Tapio Kukkonen

WWW-sovelluspalveluiden tietoturvallinen pääsynvalvonta

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

24.4.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Tapio Kukkonen WWW-sovelluspalveluiden tietoturvallinen pääsynvalvonta 41 sivua 24.4.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Ilpo Kuivanen Security Manager Antti Ropponen

Tämän insinöörityön tavoitteena oli selvittää tietoturvallisen pääsynvalvonnan toteuttaminen REST-arkkitehtuurityyliä noudattaviin WWW-sovelluspalveluihin. Tarkastelun kohteeksi otettiin verkkosovellusten pääsynvalvonnan toteuttamisen apuna käytettäviä teknisiä standardeja, joita ovat: SAML, OpenID, OAuth2 ja JWT. Näistä standardeista oli tarkoitus löytää soveltuvimmat. Lisäksi selvitettiin tietoturvan roolia REST-arkkitehtuurityyliä noudattavan WWW-sovelluspalvelun arkkitehtuurissa.

Työ aloitettiin selvittämällä REST-arkkitehtuurityylin käsite. Tämän jälkeen selvitettiin WWW-sovelluspalveluiden vakavimmat tyypilliset haavoittuvuudet ja pääsynvalvonnan käsite. Havaittiin, että WWW-sovelluspalveluiden haavoittuvuudet eivät juuri poikenneet tavanomaisten verkkopalveluiden haavoittuvuuksista. Todettujen haavoittuvuuksien ja pääsynvalvonnan käsitteen perusteella tutkittiin teknisten standardien soveltuvuutta käytettäväksi REST-arkkitehtuurityyliä noudattavien WWW-sovelluspalveluiden pääsynvalvonnassa. Tekniset standardit ilmenivät olevan suunniteltu eri tarkoituksiin ja niiden sovellettavuuden WWW-sovelluspalveluiden pääsynvalvonnassa todettiin vaihtelevan. Pohdinnat osoittivat OAuth2-standardin tarjoavan neljästä standardista laajimmat ominaisuudet pääsynvalvonnan toteuttamiseen REST-arkkitehtuurityyliä noudattavissa WWW-sovelluspalveluissa.

Lopuksi esitettiin REST-arkkitehtuurityyliä noudattavan WWW-sovelluspalvelun esimerkkiarkkitehtuuri ja pohdittiin tietoturvan roolia siinä. Tietoturvan havaittiin koskevan REST-arkkitehtuurityyliä noudattavan WWW-sovelluspalvelun arkkitehtuurissa sen jokaista komponenttia. Kuitenkin havaittiin, että arkkitehtuurissa, jossa API-kerros on eriytetty sovelluskerroksesta, on mahdollista keskittää joitain tietoturvatoimintoja API-kerrokseen.

Työssä tehdyt havainnot voivat toimia apuna REST-arkkitehtuurityyliä noudattavan WWW-sovelluspalvelun tietoturvan suunnittelussa. Osa havainnoista on myös sovellettavissa tavanomaisten verkkosovellusten pääsynvalvonnan tai tietoturvan suunnittelussa.

Avainsanat	tietoturva, pääsynvalvonta, REST, WWW-sovelluspalvelu

Author(s) Title Number of Pages Date	Tapio Kukkonen Secure Access Control of Web Services 41 pages 24 April 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Ilpo Kuivanen, Senior Lecturer Antti Ropponen, Security Manager

The object of this thesis was to find out a secure way to implement access control into REST architectural style web services. Four technical standards designed to aid in the process were considered: SAML, OpenID, OAuth2 and JWT. The purpose was to find the most suitable ones. Additionally, the role of information security in the architecture of a REST architectural style web service was examined.

The thesis was started with clarifying the concept of REST architectural style. After that the most serious vulnerabilities of web services and the concept of access control were examined. Web services were noted to be subject to most of the same vulnerabilities as ordinary web applications. Using the examined vulnerabilities and the concept of access control as a base, the applicability of the chosen four standards to aid in the implementation of access control for REST architectural style web services was examined. The technical standards appeared to be designed for different kinds of purposes and their applicability to implementation of access control for REST architectural style web services was noted to vary. The considerations indicated that OAuth2 standard offers the most features in the implementation of access control for REST architectural style web services.

Finally, an example architecture for a REST architectural style web service was presented and the role of information security in it was examined. Information security appeared to consider every component of the architecture. Nevertheless it was noted that in an architecture where the API layer is separated from the application layer, it is possible to centralize some of the security functionality into the API layer.

The observations made in the study can aid in designing the information security of a REST architectural style web service. Some of the observations can also be applied while designing the access control or information security of a regular web application.

Keywords	information security, access control, REST, web service

Sisällys

Sanasto

Base64-koodaus
: Koodaus, jossa binääridata muunnetaan merkkijonoksi, jossa on käytetty rajoitettua merkistöä.

horisontaalinen skaalaaminen
: Järjestelmän kapasiteetin lisääminen lisäämällä siihen lisää yksiköitä, esimerkiksi palvelimia.

HTTP
: (HyperText Transfer Protocol) Sovellustason tiedonsiirtoprotokolla.

JSON
: (JavaScript Object Notation) JavaScriptistä peräisin oleva, nimestään huolimatta alustariippumaton, rakenteellinen kuvauskieli.

Phising
: Tietoturvahyökkäys, jossa loppukäyttäjä yritetään harhauttaa antamaan tietoja.

SOAP
: (Simple Object Access Protocol) XML-pohjainen protokolla, jonka avulla voidaan toteuttaa rajapinta WWW-sovelluspalveluun.

TLS
: (Transport Layer Security) OSI-mallin kerroksilla 5 ja 6 toimiva salausprotokolla. Siihen viitataan usein myös sen vanhalla nimellä SSL.

WWW-sovelluspalvelu
: (engl. web service) Palvelimella toimiva ohjelma, joka tarjoaa rajapinnan käytettäväksi verkon yli.

XML
: (eXtensible Markup Language) W3C:n standardi rakenteellinen kuvauskieli, jolla voidaan jäsentää tietoa selkeästi.

1 Johdanto

Tietoturva voidaan määritellä tiedon luottamuksellisuutena, eheytenä ja saatavuutena. Luottamuksellisuus tarkoittaa tiedon suojaamista valtuuttamattomalta pääsyltä. Eheys tarkoittaa tiedon suojaamista valtuuttamattomalta käsittelyltä ja tahattomalta tuhoutumiselta. Saatavuus tarkoittaa tiedon saatavuuden turvaamista valtuutetuille osapuolille. [37.] Tietoturva on yhä merkityksellisempi asia organisaatioille ja se on ollut usein esillä mediassa, kun jonkin organisaation tietoturva on vaarantunut. Tietoturva-ala on Suomessa vahva ja sen odotetaan kasvavan niin Suomessa kuin kansainvälisestikin. Tietoturva on laaja käsite, eikä se liity pelkästään tietotekniikkaan, vaan käsittää myös esimerkiksi fyysisen tietoturvan.

Julkisten WWW-sovelluspalveluiden määrä kasvaa runsaasti. Yksi syy tähän on se, että asiakkailla on tarve päästä käsiksi verkkopalveluihin useilla erilaisilla päätelaitteilla. REST-arkkitehtuurityylin soveltaminen on tällä hetkellä suosituin tapa toteuttaa WWW-sovelluspalvelun rajapinta. Tietoturva on WWW-sovelluspalveluiden kehittäjille haasteellinen ongelma, joka tulee ottaa huomioon monessa yhteydessä. Kehittäjillä voi olla tarve rajata pääsyä tiettyihin WWW-sovelluspalvelun osiin, kuten hallintatoimintoihin tai käyttäjien tietoihin. Toisaalta WWW-sovelluspalvelut ovat samalla tavalla haavoittuvaisia hyökkäyksille niin kuin tavanomaiset verkkosovelluksetkin.

Yhä enemmän ja yhä arkaluontoisempaa tietoa tallennetaan digitaalisesti. Samaan aikaan käyttäjät vaativat parempaa käyttökokemusta myös tietoturvakriittisissä sovelluksissa – moneen palveluun voi kirjautua käyttämällä toisen palvelun tunnuksia ja kirjautumisen voi usein jättää voimaan selaimen sulkemisen jälkeenkin. Toimivan ja tietoturvallisen pääsynvalvonnan toteuttaminen niin, että loppukäyttäjän käyttökokemus pysyy hyvänä, on WWW-sovelluspalveluiden ja tavallisten verkkopalveluiden kehittäjille haasteellista. Haasteen ratkaisussa voidaan soveltaa erilaisia teknisiä standardeja, jotka on suunniteltu avustamaan verkkosovellusten pääsynvalvonnan toteuttamisessa.

Tässä työssä perehdytään lyhyesti REST-arkkitehtuurityyliin, selvitetään oleellisimmat WWW-sovelluspalveluiden tyypilliset haavoittuvuudet, selitetään pääsynvalvonnan käsite sekä tutkitaan neljää erilaista pääsynvalvontaan liittyvää teknistä standardia ja arvioidaan niiden soveltuvuutta avustaa WWW-sovelluspalvelun pääsynvalvonnan ratkaisemisessa. Lisäksi esitetään REST-arkkitehtuurityyppiä noudattavan WWW-

sovelluspalvelun esimerkkiarkkitehtuuri ja pohditaan sen eri komponenttien ja toimintojen roolia tietoturvan näkökulmasta.

2 REST-arkkitehtuurityyli WWW-sovelluspalveluissa

Nykyään organisaatioilla on tarve tarjota yhä useampia tapoja käyttää niiden tarjoamia sähköisiä palveluita. Sähköisiin palveluihin on tarve päästä käsiksi käyttäen useita erilaisia asiakassovelluksia, kuten mobiili-, työpöytä- ja verkkosovelluksia. Näiden erilaisten asiakassovelluksien määrä kasvaa jatkuvasti uusien teknologioiden syntyessä ja niiden kasvattaessa suosiotaan – tulevaisuudessa esimerkiksi autotkin voivat hakea WWW-sovelluspalveluista ajankohtaista tietoa tiestöstä. REST-arkkitehtuurityyliä sovelletaan usein WWW-sovelluspalvelun kehittämisessä usean, toisistaan riippumattoman asiakassovelluksen käyttöön.

Lyhenne REST tulee englanninkielisistä sanoista Representational State Transfer. Se on arkkitehtuurityyli hajautetuille hypermediajärjestelmille. Sen esitteli Roy Fielding vuonna 2000 tutkielmassaan. REST-arkkitehtuurityyli asettaa joukon vaatimuksia järjestelmälle, jossa sitä sovelletaan. [1.]

REST-arkkitehtuurityyliä sovelletaan runsaasti WWW-sovelluspalveluiden kehityksessä. REST ei ole standardi, eikä esimerkiksi W3C:llä ole suosituksia sitä koskien. Tästä syystä sen soveltaminen WWW-sovelluspalveluiden kehityksessä on erilaista toteutuksien välillä. World Wide Web -palvelun voidaan katsoa noudattavan REST-arkkitehtuurityyliä. [2.]

REST-arkkitehtuurityyliä pidetään yleisesti SOAP-protokollan kilpailijana WWW-sovelluspalveluiden toteutusteknologiana, vaikka ne eivät ole suoraan verrattavissa, sillä SOAP on protokolla ja REST on arkkitehtuurityyli. Siitä huolimatta REST-arkkitehtuurityylin mukainen WWW-sovelluspalveluiden toteutus kasvattaa suosiotaan verrattuna SOAP-protokollaan [18].

Useimmissa WWW-sovelluspalveluiden kehitystä REST-arkkitehtuurityylin mukaisesti koskevissa artikkeleissa sen asettamina vaatimuksina on mainittu resurssipohjaisuus ja tilattomuus.

2.1 Resurssipohjaisuus

Resurssi on tärkeä tiedon abstraktio REST-arkkitehtuurityylissä. Mikä tahansa informaatio voi olla resurssi. Informaatio voi koskea yksittäistä kohdetta tai joukkoa kohteita. Resurssi voi siis olla esimerkiksi yksittäinen tekstiartikkeli tai joukko tekstiartikkeleita verkkosivustolla. REST-arkkitehtuurityylissä resurssi voi olla luonteeltaan staattinen tai dynaaminen, ainoastaan resurssien semantiikan tulee olla staattinen. Yksittäisellä resurssilla voi olla useita eri esitysmuotoja. Esimerkiksi resurssi voidaan esittää verkkosivuna tai monella eri kuvauskielellä, kuten XML- tai JSON-kuvauskielillä. Tämä resurssin abstrakti määritelmä mahdollistaa Web-arkkitehtuurin avainominaisuuksia. [1, kohta 5.2.1.1.]

Vaatimusta sovelletaan WWW-sovelluspalveluiden rajapinnoissa pääasiassa niin, että URL-osoitteet on muodostettu tietyllä tavalla. Osoitteen viimeisenä osana on yleensä käsiteltävä resurssi. Tätä resurssia käsitellään standardien HTTP-metodien avulla. [3.]

Taulukko 1. Standardit HTTP-metodit.

HTTP-metodi	**selite**
GET	hakee kohteena olevan resurssin tai joukon resursseja
PUT	lisää tai muokkaa resurssia, riippuen siitä, että onko URL-osoitteessa annetulla tunnisteella olevaa resurssia vielä olemassa
POST	lisää uuden resurssin
DELETE	poistaa URL-osoitteessa annetun resurssin

Taulukossa 1 on selitetty lyhyesti tärkeimmät standardit HTTP-metodit. Tavanomaisessa, REST-arkkitehtuurityyliä noudattamattomassa, WWW-sovelluspalvelun rajapinnassa ei tyypillisesti käytetä hyödyksi kaikkia HTTP-metodeita. Sen sijaan pyynnöissä käytetään yleensä vain GET- ja POST-metodeita.

```
http://example.com/api/getcustomer?id=123
http://example.com/api/deletecustomer?id=124
```

Koodiesimerkki 1. REST-arkkitehtuurityyliä noudattamattomia URL-osoitteita.

Koodiesimerkissä 1 on esitetty tyypillisen WWW-sovelluspalvelun URL-osoitteita. Ensimmäisellä rivillä haetaan asiakas tietyllä tunnistenumerolla. Toisella rivillä vastaavasti poistetaan asiakas tunnistenumerolla. Molemmissa pyynnöissä tyypillisesti käytettäisiin HTTP GET -metodia.

```
GET             http://example.com/api/customers/123
DELETE          http://example.com/api/customers/124
```

Koodiesimerkki 2. REST-arkkitehtuurityyliä noudattavia URL-osoitteita.

