Lauri Rutanen

Spring Security -kehyksen hyödyntäminen yritys-sovelluksissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

07.12.2016

Tekijä(t) Otsikko Sivumäärä Aika	Lauri Rutanen Spring Security -kehyksen hyödyntäminen yrityssovelluksissa 34 sivua 07.12.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Jorma Räty

Tämän insinöörityön tavoitteena oli selvittää Spring Security -kehyksen hyödynnettävyys Spring-pohjaisen yritysohjelmiston tietoturvan toteuttamisessa. Työ keskittyi autentikoinnin ja auktorisoinnin toteuttamisen helppouden selvittämiseen Spring Security -kehyksen osalta.

Työssä käytiin kattavasti läpi teoriaa web-sovellusten tyypillisiin tietoturvauhkiin liittyen. Myös käytettyjen sovelluskehysten tekniseen toteutukseen perehdyttiin. Erityisesti nykyaikaisten Java-kielen sovelluskehysten käyttämään aspektipohjaiseen ohjelmointiin ja sen merkitykseen modulaarisen arkkitehtuurin luomisessa porauduttiin tarkemmin.

Insinöörityön tuloksia käytettiin hyväksi yritykselle tehdyssä sovelluksessa, joka hyödynsi Spring Securityn tarjoamaa infrastruktuuria. Tämän insinöörityön pohjalta rakennettiin luotettava autentikointi ja auktorisointi.

Lopputuotoksena syntyi tavoitteita vastaava selvitys Spring Security -kehyksen ominaisuuksista ja soveltuvuudesta yritysohjelmistoihin, sekä myös rajoitteista ja ongelmista.

Avainsanat	Tietoturva, Spring, Yrityssovellukset

Author	Lauri Rutanen
Title	Spring Security and Its Benefits in Enterprise Software
Number of Pages	34 pages
Date	7 December 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Jorma Räty, Senior Lecturer

The objective of this thesis was to study the usefulness of the Spring Security Framework in the context of enterprise applications, and on the other hand to serve as documentation for later use. This thesis was done as part of a project involving the renewal of the delivery accounting system for a company.

Part of the thesis involved introducing aspects of security in web applications. Also comprehensive technical details of the frameworks used were covered, especially concentrating on modern Java frameworks that utilize Aspect-Oriented Programming to create modular architecture.

The results of this thesis were used to create and document security concerns of the application delivered, such as authentication and authorization.

As for the final results of this thesis, the requirements specified regarding the study of the usefulness of Spring Security as the comprehensive security framework for enterprise software were met. In the process some limitations and issues were found.

Keywords	Security, Spring, Enterprise Software

Sisällys

Lyhenteet

1 Johdanto....1

2 Tietoturvallisuudesta yleisesti....3

3 Web-sovellusten tietoturva yleisesti....3

3.1 OWASP....4
3.2 10 yleisintä riskiä web-sovelluksissa (OWASP top 10)....4
3.2.1 Injektio....4
3.2.2 Rikkoutunut autentikointi ja sessionhallinta....5
3.2.3 Verkkosivun rakenne ei säily (XSS)....6
3.2.4 Epäturvallinen, suora viittaus objektiin....6
3.2.5 Tietoturvan virheellinen konfigurointi....7
3.2.6 Arkaluontoisen tiedon julkistaminen....7
3.2.7 Puuttuva funktiotason pääsynvalvonta....8
3.2.8 Puutteellinen pyynnön alkuperän tarkistus (CSRF)....8
3.2.9 Tunnettuja haavoittuvuuksia sisältävien komponenttien käyttö....9
3.2.10 Varmistamattomat uudelleenohjaukset....9

4 Taustatiedot....10

4.1 MVC-malli....10
4.2 Riippuvuuden kääntämisen periaate (Dependency Inversion Principle, DIP)....12
4.3 Käänteinen hallinta (Inversion of Control, IoC)....12
4.4 Riippuvuusinjektio (Dependency Injection, DI)....13
4.5 Aspektipohjainen ohjelmointi (Aspect Oriented Programming, AOP)....13

5 Käytetyt tekniikat ja ohjelmistokehykset....15

5.1 Maven....15
5.2 Spring....17
5.2.1 Rakenne....18
5.2.2 Springin IoC-säiliö....19
5.2.3 Springin AOP-moduuli....21
5.2.4 Spring Web MVC -kehys....21
5.2.5 Lopuksi....22

5.3 Spring Boot....22
5.4 Spring Security....23
6 Autentikoinnin toteutus Spring Securityllä....24
7 Pääsynhallinnan toteutus Spring Securityllä....26
7.1 Näkymätason pääsynhallinta....26
7.2 URL-tason pääsynhallinta....27
7.3 Metoditason pääsynhallinta....28
8 Muiden web-sovelluksiin liittyvien tietoturvaominaisuuksien kytkeminen Spring Securityssä....30
8.1 CSRF-token....30
8.2 Sessioiden luonnin rajoittaminen....31
8.3 HTTPS-yhteyskäytännön pakottaminen....32
9 Yhteenveto....33

Lyhenteet

AOP	Aspect-Oriented Programming. Ohjelmointiparadigma, joka tähtää modulaarisuuden ja joustavuuden lisäämiseen ohjelmakoodissa erottamalla joukon ns. läpileikkaavia ominaisuuksia (cross-cutting concerns).
DI	Dependency Injection eli riippuvuuksien injektointi on suunnittelumalli, joka toteuttaa IoC:n (Inversion of Control, hallinnan kääntäminen) mukaisen tavan riippuvuuksien hallitsemiseen.
DIP	Dependency Injection Principle eli riippuvuuden kääntämisperiaate on periaate, joka ohjaa tapaa, jolla luokkien välisiä riippuvuuksia hallitaan.
IoC	Inversion of Control, hallinnan kääntäminen. Suunnitteluperiaate, jonka mukaan ohjelmakoodin hallinta toimii päinvastaisesti: Käyttäjä ei itse kutsu omaa koodiaan vaan sitä kutsutaan jonkun muun (usein sovelluskehyksen) toimesta.
JDBC	Java Database Connectivity. Java-ohjelmointikielen rajapinta, joka määrittelee standardin tavan, jolla asiakassovellukset voivat käyttää tietokantaa.
LDAP	Lightweight Directory Access Protocol. Avoin ja toimittajaneutraali protokolla, joka määrittelee joukon hajautettuun käyttäjähakemistoon liittyviä palveluita, kuten autentikoinnin käyttäjätunnuksella ja salasanalla.
POJO	Plain Old Java Object. Tavallinen Java-ohjelmointikielen olio, joka ei toteuta mitään erityistä rajapintaa tai peri toteutusta muusta luokasta.
XML	Extensible Markup Language. Rakenteellinen kuvauskieli tiedonvälitykseen ja tallentamiseen.

1 Johdanto

Toimeksiantajalla on tarve korvata vanha työasemasovelluksena ajettava arvokuljetusten hallintajärjestelmä uudella, web-pohjaisella ratkaisulla. Tietoturvavaatimusten vuoksi järjestelmään haluttiin joustava autentikointi ja auktorisointi. Niiden toteutukseen valittiin jo itse projektissa käytössä olleen Spring-sovelluskehyksen ohjelmointimallia tukeva Spring Security -tietoturvakehys.

Tässä luvussa määritellään aluksi opinnäytetyön tavoitteet, rajoitukset ja rakenne. Näiden jälkeen annetaan taustatietoja käytetyistä teknologioista, kuten Spring, Spring Boot ja Spring Security -sovelluskehyksistä ja Java EE -standardista. Tämän lisäksi annetaan määritelmä yrityssovelluksien käsitteestä ja valotetaan sovelluskehyksien ideaa ja toimintaperiaatteita.

Insinöörityö tehtiin osana asiakasyritykselle tehtyä projektia. Sen pääasiallisena tavoitteena oli auttaa yritykselle projektina tehtävän sovelluksen tietoturvan toteutusta autentikoinnin ja auktorisoinnin osalta. Työssä myös kuvataan Spring Securityn keskeisimmät ominaisuudet teorian ja esimerkkien valossa sekä peilataan niitä toimeksiantajalle kehitetyn sovelluksen kautta yritysohjelmistojen vaatimuksiin nähden.

Opinnäytetyön tekijän tavoitteena on oppia web-sovelluksissa käytetyn, monimutkaiseksi osoittautuneen Spring Security -kehyksen perusteet ja niiden soveltaminen osana yrityssovellusta sekä sen rajoitteet ja tyypilliset sudenkuopat. Ideana on, että tämä dokumentti hyödyttäisi Java-pohjaisten web-sovellusten kehittäjiä, jotka käyttävät tai harkitsevat käyttävänsä Spring Security -sovelluskehystä autentikointia ja auktorisointia vaativiin sovelluksiin. Insinöörityön tulokset liitetään osaksi asiakasyritykselle tehdyn projektin dokumentaatiota.

Koska opinnäytetyössä keskitytään Spring Security -sovelluskehykseen, joka on yksinään jo hyvin monimutkainen, ei siinä voida esitellä kuin osa Spring-kehyksestä, johon Spring Securityn ohjelmointimalli vahvasti nojaa. Spring-kehykseen liittyvä teoria ja tausta käydään läpi siinä määrin, kuin on Spring Security -kehyksen konkreettisten esimerkkien ymmärtämiseksi tarpeellista. Opinnäytetyössä ei myöskään esitellä suoraan varsinaista asiakasyritykselle tehtävää projektia, vaan siinä käytetään projektin valossa esille tulleita Spring Securityn hyötyjä ja haittoja läpi.

Koska opinnäytetyön sisältö liittyy pääosin Javalla laadittuihin web-sovelluksiin, olettaa se lukijansa tuntevan jonkin verran Javaan ja Web-sovelluksiin liittyvää perusterminologiaa. Myös ohjelmoinnista ja erityisesti olio-ohjelmointikielten suunnittelumalleista oletetaan perustietämystä, kuten myös tietoturvan perusteista.

Koska insinöörityön toimeksiantajalla oli tarve saada web-pohjaiseen sovellukseen autentikointi ja auktorisointi, lähdettiin liikkeelle käytettävien teknologioiden valinnasta. Koska opinnäytetyön tekijällä oli jo valmiiksi jonkin verran kokemusta Spring Boot- ja Spring -sovelluskehyksistä, päädyttiin ajamaan Spring-pohjaista sovellusta Apache Tomcat -applikaatiopalvelimella Tomcatin omassa Servlet-säiliössä.

Applikaatiopalvelimella tarkoitetaan palvelinta, joka tarjoaa ohjelmallisen rajapinnan ja suoritysympäristön web-sovelluksille [2]. Apache Tomcat on Apache Software Foundationin kehittämä avoimen lähdekoodin applikaatiopalvelin, joka toteuttaa osan Java EE -spesifikaatiosta ja siinä voidaan ajaa Servlet-spesifikaation täyttäviä web-sovelluksia.

Java EE (Enterprise Edition) on Sun Microsystemsin alun perin luoma ohjelmistokehitysalusta siirrettävien ja skaalautuvien Java-ohjelmistojen kehittämiseen [1]. Spring-sovelluskehyksen kehitti vuonna 2002 tietokoneasiantuntija Rod Johnson vastaamaan ohjelmoijien tarpeita helpottamalla Java EE -pohjaisten sovellusten kehitystä [6]. Menneisyyden vuoksi Spring-sovelluskehyksen versio 4 sisältääkin tuen osalle Java EE7:ssa määritellyistä ominaisuuksista ja tuen WebSocketeille. Spring 4 tukee Servlet-teknologiaa Servlet-versiosta 2.5 eteenpäin. Spring sisältää MVC-mallin mukaisen web-sovelluskehyksen, joka käyttää Servlet-teknologiaa [6].

