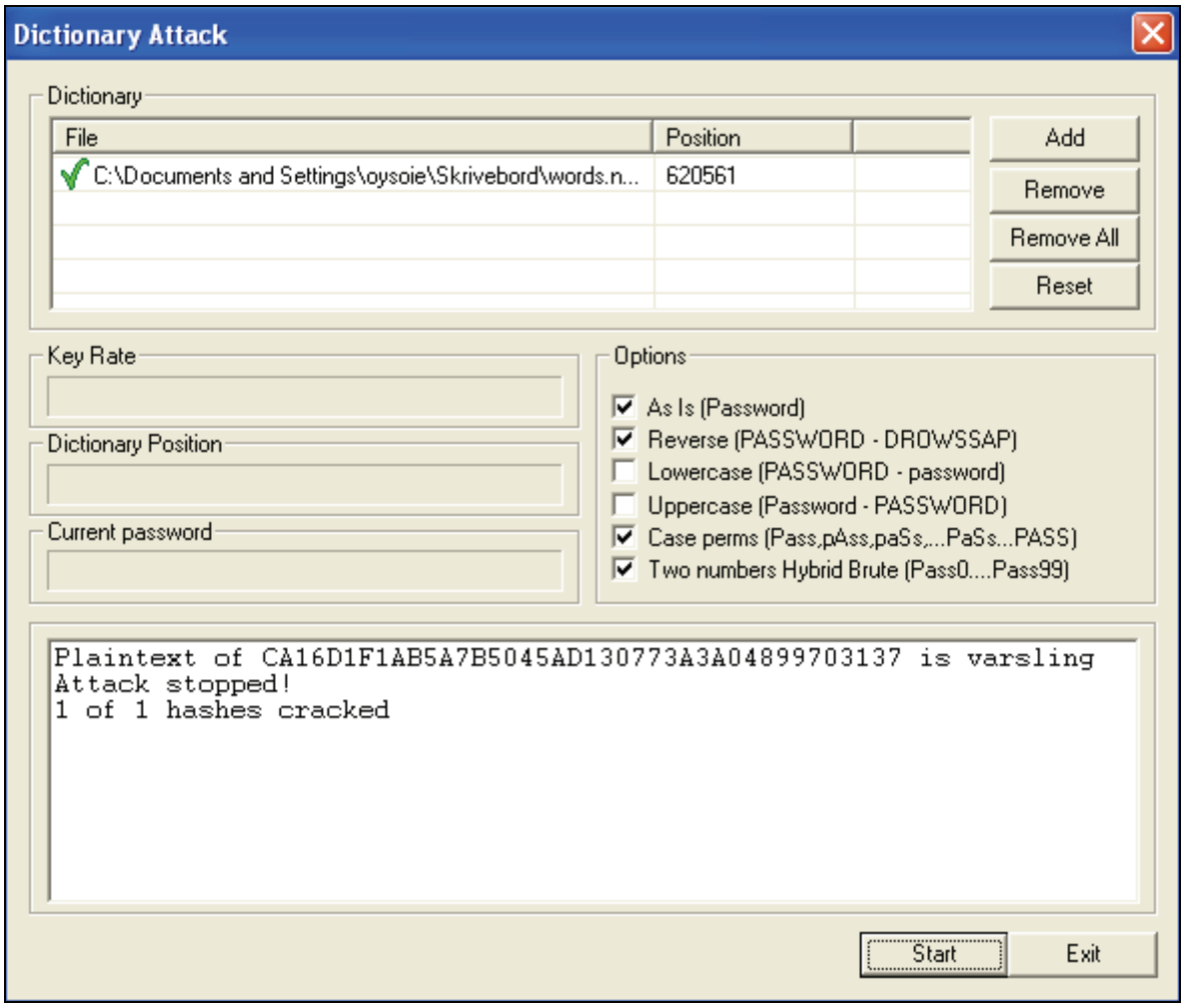


Figur 5.4: Krav til passord

Som figurene viser er kravet til brukernavnet at det må være over 4 tegn, mens passordet må være over 6 tegn. Dette er ikke spesielt strenge krav, og kan lett føre til at administratorer velger svake brukernavn og passord. Et eksempel på en lovlig kombinasjon er brukernavn *admin* og passord *varsling*. Dette er ikke en sikker kombinasjon, og kan fort avsløres av en angriper.

For å illustrere dette, ble testverktøyet *Cain & Abel* brukt til å finne passordet *varsling*, hvor passordet var hashet med hash-funksjonene SHA-1 og MD5. SHA-1 er den mest populære hash-funksjonen i dag, mens MD5 ofte blir brukt i eldre applikasjoner.[17] Figur 5.5 viser resultatet av ordbokangrepet på SHA-1-verdien av *varsling*. Dette angrepet tok litt over 37 minutter, noe som viser at et slikt svakt passord er rimelig enkelt å avsløre. Det samme angrepet ble utført på passordet hashet med MD5. Dette angrepet tok 29 minutter. Ordlisten som ble brukt inneholdt 61843 norske ord, og finnes fritt tilgjengelig på internett.[10]

[10] Ordlisten finnes på: http://www.word-list.com/



Figur 5.5: Ordbokangrep på hashet passord

Som Figur 5.5 viser, gjennomfører angrepet et søk med mange forskjellige varianter av hvert ord i ordlisten. Foruten å søke på ordet som det er, søker det på ordet i omvendt rekkefølge, alle mulige kombinasjoner av store og små bokstaver, og med et tall fra 1 til 99 bak ordet. Dermed ville et lignende søk også ha funnet passordet *varsLING, Varsling1, VarslinG99, osv*.

Dette eksempelet antar at angripere får tak i et hashet passord til en bruker. Dette kan for eksempel skje hvis en angriper klarer å lese ut data fra en database. Ved hjelp av lengden på hashen, kan angriperen begrense mulige hash-funksjoner som er brukt.

Selv om det er administratorer som bestemmer brukernavn og passord, kan en saksbehandler selv forandre passordet sitt. Denne funksjonen har de samme kravene til passord som i Figur 5.4. ”Sikker Varsling” gir dermed ingen god støtte i valg av et sikkert passord.

For å følge *Common Criteria*[11] sitt passordkrav, anbefales det at passordet minst må være 8 tegn langt, og inneholde både alfabetiske og numeriske tegn.[18] Det anbefales også at det stilles strengere krav til brukernavnet. Her kan man følge retningslinjene til USAs departement for helse- og personalressurser[19], og stille krav til at brukernavnet minst må være 6 tegn langt.

[11] Hjemmeside: http://www.commoncriteriaportal.org/

I likhet med PIN koden, har også innloggingen for administratorer og saksbehandlere en svakhet ved at man kan prøve å logge inn et ubegrenset antall ganger. Systemet gir bare en feilmelding, uavhengig av antall ganger man har prøvd å logge inn.

Resultatsammenligning
Denne testen avdekte enkelte sårbarheter i systemet som ikke styrker sikkerheten rundt konfidensialiteten til varslere. Dermed ble ikke det forventede resultatet oppnådd.

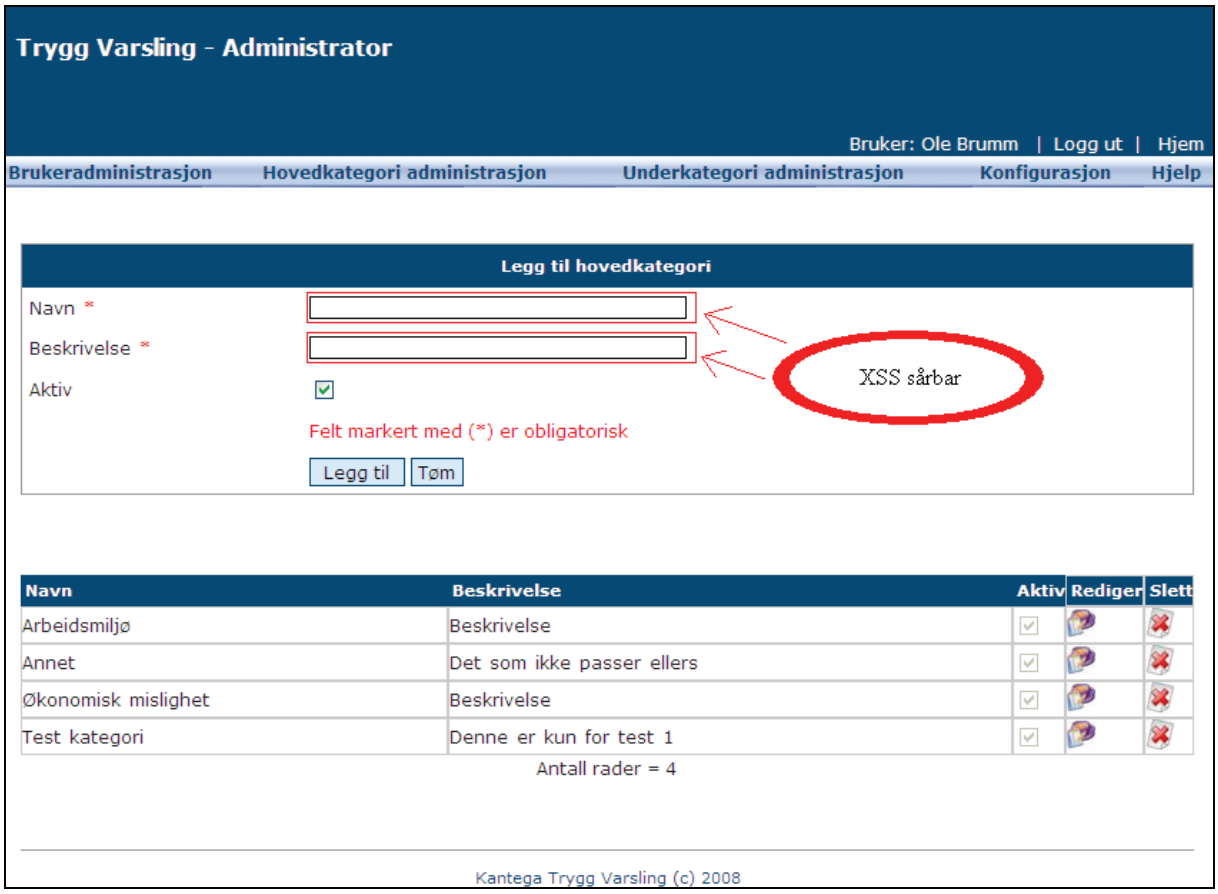
5.3.2. Resultater T-2: Cross Site Scripting (XSS)

For å unngå XSS sårbarheter, bruker "Sikker Varsling" følgende validering på inputfeltene:

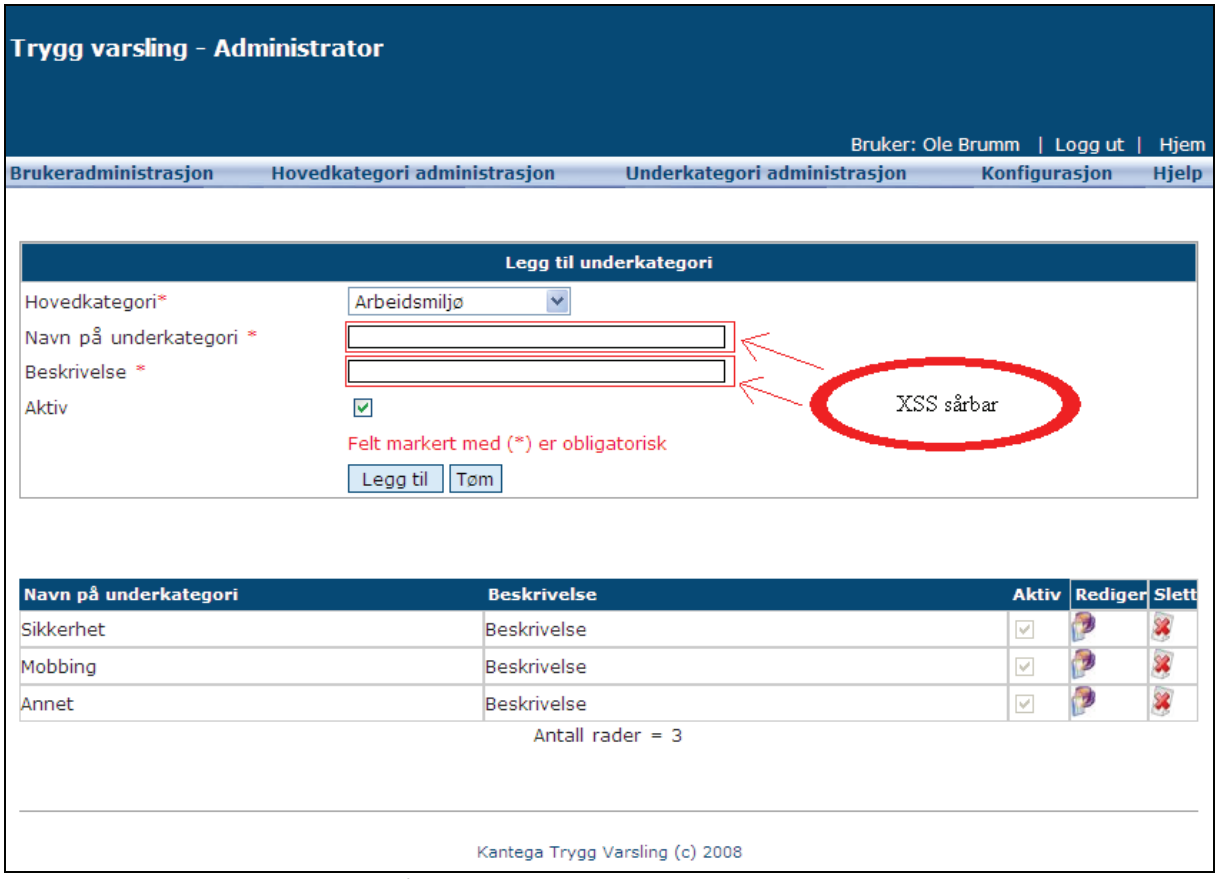
```
public static boolean validate(String data) {
  if(data != null && data.length() > 0) {
      for (int i = 0; i < data.length(); i++) {
          if(data.charAt(i) == '&'
                  || data.charAt(i) == '<'
                  || data.charAt(i) == '>'
                  || data.charAt(i) == '"'
                  || data.charAt(i) == '%'
                  || data.charAt(i) == '\''
                  || data.charAt(i) == '('
                  || data.charAt(i) == ')'
                  || data.charAt(i) == ';'
                 ) {
              return true;
          }
      }
  }
  return false;
}
```

Denne valideringen er en såkalt "blacklist" fremgangsmåte, hvor man oppretter en liste med uønskede tegn som ikke blir tillatt som input til "Sikker Varsling". Dette er ifølge Sverre H. Huseby[15] feil fremgangsmåte, sett fra et sikkerhetsperspektiv. Huseby argumenterer med at en slik fremgangsmåte er sårbar ved at man glemmer eller ikke er klar over tegn som kan utgjøre en risiko for applikasjonen. En bedre fremgangsmåte er såkalt "whitelisting". Her oppretter man en liste over alle godkjente tegn, og betrakter resterende tegn som uønskede tegn. Dette gir en større trygghet for at inputen ikke inneholder kode som utnytter applikasjonen.