Koodiesimerkissä 2 on esitetty vastaavat REST-arkkitehtuurityylin mukaiset pyynnöt. Näissä pyynnöissä hyödynnetään tarkoituksenmukaista HTTP-metodia. REST-arkkitehtuurityylin mukainen HTTP-pyyntöjen toteutus mahdollistaa kommunikaation eri osapuolten, kuten kuormanjakajien, välityspalvelimien ja välimuistien paremman toiminnan sekä helpottaa rajapinnan käyttöä sen johdonmukaisuuden vuoksi [3].

2.2 Tilattomuus

Tilattomuus-vaatimuksen mukaan palvelimella ei säilötä ollenkaan sovelluksen tilaa. Kaikki palvelimen tarvitsevat tiedot tulisi toimittaa jokaisessa asiakkaan pyynnössä. Tämä vaatimus helpottaa mm. sovellusten horisontaalista skaalautumista, koska palvelimen ei tarvitse huolehtia pyyntöjen välisestä tilasta ollenkaan. Huonona puolena ratkaisussa on se, että kun jokaisessa pyynnössä toimitetaan tarvittavat kontekstitiedot, aiheuttaa tämä ylimääräistä verkkoliikennettä. [1, kohta 5.1.3.]

Jos palvelimella ylläpidetään istuntoa, ei WWW-sovelluspalvelu ole silloin tilaton [4]. Tällöin tunnistamiseen liittyvät tiedot tulisi lähettää jokaisen pyynnön mukana. Käytännössä kuitenkin istuntoja käytetään usein hyväksi pääsynvalvonnassa, koska rajoitetun ajan voimassaolevien istuntoavaimien lähettäminen pyynnöissä on tietoturvallisempaa kuin lähettää käyttäjän pysyvät tunnistustiedot jokaisessa pyynnössä, ja usein istuntoon on tarve tallentaa tietoa. Yksittäisen avaimen käyttöoikeuksia voidaan myös rajoittaa hienojakoisemmin.

3 Tietoturva WWW-sovelluspalvelun näkökulmasta

Tietoturva koskee järjestelmässä useita sen osia, eikä se rajoitu pelkästään tietotekniikkaan. WWW-sovelluspalveluiden tietoturva tulisi niin ikään ottaa huomioon laaja-alaisesti. Tässä työssä keskitytään tietoturvan osa-alueista haavoittuvuuksien hyväksikäytön torjuntaan ja pääsynvalvontaan REST-arkkitehtuurityyliä noudattavien WWW-sovelluspalveluiden näkökulmasta.

3.1 Haavoittuvuudet

Julkiset, REST-arkkitehtuurityyliä noudattavat WWW-sovelluspalvelut omaavat pääpiirteittäin samalla tavalla tavanomaisten, selaimella käytettävien verkkosovellusten haavoittuvuuksia. Näitä hyväksikäyttäen pahantahtoinen osapuoli voi yrittää hyökätä sovellukseen. Pahantahtoisen osapuolen tavoitteena voi olla esimerkiksi päästä käsiksi yksityiseen tietoon, muokata tietoa tai vaikuttaa palvelun saatavuuteen.

WWW-sovelluspalvelun kehittäjä on siinä mielessä eri asemassa kuin tavanomaisen, selaimella käytettävän verkkosovelluksen, että WWW-sovelluspalvelun kehittäjä voi asettaa ehtoja asiakassovellukselle ja määrittää käytettävän rajapinnan vapaammin. Tavanomaisen verkkopalvelun kehittäjä noudattaa selainten asettamia ehtoja. Tämä on sekä haaste että mahdollisuus WWW-sovelluspalvelun kehittäjälle. Selaimissa on paljon tietoturvaominaisuuksia, jotka vaikeuttavat huonojen tietoturvaratkaisujen toteuttamista. Koska tietoturva on kuitenkin usein tasapainoilua tietoturvallisuuden ja käyttökokemuksen välillä, voi WWW-sovelluspalvelun rajapinnalle ja asiakasohjelmalle asettaa ehtoja, jotka tekevät WWW-sovelluspalvelusta hyvin tietoturvallisen. Toisaalta huonosti toteutettu WWW-sovelluspalvelu voi olla tietoturvaltaan erittäin heikko.

3.1.1 OWASP Top 10 -lista

OWASP (Open Web Application Security Project) on erilaisten organisaatioiden kansainvälinen hanke, joka pyrkii luomaan vapaasti saatavia resursseja, kuten tietoa ja työkaluohjelmia verkkopalveluiden paremman tietoturvan kehittämisen tueksi. OWASP-hankkeen ylläpitämä OWASP Top 10 -lista listaa kymmenen vakavinta verkkosovellusten tyypillistä haavoittuvuutta. Lista on tarkoitettu auttamaan verkkosovellusten kehittäjiä tiedostamaan nämä haavoittuvuudet ja auttamaan niiden hyväksikäytön estämisessä.

Taulukko 2. OWASP Top 10 haavoittuvuudet ja niiden mahdollinen uhka WWW-sovelluspalveluille [29].

haavoittuvuus	**kuvaus**	**uhka WWW-sovelluspal-veluille?**
1. injektio	Kun tietoa lähetetään sovellukselle ja tieto liitetään osaksi komentoa tai kyselyä, voi hyökkääjä yrittää aiheuttaa tiedon tulkinnan komentoina tai muina loogisina rakenteina.	kyllä
2. viallinen todennus tai istunnonhallinta	Sovelluksen vialliset todennus- ja/tai istunnonhal-lintatoiminnot voivat paljastaa hyökkääjälle mm. käyttäjien kirjautumistietoja tai käyttöoikeustietuei-ta.	kyllä
3. cross-site scripting (XSS)	Sovellus lähettää selaimelle tietoa, joka voi sisäl-tää hyökkääjän haitallista HTML- tai JavaScript-koodia.	osittain
4. turvattomat objekti-viittaukset	Kun sovellus viittaa objektiin tai resurssiin esimer-kiksi URL-osoitteessa, on hyökkääjän mahdollista manipuloida tätä tietoa. Jos tätä tietoa ei varmen-neta, voi hyökkääjä päästä käsiksi hänelle kuulu-mattomaan tietoon.	kyllä
5. virheelliset tietoturva-asetukset	Hyvä tietoturva vaatii oikeanlaisen konfiguraation. Kaikki palvelimella olevat sovellukset tulee olla oikein asennettu ja konfiguroitu.	kyllä
6. arkaluontoisen tiedon paljastuminen	Arkaluontoisten tietojen kanssa, kuten luottokortti-en numeroiden, tulisi olla erittäin huolellinen. Jos tällaista tietoa säilötään tai siirretään, tulisi se sala-ta.	kyllä
7. toimintojen pääsyn-valvonnan puuttuminen	Sovellukset saattavat piilottaa toiminnot, joihin ei ole oikeuksia. Sen lisäksi tulisi näiden toimintojen suorittaminen estää, kun toimintoa yritetään suorit-taa.	osittain
8. cross-site request forgery (CSRF)	CSRF-hyökkäys aiheuttaa käyttäjän selaimen lähettämään HTTP-pyynnön suojaamattomalle verkkosovellukselle, sisältäen kirjautumistietoja tai istuntoavaimia.	kyllä
9. haavoittuvuuksia sisältävien komponent-tien käyttäminen	Haavoittuvuuksia sisältävät komponentit, kuten kirjastot, sovelluskehykset tai muut ohjelmistomo-duulit voivat aiheuttaa monenlaisia uhkia.	kyllä
10. validoimattomat uudelleenohjaukset ja välitykset	Sovellukset usein uudelleenohjaavat käyttäjiä toi-sille sivuille. Ilman uudelleenohjauksien validointia, voi hyökkääjä ohjata käyttäjät pahantahtoisille sivuille.	osittain

Taulukossa 2 on kuvattu OWASP-hankkeen ylläpitämä lista kymmenestä vakavimmasta verkkosovellusten haavoittuvuudesta. Taulukossa on lisäksi kuvattu, koskevatko nämä haavoittuvuudet myös WWW-sovelluspalveluita täysin sellaisenaan vai vain osittain.

OWASP luokittelee injektion verkkosovellusten vakavimmaksi haavoittuvuudeksi. Usein sovelluksilla on tarve muodostaa komentoja tai kyselyitä dynaamisesti riippuen käyttäjän lähettämästä pyynnöstä. Hyökkääjä voi muodostaa lähetettävän tiedon niin, että sen tulkitseva komponentti tulkitsee sen tai sen osan komentona. Pahimmillaan käyttäjän syöttämää tietoa voidaan käyttää käyttöjärjestelmän komentotulkissa, jolloin hyökkääjä voi suorittaa, riippuen verkkosovellusta ajavan käyttäjätilin käyttöoikeuksista, käytännössä mitä tahansa komentoja. Injektioiden estäminen on helppoa: joko ei muodosteta dynaamisia komentoja tai kyselyitä, tai sanitoidaan dynaamisen komennon tai kyselyn muodostavat tiedot niin, että ne eivät aiheuta harmia [30]. Jälkimmäinen vaihtoehto on haasteellisempaa, sillä tapoja injektoida haitallisia merkkejä syötteeseen tai ohittaa sanitointi on monia.

Cross site scripting -haavoittuvuus ei täysin koske WWW-sovelluspalveluita, sillä se hyödyntää pääasiassa selainten toimintoja. Paljon WWW-sovelluspalveluista noudettua tietoa kuitenkin käytetään selaimissa, joten olisi suotavaa, että tämä tieto ei sisällä haitallista HTML- tai JavaScript-koodia. Mikäli haitallisten merkkien suodattamista ei suoriteta, tulisi varmistaa, että asiakassovellusten kehittäjät ovat tietoisia tästä ja voivat vastata merkkien suodattamisesta.

Koska WWW-sovelluspalveluissa ei ole käyttöliittymää, ei toimintojen pääsynvalvonnan puutuminen -haavoittuvuus täysin koske niitä. On hyvä kuitenkin huomioida, että jos WWW-sovelluspalveluun on useampi rajapinta, täytyy kaikkien näiden rajapintojen toteuttaa pääsynvalvonta tai vaihtoehtoisesti toteuttaa pääsynvalvonta komponentissa, jota nämä rajapinnat käyttävät yhteisesti.

Cross-site request forgery -haavoittuvuus tarkoittaa asiakassovelluksen, kuten selaimen, manipuloimista lähettämään HTTP-pyyntö kolmannen osapuolen haavoittuvaan palveluun. Tämä voidaan toteuttaa esimerkiksi lisäämällä HTML-sivulle ”img”-tagi, jossa on haavoittuvan palvelun URL-osoite. Tämä aiheuttaa HTTP GET -pyynnön lähettämisen kyseiseen URL-osoitteeseen. Vaarallista tässä on se, että asiakassovellukset voivat sisällyttää kolmannelle osapuolelle lähetetyissä pyynnöissä todennustietoja, ku-

ten evästeitä. Tätä hyödyntäen hyökkääjä voi manipuloida asiakassovelluksen lähettämään valtuutusta vaativia pyyntöjä loppukäyttäjän todennustiedoilla.

```
http://example.com/verkkopankki/tilisiirto?tilille=321
```

Koodiesimerkki 3. Kuvitteellisen haavoittuvan verkkopankin URL-osoite.

Koodiesimerkissä 3 on esitetty esimerkki kuvitteellisen haavoittuvan verkkopankin URL-osoitteesta, jota käytetään tilisiirron suorittamiseen ja voidaan hyödyntää CSRF-hyökkäyksessä. CSRF-hyökkäykset voidaan estää käyttämällä HTTP POST -pyyntöjä silloin, kun tehdään minkäänlaisia muutoksia sovelluksen tilaan tai tietoihin, sillä asiakassovellusten manipulointi lähettämään sen tyyppisiä pyyntöjä on yleensä hankalampaa, joskin ei mahdotonta. Lisäksi HTTP POST -tyyppiset pyynnöt tulisi todentaa soveltaen esimerkiksi Synchronizer Token Pattern -mallia, joka perustuu siihen, että asiakassovellukselle toimitetaan satunnaisesti luotu koodi (nk. CSRF Token) istunnon alustuksessa, joka edellytetään toimitettavan jokaisessa sovelluksen tilaa muokkaavassa pyynnössä [36].

WWW-sovelluspalveluissa ei yleensä käytetä uudelleenohjauksia. Kun uudelleenohjauksia käytetään, tulee asiakassovellukseen yleensä suoraan toteuttaa näiden ohjauksien seuraaminen. Jos uudelleenohjauksia käytetään, on niiden validointi toteutettava samalla tavalla kuin selainpohjaiseen sovellukseen.

3.1.2 WWW-sovelluspalveluille tyypilliset haavoittuvuudet

Vaikka selainkäyttöiset verkkopalvelut ja WWW-sovelluspalvelut jakavat suurimman osan haavoittuvuuksistaan, tulee WWW-sovelluspalveluita kehittäessä kiinnittää erityistä huomiota tiettyihin haavoittuvuuksiin. SOAP-pohjaisiin WWW-sovelluspalveluihin verrattuna REST-arkkitehtuurityyliä noudattavat WWW-sovelluspalvelut omaavat pääasiassa samat haavoittuvuudet. SOAP-protokollassa on kuitenkin tarkasti standardoituja tietoturvaominaisuuksia (mm. WS-Security-määritelmä), kun REST-arkkitehtuurityylissä tietoturvaominaisuuksien kehittäminen, käyttäen mahdollisesti valmiita standardeja tai oppaita, on itse kehittäjän vastuulla.

WWW-sovelluspalveluun kohdistuva farmaus (engl. farming) tarkoittaa sen luvatonta tai käyttöehtojen vastaista hyväksikäyttöä kolmannen osapuolen palvelussa. Tämän estäminen julkisessa WWW-sovelluspalvelussa on teknisesti mahdotonta [31].

XML- ja JSON-kuvauskieliä käytetään yleisesti WWW-sovelluspalveluissa kuvaamaan tietoa. Näissä muodoissa kuvattu tieto jäsennetään WWW-sovelluspalvelussa käyttäen joko itse kehitettyä tai kolmannen osapuolen jäsennintä. Näitä jäsentimiä vastaan saatetaan kohdistaa hyökkäyksiä. Nämä injektio-tyyppiset hyökkäykset voivat esimerkiksi aiheuttaa palvelun ylikuormittumista tai paljastaa arkaluontoista tietoa.