Laajan käytettävyytensä ja tarjoamiensa monipuolisten ominaisuuksien johdosta Spring-sovelluskehys on nykyään yksi suosituimmista Java-sovelluskehyksistä. Tämän tosiasian jajoustavuutensa vuoksi myös Spring Security -kehyksellä on suuri suosio Spring-sovelluskehittäjien joukossa.

Opinnäytetyössä viitataan paljon sanaan ”yrityssovellus”. Se on suora käännös englanninkielisestä termistä Enterprise Application, jolla tarkoitetaan yleensä yritysten käyttöön tarkoitettua ohjelmistoa, joka sisältää lähes poikkeuksetta tiettyjä toimintoja ja ominaisuuksia, jotka vaativat pysyväistiedon tallennusta (tyypillisesti tietokantaan) ja keskitetyn autentikoinnin käyttämistä (esim. Windowsin AD-palvelut) sovellukselle tunnistauduttaessa. Tyypillisimpiä yrityssovelluksia ovat ainakin

- yrityksen kirjanpitosovellukset
- asiakkuudenhallintajärjestelmät
- toiminnanohjausjärjestelmät (ERP, lyhenne sanoista Enterprise Resource Planning.)
- intranet-portaalit (esimerkiksi Microsoft SharePoint tai Liferay.).

2 Tietoturvallisuudesta yleisesti

Koska insinöörityössä puhutaan tietoturvallisuudesta web-sovellusten kontekstisssa, tässä luvussa määritellään ensin mitä tietoturvalla yleisesti tarkoitetaan. Tämän jälkeen seuraavassa kappaleessa pureudutaan web-tietoturvallisuuteen yleisellä tasolla.

Tietoturvallisuuden määritelmästä ei ole olemassa yhtä kiistatonta versiota, vaan useita erilaisia määritelmiä. Helpoiten tietoturva on ymmärrettävissä siten, että se rinnastetaan niihin käytäntöihin, toimenpiteisiin ja menettelyihin, joita voidaan käyttää tietojen suojaamiseksi.

Eräs yleisesti käytetty määritelmä kuvailee sen kolmen eri ominaisuuden toteutumisen avulla: luottamuksellisuus, eheys ja käytettävyys. Luottamuksellisuudella tarkoitetaan tiedon näkymisen estämisellä ulkopuolisille tahoille. Eheydellä taas tarkoitetaan sitä, että tiedosta voidaan aina varmistaa, että sitä ei olla päästy muokkaamaan ilman lupaa. Saavuttavuudella puolestaan tarkoitetaan sitä, että tieto on saatavilla silloin, kun sitä tarvitaan. [17.]

Määritelmää voidaan kaiken lisäksi laajentaa seuraavilla käsitteillä

- Kiistämättömyys: Tietoon kohdistunutta toimenpidettä ei voida kiistää.
- Tunnistus: Tiedon käyttäjä voidaan liittää käyttäjätunnukseen
- Todennus: Tiedon käyttäjän identiteetin varmistus.

3 Web-sovellusten tietoturva yleisesti

Web-sovellusten tietoturvallisuus on tietoturvan osa-alue, joka käsittelee erityisesti verkkosivujen, verkkosovellusten ja verkkopalveluiden tietoturvaa. Tietoturvan käsitteen tavoin Web-sovellusten tietoturvaan liittyy useita eri osa-alueita ja käsitteitä, mutta mel-

kein jokaisessa web-sovelluksessa on käytössä jonkinlainen autentikointi ja auktorisointi.

3.1 OWASP

OWASP on voittoa tavoittelematon yhdistys, jonka ydintehtävä on mahdollistaa tietoturvallisten ja luotettavien sovellusten suunnittelu, käyttö, kehitys, ylläpito ja hankinta organisaatioille. Käytännössä yhdistys tarjoaa sekä avoimeen lähdekoodiin perustuvia, kaikille ilmaisia työkaluja sekä ohjeistuksia ja tarkistuslistoja. [1.]

3.2 10 yleisintä riskiä web-sovelluksissa (OWASP top 10)

Hyvänä ohjenuorana web-pohjaisia sovelluksia kehitettäessä voidaan pitää OWASP:n (Open Web Application Security Project) vuonna 2013 kehittämää dokumentaatiota yleisemmistä web-sovelluksista löydetyistä haavoittuvuuksista.

OWASP:n tunnettu dokumentti ”OWASP top 10 for 2013” perustuu seitsemältä eri sovellustietoturvaan erikoistuneelta yritykseltä saatuun tietoaineistoon, joka sisältää yli 500 000 haavoittuvuuden tiedot, jotka on kerätty sadoista eri organisaatioista ja tuhansista eri sovelluksista. Dokumentin tarkoituksena on valistaa niin käyttäjiä, sovellusarkkitehtejä, ohjelmoijia ja yrityksiä/organisaatioita huonon sovellustietoturvan riskeistä. Se tarjoaa toimenpiteitä sekä riskien torjumiseen olemassaolevissa sovelluksissa että tietoturvariskejä ennaltaehkäiseviä toimintamalleja uusia sovelluksia suunnitellessa. [2.]

Seuraavaksi käydään läpi dokumentin kattavat haavoittuvuudet yksitellen.

3.2.1 Injektio

Yleisin web-sovelluksia koskettava haavoittuvuus on järjestelmän käyttäjältä järjestelmään tulevan datan puuttuva tai puutteellinen validointi. Käyttäjän antama syöte tai sen osa voidaan virheellisesti tulkita sovelluksessa komentona, joka voi muokata ohjelman suoritusta tavalla, joka voi johtaa hyökkääjän haluamien toimintojen suorittamiseen tai tietoturvatarkistusten kiertämiseen sovelluksessa. [1.]

Tyypillisesti injektiotyyppi on SQL-injektio, jossa tietokannan hallintajärjestelmälle (Relational Database Management System, RDBMS) suoritettavaksi tarkoitettuun kyselyyn päätyy hyökkääjän määrittelemää tietoa.

Näin voi käydä esimerkiksi silloin, kun web-sovellus vaatii tunnistautumista käyttäjätunnuksella ja salasanalla. Sovellus etsii tietokannasta tunnistautumisen yhteydessä syötetyn käyttäjätunnuksen ja salasanan ja palauttaa näiden täsmätessä käyttäjän tiedot kuvan 1 mukaisella SQL-kyselyllä:

```
SELECT * FROM User WHERE username='aatu' AND password='salasana';
```

Kuva 1: Käyttäjän tietojen hakeminen kannasta.

Mikäli käyttäjien antamia syötteitä ei validoida mitenkään eikä SQL-kyselyä suoriteta esivalmisteltuna kyselynä (Prepared statement), niin hyökkääjä voi antaa salasanaksi merkkijonon ”’ OR ’1’ =’1’ LIMIT 1”. Nyt kysely ei toimikaan tarkoitetulla tavalla, vaan kysely palauttaa ehdon ”username = ’aatu’ AND password=’salasana’” sijaan kaikki tietokantataulun User-rivit, jotka vastaavat ehtoa ”’1’ = ’1’”. Koska ehto vertailee, onko merkkijono ”1” yhtä suuri kuin ”1”, se on aina tosi ja kysely palauttaa kaikkien User-taulun käyttäjät. Koska kyselyn perään lisättiin SQL-kielinen lause ”LIMIT 1”, niin ohjeistetaan tietokantaa palauttamaan vain yhden käyttäjän tiedot.

Ylläoleva SQL-injektio on vain yksi esimerkki injektiohyökkäyksistä. On tärkeää ymmärtää, että injektio voi päästä syntymään web-sovelluksessa aina, kun on riski käsitellä validoimatonta käyttäjältä peräisin olevaa dataa siten, että se voidaan mahdollisesti sotkea suoritettavaksi ohjelmakoodiksi.

3.2.2 Rikkoutunut autentikointi ja sessionhallinta

Käyttäjän autentikointi eli tunnistaminen on toimenpide, jolla voidaan varmistua käyttäjän olevan se, kuka tämä väittää olevansa. Yleisemmin tämä toteutetaan käyttäjätunnuksen ja salasanan avulla. Onnistuneen autentikoinnin jälkeen web-sovellukset luovat palvelimelle erityisen istuntotunnisteen, joka auttaa web-palvelinta yhdistämään sille saapuvat pyynnöt niihin liittyvään käyttäjään. Autentikoituneen käyttäjän selaimen tulee esittää palvelimelle tämä istuntotunniste tehdessään pyyntöjä sille. Palvelimen tulee hallita tätä prosessia kertomalla autentikointipyynnön (käyttäjätunnuksen ja salasanan lähetys) tehneen käyttäjän selaimelle tämän sovellukselle yksilöivä istuntotunniste. [4.]

Istuntotunnisteen luomista ja hallitsemista kutsutaan istunnonhallinnaksi. Koska istuntotunnisteen on tarkoitus olla tiedossa vain palvelimella ja käyttäjällä, jolle se alunperin palvelimella luotiin, se täytyy suojata ulkopuolisilta. Mikäli istuntotunniste voidaan kaapata, mahdollistaa se hyökkääjän toimimisen istuntotunnisteen palvelimella yksilöivän käyttäjän nimissä. Istuntotunnisteen paljastumisen lisäksi hyökkääjä voi käyttää hyväksi selaimessa tai web-sovelluksessa olevaa heikkoutta, joka sallii hyökkääjän määritellä haluamansa istuntotunniste uhrille. Mikäli web-sovellus ei hoida istunnonhallintaa kunnolla, voi menettely luoda samanlaisen tilanteen, kuin jos hyökkääjä vain kaappaisi palvelimen määrittämän, ”laillisen” istuntotunnisteen. [4.]

Huonosti hoidettu istunnonhallinta on OWASP top 10 -listan vakavimpia haavoittuvuuksia levinneisyytensä johdosta. Istunnonhallintaa ja tunnistautumista on vaikea toteuttaa oikein, mutta kumpaakin tarvitaan useimmissa web-sovelluksissa jollain tasolla.

3.2.3 Verkkosivun rakenne ei säily (XSS)

Verkkosivun rakenteen muuttamiseen liittyvässä hyökkäyksessä sovellus lisää käyttäjän antamaa syötettä verkkosivulle siten, että sivuston rakenne muuttuu. Syöte voi sisältää esimerkiksi ajettavaa JavaScript-koodia, joka voi aikaansaada mahdollisesti haitallisia toimintoja web-sovellukseen kirjautuneen käyttäjän nimissä. Hyökkäys hyödyntää web-sovelluksen käyttäjän selaimen luottamusta verkkosivuun.

Normaalitilanteessa web-sovelluksen sisältämät JavaScript-ohjelmakoodit sisältävät vain käyttäjille oikeasti tarkoitettua toiminnallisuutta, kun taas XSS-hyökkäyksellä web-palvelin valjastetaan hyökkääjän avuksi. Esimerkiksi verkkopankin sivulta ladattu JavaScript pääsee käsiksi verkkopankin lomakkeisiin yms. tietoihin, voisi XSS-haavoittuvuutta hyväksikäyttämällä injektoida koodia, joka muuttaa vaikkapa verkkopankin käyttäjän tekemien maksujen kohdetta.

XSS-hyökkäys muodostaa vakavan tietoturvauhan tiedon eheydelle. Se on erittäin yleinen ja helposti hyödynnettävä haavoittuvuus. [4.]

3.2.4 Epäturvallinen, suora viittaus objektiin

Epäturvallisella, suoralla viittauksella objektiin tarkoitetaan tilannetta, jossa sovellus ”vuotaa” sen sisäisesti käyttämien viittauksien arvoja käyttäjälle. Mikäli sovellus käyttää

näitä viitteitä siten, että käyttäjä voi päästä vaikuttamaan niiden arvoihin esimerkiksi lomakkeen syötteessä tai URL-osoitteen parametrin arvossa, niin tämä voi antaa hyökkääjälle pääsyn luvattomaan tietoon. Tilanne on hyvin samankaltainen kuin injektion kanssa, mutta poikkeaa siitä oleellisesti siinä, että käyttäjä ei välttämättä muuta sovellukselle lähetettävän tiedon tulkintaa vaan itse tietoa. Toisin sanoen käyttäjä voi manipuloida sovellusta käyttämään antamaansa viitettä ja saada sovellus paljastamaan sellaisia tietoja tai sallia sellaisten toimintojen suoritus, joihin käyttäjällä ei tavallisesti olisi pääsyoikeuksia. [4.]

