Mikko Homanen

Suunnittelumallien hyödyntäminen tietoturvallisessa ohjelmistokehityksessä

Tietotekniikan kandidaatintutkielma

26. huhtikuuta 2017

Jyväskylän yliopisto

Tietotekniikka

Tekijä: Mikko Homanen

Yhteystiedot: mikko.a.homanen@student.jyu.fi

Työn nimi: Suunnittelumallien hyödyntäminen tietoturvallisessa ohjelmistokehityksessä

Title in English: Secure software development using design patterns

Työ: Kandidaatintutkielma

Sivumäärä: 28+0

Tiivistelmä: Tietoturvasta on viime aikoina tullut entistäkin tärkeämpi osa-alue ohjelmistokehityksessä. Tietoturvan merkityksen kasvaessa tarvitaan menetelmiä, joilla tietoturva voidaan huomioida koko ohjelmistokehitysprosessin ajan. Suunnittelumallien käyttämistä on esitetty erääksi keinoksi ohjelmistojen tietoturvan parantamisessa. Tässä tutkielmassa tarkastellaan, kuinka suunnittelumalleja käyttämällä voidaan vaikuttaa ohjelmistojen tietoturvaan.

Avainsanat: ohjelmistoarkkitehtuuri, ohjelmistokehitys, ohjelmistosuunnittelu, sovelluskehitys, suunnittelumalli, tietoturva, turvallisuusmalli

Abstract: Information security has become an essential part of software development. As the importance of information security grows, it is essential to find methods for taking information security into account during software development processes. Design patterns have been proposed as a mean to improve software security. This thesis examines how design patterns can be used to improve software security.

Keywords: application development, design pattern, information security, security pattern, software architecture, software design, software development

Kuviot

Sisältö

1 Johdanto

Suunnittelumalleilla tarkoitetaan aiemmin hyväksi havaittuja ohjelmistokehityksen käytäntöjä, joiden avulla pyritään parantamaan ohjelmiston laatua. Suunnittelumalleista puhutaan useimmiten oliosuuntautuneessa ohjelmistokehityksessä, joskin ideaa on sovellettu myös muissa ohjelmointiparadigmoissa. Tarkoituksena voi olla esimerkiksi ohjelmiston modulaarisuuden tai suorituskyvyn parantaminen. Suunnittelumalleista alettiin puhua 90-luvun aikana, ja aiheen perusteoksena pidetään usein niin sanottua Gang of Four -kirjaa (Gamma, Helm, Johnson & Vlissides (1994)), johon kerättiin ensimmäistä kertaa laajempi kokoelma suunnittelumalleja. Suunnittelumallien tarkoituksena on mahdollistaa kokeneiden ohjelmistokehittäjien käyttämien hyvien ratkaisujen tarjoaminen kokemattomampien käyttöön, näin helpottaen kehitystyötä. Suunnittelumalleille tyypillistä onkin se, että niitä ei niinkään kehitetä vaan pyritään löytämään analysoimalla jonkin ongelman ratkaisemiseen käytettyjä, hyväksi havaittuja malleja.

Tietoturva on lähivuosina noussut entistäkin tärkeämmäksi osa-alueeksi, ja uusista kyberhyökkäyksistä uutisoidaan jatkuvasti. Esimerkiksi lokakuussa 2016 Dynpalveluntarjoajaa vastaan kohdistettu palvelunestohyökkäys rampautti suuren osan Internetiä (Guardian (26.10.2016)). Osa hyökkäyksistä on laajempia kuin toiset, mutta yhtä lailla jokainen tietoturva-aukko voi olla kriittisen tärkeä. Tietoturvan huomioimisen rooli korostuu suurissa järjestelmissä, joissa käsitellään salassapidettävää tietoa. Aloitteleva ohjelmistokehittäjä on harvoin asiantuntija ohjelmistoturvallisuudessa, joten kokeneiden kehittäjien aikojen saatossa löytämät perusratkaisut vaikuttavat tehokkaalta tavalta parantaa ohjelmistojen tietoturvaa.

Tietoturvan merkityksen kasvaessa jatkuvasti, suunnittelumallien tutkimuksessa olennaiseksi kysymykseksi onkin tullut, millaisia malleja voidaan hyödyntää ohjelmistojen tietoturvan parantamisessa. Ohjelmistoturvallisuudessa on havaittavissa samanlaisia perusrakenteita kuin muissakin ohjelmistojen laatuun liittyvissä ongelmissa. Jo aiemmin on huomattu, että suunnittelumalleja hyödyntämällä voidaan parantaa ohjelmistojen laatua, ja tietoturva voidaan myös ajatella ohjelmiston laadul-

lisena ominaisuutena. Näin ollen voidaan kenties olettaa, että suunnittelumalleista on apua myös tietoturvan parantamisessa.

Suunnittelumallit ovat erityisen tärkeässä osassa ohjelmistoturvallisuuden saralla siitä syystä, että läheskään jokainen ohjelmistokehittäjä ei ole tietoturva-asiantuntija. Kuitenkin tietoturva olisi syytä ottaa huomioon jo ohjelmiston varhaisessa suunnittelu- sekä kehitysvaiheessa. Suunnittelumalleja käyttämällä voidaan mahdollistaa se, että kokematonkin ohjelmistokehittäjä pystyy tekemään ohjelmistoja, joissa ei ole havaittavissa yleisimpiä tietoturvariskejä.

Tutkielman tarkoituksena on esittää perusideat suunnittelumalleista sekä tarkastella niiden hyötyjä, mahdollisuuksia ja haasteita tietoturvan parantamisessa. Luvussa 2 esitellään yleisesti suunnittelumalleja, niiden taustaa sekä luokittelua ja muotoilua. Luvussa 3 suunnittelumalleja tarkastellaan tietoturvan näkökulmasta ja pyritään selvittämään, miten suunnittelumalleja voidaan hyödyntää ohjelmistoturvallisuudessa. Luvussa 4 esitellään yksityiskohtaisesti kaksi tietoturvan parantamiseen tähtäävää suunnittelumallia. Luvussa 5 esitetään yhteenveto tutkielman tuloksista.

2 Suunnittelumallit

Kaikista 1990-luvulla kehitetyistä ohjelmistotekniikan tekniikoista suunnittelumallit ovat kenties tärkein ja eniten ohjelmistokehitykseen vaikuttanut käsite (Koskimies & Mikkonen 2005, s. 101). Suunnittelumallien lähtökohtana on alusta asti ollut se, että hyvien ratkaisujen kopiointi olisi syytä mahdollistaa myös tilanteissa, joissa jonkun muun tekemän työn yksityiskohtainen tarkastelu on mahdotonta. Sama ajatus on ollut tunnettu jo pitkään muilla tekniikan aloilla. Esimerkiksi rakennusarkkitehtuurissa ja koneenrakennuksessa on olemassa tiettyjä hyväksi havaittuja standardiratkaisuja, joita hyödyntämällä säästetään suunnittelun työmäärässä.