XML-kuvauskielessä on paljon erilaisia ominaisuuksia, jotka on tarkoitettu helpottamaan XML-dokumenttien laatimista ja validointia. Joitain näitä ominaisuuksia voidaan kuitenkin käyttää hyväksi hyökkäyksissä XML-jäsentimiä vastaan.

Yksi XML-jäsentimeen kohdistuva hyökkäys on nk. “Entity expansion XML bomb”, joka käyttää hyväksi XML-kuvauskielen DTD (Document Type Definition) -määritysten ENTITY-elementtiä. Hyökkäys aiheuttaa palvelimelle ylimääräistä kuormaa, eli se on tyypiltään DoS (Denial of Service) -tyyppinen. [32.]

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2
"&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Koodiesimerkki 4. “The billion laughs” -hyökkäyksen XML-kuvaus [32].

Koodiesimerkissä 4 on esitetty “The billion laughs” -hyökkäyksen, joka on tyypiltään “Entity expansion XML bomb”, XML-kuvaus. Siinä käytetään hyväksi ENTITY-elementin kykyä sisältää toisia ENTITY-elementtejä. Haavoittuvainen XML-jäsennin tulkitsee “&lol9;” merkkijonon lopulta miljardina “lol”-merkkijonona aiheuttaen palvelimelle kuormaa. Eräs toinen XML-kuvauskielen DTD-määrityksiä hyväksikäyttävä hyökkäys perustuu saman ENTITY-elementin kykyyn viitata ulkoiseen sisältöön URL-osoitteiden avulla.

```
<!DOCTYPE external [
<!ENTITY ee SYSTEM "file:///PATH/TO/simple.xml">
]>
<root>&ee;</root>
```

Koodiesimerkki 5. ENTITY-elementin URL-viittauksia hyväksikäyttävän hyökkäyksen XML-kuvaus [32].

Koodiesimerkissä 5 on esimerkki ENTITY-elementtien URL-viittauksia hyväksikäyttävästä hyökkäyksestä. Hyökkäyksessä merkkijono “ⅇ” korvattaisiin “simple.xml”-nimisen tiedoston sisällöllä. Osoitteena voisi olla myös “http://”-alkuinen URL-osoite, joka aiheuttaa HTTP GET -muotoisen HTTP-pyynnön lähettämisen annettuun URL-osoitteeseen. Tällä hyökkäyksellä hyökkääjä voi käyttää haavoittuvaa XML-jäsennintä suorittavaa konetta hyväkseen lähettämään HTTP-pyyntöjä tai lukemaan arkaluontoisia tiedostoja. [32.]

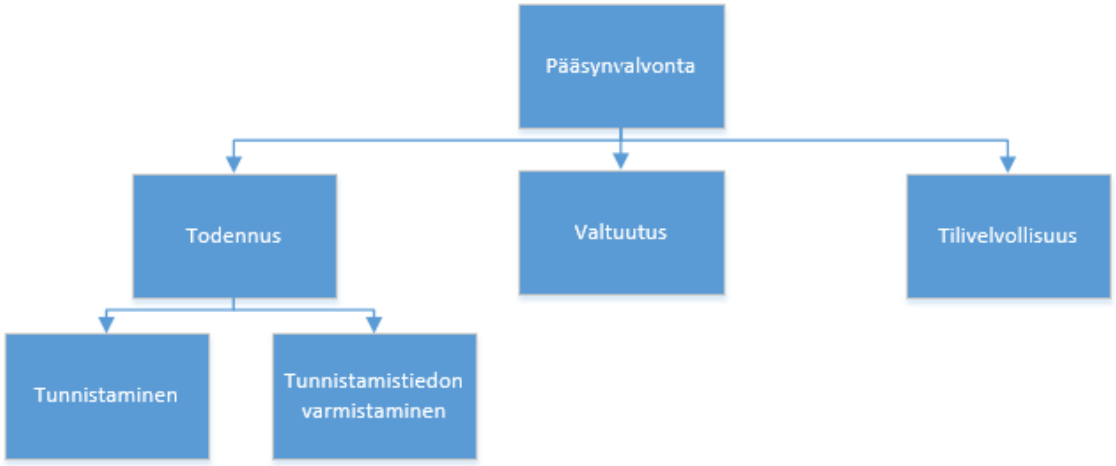
JSON-kuvauskielessä ei sen yksinkertaisuuden vuoksi ole toistaiseksi todettu paljon WWW-sovelluspalveluita koskevia haavoittuvuuksia. Koska JSON-kuvauskieli kuitenkin pohjautuu palvelinohjelmistoissakin nykyään käytettyyn JavaScript-ohjelmointikieleen, tulee jäsentämisessä noudattaa erityistä huolellisuutta, jos se tehdään käyttäen JavaScript-ohjelmointikieltä, koska haavoittuvainen jäsennin saattaa tulkita JSON-objektiin sisällytettyä JavaScript-koodia sellaisenaan mahdollistaen haitallisen ohjelmakoodin suorituksen.

Koska kuvauskielten haavoittuvuuksia hyväksikäyttävät hyökkäykset ovat injektiotyyppisiä, voidaan niiltä suojautua sanitoimalla palvelun vastaanottamat syötteet niin, että kuvauskielten rakenteissa käytetyt merkit hylätään syötteistä.

3.2 Pääsynvalvonta

Sovellusten haavoittuvuuksien hyväksikäytön estäminen on tärkeä osa tietoturvaa, mutta tietoturvariskejä on monia ja niitä hallitaan monella eri tavalla. Eräs toinen tärkeä tietoturvariskien hallintakeino on pääsynvalvonta, jolla pyritään estämään valtuuttamaton pääsy, ja mahdollistamaan valtuutettu pääsy resursseihin.

Järjestelmään, oli se sitten verkkosovellus tai mikä tahansa muu tietojärjestelmä, on usein tarve rajoittaa pääsyä niin, että sitä voivat käyttää vain valtuutetut henkilöt tai järjestelmät. Toimiva pääsynvalvonta koostuu kolmesta elementistä: todennuksesta (engl. authentication), valtuuttamisesta (engl. authorization) ja tilivelvollisuudesta (engl. accountability) [11, kohta 7.1.2]. Kuvassa 1 on mallinnettu pääsynvalvonnan käsitteiden välinen hierarkia.



Kuva 1. Pääsynvalvonnan käsitteiden välinen hierarkia.

Todennus (engl. authentication) tarkoittaa käyttäjän tai muun osapuolen tunnistamista ja tunnistamistiedon varmistamista [9, kohta 3.8]. Tunnistaminen voidaan tehdä monella eri tavalla. Tyypillisesti verkkosovelluksiin tunnistaudutaan käyttäjänimellä ja salasanalla. Tämän lisäksi voidaan vaatia myös esimerkiksi kertakäyttöinen salasana (OTP eli one time password). Myös erilaisia fyysisten apulaitteiden avulla toteutettuja todennustapoja on olemassa, kuten sormenjäljen tai tunnistekortin tarkistaminen. Niin ikään eri järjestelmät, kuten WWW-sovelluspalvelut saattavat vaatia muiden järjestelmien todennusta. Esimerkiksi WWW-sovelluspalvelu voi vaatia asiakassovelluksen todennusta tai asiakassovellus saattaa vaatia WWW-sovelluspalvelun todennusta.

Valtuutus tarkoittaa sen varmistamista, että jo todennetulla osapuolella on lupa tehdä jokin toimenpide [10]. Nämä luvat, eli käyttöoikeudet voivat määräytyä monella eri tapaa ja niihin voi liittyä paljon erilaista logiikkaa.

Tilivelvollisuus tarkoittaa sitä, että käyttäjän toimista tallennetaan jälki ja käyttäjä voidaan pitää vastuullisena näistä toimista. Tämä toteutetaan käyttämällä pääasiassa lokitusta. Tallennettuja jälkiä tutkimalla voidaan jälkikäteen huomata mahdolliset väärinkäytökset. [11, kohta 7.1.2.3.]

3.2.1 Pääsynvalvonnan mallit

Pääsynvalvonnan mallien avulla määritellään, missä tapauksessa käyttäjällä on pääsy johonkin resurssiin. Kolme tunnetuinta pääsynvalvonnan mallia ovat Discretionary access control (DAC), Mandatory access control (MAC) ja Role-based access control (RBAC). [11, kohta 7.1.3.] Näiden lisäksi on myös lukuisia muita, kuten Rule-based access control (RBAC). Mallit kuvaavat abstraktilla tasolla pääsynvalvonnan eivätkä ole sidottu pelkästään tietotekniikan kontekstiin.

Discretionary access control -malli perustuu resurssien omistajuuteen. Esimerkiksi käyttäjän luodessa tiedoston voi hän itse päättää, kenellä muilla on oikeuksia tiedostoon. Oikeuksia on erilaisia, kuten tiedostojen tapauksessa luku- ja kirjoitusoikeudet ovat erikseen. Oikeudet ovat määritelty Access control list (ACL) -tiedoissa. Tätä mallia noudattavat monet käyttöjärjestelmät. Malli on raskas, sillä jokaiselle resurssille on määritelty oikeudet erikseen. [11, kohta 7.1.3.1.]

Mandatory access control -mallin mukaan jokaisella resurssilla on suojaustaso. Mallissa jokaisella resurssia käyttävällä taholla tulee olla vastaava tai korkeampi lupataso resurssin käyttöön. Erikoista mallissa on se, että matalamman lupatason omaava käyttäjä voi luoda korkeamman lupatason resurssin, mutta ei omaa siihen käyttöoikeutta. Mallia käytetään yleisesti esimerkiksi puolustushallinnoissa. [11, kohta 7.1.3.2.]

Role-based access control, eli roolipohjainen pääsynvalvontamalli on yleisluontoisempi kuin DAC tai MAC, koska se kuvailee vain tietoturvalle oleelliset käsitteet. Yksinkertaistettuna mallissa käyttäjälle annetaan yksi tai useampi rooli ja rooleilla on tiettyjä käyttöoikeuksia. [12, kohta 10.1.] RBAC ei tarjoa suoraa ratkaisua siihen, miten voidaan sallia esimerkiksi käyttäjälle pääsyn omaan, mutta ei muiden resurssiin. Malli ei myöskään estä toteuttamasta tätä. [13.] Tällainen hienojakoisempi pääsynvalvonta toteutetaan yleensä sovelluslogiikassa. RBAC on eniten verkkosovelluksissa käytetty malli.

3.2.2 Todennusmenetelmät WWW-sovelluspalveluissa

Yksinkertaisin tapa todentaa käyttäjä WWW-sovelluspalvelussa on HTTP Basic Authentication [5]. Tällöin käyttäjän kirjautumistiedot lähetetään Base64-koodattuina HTTP-pyynnön “Authorization”-otsakkeessa jokaisessa HTTP-pyynnössä. Koska Base64-koodaus ei ole salaus, ovat kirjautumistiedot luettavissa helposti HTTP-pyynnöstä

dekoodaamalla. Tämän vuoksi tämän menetelmän kanssa tulisi aina käyttää TLS-salausprotokollaa, jolloin se on kohtuullisen tietoturvallinen [5]. Selväkieliset tunnukset voidaan lähettää myös “Authorization”-otsakkeen lisäksi HTTP-pyynnön muissakin osissa.

Sen sijaan että HTTP-pyynnöissä lähetetään selväkieliset kirjautumistiedot, voidaan niistä sen sijaan laskea tiiviste. Tiiviste tarkistetaan vertaamalla sitä käyttäjätietokantaan tallennettuun tiivisteeseen. Tyypillisesti käyttäjänimestä ei lasketa tiivistettä, vaan sitä käytetään selväkielisenä, mutta myös käyttäjänimen säilöminen tiivisteenä voi tarjota lisäturvaa siinä tapauksessa, jos käyttäjätietokanta paljastuu [6].

Tiivisteen laskemisessa tulisi käyttää nk. suolaa (engl. salt). Se on satunnainen numero, jota käytetään apuna tiivisteen laskemisessa. Suolan luonnissa tulee huolehtia siitä, että se on todella satunnainen ja käytännössä mahdoton selvittää jälkikäteen. Jos suolaa ei käytetä, ja tiivisteet lasketaan suoraan alkuperäisestä tiedosta, voidaan se saada selville käyttämällä ns. Rainbow-taulukkoa, jossa tiivisteet on yhdistetty alkuperäiseen tietoon. Esimerkiksi 16-tavuisen suolan käyttö vaatisi 2^{128} Rainbow-taulukon luonnin, mikä vaatisi huomattavan laskenta- ja tallennuskapasiteetin. [7.]

Suolaamattomat tiivisteet käyvät yhä turvattomammiksi. Internetistä löytyy erilaisia palveluita, joilla on suuria tietokantoja salasanojen tiivisteistä. Tällaisten palveluiden avulla saadaan erittäin helposti selville alkuperäinen tieto heikoista salasanoista. Crackstation (https://crackstation.net/) -palvelua testatessa, saatiin merkkijonojen ”february” ja “helmikuu” SHA-256-algoritmillä laskettujen tiivisteiden avulla alkuperäiset merkkijonot selville nopeasti. Palvelu ei saanut merkkijonojen, joissa oli numeroita; erikokoisia kirjaimia tai erikoismerkkejä, tiivisteistä selville alkuperäistä merkkijonoa.

Myös käytetyllä tiivistealgoritmilla on suuri merkitys. Laskentatehojen kasvaessa vanhempien algoritmien laskeminen voidaan suorittaa yhä nopeammin. Esimerkiksi aiemmin suosittua MD5-algoritmia ei enää pidetä turvallisena, koska se voidaan suorittaa nykyprosessoreilla hyvin nopeasti. Nykyisin yleisesti turvallisina algoritmeina pidetään esimerkiksi SHA-2-luokan algoritmeja, joihin myös SHA-256-algoritmi kuuluu.

