Olli-Pekka Kuha

CLAIMS-POHJAINEN AUTENTIKOINTI JA AUKTORISOINTI

CLAIMS-POHJAINEN AUTENTIKOINTI JA AUKTORISOINTI

Olli-Pekka Kuha
Opinnäytetyö
Kevät 2014
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, langattomat laitteet

Tekijä: Olli-Pekka Kuha
Opinnäytetyön nimi: Claims-pohjainen autentikointi ja auktorisointi
Työn ohjaaja: Kari Laitinen
Työn valmistumislukukausi ja -vuosi: Kevät 2014 Sivumäärä: 43

Tässä opinnäytetyössä selvitettiin millä tavalla web-sovelluksen autentikointi ja auktorisointi olisi järkevintä toteuttaa web-sovelluksessa tilaajan vaatimusten mukaisesti. Pääpaino työssä oli Claims-pohjaisessa autentikoinnissa, mutta työn loppupuolella mukaan selvitykseen otettiin myös Claims-pohjainen auktorisointi. Auktorisointia verrattiin paremmin tunnettuun roolipohjaiseen auktorisointiin ja näin pyrittiin saamaan selville sen tuomat edut.

Opinnäytetyössä on esitelty Claims-pohjaisen autentikointijärjestelmän perusperiaatteet ja käyty läpi muutamia esimerkkitapauksia, jotka mahdollistaisivat alkuvaatimusten mukaisen järjestelmän toteuttamisen. Työssä esiteltyjä tekniikoita ei testattu käytännössä aikataulullisista syistä. Työn lähdemateriaalina on käytetty pääasiassa verkosta löytyviä e-kirjoja ja lisäksi useita Internetistä löytyviä artikkeleita.

Työn aikana kerätyn tiedon perusteella Claims-pohjainen autentikointi ja auktorisointi ovat tällä hetkellä parhaita tapoja toteuttaa web-sovellukseen toimiva käyttäjähallinta. Ne tuovat sovellukseen modulaarisuutta ja tuovat sitä kautta sovelluksen ylläpidettävyyteen huomattavasti helpotusta. Tekniikka mahdollistaa myös käyttäjähallinnan integroimisen muiden yritysten järjestelmiin, jolloin sovelluksen käyttäjälle riittää ideaalisessa tapauksessa yksi käyttäjätunnuksen ja salasanan yhdistelmä usean eri sovelluksen käyttöön.

Asiasanat: autentikointi, auktorisointi, tunnistusmenetelmät, web-sovellukset

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Wireless devices

Author: Olli-Pekka Kuha
Title of thesis: Claims Based Authentication and Authorization
Supervisor: Kari Laitinen
Term and year when the thesis was submitted: Spring 2014 Pages: 43

The objective of this thesis work was to find the best solution for authentication and authorization in an existing web application. After careful research the conclusion was that claims-based authentication and authorization is potential alternative to satisfy the requirements, specified by the owner of the web application.

Research was made using different online sources such as e-books and articles. This thesis includes some examples which meet on the given requirements. Examples are theoretical and they haven't been evaluated because of given schedule.

Claims-based authentication is a very good alternative for most web applications. It gives them more modularity because claims-based authentication decouples authentication logic from authorization logic. Additionally, in claims-based approach your application doesn't have to authenticate users because a trusted identity provider does it for your application. Claims-based application can also be integrated with different companies' user management systems so that users can use single credentials for different applications which they have to use.

Claims-based authorization gives also more modularity for the application. It makes application maintenance much easier because authorization logic can be decoupled from the application's business logic. It makes also possible to make much fine grained authorization decisions without losing control of maintenance.

Keywords: Authentication, Authorization, Claim, Web-application

ALKULAUSE

Haluan kiittää työn tilaajaa erittäin mielenkiintoisen aiheen tarjoamisesta. Minulla ei ollut ennen työn aloittamista lainkaan aikaisempaa kokemusta erilaisista autentikointitavoista, joten käytännössä kaikki työssäni eteen tulleet tekniikat olivat täysin uusia. Tämä teki työstä haastavan, mutta toisaalta juuri tämän vuoksi työ säilyi mielenkiintoisena alusta loppuun saakka.

Suuri kiitos myös työn ohjaajille, mutta vieläkin suurempi kiitos perheelleni, joka selviytyi loistavasti kunnianhimoisen aikataulun ja sitä myötä pitkien työpäivien tuomista haasteista.

Oulussa 19.5.2014

Olli-Pekka Kuha

SISÄLLYS

1 JOHDANTO

Autentikointi eli käyttäjän tunnistaminen on tänä päivänä hyvin keskeisessä roolissa web-sovelluksissa. Yhä useammat sovellukset vaativat käyttäjiltään voimassa olevia käyttäjätunnuksia, ennen kuin sovelluksista saadaan kaikki mahdolliset ominaisuudet käyttöön. Tämä on hyvin luontevaa, koska useimmat näistä sovelluksista sisältävät jotain käyttäjäkohtaista tietoa.

Kun sovelluksen käyttäjät yksilöidään käyttäjätunnusten perusteella, voidaan henkilökohtaisen käyttökokemuksen lisäksi käyttäjille antaa sovellukseen eritasoisia käyttöoikeuksia. Käyttöoikeuksien perusteella sovelluksessa tehdään päätöksiä, onko käyttäjällä lupa suorittaa jotain toiminnallisuutta vai ei. Tätä prosessia voidaan kutsua myös auktorisoinniksi.

Useimmille meistä yksi selkeimmistä ja havainnollisimmista käyttöoikeuksiin perustuvista jaotteluista on Windows-käyttöjärjestelmissä käytetyt järjestelmänvalvojatunnus ja vierailijatunnus. Kun käyttäjä tunnistetaan järjestelmänvalvojaksi, ei hänen toimiaan tietokoneella rajoiteta juuri lainkaan, kun taas vierailijatunnuksella käyttäjän oikeuksia on voitu rajoittaa hyvinkin paljon. Yleensä vierailijalta on estetty muun muassa uusien ohjelmien asennukset ja tietokoneen asetusten muokkaaminen.

Autentikointia vaativien sovellusten määrä on kasvanut huomattavasti ja tähän yksi syy on web-sovellusten yleistyminen. Tämä näkyy käyttäjille pahimmillaan kymmenien eri käyttäjätunnusten muistelemisen muodossa, minkä seurauksena usein myös salasanojen tietoturvataso laskee, koska halutaan käyttää helposti muistettavia salasanoja unohdusten vähentämiseksi. Usein saatetaan käyttää myös samoja tunnuksia useassa eri palvelussa, jolloin on vaarana, että matalan tietoturvatason sovellus joutuu hakkereiden kohteeksi ja palvelusta saatuja tunnuksia voidaan väärinkäyttää myös muissa sovelluksissa.

Nämä haasteet tiedostetaan ja tarjolla onkin olemassa esimerkiksi useita erilaisia palveluita, jonne käyttäjätunnukset ja salasanat voidaan varastoida. Ne toi-

mivat ikään kuin virtuaalisina lompakkoina. Yksi tällainen palvelu on avoimen lähdekoodin sovellus KeePass. Sovelluksen käyttäjä voi tallentaa muistiin kaikkien käyttämiensä sovellusten salasanat ja luoda sovellukseen yhden pääsalsana, jonka avulla päästään sitten käsiksi sovelluksen muistissa oleviin salasanoihin (Reichl 2014). Tilanne ei ole kuitenkaan käytettävyyden näkökulmasta paras mahdollinen. Tällä ratkaisulla ei pyritä ratkaisemaan ongelman todellista alkuperää, vaan pyritään paikkaamaan nykyistä tilannetta.

Toinen lähestymistapa tämän ongelman ratkaisemiseksi on samojen käyttäjätunnusten käyttö useassa eri palvelussa. Tällä ei tarkoiteta sitä, että rekisteröitymisen yhteydessä annettaisiin sama käyttäjätunnus ja salasana kuin jossakin toisessa palvelussa, koska tässä lähestymistavassa rekisteröitymistä ei välttämättä edes tarvita. Pääajatuksena on, että sovelluksiin voidaan tunnistautua suoraan jonkin toisen sovelluksen tunnuksilla. Esimerkiksi Facebook, Twitter ja Microsoft tarjoavat sovelluskehittäjille mahdollisuuden käyttää käyttäjän autentikoinnissa heidän jo olemassa olevia autentikointijärjestelmiä. Tällöin käyttäjä voisi kirjautua esimerkiksi keskustelufoorumille Facebook-tunnuksilla, mikäli palvelun ylläpitäjä päättää käyttää tämän mahdollisuuden hyväkseen. Tässä mallissa pyritään korjaamaan ongelman alkuperä eikä vain paikkaamaan nykyistä tilannetta.

Ongelma ei ole pelkästään kuluttajasovelluksissa, vaan heijastuu myös yritysten käytössä oleviin sovelluksiin. Niissä ratkaisumalli on kuitenkin hieman erilainen, koska yleensä käyttäjähallintaa ei haluta ulkoistaa näiden kuluttajapalveluiden harteille. Yritysmaailmassa sovellusten autentikointijärjestelmää pyritään yleensä parantamaan perusperiaatteeltaan samanlaisella tekniikalla kuin kuluttajasovelluksissakin, mutta käyttäjähallinta pidetään jossakin määrin itsellä. Yrityssovelluksissa haaste tuleekin tasapainottelussa tietoturvan ja käytettävyyden ehdoilla. Osa autentikointiin tehtävistä muutoksista nimittäin heijastuu suoraan sovelluksen käytettävyyteen. Pohdittavaksi voi tulla esimerkiksi kaksivaiheisen tunnistautumisen tarpeellisuus tai yhteiseen käyttäjähallintaan siirtyminen.

Tässä opinnäytetyössä on tarkoitus selvittää, millä tavalla käyttäjän autentikointi on nykypäivänä järkevintä toteuttaa web-sovelluksessa. Autentikointitavan tulisi olla mahdollisimman käyttäjäystävällinen kuitenkin tietoturva huomioiden. Vaatimukset käytettävälle autentikointitavalle olivat seuraavat:

- Käyttäjän ei tarvitse erikseen kirjautua sovellukseen kun työskennellään yrityksen omassa intranetissä.
- Intranetin ulkopuolelta yrityksen käyttäjä voi kirjautua sovellukseen työpaikan Active Directory -tunnuksilla.
- Järjestelmässä pitäisi olla mahdollisuus integroitua ulkopuolisiin identiteetin tarjoajiin.

Työn alkupuolella tehdyn selvitystyön perusteella aihe päätettiin rajata koskemaan Claims-pohjaista autentikointimallia, joka on uusimpia web-sovellusten autentikoinnissa käytettäviä malleja. Työn aikana perehdytään tähän malliin ja selvitetään voitaisiinko sitä hyödyntää tilaajayrityksen sovelluksessa.

Työssä käydään läpi Claims-pohjaisen autentikoinnin toiminta ja tärkeimmät seikat, jotka täytyy huomioida järjestelmää luotaessa. Lisäksi käydään läpi mahdollisia skenaarioita, jotka täyttävät yllämainitut vaatimukset. Työn loppupuolella käydään läpi myös Claims-pohjaista auktorisointimallia ja verrataan sen toimintaa roolipohjaiseen auktorisointimalliin. Auktorisointi otettiin työhön mukaan vasta työn loppupuolella, joten sen käsittely jää hieman suppeaksi.

Yrityksessä kehitetty sovellus perustuu ASP.NET MVC -arkkitehtuuriin, jonka vuoksi työn aikana tutustutaan myös sen perusperiaatteisiin. Lisäksi tutustutaan ASP.NET-sovellusten toimintaympäristöön eli Microsoftin IIS-palvelimen toimintaan, koska se on olennainen osa ASP.NET-sovelluksen toimintaa.