Suunnittelumalli sisältää yleisellä tasolla esitetyn ratkaisun johonkin oliopohjaisessa suunnittelussa esiintyvään tavanomaiseen tilanteeseen tai kohteeseen. Mallit koostuvat kolmesta olennaisesta osasta: ongelma, ratkaisu ja seuraukset (Gamma ym. 1994, s. 1-3). Suunnittelumalleja käsitellään tutkielmassa tämän määritelmän mukaan.

Suunnittelumallin käsitteeseen liittyy olennaisesti myös sen vastakohta, antisuunnittelumalli. Antisuunnittelumalli kuvaa huonoa ratkaisua, joka sisältää perustavanlaatuisia ongelmia. Antisuunnittelumallit voidaan kuitenkin ajatella lähes yhtä hyödyllisinä kuin suunnittelumallitkin. Usein antisuunnittelumallia hyödyntämällä voidaan löytää sitä vastaava, hyvän ratkaisun esittävä suunnittelumalli (Koenig 1995).

2.1 Sovelluskohteet ja käyttötarkoitukset

Suunnittelumalli on määritelty yleisesti hyväksi havaituksi ratkaisuksi tavanomaiseen ongelmaan. Suunnittelumalleille on ominaista myös se, että niitä ei niinkään kehitetä aktiivisesti vaan pitäisi puhua suunnittelumallin löytämisestä. Yleisyysvaatimuksesta seuraa myös se, että suunnittelumallille tulee olla näytettävissä toisistaan riippumattomia sovelluskohteita, joissa sitä on sovellettu menestyksekkäästi (Koskimies & Mikkonen 2005, s. 102-103).

Suunnittelumalli ei ole käsitteenä sidottu mihinkään tiettyyn ohjelmointiparadigmaan, sillä se kuvaa määritelmän mukaankin jonkin ratkaisun yleisellä tasolla. Suunnittelumallien tausta on kuitenkin oliosuuntautuneessa ohjelmistokehityksessä, joten usein mallit esitetään sen mukaisena. Eri paradigmoissa hyödynnetäänkin erilaisia suunnittelumalleja, ja käytetty paradigma myös määrittää sitä, mistä on ylipäänsä mielekästä puhua suunnittelumallina. Esimerkiksi jonkin ohjelmointiparadigman perusolettamuksia ei ole hyödyllistä esittää suunnittelumallina (Koskimies & Mikkonen 2005, s. 103-104).

Ohjelmistokehityksessä suunnittelumallit näkyvät eniten arkkitehtuurisuunnittelussa ja yksityiskohtaisessa suunnittelussa (Koskimies & Mikkonen 2005, s. 104). Suunnittelumallien tarkoituksena on useimmiten parantaa jotain ohjelmiston laadullista ominaisuutta, joten niiden hyödyntäminen arkkitehtuurisuunnittelussa vaikuttaa luonnolliselta. Kun arkkitehtuuri on suunniteltu suunnittelumallien mukaan , voidaan ohjelmistolle olennaisimmat laatuvaatimukset ottaa helposti huomioon jo suunnittelun alkuvaiheissa.

Suunnittelumallien käytöllä ei siis pyritä lisäämään järjestelmän ominaisuuksia, vaan parantamaan jotain ohjelmiston laatuominaisuutta. Esimerkiksi useiden Gang of Four -kirjan suunnittelumallien tarkoitus on parantaa ohjelmiston muunneltavuutta. Aikojen saatossa suunnittelumalleja on kehitetty parantamaan myös lukuisia muita laatuominaisuuksia kuten tietoturvaa, tehokkuutta ja ylläpidettävyyttä. Useimmissa tapauksissa suunnittelumallin käyttö parantaa jotain laatuominaisuutta mutta saattaa myös heikentää toista. Esimerkiksi paremmasta muunneltavuudesta voi seurata heikompi ylläpidettävyys ja uudelleenkäytettävyys (Koskimies & Mikkonen 2005, s. 117-118).

2.2 Luokittelu ja muotoilu

Kuten jo aiemmin todettiin, suunnittelumallin olennaisimmat osat ovat ongelma, ratkaisu ja seuraukset. Nämä osat toimivat suunnittelumallien kuvaamisen pohjana ja useimmiten suunnittelumallit noudattavat tätä perusrakennetta. Olennaisessa

osassa mallin kuvausta on sen nimi. Hyvin kuvaava nimi kertoo käyttäjälle heti, millaiseen ongelmaan malli on tarkoitettu. Ongelma kertoo tarkemmin siitä, mihin tarkoitukseen mallia voidaan soveltaa ja milloin sen käyttäminen on hyödyllistä. Ongelma voidaan usein pukea kysymyksen muotoon. Sen yhteydessä saatetaan myös kuvata esiehtoja, joiden tulee päteä, jotta mallin soveltaminen on mielekästä. Ratkaisussa kuvataan pääelementit, joista malli koostuu, sekä niiden suhteet, vastuualueet ja yhteistyö. Ratkaisun ei ole tarkoitus kuvata suoraa toteutusta ongelmalle, vaan se annetaan korkealla abstraktiotasolla. Korkean abstraktiotason kuvauksesta tulevat ilmi suunnitteluongelmaan liittyvät elementit ja niiden suhteet ongelman ratkaisemisessa. Suunnittelumallin soveltamisella on lähes aina myös seurauksia, joiden listaamisella pyritään kuvamaan mallin soveltamisen hyödyt ja haitat (Gamma ym. 1994, s. 1-4).

Suunnittelumallit jaotellaan monesti Gamman ym. (1994, s. 9-11) mallin mukaan kahden pääkriteerin mukaan. Ensimmäinen kriteeri on tarkoitus, joka kertoo, mitä suunnittelumalli tekee. Tarkoituksen perusteella mallit voidaan jakaa kolmeen luokkaan:

- Toiminnalliset mallit (behavioral patterns) kuvaavat luokkien ja olioiden välistä yhteistoimintaa ja vastuunjakamista.
- Luontimallit (creational patterns) käsittelevät olioiden luomiseen liittyviä ongelmia ja pyrkivät usein abstrahoimaan olioiden luomisprosessin sekä piilottamaan tiedon siitä, mitä konkreettisia luokkia järjestelmä käyttää.
- Rakenteelliset mallit (structural patterns) pyrkivät yhdistämään luokkia ja olioita suuremmiksi rakenteiksi.

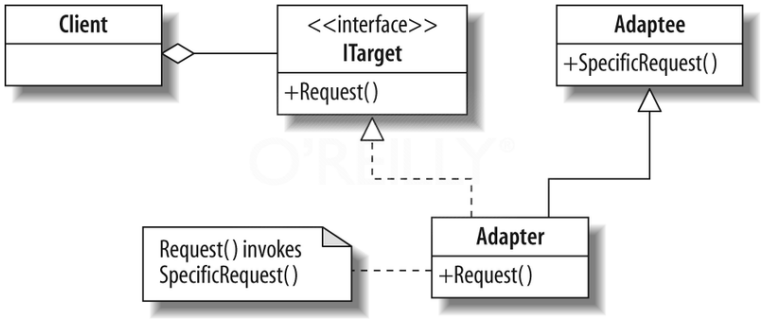
Toisena kriteerinä mallien luokittelussa käytetään ympäristöä. Ympäristön perusteella mallit jaetaan luokka- ja oliomalleihin. Luokkamallit kuvaavat luokkien ja aliluokkien välisiä suhteita. Ne siis muodostuvat perinnän kautta ohjelman käännösvaiheessa, joten suhteet ovat staattisia. Oliomallit puolestaan kuvaavat ajonaikaisia rakenteita ja ovat siten dynaamisempia. Suurin osa suunnittelumalleista kuuluu oliomalleihin.