3.2.5 Tietoturvan virheellinen konfigurointi

Tietoturvan virheellinen konfigurointi tarkoittaa kokonaisuudessaan kaikkien sovellukseen kuuluvien ja siihen liittyvien komponenttien, kuten palvelimien, tietokantojen, kehysten ja kirjastojen tietoturva-asetusten laiminlyöntiä. Tietoturvallisen konfiguraation luominen ja ylläpitäminen edellä mainituille komponenteille on erittäin tärkeää sovelluksen tietoturvan kannalta, sillä niiden laiminlyönti voi johtaa itse sovelluksen tai sen tietojen luvattomaan käyttöön.

Haavoittuvuus voi saada alkunsa esimerkiksi sovelluksen käyttämän web-palvelimen oletusasennukseen kuuluvista oletuskäyttäjätileistä, jotka ovat epähuomiossa päätyneet tuotantoversioon. Hyökkääjän on helppo kokeilla kirjautua sovellukseen oletuskäyttäjätilien tietoja käyttäen. Mikäli palvelin ilmoittaa itsestään nimen ja versionumeron, on tämän tyyppinen hyökkäys helppo automatisoida luomalla erillinen skripti, joka käy läpi tiettyä palvelinta käyttäviä sivustoja ja yrittää kirjautua oletustunnuksilla sisään.

Virheellisten konfiguraatioiden havaitseminen on yleensä helppoa, mutta vaatii usein sekä käyttäjäpohjaista että automatisoitua testausta.

3.2.6 Arkaluontoisen tiedon julkistaminen

Arkaluontoisen tiedon julkistaminen liittyy tiedon turvattomaan käsittelyyn ja suojaamiseen esim. sen siirrossa. Mikäli web-sovellus siirtää arkaluontoista tietoa verkon yli käyttäjälle ja käyttäjä sovellukselle, on tämä suojattava asianmukaisesti salauksella, eikä tämä yksin riitä, sillä salauksen on oltava riittävän vahva. Tyypillinen arkaluontoisen tiedon julkistamista koskeva hyökkäys on sessiokaappaus, jossa hyökkääjä saa salaamatonta liikennettä uhrinsa ja palvelimen välissä kuuntelemalla tietoonsa uhrin is-

tuntotunnuksen tai käyttäjätunnuksen ja salasanan, jonka jälkeen hyökkääjä voi hyödyntää kaappaamiaan arkaluontoisia tietoja haluamallaan tavalla.

Hyvä käytäntö onkin käyttää HTTP-protokollan sijaan HTTPS-protokollaa. Useat verkkosivut käyttävät nykyään HTTPS:aa ja SSL-salausta käyttäjien tietojen turvaamiseksi. Tämä ei kuitenkaan yksin riitä suojaamaan arkaluontoisen tiedon julkistamista koskevilta hyökkäyksiltä, sillä hyökkääjä voi keksiä muita keinoja saada tietoonsa arkaluontoista dataa. Useimmiten nämä keinot liittyvät enemmän web-sovelluksessa sijaitsevien muiden haavoittuvuuksien hyödyntämiseen.

3.2.7 Puuttuva funktiotason pääsynvalvonta

Puuttuvaan funktiotason pääsynvalvontaan liittyvä haavoittuvuus syntyy, kun web-sovellus ei varmenna käyttäjän oikeuksia päästä käsiksi suojattuihin toimintoihin. Vaikka web-sovellus piilottaisikin tälläisiin toimintoihin liittyvät elementit käyttöliittymästä, voi käyttäjä löytää tapoja päästä käsiksi näihin toimintoihin, kuten syöttämällä web-sovelluksen ylläpitäjä-valikkoon vievä URL-osoite selaimeen. Mikäli käyttöoikeutta ei varmenneta funktiotasolla, voi tämä johtaa määrättyjen toimintojen suorittamiseen luvatta.

Vaikka tiettyjä puuttuvan funktiotason pääsynvalvonnan haavoittuvuuksia voidaankin ehkäistä esim. asettamalla sovellukseen URL-tason pääsynvalvonta, niin ovela hyökkääjä voi keksiä muita tapoja saada web-sovellus kutsumaan suojattuja toimintoja. Tällöin hyökkääjä voisi suorittaa näitä toimintoja URL-tason suojauksesta huolimatta, ellei sovellukseen ole määritelty erikseen funktiotason pääsynvalvontaa.

Puuttuvaan funktiotason pääsynvalvontaan liittyvät haavoittuvuudet ovat yleisiä ja helposti hyödynnettävissä olevia ja muodostavat merkittävän uhan web-sovelluksille. [4.]

3.2.8 Puutteellinen pyynnön alkuperän tarkistus (CSRF)

Siinä missä verkkosivun rakennetta muuttavat hyökkäykset, kuten XSS, perustuivat selaimen palvelimeen kohdistuvan luottamuksen hyödyntämiseen, puutteellisen pyynnön alkuperän tarkastukseen liittyvät haavoittuvuudet syntyvät, kun web-palvelin luottaa sille saapuneiden pyyntöjen tulleen sille autentikoituneelta käyttäjältä.

Aiemmin mainittiin, että istuntotunniste yksilöi web-sovellukselle siihen liittyvän käyttäjän. Istuntotunniste lähetetään tyypillisesti käyttäjän selaimen pyyntöihin liitetyn HTTP-otsakkeen (header) mukana. Istuntotunniste toimii siis palvelimen ja käyttäjän selaimen välisenä jaettuna salaisuutena, jonka web-sovellusta käyttävän käyttäjän selaimen on tunnettava, jotta sen tekemät pyynnöt voitaisiin yksikäsitteisesti liittää tiettyyn käyttäjään. Mikäli esimerkiksi blog-sovellus tarjoaisi käyttäjän kaikkien luomien blog-kirjoitusten poistotoiminnon siten, että toiminto suoritettaisiin lähettämällä GET-pyyntö osoitteeseen *http://testiblogisivusto.fi/deleteAllOwnBlogs,* voisi toinen käyttäjä huijata sivustolle kirjautunutta uhria klikkaamaan kyseistä linkkiä, jolloin uhrin selain lähettää automaattisesti istuntotunnisteen pyynnön mukana kyseiseen URL-osoitteeseen. Jos blog-sivusto olisi haavoittuvainen CSRF-hyökkäykselle, niin toiminto ajettaisiin normaalisti läpi uhrin blogille.

CSRF-hyökkäystä voidaan ehkäistä sisällyttämällä erityinen salaisuus, CSRF-token, joka tulee olla jokaisen sellaisen web-sovellukselle lähtevän pyynnön yhteydessä, joka aiheuttaa jonkin tilaa muuttavan toiminnon suorituksen palvelimella. CSRF-token voidaan sisällyttää esim. HTML-sivulle, jolloin edellämainittu linkin klikkaaminen ei aiheuttaisi uhrin blogien poistoa, koska pyynnön mukana ei lähetetä CSRF-tokenia.

3.2.9 Tunnettuja haavoittuvuuksia sisältävien komponenttien käyttö

Tunnettuja haavoittuvuuksia sisältävien komponenttien, kuten kirjastojen ja sovelluskehysten käyttö on yllättävän yleinen haitta web-sovellusten tietoturvallisuudelle. Nimensä mukaisesti haavoittuvat komponentit muodostavat uhan sovelluksen tietoturvalle vieläpä siten, että niiden hyväksikäyttö voi olla erittäin helppoa, johtuen haavoittuvuuksien tunnettavuudesta. Web on pullollaan haavoittuvuuksien skannaustyökaluja, jotka voivat automatisoida hyökkäyksiä tällaisia haavoittuvia web-sovelluksia vastaan.

On tärkeää, että tähän uhkaan reagoidaan jo ohjelmiston kehitysvaiheessa käytettäviä kirjastoja ja kehyksiä valitessa. Järkevää on valita tietoturvallisen maineen, laajan käyttäjäkannan ja nopean päivittyvyyden omaavia komponentteja.

3.2.10 Varmistamattomat uudelleenohjaukset

Varmistamattomat uudelleenohjaukset tarkoittavat haavoittuvaisuuksia, jotka sallivat hyökkääjän määritellä, mille sivulle web-sovellus ohjaa käyttäjän. Haavoittuvaisuutta

voidaan hyödyntää mm. ohjaamalla verkkopankin asiakas verkkopankin sivua muistuttavalle, hyökkääjän hallitsemalle sivustolle. Mikäli hyökkäyksen kohde ei huomaa tapahtunutta, on riskinä, että tämä syöttää arkaluontoista tietoa, jotka paljastuvat hyökkääjälle. [4.]

Varmistamattomat uudelleenohjaukset ovat harvinaisin OWASP top 10 -listan haavoittuvuuksista. Tämän lisäksi ne on kohtuullisen helppo havaita. [4.]

Haavoittuvaisuus syntyy, kun haavoittuvainen palvelin käyttää uudelleenohjausosoitetta luodessaan käyttäjän määrittelemää syötettä uudelleenohjausosoitteen osana. Yksinkertainen tapa ehkäistä kyseistä haavoittuvuutta on suunnitella ja toteuttaa web-sovellus siten, että siinä käytetään mahdollisimman vähän uudelleenohjauksia, eikä niiden kohdeosoitteiden muodostamiseen käytetä käyttäjän määrittelemiä syötteitä ollenkaan.

4 Taustatiedot

Tässä luvussa käydään läpi yleiset taustatiedot insinöörityön myöhempien osien ymmärtämiseksi.

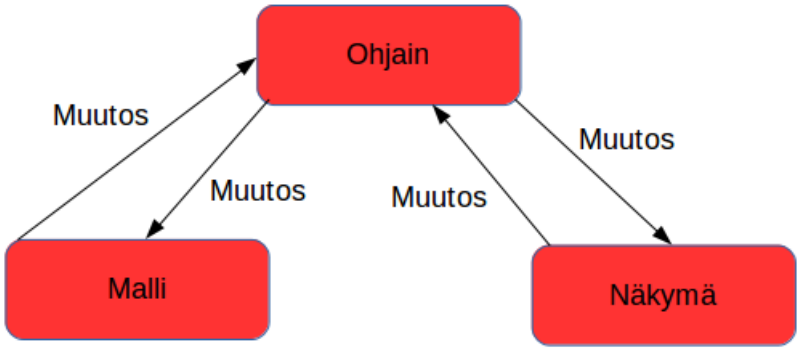
Koska insinöörityö keskittyy tietoturvaominaisuuksia tarjoavan sovelluskehyksen hyötyjen analysoimiseen, käydään seuraavissa luvuissa läpi muutama keskeinen sovelluskehyksien toimintaan liittyvä käsite. Luvussa 5 perehdytään tarkemmin käsitteiden merkitykseen varsinaisen sovellusohjelmoijan kannalta Spring-sovelluskehyksen kontekstissa.

4.1 MVC-malli

Ohjelmistokehityksessä eräs keskeinen sovellusten arkkitehtuurimalli MVC eli Model-View-Controller-malli on yleisesti hyvin suosittu käyttöliittymä- ja web-ohjelmoinnissa. Sen idea on jakaa sovellus kolmeen erilliseen osaan, jotka ovat

- malli, joka edustaa sovelluksen sisäistä tietorakennetta ja sen tilaa
- näkymä, joka vastaa sovelluksen sisäisen mallin esittämisestä käyttäjille
- ohjain, jonka tehtävä on välittää tietoa ja komentoja mallin ja näkymän välillä

Kuvasta 2 nähdään, kuinka ohjain toimii mallin ja näkymän välillä ikään kuin viestinvälittäjänä. Joissakin MVC-mallin toteutuksissa näkymä voi olla tietoinen mallista, mutta käyttää silti ohjainta tiedonvälitykseen mallille.