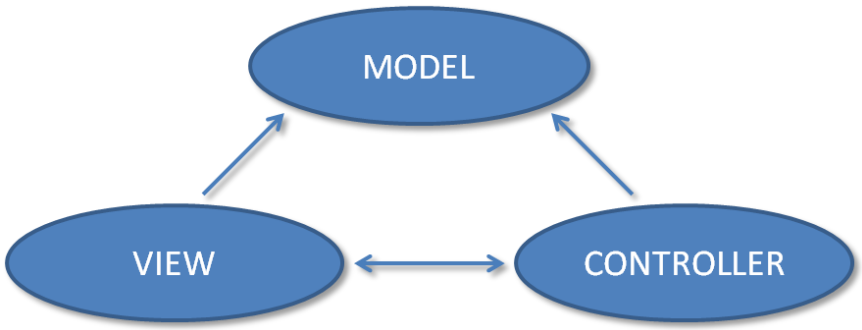
Opinnäytetyö on siis pääasiassa selvitystyötä, mutta työtä tehdessä käytiin läpi myös autentikointi- ja auktorisointitoteutuksia esitteleviä esimerkkisovelluksia. Työn lähdemateriaali koostuu useista asiaa käsittelevistä kirjoista ja lisäksi Internetistä kerätyistä lähteistä.

2 ASP. NET MVC -OHJELMISTOKEHYS

Microsoftin ASP.NET MVC on web-kehitykseen tarkoitettu ohjelmistokehys (framework), joka on rakennettu hyvin tunnetun .NET-ohjelmistokehyksen päälle. Ensimmäinen versio ASP.NET MVC -alustasta julkaistiin vuonna 2008, koska Web Forms -alusta ei enää vastannut täysin kaikkien sovelluskehittäjien tarpeita. ASP.NET MVC poikkeaa täysin Web Formsin sivupohjaisesta arkkitehtuurista ja tarjoaa Model-View-Controller (MVC)-arkkitehtuurin sen tilalle. (Chadwick – Snyder – Panda 2012.) Suomenkielessä mallia voidaan kutsua malli-näkymä-ohjainarkkitehtuuriksi.

Model-View-Controller on arkkitehtuurinen malli, jonka avulla itsenäiset sovellusosat voidaan erottaa toisistaan. MVC-arkkitehtuurin valitseminen sovellukseen tuo monia lyhyen ja pitkän tähtäimen etuja. Esimerkiksi komponenttien kehitys voidaan hajauttaa usealle taholle, koska ne eivät ole suoraan riippuvaisia toisistaan. Myös sovelluksen ylläpito helpottuu, koska yksittäisiä osia voidaan päivittää tai vaihtaa ilman, että muutoksia tarvitsee tehdä muualle. (Chadwick ym. 2012.)

MVC-malli jakaa sovelluksen kolmeen eri tasoon: Model, View ja Controller. Jokaisella näistä tasoista on tarkasti määritelty tehtävä vastuullaan ja ne eivät ota kantaa siihen miten toiset tasot hoitavat tehtävänsä. (Chadwick ym. 2012.) Kuvassa 1 on esitetty tämän mallin mukaisen sovellusosien suhde toisiinsa nähden.



KUVA 1. Model-View-Controller-arkkitehtuuri

Malli (Model) on sovelluksen osa, joka sisältää sovelluksen business-logiikan. Mallin toiminta koostuu usein luokista, joissa käsitellään muun muassa sovelluksen tietokannassa olevaa dataa.

Näkymä (View) on nimensä mukaisesti ohjelman näkyvä osa. Se on komponentti, joka tyypillisesti näyttää käyttäjälle tiettyä dataa mallista. Näkymä koostuu HTML-koodista, joka määrittää miten mallin tiedot näkyvät selaimessa.

Kontrollit (Controller) ovat luokkia, jotka hallitsevat käyttäjän lähettämiä pyyntöjä. Ne toimivat mallin ja näkymän välissä. Esimerkiksi kun käyttäjä painaa painikketta Internet-sivulla, kontrolli käsittele tämän pyynnön ja valitsee näytettäväksi oikean näkymän.

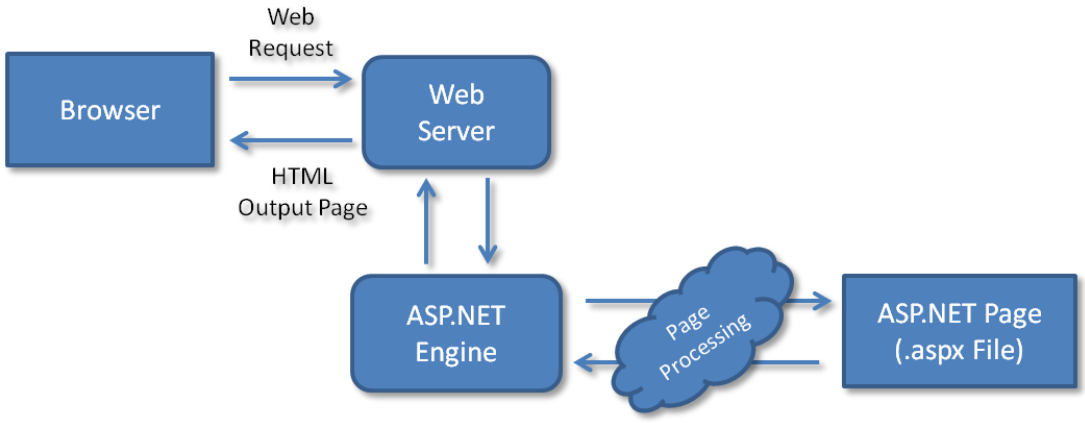
MVC on erittäin hyvä arkkitehtuuri erityisesti web-sovelluksissa käytettäväksi. Web-sovelluksissa sivujen ulkoasua halutaan usein muuttaa vastaamaan kulloinkin vallitsevia ulkoasutrendejä, mutta itse toiminnallisuus voi säilyä samana pitkänkin aikaa. Jos sovellus on rakennettu MVC-mallin mukaisesti, voidaan muutokset kohdistaa parhaimmillaan pelkästään näkymiin.

3 INTERNET INFORMATION SERVICE (IIS)

ASP.NET-sovellukset toimivat aina yhdessä web-palvelimen kanssa. Ohjelmistokehityksen aikana sovellusta ajetaan yleensä Visual Studion tarjoaman IIS Express -palvelimen kautta. Kun sovellus halutaan julkaista laajemmalle käyttäjäkunnalle, tarvitaan kuitenkin täysiversio IIS-palvelin ohjelmistosta.

Yksinkertaisimmillaan palvelimen tehtävä voi olla tavallisen HTML-sivun palauttaminen selaimelle. Käytännössä tässä tapauksessa palvelin vain lukee pyydetyn tiedoston kovalevyltä ja lähettää sen selaimeen, jonka tehtävä on näyttää tiedosto käyttäjälle. Tilanne muuttuu kuitenkin hieman monimutkaisemmaksi, kun kyseeseen tulee dynaaminen sisältö, kuten ASP.NET-sivu. (MacDonald 2011.)

Kun ASP.NET-sovelluksessa palvelimelta pyydettään esimerkiksi Default.aspx-sivua, lähettää palvelin pyynnön ASP.NET-enginen yli. ASP.NET-engine lataa pyydetyn sivun ja ajaa sen sisältämät koodit, jonka jälkeen luodaan lopullinen HTML-dokumentti lähetettäväksi takasin IIS-palvelimelle. IIS lähettää tämän HTML-dokumentin vielä takaisin sitä pyytäneelle selaimelle. Itse palvelin ei siis osaa käsitellä esimerkiksi C#-koodia, vaan se tukeutuu ASP.NET-enginen apuun raskaampien operaatioiden suorittamisessa. Kuvassa 2 on esitetty IIS-palvelimen toimintaa. (MacDonald 2011.)



KUVA 2. IIS:n toiminta web-kyselyn saapuessa (MacDonald 2011.)

4 AUTENTIKOINTI

Useimmat meistä käyttävät erilaisia autentikointijärjestelmiä päivittäin, sillä kirjautuessamme johonkin järjestelmään tai sovellukseen läpäisemme aina jonkinlaisen autentikointiprosessin. Tässä luvussa pyritään selventämään tietojärjestelmien pääsynhallintaprosessin kahta keskeisintä termiä, autentikointia ja auktorisointia. Näitä termejä kuulee monesti käytettävän sekaisin, vaikka ne ovat selvästi eroteltavissa toisistaan. Lopuksi käydään vielä tarkemmin läpi, mitä autentikointi tarkoittaa web-sovelluksissa.

4.1 Autentikoinnin ja auktorisoinnin ero

Yksi yleisimmistä tavoista hallita erilaisten järjestelmien käyttöä on identifioida käyttäjä ja sen perusteella tehdä päätös, mitä tämä käyttäjä saa järjestelmässä tehdä. Näiden kahden hallintaprosessin, autentikoinnin ja auktorisoinnin, tarkoituksena on antaa vain määrätyille henkilöille oikeus järjestelmän käyttöön ja estää ei-toivotut käyttäjät. Autentikoinnin tehtävä on varmistaa, että käyttäjä on sama henkilö, joksi hän itseään väittää, kun taas auktorisointi määrittää, mitä tämä käyttäjä saa järjestelmässä tehdä. Tämä prosessi pitäisi aina hoitaa niin, että järjestelmän käyttäjällä on vain ja ainoastaan se määrä oikeuksia, joilla hän voi hoitaa työnsä sujuvasti. (Rhodes-Ousley 2013, 7.)

4.2 Autentikointi tietojärjestelmissä

Perinteinen sovelluksen autentikointi perustuu yleensä käyttäjätunnuksen ja salasanan yhdistelmään ja se onkin ollut käytössä aivan tietokoneiden alkutaipaleelta saakka. Perusajatus on, että käyttäjätunnus on jokin julkinen tieto käyttäjästä ja se liitetään salasanaan, jonka tietää ainoastaan kyseisen käyttäjätunnuksen haltija. (Rhodes-Ousley 2013, 7.)

Käyttäjätunnuksen ja salasanan yhdistelmä ei ole tietoturvamielessä kovinkaan vahva autentikointimenetelmä, koska salasana voidaan kaapata hyvin monella eri tavalla. Pahimmassa tapauksessa se löytyy paperilapulta tietokonepöydältä

tai on muuten helposti arvattavissa. Korkeampaa tietoturvaa vaativissa sovelluksissa tulisikin käyttää monivaiheista autentikointia.

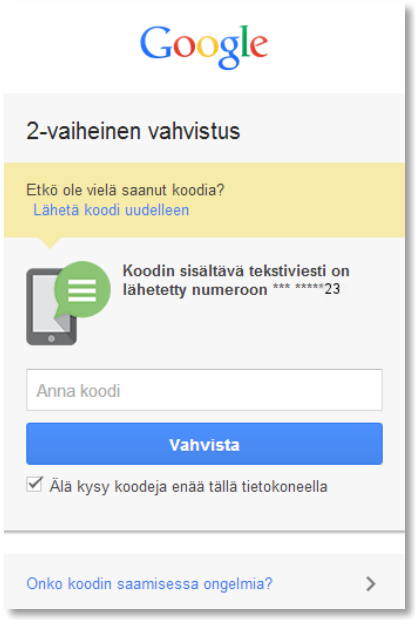
Monivaiheinen autentikointi tarkoittaa sitä, että autentikointiin käytetään vähintään kahta seuraavista asioista käyttäjätunnuksen lisäksi:

- jotain mitä käyttäjä tietää (salasana, PIN-koodi)
- jotain mitä käyttäjä omistaa (älykortti, puhelin)
- jotain mitä käyttäjä on (sormenjälki) (Rhodes-Ousley 2013, 7).