2.3 Esimerkki: Sovitin(adapter)

Rakenteellisiin malleihin kuuluva Sovitin-suunnittelumalli on kuvattu Gamman ym. (1994, s. 139-143) mukaan seuraavasti:

Ongelma: Kaksi valmiiksi olemassa olevaa luokkaa eivät ole yhteensopivia rajapintojen vuoksi. Kuinka voidaan mahdollistaa rajapintojen vuoksi epäyhteensopivien luokkien yhteistoiminta?

Ratkaisu: Luodaan epäyhteensopivien luokkien väliin sovitin, joka muuntaa jonkin luokan rajapinnan toista vastaavaksi. Sovitin voi toimia yhteen suuntaan jolloin toinen luokka on palvelun tarjoaja ja toinen käyttäjä. Sovitin voi olla myös kaksisuuntainen, jolloin molemmat rajapinnat yhtenäistetään vastaamaan toisiaan. Yhdistäminen voidaan tehdä aiemmin määritetyn mukaan oliopainotteisesti tai luokkapainotteisesti. Luokkasovitin hyödyntää moniperintää ja oliosovitin olioiden koostamista. Luokkasovittimen rakenne on esitetty kuviossa 1.



Kuvio 1. Luokkasovitin UML-kielellä kuvattuna. Saatavissa: https://i-msdn.sec.s-msft.com/dynimg/IC400935.png

Soveltuvuus: Sovitinta suositellaan käytettävissä tilanteissa, joissa

- olemassaolevaa luokkaa halutaan käyttää, mutta sen rajapinta ei vastaa tarpeita
- halutaan suunnitella uudelleenkäytettävä luokka, joka voi kommunikoida vielä tuntemattomien luokkien kanssa
- halutaan käyttää useaa olemassaolevaa aliluokkaa, mutta ei ole käytännöllistä sovittaa rajapintoja aliluokittamalla jokainen (vain oliosovitin).

Seuraukset:

Luokkasovitin:

- sovittaa palvelun tarjoajan ja pyytäjän sitoutuen konkreettiseen sovittuja-luokkaan, minkä seurauksena luokkasovitin ei toimi kun halutaan sovittaa luokka ja kaikki sen aliluokat
- mahdollistaa sovittimen ylikirjoittaa joitakin sovittautujan toimintoja, sillä sovitin on sovittautujan aliluokka

Oliosovitin:

- mahdollistaa yksittäisen sovittimen toimia useamman sovittautujan kanssa. Sovitin voi myös lisätä toiminnallisuutta kaikille sovittautujille yhtä aikaa.
- vaikeuttaa sovittautujan toiminnallisuuden ylikirjoittamista.

Muuta huomioitavaa:

- Kuinka paljon sovittamista Sovitin vaatii? Yksinkertaisimmillaan sovitin voi vaihtaa operaatioiden nimiä ja toisaalta se saattaa tukea kokonaan erilaista operaatioiden kirjoa. Sovittimen työmäärä riippuu rajapintojen yhtäläisyyksistä.
- Kaksisuuntaisten sovittimien käyttö läpinäkyvyyden tarjoamiseksi. Sovittimien ongelmana voi olla, että ne eivät näy kaikille käyttäjille. Tämä näkyvyys on ratkaistavissa kaksisuuntaisen sovittimien käytöllä.

3 Suunnittelumallit ja tietoturva

Tietoturva käsitteenä on kaikkea muuta kuin yksiselitteinen, ja erilaisia määritelmiä tietoturvalle on lukuisia. Tässä tutkielmassa tietoturva määritellään ISO/IEC 27000:2009 -standardin määritelmän mukaan kolmen pääkriteerin kautta:

- Saatavuus (availability): tiedon tulee olla saatavilla kun sitä tarvitaan.
- Luottamuksellisuus (condifentiality): tietoon on pääsy vain sellaisillä henkilöillä, joilla on siihen oikeus.
- Eheys (integrity): tieto ei saa muuttua huomaamatta tai luvattomasti.

Näiden pääkriteerien lisäksi standardin mukaan tietoturvaan voidaan liittää myös kiistämättömyys (tehtyä tekoa ei voi kiistää), tunnistus (käyttäjä voidaan yksilöidä) ja todennus (käyttäjä voidaan luotettavasti tunnistaa).

Tietoturvan parantamiseen tähtäävät suunnittelumallit ovat olleet viime vuosina aktiivisen tutkimuksen kohteena. Osittain tämä on seurausta kehittyneiden tekniikoiden ja uusien laitteiden, kuten tablettien ja älypuhelimien yleistymisestä. Kun älylaitteet ovat yleistyneet, ovat tietoturvaseikat tulleet entistä yleisemmiksi myös ohjelmistokehityksessä. Staattisten koodianalyysityökalujen lisäksi yleisimpiä ongelmia pyritään välttämään arkkitehtuuritason suunnittelussa suunnittelumallien avulla (Bunke, Sohr 2011). Vaikka ne eivät perusideansa puolesta eroa muista suunnittelumalleista merkittävästi, on monissa tapauksissa eriytetty turvallisuusmallin (security pattern) käsite kun puhutaan tietoturvaa parantavista suunnittelumalleista.

Schumacher (2003, s. 52-54) esittää tietoturvaongelmien perimmäiseksi syyksi useimmissa tapauksissa ihmiset ja inhimilliset virheet. Vaikka sama ongelma voi päteä muihinkin ohjelmistojen laatuongelmiin, ongelmat näkyvät Schumacherin mukaan erityisen vahvasti tietoturvassa, sillä läheskään jokainen ohjelmistokehittäjä ei ole tietoturva-asiantuntija. Koska suunnittelumallien pohjimmainen tarkoitus on siirtää asiantuntijoiden tietämystä kokemattomampien käyttöön, ne soveltuvat hyvin myös yleisimpien tietoturvariskien minimoimiseen.

Toisaalta taas ohjelmistot ja tietojärjestelmät ovat kehittyneet erittäin monimutkaisiksi, ja monissa tapauksissa ne käsittelevät arvokasta tietoa. Monimutkainen järjestelmä johtaa usein myös haavoittuvuuksiin, sillä laajan järjestelmän suunnittelu, kehittäminen sekä ylläpito ovat vaikeita tehtäviä. Lisäksi laajat ja monimutkaiset järjestelmät kehitetään usein pienissä palasissa. Tällöin voi olla haastavaa varmistaa järjestelmän tietoturva kokonaisuuden kannalta, vaikka yksittäiset järjestelmän osat olisivat huolellisesti suunniteltuja (Fernandez 2013, s. 34-35).