Kirjautumistietojen sijaan voidaan WWW-sovelluspalvelulle lähettää todennustietona käyttöoikeustietue (engl. token). Yleensä rajoitetun ajan voimassa oleva käyttöoikeustietue myönnetään käyttäjälle, joka on todennettu onnistuneesti. Sitä voidaan myös

käyttää käyttöoikeuden myöntämisessä kolmannen osapuolen sovellukselle esimerkiksi tietyn käyttäjän tietoihin. Tyypillisesti käyttöoikeustietue lähetetään WWW-sovelluspalvelulle HTTP-pyynnön otsaketiedoissa, URL-osoitteessa tai evästeessä. Yksi merkittävä etu niiden käytössä on mahdollisuus rajoittaa yksittäisen käyttöoikeustieteen käyttöoikeuksia. [8.]

Perinteinen verkkosovellusten istuntopohjainen todennus toteuttaa käyttöoikeustietuepohjaisen todennuksen määritelmää: käyttäjä tunnistautuu palveluun ja vastaanottaa istuntoavaimen evästeenä. Käyttäjä lähettää jokaisessa palveluun kohdistuvassa pyynnössään evästeessä olevan istuntoavaimen, joka yhdistetään palvelussa olevaan istuntoon, jolloin hän saa istunnon voimassa ollessa käyttöoikeuden palveluun. Kaikki tieto, kuten käyttäjän tiedot ja käyttöoikeudet, säilötään palvelun istunnossa, jolloin tiedot sijaitsevat palvelun palvelimella eikä arkaluontoista tietoa lähetetä pyynnöissä.

Käyttöoikeustietueen ei välttämättä täydy olla istuntoavain, vaan siinä voidaan suoraan ilmaista käyttäjän tunnistetiedot ja käyttäjään liittyviä attribuutteja. Tällöin käyttöoikeustietueen täytyy tietoturvan vuoksi olla sähköisesti allekirjoitettu, estäen sen väärentämisen ja muokkaamisen. Jos käyttöoikeustietueessa on arkaluontoista tietoa, olisi se tai ainakin kommunikaatiokanava hyvä myös salata.

Useat nykyään yleistyvät, kertakirjautumisen mahdollistavat tekniikat ja standardit ovat luonteeltaan käyttöoikeustietuepohjaisia. Tällaisia teknologioita ovat mm. OpenID, SAML ja OAuth2.

3.2.3 Valtuutus WWW-sovelluspalveluissa

Käyttäjätietojen todentamisen jälkeen WWW-sovelluspalvelun tulee selvittää, onko todennetulla käyttäjällä oikeus tehdä aiottu toiminto. Koska REST-arkkitehtuurityyppiä noudattavissa WWW-sovelluspalveluiden rajapinnoissa URL-osoite ja HTTP-metodi ovat merkitsevässä asemassa, voidaan niiden perusteella tehdä karkeajakoista valtuuttamista. Tämäntyyppinen karkeajakoinen valtuuttaminen on mahdollista erillään kohdesovelluksen logiikasta, ja se voidaan suorittaa erillisellä palvelimella.

URL-osoitteisiin ja HTTP-metodeihin perustuva roolipohjainen valtuuttaminen on mahdollista toteuttaa helposti esimerkiksi Spring Security -sovelluskehyksellä Java-teknologiaan perustuviin verkkosovelluksiin. Toteutuksessa on parasta käyttää

nk. whitelisting-menetelmää, jossa luetellaan erikseen sallitut URL-osoitteet, HTTP-metodin ja käyttäjän roolin yhdistelmät, ja kaikki muut pyynnöt evätään.

Taulukko 3. Esimerkkejä valtuutussäännöistä.

URL-osoite	Sallitut HTTP-metodit	Vaaditut roolit
http://example.com/rest/public/*	GET	-
http://example.com/rest/public/*	POST, PUT, DELETE	käyttäjät
http://example.com/rest/private/*	GET, POST, PUT, DELETE	käyttäjät
http://example.com/rest/admin/*	GET, POST, PUT, DELETE	pääkäyttäjät

Taulukossa 3 on esitetty esimerkkejä URL-osoitteisiin ja HTTP-metodeihin perustuvista roolipohjaisista valtuutussäännöistä. Kuten esimerkissä on käytetty, useissa teknologioissa on mahdollista käyttää jokerimerkkiä URL-osoitteessa, mikä helpottaa huomattavasti sääntöjen luontia ja ylläpitoa. Esimerkissä pääkäyttäjä-roolilla on myös käyttäjän käyttöoikeudet.

Taulukko 4. Esimerkkejä valtuutussääntöihin perustuvista valtuutuspäätöksistä.

URL-osoite	HTTP-metodi	rooli	lopputulos
http://example.com/rest/public/documents/1	GET	-	salli
http://example.com/rest/public/documents/1	PUT	-	estä
http://example.com/rest/public/documents/	POST	käyttäjä	salli
http://example.com/rest/private/posts/4	DELETE	pääkäyttäjä	salli
http://example.com/rest/private/	GET	-	estä

Taulukossa 4 on esitetty esimerkkejä URL-osoitteen, HTTP-metodin ja käyttäjän rooleista sekä pyynnön salliminen tai estäminen perustuen taulukossa 3 esitettyihin sääntöihin.

Esimerkeissä osoitettujen tietojen lisäksi olisi karkejakoisessa valtuuttamisessa mahdollista käyttää muitakin tietoja, kuten asiakassovelluksen IP-osoitetta, aikatietoja tai mitä tahansa käyttäjästä tallennettua tietoa, mikäli käyttäjätietokantaan on yhteys.

Vaikka URL-osoitteen, HTTP-metodin ja käyttäjän roolin avulla on mahdollista toteuttaa karkeajakoinen valtuutus, ei tämä poista tarvetta varsinaisessa sovelluksessa tapahtuvalle hienojakoiselle valtuutukselle. Tällainen hienojakoinen valtuuttaminen voisi olla esimerkiksi sitä, että käyttäjä pystyy muokkaamaan vain omia resurssejaan, kuten käyttäjätietojaan tai blogikirjoituksiaan. Tämän tyyppinen valtuutus edellyttää sovelluksen sisäisten tietojen hyödyntämistä, joten sen suorittaminen sovelluksen ulkopuolella on hankalaa. Vain hienojakoisella valtuutuksella voidaan estää taulukossa 2 esitetty haavoittuvuus numero neljä, eli turvattomat objektiviittaukset.

4 Tekniset standardit

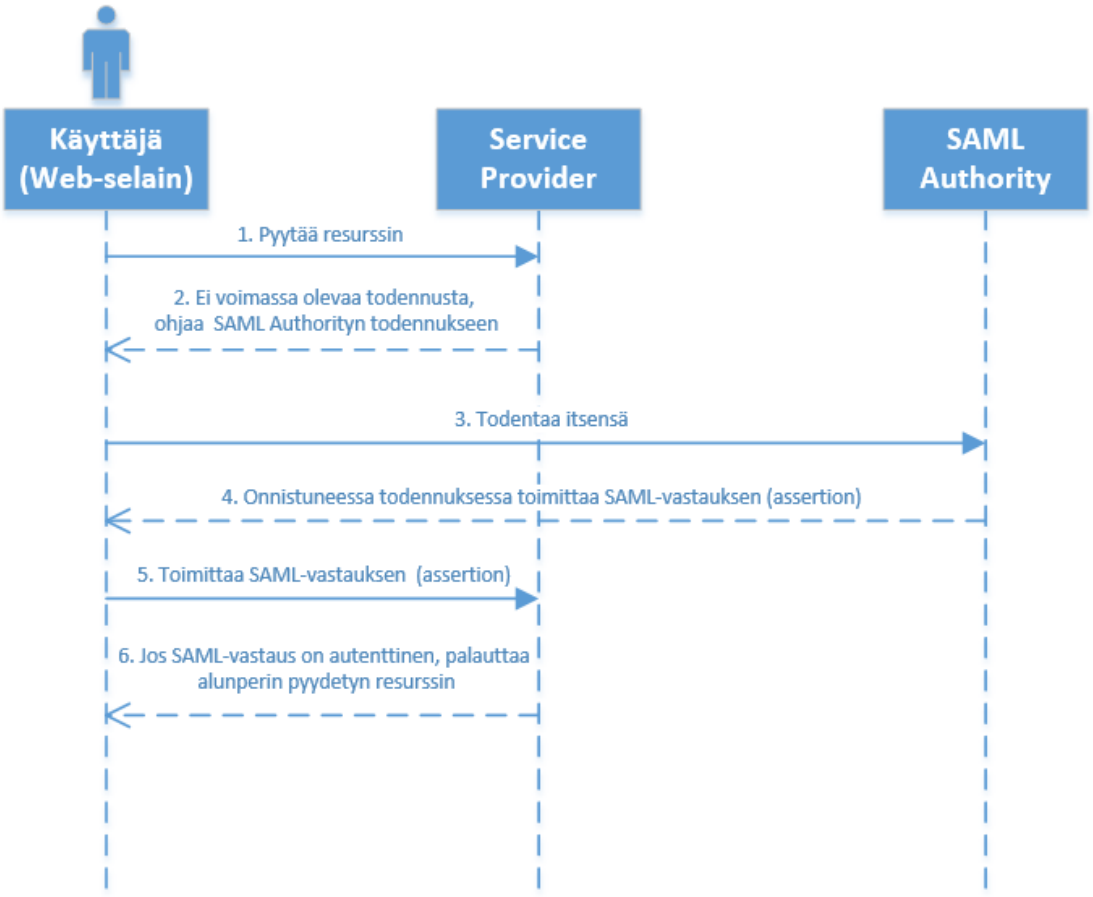
Koska toimivan, tietoturvallisen ja käyttäjäystävällisen pääsynvalvonnan toteuttaminen on verkkosovelluksen tai WWW-sovelluspalvelun kehittäjälle erittäin haastavaa ja työlästä, on kehitetty joukko erilaisia teknisiä standardeja. Nämä standardit pyrkivät helpottamaan toimivan pääsynvalvonnan toteuttamista ja joidenkin tietoturvauhkien torjumista. Standardit helpottavat myös tietoturva- ja pääsynvalvontaratkaisujen tuotteistamista.

4.1 SAML 2.0

SAML 2.0, joka on OASIS-yhtymän vuonna 2005 hyväksymä standardi, on kehys tietoturvaan liittyvän tiedon välittämiseen luotettujen osapuolten välillä. Sen tärkein komponentti on assertio (vapaasti suomennettuna vakuutus). [14, kohta 4.3.3.]

Assertioissa subjektilla (esimerkiksi käyttäjä tai tietokonejärjestelmä) on identiteetti jollain toimialueella. Assertiot voivat ilmaista tietoa subjektista, kuten todennukseen käytetyt todennusmenetelmät. Ne voivat sisältää myös valtuuttamiseen käytettävää tietoa. Assertio voi esimerkiksi kertoa suoraan, onko subjektilla valtuus käsitellä tiettyä resurssia. Assertiot toimittaa SAML-auktoriteetti (engl. SAML Authority), joka on vastuussa subjektin todentamisesta. [14, kohta 4.3.3.] SAML-standardia voidaan käyttää käyttöoi-

keustietuepohjaisen todennuksen toteuttamiseen, jolloin SAML-assertio toimii käyttöoikeustietueena.



Kuva 2. SAML Web SSO -profiilin käyttötapaus karkealla tasolla.

SAML-profiilit määrittelevät sääntöjä SAML-protokollan käyttöön tietyissä yhteyksissä. Ne yleensä sisältävät paljon valinnaisuutta, mutta voivat myös vaatia tiettyjen SAML-toiminnallisuuksien käyttöä. [16, kohta 1.1.] Kuvassa 2 on kuvattu SAML Web Browser SSO -profiilin käyttötapaus karkealla tasolla. Käyttötapauksen kulku on seuraavanlainen:

1. Käyttäjä pyytää haluamansa resurssin, yleensä verkkosivun.
2. Koska palveluntarjoajalla ei ole voimassa olevaa todennustietoa käyttäjälle (tyypillisesti istuntoa), ohjaa palveluntarjoaja käyttäjän SAML-auktoriteetin sivulle SAML-pyynnön kanssa.
3. Käyttäjä todentuu onnistuneesti SAML-auktoriteetille käyttäen tyypillisesti käyttäjänimeä ja salasanaa.

4. SAML-auktoriteetti toimittaa käyttäjälle SAML-assertion vastauksena.
5. Käyttäjä toimittaa SAML-assertion edelleen palveluntarjoajalle HTTP-pyynnössä.
6. SAML-assertion ollessa autenttinen palveluntarjoaja perustaa käyttäjälle istunnon ja palauttaa alun perin pyydetyn resurssin.

SAML-standardi ei suoraan vaadi TLS-salausprotokollan käyttöä, mutta suosittelee viestin sähköistä allekirjoitusta, jos kommunikaatiokanava ei ole turvallinen [15, kohta 5]. Standardi tukee sähköiseen allekirjoittamiseen soveltuvaa XML Signature -tekniikkaa [15, kohta 5.1]. Kun assertio on allekirjoitettu sähköisesti osapuolen yksityisellä avaimella, voi toinen osapuoli varmistua lähettäjästä käyttämällä lähettäjän julkista avainta. Tämä estää myös tietojen muokkaamisen sähköisen allekirjoittamisen jälkeen.

SAML-assertioiden tärkeimmät elementit voidaan myös salata sovelluskerroksen tasolla käyttäen XML Encryption -tekniikkaa [15, kohta 6.1]. Tämä estää mahdollisesti arkaluontoisen tiedon näkymisen ja sen mahdollisen lokitiedostoihin kirjoittamisen TLS-salauksen purkamisen jälkeen. Näin voi tapahtua esimerkiksi silloin, kun TLS-salaus on muodostettu kuormanjakajan kanssa, jolloin kuormanjakajan edelleen välittämien pyyntöjen sisältö näkyy selväkielisenä. Kuormanjakajat kuitenkin yleensä välittävät pyyntöjä edelleen turvallisessa verkossa, jolloin tietojen päätyminen vääriin käsiin on epätodennäköistä.

Tyypillisimmät käyttötapaukset SAML-pohjaisille teknologioille ovat verkkopalveluiden kertakirjautumisjärjestelmät, identiteetin federointi eri palveluiden välillä ja SOAP-protokollaa hyödyntävien WWW-sovelluspalveluiden todennus.