Yleisimmin käytetään kaksivaiheista autentikointia, joka muodostuu salasanan ja esimerkiksi älykortin yhdistelmästä. Biometristen tunnisteiden käyttö, eli esimerkiksi sormenjälkitunnistus, on vielä suhteellisen harvinaista, koska se vaatii erillisen lukulaitteen. Monivaiheinen autentikointikaan ei ole täysin murtovarma, mutta väärinkäyttö on huomattavasti vaikeampaa kuin yksivaiheisessa järjestelmässä. (Rhodes-Ousley 2013, 7.)

4.3 Kaksivaiheinen autentikointi

Aikaisemmin kuluttajasovelluksissa kaksivaiheisen autentikoinnin tyylistä tekniikkaa käytettiin lähinnä tilanteissa, joissa käyttäjätunnus tai salasana oli hävinnyt ja tarvittiin väliaikainen salasana vanhan uusimiseen. Nykyään tekniikkaa on alettu tuoda myös vakituiseen käyttöön. (Kissell 2014.) Esimerkiksi hakukonejätti Google mahdollistaa kaksivaiheisen tunnistautumisen omiin palveluihinsa. Kuvassa 3 on kuvankaappaus Google-tilin kaksivaiheisesta kirjautumisikkunasta. Salasanan syöttämisen jälkeen palvelu pyytää käyttäjää syöttämään kuusinumeroisen lukusarjan, joka lähetetään käyttäjän matkapuhelimeen SMS-viestinä.



KUVA 3. Muun muassa Google mahdollistaa SMS-viesteihin integroidun kaksivaiheinen autentikoinnin (www.google.com-sivuston kirjautuminen)

Vaikka käyttäjällä on käytössä kaksivaiheinen kirjautuminen, ei palveluun pääsy esty puhelimen ollessa tavoittamattomissa. Palvelusta on mahdollisuus tulostaa sarja kertakäyttöisiä salasanoja, joita voidaan käyttää puhelimeen saapuvan numerosarjan sijaan, mikäli puhelin ei ole saatavissa tai tekstiviestipalvelussa on muuten jotakin vialla. Tämä on erittäin tärkeä ominaisuus varsinkin, kun kyseessä on suuren käyttäjämäärän palvelu. Ilman kertakäyttöisiä salasanoja palvelun ylläpito kuormittuu asiakkaista, joiden kaksivaiheisessa kirjautumisessa on tullut ongelmia.

Googlen kaksivaiheisessa kirjautumisessa käytettävyyttä lisää myös ”Älä kysy koodeja enää tällä koneella” -ominaisuus. Käyttäjä voi asettaa tämän päälle kotona olevalle tietokoneelle, koska on epätodennäköistä, että käyttäjätunnuksia käytetään väärin juuri omalta koneelta. Ensimmäisen kaksivaiheisen kirjautumisen jälkeen tilille voidaan siis kirjautua pelkällä käyttäjätunnuksen ja salasanan yhdistelmällä ja silti tili on huomattavasti paremmin turvassa väärinkäytöksiltä.

5 CLAIMS-POHJAINEN AUTENTIKOINTI

Lähes jokainen web-sovellus tarvitsee nykypäivänä jonkinlaisen autentikointilogiikan. Microsoft tarjoaa ASP.NET-pohjaisille sovelluksille muun muassa Forms Authentication -moduulia, joka on luultavasti helpoin ja nopein tapa rakentaa ASP.NET-sovellukseen toimiva autentikointilogiikka. Tämä ei ole kuitenkaan paras vaihtoehto kaikille sovelluksille.

Nykypäivänä sovelluksista pyritään rakentamaan mahdollisimman modulaarisia. Useissa sovelluksissa autentikointilogiikka kuitenkin hieman rikkoo tätä modulaarisuutta. Se voi olla sotkeutunut auktorisointilogiikan kanssa tai muuten huonosti toteutettu, jolloin siitä voi muodostua myös tietoturvauhka. Usein sovelluskehittäjillä ei ole riittävää tietotaitoa autentikointilogiikan tietoturvallisesta toteutuksesta. Autentikointilogiikan siirtäminen toiselle taholle olisi siis vastaus usean sovelluskehittäjän pyyntöön.

Näitä haasteita on pyritty ratkaisemaan Claims-pohjaisessa autentikointimallissa. Toisin kuin perinteisesti tunnetuissa autentikointimalleissa, Claims-pohjaisessa sovelluksessa käyttäjätunnusta ja salasanaa ei koskaan syötetä suoraan itse ohjelmaan. Käyttäjätunnuksen ja salasanan sijaan käyttäjä lähettää sovellukselle nipun claimeja, jotka sisältävät käyttäjää koskevia tietoja. (Lakshmiraghavan 2013.)

Autentikoinnista ei kuitenkaan tässäkään mallissa päästä eroon, vaan käyttäjän täytyy syöttää voimassa olevat tunnistautumistiedot sovelluksen sijaan claimien myöntäjälle (Identity Provider). Käyttäjän tunnistaminen näin ollen vain siirretään erilliseen palveluun. Kun claimien myöntäjä on varmistanut syötettyjen tietojen oikeellisuuden, saa käyttäjä haltuunsa sovellukselle lähetettävät claimit. Toiminta perustuu siis sovelluksen ja claimien myöntäjän väliseen luottamukseen. (Lakshmiraghavan 2013.)

Periaatteessa claimien käytössä ole mitään uutta, koska jo hyvin alkeelliset käyttöjärjestelmät lähettivät sovelluksille claimin eli tiedon, josta sovellus tunnisti

kuka käyttäjä oli kulloinkin kyseessä. Käyttöjärjestelmä tunnisti käyttäjän käyttäjätunnuksen ja salasanan perusteella. (Baier ym. 2011, 18.)

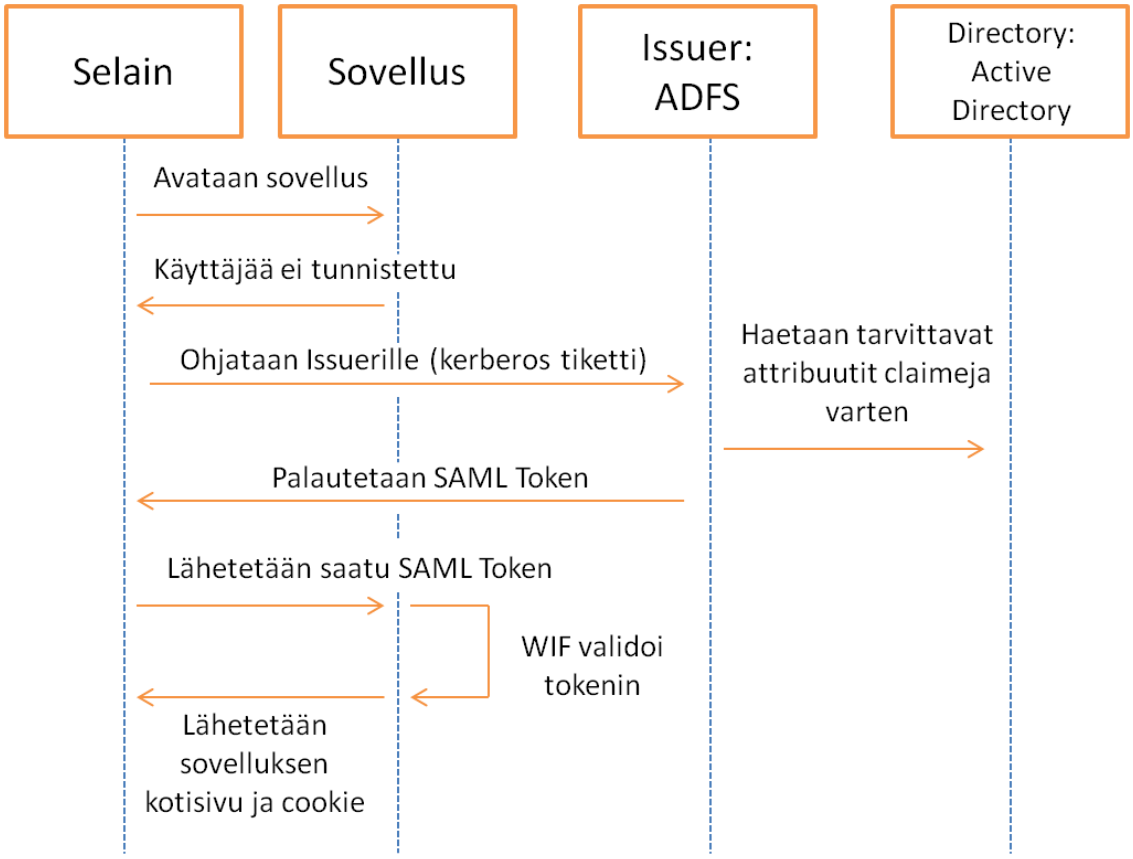
Seuraavissa alakohdissa käydään läpi miten nykyisin käytettävä Claims-pohjainen autentikointi toimii ja esitellään sen toteuttamiseen tarvittavat komponentit. Lopuksi käydään vielä läpi, mitä asioita olisi hyvä huomioida, kun tällaista järjestelmää aletaan toteuttaa. Ennen kuin käydään läpi tarkemmin Claims-pohjaisen järjestelmän periaatetta, on syytä esitellä keskeisimmät termit selityksineen.

- **Claim** - Käyttäjää kuvaava tieto, jonka sovellus hakee yleensä käyttäjätietokannasta. Koostuu tyypistä ja arvosta.
- **Token** - Paketti, joka sisältää useita claimeja
- **Identity Provider** – Palvelu, joka hoitaa käyttäjän autentikoinnin ja myöntää sovelluksen tarvitsemat claimit. Sisältää myös Security Token Servicen.
- **Issuer** – Claimit myöntävä kokonaisuus, kuten Identity Provider. Näitä kahta termiä näkee käytettävän monesti sekaisin.
- **Security Token Service** - Muodostaa tokenin, attribuuttivarastosta löytyvien tietojen perusteella

5.1 Claims-pohjaisen autentikoinnin toiminta

Koko järjestelmän keskiössä ovat claimit ja niistä muodostuvat tokenit. Oikeastaan tiivistetysti voidaankin sanoa, että koko toiminta perustuu juuri tokenien lähettelemiseen toisiinsa luottavien palveluiden välillä.

Token muodostuu yhdestä tai useammasta claimista, joista jokainen sisältää jotain käyttäjää kuvaavaa tietoa. Tokenin muodostaa Security Token Service (STS), joka saa tarvittavat tiedot tokenin muodostamista varten attribuuttitietokannasta, joka sisältää tietoja niin käyttäjästä kuin STS:n luottamista sovelluksista. (Chappell 2011, 4–5.) Kuvassa 4 havainnollistetaan claims-pohjaisen järjestelmän perusperiaatetta.