Suunnittelumallit tarjoavat mahdollisuuden ottaa yleisimmät tietoturvariskit huomioon jo korkean tason arkkitehtuurisuunnittelussa. Olennaista on tunnistaa järjestelmään kohdistuvat tietoturvariskit jo alkuvaiheessa ja suunnitella arkkitehtuuri tunnistettujen riskien pohjalta. Kun arkkitehtuuri on kuvattu suunnittelumallien avulla, on kehittäjien helppo toteuttaa riskien torjuminen (Fernandez 2013, s. 36-37).

3.1 Turvallisuusmallien luokittelu ja valinta

Turvallisuusmalleja, kuten suunnittelumalleja yleisestikin, kerätään usein katalogimuotoon jossa ne luokitellaan jonkin perusominaisuuksien mukaan. Luokittelulla pyritään tekemään sopivan mallin löytämisestä valitun ongelman ratkaisemiseen mahdollisimman helppoa. Erilaisten turvallisuusmallien määrän nopea kasvu on kuitenkin aiheuttanut sen, että sopivan mallin löytäminen voi olla erittäin vaikeaa, joten niille on pyritty kehittämään tarkempia luokittelumenetelmiä (Alvi & Zulkernine 2012). Mallien suuren määrän lisäksi sopivan mallin valintaa monimutkaistaa vielä se, että turvallisuusmalleille ei ole määritetty mitään standardimuotoa tai -syntaksia. Turvallisuusmalleja on perinteisen Gang of Four -mallin lisäksi luokiteltu esimerkiksi TCP/IP-mallin mukaisten viitekerroksien mukaan (Guan, Yang, & Wang (2016)).

Luokittelu ei siis ole missään nimessä yksiselitteistä ja turvallisuusmalleja voidaan luokitella useilla eri tavoilla. Yksittäisen luokittelumallin ylivertaisuutta muihin ei ole vielä esitetty, joten ei ole yksiselitteistä, millä tavalla turvallisuusmalleja tulisi luokitella jotta sopivan mallin valitseminen olisi helpointa. Seuraavaksi käsitellään

kolmea erilaista Hafizin, Adamczykin & Johnsonin (2007) esittelemää luokittelutapaa.

Yksinkertaisin keino luokitella turvallisuusmalleja on tehdä jako ISO-standardin mukaisen CIA-mallin mukaan kolmeen luokkaan: saatavuus, eheys ja luottamuksellisuus. Ongelmana tässä luokittelussa on se, että monet turvallisuusmallit sopivat useampaan luokkaan, joten luokittelu on liian yleinen. Sen hyötynä on kuitenkin se, että CIA-malli on yleisesti tunnettu joten luokittelu on helppo ymmärtää.

Toisena vaihtoehtona Hafiz ym. (2007) esittävät luokittelun ongelmayhteyden perusteella sen mukaan, mitä ohjelmiston osa-aluetta pyritään turvaamaan:

- Ydinturvallisuus (core security): järjestelmän sisällä olevien toimintojen turvallisuus
- Reunaturvallisuus (perimeter security): tunnistaminen, valtuuttaminen yms. järjestelmän sisääntulokohdissa
- Ulkopuolinen turvallisuus (exterior security): tiedonsiirto- ja kommunikaatioprotokollat

Tämän luokittelun etuna on se, että mallien kirjoittaminen aloitetaan useimmiten ongelman ja ongelmayhteyden kuvaamisella, joten käyttäjät tuntevat ongelmayhteyden johon ovat soveltamassa mallia. Toisaalta tässäkin luokittelussa osa malleista sijoittuu useampaan kategoriaan ja suurin osa kuuluu ydinturvallisuus-luokkaan.

Kolmas vaihtoehto on tehdä luokittelu ratkaistavan ongelman suhteen. Tällöin voidaan puhua uhkamallinnuksesta (threat modeling), jolloin turvallisuusmallit luokitellaan jotakin yleistä tietoturvauhkaa vasten. Eräs tällainen malli on STRIDE-malli:

- Spoofing: hyökkääjä pyrkii järjestelmään väärennetyllä henkilöllisyydellä
- Tampering: datan korruptoiminen tietoverkkoyhteyksien aikana
- Information Disclosure: yksityisen datan tekeminen julkiseksi tahtomatta
- Denial of Service: järjestelmä pyritään tekemään saamattomaksi
- Elevation of Privilege: pyrkimys nostaa oikeustasoa hyväksikäyttämällä haavoittuvuutta

Tämä luokittelu vaikuttaa selkeimmältä sen suhteen, mitä mallia tulisi hyödyntää jos tiedetään tarkasti, mitä ongelmaa ollaan torjumassa. Esimerkiksi hajautetut palvelinestohyökkäykset ovat nykyaikana yleisiä, joten Denial of Service -luokan malleille on paljon käyttöä. Tämäkin luokittelu kuitenkin osoittaa osan turvallisuusmalleista kaikkiin luokkiin.

Kuten edellä käsitellyt luokittelumallit osoittavat, on turvallisuusmallien luokittelussa kenties suurin ongelma se, miten turvallisuusmallit saadaan yksiselitteisesti kohdistumaan vain yhteen luokkaan. Erilaisilla luokitteluperusteilla on omat vahvuutensa, mutta riippumatta valitusta luokittelusta osa turvallisuusmalleista kuuluu useampaan kategoriaan. Näin ollen luokittelussa tulisikin hyödyntää useaa eri näkökulmaa ja esimerkiksi yhdistämällä STRIDE-malli ja ongelmayhteys päädytään jo hieman parempaan ratkaisuun (Hafiz ym. 2007).

3.2 Mallien soveltaminen ohjelmistokehityksen eri vaiheissa

Jotta turvallisuusmalleista saadaan mahdollisimman paljon hyötyä irti, on olennaista, että malleja hyödynnetään koko ohjelmiston elinkaaren ajan (Fernandez & Larrondo-Petrie 2010). Ennen kaikkea kehityksen alkuvaiheessa uhkien ja haavoittuvuuksien tunnistaminen on monissa tapauksissa puutteellista, ja turvallisuusmallien hyödyntäminen suunnittelussa voi parantaa tilannetta (Kobashi, Yoshioka, Okubo, Kaiya, Washizaki & Fukazawa (2013)). Luvussa 3.3 esitetään tarkemmin eräs keino integroida turvallisuusmallit ohjelmistokehitysprosessiin.

Ferraz, Assad & Meira (2009) toteavat, että syynä tietoturvaseikkojen vähäiseen huomioimiseen kehityksen alkuvaiheessa on usein se, että järjestelmät suunnitellaan toiminnallisten vaatimusten perusteella. Tällöin laadulliset vaatimukset kuten tietoturva jäävät toteutettaviksi myöhemmin. Monissa tapauksissa laadullisten vaatimusten täyttäminen kehityksen loppuvaiheessa osoittautuu hyvin kalliiksi ja monimutkaiseksi prosessiksi. Ferrazin ym. (2009) tulokset kuitenkin osoittivat, että suunnittelumalleja käyttämällä tätä ongelmaa voidaan pyrkiä minimoimaan.