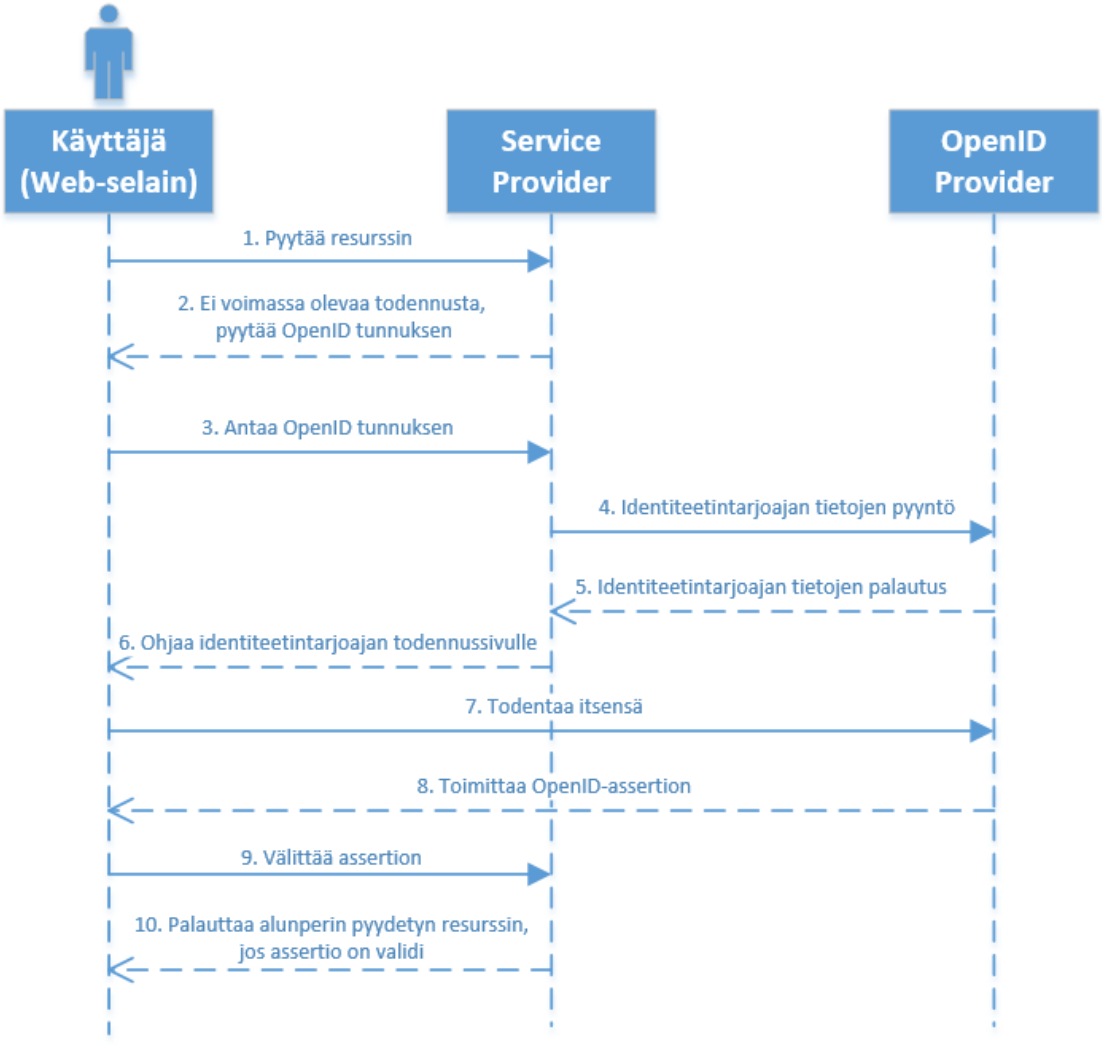
SAML on jo kypsä, tarkasti määritelty ja monikäyttöinen teknologia, minkä vuoksi se on laajasti käytössä verkkopalveluissa. Se on myös monimutkainen ja XML-pohjaisena teknologiana raskas. Vastaavat teknologiat ovat kasvattamassa suosiotaan, kuten JSON Web Token (JWT), joka pyrkii olemaan kevyempi vaihtoehto. [17.]

4.2 OpenID

OpenID on periaatteeltaan hyvin samanlainen teknologia kuin SAML-standardin Web Browser SSO -profiili. OpenID perustuu vahvasti siihen, että subjekti voi esittää identi-

teettinsä URL-osoitteena, jota käytetään useisiin palveluihin. SAML-standardiin verrattuna OpenID on hyvin kevyt teknologia, eikä sitä suunniteltu sovelluksiin, joissa on korkeat tietoturvavaatimukset. OpenID-säätiö on kehittänyt joukon identiteetin hallintaan liittyviä OpenID-määrityksiä. [19 kohta 4.4.1.] Keskeisin määritelmistä on OpenID-todennus (engl. OpenID Authentication).

OpenID Authentication 2.0 määrittelee, kuinka identiteetintarjoajat (engl. Identity Provider, IdP) välittävät todennuksen tuloksen palveluntarjoajalle tai toisin ilmaistuna resurssintarjoajalle (engl. Resource Provider, RP). [19, kohta 4.4.2.]



Kuva 3. OpenID-todennus.

Kuvassa 3 on kuvattu onnistunut OpenID-todennus karkealla tasolla. Todennuksen kulku menee seuraavalla tavalla:

1. Käyttäjä pyytää haluamansa resurssin, kuten verkkosivun.
2. Palveluntarjoaja vaatii todennusta. Koska voimassaolevaa istuntoa ei ole, pyytää palveluntarjoaja käyttäjää syöttämään OpenID-tunnuksensa.
3. Käyttäjä syöttää tunnisteensa. Tunniste on, riippuen tunnuksen tyypistä, URL- tai XRI-muotoinen.
4. Palveluntarjoaja, käyttäen käyttäjän tunnistetta hyväkseen, pyytää identiteetintarjoajalta sen tietoja, joita palveluntarjoaja tarvitsee käyttäjän todennukseen.
5. Identiteetintarjoaja palauttaa tiedot.
6. Palveluntarjoaja ohjaa käyttäjän identiteetintarjoajan sen todennussivulle.
7. Käyttäjä todentaa itsensä identiteetintarjoajan sivulla.
8. Identiteetintarjoaja palauttaa assertion käyttäjälle, joka välittää sen palveluntarjoajalle yleensä HTTP Redirect -toiminnon avulla.
9. Palveluntarjoaja validoi assertion. Sen ollessa validi, palveluntarjoaja palauttaa alun perin pyydetyn resurssin.

XRI-tunnisteiden avulla voidaan ilmaista standardilla tavalla resurssin tunniste riippumatta resurssin esitysmuodosta tai siitä, onko sillä esitysmuotoa lainkaan [20, kohta 1.1]. XRI-tunnisteet ovat samankaltaisia URL-osoitteiden kanssa abstraktoiden pois esimerkiksi sovellustason protokollan, kuten HTTP-protokollan, ja tarjoten enemmän ominaisuuksia.

Riippumatta käyttäjän tunnisteen tyypistä palveluntarjoaja selvittää sen avulla identiteetintarjoajan tietoja pyytämällä identiteetintarjoajalta ensisijaisesti XRDS-dokumentin. Jos sitä ei ole saatavilla, palveluntarjoaja pyytää HTML-dokumentin, josta se saa tarvittavat tiedot. Näistä tiedoista tärkeimmät ovat identiteetintarjoajan kirjautumissivu ja protokollan versio. [19, kohta 4.4.2.]

OpenID-määritelmälle on myös laajennoksia, jotka määrittelevät lisätoiminnallisuuksia. Attribute Exchange (AX) -määritelmä määrittelee tavan tallentaa tai hakea ylimääräisiä attribuutteja käyttäjästä. Provider Authententication Policy Extension -määritelmä määrittelee tavan kommunikoida todennussääntöjä mahdollistaen palveluntarjoajalle tavan vaatia vahvempi todennus palveluunsa. [19, kohdat 4.4.3 ja 4.4.4.]

OpenID on erityisen altis phising-tyyppisille hyökkäyksille. Pahantahtoinen palveluntarjoaja voi pyytää käyttäjältä tämän OpenID-tunnistetta ja identiteetintarjoajan oikealle

sivulle ohjaamisen sijaan ohjata käyttäjän phising-sivulle, joka tallentaa käyttäjän tunnukset. Phising-sivu voi toimia välittäjänä myös oikealle identiteetintarjoajalle, jolloin vielä käyttäjän todennus onnistuu. [21.]

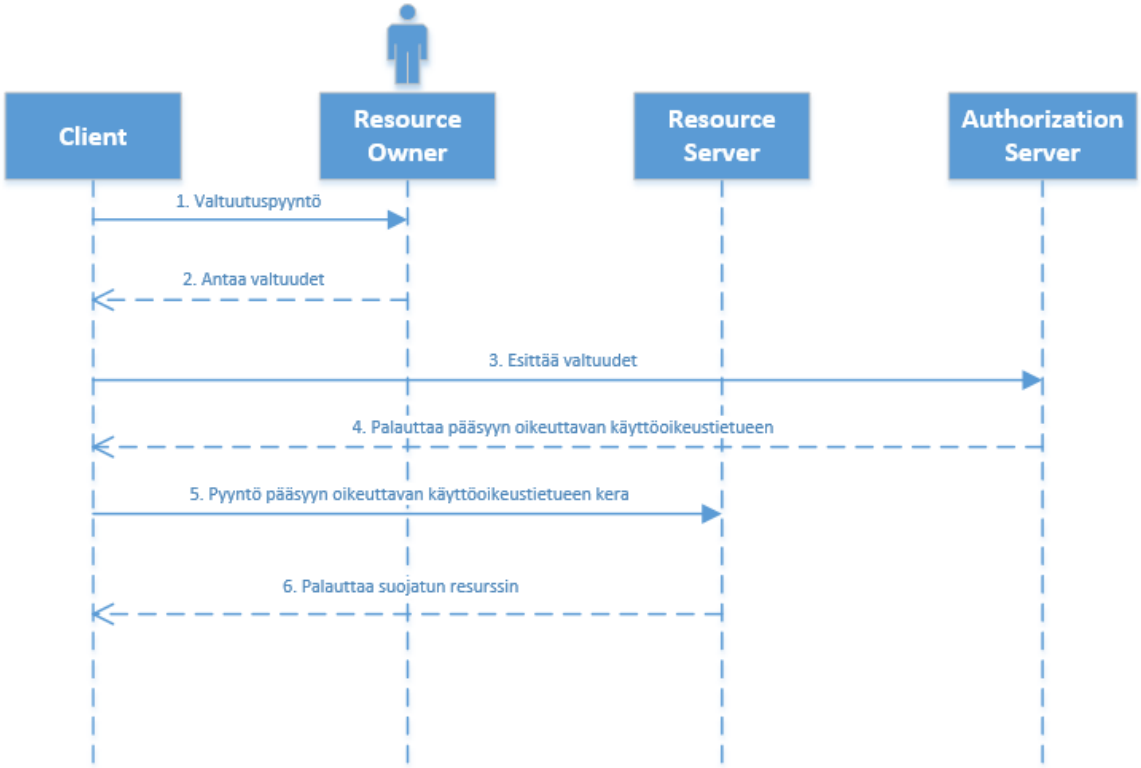
OpenID-teknologian suosio on Google Trends -palvelun mukaan ollut usean vuoden tasaisessa laskussa. Tämä on todennäköisesti seurausta uusien teknologioiden, kuten OAuth-standardin synnystä ja siitä, ettei OpenID-teknologiaa ole otettu käyttöön monessa suuressa verkkopalvelussa. Esimerkiksi Facebook-palvelulla on oma Facebook Connect -teknologia identiteetin federointiin.

4.3 OAuth2

OAuth2-määritelmä mahdollistaa kolmannen osapuolen pääsyn, jota voidaan rajoittaa, HTTP-pohjaiseen palveluun. OAuth2-määritelmä on tarkoitettu korvaamaan aiempi OAuth1-määritelmä. OAuth2-määritelmä on hyvin laaja ja määrityksiltään tarkka. Siinä käytetään neljää roolia:

- Resource owner on resurssin omistaja. Käytännössä tämä yleensä tarkoittaa verkkopalvelun käyttäjää.
- Resource server on resurssipalvelin, jossa käyttäjän tiedot sijaitsevat.
- Authorization server on valtuutuspalvelin, joka on vastuussa käyttäjän todennuksesta ja käyttöoikeustietueiden myöntämisestä. Se voi olla sama palvelin kuin resurssipalvelin, mutta yksi valtuutuspalvelin voi myös suorittaa valtuutuksen useille resurssipalvelimille.
- Client on asiakassovellus, joka pyytää pääsyä resurssiin.

OAuth2 on suhteellisen uusi teknologia, sillä sen määritelmä on valmistunut lokakuussa 2012. [22, kohta 1.]



Kuva 4. OAuth2-määritelmän toiminta abstraktisti.

Kuvassa 4 on esitetty OAuth2-määritelmän toiminta abstraktisti. Käyttötapausten todellinen kulku riippuu erityisesti siitä, että mitä valtuutustyyppiä (engl. grant) käytetään. Pääperiaatteiltaan kaikkien käyttötapausten kulku on kuitenkin seuraavanlainen:

1. Client eli asiakassovellus pyytää käyttäjältä valtuutuspyynnön. Käytännössä tämä tarkoittaa käyttäjän todentamista niin, että käyttäjälle esitetään samalla asiakassovelluksen saamat käyttöoikeudet. Tällöin käyttäjä voi päättää haluaako hän asiakassovelluksen saavan kyseisiä oikeuksia.

2. Resource owner, eli yleensä käyttäjä, antaa valtuudet sovellukselle.

3. Asiakassovellus esittää käyttöoikeudet authorization serverille, eli valtuutuspalvelimelle.

4. Valtuutuspalvelin palauttaa asiakassovellukselle pääsyn oikeuttavan käyttöoikeustietueen (engl. access token).

5. Asiakassovellus tekee pyyntöjä käyttäen pääsyyn oikeuttavaa käyttöoikeustietuetta.

6. Pääsyn oikeuttavan käyttöoikeustietueen ollessa autenttinen ja sisältävän riittävät valtuudet, käsittelee resurssipalvelin asiakassovelluksen pyynnön.