De fleste inputfeltene i "Sikker Varsling" blir validert med koden gitt ovenfor, men brukere med administratorrettigheter kan utnytte to sårbarheter i systemet. Sårbarhetene oppstår i administreringen av hovedkategorier og underkategorier. Figur 5.6 og Figur 5.7 viser hvordan en administrator kan administrere disse kategoriene, og viser hvilke felter som er sårbare.



Figur 5.6: XSS sårbare inputfelt i administrering av hovedkategorier

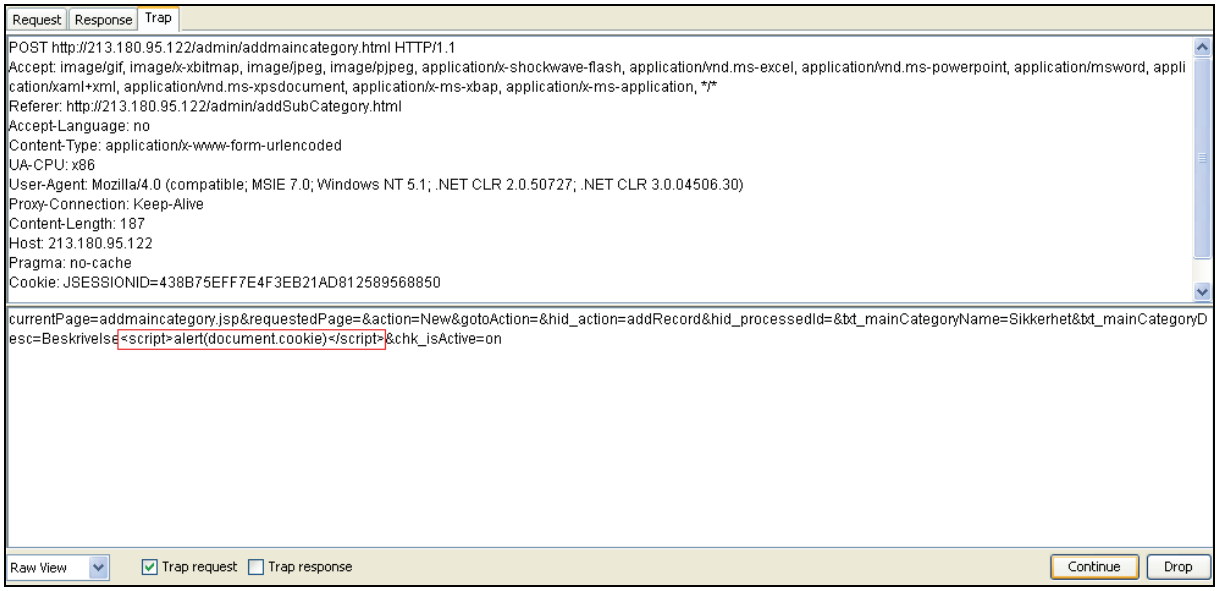


Figur 5.7: XSS sårbare inputfelt i administrering av underkategorier

De fire inputfeltene som er uthevet i figurene, blir kun validert på klientsiden. Dette betyr at testobjektet forhindrer XSS direkte i nettleseren, men er sårbar for XSS hvis man forandrer forespørselen som sendes av nettleseren. Det finnes mange programmer for å stoppe og forandre forespørsler, og under testingen ble *Paros* brukt. Testskriptet som ble brukt var et enkelt skript som gir en alarmboks som innholder sesjonsidentifikasjonen til den innloggede administratoren:

<script> alert(document.cookie) </script>

Figur 5.8 viser hvordan skriptet ble brukt i *Paros*. Den nederste delen av skjermen er en teksteditor hvor man fritt kan forandre de forespørsler som sendes fra nettleseren. Den røde firkanten viser hvor i forespørselen skriptet ble satt inn. Ifølge figuren blir skriptet satt inn i beskrivelsen av en hovedkategori fra Figur 5.6.



Figur 5.8: Innsetting av testskript i Paros

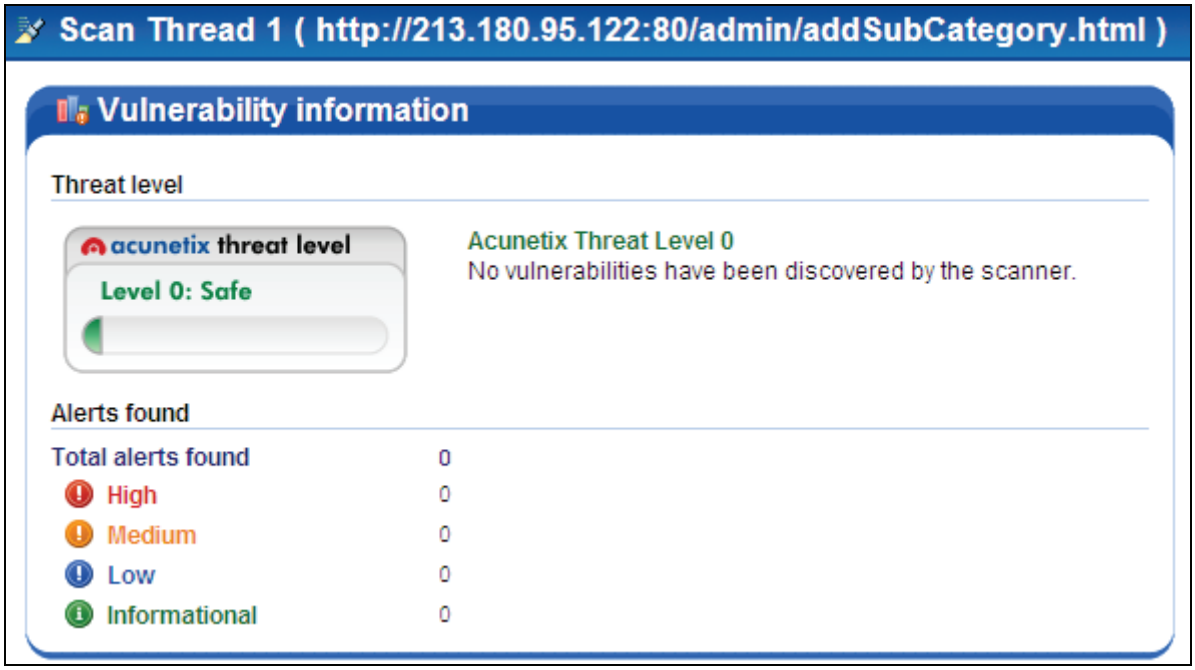
Ved å forandre forespørselen slik som Figur 5.8 viser, blir skriptet lagret sammen med hovedkategorien som blir opprettet. Dermed kommer det en alarmboks som vist i Figur 5.9 opp hver gang en administrator åpner siden med hovedkategorier.

Figur 5.9: Alarmboks ved innsetting av testskript

Dette testskriptet ble brukt som eksempel for å vise hvordan disse inputfeltene er sårbare for XSS. Testskriptet gir som sagt bare den innloggede administratoren en alarmboks med sin egen sesjonsidentifikasjon. Et mer avansert skript kan gjøre mye mer skade, ved for eksempel å sende identifikatoren til en angriper. Angriperen kan dermed bruke denne identifikatoren, og opptre som en bruker med administratorrettigheter.

Som en bemerkning kan også *Paros* brukes til å forandre forespørsler og svar over https. Dette kan enkelt gjøres siden all kommunikasjon som vises i *Paros* er i klartekst. Dermed vil "Sikker Varsling" også være sårbar for disse angrepene når systemet kjører over https.

I den automatiske delen av testingen, ble *Acunetix WVS* brukt. Dette testverktøyet fant ingen sårbarheter, heller ikke når det ble brukt til å teste sidene fra Figur 5.6 og Figur 5.7. I testingen av websiden hvor man administrerer hovedkategoriene, gav verktøyet resultatet som vist i Figur 5.10.



Figur 5.10: Resultat fra testing med Acunetix WVS

Som figuren viser, fant ikke *Acunetix WVS* noen XSS sårbarheter. Tilbakemeldingen verktøyet gav var at websiden er sikker. *Acunetix WVS* fant heller ingen sårbarheter under testingen av websiden hvor man administrerer underkategorier.

Resultatsammenligning
På grunn av de fire sårbare inputfeltene og feil fremgangsmåte for inputvalidering, har ikke det forventede resultatet blitt oppnådd.

5.3.3. Resultater T-3: Injeksjonsfeil

For å unngå SQL injeksjoner, bruker "Sikker Varsling" samme validering av input som ved validering av XSS. Kildekoden for valideringen er vist øverst i avsnitt 5.3.2. Som for XSS, er denne valideringen en svakhet i "Sikker Varsling", ved at fremgangsmåten er en "blacklist" av ulovlige tegn. For best mulig å unngå SQL injeksjoner, bør det brukes en "whitelist" fremgangsmåte, hvor inputen blir validert på bakgrunn av lovlige tegn.

Den automatiske testingen av injeksjonsfeil i ”Sikker Varsling” fant ingen sårbarheter. På grunn av tidsbegrensningen testen hadde, fikk den enkelte begrensninger:

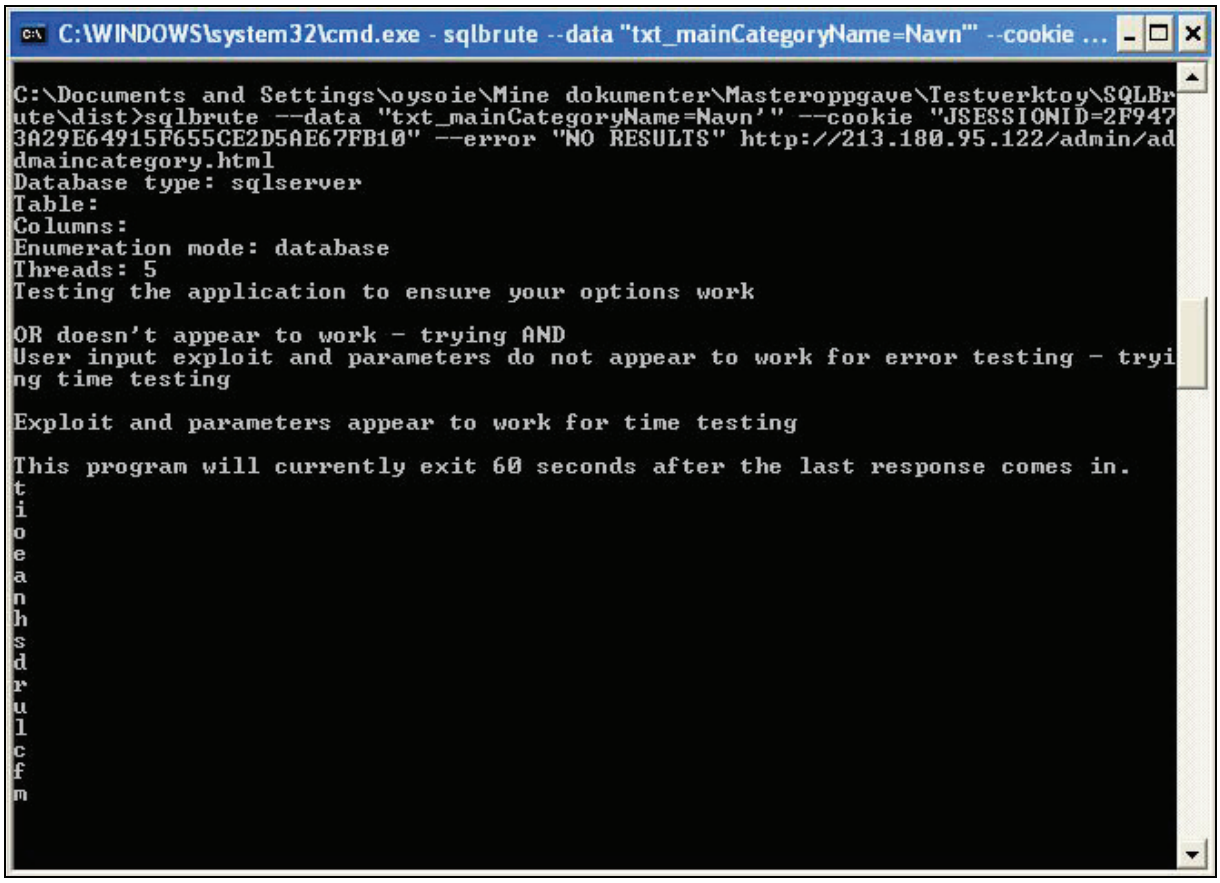
- Alle websidene som inngår i ”Sikker Varsling” er ikke blitt testet.
- På grunn av manglende dokumentasjon av testverktøyene, er det ikke sikkert testverktøyene ble utnyttet på best mulig måte.

I testingen med verktøyet *SQLiX*, avdekket ikke verktøyet noen sårbarheter. Et eksempel er testingen av websiden hvor brukere registrerer en sak. Denne siden er vist i Figur 3.5 og inneholder flere inputfelt som gjør SQL injeksjoner relevante.

Figur 5.11 viser resultatet av testingen. Som figuren viser, går *SQLiX* gjennom alle variablene som blir gitt som input til testen. For hver av disse, prøver verktøyet flere metoder for finne injeksjonsfeil.

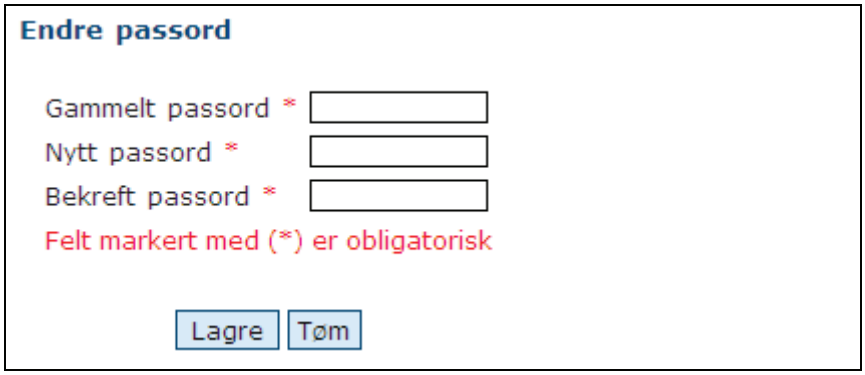
Figur 5.11: Resultat fra testverktøyet SQLiX

Heller ikke *SQLBrute* fant noen sårbarheter. Figur 5.12 viser et av resultatene fra testingen med dette verktøyet. Dette søket prøver å utnytte eventuelle sårbarheter i websiden vist i Figur 5.6. Søket tar for seg inputfeltet *Navn*, og prøver å utnytte feltet ved hjelp av forskjellige injeksjonsmetoder. Som Figur 5.12 viser, gav ikke *SQLBrute* noe fornuftig resultat.