Fernandezin (2013, s. 714) esittämän jaon mukaan turvallisuusmallit näkyvät ohjel-

mistokehityksen eri vaiheissa seuraavasti:

- Vaatimusmäärittely: käyttötapaukset määrittävät vaaditut toiminnot järjestelmälle.
- Analyysi: turvallisuusmallien avulla otetaan käyttöön turvallisuusmekanismeja.
- Suunnittelu: Analyysivaiheessa tunnistettujen turvallisuusmallien avulla suunnitellaan järjestelmän komponentit ja vuorovaikutus.
- Toteutus: turvallisuusmalleja ja aiemmissa vaiheissa ilmenneitä tietoturvasääntöjä hyödyntämällä voidaan toteuttaa suunnitteluvaiheen komponentit.

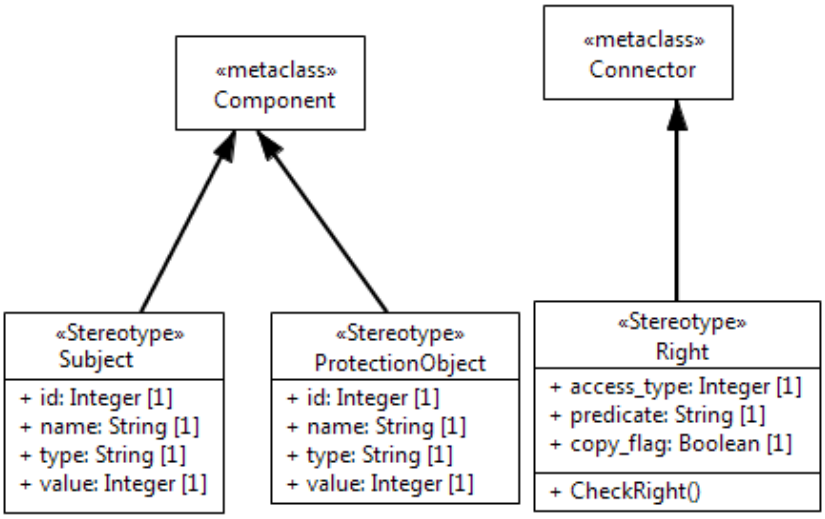
Käyttötapauksia suunnitellessa on tietoturvan kannalta olennaista tunnistaa käyttötapauksiin liittyvät tietoturvariskit ja mahdolliset heikkoudet. Käyttötapausten aktoreita voidaan tarkastella hyökkääjänä, jolloin voidaan löytää väärinkäyttöriskejä. Näin saadaan muodostettua lista järjestelmään kohdistuvista tietoturvariskeistä, jotka pyritään minimoimaan suunnitteluvaiheessa turvallisuusmalleja hyödyntämällä (Fernandez 2013, s. 715). Esimerkiksi web-palvelun kohdalla olennainen käyttötapaus voisi olla kirjautuminen palveluun, jolloin kirjautuja voi olla hyökkääjä, joka yrittää palvelunestohyökkäystä. Tällöin voidaan hyödyntää palvelunestohyökkäyksiä vastaan puolustavaa turvallisuusmallia.

Vaikka turvallisuusmallien on esitetty olevan eräs keino minimoida tietoturvaongelmia koko ohjelmiston elinkaaren ajan, ne eivät välttämättä sellaisenaan ole riittävän tehokkaita. Syynä tähän on se, että mallit kuvaavat ratkaisut useimmiten hyvin abstraktilla tasolla. Jotta mallien soveltaminen olisi tehokasta, tarvitaan keinoja abstraktin ratkaisun toteuttamiseen (Bouaziz & Coulette 2012). Esimerkiksi luvussa 4.2 tarkemmin esitetyn Valtuutus-mallin toteuttamiseksi komponenttipohjaisessa järjestelmässä Bouaziz & Coulette (2012) esittävät laajennetun UML-profiilin luomisen kolmessa vaiheessa:

1. Kartoitus: valitaan sovellettava turvallisuusmalli halutun ongelman ratkaisemiseen sekä kartoitetaan malliin liittyvät komponentit ja niiden vastuut.
2. Profiilin työstäminen: Kartoitusvaiheessa löydetyille komponenteille määri-

tetään stereotyypit metaluokkien kautta. Lisäksi määritetään stereotyyppien ominaisuudet.

3. Tulos: esimerkki vaiheiden 1 ja 2 tuloksena syntyneestä UML-profiilista on esitetty kuviossa 2.



Kuvio 2. Valtuutus-mallin laajennettu UML-profiili. Bouaziz & Coulette (2012)

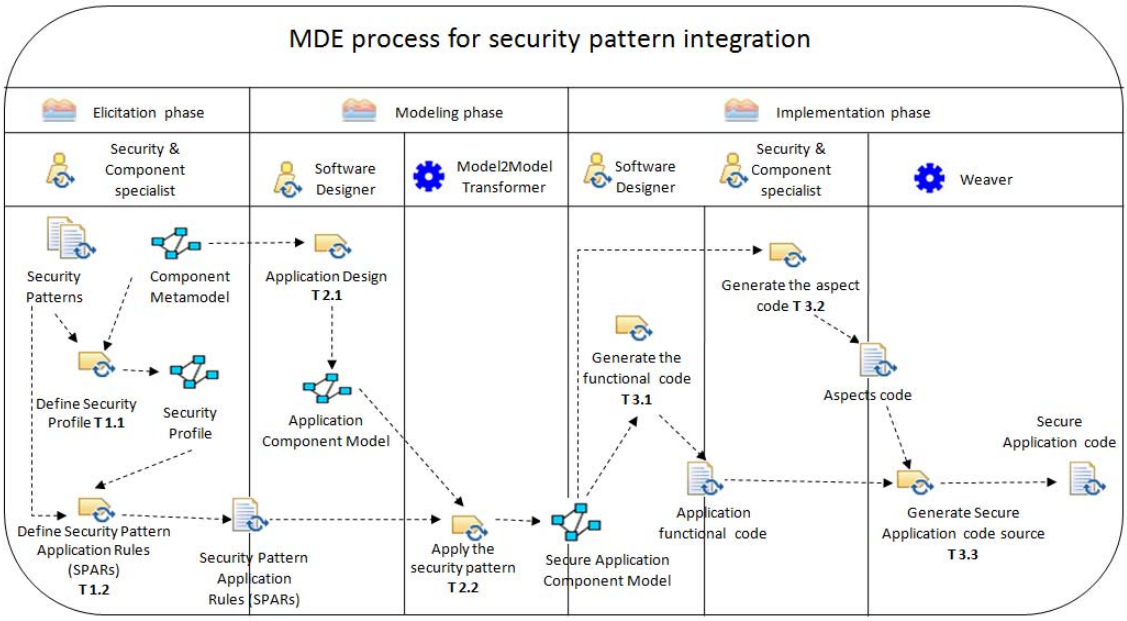
Oletetaan, että järjestelmän suunnittelussa on havaittu jokin tietoturvariski ja löydetty sen ratkaiseva turvallisuusmalli. Toteutusvaiheessa riittää että on käytettävissä mallin määrittely sekä jokin menetelmä mallin soveltamiseen. Tällöin siis kehittäjän ei itse tarvitse tuntea hyvää ratkaisua ongelmaan, kunhan toteutus tehdään mallin määrittelyn mukaisesti. Esimerkiksi edellä mainitussa web-palveluun kirjautumisessa voitaisiin hyödyntää luvuissa 4.1 ja 4.2 esitettyjä turvallisuusmalleja.