Kuva 2: MVC-mallin toimintaperiaate

MVC-mallin etuna on usein ohjelmiston selkeämpi rakenne ja vastuiden jakaminen, joka yleensä parantaa komponenttien uudelleenkäytettävyyttä ja testausta.

Javan annotaatiot ovat lähdekoodiin sisällytettävää metatietoa. Annotaatioilla voidaan merkitä eli *annotoida* niin luokkia, metodeja, muuttujia, parametreja kuin pakkauksia. Ne sisällytettiin osaksi Java-kieltä JDK-version 1.5 myötä. [16.] Kuvassa 3 on esitetty @Component-annotaatiolla varustettu luokka. Tässä tapauksessa annotaation tehtävä on kertoa Spring-kehykselle, että luokka tulee rekisteröidä sen IoC-säiliöön, johon tutustutaan tarkemmin luvussa 5.1.2.

```
@Component
public class HelloBean {
    public void sayHello() {
        System.out.println("Hello from HelloBean!");
    }
}
```

Kuva 3: Annotoitu luokka

Annotaatiot voidaan myös sisällyttää käännettyyn tavukoodiin, jolloin niiden sisältämää tietoa voidaan käyttää ajonaikaisesti reflektion avulla. Javan reflektio on tapa tarjota Ja-

va-kielisille ohjelmille pääsy niiden omaan ajonaikaiseen rakenteeseen. Sen avulla voidaan tarkastella ajonaikaisesti, mitä annotaatioita jollain luokalla tai metodilla on, tai vaikka mitä metodeja tietyllä luokalla on.

Annotaatioiden tärkeys tulee esiin, kun tarkastellaan Javan sovelluskehyksiä. Esimerkiksi Spring, Spring Security ja Hibernate käyttävät annotaatioiden tuomaa metatietoa hyväksi muuttaakseen luokkien, metodien ja muuttujien toimintaa ajonaikaisesti. Myöhemmin nähdään, miten annotaatioilla ja aspektipohjaisella ohjelmoinnilla voidaan toteuttaa metoditason tietoturva.

4.2 Riippuvuuden kääntämisen periaate (Dependency Inversion Principle, DIP)

Riippuvuuden kääntämisen periaate on Robert C. Martinin keksimä käsite, joka toimii oliopohjaisen ohjelmoinnin suunnittelupariaatteena. Yksinkertaisimmillaan se toteaa seuraavaa:

- Korkean tason moduulien ei tulisi riippua alemman tason moduuleista, vaan kummankin tulisi riippua abstraktiosta. [10.]
- Abstraktiot eivät saa riippua niiden toteutuksesta, mutta niiden toteutuksen tulisi riippua abstraktiosta. [10.]

Riippuvuuden kääntämisen periaatetta noudattamalla voidaan edesauttaa modulaaristen, laajennettavien ja testattavien ohjelmistojen rakentamista. [11.]

Kuten yllä todettiin DIP on vain periaate eikä kerro miten sen mainitsema riippuvuuden kääntäminen tulisi toteuttaa.

4.3 Käänteinen hallinta (Inversion of Control, IoC)

Käänteinen hallinta on suunnittelumalli, jolla viitataan ohjelmavirran, rajapintojen ja riippuvuuksien kääntämiseen. Sovelluskehysten toiminta perustuu juuri tähän suunnittelumalliin ja sovelluskehykset poikkeavat ohjelmakirjastoista siinä, että siinä missä käyttäjä tyypillisesti itse kutsuu ohjelmakirjastojen funktioita, niin sovelluskehys taas kutsuu käyttäjän kirjoittamaa ohjelmakoodia. [21.]

IoC on yleensä huonosti ymmärretty suunnittelumalli, ja usein siihen viitattaessa sen käsitetään koskevan ainoastaan ohjelmistokomponenttien riippuvuuksien määrittämisen hallintaa. Seuraavaksi tarkastelemme riippuvuusinjektiota, joka on konkreettinen tapa toteuttaa tämä osa-alue käänteisestä hallinnasta.

4.4 Riippuvuusinjektio (Dependency Injection, DI)

Riippuvuusinjektio on tapa toteuttaa käännetty luominen. Tämä tarkoittaa yksinkertaisuudessaan sitä, että jokin muita komponentteja toimintaansa tarvitseva luokka ei itse luo tarvitsemiaan riippuvuuksia, vaan ne injektoidaan. Koska luokka ei itse ole vastuussa riippuvuuksiensa luomisesta, sen ei tarvitse itse muuttua silloin, kun luokan tarvitsemien riippuvuuksien luominen muuttuu. Tämä johtaa löyhempiin kytköksiin luokkien välillä.

Injektoinnin eli viittauksen asettamisen toiseen olioon voidaan tehdä ainakin kolmella tavalla:

- Konstruktorin kautta tehtävä injektointi, jossa riippuvuudet annetaan kohdeluokan konstruktorin kautta parametrina.
- Setterin kautta tehtävät injektointi, jossa riippuvuudet asetetaan kohdeluokan setterin kautta annettavana parametrina.
- Rajapinnan kautta tehtävä injektointi, jossa riippuvuuksia tarvitsevan luokan tulee toteuttaa tietty rajapinta, jonka kautta riippuvuudet asetetaan.

Myöhemmin luvussa 5 nähdään, miksi nämä käsitteet ovat keskeisiä Spring-sovelluskehyksessä ja näin ollen myös Spring Security -kehyksessä.

4.5 Aspektipohjainen ohjelmointi (Aspect Oriented Programming, AOP)

Aspektipohjainen ohjelmointi on Xeroxin Palo Alton tutkimuskeskuksessa kehitetty paradigma, jonka tarkoituksena on paikata olio-ohjelmoinnin paradigman puutteita ns. läpileikkaavien ominaisuuksien modularisoimisessa. [12.]

Läpileikkaavilla ominaisuuksilla tarkoitetaan toimintoja, joita käytetään useissa luokissa. AOP:n avulla nämä ominaisuudet voidaan kerätä yhteen sijaintiin (aspektiin), jolloin

ohjelmakoodista tulee siistimpää ja näihin ominaisuuksiin kohdistuvia muutoksia voidaan helpommin hallita. Tyypillisiä yrityssovelluksissa esiintyviä läpileikkaavia ominaisuuksia ovat

- tietokantatransaktioiden hallinta
- lokitus
- monitorointi
- tietoturva
- virheenkäsittely.

Aspektipohjaisen ohjelmoinnin keskeinen voi tuntua aluksi vaikealta ymmärtää, mikäli on perehtynyt ainoastaan tyypillisimpiin ohjelmointiparadigmoihin, kuten proseduraaliseen- tai olio-ohjelmoparadigmaan. Aspektipohjainen ohjelmointi laajentaa olio-ohjelmointiparadigman käsitteitä, kuten luokkia ja metodeita yms. seuraavilla käsitteillä:

- liitoskohta (join point)
- liitoskohtamalli (join point model)
- liitoskohtamäärittely (pointcut)
- kehote (advice)
- aspekti (aspect)
- punonta (weaving).

Liitoskohta on ohjelman vuossa tarkoin määritelty kohta, joihin poikkileikattavassa järjestelmässä halutaan aspekteilla lisätä uutta toiminnallisuutta. Liitoskohdan on oltava täsmällisesti määritelty ja sen tarkan mallintamisen määrittelee aspektiohjelmointikieli, joista yleisin on AspectJ. AspectJ:n avulla voidaan liitoskohdiksi määritellä Java-kielisissä ohjelmissa esimerkiksi tietyn metodin suorittamista tai tiettyyn attribuuttiin viittaamista. Se siis viittaa tarkoin määriteltyyn ohjelman suorituksenaikaiseen tai käännöksenaikaiseen rakenteeseen ja liitoskohdat jaetaankin staattisiin ja dynaamisiin.

Kukin aspektiohjelmointikieli määrittelee oman mallinsa sille, millaisia kohtia ohjelman suorituksessa voidaan laajentaa aspekteilla. Tämän mallin nimi on *liitoskohtamalli*. Tyypillisiä liitoskohtia ovat Java-kielisissä ohjelmissa mm. metodin kutsut sekä instanssimuuttujien ja luokkien alustamisen käsittely.

Liitoskohtamäärittelyllä voidaan koota yhteen useita erikseen määriteltyjä liitoskohtia. Liitoskohtamäärittelyt voivat sisältää yhteen kokoamiensa liitoskohtien ajonaikaisia tietoja, kuten metodien parametrien arvoja.

Aspektilla tarkoitetaan läpileikkaavien ominaisuuksien modularisointia, joka voi oliopohjaisissa kielissä tarkoittaa yksinkertaisimmillaan annotoitua luokkaa, joka sisältää läpileikkaavan ominaisuuden toteuttavan toiminnallisuuden.

Kehotteella kuvataan liitoskohtaan lisättävää toiminnallisuutta, kuten esimerkiksi lokituksesta vastaavaa koodia. Siinä missä kehote ja liitokohta kuvaavat, *mitä suoritetaan* (advice) ja *milloin suoritetaan* (pointcut), niin aspektin määritelmä käsittää nämä molemmat.

Punonnalla viitataan prosessiin, jolla aspektit sidotaan osaksi suoritettavaa ohjelmakoodia. Punominen voidaan suorittaa kahdella eri tavalla: staattisesti ja dynaamisesti. Staattinen punonta suoritetaan ohjelman käännösaikana ja dynaaminen punonta taas ohjelman ajonaikaisesti. Staattisella punonnalla aikaansaadaan tyypillisesti dynaamista punontaa nopeampaa ohjelmakoodia. Esimerkiksi Spring AOP toteuttaa punonnan ajonaikaisesti. [6.]

5 Käytetyt tekniikat ja ohjelmistokehykset

Tämän luvun alikappaleissa käydään läpi insinöörityössä käytetyt ohjelmistokehykset ja tekniikat yksityiskohtaisesti.

5.1 Maven

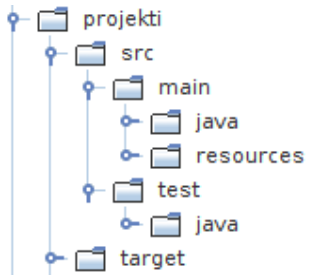
Maven on Java-kielisille ohjelmille Apache Software Foundationin kehittämä ohjelmistoprojektien hallintaan tarkoitettu työkalu. Se on yksi suosituimmista Java-pohjaisten ohjelmistoprojektien projektinhallintatyökaluista ja sillä voidaan käyttää projektinhallintaan liittyvien toimintojen automatisoimiseen, kuten

- sovelluksen kääntämiseen
- automatisoitujen yksikkö- ja integraatiotestien ajamiseen
- projektin ohjelmistokomponenttien riippuvuuksien hallintaan

- testiraporttien luomiseen.

Mavenia käyttävät projektit konfiguroidaan erillisellä XML-kielisellä POM-tiedostolla (Project Object Model). POM-tiedosto sisältää myös tiedon projektille määritellyistä riippuvuuksista sekä sinne voidaan määritellä projektin rakentamiseen, testaamiseen ja ajamiseen liittyvät vaiheet.

Työkalu on erittäin suosittu Java-kehittäjien keskuudessa, sillä se tarjoaa Convention Over Configuration -periaatteen mukaisen tavan käyttää tiettyjä konventioita ja oletuksia projektin konfiguroimiseen. Kuvassa 4 on esitetty Mavenin olettama projektin hakemistorakenne.



Kuva 4: Maven-projektin oletushakemistorakenne

Näistä oletuksista voi poiketa, mutta se on kerrottava Mavenille erikseen POM-tiedostossa. Maven on myös hyvin tuettu eri kehitysympäristöjen keskuudessa. Sitä tukevat ainakin Eclipse, NetBeans ja IntelliJ Idea. [13.]