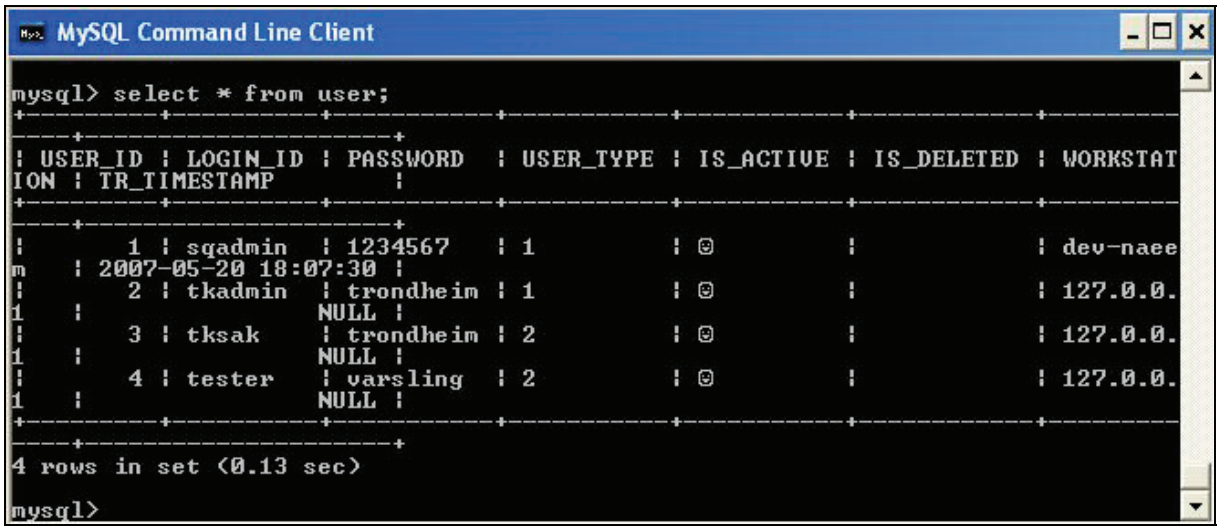
Figur 5.12: Testresultater fra SQLBrute

Den manuelle testingen tok for seg enkelte inputfelt, og testet for SQL injeksjoner. Denne testen fant en sårbarhet i saksbehandlerapplikasjonen. I websiden hvor saksbehandlere kan forandre sitt eget passord, blir det ikke gjort inputvalidering på serversiden. Den eneste valideringen som blir gjort er på klientsiden, og som for testingen av XSS, kan *Paros* brukes til å forandre http-forespørslene etter de er sendt fra nettleseren. Inputfeltene som inngår i denne websiden er vist i Figur 5.13.

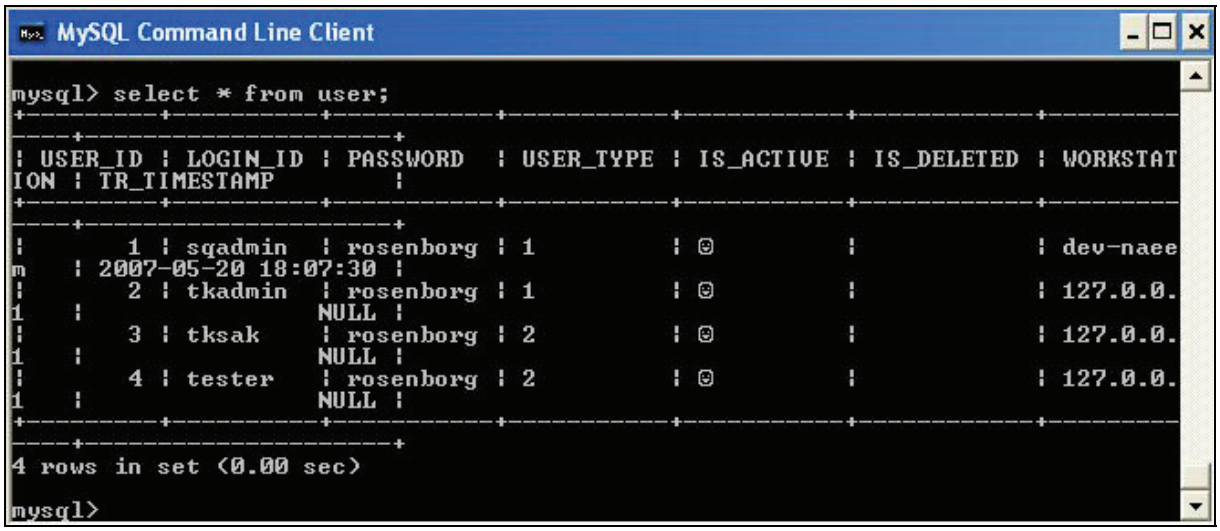


Figur 5.13: Inputfelt ved endring av passord

Som et eksempel på denne sårbarheten, ble det gjennomført et angrep som endret alle passordene for både saksbehandlere og administratorer. Figur 5.14 viser brukerne og deres passord fra databasen før angrepet ble utført, mens Figur 5.15 viser brukere og passord etter angrepet var utført.



Figur 5.14: Brukernavn og passord før angrepet ble utført

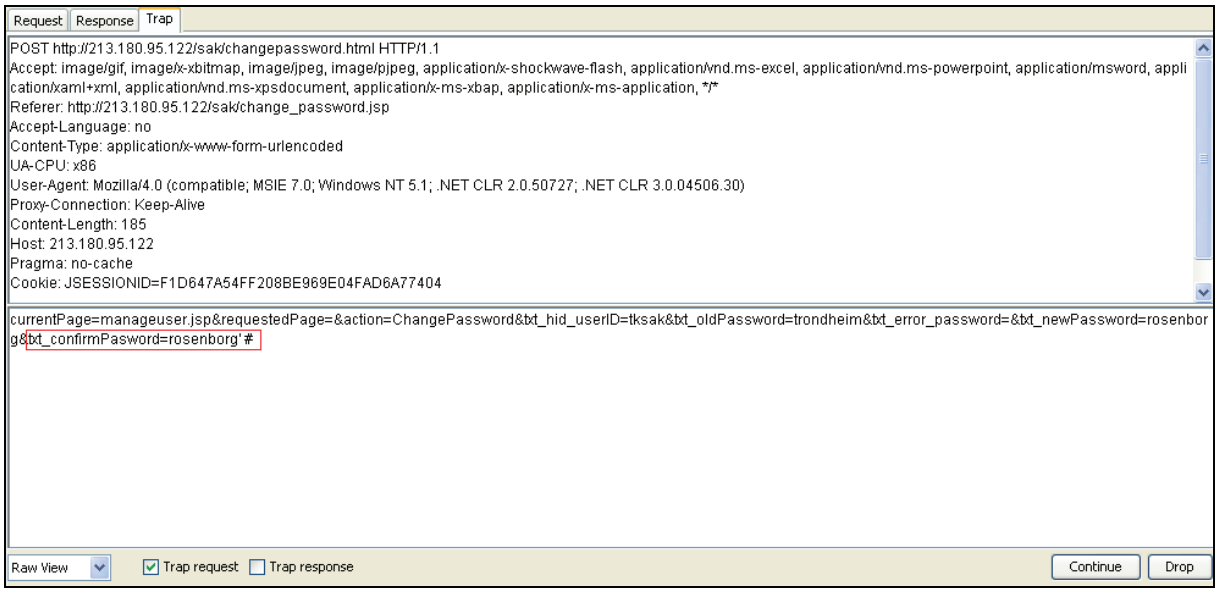


Figur 5.15: Brukernavn og passord etter angrepet var utført

Som figurene viser ble alle passordene satt til *rosenborg*. Måten dette ble gjort på var å bruke *Paros* til å sette inn ' # i inputfeltet hvor passordet bekreftes. Dermed ble spørringen mot databasen:

UPDATE user SET PASSWORD='rosenborg' # WHERE LOGIN_ID='tksak'

Alt etter #-tegnet blir betraktet som kommentarer i databasespørringen, og dermed blir alle passord satt til *rosenborg*. Figur 5.16 viser hvordan forespørselen blir endret i *Paros*. Den røde firkanten viser inputfeltet som blir forandret.



Figur 5.16: Innsetting av tegn i Paros

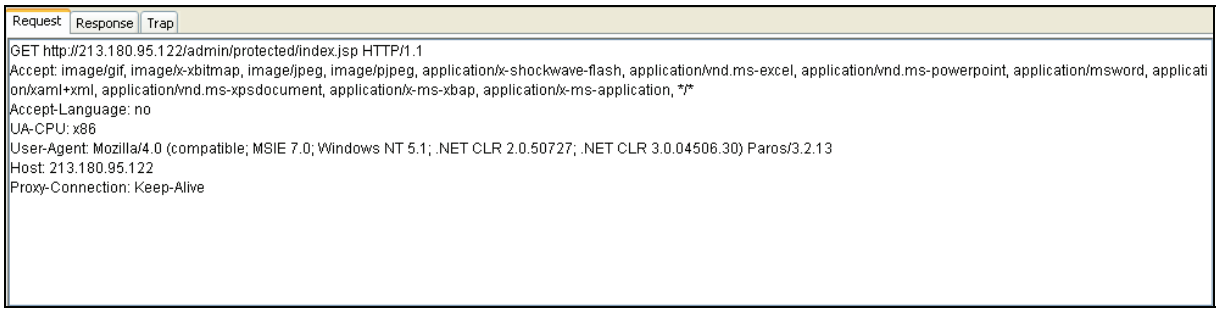
Dette kunne ha vært unngått ved hjelp av inputvalidering på serversiden. På den måten hadde ikke tegnet ’ blitt tillatt, og dermed hadde ikke databasespørringen blitt utført.

En annen ting som også kunne ha forhindret dette angrepet var hvis inputfeltene *Nytt Passord* og *Bekreft passord* hadde blitt sammenlignet på serversiden. På den måten hadde man oppdaget at inputen til disse to feltene var forskjellige, og man kunne dermed ha gitt brukeren en feilmelding tilbake. En slik fremgangsmåte er ikke sikker, siden en angriper kan legge inn de samme tegnene i begge feltene. På den måten vil det ikke bli funnet noen forskjell i sammenligningen. Inputvalidering på serversiden blir derfor den korrekte løsningen for å unngå slike angrep.

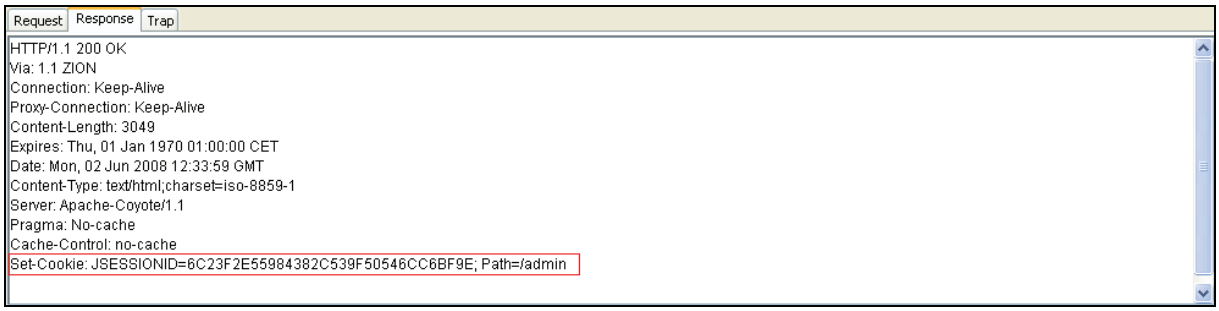
Resultatsammenligning
På grunn av den sårbare endre passord funksjonen og feil fremgangsmåte for inputvalidering, har ikke det forventede resultatet blitt oppnådd.

5.3.4. Resultater T-4: Sesjonshåndtering

Sesjonshåndteringen er gjort ved hjelp av informasjonskapsler (eng: cookies). I disse blir det lagret en identifikator, *JSESSIONID*, som brukes i kommunikasjonen mellom brukere og serveren. Denne identifikatoren blir satt av serveren etter at en bruker sender sin første forespørsel til ”Sikker Varsling”. Figur 5.17 og Figur 5.18 viser et eksempel på hvordan en slik identifikator blir opprettet. Figur 5.17 viser forespørselen som en bruker sender for å komme til innloggingssiden for administratorer. Responsen fra serveren er vist i Figur 5.18 og følger spesifikasjonen RFC 2109[20], som omhandler tilstandshåndteringen rundt HTTP. Den røde firkanten i Figur 5.18 viser hvordan sesjonsidentifikatoren blir satt for denne eksempelsesjonen.



Figur 5.17: Forespørsel til administratorsidene



Figur 5.18: Tilhørende respons fra serveren

Som Figur 5.18 viser, blir det også satt en annen variabel av serveren. *Path=/admin* blir satt for å gi informasjonskapslene beskjed om at identifikatoren kun er gyldig for websider som ligger under folderen *admin*. For saksbehandlersidene blir denne satt til *Path=/sak*, og for varslere blir den satt til *Path=/varsler*.

For å foreta en sesjonskapring, må en angriper for det første finne en gyldig sesjonsidentifikator. En gyldig identifikator er en identifikator som allerede er i bruk og som "Sikker Varsling" gjenkjenner som ekte. Som et eksempel brukes identifikatoren fra Figur 5.18. Hvis denne brukeren logger inn som en administrator, blir identifikatoren gyldig. I en slik situasjon er den sårbar for sesjonskapring. Dersom en angriper får tak i denne identifikatoren, kan angriperen enkelt opptre som administratoren ved å legge til linjen

Cookie: JSESSIONID=6C23F2E55984382C539F50546CC6BF9E

i hver forespørsel til serveren.

Siden sesjonsidentifikatoren består av 32 heksadesimale tegn, blir det praktisk umulig å gjette seg frem til en gyldig identifikator. Antall mulige kombinasjoner blir 16^{32}, og hvis man antar at man kan utføre en million kombinasjoner per sekund, vil det ta titusen trilliarder (10^{25}) år å søke gjennom alle. Dette blir dermed betraktet som meget sikkert.

En annen ting som er bra i sesjonshåndteringen, er at identifikatoren ikke blir lagret i hurtigbufferen. Fra Figur 5.18 er det er tre viktige settinger som sørger for dette:

Expires: Thu, 01 Jan 1970 01:00:00 CET
Pragma: No-cache
Cache-Control: no-cache

De to første er nødvendige for å sikre at ingen hurtigbuffere (og proxy-servere) som bruker HTTP/1.0 lagrer identifikatoren. Den siste settingen sørger for det samme, bare for hurtigbuffere (og proxy-servere) som bruker HTTP/1.1.

Til tross for en tilsynelatende god sesjonshåndtering, avdekket testingen en sårbarhet som svekker sikkerheten. Sårbarheten gjelder for genereringen av sesjonsidentifikatoren for administratorer og saksbehandlere. Identifikatoren blir gitt til begge disse brukerne før de logger inn, og blir ikke forandret ved en vellykket innlogging. Dette gjør angrepet "session fixation", som er beskrevet i [21], mulig. Et slikt angrep tar sikte på at en angriper får tak i en gyldig sesjonsidentifikator ved å gå inn på innloggingssidene til enten en administrator eller saksbehandler. Deretter må angriperen lure en gyldig bruker til å logge inn med denne sesjonsidentifikatoren, slik at identifikatoren representerer en gyldig bruker. Dermed er det fritt fram for angriperen å bruke identifikatoren for å opptre som en gyldig bruker.