3.3 Integrointi sovelluskehitysprosesseihin

Kuten luvussa 3.1 todettiin, sopivan turvallisuusmallin valintaan liittyy ongelmia, sillä niiden luokittelu ei ole yksiselitteistä. Vaikka luokittelu olisi selkeää ja sopiva malli löytyisi, turvallisuusmallin soveltamiseen voi silti liittyä ongelmakohtia. Eräs tällainen ongelma on se, että selkeää ja yleisesti hyväksyttyä tapaa integroida turvallisuusmalli alustavasta suunnittelusta lopulliseen koodin toteutukseen ei ole esitetty.

Bouaziz ym. (2013) esittävät integrointiin eräänä vaihtoehtona SCRIP-prosessin (SeCurity patterRn Integration Process), jonka tarkoitus on integroida turvallisuusmallit komponenttipohjaisiin järjestelmiin. SCRIP-prosessi muodostuu kolmesta peräkkäisestä vaiheesta: aikaansaaminen (elicitation), mallinnus (modeling) sekä toteutus (implementation). Jokainen näistä vaiheesta koostuu useasta iteraatiosta.

Aikaansaamisvaiheen tärkeimpänä roolina on turvallisuusasiantuntija (security specialist). Syötteinä vaiheessa toimivat komponentti-metamalli, joka määrittää komponenttipohjaisen järjestelmän perusolettamukset sekä turvallisuusmallit, jolla tarkoitetaan jollain tavalla luokiteltua kokoelmaa turvallisuusmalleja. Syötteiden perusteella asiantuntija tuottaa seuraavalle vaiheelle uudet syötteet, jotka ovat turvallisuusprofiili (security profile) sekä turvallisuusmallin soveltamissäännöt (security pattern application rules).



Kuvio 3. SCRIP-prosessi. Bouaziz ym. (2013)

Mallinnusvaiheessa on kaksi roolia, jotka käyttävät edeltävän vaiheen syötteitä: ohjelmistosuunnittelija, joka on henkilö, sekä mallin muuntaja (model2model transformer), joka on ohjelmistotyökalu. Ohjelmistosuunnittelija tuottaa kompomenttimetamallin pohjalta ohjelman toiminnallisen suunnitelman esimerkiksi UML-mallien avulla komponenttimalliksi. Tämän suunnitelman ja edellisessä vaiheessa synty-

neiden turvallisuusmallin soveltamissääntöjen perusteella mallin muuntaja tuottaa turvallisen komponenttimallin.

Viimeisessä vaiheessa on kolme roolia: aiemmissa vaiheissa esiintyneet ohjelmistosuunnittelija ja turvallisuusasiantuntija, sekä uutena roolina kutoja (weaver). Turvallisen komponenttimallin pohjalta ohjelmistosuunnittelija tuottaa varsinaisen ohjelmakoodin. Turvallisuusasiantuntija ja ohjelmistosuunnittelija tuottavat turvallisen komponenttimallin pohjalta yhteistyössä myös aspektikoodin (aspect code), jonka tarkoituksena on varmistaa koodin turvallisuus esimerkiksi metodikutsujen yhteydessä tarkistamalla, että kutsujalla on oikeus suorittaa metodi. Lopuksi kutojan toimesta aspektikoodi ja ohjelmakoodi yhdistetään lopulliseksi, turvalliseksi ohjelmakoodiksi.

3.4 Vaikutusten ja laadun arviointi

Yleinen oletus on, että turvallisuusmallien soveltaminen parantaa ohjelmistojen tietoturvaa tilanteissa, joissa kehittäjällä ei ole laajaa tietoturvaosaamista. Kuitenkin mallien laaja valikoima voi johtaa tilanteeseen, jossa käyttöön valitaan ongelman kannalta väärä malli tai sen toteutus tehdään huonosti. Tästä syystä tarvitaan keinoja arvioida turvallisuusmallin vaikutus haluttuun ongelmaan (Kobashi ym. 2013).

Vaikka turvallisuusmallien määrittelyä ja luokittelua on tutkittu paljon, niiden soveltamisesta on vielä verrattain vähän empiiriseen tutkimukseen perustuvaa tietoa. Olennaisena ongelmana on etenkin se, onko mallien määrittely riittävän tarkkaa, jotta niiden vaikutuksia voidaan pitää luotettavina (Duncan & de Muijnck-Hughes 2014). Yskoutin, Scandariaton & Joosen (2015) tekemässä tutkimuksessa jopa havaittiin, että oikeinkaan sovelletut turvallisuusmallit eivät välttämättä olennaisesti paranna tietoturvaa. Opiskelijoilla teetetyssä tutkimuksessa kuitenkin todettiin, että mallien avulla työskentely paransi opiskelijoiden luottamusta osaamiseensa. Tuloksista huolimatta mallien avulla työskentely koettiinkin mielekkäämmäksi kuin ilman. Tämä tukee ajatusta siitä, että mallien soveltamisesta on hyötyä erityisesti kokemattomalle kehittäjälle.

Kobashi ym. (2013) esittävät UML-kieleen perustuvaa mallitestausta erääksi keinoksi arvioida turvallisuusmallin tehokkuutta tietyn ongelman ratkaisemiseen. Esitetyn metodin mukaan suunnitteluvaiheessa tulisi tehdä kaksi tarkistusta: onko malli toteutettu tarkoituksenmukaisella tavalla ja ratkaiseeko mallin soveltaminen ylipäätään määrittelyssä löytyneen ongelman. Tuloksista ilmeni, että tällainen lähestymistapa mallin soveltamisen arvioinnissa voi ratkaista edellä mainittuja turvallisuusmallien arviointiin liittyviä ongelmia.

4 Esimerkkejä turvallisuusmalleista

Luvussa 3 esiteltiin turvallisuusmallien luokittelua, soveltamista ja vaikutuksia. Seuraavaksi esitellään tarkemmin kahden eri turvallisuusmallin määrittely. Lähes jokaisessa suuressa tietojärjestelmässä käytetään jonkinlaista käyttäjän tunnistamista ja eritasoisia käyttöoikeuksia. Tästä syystä tarkasteltaviksi on valittu turvallisuusmallit Todentaja ja Valtuutus. Todentaja (4.1) keskittyy käyttäjän tunnistamiseen ja Valtuutus (4.2) käyttäjän oikeuksien määrittämiseen.

4.1 Todentaja (authenticator)

Todentaja-malli on määritetty Fernandezin (2013, s. 114-120) mukaan seuraavasti.