KUVA 4. Claims-pohjaisen järjestelmän perusperiaate

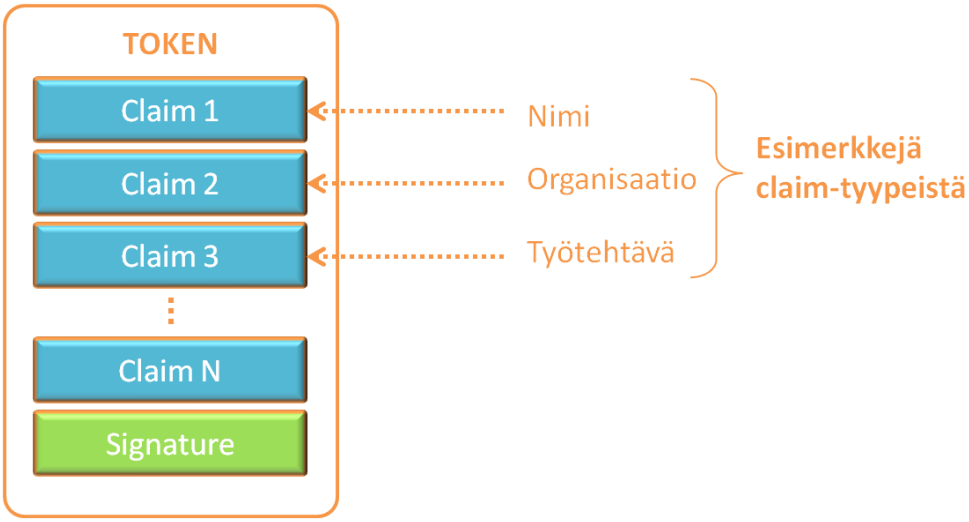
Ensimmäisessä vaiheessa käyttäjä yrittää avata yhteyden sovellukseen, mutta sovellus ei tunnista käyttäjää, jolloin käyttäjä ohjataan sovelluksen tuntemalle Identity Providerille. Tässä tapauksessa käyttäjällä on Windowsiin kirjautumisen yhteydessä saatu Kerberos-tiketti, jonka perusteella ADFS tunnistaa käyttäjän. Tämän jälkeen ADFS hakee tarvittavat tiedot käyttäjästä Active Directory -käyttäjätietokannasta ja muodostaa SAML-formaattia olevan tokenin. Token lähetetään takaisin käyttäjälle, jonka jälkeen se voidaan lähettää sitä pyytäneelle sovellukselle. Sovelluksessa tokenin käsittelyyn käytetään WIF:n tarjoamia ohjelmistomoduuleita. Kun sovellus on saanut tokenin ja sen sisältämät claimit käsiteltyä, voidaan käyttäjälle avata sovelluksen etusivu ja lähettää jatkokäyttöä varten eväste (cookie). (Baier ym. 2011, 20–21.) Suurin osa näistä vaiheista tapahtuu käyttäjän huomaamatta. Oikeastaan ainut käyttäjälle näkyvä vaihe on käyttäjätietojen syöttäminen ADFS-palveluun, mikäli ei ole mahdollista käyttää esimerkiksi Windowsin Kerberos-tikettiä.

Tämä samainen toimintapa on tuttu myös ihmisten normaalista käyttäytymisestä, jossa usein vastaanotetun tiedon hyväksyminen vaatii myös luottamuksen kahden tahon välillä. Claims-pohjaisen järjestelmän toimintaa voidaankin havainnollistaa hyvin seuraavalla arkielämän esimerkillä. Kuvitellaan, että Pekka ja Antti ovat toisilleen vain etäisesti tuttuja jonkun yhteisen kaverin kautta. Kun he tapaavat ensimmäisen kerran, tämä molempien tuntema kaveri esittelee Pekan ja Antin toisilleen. Hän kertoo esitellessään muun muassa henkiöiden nimet ja mahdollisesti muita tärkeitä tietoja heihin liittyen. Vaikka Pekka ei olisi koskaan aikaisemmin tavannut Anttia, hän luultavasti luotaisi kuulemiinsa tietoihin ja toimii jatkossa niiden mukaisesti. Tämä johtuu siitä, että Pekka luottaa kaveriinsa. Tässä esimerkissä kuvatussa tilanteessa voidaan ajatella, että Antti on claims-pohjainen sovellus, joka vastaanottaa tietoja kaveriltaan. Pekan ja Antin yhteistä kaveria voidaan ajatella Identity Providerina ja Anttia sovelluksen käyttäjänä. (Rountree 2012.)

5.1.1 Claim

Claimit sisältävät yleensä käyttäjää kuvaavia tietoja eli tietoja käyttäjän digitaalisesta identiteetistä. Käytännössä ei ole kuitenkaan olemassa mitään rajoitusta, millaista tietoa sovellukselle voidaan antaa claimien muodossa. Claim kostuu aina tyypistä ja arvosta. Claimin tyyppi voisi olla esimerkiksi ”Etunimi” ja sen arvo esimerkiksi ”Pekka”

Claimien tyypit määräytyvät käytännössä sen mukaan, mitä kohdesovellus tarvitsee toimiakseen. Hyvin yleisiä claim-tyyppejä ovat muun muassa Etunimi, Sukunimi ja Sähköpostiosoite (Baier ym. 2011, 42). Näiden yleisyys johtunee siitä, että ne ovat hyvin keskeistä tietoa sovellusten näkökulmasta, kun puhutaan käyttäjän identiteetistä. Kuvassa 5 havainnollistetaan claimien suhdetta tokeniin.



KUVA 5. Claimien ja tokenin havainnollistaminen

Käytettävien claim-tyyppien suunnitteluun ei voida antaa tarkkoja suunnitteluohjeita, eikä suoraan voida kertoa, onko jokin claim toista parempi, koska tämä riippuu hyvin pitkälti sitä käyttävästä sovelluksesta. Suunniteltaessa claim-tyyppejä onkin hyvin tärkeä ymmärtää niiden käyttökohde ja mistä koko claims-pohjaisessa järjestelmässä on kyse. Kun aletaan tekemään päätöksiä käytettävistä tyypeistä, on syytä kuitenkin punnita niiden sopivuutta käymällä läpi seuraavat kysymykset:

- Onko tämä tieto ydinosaa siitä tiedosta, miten määrittelen käyttäjän identiteetin?
- Onko Identity Providerilla valta tähän tietoon?
- Tullaanko tietoa käyttämään useammassa kuin yhdessä sovelluksessa?
- Halutaanko, että Identity Provider hallinnoi tätä tietoa vai pitäisikö sovellukseni hallinnoida sitä paikallisesti?

Claimien pääkandidaatti tyyppi lähes jokaiseen järjestelmään on sähköpostiosoite, koska siihen on yleensä tiivistetty hyvin käyttäjän identiteetti sovelluksen näkökulmasta. Siitä voidaan usein muun muassa päätellä, missä yrityksessä käyttäjä työskentelee, ja lisäksi se tarvitaan joka tapauksessa, jos päädytään käyttämään yritysten välistä yhteistä käyttäjähallintaa (Federated Identity). (Baier ym. 2011, 72.) Sähköpostiosoite on myös mitä luultavimmin käyttökelpoinen

claim mihin tahansa web-sovellukseen ja usein se on jopa määrätty pakollisten claimien joukkoon.

Sähköpostiosoitteen toimiessa hyvänä esimerkkinä käyttökelpoisesta claimista niin huonona esimerkkinä voisi olla käyttäjän valitsema sovelluksessa käytettävä teema tai väri. Tämäkin tieto voitaisiin liittää yhdeksi claimiksi, mutta se on hyvin sovelluskohtainen tieto ja sitä tuskin voidaan käyttää missään muussa sovelluksessa. Tämän kaltaiset tiedot on siis järkevintä hallita sovelluksessa paikallisesti. (Baier ym. 2011, 72.)

5.1.2 Security Token

Yksinkertaisesti tiivistettynä token voidaan mieltää claimeista koostuvaksi digitaalisesti allekirjoitetuksi paketiksi. Tämä paketointi tehdään Security Token Servicessä (STS), joka hoitaa digitaalisen allekirjoituksen lisäksi tokenin salauksen. Allekirjoituksen perusteella sovelluksessa voidaan varmistua siitä, että saapunut Token on peräisin luotetusta lähteestä. Lisäksi voidaan varmistaa, että Tokenin sisältämä tieto on pysynyt muuttumattomana lähetysten välillä. (Chappell 2011, 5.)

Tokeneita on kolmea eri formaattia: Security Assertion Markup Language (SAML), Simple Web Token (SWT) ja JSON Web Token (JWT). Näistä yleisimmin käytetään SAML-formaattia. Taulukossa 1 esitellään näiden kolmen formaatin keskeisimpiä teknisiä ominaisuuksia. (Lakshmiraghavan, 2013.)

TAULUKKO 1. Token-formaattien ominaisuudet (Lakshmiraghavan, 2013.)

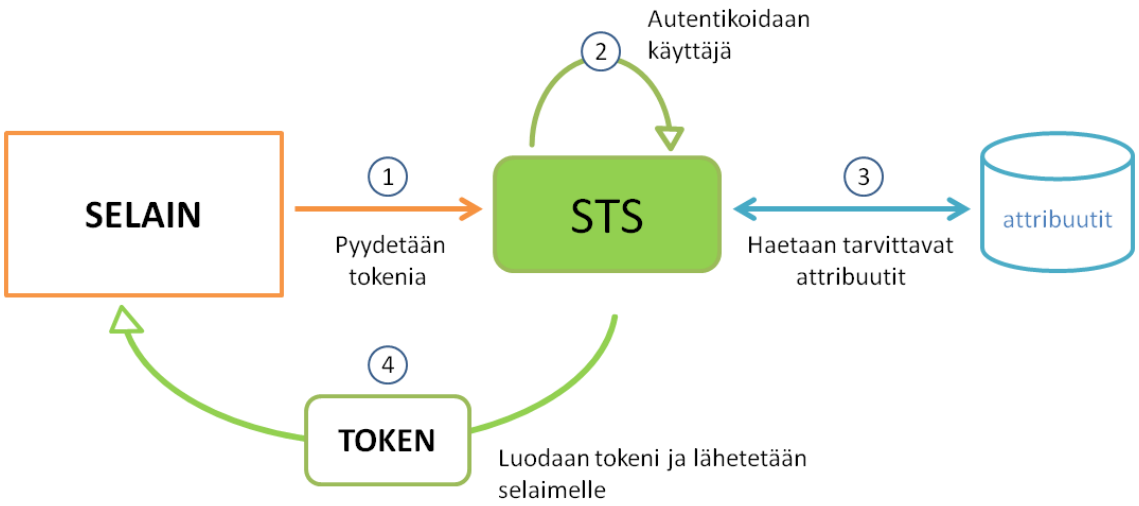
	SAML	SWT	JWT
Representation	XML	HTML Form encoding	JSON
Geared Toward	SOAP	REST	REST
Out-of-the-Box WIF Support	Yes	No	No
Protocols	WS-Trust and WS-Federation	OAuth 2.0	OAuth 2.0
Typical Carrier	HTTP body or URL	HTTP Auth header (Bearer)	HTTP Auth header (Bearer)
Support for Signing	Yes, asymmetric key - X509 certificate	Yes, HMAC SHA-256 using symmetric key	Yes, both symmetric and asymmetric signing
Support for Encryption	Yes	No	Yes

Tässä työssä ei syvennytä tarkemmin näiden formaattien teknisiin ominaisuuksiin, koska niiden käsittelyyn käytetään sovelluskehityksessä yleensä valmiita ohjelmistomoduuleita. Eri formaatteihin täytyy kuitenkin kiinnittää huomiota järjestelmän suunnitteluvaiheessa, jotta valitut järjestelmäkomponentit ovat yhteensopivia. Etenkin jos järjestelmään on tarkoitus rakentaa mahdollisuus ulkopuolisiin integrointeihin, täytyy oman STS:n tukea myös ulkopuolisen STS:n luomia formaatteja.

5.1.3 Security Token Service (STS)

Security Token Service (STS) on palvelu, johon käyttäjien sovelluskohtaiset token pyynnöt tulevat. Sen jälkeen kun STS on tunnistanut käyttäjän, muodostaa se kohdesovelluksen vaatimusten mukaisen tokenin. STS:n pitää siis autentikoida käyttäjä ennen tokenin myöntämistä. Tämä autentikointi hoidetaan yleisimmin käyttäjätunnuksen ja salasanan yhdistelmällä. (Lakshmiraghavan, 2013.) Käytettävää autentikointitapaa ei ole kuitenkaan rajoitettu millään tavalla, vaan se voidaan valita tapauskohtaisesti tietoturvavaatimusten mukaisesti. Valittavissa olevat autentikointitavat riippuvat Identity Providerin ominaisuuksista.

Käyttäjien autentikointi voidaan hoitaa joko paikallisessa STS:ssä tai se voidaan ulkoistaa jollekin toiselle luotetulle Identity Provider -palvelulle (IdP) (Lakshmiraghavan, 2013). Kun käyttäjän autentikointi annetaan toisen palvelun harteille, STS:n toiminnasta käytetään yleensä nimitystä Federation Provider. Kuvassa 6 on havainnollistettu STS:n toimintaa.