OAuth2-määritelmän mukaan asiakassovellukset voivat rekisteröityä valtuutuspalvelimelle, mutta se ei määrittele sitä, miten rekisteröinti tapahtuu. Rekisteröitäessä asiakassovellusta pitää siitä kirjata ainakin uudelleenohjauksen URL-osoite ja asiakassovelluksen luokka. OAuth2-määritelmä jakaa asiakassovellukset kahteen luokkaan: luottamukselliseen ja julkiseen, mikä perustuu siihen, voiko valtuutuspalvelin luottaa niiden kykyyn säilöä todennustietojaan turvallisesti. Nämä tiedot käsittävät lähinnä asiakassovelluksen tunnuksen, jota käytetään asiakassovelluksen todentamiseen valtuutuspalvelimella. [22, kohta 2.1.]

Valtuutuspalvelimen myöntämä pääsyyn oikeuttava käyttöoikeustietue on määritelmältään hyvin vapaamuotoinen. Se on tarkoitettu poistamaan tarve käyttää käyttäjän kirjautumistietoja jatkuvasti. Yksittäisille pääsyyn oikeuttaville käyttöoikeustietueille on myös mahdollista myöntää eri käyttöoikeuksia. Se voi sisältää viitteen, jolla saadaan haettua tietoa, kuten käyttäjän roolit, valtuutuspäätöksen tekemiseen. Pääsyyn oikeuttava käyttöoikeustietue voi sisältää myös itsessään tietoa. Tällöin tietojen tulisi olla varmennettavissa käyttämällä esimerkiksi sähköistä allekirjoitusta. [22, kohta 1.4.] OAuth2-määritelmä ei suoraan ota kantaa siihen, miten pääsyyn oikeuttavan käyttöoikeustietue todennetaan [22, kohta 10.3]. Koodiesimerkissä 6 on esitetty valtuutuspalvelimen vastaus onnistuneeseen pääsyyn oikeuttavan käyttöoikeustietueen pyyntöön. Varsinainen pääsyyn oikeuttava käyttöoikeustietue on vastauksen JSON-muotoisessa rungossa kohdassa “access_token”. Pääsyyn oikeuttava käyttöoikeustietue välitetään resurssipalvelimelle lähetettävissä pyynnöissä, mikä mahdollistaa pyyntöjen valtuuttamisen. Määritelmän mukaan se voidaan lähettää usealla eri tavalla, esimerkiksi HTTP-pyynnön “Authorization”-otsakkeessa [24, kohta 2].

```
HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache


{

  "access_token":"2YotnFZFEjr1zCsicMWpAA",

  "token_type":"example",

  "expires_in":3600,

  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",

  "example_parameter":"example_value"

}
```

Koodiesimerkki 6. Valtuutuspalvelimen HTTP-vastaus pääsyyn oikeuttavan käyttöoikeustietueen onnistuneeseen pyyntöön [22, kohta 4.3.3].

Valtuutuspalvelin voi vaihtoehtoisesti myöntää myös päivitykseen käytettävän käyttöoikeustietueen (engl. refresh token). Sitä voidaan käyttää pääsyyn oikeuttavan käyttöoikeustietueen uusimiseen käyttäjän kirjautumistietojen sijasta, kun pääsyyn oikeuttavan käyttöoikeustietueen voimassaolo on umpeutunut tai se on muuten poissa käytöstä. Päivitykseen käytettävä käyttöoikeustietue lähetetään ainoastaan valtuutuspalvelimelle, eikä koskaan esimerkiksi resurssipalvelimelle, mikä tuo lisäturvaa siinä tapauksessa, jos pääsyyn oikeuttava käyttöoikeustietue jostain syystä joutuu vääriin käsiin. [22, kohta 1.5.] Esimerkki päivitykseen käytettävästä käyttöoikeustietueesta on esitetty koodiesimerkin 6 JSON-rungon kohdassa “refresh_token”.

OAuth2-määrittelee neljä valtuutustyyppiä [22, kohta 1.3]. Valtuutustyypit määrittelevät osapuolten välisen kommunikaation kulun. Näiden eri valtuutustyyppien ansiosta OAuth2-teknologiaa voidaan soveltaa moneen erilaiseen tarkoitukseen.

Authorization code grant -valtuutustyyppi määrittelee tavan pääsyyn oikeuttavan käyttöoikeustietueen pyytämiseen pyytämällä ensin valtuutuskoodi (engl. authorization co-

de). Se on kertakäyttöinen ja lyhyen ajan voimassa oleva koodi, jolla pääsyyn oikeuttava käyttöoikeustietue voidaan pyytää. Asiakassovellus ei voi suoraan lähettää pyyntöä valtuutuspalvelimelle, vaan sen täytyy ohjata käyttäjä valtuutuspalvelimen omaan todennukseen. Valtuutuspalvelin siis vastaa käyttäjän todennuksesta. Onnistuneen todennuksen jälkeen käyttäjä ohjataan takaisin asiakassovellukseen valtuutuskoodin kera. [22, kohta 1.3.1.] Jos asiakassovellus ei ole selainpohjainen vaan esimerkiksi mobiilisovellus, asettaa tämä valtuutustyyppi haasteita asiakassovelluksen toteutukselle. Koska käyttäjä syöttää kirjautumistietonsa vain valtuutuspalvelimelle, jonka identiteetin käyttäjä pystyy varmistamaan TLS-salausprotokollan käyttämästä sertifikaatista, ei käyttäjän kirjautumistietoja missään vaiheessa syötetä asiakassovellukselle, mikä on tietoturvan kannalta hyvä ratkaisu.

Implicit grant -valtuutustyyppi on tarkoitettu erityisesti selaimessa toimiville asiakassovelluksille. Se on authorization code grant -valtuutustyypin kanssa samankaltainen, mutta valtuutuskoodin sijaan annetaan pääsyyn oikeuttava käyttöoikeustietue suoraan asiakassovellukselle. Sen lisäksi asiakassovellusta itsessään ei erikseen todenneta, vaan pelkästään käyttäjä. [22, kohta 1.3.2.] Implicit grant -valtuutustyyppi tarjoaa hieman helpomman tavan pääsyyn oikeuttavan käyttöoikeustietueen pyytämiseen asiakassovelluksille, mikä parantaa loppukäyttäjän käyttökokemusta ja suorituskykyä, mutta tekee sen tietoturvan kustannuksella.

Resource owner password credentials grant -valtuutustyyppi mahdollistaa pääsyyn oikeuttavan käyttöoikeustietueen pyytämistä suoraan käyttäjän kirjautumistiedoilla. Tätä valtuutustyyppiä tulisi käyttää vain silloin, kun käyttäjä luottaa asiakassovellukseen. [22, kohta 1.3.3.] Valtuutustyyppi mahdollistaa sen, että käyttäjän kirjautumistiedot voidaan lähettää valtuutuspalvelimelle suoraan asiakassovelluksesta. Tämä parantaa käyttökokemusta ja helpottaa toteutusta huomattavasti esimerkiksi mobiilisovelluksen tapauksessa, sillä erillistä selainta ei tarvitse avata kirjautumista varten. Tietoturvan kannalta tässä valtuutustyypissä on riskejä, sillä kirjautumistietojen syöttäminen pahantahtoiseen tai haavoittuvaiseen sovellukseen voi aiheuttaa niiden joutumisen vääriin käsiin. Myös sovellusalustan (esimerkiksi saastunut käyttöjärjestelmä tai selain) mahdolliset haittaohjelmat, kuten näppäilyjen tallentajat (engl. keylogger), voivat päästä käsiksi käyttäjän tunnuksiin.

Client credentials grant -valtuutustyyppi mahdollistaa asiakassovelluksen saavan käyttöoikeudet niihin resursseihin, joihin sillä on suoraan oikeus, eli asiakassovellus on

resource owner tai se on muuten saanut käyttöoikeudet kyseisiin resursseihin. [22, kohta 1.3.4.] Tätä valtuutustyyppiä voidaan käyttää esimerkiksi palvelinten väliseen kommunikointiin [23].

Asiakassovellus voi pyynnössään määritellä scope-parametrin pääsyyn oikeuttavan käyttöoikeustietueen pyynnössä. Tällä parametrilla kerrotaan valtuutuspalvelimelle, mitä oikeuksia asiakassovellus haluaa. Valtuutuspalvelin palauttaa myös scope-parametrin sisältäen joko saman arvon, tai eri arvon, jos myönnetyt oikeudet eroavat pyydetyistä. [22, kohta 3.] Parametri mahdollistaa hienojakoisemman valtuuttamisen. Palvelin voi kieltäytyä suoraan tietyn scope-arvon sisältävistä pyynnöistä tai antaa käyttäjän päättää, mitä oikeuksia hän antaa asiakassovellukselle.

OAuth2 luottaa tietoturvassaan TLS-salausprotokollaan. Kaikki pyynnöt, joissa lähetetään pääsyyn oikeuttavan käyttöoikeustietue, on salattava TLS-salausprotokollaa käyttäen [22, kohta 10.3]. TLS-sertifikaattien varmentaminen on myös pakollista asiakassovellukselle [22, kohta 10.9]. OAuth2-määritelmä ei kuitenkaan vaadi, vaan ainoastaan suosittelee jonkin salauksen käyttöä, kun valtuutuskoodi välitetään asiakassovellukselle. Määritelmän mukaan kuitenkaan käyttäjien (resource owner) kirjautumistietoja ei saa lähettää selväkielisenä [22, kohta 10].

Asiakassovellusten rekisteröimä “redirect_uri”-parametri ohjaa käyttäjät parametrin osoittamaan osoitteeseen onnistuneen todennuksen jälkeen. Tämä parametri on erityisen altis hyökkäyksille, sillä sen avulla käyttäjä voidaan ohjata lähettämään pyyntö valtuutuskoodin kera pahantahtoiselle sivulle. Tämä uhka liittyy taulukossa 2 esitettyyn kymmenenteen haavoittuvuuteen: validoimattomat uudelleenohjaukset ja välitykset. OAuth2-määritelmä suosittelee luottamuksellisten asiakassovellusten “redirect_uri”-parametrin ennaltarekisteröintiä ja edellyttää sitä julkisten asiakassovellusten kohdalla [22, kohta 10.6]. Toisaalta määritelmä sallii myös rekisteröimättömät asiakassovellukset, huomauttaen että niiden tietoturva vaatii erityistarkastelua [22, kohta 2.4]. Tällöin “redirect_uri”-parametrin validointi on erityisen tärkeää.

OAuth2-määritelmää on kritisoitu siitä, että se luottaa liikaa TLS-salausprotokollaan eikä määrittele esimerkiksi pyyntöjen allekirjoitusta. OAuth2-työryhmän entisen jäsenen, Eran Hammerin, mukaan TLS-salausprotokollan pettäminen mistä tahansa syystä johtaa koko tietoturva-arkkitehtuurin murtumiseen [25]. Kirjoitushetkellä luonnosvaiheessa oleva OAuth 2.0 Message Authentication Code (MAC) Tokens -määritelmä

tarjoaa keinon pyyntöjen sähköiseen allekirjoittamiseen, jolloin niiden tietoturva ei luota ainoastaan TLS-salausprotokollaan.

TLS-salausprotokollan tietoturva voi vaarantua esimerkiksi silloin, jos sertifikaatteja ei varmenneta. Tällöin ei voida saada varmuutta toisen osapuolen identiteetistä kommunikaatiossa. TLS-salausprotokollan vanhemmat versiot ovat tietoturvaltaan heikompia, eikä niihin tulisi luottaa. Salausprotokollan toteutukset voivat myös sisältää haavoittuvuuksia. Esimerkiksi Applen iOS- ja OS X -käyttöjärjestelmissä olleen TLS-kirjaston ohjelmointivirhe mahdollisti Man in the Middle -tyyppisen hyökkäyksen [26].

4.4 JSON Web Token

JSON Web Token (JWT) -määritelmä on kirjoittamishetkellä Internet Draft -tilassa oleva IETF-organisaation standardi. Se on kompakti esitystapa väitteiden (engl. claim) esittämiseen JSON-muodossa. Sen sähköinen allekirjoittaminen on mahdollista käyttäen JSON Web Signature (JWS) -määritelmää, joka on myös luonnosvaiheessa. Väitteet voidaan myös kokonaan salata käyttäen niin ikään luonnosvaiheessa olevaa JSON Web Encryption (JWE) -määritelmää. [27, kohta 1.]

JWT-käyttöoikeustietue vastaa ominaisuuksiltaan hyvin paljon SAML-assertiota, mutta jättää useita asioita avoimiksi toteuttajan tai muiden määritelmien ratkaistaviksi. JWT-määritelmä ei esimerkiksi ota kantaa siihen, millainen on osapuolten välisen kommunikaation kulku, vaan se keskittyy määrittelemään esitystavan sen sisältämälle tiedolle.

JSON Web Token -käyttöoikeustietue koostuu otsakkeesta (JWT header) ja joukosta väitteitä. Otsakkeen kentät sisältävät pääasiassa kryptograafista tietoa, jota käytetään JWT-käyttöoikeustietueen allekirjoituksessa ja salauksessa. Koska väitejoukko toimii vain JWS- ja JWE-määritelmien määrittelemänä hyötykuormana (engl. payload), on otsakkeiden yksityiskohdat määritelty JWS- ja JWE-määritelmissä. JWS- ja JWE-määritelmät käyttävät useita samoja kenttiä [28, kohta 4.1].

JWT-määritelmän mukaan väitejoukko koostuu avain-arvopareista, jossa avain on väitteen nimi merkkijonona ja arvo voi olla mikä tahansa JSON-arvo [27, kohta 2]. Määritelmä määrittelee ennalta joitain väitteitä, mutta ei edellytä niiden käyttöä. Ennalta määriteltyjä väitteitä ovat mm:

- iss (Issuer) -väitteen avulla ilmaistaan käyttöoikeustietueen myöntänyt taho.
- sub (Subject) -väitteen avulla ilmaistaan käyttöoikeustietueen subjekti.
- exp (Expiration Time) -väite ilmaisee ajan jolloin käyttöoikeustietue ei ole enään voimassa.

Väitteiden nimet jaetaan julkisiin ja yksityisiin. Julkiset väitteet tulisi olla rekisteröity IANA JSON Web Token Claims -rekisteriin tai väitteen nimen tulisi olla sellainen, että se ei ole altis törmäyksille (engl. collision) muiden nimien kanssa. Yksityiset väitteet perustuvat siihen, että käyttöoikeustietueen luoja ja käyttäjä keskenään sopivat väitteen nimen ja merkityksen. Tällöinkin tulee kuitenkin huomioida väitteiden nimien mahdolliset törmäykset. [27, kohta 4.]