Kuvassa 5 on esitetty Maven-projektin yksinkertaisin mahdollinen POM-tiedosto. Siinä määritellään projektin yksilöllisesti tunnistava groupId-tunniste, jona yleensä käytetään projektin omistaman organisaation hallinnoimaa domain-nimeä käänteisessä järjestyksessä.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>fi.omayritys.testiapp</groupId>
  <artifactId>oma-testisovellus</artifactId>
  <version>1</version>
</project>
```

Kuva 5: Minimaalinen POM-tiedosto. [14.]

Kuvan 5 esimerkissä on käytetty kuvitteellista testiapp.omayritys.fi -alidomainia. ArtifactId-tunniste ilmaisee projektista mahdollisesti luotavan jar-tiedoston nimen ilman versionumeroa. Version-tunniste taas merkitsee projektin versionumeron. [14.]

Kuten edellä todettiin, useat kehitysympäristöt tarjoavat tuen Mavenille ja sen käytölle graafisesta käyttöliittymästä, mutta siihen on myös olemassa komentorivityökalu, jolla voidaan käsin ajaa haluttuja toimintoja. Mikäli kuvan 4 mukaisen POM-tiedoston omaava Java-kielinen projekti noudattaa Maven-projektin oletushakemistorakennetta (kuva 1), niin kyseinen projekti voitaisiin kääntää ja paketoida ajettavaksi jar-tiedostoksi komennolla *mvn package.*

Mikäli Maven-työkalua käyttävä projekti tarvitsee muita ohjelmakirjastoja tai sovelluskehyksiä toimiakseen, ne voidaan määritellä pom.xml-tiedostoon kertomalla niiden groupId:n ja artifactId:n. Versionumeroa ei ole pakko määritellä, jolloin Maven päättelee sen itse ja käyttää oletuksena uusinta versiota. Riippuvuudet ladataan oletuksena Mavenin omasta pakettivarastosta ja ne tallennetaan käyttäjän kotihakemistoon .m2-hakemiston alle.

Kuvassa 6 on esitetty tarvittavat määritykset Spring-sovelluskehyksen lataamiseksi Maven-projektin käyttöön. Maven voi hakea useita riippuvuuksia, vaikka projektin pom-xml-tiedostossa olisi vain yksi riippuvuus. Tätä kutsutaan transitiivisten riippuvuuksien selvittämiseksi. [15.]

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
</dependency>
```

Kuva 6: Spring-sovelluskehyksen määritteleminen maven-projektiin

5.2 Spring

Spring on avoimen lähdekoodin sovelluskehys, joka toimii suositulla Java-ohjelmistoalustalla. Se on yksi suosituimmista Java-ohjelmistokehyksistä ja sillä on hyvin laaja käyttäjäkanta ja se on aktiivisessa kehityksessä.

Spring-sovelluskehyksestä julkaistiin vuonna 2002 ensimmäinen versio tarjoamaan helppokäyttöisempi ja modulaarisempi ohjelmointimalli Java EE -standardille, sillä Java EE -pohjainen sovelluskehitys oli tuolloin erityisen hankalaa ja Java EE -standardia hyödyntävät ohjelmat olivat tiivisti riippuvaisia Java EE:stä [4].

Spring-sovelluskehys tukee ohjelmistokehitystä Java-alustalla tarjoamalla valmiin tuen monille sellaisille toiminnoille, joita lähes jokainen sovellus vaatii, antaen sovellusohjelmoijille mahdollisuuden keskittyä sovelluksiensa varsinaisen businesslogiikan toteuttamiseen.

Spring-sovelluskehys toimii myös perustana usealle Springin sisarprojektille, kuten Spring Integration, Spring Batch, Spring Security ja Spring Data. Kaikkia Spring-sovelluskehyksen projekteja hallinnoi ja ylläpitää yhdysvaltalainen ohjelmistoyhtiö Pivotal [4].

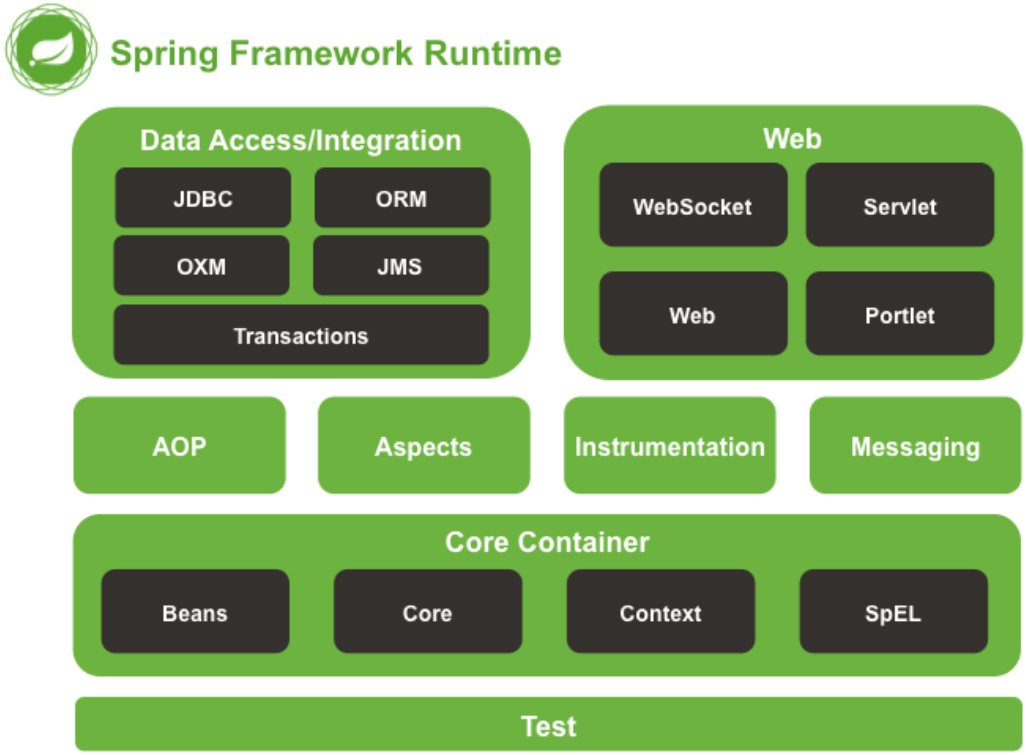
Sovelluskehys tarjoaa selkeän ja laajennettavan ohjelmointimallin, jonka keskeisimpiä tekniikoita ovat riippuvuusinjektion (DI, Dependency Injection) toteuttaminen IoC-suunnitteluperiaatetta käyttäen, deklaratiivinen ohjelmointi käyttäen hyödyksi aspektipohjaista ohjelmointia (AOP, Aspect-Oriented Programming) ja Java-kielen annotaatioiden käyttäminen osana konfigurointia.

Spring sisältää erityisen IoC-säiliön (IoC container), jonka tarkoituksena on pitää sisällään sovelluskehyksen ajonaikana hallinnoimat Java-oliot, joiden elinkaaresta ja kytkemisestä toisiinsa tämä on vastuussa.

Tässä insinöörityössä oletetaan, että käytössä on Spring Framework 4.3.4 tai uudempi, joskin insinöörityössä esitellyt asiat ovat suurimmaksi osin sovellettavissa aiempiin sovelluskehyksen versioihin.

5.2.1 Rakenne

Spring-sovelluskehys on jaoteltu ominaisuuksien puolesta noin 20 eri moduuliin. Moduulit on ryhmitelty karkeasti ottaen seitsemään eri ryhmään kuvan 7 mukaisesti.



Eri moduuliryhmien toiminnallisuudet on eritelty taulukon 1 osoittamalla tavalla.

Taulukko 1. Spring-sovelluskehyksen moduuliryhmien toiminnot.

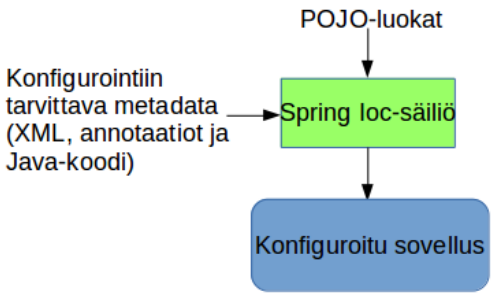
Moduuliryhmä	**Selitys**
Core Container	Sisältää Beans, Core, Context ja SpEL -moduulit, jotka määrittelevät sovelluskehyksen keskeiset toiminnallisuudet tarjoamalla sovellusohjelmoijan käyttöön IoC-säiliön ja SpEL-ilmauskielen, jolla voidaan hallita olioita ajonaikaisesti.
AOP	Sisältää tuen Aspektipohjaiselle ohjelmoinnille (AOP, Aspect-Oriented Programming), joka mahdollistaa usein käytettyjen toimintojen, kuten tietokantatrasaktioiden, virheenkäsittelyn tai lokituksen modularisoinnin ja kapseloinnin.
Data Access/Integ-	Sisältää tiedon- ja tietokantojen hallintaa tukevat moduulit, kuten

ration	JDBC ja ORM, sekä myös transaktioiden hallintaan, viestinvälitykseen ja olio-XML -muunnokseen liittyvät moduulit (Transactions, OXM, JMS).
Web	Sisältää Servlet, Web, WebSocket ja Portlet -moduulit, jotka tarjoavat tuen web-pohjaisten sovellusten kehitykseen. Tärkeimpinä näistä moduuleista Servlet-moduuli sisältää MVC-mallin mukaisen Spring Web MVC -sovelluskehyksen ja tuen REST-palveluiden toteuttamiselle, kun taas WebSocket-moduuli kattaa Websocket-rajapinnan ja sen integraation Spring-sovelluskehyksen kanssa.
Aspects	Aspects-moduuliryhmä lisää Spring-sovelluskehykseen tuen aspektipohjaisen ohjelmoinnin kirjastolle, AspectJ:lle.
Instrumentation	Instrumentation-ryhmä tarjoaa sovelluksen suorituskyvyn ja resurssien käyttöön mittaamiseen liittyvän tuen. Erityinen Instrument Tomcat -moduuli pitää sisällään Tomcat-sovelluspalvelimen suorituskykyä mittaavan rajapinnan tuen.
Messaging	Sisältää tuen viestinvälitysarkkitehtuurille ja protokollille.
Test	Sisältää tuen yksikkö- ja integraatiotesteihin.

Seuraavaksi perehdytään tarkemmin taulukon 1 ensimmäisen rivin (Core Container) moduuleihin, erityisesti IoC-säiliöön.

5.2.2 Springin IoC-säiliö

IoC-säiliö on Spring-sovelluskehyksen ydinkomponentteja, jonka toiminnan varaan koko kehyksen toiminta rakentuu. Spring-kehyksen Beans ja Context -moduulit puolestaan muodostavat IoC-säiliön perustan. IoC-säiliön toiminta on esitetty kuvassa 8.



Kuva 7: IoC-säiliön toimintaa havainnollistava kuva.

Kuten kuva 8 esittää, IoC-säiliön toiminta-ajatus on injektoida sovellusohjelmoijan kirjoittamiin IoC-säiliön hallitsemiin tavallisiin Java-luokkiin (POJO, Plain Old Java Objects) niiden tarvitsemat riippuvuudet, jolloin sovellusohjelmoijan ei tarvitse itse luoda niitä käsin. Tämä säästää valtavasti aikaa ja vaivaa sekä poistaa turhien virheiden mahdollisuutta. Vaatimuksena kuitenkin on, että sovellusohjelmoijan tulee konfiguroida Spring lataamaan Springin IoC:ta riippuvuuksien injektointiin käyttävät luokat myös IoC-säiliöön.