KUVA 6. STS:n toiminnan havainnollistaminen

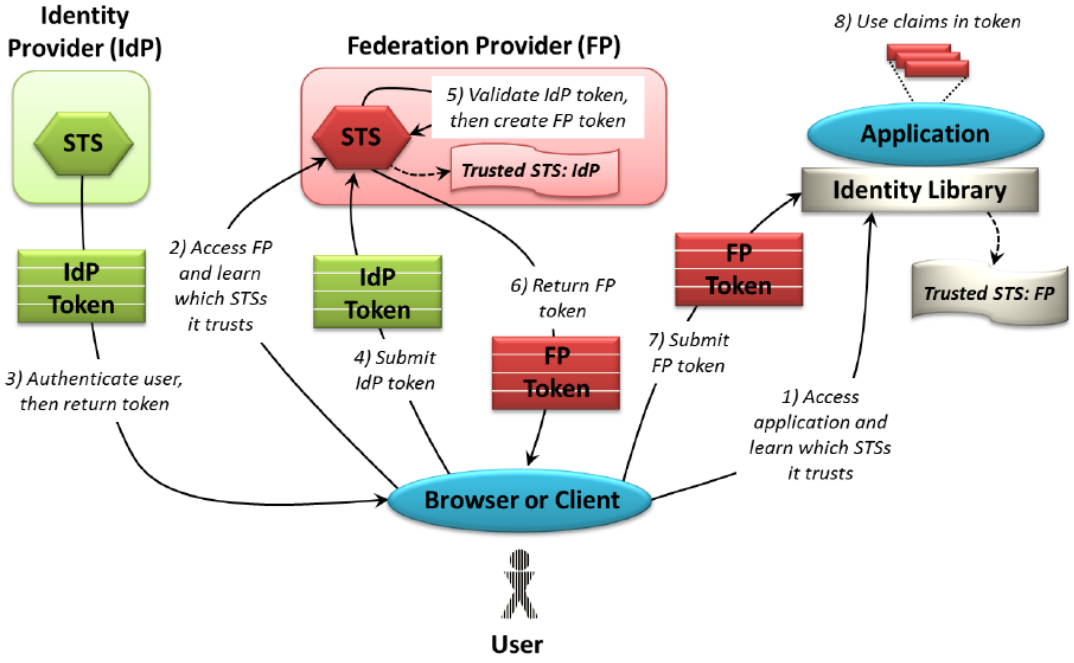
STS perustuu yleensä WS-Trust- ja WS-Federation-protokollien mukaiseen toimintaan. Nämä protokollat ovat hyvin työläitä käsitellä, minkä vuoksi onkin suositeltavaa käyttää esimerkiksi Microsoftin valmiiksi rakentamia Windows Identity Foundation -luokkia näiden protokollien käsittelyyn. Suositeltavaa on käyttää jotain valmista STS-palvelua, mutta jos mikään valmis ratkaisu ei sovi kehitettävään järjestelmään, helpottavat WIF:n sisältämät luokat myös oman STS-palvelun rakentamisessa. (Windows Identity Foundation 4.5 Overview 2014.)

5.1.4 Identity Provider (IdP)

Identity Provider on kokonaisuus, joka hoitaa käyttäjän autentikoinnin ja tokenien myöntämisen. Tätä termiä näkee käytettävän sekaisin Security Token Servicen kanssa, koska sillä on suuri vaikutus IdP:n toiminnassa. Identity Provider ei kuitenkaan ole käsitteellisesti sama asia kuin STS. Identity Provider tarvitsee miltei aina STS-komponentin, kun taas STS voi toimia myös yksinään. (Bertocci 2008.)

Parhaiten näiden kahden käsitteen eron ymmärtää, kun ymmärtää Identity Providerin ja Federation Providerin eron. Identity Provider myöntää tokenin, joka sisältää käyttäjää kuvaavaa tietoa, jolloin sillä täytyy olla yhteys johonkin käyttäjätietokantaan tarvittavien tietojen keräämiseen. Federation Provider hoitaa

yleensä pelkästään tokenien muuntamisen, jolloin ulkoisen Identity Providerin myöntämä token muunnetaan kohdesovelluksen vaatimaan muotoon. Tähän muuntamiseen tarvitaan pelkästään STS-komponentti, joka ei sellaisenaan ole kykenevä myöntämään käyttökelpoisia tokeneita millekään sovellukselle. Kuvassa 7 punaisella merkitty STS toimii juuri tokenin muuntajana, jotta käyttäjä pääsee kirjautumaan haluamaansa sovellukseen.



KUVA 7. STS Federation Providerina (Chappell 2011, 9.)

Identity Providereina toimivia tahoja on nykyään hyvin paljon. Itse asiassa jokaista yritystä, joka käyttää yrityksen omaa STS-komponenttia tokenien myöntämiseen, voidaan kutsua Identity Provideriksi. Identity Providereiden määrää lisäävät myös useat Internetissä toimivat suuryritykset, jotka toimivat kuluttajilleen Identity Providereina mahdollistaen yhden käyttäjätunnuksen ja salasanan yhdistelmän käytön useassa eri palvelussa. Tällaisia Single-Sign-On-tyylisiä palveluita tarjoavat muun muassa Facebook, Google ja Microsoft LiveID. (Chappell 2011, 7.)

Yrityssovelluksille ei kuitenkaan voida suositella näihin palveluihin integroitumista, koska tämä heikentää mahdollisuuksia hallita sovelluksen käyttäjäkuntaa.

Vaikka Internetissä toimivia Identity Providereita ei käytettäisikään, ei omaa Identity Provider -palvelua tarvitse kuitenkaan rakentaa tyhjästä.

5.2 Claims-pohjaisen järjestelmän toteuttaminen

Claims-pohjaisen järjestelmän toteuttaminen voidaan jakaa vaiheisiin, joiden avulla myös rakentuu hyvin selkeä kuva, millaisesta järjestelmästä kaiken kaikkiaan on kyse. Seuraavaksi käydään läpi neljä eri vaihetta, jotka tulevat vastaan miltei aina, kun Claims-pohjaista järjestelmää aletaan toteuttamaan. Jokaisesta vaiheesta on myös kerrottu tiivistetysti huomionarvoisia seikkoja.

5.2.1 Claimeja tukeva sovellus

Claims-pohjaisessa sovelluksessa täytyy olla logiikka, jolla voidaan käsitellä saapuvia tokeneita ja jäsentää niiden sisältämät claimit. Useissa sovelluksissa claimeja käytetään myös auktorisoinnin toteuttamiseen, joka vaatii oman logiikkansa.

Microsoftin Windows Identity Foundation (WIF) tarjoaa hyvän alustan sovelluskehittäjille claims-pohjaisen sovelluksen toteuttamiseen. Se sisältää kokoelman .NET Framework -luokkia, joita voidaan käyttää sekä Windows Communication Foundation (WCF)- että ASP.NET-sovelluksissa. WIF sisältää muun muassa Identity.Claims-propertyn, joka identifioi myönnetyt claimit, niiden myöntäjän ja niiden sisällön. (Baier ym. 2011, 46.)

5.2.2 Identity Providerin hankinta

Useimmille tahoille on helpointa ja tietoturvamielessä järkevintä käyttää jotain valmista ratkaisua. Hyvin tunnettuja vaihtoehtoja ovat muun muassa Microsoftin ADFS 2.0 ja ACS. (Baier ym. 2011, 47.) Näissä on kuitenkin omat rajoituksensa, mikäli halutaan esimerkiksi käyttää omia attribuuttivarastoja Microsoftin oman Active Directory -palvelun sijaan. Markkinoilta löytyy nykyään hyviä kaupallisia vaihtoehtoja ja myös avoimen lähdekoodin ratkaisuja kuten ThinkTecturen Identity Server.

5.2.3 Sovelluksen ja Identity Providerin luottamuksen konfigurointi

Kun sovellus on muokattu tukemaan claimeja ja sen rinnalle on rakennettu sopiva Identity Provider, on seuraava vaihe luottamuksen konfigurointi näiden kahden osapuolen välille. Sovelluksen täytyy tunnistaa käytettävä Identity Provider, koska sovelluksen autentikointilogiikka siirretään tässä vaiheessa täysin sen harteille. Seuraavat seikat ovat huomioitava, kun yhteyttä sovelluksen ja Identity Providerin välille konfiguroidaan (Baier ym. 2011, 47):

- Mitä claimeja Identity Provider tarjoaa?
- Mitä avainta sovelluksessa käytetään tokenissa olevan digitaalisen allekirjoituksen varmistamiseen?
- Mihin URL-osoitteeseen käyttäjät täytyy ohjata, kun Identity Providerilta pyydetään claimeja?

Konfigurointia helpottamaan on Visual Studioon ladattavissa Identity and Access Tool -lisäosa, joka helpottaa tämän vaiheen suorittamista huomattavasti. Sen avulla voidaan konfiguroida luottamus olemassa olevaan Identity Provideriin tai sovelluksen testausta varten käytettävään Identity Provideria simuloivaan Local STS -komponenttiin. (What’s New in Windows Identity Foundation 4.5 2014)

5.2.4 Sovelluksen tietojen konfigurointi Identity Provideriin

Identity Provider tarvitsee myös tietoja sovelluksesta, jotta se voi muodostaa sovelluksen käyttöön oikeanlaisia tokeneita. Ainakin seuraavat asiat täytyy määrittää:

- Mikä on sovelluksen Unique Resource Identifier (URI)?
- Mitkä claimit ovat sovelluksen kannalta välttämättömiä ja mitkä ovat valinnaisia?
- Täytyykö tokenit salata ja jos tarvitsee, niin millä salausavaimella?
- Mitä URL-osoitetta sovellus käyttää tokenien vastaanottamiseen? (Baier ym. 2011, 48.)

Identity Provider täytyy siis konfiguroida joka kerta, kun sitä halutaan käyttää uudessa sovelluksessa. Tämä johtuu siitä, että mikään sovellus ei ole samanlainen, jolloin yleensä myös tarvittaville claimeille on erilaiset vaatimukset. Jokaisessa Identity Providerille lähetetyssä token-pyynnössä on mukana Unique Resource Identifier (URI)-tunniste, josta käyttäjän tavoittelema sovellus tunnistetaan. Luontevinta on käyttää tunnisteena sovelluksen URL-osoitetta. (Baier ym. 2011, 49.)

5.3 Windows Identity Foundation

Windows Identity Foundation (WIF) on Microsoftin tarjoama ohjelmistokehys, jonka avulla voidaan kehittää Claims-pohjaisia sovelluksia ja token-palveluita (Rountree, 2012). Ensimmäinen julkaistu WIF-versio oli 3.5, joka julkaistiin samaan aikaan .NET 3.5 -version kanssa. WIF on integroitu osaksi .NET-ohjelmistokehystä versiosta 4.5 eteenpäin. (What's New in Windows Identity Foundation 4.5 2014.)