JSON Web Token -käyttöoikeustietueet muodostetaan koodaamalla sekä otsake että sisältö (väitejoukko), Base64Url-koodauksella. Otsake ja sisältö koodataan erikseen ja niiden koodausten tulokset erotetaan pisteellä. Otsakkeet tulkitaan UTF-8-muodossa. Base64Url-koodauksen tulos on kompakti ja siinä käytetty merkistö on rajallinen, jolloin sitä on turvallinen käyttää useissa yhteyksissä, kuten URL-osoitteissa. Jos JWT-käyttöoikeustietueen muodostamisessa käytetään JWS-allekirjoitusta tai JWE-salausta, vaatii JWT-käyttöoikeustietueen muodostaminen useampia vaiheita. [27, kohta 7.]

```
{"iss":"joe",

"exp":1300819380,

"http://example.com/is_root":true}



eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcG
xlLmNvbS9pc19yb290Ijp0cnVlfQ
```

Koodiesimerkki 7. Sisältö selväkielisenä ja sen alapuolella koodattuna Base64Url-koodauksella [27, kohta 3.1].

Koodiesimerkissä 7 on esitetty JWT-käyttöoikeustietueen koodaaminen Base64Url-koodauksella. Lopputuloksena on merkkijono, jossa on käytetty rajoitettua merkistöä. JWT-määritelmän mukaan koodauksessa ei tule käyttää nk. “padding”-merkkejä, jotka ovat tavanomaisessa Base64-koodauksessa yleensä yhtäsuuruusmerkkejä (=) [27, kohta 7].

4.5 Standardien soveltaminen WWW-sovelluspalveluissa

Esitellyt tekniset standardit helpottavat pääsynvalvonnan toteuttamista määrittelemällä vaihtelevalla tasolla sen eri osa-alueiden toteutuksen. Esimerkiksi mikään standardi ei aseta täsmällisiä vaatimuksia sille, miten tunnistamistiedon varmistaminen tarkalleen suoritetaan, jolloin sen toteuttaminen jää täysin toteuttajan vastuulle. Standardeista vain OpenID ottaa kantaa siihen, miten käyttäjä tunnistetaan. Mikään standardi ei siis yksinään pyri toteuttamaan koko pääsynvalvontaa, vaan ne pyrkivät tarjoamaan valmiin kehyksen, joka pyrkii ratkaisemaan tietyt haasteet ja jättää tiettyjä osa-alueita toteuttajan ratkaistavaksi. Tämä mahdollistaa myös hyvin erilaisten ratkaisujen toteuttamisen – joissain ratkaisuissa tavoitellaan parasta mahdollista käyttökokemusta, kun joissain tavoitellaan parasta mahdollista tietoturvaa. Standardit eivät suoraan ole suunniteltu estämään haavoittuvuuksien hyväksikäyttöä, vaan ne keskittyvät avustamaan pääsynvalvonnan ratkaisemisessa. Tästä huolimatta monessa määritelmässä otetaan kantaa myös joidenkin hyökkäysten estämiseen ja joistain määritelmistä on saatavilla dokumentaatiota haavoittuvuuksien hyväksikäytön rajoittamisesta.

Monia teknisiä standardeja voidaan käyttää täydentämään toisiaan. Esimerkiksi OAuth2-määritelmälle on olemassa profiilit, jotka määrittelevät JWT- ja SAML 2.0 -tekniikoiden hyödyntämisen OAuth2-määritelmän kanssa. Mikään ei myöskään estä SAML-määritelmän SAML-auktoriteettia tai OAuth2-määritelmän valtuutuspalvelinta käyttämään käyttäjän todentamisessa hyväksi OpenID-toteutusta.

SAML-määritelmä ei tarjoa valmista profiilia käytettäväksi REST-arkkitehtuurityyliä noudattavien WWW-sovelluspalveluiden kanssa, vaan monet profiilit keskittyvät SOAP-protokollaan. SOAP-protokollaan liittyviä profiileja ei voi soveltaa REST-arkkitehtuurityyliä noudattavissa WWW-sovellluspalveluissa, koska profiilit ovat niin tarkasti määritelty SOAP-protokollaan yhteensopivaksi. Mikään ei kuitenkaan estä käyttämästä SAML-assertioita käyttöoikeustietueina, esimerkiksi HTTP-otsakkeessa, mutta tähän ei ole valmista standardia.

OpenID-standardi edellyttää selaimen käyttöä todennusprosessissaan, jolloin se ei sovellu käytettäväksi asiakassovelluksissa, joissa ei ole selainta. OpenID ei myöskään tarjoa ratkaisua siihen, miten kerran todennettu käyttäjä todentaa pyyntöjen välissä itsensä kohdepalvelulle ilman uudelleentodennusta. Sen ratkaiseminen jää toteuttajan vastuulle. OpenID-säätiö on kehittänyt OpenID Connect -määritelmän, joka on kehitetty

OAuth2-määritelmän laajennuksena. Sen avulla asiakassovellukset voivat varmistua resurssinomistajan, eli loppukäyttäjän, identiteettitiedosta ja saada tietoa loppukäyttäjästä [35]. Käytännössä tämä tarkoittaa sitä, että identiteetintarjoaja välittää loppukäyttäjän identiteettitiedot asiakassovellukselle pääsyyn oikeuttavan käyttöoikeustietueen yhteydessä.

JWT-määritelmä tarjoaa tarkasti määritellyn tavan muodostaa käyttöoikeustietue. Se siis ratkaisee vain hyvin pienen osan pääsynvalvontaa, mutta tekee sen täsmällisesti. Tämän ansiosta JWT-määritelmää voidaan käyttää monen muun teknologian yhteydessä. Koska esimerkiksi OAuth2-määritelmän pääsyn oikeuttavan käyttöoikeustietueen muotoa ei ole tarkasti määritelty, ei mikään estä JWT-käyttöoikeustietueen käyttämistä siihen tarkoitukseen.

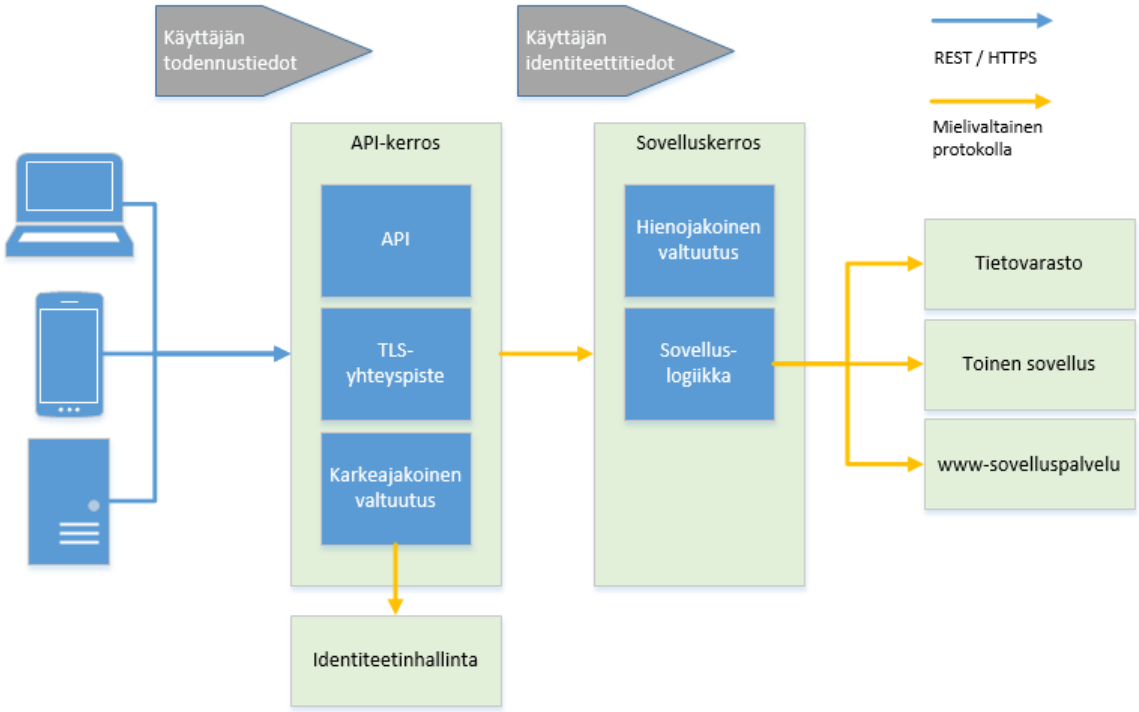
OAuth2-määritelmä on laaja ja on suunniteltu vastaamaan esimerkiksi juurikin WWW-sovelluspalveluiden tarpeisiin ja mobiilisovellusten käyttäjiin on kiinnitetty erityistä huomiota. Useiden valtuutustyyppien ansiosta se soveltuu myös moneen muuhun käyttötapaukseen. Monissa valtuutustyypeissä edellytetään selaimen käyttöä, mutta määritelmä mahdollistaa myös selaimettomien asiakassovelluksien käytön. OAuth2-määritelmän tapa erottaa asiakassovellus itse loppukäyttäjästä mahdollistaa käyttöoikeuksien myöntämisen itse asiakassovellukselle käyttäjän toimesta. Tämän ja scope-ominaisuuden ansiosta pystytään kolmannen osapuolen sovelluksille antamaan käyttöoikeuksia resursseihin hienojakoisesti, mikä on OAuth2-määritelmän oleellinen ominaisuus. OAuth2-määritelmän hyödyntäminen muiden teknologioiden kanssa rinnakkain on tehty helpoksi ja sille onkin kehitetty tai on kehitteillä useita profiileita, jotka laajentavat sen toimintaa. Määritelmää on kritisoitu sen tietoturvasta. Sitä hyödyntävässä pääsynvalvonnassa tulisikin kiinnittää erityistä huolellisuutta sen tietoturvalliseen toteuttamiseen ottaen huomioon sen eräät riskialttiit ominaisuudet. Erityisen tärkeää on varmistaa, että TLS-salausprotokolla toimii oikein.

5 Tietoturva WWW-sovelluspalvelun arkkitehtuurissa

Tietoturva on luonteeltaan niin kutsuttu ”cross-cutting concern”. Sitä ei siis mielletä yksittäiseksi toiminnallisuudeksi, joka voidaan eristää yhteen komponenttiin, vaan se on asia joka tulee ottaa huomioon arkkitehtuurin useissa eri toiminnallisuuksissa ja kom-

ponenteissa. Muita tällaisia asioita ovat mm. lokitus, välimuistitus ja transaktionhallinta. [33.]

REST-arkkitehtuurityyppiä noudattavan WWW-sovelluspalvelun arkkitehtuuri voidaan jakaa kahteen eri kerrokseen: API- ja sovelluskerrokseen, jotka voidaan myös tarvittaessa jakaa fyysisesti eri palvelimille. Sovelluskerros voi myös itsessään koostua useammasta kerroksesta. API-kerroksen tehtävänä on tarjota rajapinta ulospäin ja välittää pyynnöt edelleen sovelluskerrokselle. Sovelluskerros sisältää varsinaisen logiikan ja käsittelee pyynnön.



Kuva 5. Esimerkki WWW-sovelluspalvelun arkkitehtuurista, jossa API-kerros on eriytetty sovelluskerroksesta.

Kuvassa 5 esitetyssä WWW-sovelluspalvelun arkkitehtuurissa on mahdollista eriyttää karkeajakoinen valtuutus API-kerrokseen luvussa 3.2.3 esitetyllä tavalla. Tämä edellyttää sitä, että kerroksella on pääsy identiteetinhallintaan, josta voidaan noutaa käyttäjien tietoja. Kuvassa vaalean vihreät laatikot edustavat komponentteja, jotka voivat käytännössä olla sijoitettu eri palvelimelle. Siniset laatikot edustavat komponenttien ydintoiminnallisuuksia. API-kerros toimii tässä tapauksessa siis eräänlaisena pyyntöjen välittäjänä, johon pyritään ulkoistamaan joitain toimintoja.

Karkeajakoisen valtuutuksen eriyttäminen sovelluskerroksesta API-kerrokseen tarjoaa parhaimmillaan mm. seuraavia etuja:

- Käyttöoikeussääntöjä voidaan hallita API-kerroksessa.
- Käyttöoikeussääntöjä voidaan hyödyntää useassa sovelluksessa.
- Käyttöoikeussääntöjen muuttaminen ei vaadi sovelluksen uudelleen kääntämistä tai käyttöönottoa.
- Käyttäjän Todentaminen ja käyttäjän identiteettitietojen noutaminen voidaan ulkoistaa API-kerrokseen, jolloin sovelluskerrokselle tarjotaan käyttäjän valmiit identiteettitiedot.

Kun karkeajakoinen valtuutus on ulkoistettu API-kerrokselle, ei sovelluskerroksen tarvitse myöskään huolehtia käyttäjätietojen todentamisesta, jolloin se voi luottaa API-kerroksen sille toimittamiin käyttäjätietoihin. Tässä tulee ottaa huomioon, että API-kerroksen ja sovelluskerroksen välisen luottamuksen oletetaan olevan vahva ja niiden välisen yhteyden turvallinen. Mikäli näin ei ole, tulisi sovelluskerroksen kyetä todentamaan API-kerroksen toimittamat käyttäjätiedot ja ne tulisi myös salata.

Hienojakoinen valtuutus tulee suorittaa sovelluskerroksessa, koska ainoastaan itse sovelluksella on siihen riittävät tiedot. Esimerkiksi blogisovelluksen hienojakoiseen valtuuttamiseen voisi kuulua se, että vain blogiviestin omistaja voi poistaa tai muokata viestiä. Koska API-kerroksella ei ole pääsyä sovelluksen sisäisiin tietoihin, ei hienojakoista valtuutusta ole mahdollista toteuttaa API-kerroksessa.

Valtuutustoimintojen jakaminen tuottaa myös haasteita. Sovelluskerros luottaa kokonaan API-kerrokseen karkeajakoisessa valtuutuksessa. Jos sovellukseen olisi jostain syystä pääsy API-kerroksen ohi, sen tietoturva olisi puutteellinen.

Verkkosovellusten yhteydessä termillä API tarkoitetaan useimmiten WWW-sovelluspalvelun rajapintaa. API Management -termillä viitataan tämän rajapinnan hallintaan ja ylläpitoon. API Management pyrkii vastaamaan mm. seuraaviin haasteisiin:

- Miten voi julkaista ja päivittää rajapinta nopeasti?
- Miten voi koostaa useiden eri taustasovellusten tietoa?
- Miten tarjota pääsy rajapintaan tietoturvallisesti?