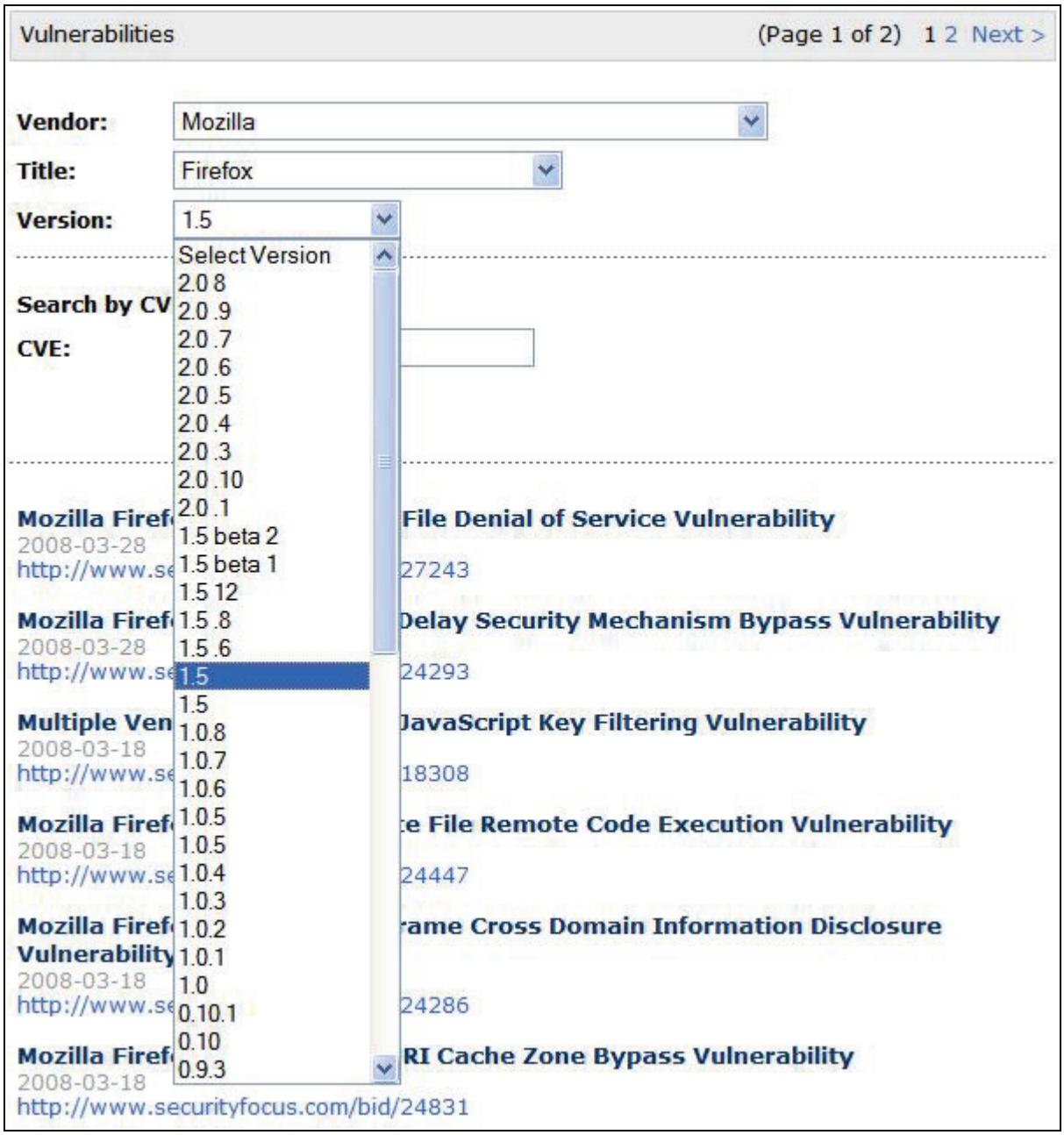
Løsningen på dette problemet er relativt enkelt. For å hindre et "session fixation" angrep, bør sesjonsidentifikatoren forandres ved en vellykket innlogging. Dermed vil ikke en angriper få tilgang til potensielle gyldige identifikatorer som kan brukes til å lure gyldige brukere.

Resultatsammenligning
Resultatet fra denne testen viste mye positivt rundt sesjonshåndteringen. Det som svekker dette synet, er sårbarheten rundt genereringen av identifikatoren. Denne sårbarheten gjør at det forventede resultatet ikke har blitt oppnådd.

5.3.5. Resultater T-5: Sårbarheter i nettlesere

I den statiske delen av denne testen ble sårbarhetsdatabasen fra *SecurityFocus*[12] brukt. Denne databasen gjør det mulig å søke etter sårbarheter i spesifikke versjoner av programvare. Figur 5.19 viser hvordan man kan utføre et slikt søk.

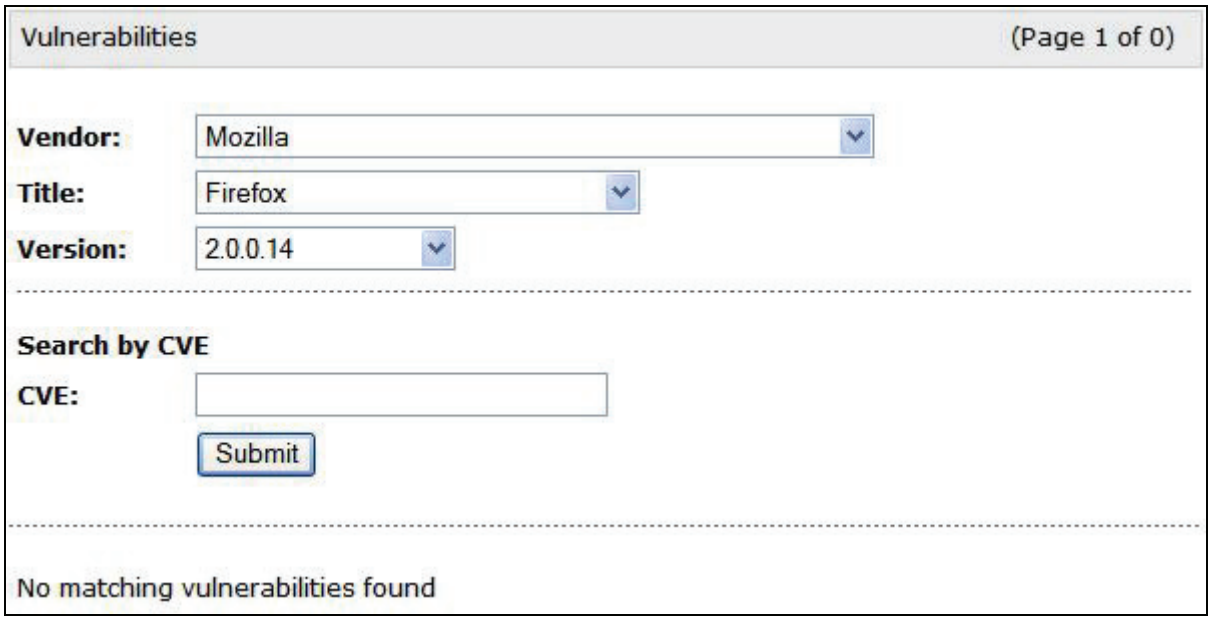
[12] Hjemmeside: http://www.securityfocus.com/vulnerabilities



Figur 5.19: Søk etter sårbarheter i Mozilla Firefox, versjon 1.5

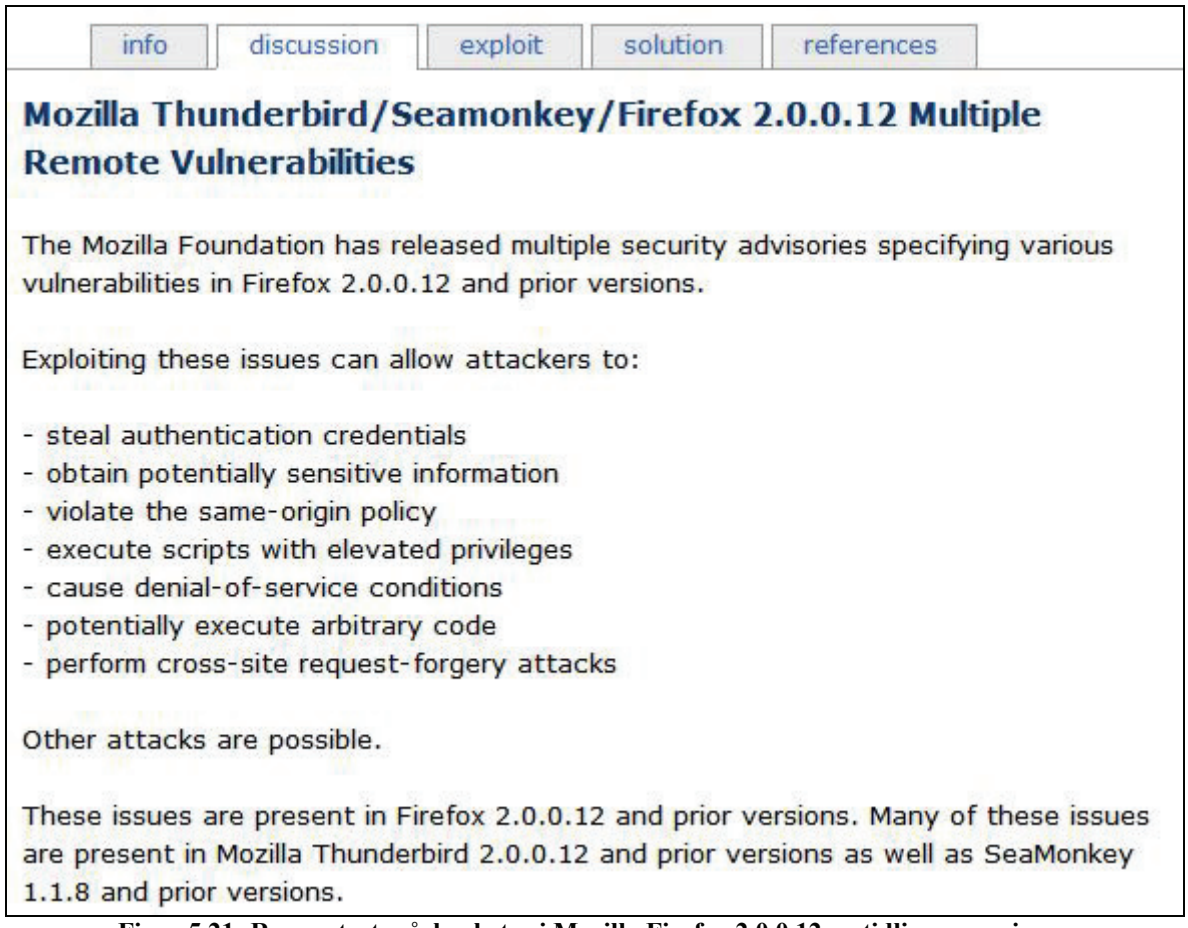
Søket i Figur 5.19 viser hvordan man kan søke etter kjente sårbarheter i Mozilla Firefox, versjon 1.5. Søket gav 43 treff som alle er rapporterte sårbarheter for versjon 1.5.

I et annet søk var målet å finne sårbarheter for Firefox sin siste versjon, 2.0.0.14. Som Figur 5.20 viser, gav dette søket ingen treff. Dette betyr ikke at versjonen er fri for sårbarheter, men at det ikke er rapportert sårbarheter for denne versjonen. Dermed har en angriper mindre informasjon om hvilke angrep man kan gjøre mot en slik nettleserversjon.



Figur 5.20: Søk etter sårbarheter i Mozilla Firefox 2.0.0.14

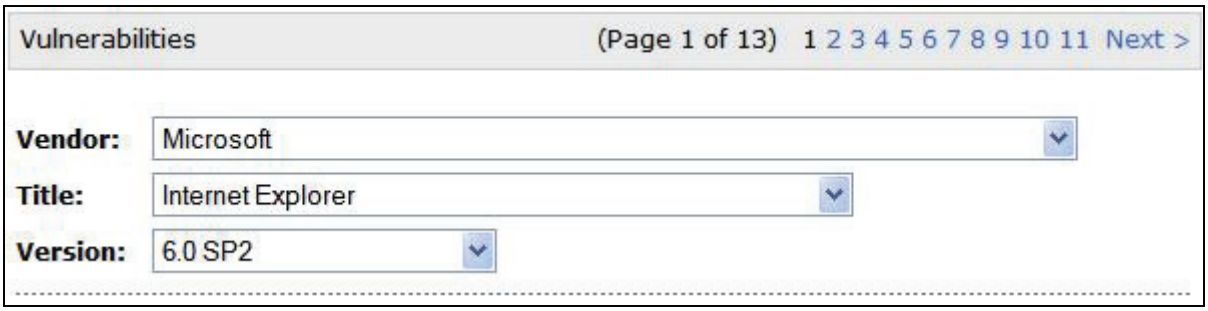
Et annet eksempel på hvorfor det er viktig for brukerne å oppdatere nettleserne, er vist i Figur 5.21. Figuren viser et av resultatene i et sårbarhetssøk i *SecurityFocus*. Som den rapporterte saken viser, har Firefox 2.0.0.12 og tidligere versjoner en del sårbarheter som en angriper kan utnytte.



Figur 5.21: Rapporterte sårbarheter i Mozilla Firefox 2.0.0.12 og tidligere versjoner

Det som figuren ikke viser, men som også er registrert i denne saken, er at Firefox 2.0.0.13 ikke er utsatt for disse sårbarhetene. Dermed viser også dette eksempelet at oppdatering av nettleseren er viktig.

Det samme gjelder for Internet Explorer (IE). For å illustrere dette viser Figur 5.22 og Figur 5.23 søkene gjort for IE6.0 og IE7.0. Figurene viser bare hvilke søkekriterier hvert søk hadde, samt antall sider med rapporterte sårbarheter.



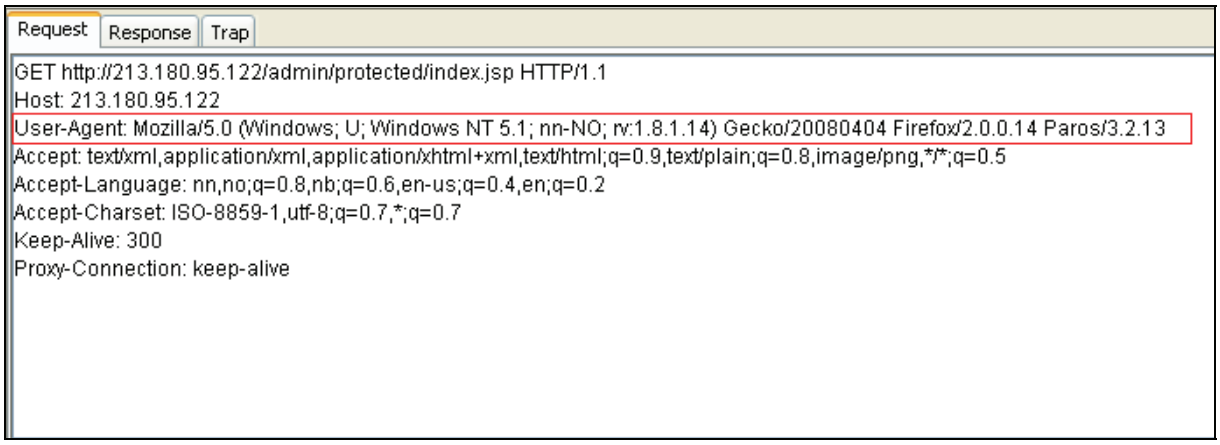
Figur 5.22: Søk etter sårbarheter i Internet Explorer 6.0

Figur 5.23: Søk etter sårbarheter i Internet Explorer 7.0

Som figurene viser er det stor forskjell på antall rapporterte sårbarheter. Søket på IE6.0 resulterte i hele tretten sider med sårbarheter (370 rapporterte sårbarheter), mens det til sammenligning bare var tre sider for søket på IE7.0 (65 rapporterte sårbarheter). Hvis brukere bruker IE7.0, vil dette gi angripere mindre informasjon om mulige angrep.