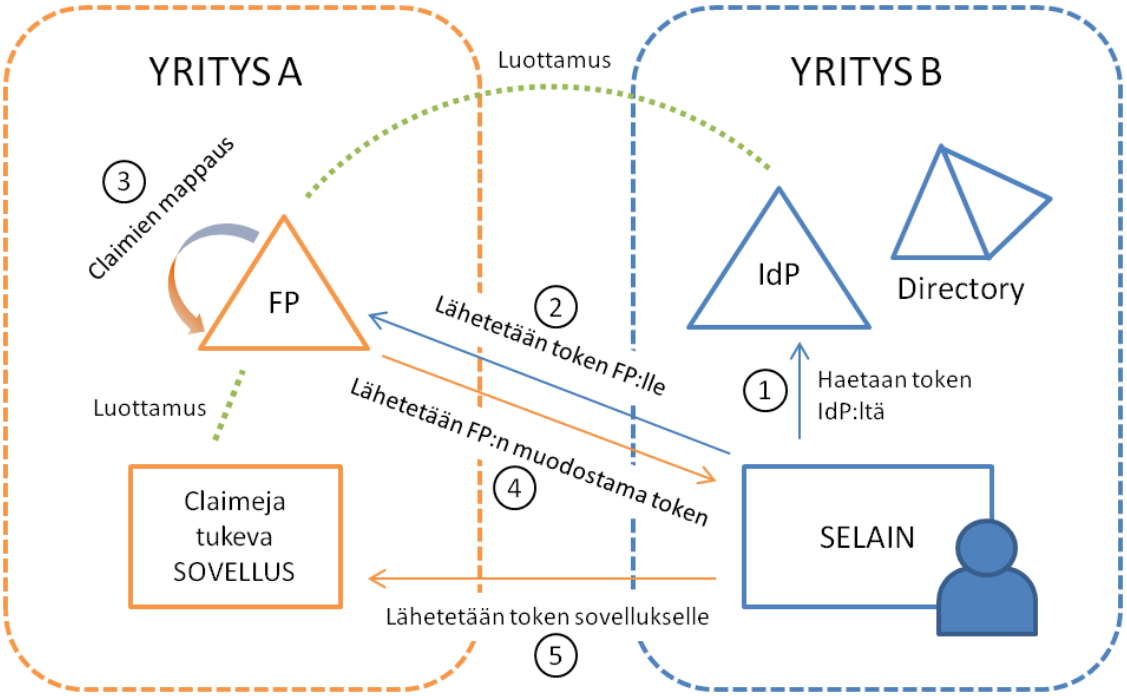
WIF on vain yksi osa Microsoftin tuoteperhettä, jonka avulla yritys pyrkii helpottamaan Claims-pohjaisten sovellusten toteuttamista. Muita samaan tuoteperheeseen kuuluvia palasia ovat Active Directory Federation Services (ADFS 2.0) ja Windows Azure Access Control Services (ACS).

WIF on rakennettu toimimaan WS-Trust- ja WS-Fredaration-protokollien mukaan, jolloin sovelluskehittäjille voidaan tarjota valmiita API-rajapintoja Claims-pohjaisen sovelluksen kehittämiseen. Sovellus voi käyttää WIF:n tarjoamia luokkia muun muassa STS:n lähettäminen tokenien prosessointiin ja identiteettiin pohjautuvien päätösten tekemiseen sovelluksessa tai web-servicessä. Tarvittaessa WIF:n avulla voidaan rakentaa myös oma STS-komponentti. (Windows Identity Foundation 4.5 Overview 2014.)

5.4 Integroitumisen mahdollistavat arkkitehtuurit

Yksi Claims-pohjaisen sovelluksen suurimmista hyödyistä on käyttäjähallinnan helpottuminen. Käyttäjähallintaa helpottaa yhteiseen käyttäjähallintaan siirtymi-

nen (Federated Identity). Yhteinen käyttäjähallinta tarkoittaa käytännössä sitä, että käyttäjä voi kirjautua oman yrityksensä sisäisillä tunnuksissa myös yhteistyöyrityksen sovellukseen. Kuvassa 8 on esitetty, miten tämän kaltainen järjestelmä voidaan toteuttaa.



KUVA 8. Integroituminen muiden yritysten kanssa

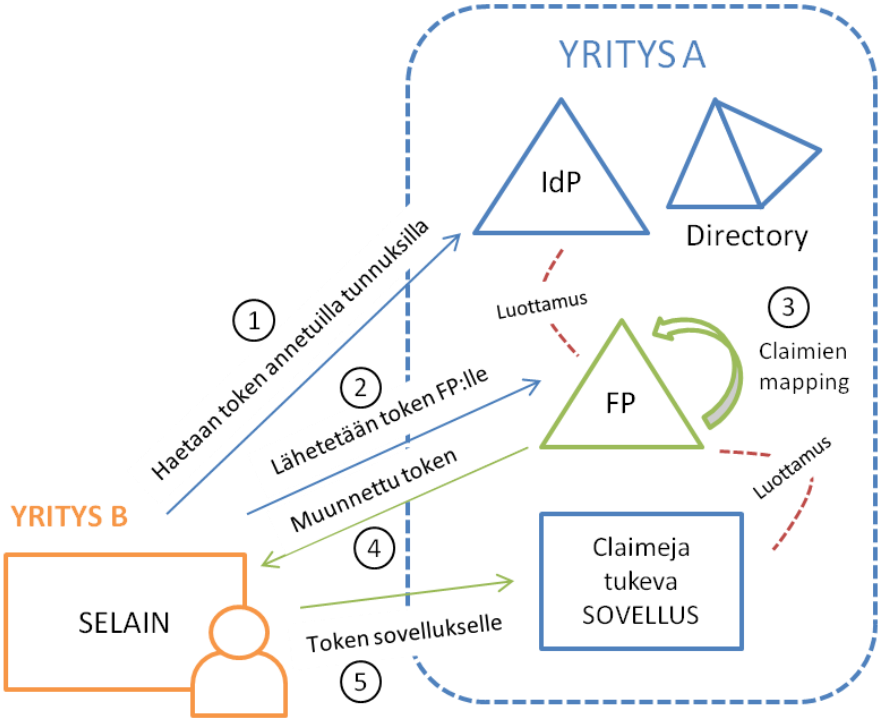
Kuvasta 8 on jätetty kuvan selkeyttämisen vuoksi ensimmäinen vaihe kokonaan pois. Käytännössä ensimmäiseksi käyttäjä ottaisi yhteyden A-yrityksen sovellukseen. Sovellus ei pysty tunnistamaan käyttäjää, joten käyttäjä ohjautuu pyytämään tokenia oman yrityksensä Identity Providerilta (IdP), kuten kuvassa 4 aikaisemmin esitettiin. IdP:n myöntämä token lähetetään tämän jälkeen A-yrityksen Federation Providerille (FP), joka muuntaa tokenin kohdesovelluksen tukemaan muotoon. Käytännössä FP käsittelee tokenin sisältämät claimit poistaen ylimääräiset ja lisäten puuttuvat tiedot. Tämän jälkeen muokattu token lähetetään takaisin selaimelle, josta se edelleen lähetetään kohdesovelluksen käytettäväksi. B-yrityksen käyttäjä pystyy näin ollen kirjautumaan oman yrityksensä tunnuksilla A-yrityksen sovellukseen.

Koko toiminta perustuu yritysten välille konfiguroituun FP:n ja IdP:n luottamukseen. Kun FP pystyy tunnistamaan saapuvasta tokenista sen alkuperän, voi se tämän tiedon perusteella tehdä claimeille mappauksen ennalta konfiguroitujen asetusten perusteella. Kuvassa 9 on esimerkki, miten mappaus käytännössä voisi toimia. Kuvassa voidaan ajatella, että Litware on yritys B ja Adatum on yritys A.

Input Conditions	Output claims
Claim issuer: Litware *Claim type*: Group, *Claim value*: Sales	*Claim issuer*: Adatum *Claim type*: Role; *Claim value*: Order Tracker
Claim issuer: Litware	*Claims issuer*: Adatum *Claim type*: Company; *Claim value*: Litware
Claim issuer: Litware *Claim type*: name	*Claims issuer*: Adatum *Claim type*: name; *Claim Value*: Copied from input

KUVA 9. Claimien muuntaminen kohdesovelluksen vaatimusten mukaiseksi (Baier ym. 2011, 112.)

Kuvan 9 kaltainen tilanne on käytännössä ideaalisin tapaus, kun puhutaan yhteisestä käyttäjähallinnasta. Monesti tulee kuitenkin vastaan tilanne, että yhteistyökumppani onkin pienempi yritys, joilla ei ole resursseja hankkia omaa Identity Provideria. Yleensä pienemmän yrityksen käyttäjille pitäisikin tarjota tunnukset oman yrityksen Identity Providerille. Kuvassa 10 on esitetty vaihtoehto, miten tämän kaltainen tilanne voitaisiin toteuttaa.



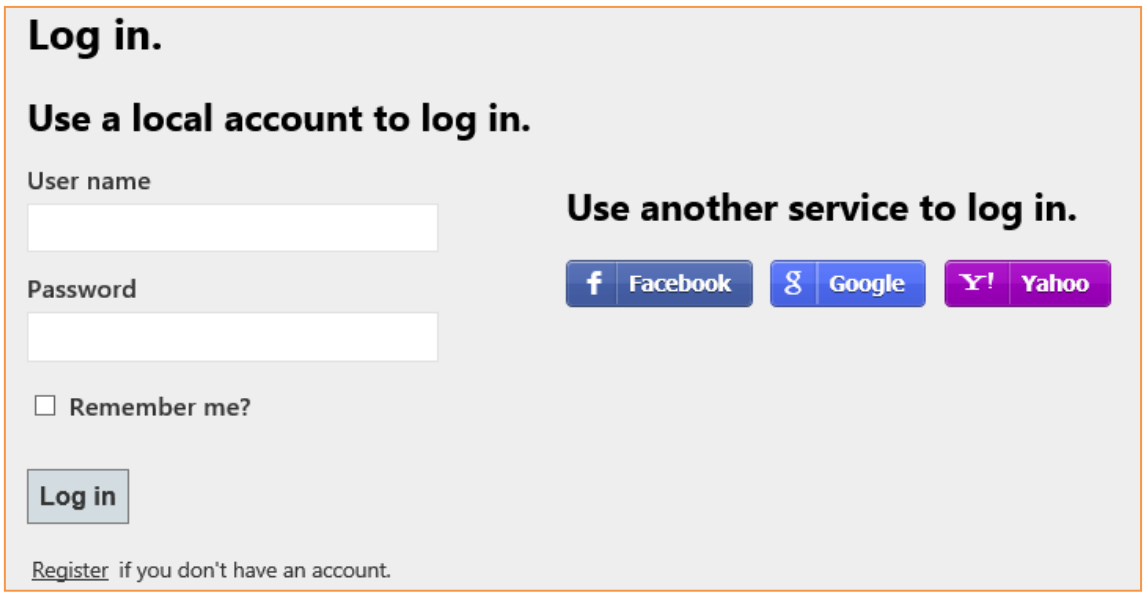
KUVA 10. Sovelluksen käyttö ilman omaan Identity Provideria

Yritys A pystyy tällaista arkkitehtuuria käyttämällä tarjoamaan sovelluksen käyttöön myös yrityksille, jotka eivät omista omaa Identity Provideria. Yrityksen A sovellukselle on edelleen konfiguroitu luottamus vain yhteen STS-komponenttiin, mutta Identity Providerin kautta hallitaan nyt myös ulkopuolisten käyttäjien tunnuksia. Tämä toteutustapa ei ole ideaalinen, koska se tuo jonkin verran lisäkustannuksia ylläpidon muodossa. Kun käyttäjille tarjotaan omat käyttäjätunnukset, täytyy olla myös ylläpitopalvelu muun muassa salasanojen resetointia ja uusien tunnusten luomista varten. Tämä kaikki on poispäin yhteisen käyttäjähallinnan periaatteista, joten tätä tulisi välttää mikäli mahdollista.

5.5 Home Realm Discovery

Kun yhteistä käyttäjähallintaa tukeva yritys on konfiguroinut luottamussuhteita usean eri yrityksen välille, tulee Federation Providerin toimintaan lisähaasteita. Federation Providerin täytyy jollakin keinolla saada selville, mihin käyttäjät kulloinkin ohjataan eli mikä taho pystyy autentikoimaan käyttäjän. Tästä toiminnosta käytetään termiä Home Realm Discovery (HRD). Tämän toiminnallisuuden toteuttamiseen on ainakin kaksi erilaista lähestymistapaa.