- Miten varmistaa, että rajapinta skaalautuu?
- Miten analysoida rajapinnan käyttöä?
- Miten julkaista rajapinnan dokumentaatio? [34.]

Näihin ongelmiin vastaavat API Management -ratkaisut on useimmiten tuotteistettu valmiiksi ohjelmistoksi, pilvipalveluksi tai laitteeksi. Tällaisia tuotteita tarjoavat esimerkiksi IBM, Oracle ja Apigee.

API Management -tuotteet ovat oleellisia WWW-sovelluspalveluiden tietoturvassa, sillä ne voivat toteuttaa kuvan 5 esimerkkiarkkitehtuurin API-kerroksen. Tällöin API Management -tuote voi toteuttaa karkean valtuutuksen. Sen lisäksi tuotteen vastuulla voi olla esimerkiksi palvelunestohyökkäysten esto, TLS-salausprotokollan päätepisteenä toiminen tai haitallisten pyyntöjen esto. Tuotteet usein sisältävät hallintaa helpottavia työkaluja. Niiden avulla voidaan esimerkiksi hallita keskitetysti konfiguraatioita ja pääsynvalvontaan liittyviä käyttöoikeuksia ja sääntöjä.

Tietoturvatoimintojen lisäksi tuotteet usein tarjoavat keinoja seurata WWW-sovelluspalvelun käyttöä, jolloin käytöstä voidaan luoda tilastoja, joiden avulla voidaan analysoida rajapinnan käyttöä tai laskuttaa rajapinnan käyttäjiä.

6 Yhteenveto

Insinöörityön tavoitteena oli selvittää WWW-sovelluspalveluiden vakavimmat haavoittuvuudet ja niiden hyväksikäytön estäminen sekä tutkia teknologioita, joita voidaan käyttää apuna REST-arkkitehtuurityyliä noudattavien WWW-sovelluspalveluiden pääsynvalvonnan toteuttamisessa. Näiden lisäksi tutkittiin tietoturvan suhdetta REST-arkkitehtuurityyliä noudattavan WWW-sovelluspalvelun arkkitehtuuriin. Työ aloitettiin määrittelemällä lyhyesti REST-arkkitehtuurityyli sen tietoturvan ja pääsynvalvonnan kannalta oleellisin osin.

WWW-sovelluspalveluiden haavoittuvuudet poikkeavat vain hieman tavanomaisten verkkopalveluiden haavoittuvuuksista. Suurimpana erona on, että tavanomaisia verkkopalveluita käytetään yleensä selaimen avulla, jolloin selain joko tuo lisäturvaa tai mahdollisesti aiheuttaa lisää haavoittuvuuksia. WWW-sovelluspalvelun tietoturvallises-

ta toteutuksesta huolehtiminen on sen kehittäjän vastuulla, eikä esimerkiksi REST-arkkitehtuurityyli tarjoa siihen valmiita ratkaisuja.

Työssä selvitettiin pääsynvalvonnan käsite ja sen tärkeimpien osa-alueiden suhde REST-arkkitehtuurityyliä noudattaviin WWW-sovelluspalveluihin. Pääsynvalvonta on kattotermi ja sen osa-alueita ovat: todennus, valtuutus ja tilivelvollisuus, joista todennus voidaan jakaa edelleen tunnistamiseen ja tunnistamistiedon varmistamiseen. Työssä esitettiin myös, että karkeajakoinen valtuutus on mahdollista toteuttaa REST-arkkitehtuurityyliä noudattavissa WWW-sovelluspalveluissa käyttäen apuna URL-osoitetta ja HTTP-metodia osoittamaan käyttäjän haluama toimenpide.

SAML-standardin todettiin olevan raskas ja olevan WWW-sovelluspalveluiden yhteydessä keskittynyt tukemaan SOAP-protokollaa. Sen soveltaminen REST-arkkitehtuurityyliä noudattavissa WWW-sovelluspalveluissa on hankalaa. OpenID-standardi taas keskittyy identiteetin välittämiseen identiteetintarjoajalta palveluntarjoajalle. Se tarjoaa apua vain pieneen osaan pääsynvalvonnan ratkaisemiseen, ja standardissa edellytetään selaimen toimintojen käyttöä, jolloin se ei sovellu käytettäväksi asiakassovelluksissa, joissa ei ole selainta. OAuth2-standardi soveltuu vastaamaan WWW-sovelluspalveluiden pääsynvalvonnan tarpeisiin. Sen eri valtuutustyypit mahdollistavat erilaisten päätelaitteiden käyttämisen ja sen määritelmään on tarjolla laajennoksia, jotka mahdollistavat esimerkiksi muiden standardien hyödyntämisen sen kanssa. OAuth2-standardin tietoturvaan tulee kiinnittää erityistä huomiota, sillä jotkin sen ominaisuudet ja valtuutustyypit ovat riskialttiita. JWT-standardi määrittelee muodon käyttöoikeustietueelle sekä mahdollistaa sen sähköisen allekirjoittamisen ja salaamisen. Standardin suppeuden ansiosta sitä voidaan käyttää yhteistyössä esimerkiksi OAuth2-standardin kanssa ilman, että standardien vaatimukset ovat ristiriidassa toistensa kanssa.

Insinöörityössä tutkittujen standardien käyttötarkoitus oli jokaisen kohdalla erilainen, joten niitä ei täysin voi pitää toistensa kilpailijoina, vaan parhaimmillaan niitä voi käyttää yhdessä. Laajimmat mahdollisuudet pääsynvalvonnan toteuttamiseen WWW-sovelluspalveluissa tarjoaa OAuth2, jonka kanssa on myös mahdollista käyttää muita standardeja eri tavoin laajennoksina.

Tietoturva on asia REST-arkkitehtuurityyliä noudattavien WWW-sovelluspalveluiden arkkitehtuurissa, joka koskee kaikkia sen komponentteja. Arkkitehtuurissa on mahdol-

lista eriyttää API-kerros sovelluskerroksesta, jolloin myös karkeajakoinen valtuutus voidaan toteuttaa erillään sovelluksen logiikasta. API-kerroksen todettiin olevan mahdollista toteuttaa käyttäen API Management -tuotetta, jossa muiden toimintojen ohella on myös tietoturvatoimintoja ja usein niissä on mahdollista toteuttaa karkeajakoista valtuutusta.

Lähteet

1 Fielding, Roy. Architectural Styles and the Design of Network-based Software Architectures, Chapter 5: Representational State Transfer (REST). <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Luettu 06.02.2014.

2 Elkstein, M. 2008. What is REST? Verkkodokumentti. <http://rest.elkstein.org/2008/02/what-is-rest.html>. Luettu 06.02.2014.

3 Juergen Brendel. 2010. REST constraints: A benefit-focused discussion, part 1. Verkkodokumentti. <http://blogs.mulesoft.org/rest-constraints-a-benefit-focused-discussion-part-1/>. Päivitetty 26.10.2010. Luettu 07.02.2014.

4 Arnaud Bouchez. 2011. How to implement RESTful authentication. Verkkodokumentti. <http://blog.synopse.info/post/2011/05/24/How-to-implement-RESTful-authentication>. Päivitetty 24.05.2011. Luettu 07.02.2014.

5 Joshua Thijssen. How do I let users log into my RESTful API? Verkkodokumentti. <http://restcookbook.com/Basics/loggingin/>. Luettu 10.02.2014.

6 Lucas Kauffman. 2012. RE: Why don't people hash and salt usernames before storing them. Verkkodokumentti. <http://security.stackexchange.com/questions/25374/why-dont-people-hash-and-salt-usernames-before-storing-them#25380>. Päivitetty 13.12.2012. Luettu 10.02.2014.

7 Michael Howard, David Leblanc ja John Viega. 2010. 24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them. McGraw-Hill/Osborne.

8 Chris Schmidt. 2001. Token Based Authentication -- Implementation Demonstration. Verkkodokumentti. <http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/>. Luettu 10.02.2014.

9 James Michael Stewart. 2009. CompTIA Security+ Review Guide. Sybex.

10 Anna Mar. 2013. Authentication vs Authorization. Verkkodokumentti. <http://simplicable.com/new/authentication-vs-authorization>. Päivitetty 08.06.2013. Luettu 10.02.2014.

11 Abhay Bhargav ja B.V. Kumar. 2011. Secure Java: For Web Application Development. Auerbach Publications.

12 David F. Ferraiolo, D. Richard Kuhn ja Ramaswamy Chandramouli. 2007. Role-Based Access Control, Second Edition. Artech House.

13 Role Based Access Control - Frequently Asked Questions. 2014. Verkkodokumentti. National Institute of Standards and Technology, Computer Security Division, Computer Security Resource Center. <http://csrc.nist.gov/groups/SNS/rbac/faq.html>. Päivitetty 14.01.2014. Luettu 10.02.2014.

14 Elisa Bertino, Lorenzo Martino, Federica Paci ja Anna Squicciarini. 2010. Security for Web Services and Service-Oriented Architectures. Springer.

15 Conor P. Cahill et al. 2005. Assertions and Protocols for the OASIS Security Assertion Markup Language(SAML) V2.0. Verkkodokumentti. OASIS. <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>. Päivitetty 15.03.2005. Luettu 19.02.2014.

16 Conor P. Cahill et al. 2005. Verkkodokumentti. Profiles for the OASIS Security Assertion Markup Language(SAML) V2.0. Verkkodokumentti. OASIS. <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>. Päivitetty 15.03.2005. Luettu 19.02.2014.

17 M. Jones et al. 2014. JSON Web Token (JWT): Appendix B. Relationship of JWTs to SAML Assertions. Verkkodokumentti. OAuth Working Group. <http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html#SAMLRelationship>. Päivitetty 14.02.2014. Luettu 20.02.2014.

18 REST in peace, SOAP. 2010. Verkkodokumentti. Pingdom. <http://royal.pingdom.com/2010/10/15/rest-in-peace-soap/>. Päivitetty 15.10.2010. Luettu 20.02.2014.

19 Elisa Bertino ja Kenji Takahashi. 2011. Identity Management: Concepts, Technologies, and Systems. Artech House.

20 Peter Davis et al. 2005. Extensible Resource Identifier (XRI) Syntax V2.0. Verkkodokumentti. OASIS. <https://www.oasis-open.org/committees/download.php/15377>. Päivitetty 14.11.2005. Luettu 22.02.2014.

21 Ben Laurie. 2007. OpenID: Phishing Heaven. Verkkodokumentti. <http://www.links.org/?p=187>. Päivitetty 19.01.2007. Luettu 23.02.2014.

22 D. Hardt et al. 2012. RFC 6749: The OAuth 2.0 Authorization Framework. Verkkodokumentti. IETF. <http://tools.ietf.org/html/rfc6749>. Luettu 24.02.2014.

23 Client Credential Grant Type with OAuth 2.0. 2013. Verkkodokumentti. Soasecurity. <http://soasecurity.org/2013/12/02/client-credential-grant-type-with-oauth-2-0/>. Päivitetty 02.12.2013. Luettu 24.02.2014.

24 M. Jones et al. 2012. RFC: 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage. Verkkodokumentti. IETF. <http://tools.ietf.org/html/rfc6750>. Luettu 27.02.2014.

25 Eran Hammer. 2010. OAuth Bearer Tokens are a Terrible Idea. Verkkodokumentti. <http://hueniverse.com/2010/09/oauth-bearer-tokens-are-a-terrible-idea/>. Päivitetty 29.09.2010. Luettu 01.03.2014.

26 Alex Radocea. 2010. Details about Apple SSL vulnerability and iOS 7.0.6 patch. Verkkodokumentti. <http://www.crowdstrike.com/blog/details-about-apple-ssl-vulnerability-and-ios-706-patch/>. Päivitetty 21.02.2014. Luettu 01.03.2014.

27 M. Jones, J. Bradley ja N. Sakimura. 2014. Draft: JSON Web Token (JWT). Verkkodokumentti. IETF. <http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-16>. Päivitetty 14.02.2014. Luettu 01.03.2014.

28 M. Jones, E. Rescorla, J. Hildebrand. 2014. Draft: JSON Web Encryption (JWE). Verkkodokumentti. IETF. <http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-21#section-4>. Päivitetty 14.02.2014. Luettu 01.03.2014.

29 Top 10 2013. 2013. Verkkodokumentti. OWASP. <https://www.owasp.org/index.php/Top_10_2013-Top_10>. Päivitetty 23.06.2013. Luettu 04.03.2014.

30 SQL Injection Prevention Cheat Sheet. 2012. Verkkodokumentti. OWASP. <https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>. Päivitetty 06.12.2012. Luettu 04.03.2014.

31 REST Security Cheat Sheet. 2013. Verkkodokumentti. OWASP. <https://www.owasp.org/index.php/REST_Security_Cheat_Sheet>. Päivitetty 03.12.2013. Luettu 04.03.2014.

32 Dame Jovanoski. 2013. XML vulnerabilities.Verkkodokumentti. Infosec. <http://resources.infosecinstitute.com/xml-vulnerabilities/>. Päivitetty 06.05.2013. Luettu 06.03.2014.

33 Matthew D. Groves. 2012. Terminology: cross cutting concern. Verkkodokumentti. Crosscuttingconcerns. <http://crosscuttingconcerns.com/Terminology-cross-cutting-concern>. Päivitetty 19.03.2012. Luettu 21.03.2014.

34 Katherine Sanders 2013. What is API Management and why do I need it?. Verkkodokumentti. Mobile Business Insights. <http://asmarterplanet.com/mobile-enterprise/blog/2013/06/what-is-api-management-and-why-do-i-need-it.html>. Päivitetty 28.06.2013. Luettu 21.03.2014.

35 N. Sakimura et al. 2014. OpenID Connect Core 1.0. Verkkodokumentti. OpenID-säätiö. <http://openid.net/specs/openid-connect-core-1_0.html>. Päivitetty 25.02.2014. Luettu 21.03.2014.

36 Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. 2013. OWASP. <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet>. Päivitetty 07.08.2013. Luettu 30.03.2014.

37 Chad Perrin. 2008. The CIA Triad. Verkkodokumentti. Techrepublic. <http://www.techrepublic.com/blog/it-security/the-cia-triad/>. Päivitetty 30.06.2008. Luettu 02.04.2014.