Den dynamiske delen av testen ble begrenset til å teste hvordan ”Sikker Varsling” håndterer forespørsler fra forskjellige versjoner av Internet Explorer og Firefox. I testingen ble Internet Explorer 6.0 og 7.0 testet, samt Mozilla Firefox 1.0, 1.5 og 2.0.0.14.

Resultatet av testingen viste at ”Sikker Varsling” ikke tar hensyn til hvilken nettleser som blir brukt. Alle nettleserversjonene fikk samme respons, og ingen av de eldre nettleserne fikk anbefaling eller krav om en oppdatering. En slik anbefaling eller et slikt krav er ikke vanskelig å få til. Hver forespørsel til ”Sikker Varsling” inneholder et tekstfelt som viser hvilken nettleser som blir brukt. Den røde firkanten i Figur 5.24 viser hvordan et slikt tekstfelt ser ut for Firefox 2.0.0.14.



Figur 5.24: Forespørsel fra Mozilla Firefox 2.0.0.14

Som figuren viser, inneholder det innrammede tekstfeltet teksten Firefox/2.0.0.14. Dette gir ”Sikker Varsling” informasjon om både nettleser og versjon. Ut fra dette kan applikasjonen gi tilbakemelding til brukeren hvis nettleseren bør oppdateres.

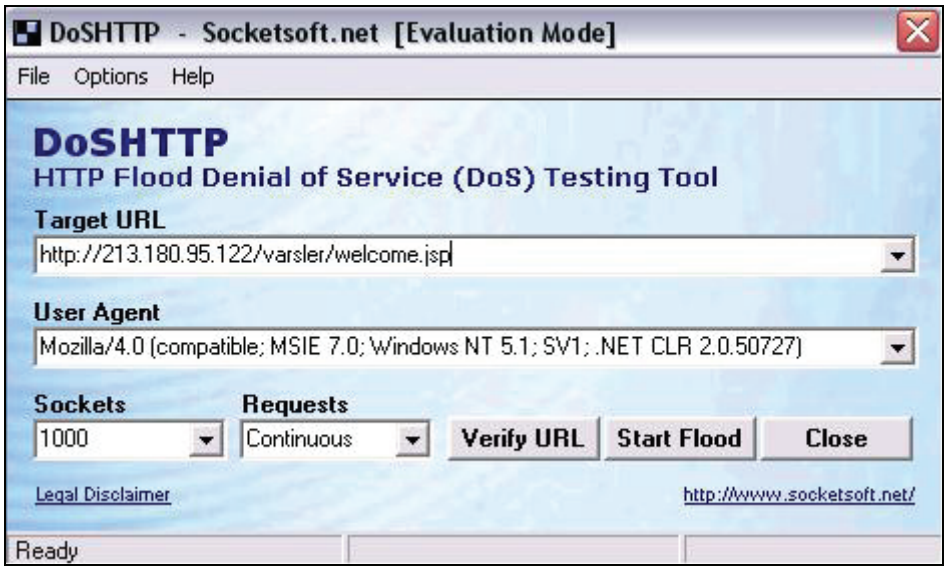
Det skal legges til at en angriper enkelt kan forandre dette tekstfeltet før forespørselen blir sendt til ”Sikker Varsling”, og dermed lure applikasjonen til å tro at en annen nettleser eller versjon blir brukt. Uavhengig av dette, er målet med tilbakemeldinger eller krav om oppdateringer basert på en gyldig bruker av ”Sikker Varsling”. Ved å få disse til å ha oppdaterte nettlesere, vil de være mindre utsatt for sårbarheter og angrep introdusert av nettleseren.

Resultatsammenligning
Siden ”Sikker Varsling” ikke stiller noen krav til brukernes nettleser, har ikke det forventede resultatet blitt oppnådd.

5.3.6. Resultater T-6: Denial of Service (DoS)

Prøveversjonen av testverktøyet *DoSHTTP* inneholdt en feil. Denne feilen gjorde seg gjeldende midt i første testgjennomkjøring, ved at programmet avsluttet seg selv uten noe forvarsel. Etter dette gav programmet beskjed om at prøvelisensen var utgått, og at man må betale for å få en utvidet lisens. Testen ble utført på to forskjellige maskiner, og begge gangene avsluttet programmet uten forvarsel.

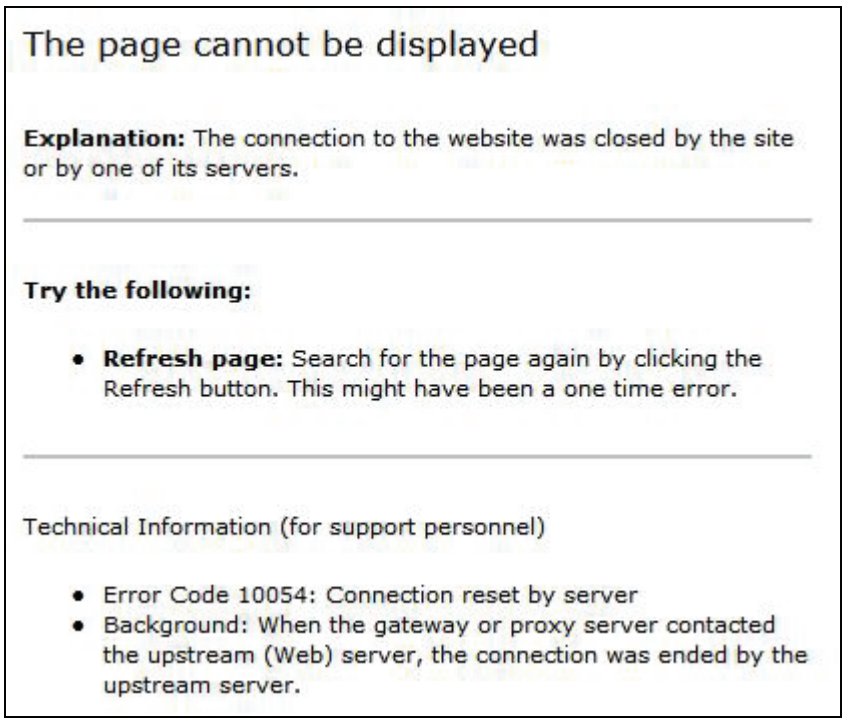
Før programmet avsluttet, ble det gjennomført to tester mot fremsiden av varslerapplikasjonen. Innstillingene som ble satt for testene er vist i Figur 5.25.



Figur 5.25: Innstillinger i testverktøyet DoSHTTP

Ifølge applikasjonsloggen var det bare den ene testgjennomkjøringen som fikk generert et relativt stort antall forespørsler til serveren. Ut fra loggen ble det generert 1828 forespørsler på 4,422 sekunder. Dette tilsvarer litt over 400 forespørsler pr sekund. Selv om dette gikk fint gir ikke testresultatene noe grunnlag for å si at tilgjengeligheten for varslere er sikker. Fremtidige tester med *DoSHTTP* bør teste over en lengre periode for å se at "Sikker Varsling" håndterer et stort antall forespørsler samtidig over lengre tid. Hvor mange forespørsler "Sikker Varsling" skal håndtere, er det ikke satt konkrete krav til. Fremtidige tester bør avklare dette for å ha et spesifikt mål for testingen.

Resultatene fra testingen med *Microsoft Web Application Stress Tool* fikk fremprovosert en feilmelding i varslerapplikasjonen. Feilmeldingen er vist i Figur 5.26, og viser at serveren lukket forbindelsen til nettleseren.



Figur 5.26: Feilmelding etter lasttest

Denne feilmeldingen ble fremprovosert etter at testverktøyet ble kjørt fra to maskiner, hvor gjennomsnittlig antall forespørsler var 374 per sekund over en periode på 5 minutter. Det var ikke nok til at serveren gikk ned, men enkelte forespørsler fikk feilmeldingen som vist over. Dette kan betraktes som en mild variant av et DoS angrep for de brukerne som får denne feilmeldingen. Dette er ikke blitt vurdert som kritisk, siden en oppdatering av websiden ofte var nok til å løse problemet.

Testen avdekket ingen sikkerhetsmekanismer for å oppdage slike angrep. Dette kunne for eksempel ha vært et Intrusion Detection System (IDS). For et fremtidig produksjonsmiljø, anbefales det at slike sikkerhetsmekanismer tas i bruk.

Resultatsammenligning
Mangelen på krav til antall forespørsler gjør denne resultatsammenligningen vanskelig, men på grunn av feilmeldingen i Figur 5.26, ble ikke det forventede resultatet fullstendig oppnådd.

5.3.7. Resultater T-7: Håndtering av feilmeldinger

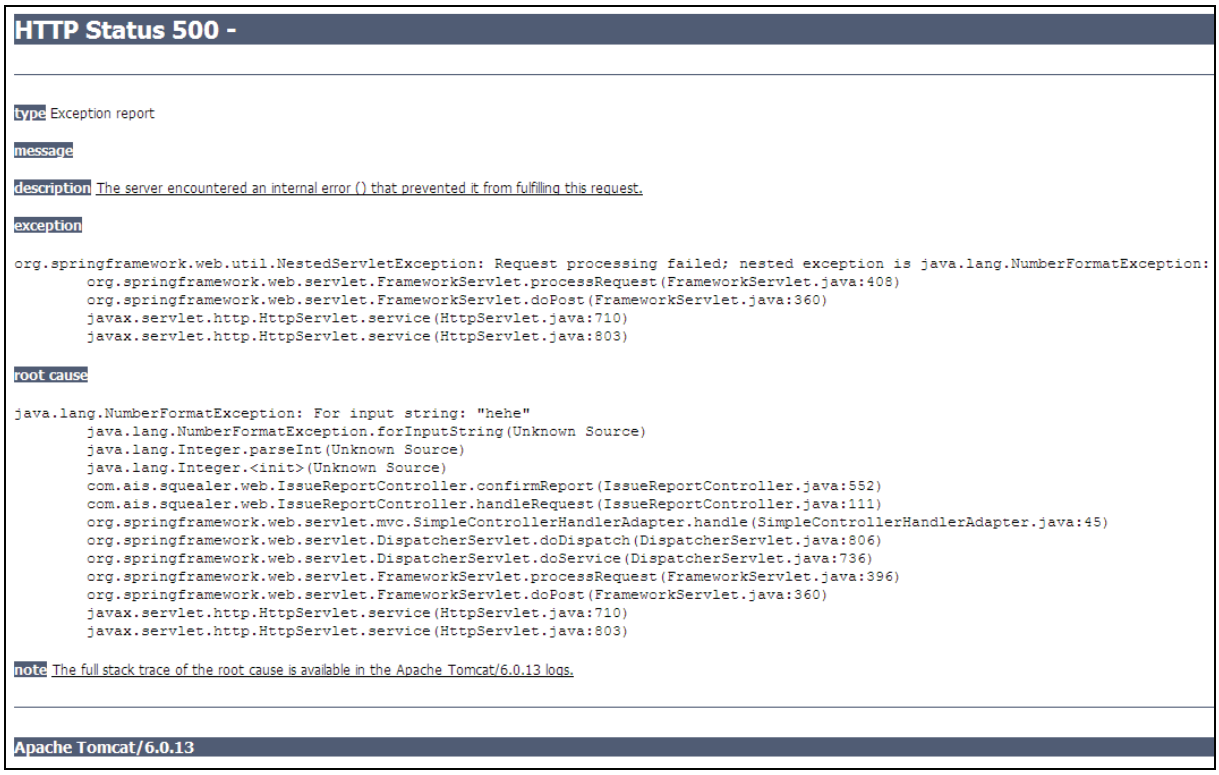
Feilmeldingene i Figur 5.1 og Figur 5.2 er gode eksempler på feilmeldinger som gir en angriper unødvendig mye informasjon. Dette er tidligere diskutert i avsnitt 5.3.1.

Ved feil innlogging for administratorer og saksbehandlere gir ”Sikker Varsling” en sikkerhetsmessig *riktig* feilmelding tilbake til brukeren. Figur 5.27 viser denne feilmeldingen.

Figur 5.27: Feilmelding ved innlogging

Det som er bra med denne feilmeldingen er at en potensiell angriper ikke får informasjon om det var brukernavnet eller passordet som var feil. En mer spesifikk feilmelding vil gi gyldige brukere mer konkret hjelp ved feil i innloggingen, men det vil samtidig gi en angriper mer informasjon om hva for felt som ble korrekt utfylt.

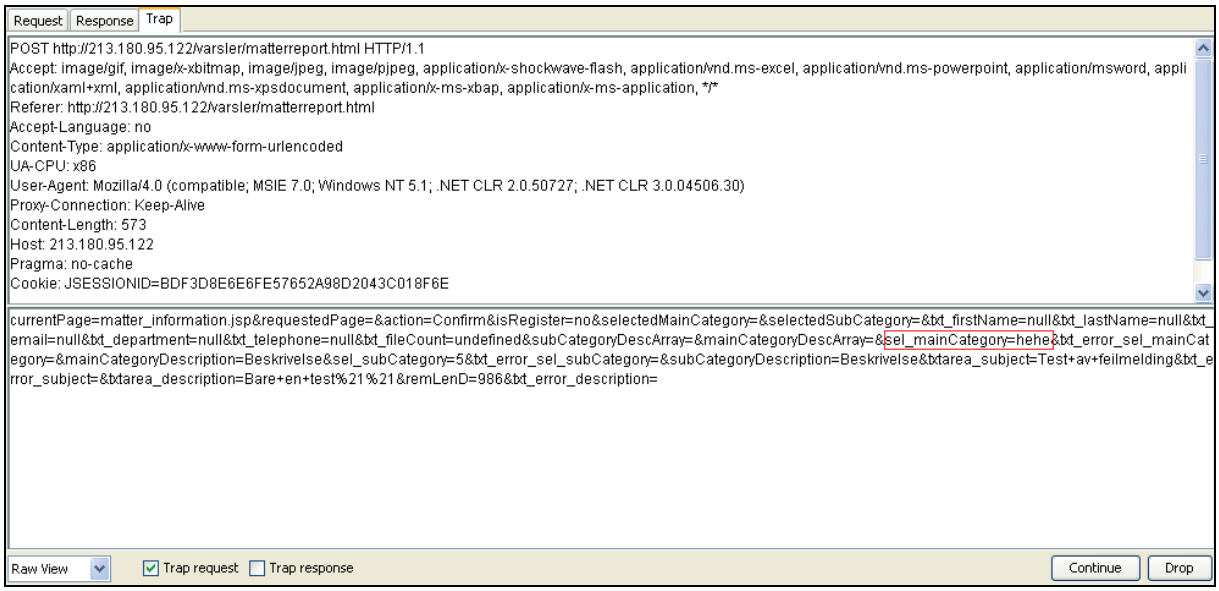
Denne testen avdekket i tillegg en sårbarhet i varslerapplikasjonen. Når en varsler registrerer en sak (se Figur 3.5), velger vedkommende en hovedkategori og en underkategori. Hver av disse valgte kategoriene blir sendt til serveren i form av et tall. Hvis serveren mottar noe annet enn et tall, får varsleren en feilmelding som vist i Figur 5.28.