Springille täytyy myös kertoa joko Java- tai XML-pohjaisella konfiguraatiolla miten se injektoi riippuvuudet ja ottaa halutut oliot käyttöön ajettavaan sovellukseen. Nykyään suositaan Java-pohjaista konfiguraatiota, jossa käytetään annotaatioita riippuvuuksien määrittelemiseen, mutta kumpaakaan tapaa ei voida objektiivisesti pitää toista parempana, sillä siinä missä XML-pohjainen konfigurointi vaatii yleensä enemmän työtä, se myös jättää ohjelmakoodin siistimmäksi, eikä XML-konfiguraatioon kohdistuvat muutokset vaadi sovelluksen uudelleen kääntämistä muutosten jälkeen. Annotaatiot taas liitetään suoraan ohjelmakoodiin, jolloin ne ovat selkeästi näkyvissä niihin liittyvissä luokissa ja ne ovat XML:n verrattuna tiiviimpi esitystapa. [6.]

Spring tarjoaa kaksi eri abstraktiotason rajapintaa, joiden avulla voidaan hallita Spring-kehyksen IoC-säiliötä: BeanFactory ja ApplicationContext. BeanFactory sallii matalamman tason rajapinnan käytön IoC-säiliölle. ApplicationContext on tavallisesti sovellusohjelmoijalle riittävä ja helppokäyttöisempi rajapinta, jonka avulla IoC-säiliötä voidaan manipuloida.

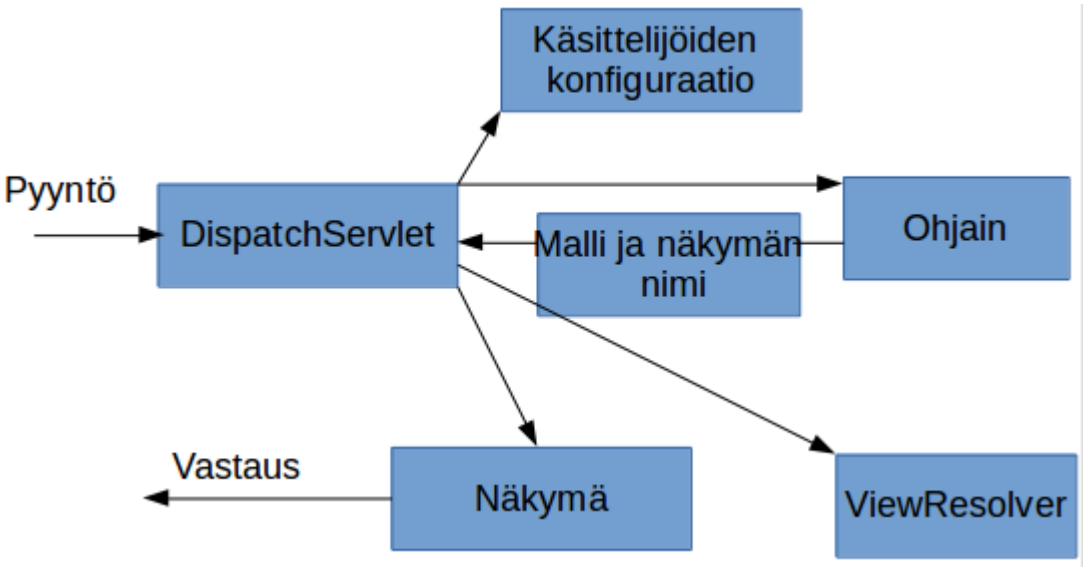
5.2.3 Springin AOP-moduuli

Spring AOP -moduuli tarjoaa sovellusohjelmoijan käyttöön Spring-kehyksen oman AOP-toteutuksen. Toteutus poikkeaa muista AOP-toteutuksista, kuten AspectJ:sta siinä, että se pyrkii tarjoamaan ratkaisuja yrityssovelluksissa esiintyviin ongelmiin eikä olemaan kattavin AOP-toteutus.

5.2.4 Spring Web MVC -kehys

Spring Web MVC (Lyhennettynä Spring MVC) on MVC-arkkitehtuurin hieman mukaillen toteuttava Springin web-sovelluskehys. Spring MVC käyttää niin sanottua *etuohjainta* (Front controller) käyttäjän pyyntöjen käsittelyn delegoimiseen varsinaisille ohjaimille.

Etuohjaimena toimiva Spring-kehyksen *DispatchServlet* delegoi sovellukselle saapuneet pyynnöt eteenpäin ohjaimille, jotka käsittelevät pyynnön. Ohjain on vastuussa *mallin* luomisesta ja *näkymän* nimen palauttamisesta etuohjaimelle (kuva 9).



Kuva 8: Spring MVC:n etuohjain. [5, s. 132.]

Kun etuohjain saa tietoonsa pyynnön vastauksena palautettavan näkymän nimen varsinaiselta ohjaimelta, se kysyy Springin WebApplicationContext:n konfiguroidulta ViewResolver-luokan instanssilta näkymän nimeä vastaavaa näkymää ja välittää näkymälle ohjaimelta saadun mallin tiedot. Näkymä renderöi annetun mallin tiedot ja palauttaa tämän etuohjaimelle, joka luo tästä käyttäjälle palautettavan vastausobjektin. [5, s. 132.]

5.2.5 Lopuksi

Kuten viime luvussa nähtiin, Spring on geneerinen sovelluskehys ja sisältää runsaita toimintoja ja integraatioita eri teknologioiden välillä. Sen sisältämä Spring Web MVC -moduuli pohjautuu Servlet-pohjaiseen teknologiaan, jota käytetään yhä laajalti mm. Java EE -standardiin nojautuvissa sovelluskehyksissä.

5.3 Spring Boot

Spring Boot kehitettiin vastaamaan Spring-sovelluskehystä vaivanneisiin ongelmiin, jotka liittyivät kehyksen vaikeaan käyttöönottoon ja XML-konfiguraation määrään [9].

Spring Boot on ollut keskeisessä osassa Spring-sovelluskehyksen suosion kasvattamisessa siitä syystä, että sillä voidaan laatia ajettavia web-sovelluksia ilman erillisen sovelluspalvelimen konfiguroimista, sillä Spring Boot:illa laaditut web-sovellukset sisältävät oletuksena itsenäisesti ajettavan niin sanotun *embedded* eli sulautetun Tomcat-sovelluspalvelimen, mutta se tukee myös Jetty- ja Undertow -sovelluspalvelimia [9].

Spring Boot sisältää valmiita starter-paketteja, jotka ovat yhdistelmä toiminnallisuutta tarjoavia kirjastoja ja sovelluskehyksiä ja niihin liittyviä järkeviä oletuksia tarjoavia asetuksia. Käytettäessä Maven-projektinhallintatyökalua riittää, että projektin pom.xml-tiedostoon määritellään riippuvuudet näihin starter-paketteihin. Tällöin projektin käyttöön voidaan ladata täysin toimivat oletusasetukset sisältävä sulautettu (embedded) Tomcat-sovelluspalvelin ja web-sovellusten kehitykseen Spring MVC -moduulilla tarvittavat riippuvuudet. Valmiit asetukset eivät kuitenkaan estä Spring Bootin käyttäjiä tarjoamasta omia asetuksiaan, vaan Spring Bootin starter-pakettien mukanaan tuomat oletukset väistyvät käyttäjän määrittelemien omien asetusten tieltä. [9.]

5.4 Spring Security

Spring Security on Pivotalin hallinnoima sovelluskehys, joka tähtää järeän ja mukautettavan autentikointi- ja auktorisoinnin toteuttamiseen Java-sovelluksille. Se on Spring-sovelluskehyksen eräs lukuisista sisarprojekteista, ja käytännössä suuri osa Spring-kehyksellä laadituista sovelluksista käyttävät Spring Securityä juurikin sen vuoksi, koska Spring Security pohjautuu Spring-sovelluskehyksen tarjoamaan ohjelmointimalliin ja koska sen tarjoamat web-tietoturvaominaisuudet pohjautuvat Spring Web MVC:n käyttämään Servlet-teknologiaan [8].

Autentikointitasolla Spring Security tukee integraatiota autentikoitumiseen mm. seuraavilla teknologioilla:

- Lomakepohjainen autentikointi (Yksinkertaisten sovellusten tarpeisiin, esim. käyttäjätunnus ja salasana -kentät lomakkeella.)
- OpenID (Avoin ja hajautettu tunnistusprotokolla, jota käyttävät mm. Google, Amazon, Flickr, Paypal ja Steam.)
- LDAP (Lightweight Directory Access Protocol. Käytännössä autentikoidaan esim. Windows-ympäristöissä AD-palvelinta vasten käyttäen LDAP-protokollaa.)
- Kerberos-protokolla

- CAS Single Sign-On
- JAAS (Java Authentication and Authorization Services, Java SE -alustan toteutus PAM-moduuleista.)

Edellä mainittujen lisäksi on mahdollista toteuttaa täysin räätälöityjä autentikointitapoja, jotka hyödyntävät vaikkapa kaksivaiheista tunnistusta tai biometristä tunnistetta.

Taulukossa 2 on esitetty kehyksen keskeisimpiä käsitteitä.

Taulukko 2: Spring Securityn keskeisimpiä luokkia ja rajapintoja.

Rajapinnan tai olion nimi	**Kuvaus**
Authentication	Toimii käyttäjän ”kulkulupana” sovelluksessa, joka kertoo, kuka tämä on ja mitä käyttöoikeuksia tällä on.
GrantedAuthority	Kuvaa käyttäjälle myönnettyä valtuutusta. Authentication-rajapinnan toteuttaja voi sisältää useita tällaisia.
SecurityContext	Pitää sisällään nykyisen säikeen Authentication-rajapinnan toteuttajaobjektin ja mahdollisesti muuta pyyntökohtaista tietoa.
SecurityContextHolder	Tarjoaa hallitun pääsyn kulloinkin sovellusta käyttävän käyttäjän Authentication-objektiin.
UserDetails	Pitää sisällään tarvittavat tiedot Authentication-objektin rakentamiseen.
UserDetailsService	Palvelu, jonka avulla voidaan ladata UserDetails-objektin käyttäjätunnuksella. Käytetään autentikoinnissa.

Käyttäjän yrittäessä autentikointia Spring Securityn turvaamassa sovelluksessa UserDetailsService on vastuussa autentikointia yrittävän käyttäjän tietojen lataamisesta. Se sisältää metodin nimeltä loadUserByUsername. UserDetailsService on rajapinta, jolloin sille voidaan antaa useita toteutuksia. Tämä takaa joustavuuden autentikointia toteutettaessa. [8.]

UserDetails erottaa käyttäjän määrittelemän käyttäjän tiedot Spring Security -kehyksen Authentication-objektin tiedoista. Toisin sanoen UserDetails kertoo sen, mitä tietoja Spring Security vähintään tarvitsee, jotta käyttäjälle voidaan antaa ”kulkulupa” (Authentication-objekti) sovelluksessa. [8.]

6 Autentikoinnin toteutus Spring Securityllä

Tässä luvussa käydään läpi lyhyen esimerkin avulla käytännön toteutuksen LDAP-pohjaisen autentikoinnin toteuttamiseksi.

Käyttäjätunnus ja salasana voidaan myös ulkoistaa sovelluksen ulkopuolelle. Usein tämä on toivottavaa jo siitä syystä, että näin ollen yrityskäyttöön suunnattuihin sovelluksiin ei tarvita jokaiseen luoda omia tunnuksia, vaan niitä voidaan käyttää samoilla tunnuksilla, joilla työntekijä kirjautuu työasemalleen. Spring Security kykenee käyttämään LDAP-protokollalla ladattuja käyttäjätietoja, kuten käyttäjätunnusta ja salasanaa.

Esimerkissä käytetään Springiin sisällytettyä ApacheDS-palvelinta yksinkertaisuuden vuoksi. Todellisuudessa Spring Security tukee ”oikeita” LDAP-palvelimia ja myös Microsoftin AD-palvelinta, joskin vaatii Microsoftin AD-palvelimen tukemiseen erillisen moduulin. Seuraavaksi käydään läpi vaadittavat konfiguraatiot POM-tiedostoon ja Spring Securityn konfiguraatioon.