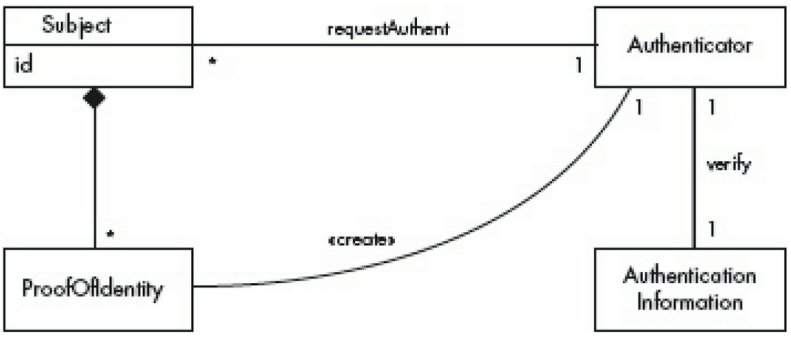
Tarkoitus: kun käyttäjä tunnistautuu järjestelmään, Todentaja-mallin avulla voidaan varmistaa, että käyttäjä on juuri se, joka väittää olevansa.

Ongelma ja ongelmayhteys: Kuinka voidaan estää väärillä henkilöllisyyksillä esiintyvien käyttäjien pääsy järjestelmään? Ongelma on erityisen tärkeä järjestelmissä, jotka sisältävät arvokasta tietoa esimerkiksi liiketoiminnasta. Tietoon tulee olla tällaisissa tapauksissa pääsy vain niillä, joilla on siihen selkeä syy.

Ratkaisu: Käytetään yksittäistä järjestelmän sisääntulokohtaa käyttäjän toimintojen vastaanottamiseen ja sovelletaan tässä protokollaa henkilöllisyyden varmistamiseksi.

Rakenne: Todentajan rakenne on esitetty kuviossa 4. Käyttäjä (subject) pyytää pääsyä järjestelmään. Todentaja (authenticator) vastaanottaa pyynnön ja soveltaa protokollaa käyttäen jotain todentamistietoa (Authentication Information). Mikäli todennus onnistuu, Todentaja luo käyttäjälle henkilöllisyydenvarmennuksen (Proof of Identity), jonka avulla käyttäjä voi varmentaa henkilöllisyytensä.

Toteutus: Keskitetyissä järjestelmissä on käyttöjärjestelmän vastuulla luoda sessio vastauksena käyttäjän pyyntöön. Kun pyyntö on käsitelty, käyttäjällä on pääsy o-



Kuvio 4. Todentaja (authenticator) -mallin rakenne. (Fernandez 2013, s. 117)

.

mien oikeuksiensa mukaisiin resursseihin. Tietokantajärjestelmällä voi olla erikseen oma tunnistautumisjärjestelmänsä, vaikka se toimisi käyttöjärjestelmän päällä. Järjestelmästä ja sen käsittelemän tiedon arvosta riippuu, kuinka tarkasti ja monitasoisesti tunnistaminen tulee tehdä. Erilaisia tunnistukseen käytettäviä protokollia on useita, ja niiden luomiselle on olemassa omia mallejaan.

Seuraukset: todentaja-mallin hyödyntämisellä on useita seurauksia. Mallin hyötyjä ovat:

- Riippuen käytetystä todennusprotokollasta, voidaan todentaa erilaisia käyttäjiä ja valtuuttaa käyttäjät erilaisilla keinoilla.
- Koska tunnistustieto on eriytetty, se voidaan tallentaa turvatulle alueelle johon kaikilla käyttäjillä on korkeintaan lukuoikeus.
- Tarvittavan todennuksen vahvuuden mukaan toteutus voidaan tehdä eri hintaluokan ratkaisuilla. Mahdollisuuksia ovat alemmasta suojaustasosta alkaen salasana, ID-kortti ja esimerkiksi erilaiset biometriikat kuten sormenjälkitunnistus.
- Todennus voidaan tehdä sekä keskitetyssä että hajautetussa ympäristössä
- Tuotettua henkilöllisyydenvarmennusta voidaan uudelleenkäyttää, mikä parantaa suorituskykyä.

Mallin mahdollisia haittoja ovat:

- Todennus vie aikaa, vaikutus riippuu käytetystä todennusprotokollasta.
- Mitä korkeamman tason todennus toteutetaan, sitä monimutkaisempi järjestelmästä tulee.
- Mikäli protokolla on monimutkainen, käyttäjät voivat turhautua.

Vaihtoehdot: Yksittäinen kirjautuminen (single sign on), jossa käyttäjä tunnistautuu kerran ja tätä tietoa voidaan käyttää useilla toimialueilla tietyn ajan verran.

Tunnetut käyttökohteet: Todentaja-mallin mukainen todentaminen näkyy jossain muodossa lähes jokaisessa kaupallisessa käyttöjärjestelmässä. Todentaminen on käytössä myös esimerkiksi internet-kaupoissa kuten eBay ja Amazon.

4.2 Valtuutus (authorization)

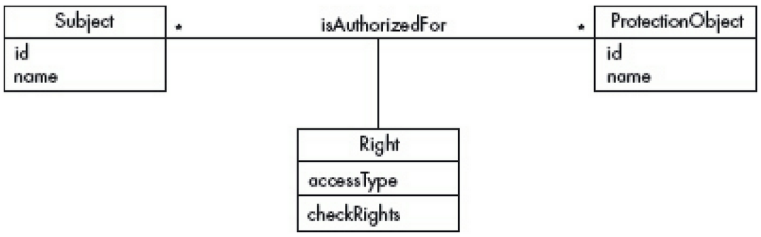
Valtuutus (authorization) -malli on määritetty Fernandezin (2013, s. 149-154) mukaan seuraavasti:

Tarkoitus: Valtuutus-malli kuvaa ratkaisun sille, kuinka voidaan määrittää, mihin resursseihin käyttäjällä on oikeus ja mitä käyttäjä voi resurssilla tehdä.

Ongelma ja ongelmayhteys: Kuinka voidaan määrittää, kenellä on pääsy tiettyyn resurssiin järjestelmässä? Kun käsitellään tietoa, jonka ei tule olla jokaisen käyttäjän saatavilla, tarvitaan tapa hallita oikeuksia ja käsitellä resursseja. Hallintajärjestelmän tulee toimia samalla tavalla riippumatta resurssin tyypistä. Oikeuksien muuttamisen tulee lisäksi olla helppoa.

Ratkaisu: Jokaiselle käyttäjälle, jolla on pääsy johonkin resurssiin, määritetään tieto (protection object) siitä, mihin resurssiin on pääsy ja mitä käyttäjä voi resurssille tehdä.

Rakenne: Valtuutuksen rakenne on esitetty kuviossa 5. Käyttäjä (subject) yrittää saada pääsyn resurssiin (ProtectionObject) jollain tavalla. Käyttäjän ja resurssin välinen assosiaatio (isAuthorizedFor) määrittää valtuutuksen, josta malli on saanut nimensäkin. Assosiaatioluokka Oikeus (Right) kuvaa pääsyn tyypin (luku/kirjoitus), joka



Kuvio 5. Valtuutus-mallin rakenne. (Fernandez 2013, s. 151)

käyttäjällä on resurssiin.

Toteutus: Organisaatio määrittää omistamalleen järjestelmälle omien käytäntöjensä mukaan käyttäjien oikeudet niihin resursseihin, joihin käyttäjille tarvitaan pääsy. Malli on abstrakti ja sille on useita eri toteutusmahdollisuuksia. Eräs yleinen toteutusmalli on Pääsyoikeuslista (Access Control List), jossa jokaisella turvatulla resurssilla on olemassa lista käyttäjistä joilla on pääsy resurssiin.