Ensimmäinen ja helpoin tapa ratkaista tämä haaste on pyytää käyttäjää auttamaan oikean Identity Providerin valinnassa. Käytännössä Federation Provider avaa käyttäjälle kirjautumissivun, josta käyttäjä sitten voi tunnistaa oman yrityksensä ja valita sen kirjautumispaikakseen. Huomattavaa on, että käyttäjä ei hyödy mitään, vaikka painaisi väärän yrityksen kirjautumisvaihtoehtoa, koska hänellä ei luultavasti ole tunnuksia muuhun, kuin oman yrityksen Identity Provideriin. Seuraavan kerran, kun käyttäjä tulee palveluun, samaa kysymystä ei tarvitse enää tehdä, koska tieto voidaan tallentaa selaimen välimuistiin. (Baier ym. 2011, 34.) Kuvassa 11 on esimerkki käyttäjälle näkyvästä kirjautumisikkunasta.



Kuva 11. Useamman kirjautumisvaihtoehdon mahdollistava kirjautumisikkuna

Toinen tapa ratkaista tämä haaste on lisätä whr-parametri siihen linkkiin, mitä kautta käyttäjät tulevat sovellukseen. Whr-parametrin avulla Federation Provider pystyy tunnistamaan käyttäjän Identity Providerin ja näin ollen pystyy myös suoraan ohjaamaan käyttäjän oikeaan paikkaan ilman, että käyttäjän tarvitsee itse valita käytettävää Identity Provideria. (Baier ym. 2011, 34.) Whr-parametrin sisältävä linkki syötetään seuraavassa muodossa:
https://ExampleApp.com/?WHR=IdentityProvider-URI-here

5.6 Saavutettavat hyödyt

Claimeihin pohjautuvan autentikoinnin yksi suurimmista eduista on se, että sovelluksesta saadaan rakennettua huomattavasti modulaarisempi ja samalla myös ylläpidettävämpi. Modulaarisuutta lisää se, että sovelluksen kirjautumislogiikka voidaan erottaa auktorisoinnista ja samalla koko sovelluksesta. Claims-pohjaisessa järjestelmässä käyttäjän autentikointi siirretään sovelluksen luottamalle Identity Providerille. Tämä siis mahdollistaa esimerkiksi autentikointilogiikan vaihtamisen kaksivaiheiseen tunnistautumiseen ilman, että itse ohjelmakoodiin tarvitsee tehdä lainkaan muutoksia. (Baier ym. 2011, 49.)

Claimeihin pohjautuva autentikointijärjestelmä mahdollistaa myös sovelluksen käyttämisen yli yritysten rajojen. Yritysten autentikointijärjestelmien STS-komponenttien välille voidaan luoda luottamussuhde, jolloin käyttäjä voi tunnistautua yhteistyöyrityksen sovellukseen oman yrityksenä tunnuksilla. Tämä tekee sovelluksesta käyttäjäystävällisemmän ja helpottaa käyttäjähallintaa. Ideaalitapauksessa yritysten tarvitsisi huolehtia enää oman yrityksensä käyttäjätietojen ajantasaisuudesta. (Baier ym. 2011, 49.) Käytännössä useissa yrityksissä joudutaan vielä ylläpitämään myös yrityksen ulkopuolisia käyttäjätunnuksia, koska varsinkaan pienemmillä yrityksillä ei ole tarvittavia resursseja claim-pohjaisen järjestelmän toteuttamiseen.

5.7 Järjestelmän huonot puolet

Yksi yleisimmin tunnetuista vastaväitteistä yhteiselle käyttäjähallinnalle, on tietoturvauhan kasautuminen yhteen paikkaan. Taustalla on ajatus, että kun käyttäjätiedot käsitellään sovelluskohtaisesti, hakkerin päästyä käsiksi näihin tietoihin jää tietomurron vaikutus hyvin pieneksi verrattuna yhteiseen käyttäjähallintaan. Yhteisessä käyttäjähallinnassa hakkerilla olisi tietomurron tapahtuessa pääsy kaikkiin sovelluksiin, joiden autentikointi on ulkoistettu murretulle Identity Providerille. Tämän uhkan todennäköisyys on kuitenkin nykyään katsottu niin pieneksi, että saavutettavat hyödyt yleensä ovat painoarvoltaan merkittävämpiä. (Rountree 2012.)

Huonoiksi puoliksi voitaneen laskea myös järjestelmän kehittämisestä aiheutuvat kustannukset, jotka mitä luultavimmin ovat suuremmat kuin perinteisessä autentikointimallissa. Tässä kuitenkin on tärkeä hahmottaa tarkasti, onko järjestelmä sovelluksen haltijan mielestä investoinnin arvoinen. Kuluja aiheutuu Identity Providerin kehittämisestä tai hankinnasta ja sovellukseen vaadittavista muutoksista. Vaaditaan siis mahdollisesti laitteisto, ohjelmisto ja henkilötyötunteja järjestelmän toteuttamiseen. (Rountree 2012.)

Haasteita voi tulla myös järjestelmän toteuttamisessa, koska käytettävät teknologiat ovat vielä melko uusia. Läheskään jokaisessa yrityksessä ei ole näihin teknologioihin erikoistunutta tekijää ja jopa alihankkijan löytäminen voi olla haasteellista. Uutuudesta johtuu myös löytyvän dokumentaation niukkuus. Dokumentaatioissa ei ole välttämättä kerrottu käytettyjä ratkaisuja tarpeeksi tarkasti henkilöille, joilla ei ole kokemusta tästä osa-alueesta. (Rountree 2012.)

6 AUKTORISOINTI

Claimeja voidaan hyödyntää tehokkaasti myös sovelluksen auktorisoinnin toteuttamisessa. Sovelluksissa, jotka on toteutettu .NET 4.5 -Frameworkin vanhemmilla versioilla, on usein käytetty roolipohjaista auktorisointia. Roolipohjainen lähestymistapa tuo kuitenkin useita haasteita auktorisoinnin toteuttamiseen ja etenkin ylläpidettävyyteen.

6.1 Claims-pohjainen auktorisointi ja roolipohjainen auktorisointi

Roolipohjaisessa auktorisoinnissa jokaiselle autentikoidulle käyttäjälle annetaan jokin rooli, jonka mukaan määräytyy mitä käyttäjä voi sovelluksessa tehdä. Rooli voi määräytyä esimerkiksi työtehtävän tai aseman mukaan. Henkilö voisi esimerkiksi toimia elektroniikkaliikkeen myyjänä jolloin hänelle voitaisiin antaa myyjä-rooli, joka oikeuttaisi hänet käyttämään myyjille tarkoitettuja sovelluksen ominaisuuksia. Yksi tällainen ominaisuus voisi olla uuden asiakkaan lisääminen järjestelmään. Kuvassa 12 on esitetty, miten tämänkaltainen auktorisointi voitaisiin toteuttaa Windows Identity Foundation -ohjelmistokehyksen mukaisesti.

```
public void AddCustomer(Customer customer)
{
    if (Thread.CurrentPrincipal.IsInRole("Sales"))
    {
        // add customer
    }
}
```

KUVA 12. Roolipohjainen auktorisointi (Baier 2011.)

Tämän kaltainen auktorisointi sitoo hyvin tiukasti yhteen auktorisointilogiikan sovelluksen business-logiikkaan (Baier 2011). Tästä johtuen sovelluksen ylläpidettävyys heikentyy huomattavasti, varsinkin suuremmissa sovelluksissa. Yrityksen kasvaessa tarvitaan yhä uusia rooleja ja kohta roolien määrä ei ole enää järkevästi hallittavissa. Esimerkiksi jos organisaatiomuutoksen jälkeen vain

osalle myyjistä haluttaisiin antaa asiakkaanlisäysoikeus, täytyisi luoda uusi alemman tason myyjä-rooli henkilöille, joilta tämä yksi ominaisuus halutaan estää.

Toisin kuin roolipohjaisessa auktorisoinnissa, claims-pohjaisessa auktorisoinnissa ohjelmaan määritellään kullekin toiminnallisuudelle avainparit, joiden avulla toiminnallisuuteen pääsee käsiksi. Toisin sanoen avainparien avulla kerrotaan mitä ohjelma tekee, eikä suoraan oteta kantaa kuka sen saa suorittaa. (Baier 2011.) Kuvassa 13 on toteutettu sama ominaisuus kuin kuvassa 12, mutta auktorisoinnissa käytetään claimeja.

```
public void AddCustomer(Customer customer)
{
    try
    {
        ClaimsPrincipalPermission.CheckAccess("Customer", "Add");

        // add customer
    }
    catch (SecurityException ex)
    {
        // access denied
    }
}
```

KUVA 13. Claims-pohjainen auktorisointi WIF-ohjelmistokehyksellä (Baier 2011.)

Claims-pohjaisesta auktorisoinnista voidaan siis rakentaa huomattavasti tarkempi ilman, että sovelluksen ylläpidettävyys tulee mahdottomaksi. Claimien avulla sovelluksen käyttäjille on mahdollista rajata vain ja ainoastaan se määrä oikeuksia, joita he tarvitsevat. Tämä on myös tietoturvamielessä auktorisoinnin tavoitetila.

Sovelluksen elinkaaren aikana on luontaista, että sovelluksen auktorisoinnissa tapahtuu muutoksia. Muutoksissa on yleensä kyse siitä, että päivitetään kritee-

reitä, kuka saa suorittaa jonkin sovelluksen toiminnallisuuden. Harvinaisempi tarve muutosten yhteydessä on päivittää, mitä jokin toiminnallisuus tekee. (Baier 2011.) Käytännössä auktorisointiin liittyviä muutoksia ohjelmakoodiin täytyy tehdä ainoastaan silloin, kun jokin toiminnallisuus muuttuu huomattavasti alkuperäiseen tilanteeseen verrattuna.

6.2 Auktorisointisääntöjen keskittäminen

Claims-pohjainen auktorisointi mahdollistaa auktorisointisääntöjen keskittämisen yhteen paikkaan, joka on ylläpidollissessa mielessä huomattavasti parempi vaihtoehto kuin sääntöjen ripottelu ympäri sovellusta. WIF-ohjelmistokehys sisältää ClaimsAuthorizationManager-luokan, jota periyttämällä voidaan rakentaa oma luokka, jossa käytettävät auktorisointisäännöt esitellään. Kuvassa 14 on runko, johon auktorisointisäännöt voidaan rakentaa.

```
1  public class CustomClaimsAuthorizationManager : ClaimsAuthorizationManager
2  {
3      public CustomClaimsAuthorizationManger()
4      {
5      }
6
7      public override bool CheckAccess(AuthorizationContext context)
8      {
9          return false;
10     }
11 }
```

KUVA 14. Oman auktorisointiluokan runko

Kuvassa 13 käytetty ClaimsPrincipalPermission tekee siis kutsun ClaimsAuthorizationManager-oliolle, jossa auktorisointi päätökset tehdään. Itse asiassa ClaimsAuthorizationManager-luokka sisältää CheckAccess-metodin, jonne omat auktorisointi säännöt voidaan esitellä. Tässä metodissa päätösten tekemiseen käytetään kolmea arvoa: action, resource ja principal. Metodi palauttaa sitten true- tai false -arvon riippuen käyttäjälle asetetuista oikeuksista. Kuvassa 15 on rakennettu CustomAuthorizationManager-luokka, jossa asetetaan Sales-claimin omaavalle käyttäjälle oikeus käyttää ominaisuutta, jolla lisätään järjestelmään uusi käyttäjä.

```
14  public class CustomClaimsAuthorizationManager : ClaimsAuthorizationManager
15  {
16      public CustomClaimsAuthorizationManger()
17      {
18      }
19
20      public override bool CheckAccess(AuthorizationContext context)
21      {
22          if (context.Action.First().Value == "Add" &&
23              context.Resource.First().Value == "Customer")
24              {
25                  Claim claim = context.Principal.Claims.SingleOrDefault(
26                  c => c.Type == "https://exampleapp.com/identity/claims/role")
27                  if (claim != null && claim.value == sales)
28                  {
29                      return true;
30                  }
31              }
32          return false;
33      }
34  }
```

KUVA 15. CheckAccess-metodiin lisätty auktorisointisääntö

ClaimsAuthorizationManager-luokan käyttö mahdollistaa auktorisointisääntöjen selkeän erottamisen business-logiikasta ja samalla se mahdollistaa auktorisointi sääntöjen päivittämisen ilman, että itse sovellukseen tarvitaan muutoksia. (Claims Based Authorization Using WIF 2014.)