Jotta Apache DS ja Spring-kehyksen LDAP-tuki saadaan käyttöön, täytyy projektin pom.xml-tiedostoon määritellä kuvan 10 mukaiset riippuvuudet. Spring Boot -kehyksen Spring Securityn Starter-pakettia hyödyntämällä voidaan Apache DS konfiguroida Spring-sovellukseen lähes automaattisesti.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-ldap</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.directory.server</groupId>
    <artifactId>apacheds-server-jndi</artifactId>
    <version>1.5.5</version>
</dependency>
```

Kuva 9: Apache DS- ja LDAP-tuen Spring Securitylle tarjoavat riippuvuudet.

Seuraavaksi täytyy luoda Apache DS:n konfiguraatio. Tämä voidaan tehdä Java-pohjaisesti ylikirjoittamalla WebSecurityConfigurerAdapter-luokan aliluokassa configGlobal-metodi kuvan 11 mukaisella tavalla.

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .ldapAuthentication()
        .userDnPatterns("uid={0},ou=people")
        .groupSearchBase("ou=groups")
        .contextSource().ldif("classpath:test-server.ldif");
}
```

Kuva 10: LDAP-autentikoinnin käyttöönotto Java-pohjaisesti.

Metodille injektoidaan käyttöön AuthenticationManagerBuilder-luokan instanssi, jolle kerrotaan, että halutaan käyttää LDAP-autentikointia ja että LDAP-palvelimen konfiguraatio ladataan tiedostosta ”test-server.ldif”, joka sijaitsee luokkapolussa.

Muut asetukset määrittelevät LDAP:n liittyviä yksityiskohtia, joihin ei mennä tarkemmin. Lisää tietoa LDAP-protokollasta löytyy IETF:n sivuilta osoitteesta https://tools.ietf.org/html/rfc4511. Oleellista on, että sovelluksen käyttäjien nimet ja salasanat sijaitsevat LDAP-palvelimella, eivätkä sovelluksessa tai sovelluksen kannassa.

7 Pääsynhallinnan toteutus Spring Securityllä

Web-sovelluksessa pääsynhallinta voidaan toteuttaa usealla eri tasolla, joita ovat

- Näkymätaso eli se, mitä käyttäjä näkee web-sivulla. Esim. ylläpitäjä näkee linkin ylläpitovalikkoon, kun tavalliset käyttäjät eivät.
- URL-taso. Tietyt sovellukseen liittyvät URL-osoitteet suojataan käyttöoikeustarkistuksin.
- Metoditaso. Tietyt metodit suojataan sovelluksen siten, että vain niihin oikeutetut henkilöt voivat niitä käyttää.
- Objektitaso. Sovelluksessa voidaan rajata vastauksena palautettavasta tulosjoukosta näytettäväksi vain ne, johon käyttäjällä on oikeudet.

Tässä luvussa syvennytään näistä näkymätason, URL-tason ja metoditason suojauksen toteuttamiseen Spring Security -sovelluskehyksellä. On tärkeää huomata, että pääsynhallinta on täysin eriytetty käyttäjän autentikoinnista. Tämä sallii autentikointitavasta riippumattoman pääsynhallinnan toteutuksen sovellukseen.

7.1 Näkymätason pääsynhallinta

Spring Security -sovelluskehys voidaan integroida käytettävissä olevaan näkymäteknologiaan, kuten JSP:hen tai Thymeleafiin. Thymeleaf on palvelinpuolen templatointimoottori, joka sisältää integraation Spring-kehykseen. [21.]

Thymeleaf sisältää erillisen nimiavaruuden Spring Security -kehyksen tuelle. Tuki otetaan käyttöön määrittelemällä riippuvuus Thymeleafin Spring Security -integraatiomoduuliin projektin POM-tiedostoon kuvan 12 mukaisesti.

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity4</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>
```

Kuva 11: Thymeleafin ja Spring Securityn integraation välinen riippuvuus.

Thymeleaf-extras-springsecurity4-moduuli laajentaa Thymeleafin toimintaa siten, että sen luonnollisissa näkymätemplateissa voidaan määritellä esim. käyttäjäroolin mukaan käyttäjälle palautettavaan HTML-sivuun mukaan tulevat elementit. Tämä mahdollistaa ylläpitokäyttäjille tarkoitettujen HTML-elementtien poisjättämisen tavalliselle käyttäjälle tarkoitetusta vastauksesta. Tämä tehdään käyttämällä Thymeleafin sivupohjassa erityistä sec:authorize-tagia, jolle voidaan antaa boolean-arvon palauttava SpEL-ilmaus. Kuvassa 13 on esitetty uloskirjautumiseen tarkoitetun linkin sisältävä div-elementti, joka sisällytetään käyttäjälle lähetettävään vastaukseen vain, jos sivulla vieraileva käyttäjä on kirjautunut.

```
<div sec:authorize="#{isAuthenticated()}">
  <a th:href="@{/logout}">Log out</a>
</div>
```

Kuva 12: Ehdollisesti renderöitävä HTML-elementti.

Vaikka näkymätason tietoturva voi parantaa tietoturvaa ja varsinkin vähentää tarpeettoman tiedon lähetystä palvelimen ja käyttäjän välillä, ei siihen tule luottaa. Seuraavaksi tarkastellaan URL-tason pääsynhallintaa.

7.2 URL-tason pääsynhallinta

Jos web-sovelluksen halutaan rajaavan tiettyjen URL-osoitteiden käyttöä käyttäjän roolin mukaan, täytyy siihen luoda asetukset toimintoa varten. Java-kielisten asetusten tekeminen onnistuu luomalla Spring IoC-säilön hallitsema Java-konfiguraation sisältävä aliluokka WebSecurityConfigurerAdapter:sta.

Tämän lisäksi luokalle täytyy määritellä configure-metodin ylikirjoituksessa haluttuja URL-osoitteita koskevat määritykset, joiden pohjalta pääsyä niihin rajoitetaan. Konfiguraatioluokalle täytyy myös antaa @EnableWebSecurity-annotaatio, tai täytyy annotaatio olla merkittynä johonkin muuhun luokkaan.

7.3 Metoditason pääsynhallinta

Metoditason pääsynhallinnalla tarkoitetaan käyttöoikeuksien tarkistamista ennen metodin suoritusta. Tämä tarjoaa URL-tason pääsynhallintaa hienojakoisemman tavan hallita pääsyä sovelluksen ominaisuuksiin, sillä sen avulla voidaan määritellä käyttöoikeuksia yksittäisten metodien tasolla, eikä tiettyyn URL-osoitteeseen kohdistuvien pyyntöjen perusteella.

Spring Security -kehykselle määritellään erillisillä annotaatioilla halutuille metodeille tai kokonaisille luokille, millä käyttöoikeuksilla niitä voidaan kutsua. Mikäli annotaatio on annettu luokkatasolla, koskee se tällöin jokaista luokan julkista metodia. Taulukossa 3 on esitetty Spring Securityn tarjoamat valmiit annotaatiot metoditason käyttöoikeuksien tarkistukseen.

Taulukko 3: Spring Securityn metoditason käyttöoikeuksien tarkistukseen liittyvät annotaatiot.

Annotaatio	**Toiminta**
@PreAuthorize	Suorittaa käyttöoikeuksien tarkastuksen ennen metodin suorittamista.
@PostAuthorize	Tarkistaa käyttöoikeudet vasta metodin suorituksen jälkeen.

Kukin taulukossa 3 esiintyvistä annotaatioista parametrisoidaan Springin omalla SpEL-kielellä määritellyllä ilmauksella (expression).

Käyttöoikeuksien määrittäminen tehdään käyttämällä SpEL-ilmauksia. Spring Security määrittelee joitain yleisesti käytettyjä ilmauksia, joita voidaan käyttää. Taulukossa 4 on esitetty muutamia esimerkkejä näistä Spring Securityyn valmiiksi määritellyistä SpEL-ilmauksista.

Taulukko 4: Spring Securityyn valmiiksi määriteltyjä SpEL-ilmauksia. [8.]

SpEL-ilmaus	**Parametrit**	**Kuvaus**
hasRole([role])	Rooli merkkijonona.	Palauttaa toden, jos nykyinen käyttäjä omaa määritellyn roolin.
hasAuthority([authority])	Auktoriteetti merkkijonona.	Palauttaa toden, jos nykyinen käyttäjä omaa määritellyn auktoriteetin.
isAuthenticated()	-	Palauttaa toden, jos nykyinen käyttäjä on autentikoitu.
IsFullyAuthenticated()	-	Palauttaa toden, jos nykyinen käyttäjä on autentikoitu ilman remember me -toimintoa.
IsRememberMe()	-	Palauttaa toden, jos nykyinen käyttäjä on autentikoitu remember me -toiminnon avulla.

Kuten taulukosta 4 nähdään, tarjoaa Spring Security useita valmiita SpEL-ilmauksia käyttöoikeuksien rajaamiseen. Tämän lisäksi sovellusohjelmoija voi määritellä omia ilmauksiaan, jotka voivat käyttää Springin IoC-säilön hallittavaksi annettuja olioita. Modulaarisuutta ja uudelleenkäytettävyyttä voidaan tehdä luomalla omia annotaatioita, jotka toimivat mainittujen @PreAuthorize:n ja @PostAuthorize:n tavoin. Käytännössä tämä tekee käyttöoikeuksien rajaamismekanismista erittäin joustavan ja laajennettavan.

Ennen kuin metoditason pääsynhallinta voidaan ottaa käyttöön, täytyy se konfiguroida. Ominaisuuden käyttäminen Spring Security -kehyksessä edellyttää metoditason turvallisuuden käyttöönottoa joko Java- tai XML-pohjaisella konfiguraatiolla. Java-pohjaisella konfiguraatiolla tämä voidaan tehdä luomalla @EnableGlobalMethodSecurity-annotaatiolla varustettu luokka ja saada se ladattua Spring IoC-säiliöön.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MethodSecurityConfig {
}
```

Kuva 13: Metoditason turvallisuustarkistukset sovellukseen asettava luokka.

Luontevinta on asettaa annotaatio juuri Spring Securityn muitakin web-tietoturvaan liittyviä asetuksia kuvaavaan konfiguraatioluokkaan, jotta tietoturvaan liittyvät asetukset ovat samassa paikassa. @EnableWebSecurity-annotaatiolle voidaan antaa erilaisia parametreja (kuva 14).

Annotaation attribuutin prePostEnabled arvolla ”true” otetaan käyttöön metodin suorituksen ennen ja jälkeen käyttöoikeuksien tarkistamisesta vastaavat annotaatiot (taulukko 2).

8 Muiden web-sovelluksiin liittyvien tietoturvaominaisuuksien kytkeminen Spring Securityssä

Seuraavaksi tutustutaan erityisesti web-sovelluksiin liittyvien tietoturvaominaisuuksien parantamiseen Spring Securityllä.

8.1 CSRF-token

Kuten OWASP:n top 10 -listauksessa nähtiin, CSRF-hyökkäys on eräs vakava web-sovelluksia koskeva haavoittuvaisuus. Spring Security pakottaa web-sovelluksen käyttämään erillistä CSRF-tokenia, joka on web-sovelluksen ja käyttäjän jakama salaisuus, joka luodaan menetelmillä, jotka mahdollistavat sen ennalta-arvaamattomuuden. CSRF-token lähetetään aina palvelimelle tilaa muuttavien HTTP-pyyntöjen mukana, joita ovat esim. POST, PUT, DELETE ja PATCH. Mikäli käyttäjän selaimen lähettämä CSRF-token ei vastaa palvelimen antamaa CSRF-tokenia, niin pyyntö evätään.