Figur 5.28: Feilmelding ved registrering av sak

Som figuren viser, gir denne feilmeldingen et lite utdrag fra kildekoden. Dette gir en angriper unødvendig informasjon, og slike situasjoner bør derfor unngås. Løsningen er å validere inputen på serversiden, og gi en generell feilmelding tilbake ved feil input.

For å fremprovosere denne feilmeldingen ble *Paros* brukt. Forespørselen ble forandret slik som Figur 5.29 viser. Den røde firkanten viser parameteren som ble forandret. Opprinnelig var parameteren: *sel_mainCategory=1*. Ved å forandre den til *sel_mainCategory=hehe*, fikk serveren en uventet input som genererte feilmeldingen i Figur 5.28.



Figur 5.29: Fremprovosering av feilmelding ved hjelp av Paros

Resultatsammenligning
De feilmeldingene som ble avdekket under test T-1, og feilmeldingen som ble avdekket i denne testen, gjør at det forventede resultatet ikke har blitt oppnådd.

5.3.8. Resultater T-8: URL håndtering

Resultatene fra denne testen avdekket ingen feil. Alle identifiserte websider ble testet, og ingen av sidene avdekket noen sikkerhetsfeil. Sidene som ble testet i varslerapplikasjonen, og tilhørende kommentarer er gitt i Tabell 5.1:

Webside	Kommentar
http://213.180.95.122/varsler/welcome.jsp	Dette er fremsiden.
http://213.180.95.122/varsler/registration.jsp	Her kommer man til registreringssiden. Dette er OK.
http://213.180.95.122/varsler/matterreport.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/varsler/matterinformationsubmit.html	Her kommer man til startsiden. Dette er OK.
http://213.180.95.122/varsler/find_reported_matter.jsp	Her kommer man til søkesiden for PIN koden. Dette er OK.
http://213.180.95.122/varsler/findreport.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/varsler/squealerfollowup.html	Her kommer man til fremsiden. Dette er OK.

Tabell 5.1: Testede sider i varslerapplikasjonen

Sidene som ble testet i administrasjonsapplikasjonen, og tilhørende kommentarer er gitt i Tabell 5.2:

Webside	Kommentar
http://213.180.95.122/admin/protected/index.jsp	Dette er fremsiden.
http://213.180.95.122/admin/admin.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/manageuser.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/addmaincategory.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/addSubCategory.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/help.html?gotoAction= GotoConfigurations	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/help.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/admin/logout.jsp	Her kommer man til fremsiden. Dette er OK.

Tabell 5.2: Testede sider i administrasjonsapplikasjonen

Sidene som ble testet i saksbehandlerapplikasjonen, og tilhørende kommentarer er gitt i Tabell 5.3:

Webside	**Kommentar**
http://213.180.95.122/sak/protected/index.jsp	Dette er fremsiden.
http://213.180.95.122/sak/activereports.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/statuswiseprocessedreports.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/processreport.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/viewselectedarchivereport.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/deletebogusreports.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/statisticslist.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/statistics_list.jsp	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/registration.jsp	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/iaomatterreport.html	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/change_password.jsp	Her kommer man til fremsiden. Dette er OK.
http://213.180.95.122/sak/help.html	Her kommer man til fremsiden. Dette er OK.

Tabell 5.3: Testede sider i saksbehandlerapplikasjonen

Resultatsammenligning
Ingen URL-er tillot uautorisert tilgang til ”Sikker Varsling”. Dermed har det forventede resultatet blitt oppnådd.

6. Evaluering

Dette kapittelet gir en evaluering av prosjektet. Dette innebærer en evaluering av valg av tester, testgjennomføringen, testresultatene, samt en evaluering av testverktøyene. På bakgrunn av dette, vil dette kapittelet også gi et forslag til testprosedyre for lignende testprosjekter.

6.1. Evaluering av testene

Valget av testene i prosjektet var på bakgrunn av risikostyringen i avsnitt 3.2. Identifikasjon av potensielle trusler og risikoer ble stort sett gjort på egenhånd, hvor egen kunnskap og OWASP topp ti liste[13], sammen med andre litteraturkilder[3][15], ble brukt. For å bedre kvaliteten, ble resultatet av identifikasjonen i tillegg evaluert av prosjektets veiledere. På bakgrunn av dette er oppfatningen at trussel- og risikoidentifikasjonen er tilstrekkelig god til å dekke de mest relevante truslene og risikoene mot "Sikker Varsling".

Prosessen med å rangere risikoene ble i hovedsak gjort på samme måte som for identifikasjonen. Graderingen av sannsynlighet og konsekvens har stort sett blitt gjort på egenhånd, men med innspill fra veilederne. Mangelen på erfaringer fra lignende prosjekter kan ha vært en svakhet ved at rangeringen ikke avbildet det virkelige risikobildet. Faren ligger i at prosjektet ikke prioriterte og testet de mest kritiske risikoene i forhold til "Sikker Varsling". Dette var en risiko for selve utfallet av prosjektet, men på grunn av manglende erfaringer, også en risiko som måtte aksepteres.

Tidsbegrensningene i prosjektet gjorde at kun enkelte av risikoene ble testet. Valget av hvilke risikoer som ble testet er dokumentert i avsnitt 4.1, som også gir en begrunnelse for disse valgene. Hovedtanken var at risikoene med høyest risikonivå skulle prioriteres først. I tillegg ble enkelte risikoer valgt bort på grunn av vanskeligheten med å teste de, eller at lignende risikoer var testet tidligere. Dette ble gjort for å få en mest mulig bred dekning av mulige sikkerhetsrisikoer mot "Sikker Varsling". Samtidig gjorde det at prosjektet fikk prøvd forskjellige typer testverktøy og forskjellige fremgangsmåter for sikkerhetstesting.

Testene dekker sikkerhetskritiske funksjoner og angrep mot både varslerapplikasjonen, saksbehandlerapplikasjonen, og administrasjonsapplikasjonen. Konfidensialitet, integritet og tilgjengelighet har stått sentralt for alle disse tre applikasjonene. I tillegg har mange av de sentrale angrepene i topp ti listen til OWASP[13] blitt testet. På bakgrunn av dette har valg av tester blitt vurdert som god.

6.2. Evaluering av testgjennomføringen

Testgjennomkjøringen har stort sett fulgt fremgangsmåten som er dokumentert i avsnitt 4.2, som omhandler omfang, testmetode, testverktøy og forventet resultat for hver av testene.

[13] Hjemmeside: http://www.owasp.org/index.php/Top_10_2007

Erfaringen fra prosjektet er at en slik testbeskrivelse gjør selve testgjennomføringen enklere. Beskrivelsen gav en strukturert fremgangsmåte for hvordan hver enkelt test skulle utføres og hvilket omfang de hadde. Dette var spesielt nyttig i starten av hver test, siden beskrivelsene gav klare retningslinjer for hvordan det skulle testes. Samtidig var det noen tester som, avhengig av identifiserte feil under testingen, ikke fulgte testbeskrivelsen underveis. Dette ble gjort for å kunne utforske identifiserte feil, og teste for lignende feil i samme deler av systemet. Bakgrunnen for dette var at feil har en tendens til å samle seg. Dette kan for eksempel være innenfor samme komponent av et system.

Tidsplanen har delvis blitt fulgt. Som fremdriftsplanen i Appendiks C viser, var det 3,5 uker testmiljøet ikke var tilgjengelig for testing. Dette medførte litt forsinkelser for testene T-1, T-7 og T-8[14]. Dette førte til at T-7 og T-8 ble testet samme uke, noe som ytterligere begrenset omfanget av disse to testene. Testingen av T-1 ble spredt utover flere uker (vises ikke i fremdriftsplanen), noe som samlet resulterte i et akseptabelt testomfang. Den begrensede testingen av T-7 og T-8 kan forsvares med at de ikke var så sentrale og krevende som de andre testene. Den største svakheten i testgjennomføringen har blitt vurdert til å være den automatiske testingen av T-2 og T-3. Tidsbruken for å lære seg testverktøyene og utnytte de best mulig, er hovedgrunnen til dette. Bortsett fra dette, ble alle de planlagte testene utført og samlet sett blir testgjennomføringen sett på som tilstrekkelig god.

6.3. Evaluering av testresultater

Det er vanskelig å fastslå hvor dekkende de identifiserte sikkerhetsfeilene er i forhold til sikkerhetssituasjonen til ”Sikker Varsling”. Med tanke på at det er praktisk umulig å eliminere alle sikkerhetsfeil i et komplekst system, vil det mest sannsynlig være en del gjenværende sikkerhetsfeil i ”Sikker Varsling”. Det som også taler for dette argumentet, er tidsbegrensingen som hver test i dette prosjektet har hatt. Tidsbruken i hver test har ikke bare gått med til effektiv testing, men også til opplæring rundt kildekoden, testverktøy og applikasjonen generelt. Derfor vil det være vanskelig å påstå at hver test har vært omfattende nok til å avdekke alle sikkerhetsfeil i sin kategori.

Samtidig har testene avdekket en del sikkerhetsfeil hvor alvorlighetsgraden på hver enkelt er varierende. Alvorlighetsgraden har blitt delt inn i tre forskjellige kategorier:

- **Stor** – Disse feilene er kritiske for sikkerheten til ”Sikker Varsling”. Det anbefales at disse feilene blir rettet før systemet blir satt i produksjon.
- **Medium** – Disse feilene svekker sikkerheten til ”Sikker Varsling”, men er ikke kritiske for en eventuell produksjonsdato. Avhengig av hvor mye arbeid som kreves for å rette disse feilene, anbefales det at alle blir vurdert rettet.
- **Liten** – Disse feilene har mindre betydning for sikkerheten til ”Sikker Varsling”. Bare hvis feilene er enkle å rette, bør dette gjennomføres.

Tabell 6.1 gir en oversikt over de identifiserte sikkerhetsfeilene i ”Sikker Varsling”, og alvorlighetsgraden til hver enkelt feil. Tabellen viser også i hvilken test disse feilene ble identifisert.

[14] Disse identifikatorene refererer til kolonnen *Test ID i* Tabell 4.1.

Sikkerhetsfeil	Test ID[14]	Alvorlighetsgrad
Mulighet for Cross Site Scripting i kategoriadministrasjon for administratorer	T-2	Stor
Mulighet for SQL injeksjon i endre passord funksjonen for saksbehandlere.	T-3	Stor
Generering av sesjonsidentifikator, hvor identifikatoren ikke forandres ved innlogging	T-4	Stor
Svake krav til brukernavn og passord for saksbehandlere og administratorer	T-1	Medium
Mulighet for å prøve innlogging et ubegrenset antall ganger med PIN kode eller brukernavn og passord	T-1	Medium
Fremgangsmåte for å unngå Cross Site Scripting og SQL injeksjon, hvor "blacklisting" er brukt	T-2	Medium
Ingen krav til nettleserversjon	T-5	Medium
Spesifikke feilmeldinger rundt PIN koden som gir for mye informasjon	T-1	Liten
Denial of Service angrep skaper enkelte feilmeldinger fra server	T-6	Liten
Fremprovosering av detaljert feilmelding ved forandring av input	T-7	Liten

Tabell 6.1: Gradering av identifiserte sikkerhetsfeil

Som tabellen viser, er det bare T-8 som ikke har funnet sikkerhetsfeil.

6.4. Evaluering av testverktøyene

Valg av testverktøy ble stort sett gjort ut fra egne vurderinger. Som nevnt i avsnitt 2.2.5, ble listen med testverktøy fra *Insecure.Org* brukt som grunnlag for valgene som ble tatt. I tillegg ble det søkt en del på Internett for å finne andre testverktøy som kunne brukes. Uten tidligere erfaringer med slike testverktøy, ble det litt tilfeldig hvilke verktøy som ble valgt. Resultatet av dette ble at enkelte testverktøy ikke fungerte som forventet. Dette var en risiko som hele tiden har vært kjent, og som måtte aksepteres som en del av prosjektet.

Mye av tiden i dette prosjektet gikk med til å lære seg å bruke testverktøyene. Som Appendiks A viser, omfatter prosjektet sju forskjellige testverktøy. I noen av testene har den manuelle testingen funnet sårbarheter som ikke har blitt avdekket av testverktøyene. Dette gjelder spesielt for sårbarhetene som omhandler Cross Site Scripting (XSS) og SQL injeksjoner. På bakgrunn av dette må det stilles spørsmål rundt disse testverktøyene. Enten er ikke testverktøyene grundige nok i søkene sine, eller så har de ikke blitt godt nok utnyttet.

Samtidig var det noen av testverkøyene som ble godt utnyttet og som var til god hjelp i sikkerhetstestingen. *Cain & Abel* og *Paros* var begge gode testverktøy som ble utnyttet godt. *Windows Web Application Stress Tool* var også et godt verktøy, men passet ikke helt til Denial of Service (DoS) testen. Til DoS testen mener prosjektet at en fungerende versjon av *DoSHTTP* kan være et godt testverktøy. Med tanke på risikoen som lå i valg av testverktøy, ble resultatet omtrent som forventet; noen av testverktøyene fungerte godt, mens andre fungerte dårlig.

6.5. Forslag til testprosedyre

I avsnitt 1.2 står det at dette prosjektet har som mål å resultere i en testprosedyre for systemet ”Sikker Varsling” og lignende systemer. Grunnlaget for å utvikle en slik prosedyre er gjort og dokumentert i de tidligere kapitlene i rapporten.

Som Figur 2.4 viser, bør det tenkes sikkerhet tidlig i utviklingsprosessen. Dette prosjektet har ikke hatt anledning til å være med på et utviklingsløp, og har derfor ikke fått erfaringer med sikkerhetsmetoder som kan brukes tidlig i utviklingsprosessen. Unntaket er risikostyring, som anbefales å være en kontinuerlig prosess gjennom hele utviklingsprosessen, samt gjennom drifts- og vedlikeholdsfasen.

På bakgrunn av teoretisk kunnskap, anbefales det at fremtidige utviklingsprosesser vurderer følgende sikkerhetstiltak:

Sikkerhetskrav
Kravspesifikasjonen bør inneholde spesifikke sikkerhetskrav som fastsetter hvilke krav man har til sikkerhet. Dette er en viktig metode som får prosjektteamet til å tenke sikkerhet så tidlig som mulig.

Risikostyring
Erfaringene fra dette prosjektet tilsier at risikostyring er helt sentralt for å kunne fokusere og teste de mest kritiske sikkerhetsrisikoene i et system.

Designgjennomgang
Som nevnt i [2], kommer rundt halvparten av alle sårbarhetene fra systemdesignet. Designgjennomgang er derfor en sentral metode for å unngå at sårbarheter oppstår.

Kodegjennomgang
Statiske metoder ved hjelp av testverktøy kan automatisere prosessen med å finne sårbarheter i koden. Testverktøy kan dermed effektivisere kodegjennomgangen.

Penetrasjonstesting
Hovedfokuset i dette prosjektet har vært forskjellige metoder for penetrasjonstesting av ”Sikker Varsling”. På bakgrunn av erfaringene i prosjektet bør fremtidig sikkerhetstesting av ”Sikker Varsling” og lignende systemer vurdere en testprosedyre som gitt i avsnitt 6.5.1.

6.5.1. Testprosedyre for penetrasjonstesting

Dette avsnittet vil gi et forslag til testprosedyre for penetrasjonstesting av webapplikasjoner. Denne testprosedyren er basert på de erfaringene og resultatene som dette prosjektet har gitt. Testprosedyren består av risikoer som bør testes og som er vurdert til å være kritiske i webapplikasjoner som ”Sikker Varsling”.

Cross Site Scripting (XSS) og Injeksjonsfeil
Dette er to av de største risikoene mot webapplikasjoner. En angriper kan gjøre mye skade på webapplikasjonen hvis ikke inputfeltene blir validert på en korrekt måte. Som nevnt tidligere, er ”whitelisting” den korrekte fremgangsmåten for å få dette til. Et annet viktig poeng er at

valideringen må skje på serversiden. Validering kun på klientsiden er en veldig svak fremgangsmåte for å beskytte seg mot slike risikoer.

Krav til autentisering
For webapplikasjoner som krever autentisering, er det viktig at det stilles krav som gir en sikker løsning. Dette kan for eksempel være krav til brukernavn, passord, PIN kode, sesjonsidentifikasjon eller lignende. Vellykkede angrep mot autentiseringen kan føre til at angripere får full tilgang til applikasjonen.

Tilgjengelighet
En webapplikasjon er ikke nyttig for brukerne hvis den ikke er tilgjengelig. Angrep på tilgjengeligheten bør derfor gjennomføres for å teste om applikasjonen motstår unormal stor last. Dette kan for eksempel være at applikasjonen motstår et Denial of Service (DoS) angrep. Bruk av Intrusion Detection Systems (IDS) eller brannmur for å gjenkjenne slike eller lignende angrep kan være til stor nytte for en webapplikasjon.

Håndtering av feilmeldinger
Å gi brukerne de rette feilmeldingene er en balansegang mellom funksjonalitet og sikkerhet. Feilmeldinger i en webapplikasjon bør ikke gi konkret informasjon som hjelper en angriper. Samtidig bør feilmeldingene være informative, slik at gyldige brukere har nytte av dem. Denne balansegangen er viktig å tenke over når man tester feilmeldinger i en webapplikasjon.

Kryptering av kommunikasjon og lagring
Kryptering av informasjon er viktig for å unngå at angripere kan lese informasjonen i klartekst. Dette er ikke blitt testet i dette prosjektet siden all informasjon skal krypteres i produksjonsmiljøet. Uansett er det viktig å tenke på hvilke krypteringsalgoritmer som brukes når man tester dette. Svake algoritmer gjør det lettere for angripere å dekryptere informasjonen. Siden dette ikke ble testet, har en analyse av sterke og svake krypteringsalgoritmer ligget utenfor omfanget av dette prosjektet.

Uansett applikasjon, er det svært viktig å foreta en risikostyring av webapplikasjonen. En slik risikostyring vil mest sannsynlig avdekke andre risikoer enn de som er nevnt over. Dette vil påvirke hvilke tester som bør utføres. Hovedformålet med testprosedyren som er gitt ovenfor er å gi veiledning til lignende testprosjekter og forhåpentligvis bidra til å avdekke de mest kritiske sårbarhetene.

7. Konklusjon

Oppgavemål
Hovedformålet med dette prosjektet var å gjennomføre en sikkerhetstest av ”Sikker Varsling”. Sikkerhetstesten hadde som formål å prøve forskjellige tester og testverktøy. Dette skulle legge grunnlaget for en fremtidig testprosedyre for ”Sikker Varsling” og lignende websystemer.

Testresultat
Resultatet av dette prosjektet har avdekket en del sikkerhetsfeil i ”Sikker Varsling”. Alvorlighetsgraden for hver enkelt feil har vært varierende, slik Tabell 6.1 viser. Enkelte feil har blitt vurdert til å være såpass alvorlige at de er kritiske for omdømmet til systemet. Vellykkede angrep som utnytter enkelte av disse feilene, kan i verste fall føre til at konfidensialiteten til varslere blir avslørt. Hvis dette skulle skje, mister hele systemet hovedfunksjonaliteten sin; at ansatte i en organisasjon kan varsle et forhold anonymt. Sikring av denne anonymiteten er derfor helt avgjørende for ”Sikker Varsling”.

Mange av resultatene fra sikkerhetstestingen har hatt en felles sikkerhetsfeil. Valideringen av enkelte input har ikke skjedd på serversiden, og ”Sikker Varsling” har dermed vært sårbar for en del kritiske angrep. Dette viser igjen et omfattende aspekt ved programvaresikkerhet; alle inputfelt må valideres. En angriper trenger bare ett sårbart inputfelt for å angripe et system. I tillegg har prosjektet avdekket en svakhet i selve valideringen. De inputfeltene som valideres bruker ”blacklisting” av tegn istedenfor ”whitelisting” av tegn. Dette er diskutert i avsnitt 5.3.2.

Erfaringer
Prosjektet har vist at programvaresikkerhet er en omfattende oppgave som krever at man inkluderer sikkerhet tidlig i utviklingsprosessen. Det anbefales at fremtidige utviklingsprosesser vurderer de forskjellige sikkerhetsmetodene nevnt i avsnitt 6.5. Disse er viktige metoder for å bidra til en tilfredsstillende sikkerhet i webapplikasjoner.

Prosjektet har også vist at sikkerhetstesting er en viktig og omfattende oppgave. Siden ”Sikker Varsling” var ferdigutviklet når prosjektet startet, har penetrasjonstesting av systemet vært hovedfokuset gjennom prosjektet. Penetrasjonstestingen har vært risikobasert, ved at risikostyringen i avsnitt 3.2 har påvirket hvilke tester som ble prioritert. Basert på erfaringene fra dette prosjektet, gis det en sterk anbefaling om å gjennomføre en risikostyring for å fokusere arbeidet på de mest kritiske risikoene. I tillegg gir en slik risikostyring en god oversikt over sikkerhetssituasjonen i et system.

Erfaringene og resultatene fra dette prosjektet gav grunnlaget for forslaget til testprosedyre, gitt i avsnitt 6.5.1. Denne testprosedyren gir en oversikt over risikoer som prosjektet mener er viktige å teste i websystemer som ”Sikker Varsling”.

Testverktøyene som er brukt i dette prosjektet, har vist at prosessen med å finne de rette testverktøyene er vanskelig og tidkrevende. Det er vanskelig på grunn av mangfoldet av testverktøy. Det store utvalget gir en utfordring når man skal velge hvilke verktøy man vil utforske. I tillegg er dette tidkrevende siden hvert testverktøy krever at man lærer seg å bruke, og utnytte det på en god måte. Som prosjektet har vist, er det enkelte testverktøy som er vanskelige å bruke og som ikke gir de forventede resultatene. Samtidig er gevinsten med å

finne de rette verktøyene såpass stor at prosessen kan være verdt det. Gode testverktøy resulterer i mer effektiv testing, noe som igjen utvider omfanget av testingen.

Videre arbeid
For ”Sikker Varsling” anbefales det å gjennomføre en ny risikostyring for å revurdere hvilke risikoer som er kritiske. Dette bør gjøres etter retting av sikkerhetsfeilene som har blitt identifisert i dette prosjektet. I tillegg anbefales det at videre arbeid finner gode testverktøy som effektiviserer testingen av Cross Site Scripting og SQL injeksjoner for websystemer som ”Sikker Varsling”. Som nevnt i avsnitt 6.4, er gjennomføringen av disse automatiske testene noe svak.

For andre systemer, anbefales det at sikkerhet inkluderes tidlig i utviklingsprosessen. Et slikt arbeid kan evaluere og forbedre forslagene til sikkerhetstiltak som er gitt i avsnitt 6.5. Dette arbeidet vil i tillegg også kunne forbedre testprosedyren for penetrasjonstesting som er gitt i avsnitt 6.5.1.

8. Bibliografi

[1] Barry Boehm og Vic Basili. *Software Defect Reduction Top 10 List, IEEE Computer.* 2001.

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

[3] Mike Andrews og James A. Whittaker. *How to Break Web Software.* Addison-Wesley, 2006.

[4] Dataforeningen, Terminologi for test av programvare. *http://dataforeningen.no/filestore/Terminologifortestavprogramvare.pdf*

[5] Rebecca Bace og Peter Mell. *Intrusion Detection Systems*. NIST Special Publication on Intrusion Detection System.

[6] Norges offentlige utredninger (NOU) 2000:24. *Et sårbart samfunn – Utfordringer for sikkerhets- og beredskapsarbeidet i samfunnet.* 2004.

[7] Colin Boyd. *Information Security Fundamentals, Lecture notes.* Queensland University of Technology, Australia, 2007.

[8] National Institute of Standards and Technology (NIST). *SP 800-30: Risk Management Guide for Information Technology Systems.* 2002.

[9] Andreas Spillner, Tilo Linz og Hans Schaefer. *Software Testing Foundations.* dpunkt.verlag, 2006.

[10] Andreas Spillner, Thomas Rossner, Mario Winter og Tilo Linz. *Software Testing Practice: Test Management.* Rocky Nook, 2007.

[11] Ron Patton. *Software Testing.* Sams Publishing, 2001.

[12] CERT Coordination Center (CERT/CC). *http://www.cert.org/stats/vulnerability_remediation.html#vul-notes-published.* 2008.

[13] Open Web Application Security Project (OWASP). *OWASP Testing Guide V2.* 2007.

[14] Institute for Security and Open Methodologies (ISECOM). *Open-Source Security Testing Methodology Manual 2.1.1.* 2005.

[15] Sverre H. Huseby. *Innocent Code – a security wake-up call for web programmers.* John Wiley & Sons, 2004.

[16] ISO 17799. *Information Technology – Code of Practice for Information Security Management.* 2000.

[17] Bruce Schneier. *Cryptanalysis of MD5 and SHA: Time for a New Standard.* Computerworld. 2004.

[18] Common Criteria. *Common Methodology for Information Technology Security Evaluation, Version 3.1, Revision 2 (CEM v3.1).* 2007.

[19] U.S. Department of Helth and Human Services. *https://grants.hrsa.gov/webExternal/help/hlpPage.asp?hF=help_password*.

[20] D. Kristol og L. Montulli. *RFC 2109 - HTTP State Management Mechanism.* 1997.

[21] Mitja Kolšek. *Session Fixation Vulnerability in Web-based Applications.* ACROS Security, 2002.

Appendiks A Testverktøy

Dette appendikset gir en oversikt over de forskjellige testverktøyene som har blitt brukt i dette prosjektet. Hvert testverktøy er kort beskrevet med tanke på formål og funksjonalitet. I tillegg inkluderer appendikset egne erfaringer fra bruken av programmene, samt referanser til mer informasjon og nedlastningssider. Hvert testverktøy er beskrevet i et eget avsnitt under.

A.1 Cain & Abel

Beskrivelse
Cain & Abel baserer seg kun på Windows. Programmet er laget for å blant annet knekke krypterte passord ved hjelp av forskjellige angrepsmetoder. Det kan også brukes til ”Man-In-The-Middle” angrep, slik at en angriper kan lese kryptert informasjon. Cain & Abel inneholder i tillegg en omfattende brukermanual, som forklarer funksjonene i programmet.

Erfaringer
Erfaringene med Cain & Abel er positive. Brukermanualen gjør det enkelt å forstå de mange funksjonene som ligger i programmet. Bruken av programmet, har i hovedsak vært å avsløre hashede passord. Siden testmiljøet til ”Sikker Varsling” ikke brukte https, ble det ikke gjennomført et ”Man-In-The-Middle” angrep på applikasjonen. Et slikt angrep ble derimot prøvd ut på en annen applikasjon, hvor Cain & Abel genererer et falskt sertifikat som, hvis offeret aksepterer, gjør at programmet kan dekryptere all kommunikasjon mellom offeret og webserveren. Dette kan derimot bare gjøres på andre maskiner i samme nettverk som angriperen.

Referanser
Hjemmeside: http://www.oxid.it/cain.html

A.2 Paros

Beskrivelse
Dette testverktøyet kan brukes til skanne websider for sårbarheter, og forandre http-forespørsler og svar. Programmet er skrevet i Java, og har et grafisk brukergrensesnitt. Verktøyet er ikke oppdatert siden 2004.

Erfaringer
På grunn av at programmet ikke har blitt oppdatert siden 2004, har det bare blitt brukt til å forandre http-forespørsler og svar. Grensesnittet er brukervennlig, og gjør programmet enkelt å bruke. Dette har ført til at Paros har blitt brukt mye i løpet av prosjektet for å se på og forandre http-forespørsler og svar.

Referanser
http://www.parosproxy.org/functions.shtml

A.3 Acunetix WVS

Beskrivelse
Acunetix WVS er et testverktøy som skanner websider for kjente sårbarheter som SQL injeksjoner og Cross site scripting (XSS). Verktøyet har også annen funksjonalitet som er relevant i en sikkerhetsanalyse av websider, som for eksempel å crawle websider og dokumentering av resultatene. Acunetix er et kommersielt testverktøy som er godt dokumentert.

Erfaringer
Prøveversjonen av dette verktøyet kan bare brukes til å teste for XSS. Dermed ble dette verktøyet brukt i test T-2, som er beskrevet i avsnitt 4.2.2. På bakgrunn av resultatene i avsnitt 5.3.2, hvor verktøyet ikke avdekket noen sårbarheter, er det blitt konkludert med at verktøyet ikke er grundig nok i søket etter XSS sårbarheter, eller at det ikke ble godt nok utnyttet. Det mest sannsynlige er en dårlig utnyttelse, og dette skyldes manglende tid til å utforske testverktøyet godt nok.

Referanser
Hjemmeside: http://www.acunetix.com/vulnerability-scanner/
Brukermanual: http://www.acunetix.com/vulnerability-scanner/wvs5manual/websecurity-scanner.htm

A.4 OWASP SQLiX v1.0

Beskrivelse
SQLiX skanner websider for SQL injeksjoner. Testverktøyet bruker to forskjellige metoder i testingen. Den ene er å generere feil ved hjelp av metategn som enkle og doble anførselstegn. Metoden går ut på å legge til disse metategnene i SQL-forespørsler. Dette kan føre til syntaksfeil, hvor SQL-feilmeldinger blir vist i http-svaret. Den andre metoden er injeksjon av uttrykk, hvor programmet legger logiske uttrykk til i SQL-forespørselen. Ut fra http-svarene kan en fastslå om websiden er sårbar for SQL injeksjoner. Et kort eksempel på denne metoden er gitt på hjemmesiden til testverktøyet.