Vaikka auktorisoinnin hallinta saadaan tällä tavalla keskitettyä yhteen paikkaan, joudutaan vielä kehittämään jokin tapa hallita tätä luokkaa. Kun sääntöjä kovakoodataan tähän luokaan, voi luokan koko kasvaa jossakin vaiheessa hallitsemattoman suureksi. Tämä tulisikin ottaa huomioon heti alkuvaiheessa ja kehittää tai hankkia työkalu, jolla sääntöjä voidaan rakentaa ja muokata helposti. Luokan koon kasvua voidaan myös rajoittaa, kun tehdään selvä suunnitelma, miten auktorisointi sovelluksessa halutaan toteuttaa. Huonosti tehty auktorisointisuunnitelma on varma tie tehdä luokasta erittäin kompleksinen.

7 YHTEENVETO

Tässä työssä oli tavoitteena selvittää, voidaanko web-sovelluksen autentikointi toteuttaa niin, että saavutettaisiin seuraavat vaatimukset:

- Käyttäjän ei tarvitse erikseen kirjautua sovellukseen kun työskennellään yrityksen omassa intranetissä.
- Intranetin ulkopuolelta yrityksen käyttäjä voi kirjautua sovellukseen työpaikan Active Directory -tunnuksilla.
- Järjestelmässä pitäisi olla mahdollisuus integroitua ulkopuolisiin identiteetin tarjoajiin.

Alkuselvitysten perusteella työ rajattiin koskemaan Claims-pohjaista autentikointia, joka on ollut paljon esillä .NET 4.5 -ohjelmistokehyksen julkaisemisen jälkeen. Tämä johtunee osittain siitä, että Microsoft otti ohjelmistokehykseen mukaan ohjelmistokomponentteja, jotka antavat tuen Claims-pohjaisille sovelluksille ja järjestelmille.

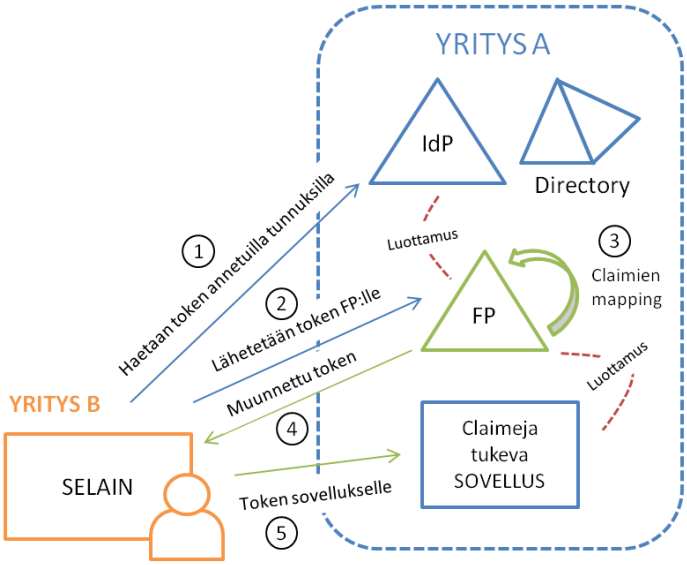
Työssä käytiin läpi Claims-pohjaisen järjestelmän toimintaa ja sitä kautta pyrittiin löytämään ratkaisut, jotka mahdollistaisivat alkuvaatimukset. Työn aikana kerätyn tiedon perusteella kaikki alkuvaatimukset ovat toteutettavissa Claims-pohjaisella arkkitehtuurilla.

Alkuvaatimukset täyttävän Claims-pohjaisen järjestelmän toteuttaminen vaatii sovellukseen tehtäviä muutoksia, mutta lisäksi tarvitaan Identity Provider hoitamaan käyttäjän autentikointi. WIF-ohjelmistokehys mahdollistaa oman Identity Providerin kehittämisen, mutta työmäärä saavutettavaan hyötyyn nähden ei ole järkevä Tämän vuoksi suositeltavaa onkin käyttää Identity Providerina jotakin valmista ratkaisua, joita on tarjolla useita.

IdP:n valintaa tehtäessä tulee ensin kartoittaa, millaiseen ympäristöön se aiotaan integroida ja millaisia vaatimuksia sille asetetaan. Suunnitteluvaiheessa olisi hyvä löytää vastaukset ainakin seuraaviin kysymyksiin: Onko vaatimuksia

yhteiselle käyttäjähallinnalle? Tarvitseeko IdP:n toimia myös yrityksen ulkopuolisille käyttäjille? Täytyykö löytyä tuki sosiaalisten palveluiden IdP:lle?

Alkuvaatimusten mukainen järjestelmä voitaisiin toteuttaa esimerkiksi aikaisemmin työssä esitellyn arkkitehtuurin mukaisesti. Kuvassa 16 on esitelty tämä arkkitehtuuri.



KUVA 16. Alkuvaatimukset täyttävä arkkitehtuuri

Yrityksen A käyttäjille voidaan antaa tunnukset Identity Providerin luottamasta Active Directory -tietokannasta, jolloin kirjautuminen sovellukseen onnistuu samoilla tunnuksilla kuin yrityksen muihinkin järjestelmiin. Intranetistä kirjauduttaessa voidaan käyttää esimerkiksi alakohdassa 5.5 Home Realm Discovery esiteltyä whr-parametrin sisältämää linkkiä, jolloin kirjautumista ei välttämättä tarvita päivän ensimmäisen koneelle kirjautumisen jälkeen. Kuvassa 16 on myös Federation Provider (FP), jota voidaan käyttää ulkopuolisiin järjestelmiin integroitumiseen. Työn aikataulun vuoksi tätä kokoonpanoa ei ehditty testata, mutta työn aikana kerätyn tiedon perusteella tämän kaltaista arkkitehtuuria suositellaan muun muassa silloin, kun käytetään Microsoftin tarjoamia palveluita.

Työn tekemiseen tuli työn puolessa välissä kuukauden tauko, jonka jälkeen työn painopisteitä muutettiin hieman alkuperäisestä suunnitelmasta. Työhön otettiin tällöin mukaan myös Claims-pohjainen auktorisointi.

Claims-pohjaisen auktorisoinnin (CBA) etuja pyrittiin selvittämään vertaamalla sitä paremmin tunnettuun roolipohjaiseen auktorisointiin (RBA). Lopputuloksena voidaan sanoa, että Claims-pohjaisen auktorisoinnin avulla saavutetaan entistä modulaarisempi sovellus. CBA:ssa päätökset tehdään claimeihin perustuvalla logiikalla, jolloin auktorisoinnista voidaan rakentaa tarkempi ja sovelluksesta ylläpidettävämpi. Toteutuksessa suurimpana erona on, että CBA:ssa jokaiselle toiminnolle annetaan avainparit, joiden avulla kuvataan, mitä toiminnallisuus tekee, eikä oteta suoraan kantaa, kuka sen saa suorittaa, kuten RBA:ssa.

Auktorisointipäätösten tekeminen voidaan kerätä yhteen paikkaan esimerkiksi käyttämällä WIF-ohjelmistokehyksessä olevaa ClaimsAuthorizationManager-luokkaa. Tästäkin luokasta voi kasvaa hallitsemattoman suuri, mikäli selvää auktorisointi suunnitelmaa ei ole olemassa. Tässä työssä ei kuitenkaan perehdytty auktorisointisääntöjen hallintaan lainkaan. Mikäli tämän kaltainen auktorisointi otetaan käyttöön, olisi syytä selvittää miten näiden sääntöjen muodostaminen ja hallinta on ylläpidollisessa mielessä järkevintä toteuttaa.

Työ oli aihealueeltaan melko haastava, mutta erittäin mielenkiintoinen. Haastavaksi aiheen teki saatavilla olevan tiedon hajanaisuus. Myöskään selkeitä ratkaisuja ei löytynyt joita olisi suoraan voinut verrata alkuvaatimuksiin. Vaikka työssä ei saatu annettua valmiiksi testattua toteutusmallia, onnistuttiin siinä kuitenkin poimimaan useita tärkeitä seikkoja, jotka täytyy huomioida claims-pohjaisen järjestelmän suunnitteluvaiheessa.

Claims-pohjainen autentikointi ja auktorisointi ovat hyvin varteenotettavia vaihtoehtoja, kun näitä osa-alueita halutaan uudistaa sovelluksesta. Ne ovat uusimpia tekniikoita ja niissä on pyritty ottamaan huomioon nykyaikaisten sovellusten vaatimukset ja lisäksi parantamaan nykyisin käytössä olevien järjestelmien huonoja puolia.

LÄHTEET

Baier, Dominick – Bertocci, Vittorio – Brown, Keith – Densmore, Scott – Pace, Eugenio – Woloski, Matias 2011. A Guide to Claims-Based Identity and Access Control. Authentication and Authorization for Services and the Web, Second Edition. Microsoft Corporation.

Baier, Dominick 2011. What I like about WIF's Claims Based Authorization. Saatavissa: http://leastprivilege.com/2011/04/30/what-i-like-about-wifs-claims-based-authorization/. Hakupäivä 22.4.2014.

Bertocci, Vittorio 2008. An Identity Provider and its STS: preliminary considerations. Saatavissa: http://www.cloudidentity.com/blog/2008/11/26/an-identity-provider-and-its-sts-preliminary-considerations/. Hakupäivä 18.4.2014.

Chadwick, Jess – Snyder, Todd – Panda, Hrusikesh 2012. Programming ASP.NET MVC 4. O'Reilly Media.

Chappell, David 2011. Claims-based identity for windows. Technologies and scenarios. Saatavissa: http://www.davidchappell.com/writing/white_papers/Claims-Based_Identity_for_Windows_v3.0--Chappell.docx. Hakupäivä 6.2.2014.

Claims Based Authorization Using WIF. 2014. Microsoft Developer Network. Saatavissa: http://msdn.microsoft.com/en-us/library/hh545448(v=vs.110).aspx Hakupäivä 21.4.2014.

Kissell, Joe 2014. Take Control of Your Passwords. TidBITS Publishing. Luettavissa Safari Books Online –palvelussa, vaatii käyttäjätunnukset.

Lakshmiraghavan, Badrinarayanan 2013. Pro ASP.NET Web API Security. Securing ASP.NET Web API. Apress. Luettavissa Safari Books Online – palvelussa, vaatii käyttäjätunnukset.

MacDonald, Matthew 2012. Beginning ASP.NET 4.5 in C#. Apress.

Reichl, Dominick 2014. KeePass Password Safe. Saatavissa: http://www.keepass.info/. Hakupäivä 28.4.2014.

Rhodes-Ousley, Mark 2013. Information Security. The Complete Reference, Second Edition. McGraw-Hill Companies. Luettavissa Safari Books Online – palvelussa, vaatii käyttäjätunnukset.

Rountree, Derrick 2012. Federated Identity Primer. Syngress. Luettavissa Safari Books Online –palvelussa, vaatii käyttäjätunnukset.

What's New in Windows Identity Foundation 4.5. 2014. Microsoft Developer Network. Saatavissa: http://msdn.microsoft.com/en-us/library/hh873305(v=vs.110).aspx. Hakupäivä 27.4.2014.

Windows Identity Foundation 4.5 Overview. 2014. Microsoft Developer Network. Saatavissa: http://msdn.microsoft.com/en-us/library/hh291066(v=vs.110).aspx. Hakupäivä 25.4.2014.