Mikäli sovellus käyttää näkymien luomiseen Thymeleafia, voidaan käyttää hyväksi Thymeleafin ja Springin välistä integraatiota. Tämä tarkoittaa sitä, että Thymeleaf osaa lisätä CSRF-tokenin käsittelemilleen HTML-pohjille, jotka sisältävät HTML:n form-tagin. Toinen vaatimus on käyttää jotain Thymeleafin nimiavaruuteen kuuluvaa attribuuttia form-tagin sisällä. Kuvassa 15 on esitetty yksinkertainen HTML-pohja, joka sisältää yksinkertaisen lomakkeen.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    <head></head>
    <body>
        <h1>Anna nimesi:</h1>
        <form method="POST" th:action="@{/nimi}">
            <label for="etunimi">Etunimi:</label>
            <input type="text" autofocus="autofocus" required="true" name="etunimi"/>
            <label for="sukunimi">Sukunimi:</label>
            <input type="text" required="true" name="sukunimi"/>
            <input type="submit" value="lähetä"/>
        </form>
    </body>
</html>
```

Kuva 14: Yksinkertaisen lomakkeen sisältävä Thymeleaf-sivupohja

Kun kuvan 15 mukainen pohja käsitellään ja palautetaan käyttäjän selaimeen, sisältää se automaattisesti CSRF-tokenin arvon piilotettuna input-elementtinä kuvan 16 mukaisesti.

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
    <head></head>
    <body>
        <h1>Anna nimesi:</h1>
        <form method="POST" action="/nimi">
            <label for="etunimi">Etunimi:</label>
            <input type="text" autofocus="autofocus" required="true" name="etunimi" />
            <label for="sukunimi">Sukunimi:</label>
            <input type="text" required="true" name="sukunimi" />
            <input type="submit" value="lähetä" />
        <input type="hidden" name="_csrf" value="a3f51ea5-bd35-442d-9b85-9ae701aa75ab" /></form>
    </body>
</html>
```

Kuva 15: Palvelimen palauttama HTML-sivu, joka sisältää CSRF-tokenin.

8.2 Sessioiden luonnin rajoittaminen

Kuten luvussa 3.2.2 kuvailtiin, Session Fixation -hyökkäys mahdollistaa hyökkääjän pääsyn toisen käyttäjän käyttöoikeuksilla sovellukseen, mikäli web-sovellus ei käsittele

sessiota oikein. Näin voisi päästä käymään silloin, kun hyökkääjä voi jollain keinolla asettaa haluamansa sessioavaimen toiselle käyttäjälle.

Spring Security voidaan konfiguroida pyytämään aina uusi sessio käyttäjälle, kun tämä kirjautuu sisään sovellukseen. Koska palvelin luo sessioavaimen siten, ettei hyökkääjä pysty ennustamaan, mikä uuden sessioavaimen arvoksi tulee, ehkäisee tämä menettely Session Fixation -hyökkäyksen. Jotta tämä onnistuisi, täytyy Springin IoC-säiliöön luoda HTTP-session tapahtumista tietoja julkaiseva httpSessionEventPublisher-papu kuvan 17 mukaisesti.

```
@Bean
public ServletListenerRegistrationBean<HttpSessionEventPublisher> httpSessionEventPublisher() {
    return new ServletListenerRegistrationBean<>(new HttpSessionEventPublisher());
}
```

Kuva 16: httpSessionEventPublisher-pavun määritys

HttpSessionEventPublisher-pavun lisäksi session käsittely täytyy ylikirjoittaa WebSecurityConfigurerAdapter-luokan aliluokan configure()-metodi (kuva 18). HttpSecurity-konfigurointiobjektin avulla voidaan määrittää session luontiin liittyvä strategia ja yhden käyttäjän samanaikaisten sessioiden lukumäärä.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.ALWAYS).maximumSessions(1);
    http
            .authorizeRequests()
            .anyRequest().fullyAuthenticated()
            .and()
            .formLogin();
}
// ... loput luokasta
```

Kuva 17: Yhden aktiivisen session määrittäminen per web-sovelluksen käyttäjä

Kuvan 18 HttpSecurity-objektille kutsutaan sessionManagement()-metodia, jonka jälkeen metodikutsut ketjutetaan peräkkäin. SessionCreationPolicy-metodikutsu arvolla SessionCreationPolicy.ALWAYS pakottaa session luonnin käyttäjälle, kun taas maximumSessions-metodikutsulla asetetaan vain yksi sessio per käyttäjä.

8.3 HTTPS-yhteyskäytännön pakottaminen

Spring Securitylle voidaan ohjeistaa, että tiettyihin URL-osoitteisiin kohdistuvat pyynnöt tulee tehdä salattuna HTTPS-protokollan yli. Tämä voidaan tehdä CSRF-tokenin konfiguroinnin kaltaisesti WebSecurityConfigurerAdapter-luokan aliluokassa käyttämällä HttpSecurity-konfigurointiobjektia configure()-metodin toteutuksessa. Kuvassa 19 on annettu esimerkki, miten web-sovelluksen URL:iin ”/secure” vaaditaan HTTPS-protokollan käyttö.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

        @Override
        protected void configure(HttpSecurity http) throws Exception {

            http.antMatcher("/secure").requiresChannel().anyRequest().requiresSecure();

        }
        // ... loput luokasta
```

Kuva 18: Esimerkkikonfiguraatio HTTPS-yhteyskäytännön pakottamisesta

Tietoturvallisempi tapa olisi asettaa HTTPS-protokolla ainoaksi hyväksytyksi protokollaksi koko web-sovellukselle, sillä salatun yhteyden käyttö vähentää Man-In-The-Middle -hyökkäyksen riskiä ja arkaluontoisten tietojen päätymistä ulkopuolisten käsiin.

9 Yhteenveto

Loppuyhteenvetona voidaan todeta, että Spring Security osoittautui erittäin hyödylliseksi työkaluksi autentikoinnin ja pääsynhallinnan toteuttamisen tueksi ja ennen kaikkea tietoturvan sisällyttämiseksi osaksi sovellusta ilman, että varsinaiseen sovelluslogiikkaan tarvitsi lisätä autentikointiin tai pääsynhallintaan liittyvää koodia. Myös kehyksen laajennettavuus osoittautui merkittäväksi vahvuudeksi. Samalla toisaalta voitiin todeta, että siinä missä kehyksen onnistunut käyttö ei aina edellytä syvällistä tietämystä kehyksen sisäisestä rakenteesta ja teknisestä toteutuksesta, niin niistä on hyötyä kehyksen kaikkia ominaisuuksia hyödynnettäessä.

On kuitenkin huomattava, että jotain jäi myös puuttumaan, kuten valmiiden ratkaisujen tarjoaminen esim. käyttäjien syötteiden validoinnille, sillä Spring Security ei sisällä valmiita validoijia sovellusohjelmoijan käyttöön, vaan tähän tarkoitukseen täytyy erikseen käyttää jotain muuta työkalua. Myöskään SQL-injektoita vastaan ei Spring Security tar-

jonnut mitään keinoa suojautua, mutta toisaalta se sallii esim. HDIV-tietoturvakehyksen käyttämisen tätä tarkoitusta varten.

Insinöörityötä tehdessä osoittautui myös, että varsinkin metoditason tietoturvan toteuttamisessa täytyy olla tarkkana, sillä sen tekninen toteutus perustuu useaan vaikeasti ymmärrettävissä olevaan tekniikkaan, kuten aspektipohjaisuuteen. Mikäli näistä ei ole riittävää ymmärrystä, on mahdollista saada aikaan tietoturvariskejä. Monimutkaisuudesta johtuvia haittoja voidaan kompensoida riittävällä ohjelmistokehittäjien ohjeistuksella ja tiettyjen ohjelmointikäytäntöjen noudattamisella, sekä tietysti kattavalla testaamisella.

Kaiken kaikkiaan Spring Security soveltuu Spring-pohjaisten sovellusten tietoturvaratkaisuksi, joka tarjoaa luotettavaa ja modulaarista tapaa autentikoinnille ja auktorisoinnille. Koska se on riippuvainen Spring-kehyksestä, se ei sovellu juuri sellaisten projektien käyttöön, jotka on rakennettu ilman Spring-kehystä. Tällaisia käyttötapauksia varten on olemassa Apache Shiro, joka tarjoaa Spring Securityn tavoin valmiita ratkaisuja autentikointiin ja auktorisointiin. [16.]

Lähteet

1 Java Platform, Enterprise Edition. 2016. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition>. Päivitetty 17.09.2016. Luettu 30.10.2016.

2 Application Server. 2016. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Application_server>. Päivitetty 04.10.2016. Luettu 30.10.2016.

3 About The Open Web Application Security Project. 2013. Verkkodokumentti. OWASP. <https://www.owasp.org/index.php/About_OWASP>. Päivitetty 4.7.2013. Luettu 29.10.2016.

4 OWASP Top 10 2013. 2013. Verkkodokumentti. OWASP. <https://www.owasp.org/index.php/Top_10_2013>. Päivitetty 27.10.2016. Luettu 15.10.2016.

5 Walls, Craig. 2014. Spring In Action. 4th edition. New York: Manning.

6 Spring Framework Reference Documentation. 2016. Verkkodokumentti. <http://docs.spring.io/spring/docs/4.3.3.RELEASE/spring-framework-reference/htmlsingle>. 2016. Luettu 05.11.2016.

7 Ryan, Fintan. 2016. Language Framework Popularity: A Look At Java. Verkkodokumentti. Redmonk. <http://redmonk.com/fryan/2016/09/08/language-framework-popularity-a-look-java>. Luettu 05.11.2016.

8 Spring Security Framework Reference Documentation. 2016. Verkkodokumentti. Pivotal. <http://docs.spring.io/spring-security/site/docs/4.2.0.RELEASE/reference/htmlsingle/>. Luettu 10.11.2016.

9 Spring Boot Reference Guide. 2016. Verkkodokumentti. Pivotal. <http://docs.spring.io/spring-boot/docs/1.4.2.RELEASE/reference/htmlsingle/>. Luettu 25.11.2016.

10 DIP in the Wild. 2013. Verkkodokumentti. <http://martinfowler.com/articles/dipInTheWild.html>. Päivitetty 21.03.2013. Luettu 8.11.2016.

11 Dependency inversion principle. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Dependency_inversion_principle>. Päivitetty 23.11.2016. Luettu 26.11.2016.

12 Aspect-oriented programming. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Aspect-oriented_programming>. Päivitetty 01.11.2016. Luettu 20.11.2016.

13 Apache Maven IDE Integration. Verkkodokumentti. Apache Software Foundation. <https://maven.apache.org/ide.html>. Päivitetty 17.11.2016. Luettu 21.11.2016.

14 Introduction to the POM. Verkkodokumentti. Apache Software Foundation.

15 <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>. Päivitetty 17.11.2016. Luettu 21.11.2016.

16 Introduction to the Dependency Mechanism. Verkkodokumentti. Apache Software Foundation. <https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html>. Päivitetty 17.11.2016. Luettu 21.11.2016.

17 Java annotation. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Java_annotation>. Päivitetty 22.11.2016. Luettu 24.11.2016.

18 Apache Shiro Features Overview. Verkkodokumentti. Apache Software Foundation. <http://shiro.apache.org/features.html>. 2016. Luettu 22.11.2016.

19 Tietoturva. Verkkodokumentti. Wikipedia. <https://fi.wikipedia.org/wiki/Tietoturva>. Päivitetty 19.08.2016. Luettu 15.11.2016.

20 GUI Architectures. 2006. Verkkodokumentti. Fowler, Martin. <http://www.martinfowler.com/eaaDev/uiArchs.html>. Päivitetty 18.06.2006. Luettu 14.11.2016.

21 Tutorial: Using Thymeleaf. 2016. Verkkodokumentti. Thymeleaf. <http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>. Päivitetty 08.10.2016. Luettu 24.11.2016.

22 Inversion of Control Containers and the Dependency Injection Pattern. 2004. Verkkodokumentti. Fowler, Martin. <http://www.martinfowler.com/articles/injection.html>. Päivitetty 23.01.2004. Luettu 17.11.2016.