Erfaringer
Bruken av dette testverktøyet gav ingen gode erfaringer. Dette skyldes i hovedsak at programmet gav den samme outputen for hver test. Outputen viste feilmeldinger og fant ingen resultat som vist i Figur 5.11. På grunn av manglende brukermanual, hvor det ikke er noen eksempler på hvordan man håndterer postdata, tas det høyde for at verktøyet ikke er blitt utnyttet på best mulig måte.

Referanser
Hjemmeside: http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project

A.5 SQLBrute

Beskrivelse
Dette testverktøyet baserer seg på blind SQL injeksjon, og bruker en ”brute force” algoritme for å hente ut data fra databaser.

Erfaringer
SQLBrute er basert på en god ide, men som Figur 5.12 på side 47 viser, gir det ikke et fornuftig resultat tilbake. På grunn av en mangelfull brukermanual, tas det høyde for at verktøyet ikke er utnyttet godt nok.

Referanser
Hjemmeside: http://www.justinclarke.com/archives/2006/03/sqlbrute.html
Blind SQL injeksjon: http://www.imperva.com/resources/adc/blind_sql_server_injection.html

A.6 DoSHTTP

Beskrivelse
Dette testverktøyet oppretter flere asynkrone forbindelser som alle prøver å overfylle en gitt webserver med http forespørsler. Verktøyet overvåker ytelsen og genererer en rapport etter at angrepet er utført. Verktøyet kan brukes fritt i 14 dager, men for ytterligere bruk må det betales for en lisens til programmet.

Erfaringer
DoSHTTP har et meget enkelt brukergrensesnitt som gjør det enkelt å utføre et *Denial of Service* angrep. Uavhengig av dette, inneholdt prøveversjonen en feil, slik at programmet ikke kunne brukes etter en testgjennomkjøring. Dette begrenset erfaringene med testverktøyet.

Referanser
Hjemmeside: http://www.socketsoft.net/products.asp?p=doshttp

A.7 Microsoft Web Application Stress Tool

Beskrivelse
Dette testverktøyet er ment for å lastteste webapplikasjoner. Det vil si at verktøyet prøver å simulere et reelt bruksmønster. Dette bruksmønsteret kan for eksempel være testing av normal last over en lengre periode, eller en stresstest som simulerer en unormal stor last over en kortere periode. Verktøyet har også en rapporteringsfunksjon og en omfattende brukermanual.

Erfaringer
Testverktøyet har et grafisk brukergrensesnitt som gjør dette verktøyet enkelt å bruke. I tillegg gir brukermanualen en god forklaring på funksjonene i verktøyet. På tross av at det er enkelt å bruke, passet ikke verktøyet helt til formålet. Målet var å gjennomføre et Denial of Service (DoS) angrep på varslerapplikasjonen, men verktøyet klarte ikke å generere nok forespørsler

samtidig. Selv om verktøyet ble kjørt på to maskiner samtidig, ble det bare fremprovosert enkelte feilmeldinger. For enkle lasttester derimot, vil dette verktøyet passe bra.

Referanser

Hjemmeside: http://www.microsoft.com/downloads/details.aspx?FamilyID=e2c0585a-062a-439e-a67d-75a89aa36495&displaylang=en

Appendiks B Risikostyring

Dette appendikset gir en del tilleggsinformasjon til avsnitt 2.3 om risikostyring. Det gir eksempler på trusselkilder, diskuterer forskjellen mellom kvalitativ og kvantitativ risikosetting og definerer en risikomatrise og dets begreper.

B.1 Trusselkilder

Ifølge guiden om risikostyring fra NIST[8], er det tre hovedtyper trusselkilder. Den første typen er naturtrusler som innebærer naturkatastrofer som kan ødelegge eller skade et system. Eksempler på slike trusselkilder er jordskjelv, oversvømmelse, stormer, etc. Slike trusselkilder er vanskelige å unngå, men det er mulig å delvis beskytte seg mot enkelte angrep. I områder med for eksempel stor fare for oversvømmelse, bør en risikostyring inkludere dette, og foreslå sikkerhetsmekanismer for å redusere risikoen.

Den andre hovedtypen er menneskelige trusselkilder. Slike kilder kan skape trusler mot systemet med vilje eller utilsiktet. Når det skjer med vilje er det intensjonen å utnytte en eller flere sårbarheter i systemet, mens en utilsiktet handling kan være at en autorisert bruker er uaktsom og dermed utgjør en trussel mot systemet.

Den siste typen er miljøbestemte trusselkilder. Dette er kilder som forårsaker trusler i omgivelsene til systemet. Ifølge guiden kan eksempel på dette være langvarige strømavbrudd, vannlekkasjer, forurensning, etc.

B.2 Kvalitativ vs. kvantitativ

Det finnes to forskjellige måter å definere sannsynlighet, konsekvens og risiko på. Den ene er en kvalitativ definisjon, hvor man bruker beskrivende ord for å definere forskjellige nivåer. En mye brukt kvalitativ skala er tredelt og består av Høy, Middels og Lav. I tillegg til slike beskrivende ord, er det viktig at hver av disse blir definert mer konkret. Dette gjør det lettere å forstå hva som ligger i hvert begrep.

Den andre måten man kan bruke er en kvantitativ tilnærming. I dette ligger det at man tilegner numeriske verdier til både sannsynlighet, konsekvens og risiko. Den store fordelen med en slik metode er at den gir konkrete tall på hvor stor konsekvens en risiko vil ha. Dette kan for eksempel være i antall kroner man taper hvis en risiko inntreffer. Ulempen med en slik metode, sammenlignet med en kvantitativ metode, er vanskeligheten med å sette korrekte verdier. For å klare dette trengs det en god del erfaring fra tidligere lignende systemer.

Det er også mulig å bruke en kombinasjon av kvalitativ og kvantitativ definering. En slik hybrid fremgangsmåte kan for eksempel tilegne verdier for hvert av de kvalitative nivåene. Utfordringen her er, i likhet med en ren kvantitativ fremgangsmåte, å sette korrekte verdier til hvert kvalitativt nivå. Fordelen med en slik kombinasjon kan være at det er lettere å sette verdier på noen få nivåer, enn å sette verdier på alle risikoene i et system.

B.3 Risikomatrise

I arbeidet med å definere en risikomatrise, er det i først og fremst størrelsen man må bestemme seg for. Det er ingen begrensning på hvor stor en slik matrise kan være, men i de fleste tilfellene er størrelsen 3 x 3, 4 x 4 eller 5 x 5. I guiden fra NIST[8], er eksempelmatrisen 3 x 3. Her blir risikonivåene delt inn i tre forskjellige verdier, noe som kan gjøre det vanskelig å skille risikoer fra hverandre. To risikoer klassifisert som *Medium* kan ha ganske forskjellige utfall, men på grunn av at skalaen bare er tredelt, er det vanskelig å se denne forskjellen.

Risikomatrisen i dette eksempelet er 4 x 4, og definisjonene i denne matrisen er tatt fra Colin Boyd[7]. Matrisen er vist i Tabell B.1, hvor radene representerer sannsynligheten og kolonnene representerer konsekvensene.

	Ubetydelig	**Minimal**	**Moderat**	**Stor**
Høy	Medium	Høy	Veldig høy	Veldig høy
Medium	Medium	Medium	Høy	Veldig høy
Lav	Lav	Medium	Medium	Høy
Usannsynlig	Lav	Lav	Medium	Medium

Tabell B.1: Eksempel på en 4 x 4 risikomatrise

For å gjøre tabellen mer forståelig, vil de neste tre tabellene forklare begrepene som er brukt i risikomatrisen. Dette er kun et eksempel, og forslagene til definisjoner er utarbeidet med bakgrunn fra flere forskjellige litteraturkilder[2][7][8].

Sannsynlighetsbegrepene:

Høy	Er forventet å skje under de fleste forhold (en eller flere ganger pr. år)
Medium	Hendelsen vil sannsynligvis skje under de fleste forhold (en gang hvert andre år)
Lav	Hendelsen vil skje etter en viss tid (en gang hvert femte år)
Usannsynlig	Hendelsen kan oppstå etter lang tid (en gang hvert tiende år)

Tabell B.2: Definisjon av sannsynlighetsbegrepene

Konsekvensbegrepene:

Høy	Vil kunne føre til betydelige finansielle tap, stor skade på organisasjonen eller alvorlige menneskelige skader eller død
Moderat	Vil føre til finansielle tap, skade på organisasjonen eller menneskelige skader
Minimal	Kan føre til små finansielle tap eller negativ omtale av organisasjonen.
Ubetydelig	Kan føre til veldig små finansielle tap

Tabell B.3: Definisjon av konsekvensbegrepene

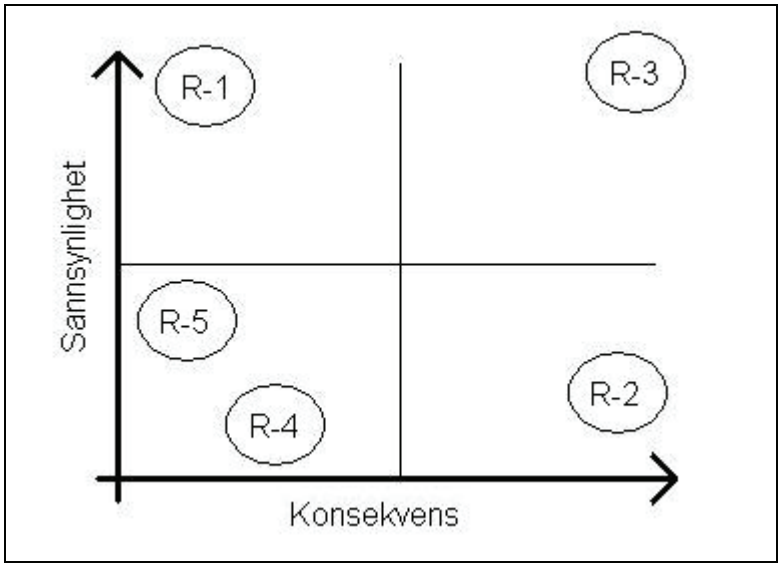
Risikobegrepene:

Veldig høy	Risikokontrollerende tiltak bør innføres umiddelbart
Høy	Systemet kan fortsatt operere, men det bør settes opp en plan for å redusere risikoen som bør iverksettes så fort som mulig
Medium	Det bør settes opp en handlingsplan som tar sikte på å redusere risikoen. Planen bør gjennomføres innen rimelig tid
Lav	Systemansvarlige og andre ledere bør drøfte risikoen for å analysere om tiltak er nødvendige.

Tabell B.4: Definisjon av risikobegrepene

B.4 Risikograf

En annen måte å illustrere risikoer på er ved hjelp av en risikograf. En slik graf har to akser, der X-aksen representerer konsekvensen og Y-aksen representerer sannsynligheten. Et eksempel på en slik graf er gitt i Figur B.1.



Figur B.1: Risikograf

Som figuren viser, plotter man inn de forskjellige risikoene basert på hvor stor konsekvens og sannsynlighet de har. I dette eksempelet er risiko R-3 mest kritisk, og risikoer i denne

kvadranten har risikonivå høy. Risikoer i kvadrantene der R-1 og R-2 er, blir betraktet som medium, og kvadranten hvor R-4 og R-5 er, inneholder risikoer som blir ansett som lave. En slik graf er kun en annen måte å representere risikovurderingen på. Også her må man definere hva som ligger i begrepene lav, medium og høy, samtidig som man må bestemme konsekvens og sannsynlighet for hver enkelt risiko.

Appendiks C Fremdriftsplan

Dette appendikset viser fremdriftsplanen som ble brukt i dette prosjektet. Planen er delt inn i uker, hvor det for hver uke ble satt opp en plan for arbeidsoppgavene. Arbeidsoppgavene inkluderer planlagt arbeid mot ”Sikker Varsling” og planlagt arbeid for rapporten. Planen har også en kolonne som beskriver hvilket arbeid som ble gjort, og en kolonne for eventuelle notater. Fremdriftsplanen er vist i Tabell C.1.

Uker	Plan for arbeidsoppgaver	Rapportplan	Arbeid gjort	Notater
Uke 2	Utarbeiding av oppgavetekst		Utarbeidet oppgaveteksten	
Uke 3	Lese til teoridel, prøve ut SV	Skrive innledningskap.	Lest teori, skrevet utkast til innledningskap.	
Uke 4	Lese til teoridel, prøve ut SV	Begynne på teorikap.	Lest teori, startet på teorikap.	
Uke 5	Lese til teoridel	Teorikap.	Lest teori, skrevet teorikap.	
Uke 6	Risikostyring - lære systemet	Teorikap.	Tilgang til SV, skrevet teorikap.	
Uke 7	Risikostyring - finne risikoer	Begynne på prosjektutdypingskap.	Sett på SV, skrevet utkast til risikoer	
Uke 8	Analysere risikostyringen - rangering	Prosjektutdypingskap.	Rangerte risikoer, startet på prosjektutdypinskap.	
Uke 9	Analysere risikostyringen, Rapportskriving	Prosjektutdypingskap.	Skrevet prosjektutdypingskap.	
Uke 10	Rapportskriving	Prosjektutdypingskap.	Testet T-1, rapportgjennomgang	Ingen testing f.o.m onsdag
Uke 11	Testmetode / Testverktøy	Begynne å skrive resulatkap.	Skrevet testutføringskap.	Ingen testing hele uken
Uke 12	Testmetode / Testverktøy	Resultatkap.	Testet T-2	Påskeferie f.o.m torsdag
Uke 13	Testmetode / Testverktøy	Resultatkap.	Testet T-2, skrevet resultatkap.	Påskeferie t.o.m torsdag
Uke 14	Testmetode / Testverktøy	Resultatkap.	Testet T-3	
Uke 15	Testmetode / Testverktøy	Resultatkap.	Testet T-4, skrevet resultatkap.	
Uke 16	Testmetode / Testverktøy	Resultatkap.	Testet T-5, skrevet resultatkap.	
Uke 17	Testmetode / Testverktøy	Resultatkap.	Testet T-6, skrevet resultatkap.	
Uke 18	Testmetode / Testverktøy	Evalueringskap.	Rapportgjennomgang	Ingen testing hele uken
Uke 19	Testmetode / Testverktøy	Evalueringskap.	Rapportgjennomgang, startet evalueringskap.	Ingen testing hele uken
Uke 20	Rapportskriving	Evalueringskap.	Testet T-7 og T-8, skrevet resultatkap.	
Uke 21	Rapportskriving	Konklusjonskap.	Skrevet evalueringskap.	
Uke 22	Rapportskriving	Gjennomgang av rapport	Skrevet evalueringskap. og konklusjonskap.	
Uke 23	Finpusse rapport for innlevering	Klar for innlevering fredag 6. juni	Rapportgjennomgang	Levert 6. juni ☺
Uke 24		Siste frist for innlevering 10. juni		

Tabell C.1: Fremdriftsplan for prosjektet