Seuraukset: Valtuutus-mallilla on seuraavat hyödyt:

- Malli toimii kaikentyyppisten resurssien kanssa.
- Malli tukee useita erityyppisiä käyttäjiä kuten suorittavia prosesseja, käyttäjiä, rooleja tai käyttäjäryhmiä. Turvatut resurssit voivat olla esimerkiksi I/O -laitteita tai tiedostoja.
- Pääsytyypit voidaan määrittää yksilöivästi ja ne voivat olla sovelluskohtaisia. Oikeuksien lisääminen ja muuttaminen on helppoa.
- Pääsypyynnössä ei välttämättä tarvitse täsmentää tiettyä kohderesurssia, vaan resurssi voidaan päätellä joissain tapauksissa olemassa olevasta turvatusta resurssista. Tämä vähentää tarvittavien sääntöjen määrää.

Mallilla on myös seuraavat mahdolliset haitat:

- Mikäli käyttäjiä on paljon, joudutaan myös pääsysääntöjä kirjoittamaan paljon. Tämä voi vaikeuttaa ylläpitoa.
- Ylläpitäjien voi joissain tapauksissa olla hankalaa varmistaa, miksi käyttäjä tarvitsee pääsyn johonkin resurssiin.
- Valtuutussääntöjen määrittely yksin ei riitä, vaan tarvitaan myös toimeenpa-

nomekanismeja.

Tunnetut käyttökohteet: Valtuutus-malli kuvaa yksinkertaisimman tyyppisen valtuutussäännön, jonka päälle voidaan rakentaa monimutkaisempia oikeuksien hallinnan malleja. Valtuutus toimii siis pohjana oikeuksien hallinnalle useimmissa kaupallisissa järjestelmissä ja tietokantaratkaisuissa kuten UNIX, Windows ja Oracle.

5 Yhteenveto

Tutkielmassa pyrittiin selvittämään, kuinka suunnittelumalleja voidaan hyödyntää ohjelmistojen tietoturvan parantamisessa. Luvussa 2 tarkasteltiin suunnittelumallien taustaa sekä määrittelyä ja luokittelua yleisellä tasolla. Suunnittelumalleihin perehdyttiin tarkemmin tietoturvan näkökulmasta luvussa 3. Samalla erotettiin turvallisuusmallin käsite yleisestä suunnittelumallin käsitteestä. Lisäksi tarkasteltiin keinoja valita oikea malli oikean ongelman ratkaisemiseen, jotta malleja voidaan hyödyntää tehokkaasti. Luvussa käsiteltiin myös mallien soveltamista ohjelmistokehityksen eri vaiheissa sekä niiden integrointia kehitysprosesseihin. Lopuksi luvussa 4 esiteltiin tarkemmat määritelmät kahdelle eri turvallisuusmallille.

Yleisesti ajatellaan, että suunnittelumalleja käyttämällä ohjelmistojen laatu paranee. Näin ollen oletuksena tutkielman alussa oli, että turvallisuusmalleja hyödyntämällä saadaan parannettua myös ohjelmistojen tietoturvaa. Lähdemateriaalin perusteella vaikuttaa siltä, että turvallisuusmallien hyötyä ei yleensä kyseenalaisteta ja tutkimus aiheesta on aktiivista. Toisaalta, Yskoutin ym. (2015) tekemän tutkimuksen tulos oli, että malleja hyödyntämällä ei välttämättä saavuteta merkittävää etua ohjelmistojen tietoturvassa. Samalla tulokset kuitenkin osoittivat, että malleista on apua kokemattomille kehittäjille.

Suunnittelumalleihin siis liittyy vielä ongelmia, jotka vaativat lisää tutkimusta. Tutkielman tulokset osoittivat, että etenkin sopivan mallin valintaan ja toteutukseen tarvitaan tarkempia menetelmiä. Vaikka turvallisuusmallien tutkimus on aktiivista, tutkimustieto niiden hyödyntämisen vaikutuksista ohjelmistojen tietoturvaan on vielä verrattain puutteellista. Jatkossa turvallisuusmallien tutkimuksessa on tarpeen tarkastella enemmän turvallisuusmallien soveltamista, jotta niiden hyödyntämisen vaikutuksista saataisiin vahvempaa tietoa.

Kirjallisuutta

Alvi, A. & Zulkernine, M. 2012. *A Comparative Study of Software Security Pattern Classifications*. 2012 Seventh International Conference on Availability, Reliability and Security.

Bouaziz, R. & Coulette, B. 2012. *Secure Component Based Applications through Security Patterns*. 2012 IEEE International Conference on Green Computing and Communications.

Bouaziz, R., Kallel, S. & Coulette, B. 2013. *An Engineering Process for Security Patterns Application in Component Based Models*. 2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.

Bunke M. & Sohr K. 2011. *An Architecture-Centric Approach to Detecting Security Patterns in Software*. In: Erlingsson Ú., Wieringa R., Zannone N. (eds) Engineering Secure Software and Systems. ESSoS 2011. Lecture Notes in Computer Science, vol 6542. Springer, Berlin, Heidelberg

Duncan, I. & de Muijnck-Hughes, J. 2014. *Security Pattern Evaluation*. 2014 IEEE 8th International Symposium on Service Oriented System Engineering.

Fernandez, Eduardo B. 2013. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. 2013 John Wiley & Sons Ltd. Luvut 1, 5, 6 & 16.

Fernandez, Eduardo B. & Larrondo-Petrie, Maria M. 2010. *Designing Secure SCADA Systems Using Security Patterns*. 2010 43rd Hawaii International Conference on System Sciences.

Ferraz, F., Assad, R. & Meira, S. 2009. *Relating Security Requirements and Design Patterns: Reducing Security Requirements Implementation Impacts with Design Patterns*. 2009 Fourth International Conference on Software Engineering Advances.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley 1994. Luvut 1 & 4.

Guan, H., Yang, H., & Wang, J. 2016. *An Ontology-based Approach to Security Pattern Selection*. International Journal of Automation and Computing. 13(2), April 2016, 168-182.

Hafiz, M., Adamczyk, P. & Johson, Ralph E 2007. *Organizing Security Patterns*. IEEE

Software (Volume: 24, Issue: 4, July-Aug. 2007)

Kobashi, Yoshioka, Okubo, Kaiya & Washizaki 2013. *Validating Security Design Patterns Application Using Model Testing*. 2013 International Conference on Availability, Reliability and Security.

Koenig, A. 1995. *Patterns and Antipatterns*. Journal of Object-Oriented Programming. 8 (1): 46–48

Koskimies, K. & Mikkonen, T. 2005. *Ohjelmistoarkkitehtuurit*. 2005 Talentum Oyj. Luku 5.

Schumacher, M. 2003. *Security Engineering with Patterns*. Springer-Verlag Berlin Heidelberg 2003. Luku 4.

Woolf, N. *DDoS attack that disrupted internet was largest of its kind in history, experts say*. Guardian 26.10.2016. Saatavissa: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet. Viitattu 24.3.2017.

Yskout, K., Scandariato, R. & Joosen, W. 2015. *Do Security Patterns Really Help Designers?* 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering.