Janne Laitinen

TIETOTURVA JAVA-POHJAISISSA INTERNET-SOVELLUKSISSA

Tietotekniikan pro gradu -tutkielma

Ohjelmistotekniikan linja

12/5/2001

Jyväskylän yliopisto
Tietotekniikan laitos

Tekijä: Janne Laitinen

Yhteystiedot: email: Janne.Laitinen@yomi.com, GSM: +358 50 386 5648

Työn nimi: Tietoturva Java-pohjaisissa Internet-sovelluksissa

Title in English: Security in Java Based Internet Applications

Työ: Pro gradu -tutkielma

Sivumäärä: 118

Linja: Ohjelmistotekniikka

Teettäjä: Jyväskylän yliopisto, tietotekniikan laitos

Avainsanat: Java, tietoturva, kryptografia, Internet, WWW

Keywords: Java, security, cryptography, Internet, WWW

Tiivistelmä: Internet tarjoaa laajan alustan hajautettujen sovellusten toteuttamiseen Javalla. Jos sovellusten käytössä on arkaluontoista tietoa, nousee turvallisuuden huomioonotto tärkeään rooliin toteutusta. Tässä tutkielmassa luodaan yleiskatsaus Internetin tietoturvaan liittyviin asioihin ja tarkastellaan tarkemmin Javan tarjoamia turvaominaisuuksia, joihin oleellisena osana kuuluu Javan turvamalli. Käsiteltynä on myös yleisimpien kryptografian menetelmien käyttötapoja Javan kanssa sekä käytettäviä tietoliikenneyhteystekniikoita.

Abstract: Internet offers a vast platform for the implementation of distributed applications with Java. If the use of the applications entails delicate information, the consideration of security has an important role in the implementation. In this thesis I shall cast an overview at questions concerning the security of Internet and I shall examine the security features offered by Java, especially its security model, which forms its integral parts. I shall also treat the most common uses of cryptographic methods with Java as well as data communication techniques that are being used.

Termiluettelo

API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
CA	*Certificate Authority*
CCITSE	*Common Criteria for Information Technology Security Evaluation*
CERT	*Computer Emergency Response Team*
COAST	*Computer Operations Audit and Security Technology*
CORBA	*Common Object Request Broker Architecture*
CRL	*Certificate Revocation List*
DN	*Distinguished Name*
DNSSEC	*Domain Name System Security*
DSA	*Digital Signature Algorithm*
EJB	*Enterprise Java Beans*
IDEA	*International Data Encryption Standard*
IETF	*Internet Engineering Task Force*
IIOP	*Internet Inter-Orb Protocol*
IPSEC	*IP Security Protocol Working Group*
ITM	*Information Technology Management*
ITU	*International Telecommunication Union*
J2EE	*Java2 Enterprise Edition*
JAR	*Java Archive*
JCE	*Java Cryptography Extension*
JDBC	*Java Database Connectivity*
JDK	*Java Development Kit*
JNDI	*Java Naming and Directory Interface*
JVM	*Java Virtual Machine*
MAC	*Message Authentication Code*
ODBC	*Open Database Connectivity*
OMG	*Object Management Group*

ORB	*Object Request Broker*
OSI	*Open System Interconnection*
PCT	*Private Communication Technology*
PEM	*Privacy Enchanced Mail*
PGP	*Pretty Good Privacy*
RFC	*Request For Comments*
RMI	*Remote Method Invocation*
RSA	*Rivest-Shamir-Adleman cryptography algorithm*
S-HTTP	*Secure HTTP*
S/MIME	*Secure / Multipurpose Internet Mail Extensions*
SDSI	*Simple Distributed Security Infrastructure*
Secure RPC	*Secure Remote Procedure Call*
SET	*Secure Electronic Transactions*
SKIP	*Simple Key management for Internet Protocols*
SSH	*Secure Shell*
SSL	*Secure Socket Layer*
TCB	*Trusted Computing Base*
TCSEC	*Trusted Computer System Evaluation Criteria*
TLS	*Transport Layer Security*
Triple-DES	*Triple Data Encryption Standard*

Sisällysluettelo

1 Johdanto

Internetin nopeasti laajentuessa verkon turvallisuuskysymykset ovat nousseet voimakkaasti esille. Yhä useampien yrityksien ja yhteisöjen lähiverkot on liitetty kiinteästi Internetiin. Tästä on seurannut mahdollisuus tehdä sovelluksia, jotka toimivat hajautettuna eri paikoissa avointa verkkoa.

Tyypillisiä esimerkkejä tällaisista sovelluksista on WWW-selaimen avulla käytettävät palvelut, joita voidaan käyttää mistä tahansa Internetiin liitetyltä koneelta. Myös varsinainen sovellus voi olla jaettuna useisiin erillisiin osiin, ”komponentteihin”, jotka on hajautettu eri osiin verkkoa. Kun käytettävä sovellus tarvitsee jotain palvelua, se lataa tarvittavan komponentin etäkoneelta käyttöönsä. Palveluja voidaan suorittaa etäkoneella, jolloin etäkone suorittaa esimerkiksi varsinaisen laskennan ja palauttaa tuloksen palvelua pyytäneen ohjelman käytettäväksi. Tällaisien sovelluksien toteutuksessa täytyy ottaa huomioon ulkopuolisten verkkoyhteyksien aiheuttamat riskit tietoturvalle.

Keskeisimpiä asioita, jotka voivat hajautettujen järjestelmien tietoturvassa epäonnistua, ovat mm. seuraavat [30]: Todentamattoman tiedon vastaanotto ja käyttö, tiedon tai ohjelmakoodin eheyden mahdollinen rikkoutuminen, eri palvelun toimimattomuus, käyttäjän (virheellisten) tekojen hylkääminen, ilkeämielinen ohjelmakoodi ja verkkoliikenteen suuruuden analysointi.

Javasta on tullut suosittu kieli, joka soveltuu käytettäväksi hajautettujen sovelluksien toteutuksiin. Sen suunnittelussa on alunperin otettu huomioon hajautetuissa sovelluksissa tarvittavia turvavaatimuksia ja niinpä se sisältää useita turvaominaisuuksia, joiden avulla voidaan toteuttaa periaatteessa turvallisia hajautettuja sovelluksia myös avoimeen verkkoon. Vaikka sovellukset ovatkin näennäisesti turvallisia, täytyy kuitenkin muistaa, että avoimessa verkossa toimivien sovellusten käytössä piilee aina riski, että myös asiaton pääsee sovellukseen käsiksi. Vaikka johonkin sovellukseen ei tällä hetkellä olisi päästy tunkeutumaan, se ei tarkoita, että sovellus olisi murtamaton.

Javan turvallisuudesta puhuttaessa tarkoitetaan yleensä mahdollisuutta suorittaa tuntemattomia ohjelmia ilman pelkoa, että ohjelmat pääsisivät käsiksi käyttäjän järjestelmään. Näin mahdolliset ilkeämieliset sovelluksetkaan eivät pääse vahingoittamaan

käyttäjän konetta. Lisäksi oleellisen osan turvallisuudesta muodostaa kommunikoivien osapuolinen tunnistaminen ja tietojen siirtyminen muuttumattomana verkon yli. [14]

Tässä työssä esitellään Javan turvamallin rakennetta ja tekniikoita, joita käytetään hajautettujen Java-sovelluksien toteutuksissa. Alussa luodaan johdattelua Internetin yleisiin tietoturvan osa-alueisiin ja myöhemmissä luvuissa keskitytään Javan turvamalliin ja käytettäviin tekniikoihin sovelluksia hajautettaessa.

Luvussa kaksi käsitellään yleisesti Internetin tietoturvan osa-alueita ja niihin kuuluvia asioita. Kolmannessa luvussa esitellään lyhyesti Java-kielen ja -ohjelmien yleistä rakennetta ja toimintaa. Neljäs luku käsittelee Javan tietoturvamallia ja sen eri komponentteja sekä niiden sijoittumista varsinaiseen tietoturvamalliin. Luku viisi keskittyy Javan mahdollisuuksiin ja sen tarjoamiin rajapintoihin eri kryptografisten menetelmien käytössä. Kuudes luku käsittelee muutamia tietoliikenteen salaukseen käytettäviä menetelmiä. Lisäksi luvussa käydään läpi Javan yhteydessä käytettäviä tekniikoita hajautettujen sovellusten yhteydessä. Seitsemännessä luvussa esitellään toteutetun esimerkkisovelluksen rakenne ja toteutus. Viimeinen luku sisältää työn yhteenvedon.

2 Internet-sovelluksien tietoturva

Internet on laajentunut viime vuosien aikana räjähdysmäisesti ja samalla sinne on tullut lukuisia palveluja, joita käytettäessä käyttäjän ja palveluntarjoajan välillä liikkuu myös arkaluontoista tietoa. Tämä on tehnyt tietoturvasta keskeisen osan tietoliikennettä. Tietoturvasta on tullut keskeinen kysymys niin yrityksille, organisaatioille ja valtioille kuin koko yhteiskunnan toiminnalle. Tässä luvussa käydään läpi yleisiä tietoturvan osa-alueita Internet-sovelluksissa.

2.1 Yleistä

Yleisimpiä tietoturvaa vaativia palveluita ovat WWW-pohjaiset kauppapaikat, joita on viime aikoina perustettu yhä enemmän ja enemmän. Näissä käytetään usein maksutapana luottokorttia, jolloin kortin numero joudutaan lähettämään verkon yli asiakkaan koneelta palvelun tarjoajan palvelimelle. Näiden kahden pisteen välinen tietoliikenne täytyy salata, jottei numeroista mahdollisesti kiinnostunut henkilö pääse niitä lukemaan. Tällaisten verkkopalveluiden myötä tietoturvasta on tullut tärkeä osa-alue verkottuneessa yhteiskunnassa.

Tietoturvallisuus koostuu useasta eri tekijästä, sillä niin laitteistot, ohjelmistot, välitettävät tiedot kuin tietoliikenneyhteydet on suojattava. Suojaamismenetelmiä voidaan jaotella teknisiin, fyysisiin ja hallinnollisiin menetelmiin. Tekniset menetelmät perustuvat laitteistojen ja ohjelmistojen tietoturvaratkaisuihin, joista varsinkin jälkimmäiset mielletään yleisesti tietoturvan muodostamaksi kokonaisuudeksi. Fyysisten menetelmien tarkoituksena on estää mahdollisen tunkeutujan pääsy fyysisesti lähiverkkoon. Ne muodostavat yhdessä teknisten menetelmien kanssa tietoturvan perustan, sillä ilman niitä hallinnolliset tietoturvaratkaisut ovat tehottomia. Hallinnollisilla menetelmillä tarkoitetaan tavallisten käyttäjien toimintatapoja ja heille myönnettyjä oikeuksia.

2.2 Tietoturvan osa-alueita

Tietoturva voidaan jaotella eri osa-alueisiin. Internet Engineering Task Force (IETF) on päätynyt luokittelemaan tietoturvan kuuteen eri osa-alueeseen, joita ovat: 1)

luottamuksellisuus, 2) **eheys**, 3) **todennus**, 4) **kiistämättömyys**, 5) **pääsynvalvonta** ja 6) **käytettävyys**. [10]

Tässä jaottelussa luottamuksellisuudella tarkoitetaan sitä, että tietojärjestelmässä oleva tai siirretty tieto on vain niiden käytettävissä, joille se on tarkoitettu. Näin suojataan tiedon omistusoikeutta ja yksityisyyttä. Luottamuksellisuus on tapa suojautua esimerkiksi salakuuntelua tai monitorointia vastaan. Tiedon eheydellä varmistetaan, että tieto pysyy siirrettäessä ja säilytettäessä muuttumattomana. Tämän varmistamiseen liittyy aina lähettäjän todennus. Eheys yhdessä todennuksen kanssa varmistaa sen, että lähettäjän lähettämä tieto menee perille vastaanottajalla muuttumattomana juuri samassa muodossa kuin se lähetettiinkin.

Tiedon luottamuksellisuuden ja eheyden taso ja tarve ovat toisistaan riippumattomia. Tieto voi olla kaikille saatavilla olevaa julkista tietoa, mutta sen on oltava myös oikeaa. Jos esimerkiksi julkisen viranomaistiedotteen sisältö muuttuu, voi siitä aiheutua suurta vahinkoa. Kuitenkin on mahdollista, että jokin tieto voi olla hyvinkin luottamuksellista vaikkei se olekaan virheetöntä. Eheys varmistaa myös sen, että tieto pysyy muuttumattomana vaikka laitteistoissa tai ohjelmistoissa ilmenisi vikoja. Eheysvaatimusta rikotaan aktiivisilla hyökkäyksillä, jolloin alkuperäisiä tietoja pyritään muuttamaan luvattomasti. Hyökkäyksiä ei pelkän eheysvaatimuksen yhteydessä pyritä torjumaan vaan ainoastaan havaitsemaan.

Todennus on menetelmä, jolla jokin tietty taho (henkilö, tieto) erotetaan luotettavasti muista tahoista. Esimerkiksi Internet-pankkipalveluissa käyttäjä todennetaan henkilökohtaisella käyttäjätunnuksella ja salasanalla. Kiistämättömyydellä tarkoitetaan sitä, ettei tiedon lähettäjä voi kiistää lähettäneensä tietoa ja olleensa jossakin tapahtumassa osapuolena. Kiistämättömyys on muoto todeta tietolähteen alkuperä ja se toteutetaan käytännössä sähköisellä allekirjoituksella. Yhdessä todennuksen kanssa tämä on oleellinen osa monien tietoverkkojen kautta tapahtuvien palvelujen toteuttamiselle.

Pääsynvalvonta koostuu toimista, joilla valvotaan käyttäjien pääsyä koneella oleviin tiedostoihin ja palveluihin. Tällöin tarkastetaan, onko tietoa hakevalla osapuolella oikeus päästää käyttämään hakemaansa tietoa tai palvelua. Pääsynvalvonnalla on tarkoitus osaltaan olla varmistamassa tiedon luottamuksellisuutta ja eheyttä. Tällä estetään myös

järjestelmään kohdistuvia hyökkäyksiä, jolloin turvataan tiedon saatavuutta. Tiedon käytettävyys varmistaa sen, että tieto on sellaisten tahojen käytettävissä, joille se on tarkoitettu ja jotka sitä tarvitsevat. Myös käytettävyys voi vaarantua erilaisten tietoja kohtaan tehtyjen hyökkäysten takia. [10, 36]

2.3 Teknisiä tietoturvaratkaisuja

Eri tietoturvan osa-alueilla on omat tyypilliset tekniset ratkaisunsa. Ne perustuvat useimmiten salasanojen ja käyttäjätunnusten käyttöön, virheitä korjaaviin protokolliin, automaattisiin virustarkistuksiin, palomuureihin ja kryptografisiin menetelmiin.

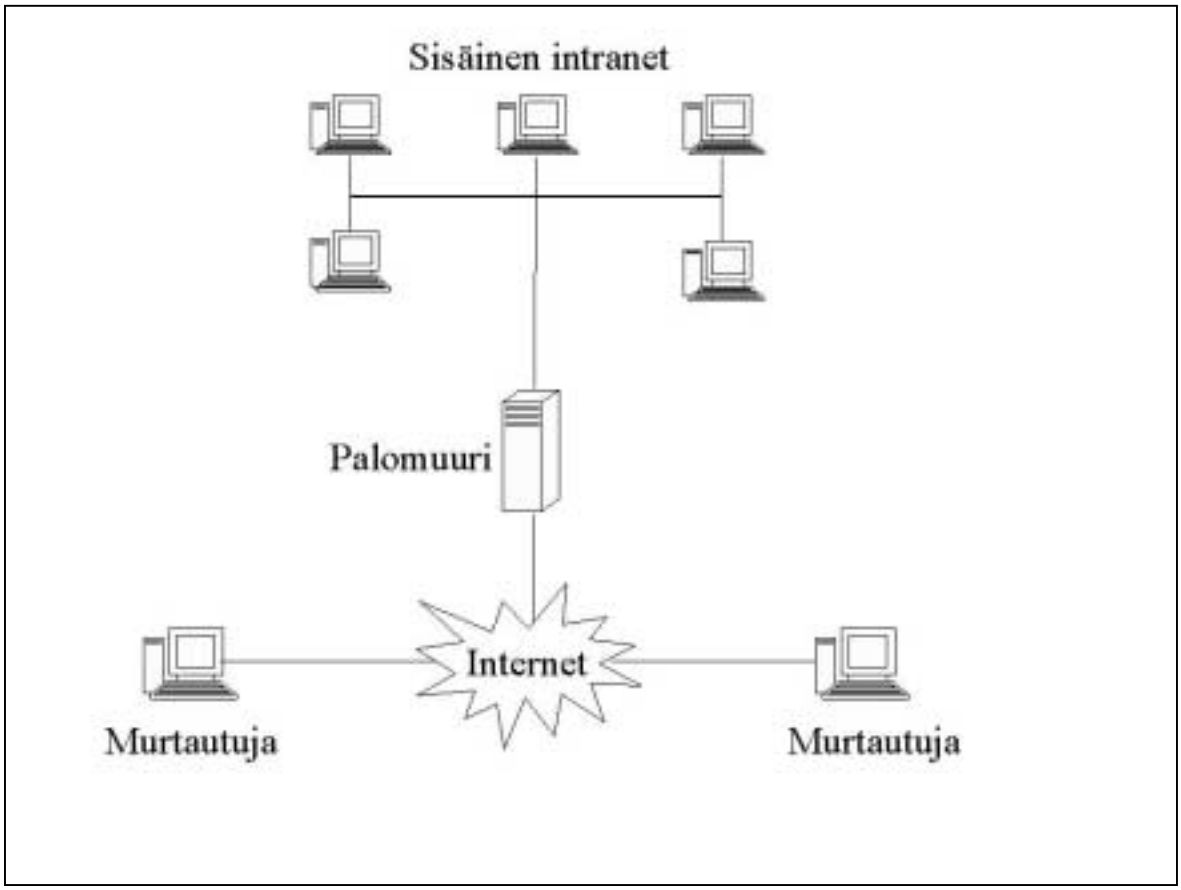
Salasanoilla ja käyttäjätunnuksilla valvotaan eri käyttäjien sisäänpääsyä tietojärjestelmiin ja varmistetaan se, että tieto pysyy eheänä ja luottamuksellisena. Tämä tapahtuu määrittelemällä salasanojen avulla, kenellä on oikeus päästä lukemaan ja muuttamaan mitäkin tietoa. Salasanat eivät ole kuitenkaan täysin luotettava keino, koska salasanoja on usein helppo saada selville tiedonsiirron aikana ja salasanat ovat monesti helposti arvattavia. Virheitä korjaavat protokollat valvovat, että tieto pysyy eheänä tiedonsiirron aikana. Ne toimivat usein jollakin tarkistussummaperiaatteella, jolloin tarkistussumman avulla huomataan, jos bittejä on muuttunut tiedonsiirron aikana.

Automaattisilla virustarkistuksilla pyritään estämään virusten tunkeutuminen tietojärjestelmiin ja yritysten sisäisiin verkkoihin. Automaattisia virustarkistuksia voidaan laittaa myös yksittäisille työasemille, mutta silloin on usein haittana työaseman suorituskyvyn heikkeneminen taustalla pyörivän tarkistusohjelman takia. Toinen mahdollisuus on sijoittaa virustarkistus yrityksen sisäisen verkon ja ulkopuolisen verkon väliselle proxy-palvelimelle, jonka haittana on myös palvelimen toiminnan hidastuminen.

Tekninen tietoturva perustuu aina vahvasti erilaisten kryptografian eli salakirjoitustieteen menetelmien käyttöön. Kryptografisilla menetelmillä pyritään varmistamaan tiedon eheys, luottamuksellisuus, tietolähteen todennus ja kiistämättömyys. Salakirjoituksella tarkoitetaan jotain matemaattista menetelmää, jolla selväkielinen viesti muutetaan ulkopuoliselle ymmärtämättömään muotoon. Viestin oikealla vastaanottajalla on ns. salakirjoitusavain, joka määrää kuinka salakirjoitetun viestin muunnos tapahtuu ja jonka avulla viestin saa purettua takaisin alkuperäiseen muotoonsa.

2.4 Palomuurit

Palomuuri toimii yrityksen sisäisen verkon ja ulkopuolisen verkon (yleensä Internet) välisenä erottajana. Palomuuri estää luvattoman pääsyn suljettuun sisäiseen verkkoon tahoilta, jotka saattavat olla kiinnostuneita verkon sisällöstä. Palomuurin avulla tietoturvapalvelut saadaan keskitettyä yhteen pisteeseen, jolloin tietoturvapalvelujen asennus ja ylläpito tulevat yksinkertaisimmiksi. Kuvassa 1 on esitetty palomuurin tyypillinen sijoituspaikka yrityksen sisäisen intranetin ja ulkoisen verkon väliin. Näin palomuuri estää yrityksen verkkoon yrittävän murtautujan sisäänpääsyn. Yleensä palomuuri sisältää liikennettä suodattavan reitittimen ja proxy-palvelimen.



Kuva 1: Palomuurin tyypillinen sijoituspaikka.

Suodattavan reitittimen toteutus on pääpiirteissään yksinkertaista. Reititin tarkistaa sille tehtyjen asetuksien perusteella jokaisen läpimenoa yrittävän paketin otsikkotiedoista, voiko paketti jatkaa matkaansa. Otsikkotiedoissa on mm. paketin lähtö- ja

määränpääosoitteet. Proxy-palvelin toimii yhdyskäytävänä esimerkiksi telnet-, HTTP- ja FTP-yhteyksillä. Palvelimen avulla voidaan toteuttaa sovellukseen kirjautuminen, käyttäjän tunnistus ja liikenteen yleinen kirjaaminen. Yleisesti palvelin tarkistaa käyttäjien oikeudet liikennöintiin käyttäjätunnuksen ja –salasanan perusteella. Ulospäin menevät paketit otsikoidaan yleensä siten, että kaikki lähtevät paketit näyttävät tulevan palomuurista, jolloin pakettien lähettäjän tarkka osoite ei selviä ulkopuolisille.

Palomuuritkaan eivät ratkaise kaikkia tietoturvauhkia. Myös sallituiksi määritellyistä osoitteista voi tulla viruksia ja sisäisessä verkossa palomuurin takanakin voi olla tietoturvauhkia. Palomuurit voivat aiheuttaa myös liiallista turvallisuuden tunnetta, jolloin saatetaan luottaa kiinteisiin salasanoihin niiden säännöllisen vaihtamisen sijasta. Ja jos joku onnistuu murtamaan palomuurin, on hänellä silloin esteetön pääsy kaikkiin palomuurin takana oleviin koneisiin. Tämän takia saattaa olla tarpeellista sijoittaa myös intranetin sisällä tärkeiden kohteiden eteen vielä erillinen palomuuri.

Palomuurien hyvien puolien lisäksi niistä aiheutuu myös haittoja. Palomuurista saattaa muodostua pullonkaula, jos intranetin ja ulkopuolisen verkon välillä on paljon liikennettä. Uusimmissa palomuureissa nopeus on kasvanut, joten tätä ongelmaa ei usein ilmene. Lisäksi palomuuri saattaa rajoittaa pääsyä palveluihin, joihin käyttäjillä olisi tarvetta, kuten telnet- ja FTP-palveluihin. Palomuurit eivät myöskään suojaa siinä tapauksessa, jos intranetin johonkin koneeseen on pääsy suojaamattoman modeemin välityksellä. [4, 23, 27]

2.5 Käyttäjiin perustuva tietoturva

Mitkään tekniset ratkaisut eivät takaa tietoturvallisuutta, elleivät myös käyttäjät itse huolehdi oikeista toimintatavoistaan. Erityisen tärkeässä osassa ovat tietojärjestelmistä ja verkoista vastaavat henkilöt, joiden täytyy huolehtia, että järjestelmien turvaratkaisut rakennetaan tietoturvan kannalta oikein. Oleellinen osa toimintaa on myös jatkuva verkkojen tarkkailu ja nopea reagoiminen epätavallisiin tilanteisiin. Vastuuhenkilöillä on yleinen salassapitovelvollisuus luottamuksellisten tietojen osalta, ja heidän täytyy tietää, mitä tietoja käyttäjistä saa kerätä ja kuinka.

Tavallisten käyttäjien toimintatavat ratkaisevat myös hyvin paljon. Jos joku huolimaton tai tietoturvasta piittaamaton henkilö jättää käyttäjätunnuksen ja salasanan helposti havaittavaan paikkaan, on tietoturva-aukko valmis. Samoin palomuurin ohittamista esimerkiksi modeemilla on vältettävä. Yleisesti tietoturva vaarantuu eniten, jos tavallisella käyttäjällä on liian laajat käyttöoikeudet ja välinpitämätön asenne tietoturvaa kohtaan. Yritysten tietoturvaratkaisuissa on muistettava suhteuttaa suojan taso riskin suuruuteen eli vältettävä turvajärjestelyiden ylimitoitusta. Ylimitoitetussa järjestelmässä kaikki eivät jaksa noudattaa turvamenetelmiä, jolloin tietoturvan taso saattaa laskea.

2.6 Kryptografia

Kryptografia eli salakirjoitustiede ja siihen perustuvat menetelmät ovat nykyään nousseet keskeiseen osaan tietoturvaan liittyvien ongelmien ratkaisemisessa. Salakirjoituksella (*engl. encryption*) tarkoitetaan toimintaa, jossa alkuperäinen viesti muutetaan jollakin matemaattisella menetelmällä ulkopuoliselle käsittämättömään muotoon. Tämä menetelmä perustuu salakirjoitusavaimen käyttöön, joka määrää viestin muunnoksen. Viestin vastaanottaja käyttää tulkinta-avaimeen perustuvaa käänteistä menetelmää, jolla viesti tulkitaan (*engl. decryption*) takaisin selkokieliseksi.

Salakirjoitusmenetelmät jaotellaan julkisen avaimen (*engl. Public Key Cryptography*) ja salaisen avaimen (*engl. Secret Key Cryptography*) menetelmiin. Julkisen avaimen käyttöön perustuvassa salauksessa käytetään salaukseen ja purkamiseen eri avaimia. Tietojen välittämistä varten kumpikin osapuoli luo itselleen kaksi avainta, julkisen ja salaisen. Nämä avaimet ovat toisistaan riippuvaisia siten, että salaisella avaimella salattu viesti voidaan purkaa vain toista (julkista) avainta käyttäen. Tällöin tietoja lähettävä osapuoli voi jakaa julkisen avaimen kaikille, joille hänelle on tarve lähettää salattua tietoa.

Salaisen avaimen menetelmässä käytetään samaa avainta, jonka tulee olla sekä lähettäjän että vastaanottajan tiedossa, sekä tiedon salaamiseen että purkamiseen. Yleensä tietojen salauksessa käytetään salaisen avaimen menetelmää, koska julkisen avaimen menetelmät on huomattavasti raskaampia. Julkisen avaimen menetelmää käytetään lähinnä salaisen avaimen välitykseen tiedon salaukseen käytettävää salakirjoitusta varten sekä todennuksen, kiistämättömyyden ja eheyden varmistamiseen.

Salakirjoitusmenetelmän tehokkuuteen vaikuttaa käytetyn avaimen pituus, käytetyn algoritmin vahvuus ja algoritmiin mahdollisesti kätketyt salaovet. Käytetyn avaimen pituus tulee olla sellainen, ettei sen murtaminen onnistu nykyaikaisilla tietokoneilla järkevässä ajassa. Nykyisillä laskentatehoilla riittävä avaimen pituus symmetrisissä menetelmissä on yli 100 bittiä ja asymmetrisissä menetelmissä yli 1000 bittiä. Algoritmin salaovilla tarkoitetaan mm. sitä, että salatusta aineistosta pystyy päättelemään salaisen avaimen.

Käytetyn algoritmin vahvuutta eli sitä, kuinka vaikeaa algoritmin murtaminen on, täytyy kuitenkin tulevaisuudessa kasvattaa yhä enemmän ja enemmän tietokoneiden laskentatehon kasvaessa. Tällä hetkellä niin sanottuja vahvoja salakirjoitusmenetelmiä ovat mm. symmetrisistä (salaisen avaimen) menetelmistä Triple-DES, IDEA ja Blowfish sekä asymmetrisistä (julkisen avaimen) menetelmistä RSA ja DSS. Lupaavia uusimpia julkisen avaimen menetelmiä ovat elliptisten käyrien menetelmät. Niiden uskotaan olevan nykyisin käytössä olevia menetelmiä tehokkaampia laskennaltaan ja siten kevyempiä ratkaisuja esimerkiksi erilaisiin älykortteihin.

Julkisen avaimen kryptografian sovelluksista digitaalinen allekirjoitus on eräs tärkeimmistä tietoturvan palveluista. Sitä tarvitaan monissa tietoverkkosovelluksissa, joissa tietyn dokumentin alkuperä täytyy todentaa. Viestin lähetyksen yhteydessä lähettäjän digitaalinen allekirjoitus luodaan hänen henkilökohtaisella salaisella avaimella ja vastaanottaja todentaa sanoman lähettäjän julkisella avaimella. Digitaalisen allekirjoituksen toteuttamiseen käytetään nykyisin kolmea menetelmää, jotka ovat hash-funktioiden käyttö, RSA-menetelmät ja DSS-standardi. Hash-funktio on iteratiivinen yksisuuntainen funktio (ts. tiivistettä ei voi muuttaa takaisin sanomaksi), joka muodostaa mielivaltaisen pitkästä sisääntulosanomasta vakiomittaisen tiivisteen. Tiivistettä ei voi enää palauttaa alkuperäiseksi sanomaksi.

RSA-menetelmä ja esimerkki sen käytöstä

RSA-menetelmän käyttö perustuu seuraavaan tulokseen [28]:

Olkoon p ja q eri alkulukuja ja suurempia kuin 1, n = pq ja m = (p – 1)(q – 1). Olkoon edelleen kokonaisluku e ≥ 2 sellainen, että e ≡ 1 (mod m), ja kokonaisluku x sellainen, että x:n ja n:n suurin yhteinen tekijä on 1. Silloin pätee, että x^e ≡ x (mod n).

Todistus sivuutetaan. Edellä yhtälöllä muotoa a ≡ b (mod c) tarkoitetaan sitä, että erotus a – b on kokonaisluvulle c tasan jaollinen, eli luvut a ja b ovat keskenään kongruentteja modulo c. Käänteisalkion määritelmä on tässä seuraava: renkaassa Z(n) alkio b on alkion a käänteisalkio, jos a≡1 (mod n), merkitään b=a^(-1)

Oheisessa esimerkissä käytetään jakoyhtälöä m = qn + r, jossa 0 ≤ r ≤ n. Jakoyhtälössä m on jaettava, n on jakaja, q on osamäärä ja r jakojäännös.

Esimerkki.

*Koodin laatija valitsee kaksi suurta alkulukua, esimerkiksi p = 11 ja q = 13. Laatija laskee luvut n = pq = 143 ja m = (p – 1)(q – 1) = 120. Laatija valitsee koodausavaimen k siten, että k:n ja m:n suurin yhteinen tekijä on 1, esimerkiksi k = 7. Laatija laskee luvulle k käänteisalkion k' = k^(-1). Jakoyhtälön mukaisesti 120 = 17 * 7 +1, joten 17 * 7 = 120 – 1 ≡ –1 (mod 120). Siten –17 * 7 ≡1 ja edelleen (120 – 17) * 7 ≡1 eli 103 * 7 ≡ 1. Niinpä nyt k' = 103. Laatija antaa viestin lähettäjälle tiedoksi vain luvut k ja n ja vastaanottajalle luvut k' ja n. Luku k' on purkuavain.*

*Lähettäjä lähettää viestin x koodattuna kaavalla r ≡x^k (mod n). Jos x = 9, niin r ≡ 9^7 (mod 143). Koska 9^2 = 81 ja 9^4 = 6561 = 43 * 143 + 126 ≡126, niin koodatuksi viestiksi saadaan r ≡9^7 ≡9^(4 + 2 + 1) ≡9^4 + 9^2 * 9 ≡126 * 81* 9 ≡91854 ≡48.*

*Vastaanottaja purkaa koodatun viestin r kaavalla x ≡ r^(k') (mod n), eli x ≡48^103 (mod 143). Purkutulos voidaan varmistaa seuraavasti: Kun lasketaan modulo 143, niin 48^2 ≡2404 ≡16, 48^4 ≡16^2 ≡113, 48^8 ≡113^2 ≡42, 48^16 ≡42^2 ≡48, 48^32 ≡ 48^2 ≡16, 48^64 ≡16^2 ≡113 ja siten 48^103 ≡48^(64 + 32 + 4 + 2 + 1) ≡48^64 * 48^32 * 48^4 * 48^2 * 48 ≡113 * 16 * 113 * 16 * 113 ≡9 ≡x.*

Hash-funktioon perustuvassa digitaalisessa allekirjoituksessa lähetettävä sanoma tiivistetään ensin hash-funktiolla, tiiviste salataan salaisella avaimella, liitetään se alkuperäiseen sanomaan ja lähetetään vastaanottajalle. Tämä alkuperäisen sanoman ja salatun tiivisteen yhdistelmä toimii allekirjoituksena. Digitaalinen allekirjoitus voidaan nyt varmentaa muodostamalla vastaanotetusta alkuperäisestä sanomasta tiiviste samalla hash-funktiolla kuin lähettäjäkin ja salata se samalla salaisella avaimella. Jos tämä salattu tiiviste

ja vastaanotettu salattu tiiviste ovat samoja, on vastaanotettu sanoma aito ja ehyt. Menetelmä on helpohkosti murrettavissa, koska vastaanottajalla täytyy olla kopio lähettäjän salaisesta avaimesta. Hash-funktioiden ominaisuuksiin kuitenkin kuuluu, että vaikka kaksi sanomaa olisivat lähes samanlaisia sisältäen vain pienen eron, ovat niiden tiivisteet täysin erilaisia.

RSA-menetelmää käytettäessä hash-funktiolla tehdään ensin tiiviste, salataan se lähettäjän salaisella avaimella ja liitetään alkuperäiseen sanomaan, jonka jälkeen koko alkuperäinen sanoma ja salattu tiiviste lähetetään vastaanottajalle. Vastaanottaja erottelee tiivisteen ja sanoman toisistaan, laskee hash-funktiolla vastaanotetusta sanomasta todellisen tiivisteen ja purkaa vastaanotetun salatun tiivisteen lähettäjän RSA-avaimella. Jos nyt molemmat tiivisteet ovat täysin identtiset, on vastaanotettu sanoma aito. Allekirjoitus voidaan toteuttaa myös ilman hash-funktioiden käyttöä, mutta tällöin RSA-salausta täytyy soveltaa koko sanomaan ja tämän toteuttaminen on raskasta.

DSS-standardi on yleisesti käytössä oleva digitaalisen allekirjoituksen menetelmä. Se perustuu DSA-algoritmin käyttöön, joka käyttää digitaalisen allekirjoituksen tuottamiseen diskreetin logaritmin ongelmaa. Menetelmän mukainen digitaalinen allekirjoitus muodostetaan samaan tapaan kuin edellä RSA-menetelmässä, mutta salauksessa käytetään RSA-algoritmin sijasta DSA-algoritmia. [13, 23, 25]

2.7 Internetin tietoturvaprotokollat

Internetissä on useita tietoturvaan liittyviä protokollia, koska sen palveluita ja yhteyksiä voidaan käyttää monin eri tavoin. Internetin protokollien kehityksessä ei tietoturvaa ole alunperin otettu huomioon ja tietoturvan merkitys onkin kasvanut vasta 90-luvulla. Seuraavassa taulukossa on lueteltu tärkeimpiä Internetin tietoturvaan liittyviä protokollia.

Protokolla	**Tarkoitus**

CyberCash	Elektroninen maksaminen
DNSSEC	Domain-nimijärjestelmän tietoturva
IPSEC	Pakettitason salaus
PCT	TCP/IP-tason salaus
PGP	Suojattu sähköposti
S/MIME	Suojattu sähköposti
S-HTTP	Web-salauksen suojaus
Secure RPC	Etäproseduurikutsujen suojaaminen
SET	Luotettava elektroninen maksaminen
SSL	TCP/IP-tason salaus
SSH	Suojatut pääteyhteydet
TLS	TCP/IP-tason salaus

Taulukko 1: Internetin tietoturvaprotokollia.

CyberCash:n ideana on toimia neutraalina kolmantena osapuolena ostotapahtumassa, johon osallistuu ostajan ja myyjän lisäksi myös perinteinen pankki. CyberCash-protokolla muodostaa luotettavan tavan siirtää maksusuoritus ostajan tililtä myyjän tilille verkossa. DNSSEC on Internetin nimipalvelimien käyttämä sertifiointimenetelmä, jota käytetään dynaamisissa yritysverkoissa käyttäjien tunnistamiseen. DNSSEC tunnistaa verkon palvelukoneet julkisen avaimen sertifikaattien avulla.

IPSEC on laajassa käytössä olevien IP-verkkoprotokollien laajennus, jolla salataan ja sinetöidään jokainen verkkoon lähetettävä paketti. PCT on Microsoftin kehittämä salausprotokolla, joka vastaan Netscapen SSL:ää. PCT on sovellusriippumaton ja sen toimintaperiaatteena on, että palvelin tunnistetaan aina ja samoin käyttäjä, jos palvelin ei tätä tunne. PGP on yleinen sähköpostin salaukseen käytetty menetelmä, joka tarjoaa turvapalveluista autentikoinnin (käyttää digitaalista allekirjoitusta) sekä luotettavuuden (käyttää salausta). S/MIME on sähköpostin salaukseen tarkoitettu protokolla. Se varmistaa sanoman todennuksen, eheyden ja kiistämättömyyden digitaalisen allekirjoituksen avulla sekä sanoman salaamisen kryptausta käyttäen.

S-HTTP (Secure HTTP) on turvaominaisuuksin varustettu versio HTTP:stä WWW-käyttöä varten. S-HTTP:n avulla jokaiselle dokumentille voidaan määritellä oma turvatasonsa: salataan ja/tai allekirjoitetaan. Se toteuttaa luotettavuuden, todennuksen, sanoman eheyden ja kiistämättömyyden osa-alueet. SET on Visan ja MasterCardin

kehittämä menetelmä turvallisiin maksuihin verkossa luottokortin avulla. Se perustuu vahvojen salaus- ja allekirjoitusmenetelmien käyttöön.

SSL on yleisesti käytetty avoimia tietoverkkoja käyttävien kahden sovelluksen välinen turvallinen istuntotaso. Se voidaan yhdistää suoraan esimerkiksi selaimeen tai se voi toimia useiden sovellusten käytössä. SSH on vahvoja kryptografisia algoritmeja käyttävä pääteyhteyksien salauksiin tarkoitettu menetelmä. TLS-protokolla perustuu Netscapen julkaisemaan SSL-protokollaan. TLS on alaspäin yhteensopiva SSL:n kanssa, joten TLS toteutus pystyy kommunikoimaan SSL:ään perustuvien toteutuksien kanssa. [4, 13, 27]

2.8 Tietoturvan arvioinnista

Tietoturvan arvioinnissa on usein käytetty 80-luvun alkupuoliskolta peräisin olevaa USA:n puolustusministeriön teosta U.S. Trusted Computer System Evaluation Criteria (*TCSEC*), joka tunnetaan yleisesti ”oranssina kirjana”. Sen tavoitteena on tarjota standardimittari erilaisten tietojärjestelmien turvallisuuden vertailuun, ohjeistaa turvallisten järjestelmien suunnitteluperusteita ja tarjota väline turvallisuusvaatimusten määrittelyyn. [1]

Oranssi kirja jaottelee järjestelmien turvallisuuden neljään pääluokkaan A, B, C ja D (joista A on paras). Lisäksi nämä luokat on jaoteltu yksityiskohtaisempiin luokkiin A1, B3, B2, B1, C2 ja C1. D-luokkaan kuuluvaa järjestelmää voidaan pitää käytännössä turvattomana, jollaisia itse asiassa monet nykyiset järjestelmät ovatkin. C-luokkaan kuuluvan järjestelmän täytyy jo tarjota useita turvatarkastuksia, joista yksi tärkeimmistä on käyttäjän todennus. B-luokkaan kuuluvassa järjestelmässä turvallisuus on täytynyt ottaa huomioon jo järjestelmän suunnitteluvaiheessa ja A-luokan järjestelmän on rakennuttava todistetun formaalin turvallisuusmallin pohjalle. [1]

Toiminnallinen luokka	**Ryhmiä**
Turva-auditointi	Auditoitavien tapahtumien valinta, tietojen keruu, tallennus, analyysi, automaattinen reagointi, tietojen tarkastelu.
Kryptografinen tuki	Avaintenhallinta, kryptografisten algoritmien asianmukainen käyttö.
Tietoliikenne	Alkuperän kiistämättömyys, vastaanottajan kiistämättömyys.
Käyttäjän tietojen suojaaminen	Pääsynvalvonnan politiikka ja toimet, datan autentikointi, talletetun tiedon eheys, takaisinkierto, hävitetyn tiedon suojaus, kohdejärjestelmän sisäinen tiedonsiirto.
Tunnistus ja autentikointi	Käyttäjän tunnistaminen ja autentikointi, autentikoinnin

	epäonnistumisen seuraukset, käyttäjän turva-attribuuttien määrittely, käyttäjän ja käyttäjän prosessien turva-attribuuttien välinen kytkentä ja salasanojen spesifiointi.
Turvahallinto	Turva-attribuutit, niiden peruuttaminen ja voimassaolon päättyminen, turvamekanismit ja niihin liittyvät tiedot, tietoturvaan liittyvien roolien määrittelyt.
Yksityisyys	Nimettömyys, peitenimisyys, kytkemättömyys, resurssin tai palvelun käytön havaitsemattomuus.
Luotettujen toimintojen suojaaminen	Taustalla olevan abstraktion testaus, kaatuessa turvallinen, luotettava toipuminen, fyysinen suoja, toistojen havaitseminen, viittausten välitys, vyöhykkeiden erottelu, aikaleimat.
Resurssien hyödyntäminen	Vikasietoisuus, prioriteettien käyttö, palvelun eston torjuminen resurssien jaolla.
Käyttäjän istuntojen muodostaminen	Istunnon muodostamisen estäminen, käyttäjälle näytettävät tiedotteet ja varoitukset, tiedot aiemmista istunnoista.
Luotettava polku tai kanava	Käyttäjän ja järjestelmän välillä, järjestelmän erillisten osien välillä.

Taulukko 2: Toiminnalliset luokat ja niihin kuuluvia ryhmiä CCITSE-mallissa. [5]

Tämän ja muutaman muun standardin myötä on koottu malli nimeltä Common Criteria for Information Technology Security Evaluation (*CCITSE*). Tästä mallista on tullut myös ISO-standardi IS 15408. Siinä turvallisuutta koskevat vaatimukset muodostavat hierarkisen rakenteen, joiden osilla on lisäksi keskinäisiä riippuvuuksia. Päätasolla on yksitoista toiminnallista ja seitsemän vakuuttavuusluokkaa. Tämän luokkajaon alla on ryhmä-, komponentti- ja alkiotasot. Edellisellä sivulla on taulukoitu toiminnalliset luokat siihen kuuluvine ryhmineen ja seuraavassa taulukossa on vastaava esitys vakuuttavuusluokista.

Vakuuttavuusluokka	**Ryhmiä**
Konfiguraation hallinta	Automatisointi, kyvykkyys, laajuus.
Järjestelmän toimitus ja toiminta	Toimitus ja jakelu, asennus, generointi ja aloitus.
Kehitystyö	Turvapolitiikan mallintaminen, toiminnallinen spesifikaatio, korkean ja matalan tason suunnittelu, toteutuksen esitys, turvamekanismin sisäiset ominaisuudet.
Ohjemateriaali	Ylläpitäjälle, käyttäjälle.
Elinkaaren tuki	Kehitysympäristön turvallisuus, vikojen korjaaminen, elinkaaren määrittely, työkalut ja tekniikat.
Testit	Kattavuus, syvyys eli yksityiskohtaisuus, toiminnallinen testaus, riippumaton testaus arvioijan toimesta
Haavoittuvuuden arviointi	Piilokanavien analyysi, havaitsematta jäävän väärinkäytön torjunta, turvamekanismien vahvuus suoran hyökkäyksen kannalta, haavoittuvuusanalyysi.

Taulukko 3: Vakuuttavuusluokat ja niihin kuuluvia ryhmiä CCITSE-mallissa. [5]

Toiminnalliset komponentit määrittelevät sen, mitä järjestelmän täytyy pystyä tekemään. Niistä kootaan paketteja, mutta arviointikriteerejä niistä muodostuu vasta, kun niihin on yhdistetty jokin vakuuttavuustaso. Vakuuttavuuskomponentit ilmaisevat vaatimuksia järjestelmän kehittäjän toimille ja niiden järjestykselle. Ne muodostavat hierarkian, joka tarjoaa osin vastaavuutta oranssin kirjan luokitukseen verrattuna.

Valituista komponenteista kootaan suojaprofiili yhdistämällä niihin jokin taulukossa 4 olevista arvioinnin vakuuttavuustasoista. Puoliformaalilla tarkoitetaan taulukossa sitä, että asia on ilmaistu kielellä, jolla on rajattu syntaksi ja määritelty semantiikka. Suojaprofiili on toteutuksesta riippumaton joukko vaatimuksia ja sen tarkoitus on olla käytettävissä samanlaisena useissa samantyyppisissä järjestelmissä. Järjestelmää voidaan arvioida suhteessa johonkin olemassaolevaan suojaprofiiliin tai mihin tahansa komponenttipakettiin, johon on liitetty jokin vakuuttavuustaso. Tiettyä järjestelmää voidaan arvioida myös pelkästään suhteessa sen omaan turvatavoitteeseen, joka vastaa rakenteeltaan turvaprofiilia, ja on tyypillisesti tuotteen valmistajan itse laatima. [5]

Vakuuttavuustaso	**Tason vaatimus**
EAL1	Toiminnallisesti testattu.
EAL2	Rakenteellisesti testattu.
EAL3	Metodologisesti testattu ja tarkastettu.
EAL4	Metodologisesti suunniteltu, testattu ja katselmoitu.
EAL5	Puoliformaalisti suunniteltu ja testattu.
EAL6	Puoliformaalisti verifioitu suunnitelma ja testattu.
EAL7	Formaalisti verifioitu suunnitelma ja testattu.

Taulukko 4: Arvioinnin vakuuttavuustasot CCITSE-mallissa. [5]

2.9 Virukset

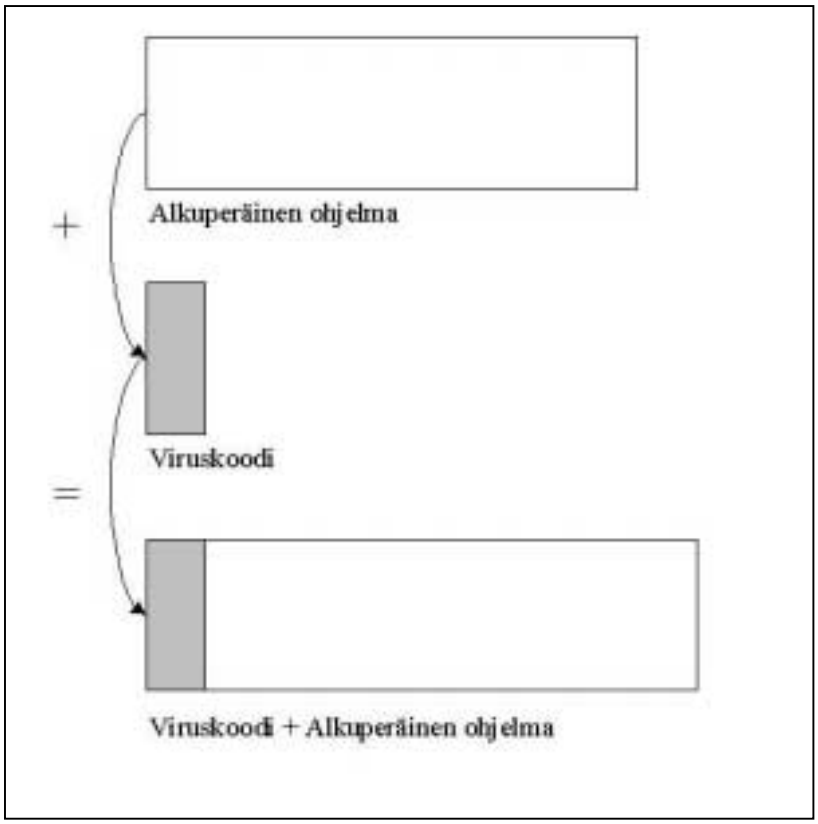
Sanalla virus tarkoitetaan tietokoneohjelmaa, jonka tarkoituksena on levitä mahdollisimman huomaamattomasti ohjelmasta toiseen. Virus tarvitsee isäntäohjelman, johon se liittää oman ohjelmakoodinsa. Tämän jälkeen virus voi aktivoitua joko heti, jonakin tiettynä ajankohtana tai tietyn toiminnon yhteydessä. Viruksia voidaan jaotella leviämistapojensa perusteella eri alaryhmiin kuten tiedosto-, makro-, linkki- ja käynnistyslohkoviruksiin.

Ensimmäisenä PC-viruksena pidetään vuonna 1986 havaittua *Brainiä*. Ensimmäisten virusten pääasiallinen leviämiskanava olivat levykkeet, mutta nykyisin Internet toimii avoimena kanavana, jota myöten viruksetkin pääsevät leviämään helposti ja nopeasti. Niinpä virukset aiheuttavat merkittäviä tietoturvauhkia vuosittain.

Nykyisin yleisin virustyyppi on makrovirus, joka leviää esimerkiksi sähköpostin liitetiedostojen mukana. Vaikka makromekanismit ovatkin hyödyllisiä monissa ohjelmissa, ovat ne tehneet virusten teon entistä helpommaksi. Erityisesti Word- ja Excel-dokumentit ovat suosittuja virusten tekijöiden kohteita, koska niillä on suuri käyttäjämäärä ja niiden makrokieli on joustavaa.

Troijan hevoset ovat ohjelmia, jotka toimivat erilailla kuin käyttäjä odottaa. Ne saattavat toki tehdä sen mitä lupaavatkin, mutta niihin on kätketty myös muu, usein haitallinen, toiminto. Esimerkiksi sisäänkirjautumis-skriptiksi naamioitu Troijan hevonen saattaa toimia normaalisti laskemalla käyttäjän tunnuksella ja salasanalla sisään, mutta tämän lisäksi se voi tallentaa tunnukset ja salasanat myöhempää haitallista käyttöä varten.

Madot ovat itseään verkossa levittäviä ohjelmia. Ne eivät tarvitse isäntäohjelmaa vaan toimivat itsenäisinä ohjelmina. Tunnetuin esimerkki madoista on Robert Morrisin *Internet Worm* vuodelta 1988. Se jumiutti koko silloisen Internetin kopioimalla itseään hetkessä tuhansiin koneisiin. Muita nimityksiä erilaisille haitallisille ohjelmakoodeille ovat



Kuva 2: Viruksen tarttuminen ohjelmaan.

esimerkiksi *malmware*, joka on yleisnimitys kaikille vahinkoa tuottaville ohjelmille, sekä *hoax*, jolla tarkoitetaan aiheettomia virusvaroituksia.

Virukset leviävät tarttumalla ohjelmasta toiseen. Virukset voivat tarttua mihin tahansa tiedostoon, mutta leviämisen edistämiseksi ne on yleensä tehty tarttumaan vain ajokelpoisiin kohteisiin. Tarttuminen tapahtuu yksinkertaisimmillaan kuvassa 2 esitetyllä tavalla, jossa virus kirjoittaa koodinsa varsinaisen ohjelmakoodin alkuun. Tämä on hyvin tehokas tapa, koska viruksen kirjoittajan ei tarvitse tietää mitään kohdeohjelman rakenteesta. Toinen hieman monimutkaisempi tapa on, että viruskoodi sijoittuu osaksi suoritettavan ohjelman alkuun ja osaksi loppuun. Nykyisin useissa tekstinkäsittelyohjelmissa on ns. makroja, jotka sisältävät suoritettavaa koodia. Näillä makrokielillä tehdyt makrovirukset ovatkin nykyisin yleisin virustyyppi ja ne leviävät helposti makroja sisältävien dokumenttien mukana.

Yleensä viruksista puhuttaessa huomioidaan vain niiden aktivoituminen, vaikka virusten pääasiallinen tarkoitus on leviäminen. Usein käyttäjä ei edes huomaa viruksen tarttuneen ohjelmaan ennen sen aktivoitumista. Aktivoituessaan virus voi tehdä miltei mitä tahansa ilkeämielistä, tai sitten se vain ilmoittaa olemassaolostaan käyttäjälle jollakin kyseiselle virukselle ominaisella tavalla. Käyttäjän kannalta suurinta vahinkoa aiheuttava toiminta on vähitellen tapahtuva tiedostojen vahingoittaminen. Tällöin ennen viruksen havaitsemistakin tehdyt varmuuskopiot sisältävät yleensä jo vahingoittuneita tietoja.

Virusten tärkein torjuntakeino on käyttäjän oma varovaisuus, jolloin kaikki tuntemattomat liitetiedostot jätetään avaamatta ennen niiden tarkastusta virustorjuntaohjelmalla. Samoin alkuperältään tuntemattomien ohjelmien käyttöä on hyvä välttää ja tärkeiden tiedostojen varmuuskopiointi on syytä suorittaa säännöllisesti. Torjuntaohjelmat ovat tapa etsiä mahdollisia viruksia kovalevyltä. Tällaisen ohjelman toiminta perustuu sen sisältämään arkistoon viruksista, joita verrataan ohjelmalle annettuun ohjelmanpätkään (joka on esimerkiksi dokumentissa olevaa makrokoodia). Jos arkistosta löytyy verrattavaan koodiin nähden identtinen ohjelmanpätkä, on virus löytynyt. [13]

2.10 Sähköpostin tietoturva

Internet sähköposti on alunperin perustunut SMTP-siirtoprotokollaan ja sanomien rakenne ITM-standardiin. ITM:n lisäksi järjestelmään on standardoitu erityyppisten sanomaobjektien kuten teksti, grafiikka, kuva, video ja salattu sanoma välittäminen järjestelmien välillä MIME-protokollan avulla. SMTP reitittää sanomat luotettavaa kanavaa pitkin lähettäjältä vastaanottajalle. Se ei kuitenkaan sisällä minkäänlaisia tietoturvaominaisuuksia, joten lähetettyjen sanomien sisältöä on ulkopuolisten suhteellisen helppo väärentää. Tämän takia 1995 aloitettiin turvallisemman S/MIME-standardin kehittäminen. Siinä lisättiin MIME-standardiin kryptografisia ominaisuuksia, joita ovat digitaalinen allekirjoitus ja tietojen salaus kryptauksella. S/MIME on saavuttanut sähköpostituotteissa defacto-standardin aseman ja se on määritelty virallisesti IETF:n RFC-standardeissa.

Yleisin sähköpostin turvaamiseen tarkoitettu menetelmä on PGP, jossa on käytössä sekä symmetrinen että asymmetrinen salaus ja digitaalinen allekirjoitus hash-funktion avulla. PGP yhdistää näiden kryptografian menetelmien parhaat puolet ja on erittäin varma ja turvallinen. Sen ainoa heikkous on julkisen avaimen jakelu. Tähän on turvallisinta käyttää ns. luotettavaa kolmatta osapuolta, joka antaa varmasti oikean avaimen. Toinen yleinen sähköpostin salausmenetelmä on RSA:han ja julkisen avaimen käyttöön perustuva PEM-menetelmä, jossa lähettäjä ei pysty kiistämään viestin lähettämistä. [22]

Pelkän viestin salauksen lisäksi on nykyisin mahdollista lähettää viestejä, jotka muuttuvat lukukelvottomiksi tietyn ajan kuluttua. Tällaistenkin viestien käyttö perustuu salaukseen, mutta niissä viestin avaamiseen tarvittavan avain on voimassa ennalta määrätyn ajan, jonka jälkeen salausta ei voi enää purkaa. [16]

Nykyisin yleistyneet ns. webmail-palvelut tuovat helpotusta liikkuvan henkilön elämään. Yleensä yritysten työntekijöiden henkilökohtaisiin POP-postilaatikoihin saa yhteyden vain palomuurin sisällä olevilta koneilta. Webmail mahdollistaa yhteydenoton WWW-selaimen kautta suoraan käyttäjän omaan POP-postilaatikkoon mistä tahansa Internetiin liitetyltä koneelta. Koska webmailissa ei ole mitään verkon asettamia rajoituksia, on postilaatikon ainoa suoja käyttäjän oma salasana. Tämän takia hyvä salasana ja sen säännöllinen vaihtaminen on tärkeää.

Tunnettuun Microsoftin omistamaan Hotmail webmail-palveluun onnistuttiin tekemään tietomurto 30.7.1999. ”Ruotsalaiselle WWW-sivulle laitettiin yhdeksän rivin mittainen HTML-koodinpätkä, jolla kuka tahansa pystyi lukemaan Hotmailin 50 miljoonaa postilaatikkoa ilman salasanaa, pelkän käyttäjätunnuksen varassa” [12]. Tämä runsaasti julkisuutta saanut tietomurto oli murtautujien huomionosoitus Microsoftin heikkoa tietoturvaa vastaan.

3 Java

Tässä luvussa esitellään Java-kielen ja -ohjelmien yleistä rakennetta.

3.1 Yleistä Javasta

Nykyiset ohjelmistojen kehitysmenetelmät sallivat ohjelmien muodostamisen useista erillisistä komponenteista. Näin voidaan muodostaa suuriakin sovelluksia yksittäisistä komponenteista, joita ajetaan mahdollisesti eri prosessoreilla fyysisesti eri paikoissa sijaitsevissa tietokoneissa. Internet ja etenkin WWW tarjoaa nykyisin käyttökelpoisen ympäristön hajautetuille sovelluksille. Jopa maapallonlaajuisesti toimivien hajautettujen sovellusten teko on periaatteessa mahdollista, mutta näiden käyttökelpoisuutta rajoittaa Internetin turvallisuuden puute sekä verkossa olevien koneiden erilaiset käyttöjärjestelmät. Java tarjoaa ratkaisun molempiin ongelmiin: ohjelmat ovat sekä turvallisia että käyttöjärjestelmästä riippumattomia ja voivat sijaita hajautettuna eri puolella Internetiä.

Vuoden 1995 lopulla tapahtuneen julkistamisen jälkeen Java saavutti heti suuren suosion. Vaikka se on kuin mikä tahansa muukin ohjelmointikieli luokkakirjastoineen, on siinä monia hyviä ominaisuuksia verkkoympäristössä tapahtuvaan ohjelmointiin. Java on C++:n pohjalta kehitetty puhtaasti oliopohjainen kieli, jossa lähes kaikki esitettävät asiat ovat olioita, poikkeuksena pelkästään numeeriset-, merkki- ja totuusarvotietotyypit. [6]

Java on hajautettu kieli, joka sisältää hyvän tuen verkkoliikennöintiin. Valmiiksi toteutettujen luokkien avulla esimerkiksi etäkoneella olevien tiedostojen luku onnistuu melkein yhtä helposti kuin paikallisen tiedoston luku. Samoin Remote Method Invocation (RMI) API:n avulla voidaan Java-ohjelmasta kutsua etäolioiden metodeja aivan samoin kuin ne olisivat paikallisia olioita. Javan hajautettuun luonteeseen antaa käyttökelpoisen lisän mahdollisuus ladata luokkia dynaamisesti. Näin Java-tulkki pystyy lataamaan ja suorittamaan koodia Internetistä. Näin tapahtuu esimerkiksi silloin, kun selainohjelma lataa Java-sovelman ja ajaa sen.

3.2 Java-kielen rakenne

Javan erikoisuutena on, että kaikki ohjelmat sekä käännetään että tulkataan. Sen lähdekoodit käännetään ensin Java-kääntäjällä laitteistoriippumattomiksi tavukoodeiksi ja tämän jälkeen ne voidaan ajaa missä ympäristössä tahansa, johon Java-tulkki ja suoritusjärjestelmä on asennettu. Tähän ajoympäristöön kuuluu Java virtuaalikone (*engl. Java Virtual Machine, JVM*) ja Java API (*Application Programming Interface)*. Kielen kehittäjä Sun jakaa Javaa JDK (*Java Development Kit*) -kehitysympäristönä, joka sisältää mm. Java-kääntäjän, JVM:än, API:n, appletviewer-sovelluksen sovelmien ajoon sekä muita apuohjelmia.

Javalla toteutettavat ohjelmat ovat kahta eri tyyppiä: sovelluksia (*engl. application*), jotka ovat tavallisia ohjelmia, ja sovelmia (*engl. applets*), joita voidaan ajaa erityisellä appletviewer-sovelluksella tai WWW-selaimella. Sovelmilla on rajoitettu pääsy tietokoneen resursseihin, joten ne eivät voi esimerkiksi lukea tai kirjoittaa tiedostoja kuten muut Java-sovellukset. Käytännössä sovelmat ladataan verkon yli osana web-dokumenttia ja ajetaan asiakkaan selaimella.

Koska Java on oliopohjainen kieli, rakentuvat kaikki Java-ohjelmat luokista, olivat ne sitten varsinaisia ohjelmia tai sovelmia. Java ydin API koostuu runsaasta määrästä valmiita luokkia, joita käytetään ohjelmoitaessa hyväksi. Valmiit luokat jaotellaan niiden käyttötarkoituksien mukaan eri paketteihin (*engl. packages*), jotka sisältävät useita samantyyppisiin tehtäviin suunniteltuja luokkia. Esimerkiksi verkkoyhteyksiin liittyviä luokkia sijaitsee `java.net`-paketissa ja eräs kyseisen paketin luokka on yhteyksien muodostamiseen käytetty `Socket`. Myös ohjelmoijan on syytä sijoittaa tekemänsä luokat eri paketteihin niiden käyttötarkoituksien mukaan.

Tavallisten luokkien lisäksi API sisältää ns. abstrakteja luokkia, joissa on esitelty niiden sisältämät metodit, mutta varsinainen metodien toteutus puuttuu. Tällaisia luokkia voidaan periä, jolloin uusi luokka sisältää perityn yliluokan metodit sekä mahdollisia käyttäjän lisäämiä uusia metodeja. Yliluokasta perityille metodeilla voidaan uudessa luokassa kirjoittaa toiminnallisuus. Samoin API:ssa on lukuisia rajapintoja (*engl. interface*), jotka sisältävät metodeja ilman toiminnallisuutta. Ennen rajapinnan käyttöä sille täytyy tehdä rajapinnan toteuttava luokka, johon varsinainen metodien toiminnallisuus

tehdään. Koko Javan luokkahierarkian ylimmäisenä juuri-luokkana toimii `Object`-luokka, josta kaikki muut luokat peritty.

Varsinainen ohjelmointi tapahtuu koostamalla ohjelma luokista, jotka ovat erikoistuneet eri tehtäviin. Tässä käytetään hyväksi valmiiden luokkien palveluja sekä tehdään uusia omia luokkia tehtäviin, joihin valmiita luokkia ei ole saatavilla. Kun luokat dokumentoidaan hyvin ja niistä tehdään mahdollisimman yleiskäyttöisiä, voidaan niitä käyttää hyväksi myös muissa ohjelmissa. Niinpä ydin API:kin on jatkuvasti laajentunut lukuisilla luokilla uusien JDK-versioiden myötä.

Java-ohjelmat ajetaan JVM:ssä, jonne jokainen suoritettava luokka ladataan luokkalataajan avulla. Luokka voi sijaita joko paikallisella tai etäkoneella. Java-ohjelmia käännettäessä ja ajettaessa käytetään ns. CLASSPATH-muuttujaa, joka osoittaa Javan luokkakirjaston sijaintipaikan kovalevyllä. Näin Java-tulkki ja JVM löytävät valmiit luokat ja osaavat niitä käyttää.

Javan pavut (*engl. Java Beans)* ovat Javan tarjoama komponenttimalli. Se koostuu kahdesta pääelementistä, jotka ovat varsinainen komponentti ja säiliöluokka (*engl. container*). Komponentit voivat vaihdella pienistä visuaalisista komponenteista, kuten painonappi, suurempiin sovelluksiin, kuten tekstinkäsittelyohjelman HTML-selain. Komponenteilla voi olla visuaalinen ulkonäkö tai komponentti voi olla näkymätön, mutta silti se suorittaa tapahtumia ja pitää yllä komponentin tilaa. Säiliöluokka sisältää toisiinsa liittyvät komponentit tarjoten kontekstin komponenttien järjestäytymiselle ja kanssakäymiselle toistensa kanssa. Säiliöluokka voi olla esimerkiksi lomake, sivu tai kehys.

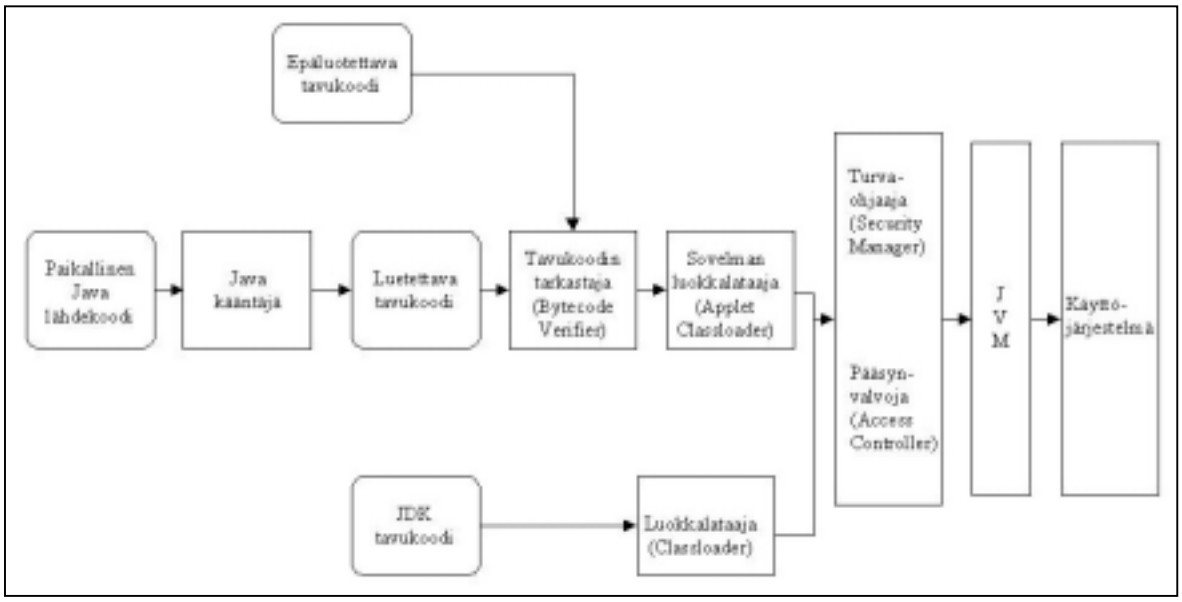
Säikeet muodostavat merkittävän osan Java-ohjelmien suorituksesta. Ne ovat ohjelman sisällä toimivia ”kevyitä prosesseja”, joilla on mahdollisuus kommunikoida ja kilpailla suoritusajasta muiden säikeiden kanssa. JVM pitää huolta säikeiden perustoiminnoista. Se aloittaa sovelluksen main-metodin suorituksen yhdessä säikeessä, jakaa säikeille suoritusaikaa prioriteettien mukaan ja lukitsee tarvittaessa tietyn resurssin yhden säikeen käyttöön. Kaikki sovellukset ja varsinkaan sovelmat eivät aina tarvitse useampia säikeitä. Niitä voi kuitenkin tarvittaessa käynnistää yleensä ainakin useita kymmeniä riippuen laitteesta ja käyttöjärjestelmästä sekä sen ja JVM:n versiosta. [6]

Seuraavassa luettelossa on listattu muutamia Javan piirteitä, jotka vähentävät tavallisia ohjelmoijien tekemiä virheitä ja jotka vaikuttavat merkittävästi turvallisempien ohjelmien kehitykseen.

- Vahva tyypitys: Luokka- ja tyyppimuunnokset sekä metodikutsujen oikeellisuus tarkistetaan Javassa aina. Tämä varmistaa, ettei käyttäjä yritä sijoittaa jotain tietotyyppiä vääräntyyppiseen referenssiin.
- Osoittimien puuttuminen: Osoittimien puuttuminen estää ohjelmoijaa osoittamasta vääriin muistialueisiin. Näin estetään erityisesti C++-kielessä yleinen ohjelmointivirhe, vääriin muistialueisiin viittaavat osoittimet. Osoittimien puute saattaa toisaalta estää joidenkin matalan tason ohjelmien kuten laiteajureiden toteuttamisen Javalla.
- Taulukoiden indeksien tarkistus: Java-kääntäjä tarkistaa automaattisesti, että taulukkoon sijoituksessa ei ylitetä taulukon kokoa. Näin estetään taulukolle varatun muistialueen ulkopuolelle vahingossa tapahtuvat sijoitukset.
- Automaattinen muistiroskien keruu: Javassa ohjelmoijan ei tarvitse huolehtia olioiden tuhoamisesta. Java tunnistaa automaattisesti oliot, joihin ei ole enää viittausta ja ne poistetaan. Tästä järjestelmästä käytetään nimitystä muistiroskien keruu (*engl. garbage collection*).

3.3 Javan turvamalli

Minkä takia puhutaan Javan tietoturvasta? Javahan on vain ohjelmointikieli, eikä aiemmin ole puhuttu vaikkapa C++:n tai Pascalin tietoturvasta. Javan turva-ajattelu poikkeaa suuresti perinteisestä lähestymistavasta. Tavallisimmat kotikäytössä olevat käyttöjärjestelmät sallivat ohjelmien pääsyn helposti myös järjestelmäresursseihin. Tämän takia ylläpitäjien täytyy varmistaa, että eri käyttäjien oikeudet on oikein asetettu ja että käyttäjät varmistavat kaikkien tuntemattomien ohjelmien turvallisuuden ennen niiden ajamista. Javassa vieraiden sovelluksien pääsyä resursseihin rajoitetaan sen ajonaikaisella turvajärjestelmällä, joka perustuu ns. hiekkalaatikkomalliin (*engl. The Java Sandbox*).



Kuva 3 : Javan turvamalli tietovirtakaaviona [33].

JVM suorittaa tuntemattoman Internetistä ladatun sovelman omassa hiekkalaatikossaan, josta sillä on rajoitettu pääsy matalan tason resursseihin. Ohjelma ei myöskään pääse vahingoittamaan toisia hiekkalaatikoita, vaikka se omassa laatikossaan saisikin tuhoa aikaan. Javan hiekkalaatikkomalli koostuu useasta eri komponentista ja niiden yhteistyöstä. Oheisesta tietovirtakaaviossa hiekkalaatikkoon kuuluu tavukoodin tarkastaja, luokkalataaja ja turvaohjaaja/pääsynvalvoja.

Ensimmäisenä osana hiekkalaatikkomallia on luokkalataaja (*engl. Classloader*). Luokkalataaja hakee sovelman ohjelmakoodin etäkoneelta ja valvoo, etteivät suoritettavat sovelmat korvaa järjestelmätason komponentteja tai muodosta omia luokkalataajiaan. Luokkalataaja siis määrittelee sen, milloin ja miten sovelmat voivat ladata uusia luokkia. [15]

Seuraavana komponenttina mallissa on tavukoodin tarkastaja (*engl. Bytecode Verifier*). Sillä tarkistetaan jokainen luokkalataajan lataama sovelma ennen sen suorittamista ja varmistetaan, että tavukoodi täyttää kaikki Javan säännöt. Seuraavassa on lueteltu joitakin tavukoodin tarkastajan tekemiä toimenpiteitä [14] :

- Luokan eheyden tarkastusvaihe 1 tiedoston latauksen aikana: Tavukoodin tarkastaja tarkastaa jokaisen luokkatiedoston eheyden, allekirjoituksen ja sen, että

luokkatiedoston rakenne on oikea. (Luokkien allekirjoituksesta on kerrottu luvussa 5.)

- Luokan eheyden tarkastusvaihe 2, kun tiedosto on ladattu: Tavukoodi-muodossa olevan luokan eheys todetaan pintapuolisella tavukoodin tarkastuksella, ilman koodin varsinaista suoritusta. Tällä varmistetaan, että: 1) luokalla on yliluokka, ellei kyseinen luokka ole `Object`-luokka, 2) yliluokka ei ole final-luokka, eli yliluokka voidaan periä, 3) luokka ei yritä kumota yliluokkansa final-tyyppisiä metodeja. Eheyden tarkastuksessa varmistetaan myös, että kaikilla metodeilla ja referensseillä on Javan sääntöjen mukaiset nimet.
- Tavukoodin eheyden tarkastus: Myös tavukoodin tuleva käyttäytyminen ajonaikana tutkitaan (ilman koodin varsinaista suoritusta). Tähän kuuluu mm. tietovirtojen analysointi, pinojen tarkistus sekä metodien argumenttien ja tavukoodin operandien tyyppien tarkastus. Näillä tarkistuksilla varmistetaan, ettei pinoissa tapahdu yli- tai alivuotoja, metodeja kutsutaan oikean tyyppisten parametrien kanssa, tavukoodissa on oikea määrä oikean tyyppisiä operandeja ja tietotyyppejä käytetään oikein.
- Ajonaikaisen eheyden tarkastus: Tämä vaihe käynnistetään tarvittaessa ajonaikana esimerkiksi silloin, kun jokin metodi palauttaa luokan, joka on peritty jostakin toisesta luokasta. Tällöin tarkastetaan, että luokka on todella peritty siitä luokasta josta sen väitetäänkin olevan peritty.

Näiden tarkistusten jälkeen luokkatiedoston ei pitäisi sisältää laittomia osoittimia, laitonta tavukoodia, laittomia tavukoodin parametreja, yli- tai alivuotavia pinoja, tai suorittaa laittomia muunnoksia. Kuitenkin tällaisella tiedoston analysoinnilla on mahdotonta tutkia täysin varmasti, kuinka koodi ajettaessa käyttäytyy. Erityisesti pinojen yli- ja alivuotojen täydellinen tarkastus osoittautuu mahdottomaksi ilman tavukoodin tulkkausta ja suoritusta.

Kolmantena ja tärkeimpänä osana Javan turvamallissa on turvaohjaajan (*engl. Security Manager*) ja pääsynvalvojan (*engl. Access Controller*) muodostama kokonaisuus. Niiden tehtävänä on suorittaa ajonaikainen tarkistus metodeille, jotka kutsuvat I/O-palveluita, yrittävät pääsyä verkkoon tai yrittävät luoda uusia luokkalataajia eli `ClassLoader`-

luokan olioita. Näin siis päätetään, voidaanko haluttu toimenpide suorittaa vai ei. Pääsynvalvojan noudattamaa politiikkaa pystytään muokkaamaan kätevästi erillisen tiedoston avulla, kun taas turvaohjaajan päätökset ovat itse ohjelmakoodissa. Seuraavassa esimerkissä esitetään, kuinka turvaohjaajaa käytetään `mkdir()`-systeemikäskyä kutsuttaessa. [27]

```
public boolean mkdir(String path)
   SecurityManager security = System.getSecurityManager();
   if (security != null) {
      security.checkWrite(path);
   }
   return mkdir0(path);
}
```

Kun julkista metodia `mkdir()` kutsutaan, turvaohjaaja tarkastaa, onko sen kutsuminen laillista. Jos suoritusta ei sallita, heitetään turvallisuuspoikkeus ja `mkdir()`-metodin suoritus loppuu. Jos taas turvaohjaajan `checkWrite()`-metodi palautuu normaalisti, on parametrina annettu hakemisto mahdollista muodostaa. Tällöin kutsutaan yksityistä metodia `mkdir0()`, joka suorittaa varsinaisen hakemiston luonnin. Esimerkiksi kaikki Sunin tekemät Java-luokkakirjastot käyttävät turvaohjaajaa päättämään vaarallisten metodien suorittamisesta.

JDK 1.2:sta lähtien turvamalliin tuli mukaan Javan suojattu alue-malli (*engl. Java Protection Domains*), joka laajentaa alkuperäistä hiekkalaatikkomallia. Tässä uudessa mallissa allekirjoitetulla Java-ohjelmalla on pääsy myös sellaisiin järjestelmäresursseihin, joihin sille on annettu lupa erillisessä policy-tiedostossa. Tämä laajennus toi mukanaan edellä kerrotun pääsynvalvojan turvallisuusohjaajan rinnalle. Seuraavassa luvussa käsitellään yksityiskohtaisemmin JDK 1.2-version turvamallin rakennetta. Oleellisimmista eroista vanhempiin versioihin verrattuna on myös kerrottu lyhyesti. Uusimman JDK 1.3:n turvamallin rakenne ei ole ratkaisevasti muuttunut, ainoastaan eri metodeissa on tapahtunut pieniä muutoksia. [18]

4 Javan turvamallin komponentit tarkemmin

Tässä luvussa käsitellään turvamallin komponentit yksityiskohtaisemmin ja samalla selvitetään niiden suhdetta tietoturvan eri osa-alueisiin.

4.1 Turvamallin tarkoitus

Koko tässä luvussa esiteltävän turvamallin tarkoitus tietoturvan kannalta on käyttäjän omalla koneella tapahtuva tarkkailu väärinkäyttöä vastaan. Hiekkalaatikossa satunnaiselta WWW-sivulta ladattu sovelma ei pääse käsiksi käyttäjän järjestelmän resursseihin ja varmistaa näin sovelman turvallisen ajon. Kun lisäksi otetaan suojattu alue -laajennus mukaan, voi käyttäjä tarkoin määritellä, mitä tuntemattomat ohjelmakoodit saavat tehdä. Tämän mallin toiminta vastaa tietoturvan määritelmässä kokonaisuudessaan pääsynvalvontavaatimusta. Lisäksi luokkalatauksen ja varsinkin suojattu alue -laajennuksen yhteydessä toteutuvat eheys- ja todennusvaatimukset.

Vaikka Javan luokkalatausmekanismi mahdollistaa luokkien siirron toteuttamisen turvallisesti, ei ole pois suljettu mahdollisuus, että luokkien latauspaikan ohjelmakoodit olisivat epäluotettavia. Latauspaikan käyttäjä saattaa olla tietämätön, että hänen luokkansa sisältävät myös mahdollisia vihamielisiä ohjelmakoodin pätkiä kuten viruksia. Tällöin ei auta, vaikka luokan lähettäjä on todennettu ja luokan muuttumattomuus siirron aikana on varmistettu. Jos käyttäjä nyt erehtyy sallimaan liian suuria valtuutuksia sovelmalle, voi sillä olla avoimet ovet tehdä vahingollisia toimenpiteitä käyttäjän järjestelmään.

4.2 Javan luokkalataajat

Luokkalataajat muodostavat Javan tavan lukea tavukoodi JVM:ään ja muuttaa se luokkamäärityksiksi. Luokkalataaja muodostaa tärkeän osan Javan turvamallista, koska vain se tietää, mistä tietty luokka on ladattu ja onko se allekirjoitettu vai ei. Niinpä turvallisia Java-sovelluksia tehtäessä täytyy ymmärtää luokkalataajan rooli ja kuinka tehdä tai käyttää turvallista luokkalataajaa.

Luokkalataaja toimii yhdessä turvaohjaajan ja pääsynvalvojan kanssa JVM:n määrätessä käytettävää turvapolitiikkaa. Se antaa turvaohjaajan selvittää luokasta tiedot,

joiden perusteella JVM päättää käytettävästä turvapolitiikasta. Luokkalataajan avulla JVM pystyy erottelemaan samannimiset luokat toisistaan. Tämä on mahdollista, koska kukin luokka ladataan omalla luokkalataajallaan ja viittaus luokkalataajiin säilytetään myöhempää käyttöä varten. Luokkalataajan rooli tietoturvan kannalta on turvata luokkien siirto eheänä JVM:ään.

4.2.1 Luokkalataajien tyyppejä

Sisäinen luokkalataaja käyttää käyttöjärjestelmän omia tiedostonavausmetodeja luokkatiedostojen avaukseen ja lukemiseen. Jos joku näistä luokista sisältää viittauksia muihin luokkiin, huolehtii sisäinen luokkalataaja myös niiden lataamisesta. Sisäistä luokkalataajaa ei pystytä ylikirjoittamaan.

Sovelman täytyy pystyä lataamaan luokkia HTTP:tä käyttäen verkosta. Sovelmien luokkalataajat käyttävät tyypillisesti URL-luokkalataajaa luokkatiedoston datan lukemiseen. Java API:ssa ei ole yhtä ainoaa standardia sovelmien luokkalataajasta, vaan jokainen Java-selain on vastuussa oman luokkalataajansa toteutuksesta. Tavallisesti useiden selaimien luokkalataaja perustuu Sunin appletviewerin luokkalataajaan.

JDK 1.1:stä lähtien Java API sisältää RMI-luokkalataajan, jota useat sovellukset voivat käyttää. Nimestään huolimatta sitä ei ole pakko käyttää RMI-sovelluksissa. Täsmällisesti ilmaistuna se ei edes ole luokkalataaja, koska sitä ei ole peritty `ClassLoader`-luokasta. `RMIClassLoader` on hyvin samanlainen kuin sovelmien luokkalataaja, koska se käyttää HTTP-protokollaa lataamaan haluttu luokkatiedosto etäkoneelta ja muodostaa luokan tämän tiedoston datan perusteella.

Java API:in on versiosta 1.2 kuulunut `java.security`-paketissa oleva luokkalataaja nimeltään `SecureClassLoader`. Tällä luokalla on suojattu rakentaja (*engl. constructor*), joten sen pääasiallisena tehtävänä on toimia perustana muita luokkalataajia muodostettaessa. Sen erityispiirteenä on, että jokainen luokka, jonka se lataa, liitetään suojattuun alueeseen. Pääsynvalvoja käyttää tätä tapaa luokkien suojaamiseen, joten jokainen pääsynvalvojan suojaama luokka täytyy olla ladattuna `SecureClassLoader`-luokasta perityllä luokkalataajalla.

API 1.2 sisältää myös monikäyttöisen `URLClassLoader`-luokan, jota voi käyttää luokkien lataamiseen eri URL:eista. Tämän luokan tyyppistä oliota luotaessa annetaan rakentajalle taulukko, joka sisältää ne URL:it, joista URL-luokkalataaja yrittää tarvittavan luokan ladata. Luokkalataaja etsii tarvittavaa luokkaa taulukossa olevien URL:ien sijaintijärjestyksessä. URL-luokkalataaja osaa ladata myös hakemistossa tai JAR-tiedostossa olevia luokkia. URL-luokkalataaja voi pyytää toista luokkalataajaa yrittämään luokan latausta. Jos toinen luokkalataaja ei pysty luokkaa lataamaan, lataa URL-luokkalataaja sen itse.

4.2.2 Luokkalataajien käyttö

Käytettävä luokkalataaja riippuu käyttäjän tarpeesta. URL-luokkalataajalla voi ladata sekä yksittäiset että JAR-tiedostoissa olevat luokat käyttäen tiedosto- ja HTTP-perustaisia URL:eja. URL-luokkalataajaa ei pysty käyttämään, kun luokkia ladataan muualta kuin HTTP:n kautta tai tiedostojärjestelmästä. Se ei ole myöskään paras vaihtoehto, jos luokkia ladataan eri isäntäkoneilta (*engl. host*) ja tiedossa on jo ennalta, missä mikäkin luokka sijaitsee. Java 1.1:stä käytettäessä yhdestä paikasta luokkia ladatessa on yksinkertaisinta käyttää RMI-luokkalataajaa.

Luokkalataajan käyttö on yksinkertaisimmillaan hyvin helppoa. Ensin luodaan luokkalataaja-olio ja käytetään sen `loadClass()`-metodia lataamaan haluttu luokka. Tämän jälkeen muodostetun luokan tyyppisen olion metodeja voidaan käyttää useilla eri tavoilla. Luokan staattiset metodit voidaan suorittaa JVM:än natiivimetodi-rajapintaa käyttäen. Tätä tekniikkaa JVM käyttää sovellusten `main()`-metodin suorittamiseen, kun tämä sovelluksen käynnistävän metodin sisältävä luokka on ladattu.

Määrättyä luokkaa edustava olio voidaan muodostaa `Class`-luokan `newInstance()`-metodilla, jos luokalla on käytettävissä parametriton rakentaja. Esimerkiksi appletviewer käyttää tätä tekniikkaa: se lataa sovelman aloitusluokan, muodostaa sovelman instanssin (jolloin kutsutaan sovelman parametritonta rakentajaa) ja kutsuu lopuksi sovelman `init()`-metodia. Tämä on käytetyin tapa metodien suorittamiseen, koska se on yksinkertaisin ja toimii kaikissa Javan versioissa.

JDK 1.1:stä alkaen rakennekuvaus (*engl. Reflection*) API:a on voitu käyttää kutsumaan luokan staattista metodia sekä muodostamaan olion instansseja ja suorittamaan sen metodeja. Tämä mahdollistaa luokan joustavan käytön, koska se sallii myös parametrien välittämisen olion rakentajalle.

4.2.3 Luokkalataajan toteutus

Osa luokkalataajan turvaominaisuuksista riippuu sen sisäisestä toteutuksesta. Luokkalataajan toteutuksessa on kaksi tapaa: uusi luokkalataaja peritään joko `ClassLoader`- tai `SecureClassLoader`-luokasta. Itse asiassa `SecureClassLoader` on peritty `ClassLoader`ista, joka on siis abstrakti kaikkien luokkalataajien kantaluokka.

Luokkalataajaa toteutettaessa täytyy useiden eri metodien avulla tehdä tarkistuksia, jotta luokan lataaminen tapahtuu turvallisesti. JDK 1.1:tä käytettäessä luokan latausmetodi on abstrakti, joten ohjelmoijan täytyy itse tehdä sen toteutus. Tyypillisesti luokkalataajan `loadClass()`-metodin toteutus on seuraavanlainen: Ensin tarkastetaan `findLoadedClass()`-metodilla, onko kyseinen luokka jo ladattu. Jos näin on, palautetaan viite kyseiseen luokkaan, muuten jatketaan tarkistamalla turvaohjaajan avulla, onko kyseinen luokka käytettävissä.

Seuraavaksi `findSystemClass()`-metodilla etsitään luokkaa käyttäjän määrittelemästä CLASSPATHista sisäistä luokkalataajaa apuna käyttäen. Jos luokka löytyy, palautetaan siihen viite. Muuten luokka täytyy määritellä muualta (verkosta) ladatusta tavukoodista. Tämä tapahtuu `defineClass()`-metodilla, joka ajaa datan tavukoodin tarkastajan läpi ja muodostaa `Class`-olion. Tämä metodi myös varmistaa, että luokan nimi on sama luokkatiedostossa kuin metodille annetussa parametrissa. Tämän jälkeen voidaan vielä `resolveClass()`-metodilla tarkastaa, tarvitaanko kyseisestä luokasta useampia viitteitä. JDK 1.2:sta käytettäessä on helpompaa käyttää valmista `loadClass()`-metodin toteutusta ja toteuttaa itse ainoastaan `findLocalClass()`-metodi, joka ottaa tavukoodia vastaan ja määrittelee siitä luokan.

Turvallisuutta ajatellen `loadClass()`-metodi on tärkeä, koska sen toteutuksesta riippuu, kuinka turvallisesti Java-sovellus toimii. Tässä suhteessa esimerkiksi järjestys, jossa metodi etsii luokkia, on tärkeä. Jos käyttäjä tarvitsee `java.lang.String`-luokkaa, on tärkeää, että luokka palautetaan Java API:sta eikä ladata jostakin muualta, jolloin sen toteutus voi olla turvaton.

JDK 1.2:sta alkaen API:ssa on ollut mukana `SecureClassLoader`-luokka, jota voi ja kannattaakin käyttää oman luokkalataajan yliluokkana. Siitä on hyötyä suojattu alue -mallia käytettäessä, jota tarkastellaan tarkemmin pääsynvalvojan käsittelyn yhteydessä. Tässä luokkalataajassa on kaksi uutta metodia. Toinen niistä on `defineClass()`, jolle voidaan antaa parametrina koodin lähde tai suojattu alue sekä taulukko allekirjoituksia. Toinen uusi metodi on `getCodeSource()`, joka muodostaa CodeSource-olion annetun URL:in ja allekirjoitustaulukon avulla. Tämän käyttö on suositeltavampaa suoran `CodeSource`-luokan instantioinnin sijasta, koska metodi säilyttää välimuistissa kaikki palauttamansa oliot.

Jotta turvapolitiikka toteutuisi, täytyy luokkalataajan tekijän itse lisätä tarkistus siitä, onko haluttuun luokkaan käyttöoikeus. Kun luokka ladataan paketista, tämä tarkistus on lisättävä `findLocalClass()`-metodiin. URL-luokkalataaja ei kuitenkaan kutsu tarkistusmetodia. Tämän takia URL-luokkalataajaa käyttävissä ohjelmissa oletusturvamalli ei tee luokan käyttöoikeuden tarkistusta. Jos sovelluksella on silloin pääsy johonkin pakettiin, pääsee se samalla käsiksi kaikkiin kyseisen paketin sisältämiin luokkiin. JDK 1.2:ssakaan `checkPackageAccess()` ei kutsu suoraan turvaohjaajan vastaavaa luokkaa, vaan tämä täytyy itse lisätä.

Luokkalataajat pystyvät lataamaan yhden luokan sijasta myös tiedoston, joka sisältää monta luokkaa. Näiden JAR-tiedostojen käyttäminen nopeuttaa luokkien lataamista, koska HTTP-yhteyden muodostaminen kestää suhteellisen kauan verrattuna luokan siirtämiseen. JAR-tiedostojen käytön haittana on, että tiedosto on aina ladattava kokonaan, vaikka se sisältäisikin luokkia, joille ei ole sovelluksen jokaisella käyttökerralla käyttöä. JAR-tiedostojen käyttö on kuitenkin yleistä ja niitä käytetään ohjelmissa, joissa kaikkia JAR:in sisältämiä luokkia käytetään. Turvallisuusnäkökohdasta JAR-tiedostot ovat myös tärkeitä,

koska ne mahdollistavat luokkien digitaaliset allekirjoitukset ja näin luokkien todennuksen. [18]

4.3 Turvaohjaaja

Turvaohjaaja on Javan hiekkalaatikossa keskeisessä osassa päättämässä siitä, mitkä toiminnat ovat sallittuja ja mitkä kiellettyjä. Jos ohjelma esimerkiksi haluaa avata tiedoston, päättää turvaohjaaja, voiko avauksen suorittaa. `SecurityManager`-luokka muodostaa Java API:ssa rajapinnan, jonka kautta muut luokat tarkistavat metodien suoritusoikeudet. Tämä tapahtuu pääasiassa seuraavan toimintoketjun kautta:

1) Ohjelmoija tekee Java API:lle pyynnön suorittaa toiminto.
2) Java API kysyy turvaohjaajalta, voiko toiminnon suorittaa.
3) Jos toimintoa ei sallita, turvaohjaaja heittää poikkeuksen Java API:lle, joka välittää sen edelleen käyttäjälle.
4) Muuten Java API suorittaa toiminnon normaalisti.

Laitonta metodia yritettäessä heitettävä `SecurityException`-poikkeus on `RuntimeException`-luokan aliluokka. Ajonaikaisia poikkeuksia ei Javassa tarvitse ottaa erikseen kiinni, eikä kyseisen poikkeuksen heittävässä metodissa tarvitse esitellä poikkeusta.

Luokkia sanotaan luotetuiksi tai epäluotetuiksi niiden alkuperän mukaan. Yleensä luotetuille luokille sallitaan turvaohjaajan toteutuksella enemmän toimintoja kuin epäluotetuille. Eri Java-versioiden välillä on merkittäviä eroja perusteissa, joilla luokkaa pidetään luotettuna. Java 1.0:ssa luokka on luotettu jos se on ladattu CLASSPATH:ista, kun taas luokkalataajan avulla muualta ladattu luokka on epäluotettu. Java 1.1:ssä pätevät muuten samat säännöt, paitsi JAR-tiedostosta ladatulle luokalle voidaan tiedoston sisältämän digitaalisen allekirjoituksen perusteella myöntää lisäoikeuksia.

Java 1.2:ssa ydin API:sta ladattu luokka on luotettu ja sillä on siis täydet oikeudet kaikkiin operaatioihin. Muuten luokkien oikeudet määrätään sen perusteella, mistä ne on ladattu. Nämä säännöt pätevät oletusturvaohjaajaan. Muuten ohjelmoija voi tehdä itse hyvinkin hienojakoista turvapolitiikka noudattavan turvaohjaajan. Kaikista eri

mahdollisuuksista huolimatta varsinkin selainsovellukset käyttävät yleensä jo Java 1.0:ssa olevaa oletusturvapolitiikka, eli CLASSPATHista ladatut luokat ovat turvallisia ja muut eivät.

4.3.1 Turvaohjaajan toiminta

`System`-luokka (ympäristöriippumaton rajapinta järjestelmäfunktioihin) sisältää kaksi turvaohjaajan yhteydessä käytettävää metodia. `GetSecurityManager()` palauttaa viitteen aiemmin asetettuun turvaohjaajaan. `SetSecurityManager()`-metodi asettaa järjestelmän turvaohjaajan parametrina välitettävään olioon. Tätä metodia voidaan kutsua vain kerran, koska jokaisessa JVM:ssä käytetään ainoastaan yhtä turvaohjaajaa. Kun se on asetettu, kaikki kyseisessä JVM:ssä ajettavat luokat käyttävät siis sitä. Sitä ei myöskään voi poistaa, vaan se on voimassa koko JVM:n olemassaolon ajan.

Tämän takia sovellusta toteutettaessa täytyy turvaohjaaja suunnitella huolellisesti, jotta se sisältää kaikki tarvittavat turvapolitiikat. Javan käytön mahdollistavissa selaimissa turvaohjaaja asetetaan selaimen toimesta, joten sovelma ei pysty sitä itse asettamaan. Näin on tietenkin oltavakin, koska turvaohjaaja estää sovelmaa tekemästä laittomia operaatioita. Tosin eri valmistajien selaimien käyttämät turvapolitiikat sovelmissa saattavat vaihdella jonkin verran.

4.3.2 Turvaohjaajan tarjoamia palveluja

Käytetyimpiä turvaohjaajan tarjoamia palveluja ovat tiedostojen käytön valvontaan käytetyt metodit. Näihin kuuluvat:

- `checkRead()`, joka tarkastaa voiko ohjelma lukea parametrina annettua tiedostoa.
- `checkWrite()`, joka tarkastaa, onko parametrina annettuun tiedostoon kirjoittaminen mahdollista.
- `checkDelete()`, joka varmistaa voiko tiedoston tuhota.

Verkkoyhteyden muodostaminen perustuu Javassa aina verkkoistukan käyttöön joko suoraan `Socket`-luokalla tai epäsuorasti toisen, kuten `URL`-luokan läpi. Epäluotettavat

luokat pystyvät oletuksena avaamaan istukan vain koneella, josta ne on ladattu. Tällä rajoituksella estetään sovelman yhteydenotto kolmannelle koneelle verkon yli. Näin sovelma ei pysty esimerkiksi ottamaan yhteyttä muiden koneiden sähköpostipalvelimille ja lähettämään sähköpostia sovelman käyttäjän nimissä. Toinen rajoituksen tarkoitus on estää sovelmaa ottamasta yhteyttä käyttäjän lähiverkon koneisiin ja lähettämään verkon palveluista tietoa HTTP-protokollan kautta edelleen muualle.

Verkkoistukoita on kahdenlaisia, asiakkaan (*engl. client*) ja palvelimen *(engl. server)* päässä olevia. Asiakasistukan tehtävä on muodostaa yhteys palvelinistukkaan. Palvelinistukka taas odottaa asiakkaan kutsuja ja on valmiina reagoimaan niihin. Turvaohjaajan käyttämät metodit verkkoon pääsyn tarkkailuun ja niiden tehtävät ovat:

- `checkConnect()` metodia käytetään tarkastamaan, voiko ohjelma avata asiakasistukan parametreina annettuun porttiin ja isäntäkoneeseen.
- `checkListen()` tarkastaa, voiko ohjelma luoda palvelinistukan, joka kuuntelee parametrina annettua porttia.
- `checkAccept()` tarkastaa, voiko ohjelma hyväksyä parametreina annetusta isäntäkoneesta ja portista lähtöisin olevan yhteyden.
- `checkMulticast()` tarkastaa, voiko ohjelma muodostaa multicast istukka -yhteyden parametrina annettuun osoitteeseen.
- `checkSetFactory()` varmistaa, voiko ohjelma muuttaa istukan oletustoteutusta.

Java 1.0:ssa epäluotettujen luokkien ei sallita muodostavan palvelin-istukoita, jolloin `checkListen()`- ja `checkAccept()`-metodit heittävät aina turvapoikkeuksen sovelman yrittäessä kyseistä toimenpidettä. 1.1:stä lähtien epäluotetut luokat voivat muodostaa palvelinistukan, jos sen porttinumero on suurempi kuin koneen etuoikeutetun portin numero (yleensä 1024).

Turvaohjaaja sisältää useita metodeja, jotka suojaavat JVM:n koskemattomuutta. Nämä metodit aitaavat epäluotetut luokat niin, etteivät ne voi kiertää turvaohjaajan ja Java API:n suojauksia.

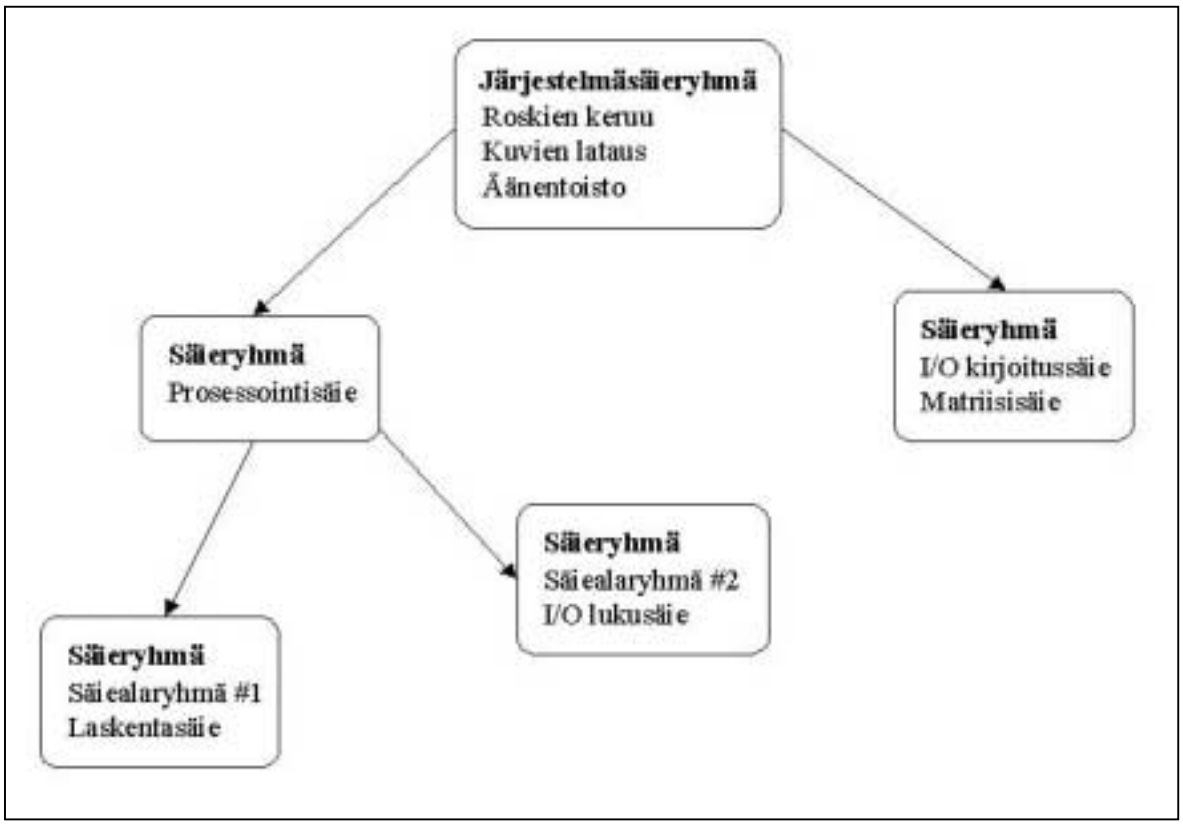
- `checkCreateClassLoader()` tarkastaa, voiko ohjelma muodostaa uuden luokkalataajan. Tätä metodia kutsuu ainoastaan luokkalataajan rakentaja.
- `checkExec()` varmistaa, saako sovellus suorittaa järjestelmäkomentoja. Tällä estetään epäluotettavia luokkia käynnistämästä mahdollisia haitallisia prosesseja.
- `checkLink()` estää epäluotetun luokan linkittämästä jaettua kirjastoa JVM:ään. Tämä täytyy tehdä, jos aiotaan suorittaa natiivia C-koodia JVM:ssä.
- `checkExit()` estää epäluotettavaa koodia pysäyttämästä JVM:ää.
- `checkPermission()` tarkastaa, onko nykyiselle säikeelle myönnetty parametrina tuotua oikeutta.

Java pohjautuu hyvin voimakkaasti säikeiden käyttöön suorituksen aikana. Yksinkertainen Java-ohjelma, jossa on kuvia ja ääntä, luo useita säikeitä automaattisesti. Näitä järjestelmätason säikeitä käytetään mm. roskien keruuseen, graafisen käyttöliittymän toimintaan, kuvien hakuun jne. Epäluotettu luokka ei voi näihin säikeisiin koskea, koska se saattaisi vahingoittaa JVM:ää tai muita sovelmia. Turvaohjaaja sisältää seuraavat metodit säikeiden suojaamiseen:

- `checkAccess()` tarkistaa, voiko ohjelma muuttaa parametrina annetun säikeen tai säieryhmän tilaa. Sovelmat voivat koskea vain säikeisiin, jotka ne ovat itse luoneet.
- `getThreadGroup()` välittää oletussäieryhmän, johon vasta luotu säie voidaan liittää.

Näiden metodien toiminta on hyvin suoraviivaista. Epäluotettu luokka voi käsitellä vain luomiaan ja omaan säieryhmäänsä kuuluvia säikeitä. `GetThreadGroup()` on erilaisessa asemassa verrattuna muihin turvaohjaajan metodeihin. Se ei nimittäin päätä, myönnetäänkö luokalle johonkin resurssiin pääsy, vaan sen tarkoituksena on päättää oletussäieryhmä, johon tietty säie liitetään. Metodia käytetään, jos uuden säikeen luonnin yhteydessä ei ole määritelty mihin ryhmään se kuuluu. Oletuksena se on kutsuvan säikeen säieryhmä, mutta turvaohjaaja voi käyttää valinnassa myös muuta logiikkaa, jotta kuvassa 4 oleva hierarkia toteutuisi.

Java-sovellusten säikeet järjestellään kuvassa 4 olevalla tavalla. Teoriassa turvaohjaajan politiikan pitäisi käyttää tätä hierarkiaa niin, että säikeet voivat käsitellä vain alapuolellaan olevia säikeitä. Tätä mallia käytetään Java 1.2:sta lähtien. Versiossa 1.1



Kuva 4: Javan säiehierarkia.

jokaiselle sovelmalle annetaan yksilöllinen säieryhmä ja samassa ryhmässä olevat säikeet voivat käsitellä toisiaan. 1.0:ssa kaikki säikeet voivat käsitellä toisiaan ja niinpä epäluotettavilla luokilla ei ole oikeutta omien säikeiden luomiseen.

Selaimilla on pääsy tiettyihin järjestelmätason resursseihin, joita ei tulisi olla epäluotettavien luokkien ulottuvilla. Seuraavaksi on esitelty metodeja, joilla turvaohjaaja käsittelee kyseisiä resursseja.

- `checkPrintJobAccess()` estää epäluotetun luokan pääsyn tulostamaan.

- `checkSystemClipboardAccess()` estää epäluotetun luokan lukemasta järjestelmän leikepöytää. (Luokan itsensä muodostamaa leikepöytää se voi lukea.)
- `checkAwtEventQueueAccess()` estää epäluotetun luokan pääsyn JVM:n järjestelmätapahtumajonoon.
- `checkPropertyAccess()` estää epäluotetun luokan pääsyn JVM:n sisältämiin globaaleihin järjestelmän ominaisuuksiin (*engl. properties*) kuten nimi ja kotihakemisto. Yleensä turvaohjaaja kirjoitetaan niin, että sovelmilla on pääsy tiettyihin ennalta nimettyihin ominaisuuksiin.
- `checkTopLevelWindow()` tarkastaa, voiko epäluotettu luokka luoda ikkunan. Paluuarvona on tosi, jos ikkuna voidaan luoda normaalisti. Epätosi palautetaan, jos ikkuna voidaan luoda, mutta siihen täytyy lisätä tunnisteotsikko, josta käyttäjä näkee, että kyseessä on sovelman luoma ikkuna. Metodi voi myös heittää turvapoikkeuksen, jos ikkunan aukaisu on kielletty.

JVM sisältää myös muutamia metodeja, jotka suojelevat Java-kielen rakenteellisen turvallisuuden ajatusta. Tällaisia ovat:

- `checkMemberAccess()` sallii epäluotetun luokan pääsyn muissa luokissa ainoastaan julkisiin tietoihin.
- `checkPackageAccess()` tarkastaa, voiko epäluotettu luokka päästä käsittelemään määrätyssä paketissa olevia luokkia.
- `checkPackageDefinition()` tarkastaa, voiko epäluotettu luokka ladata tietyssä paketissa olevia luokkia.
- `checkSecurityAccess()` estää epäluotettua luokkaa käsittelemästä `java.security` -paketissa olevia metodeja. [20]

4.4 Pääsynvalvoja

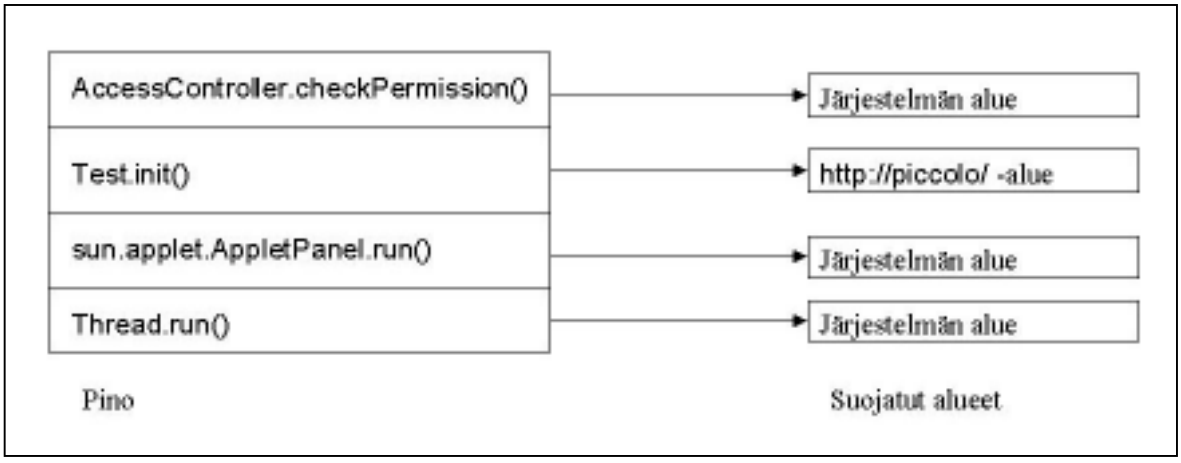
Pääsynvalvoja on uusimmissa JDK:n versioissa käytetty turvamallin osa, joka on tullut turvaohjaajan rinnalle päättämään turvatarkistuksista. Pääsynvalvoja on ollut mukana Java API 1.2:sta lähtien. Aikaisemmissa versioissa turvaohjaaja päätti itse käytettävän turvapolitiikan ja tällöin politiikan vaihto vaati koko turvaohjaajan vaihdon.

JDK 1.2:sta lähtien päätökset siirtyivät osin pääsynvalvojalle. Se päättää turvapolitiikan tätä varten määritellyn tiedoston avulla. Tämä mahdollistaa hyvin joustavien turvamenettelyjen toteutuksen joihin myös käyttäjä voi vaikuttaa, koska policy-tiedosto on vapaasti muokattavissa eikä ohjelmakoodiin tarvitse koskea. Javan ydin API ei koskaan kutsu pääsynvalvojaa, jollei turvaohjaaja ole asetettu kyseiseen JVM:ään. Pääsynvalvoja koostuu seuraavilla sivuilla esitellyistä neljästä osasta.

Pääsynvalvojan rooli tietoturvan osa-alueissa kuuluu nimensä mukaisesti pääsynvalvontaan. Sen varsinainen toiminta toteutetaan `AccessController`-luokan metodeilla ja seuraavilla sivuilla esitettyjen luokkien avulla. Luokasta ei voi luoda olioita vaan käytettävissä ovat luokan sisältämät staattiset metodit, joita ovat seuraavat:

- `checkPermission()` tarkastaa onko parametrina annettu oikeus myönnetty ohjelmalle, jossa pääsynvalvoja on käytössä. Jos pääsyä ei myönnetä heitetään `AccessControlException`, muuten metodi palautuu normaalisti.
- `beginPrivileged()` aloittaa ohjelmalohkon, jonka sisällä olevalla koodilla on samat oikeudet kuin tätä metodia kutsuvallakin luokalla.
- `endPrivileged()` lopettaa edellisellä `beginPrivileged()`-metodilla aloitetun ohjelmalohkon.

Pääsynvalvojan päätös toiminnon suorituksen myöntämisestä riippuu suojattujen alueiden muodostamasta pinosta ennen tarkistuksen kutsua. Kuvitellaan esimerkiksi käytössä olevan sovelma, joka yrittää muodostaa verkkoyhteyden. Olkoon tämä sovelma ladattu osoitteesta `http://piccolo`. Silloin appletviewer-katseluohjelmaa käytettäessä sovelma käynnistyy normaalisti erillisessä säikeessä ja suojatuista alueista sekä metodikutsuista muodostuu kuvassa 5 esitetyn kaltainen pino.



Kuva 5: Suojattu alue -pino verkkoyhteyttä muodostavan sovelman yhteydessä. [20]

Ensin siis kutsutaan säikeen käynnistävää `run()`-metodia. Tämän jälkeen säie kutsuu applettPanelin `run()`-metodia, joka mm. luo HTTP-pohjaisen luokkalataajan, jota käytetään varsinaisen sovelman lataamiseen, ja muodostaa itse sovelman. Tämän jälkeen appletPanel kutsuu sovelman `init()`-metodia ja sovelma edelleen `checkPermission()`-metodia.

Seuraavaksi `checkPermission` tarkastaa kaikki pinossa olevat alueet ja niille myönnetyt oikeudet alkaen pinon päältä. Sovelman aluetta lukuunottamatta muut pinon luokat kuuluvat järjestelmän omaan alueeseen, joilla ei ole oikeutta minkään järjestelmätoiminnon suorittamiseen. Jos policy-tiedostossa on määritelty oikeus muodostaa verkkoyhteys `http://piccolo`-osoitteesta ladatuille sovelmille, huomaa pääsynvalvoja sen sovelman aluetta tarkastaessaan ja sallii näin yhteydenoton kyseiseen osoitteeseen.

Pääsynvalvoja yhdessä turvaohjaajan kanssa muodostaa voimakkaimman tietoturvapiirteen Java-ympäristössä. Se suojelee järjestelmän tärkeimpiä osia mahdollisilta vihamielisiltä sovelmilta ja antaa kuitenkin käyttäjälle mahdollisuuden sallia luotetuista paikoista ladatuille sovelmille pääsyn järjestelmän resursseihin. Seuraavissa alaluvuissa on esitelty lyhyesti pääsynvalvojan käyttämiä luokkia.

4.4.1 Koodin lähde

Koodin lähde eli `CodeSource`-luokan olio esittää URL:in, josta luokka on ladattu, sekä mahdollisen digitaalisen allekirjoituksen avaimet. Näitä olioita luo ja käyttää `SecureClassLoader`-luokka. `CodeSource`-luokka sisältää seuraavia metodeja:

- `CodeSource()` on koodin lähde -olion rakentaja. Se muodostaa kyseisen olion koodista, joka on ladattu parametrina annetusta URL:ista. Toisena parametrina on taulukko, joka sisältää mahdollisesti salakirjoitetun koodin julkiset avaimet.
- `equals()` vertaa kahta koodin lähde -oliota, jotka ovat samat jos ne on ladattu samasta URL:ista.
- `getLocation()` palauttaa URL:in, joka on annettu kyseisen olion rakentajalle.

- `getKeys()` palauttaa kopion kyseisen olion rakentajalle annetusta avaintaulukosta.

4.4.2 Oikeudet

Pääsynvalvojan käyttämä peruskokonaisuus on abstraktin `Permission`-luokan olio. Kun tämä oikeus-olio liitetään johonkin luokkaan, esittää se kyseiselle luokalle myönnetyt oikeudet. Oikeus-olioilla on kolme ominaisuutta, jotka ovat tyyppi, nimi ja toiminnat.

Oikeuksien tyyppi ilmaisee mihin kyseinen oikeus liittyy. Esimerkiksi tiedoston käsittelyyn liittyvä oikeus on tyyppiä `FilePermission`. Nimi taas yksilöi olion, johon oikeus liittyy. `FilePermission`illa on siis tiedoston nimi, johon se tarjoaa sisäänpääsyn. Nimet sisältävät usein korvausmerkkejä (*engl. wildcards*), jolloin yksi tiedosto-oikeus voi sisältää pääsyn useampaan tiedostoon.

Oikeus-olio voi sisältää yhden tai useampia toimintoja. Niiden mukanaolo riippuu oikeus-olion tyypistä. Tiedoston käsittelyn tapauksessa toiminnot voisivat olla luku, kirjoitus ja poisto. Oikeudet toimivat kahdessa eri roolissa. Ensinnäkin ne sallivat Java API:n pääsyn useisiin resursseihin. Toisaalta ohjelmoija voi luoda omia oikeus-olioitaan käytettäväksi omissa ohjelmissa ja määritellä täysin niiden nimet ja toiminnot. Java API sisältää seuraavat perusoikeudet, jotka toteutetaan seuraavissa luokissa:

- `java.io.FilePermission` edustaa tiedostojen oikeuksia.
- `java.net.SocketPermissioin` edustaa verkkoistukoiden kanssa käytävän kommunikoinnin oikeuksia.
- `java.util.PropertyPermission` edustaa Javan ominaisuuksiin pääsyn oikeuksia.
- `java.lang.RuntimePermission` sisältää Javan ajonaikaisiin toimintoihin pääsyn oikeudet.
- `java.awt.AWTPermission` sisältää ikkunointiin liittyviin resursseihin pääsyn oikeudet.
- `java.net.NetPermission` edustaa kolmen eri luokan käytössä tarvittavia oikeuksia. Ensimmäinen käyttötarkoitus on `Authenticator`-luokan kanssa,

joka tarjoaa HTTP-autentikoinnin salasanasuojatuilla web-sivuilla. Toisaalta se tarjoaa mahdollisuuden muodostaa multicast-tyyppisiä istukoita (tällä tarkoitetaan istukoita, joiden avulla voidaan lähettää dataa usealle vastaanottajalle yhtäaikaa) ja kolmantena käyttötarkoituksena on `URLClassLoader`in kuuntelijan asetus.

- `java.security.SecurityPermission` määrittää oikeudet security-paketin käyttöön.
- `java.io.SerializablePermission` edustaa olion serialisointiin ja takaisin olioksi muunnokseen liittyviä oikeuksia.
- `java.lang.reflect.ReflectPermission` mahdollistaa rakennekuvaus API:n kanssa käytettävien olioiden `accessible`-lipun asettamisen.
- `java.security.UnresolvedPermission` määrittää Java API:n sisäisesti käyttämät ulkoiset oikeudet ennenkuin oikeuksien määrittelemä luokka löytyy.
- `java.security.AllPermission` edustaa oikeuksia, joita voi käyttää missä tahansa toiminnoissa, joilla voi olla myös oma oikeusluokkansa.

Kaikki edellä esitetyt oikeusluokat perustuvat siis `Permission`-luokkaan. Tätä luokkaa käytetään yliluokkana, kun kirjoitetaan omia oikeusluokkia. Tämän luokan metodeja ei kuitenkaan yleensä käytetä suoraan omasta ohjelmakoodista, vaan niitä pääsynvalvoja käyttää. Näitä metodeja ovat seuraavat:

- `Permission()` `Permission`-luokan rakentaja.
- `equals()` vertaa, onko kaksi oikeus-oliota samoja.
- `hashCode()` palauttaa olion hajautuskoodin. (Jokaisen `Permission`-luokan aliluokalla täytyy olla oma hajautuskoodinsa.)
- `getName()` palauttaa oikeus-olion nimen.
- `getActions()` palauttaa oikeus-olion toiminnot.
- `toString()` palauttaa luokan ja oikeuden nimen sekä toiminnot muodossa (”Oikeusluokka”, ”Oikeuden nimi”, ”Toiminnot”).

- `implies()` päättää, onko jokin muu oikeus voimassa, jos tietty oikeus on jo käytössä. Metodi suorittaa myös korvausmerkkien huomioonoton, jolloin esimerkiksi tiedosto-oikeus `/myclasses/` sisältää oikeudet `/myclasses/abc/OwnApplett.class`-tiedostoon.
- `newPermissionCollection()` palauttaa oikeusjoukon.
- `checkGuard()` kutsuu turvaohjaajaa tarkastamaan, onko oikeus-olio sallittu vai ei.

Omaa oikeusluokkaa tehdessä `BasicPermission`-luokka on hyödyllinen apu. Se toteuttaa perustan oikeuksille, joilla ei ole mitään toimintoja. Näitä perusoikeuksia voidaan ajatella kaksiarvoisena, ne joko ovat tai eivät ole. Tästä huolimatta `BasicPermission`-luokasta perityille luokille voidaan tehdä vapaasti eri toimintoja. Tämän luokan pääasiallinen hyöty on sen tavassa käsitellä korvausmerkkejä oikeuksien nimissä. Nimet muodostetaan pilkulla erotettuna hierarkisesti. Niinpä esimerkiksi yritys abc voisi muodostaa oikeuksia seuraavasti: abc.readDatabase, abc.WriteDatabase, abc.runPayroll, abc.salesDepartent.accessCheck, jne. Nyt näitä oikeuksia voidaan käyttää joko koko nimillään tai korvausmerkin kanssa abc.*, joka täsmäisi kaikkiin edellisiin oikeuksiin.

`PermissionCollection`-luokka tarjoaa mahdollisuuden koota useampia oikeuksia samaan kokonaisuuteen, jolloin niitä voidaan käsitellä yhtenä yksikkönä. Tätä luokkaa käytettäessä oikeuksien tulisi olla samaan `Permission`-luokkaan kuuluvia, eli esimerkiksi `FilePermission`-luokan olioilla pitäisi olla oma kokoelmansa ja `SocketPermission`-luokan instansseilla oma. Jos halutaan koota myös erityyppisiä oikeuksia samaan kokonaisuuteen, käytettävissä on `Permissions`-luokka.

4.4.3 Politiikka

Pääsynvalvoja käyttää edellä kuvattujen lisäksi `Policy`-luokkaa, jonka perusteella määritellään käytettävä turvapolitiikka. `Policy`-luokka muodostaa kartoituksen koodin lähde- ja oikeus-olioiden kanssa niin, että määrätyistä paikoista ladatuille luokille saadaan halutut oikeudet. `Policy`-luokan sisältämiä metodeja ovat seuraavat:

- `Policy()` on politiikka-olion rakentaja. Se alustaa olion lukemalla java.policy-tiedoston, jos sellainen löytyy.
- `getPolicy()` palauttaa olemassa olevan politiikka-olion.
- `setPolicy()` asettaa parametrina annetun politiikka-olion käyttöön. Jos aiemmin asetettu politiikka-olio on olemassa, korvautuu se uudella.
- `evaluate()` luo oikeus-olion, jonka sisältämät oikeudet myönnetään luokille, jotka ovat parametrina annetussa koodin lähde -oliossa.
- `refresh()` päivittää politiikka-olion. Jos olio on alustettu tiedostolla, luetaan tämä tiedosto uudelleen ja asetetaan uusi politiikka-olio käyttöön.

Oletus politiikka-olio muodostetaan `PolicyFile`-luokan avulla. Se lukee tiedoston ja muodostaa sen pohjalta oikeudet. `Policy`- ja `PolicyFile`-luokat antavat järjestelmien käyttäjille mahdollisuuden muuttaa ohjelman turvapolitiikkaa tiedoston avulla ilman, että varsinaista ohjelman lähdekoodia tarvitsisi muokata.

4.4.4 Suojattu alue

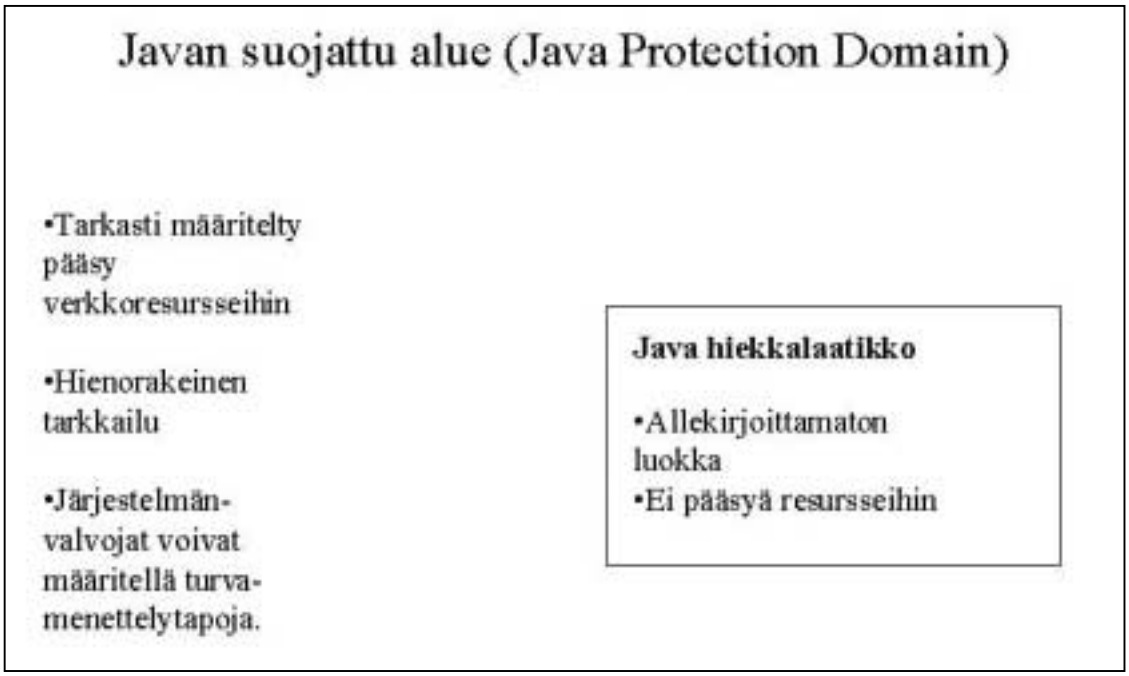
Suojattu alue on ryhmittely koodin lähde- ja oikeustyyppisiä olioita. Se esittää kaikki oikeudet, jotka on myönnetty tietyille koodin lähde -olioille. Tämä malli laajentaa Javan alkuperäistä hiekkalaatikkomallia JDK 1.2 versiosta lähtien ja sitä on havainnollistettu kuvassa 6. Siinä allekirjoitetuilla Java-ohjelmilla on oikeus käyttää resursseja, jotka on määritelty edellä esitetyssä politiikka-tiedostossa. Allekirjoittamattomat sovelmat sen sijaan ajetaan alkuperäisen hiekkalaatikkomallin mukaisesti.

`ProtectionDomain`-luokan sisältämiä metodeja ovat:

- `ProtectionDomain()` rakentaja, joka muodostaa parametreina annettuun koodin lähteeseen ja oikeuksiin perustuvan suojatun alueen.
- `getCodeSource()` palauttaa kyseisessä suojatussa alueessa käytössä olevan koodin lähteen.
- `getPermissions()` palauttaa kyseissä suojatussa alueessa käytössä olevat oikeudet.

- `implies()` tarkastaa, sisältyykö annettu oikeus tämän suojatun alueen sisältämään oikeus-olioon.

Kun tietty luokka kuuluu suojattuun alueeseen, tarkoittaa tämä sitä, että kyseinen luokka on ladattu koodin lähteessä määrätyltä sivulta, se on allekirjoitettu luokan lähteessä olevilla julkisilla avaimilla ja sillä on oikeudet resursseihin, jotka on esitetty oikeus-



Kuva 6: Javan suojattu alue [31].

oliossa. Jokainen JVM:ssä oleva luokka voi kuulua vain yhteen suojattuun alueeseen, jonka luokkalataaja määrittelee luokkaa ladatessaan. Kaikki luokkalataajat eivät tätä kuitenkaan tee. Luokat, jotka sijaitsevat järjestelmän CLASSPATH:issa, eivät omaa selkeää suojattua aluetta. Niiden voidaankin ajatella kuuluvan järjestelmän suojattuun alueeseen.

Suojattu alue tarjoaa useita parannuksia pelkkään hiekkalaatikkomalliin verrattuna. Yksi merkittävämmistä tällaisista on käyttäjän mahdollisuus muokata käytettävää turvapolitiikkaa suoraan ASCII-muotoisesta politiikka-tiedostosta käsin, ilman Java-koodin muokkaamista ja uudelleen kääntöä. Samoin itse halutun turvapolitiikan muodostaminen on huomattavasti helpompaa kyseisen tiedoston avulla kuin

ohjelmakoodista käsin. Hiekkalaatikkomallissa tämä vaati omien luokkien ohjelmointia `SecurityManager`- ja `ClassLoader`-luokista. [20, 31]

5 Kryptografia ja Java

Kryptografian tarkoituksena on muuttaa lähetetty data sellaiseen muotoon, etteivät sivulliset pysty sitä tulkitsemaan. Tämän toteuttamiseen on olemassa monia eri algoritmeja ja menetelmiä, kuten luvussa kaksi on esitelty. Näiden menetelmien mukanaolo on tärkeää hajautettuja sovelluksia avoimeen tietoverkkoon toteutettaessa, jotta sovelluksen todennus- ja eheysvaatimukset toteutuisivat.

Itse ohjelmointikieleenhän ei ole mitenkään järkevää toteuttaa olemassa olevia kryptografisia menetelmiä, vaan tarvittaessa kannattaa hyödyntää valmiita tehokkaiksi todettuja toteutuksia. Niinpä Javaan on toteutettu laaja joukko rajapintoja, joiden avulla päästään helposti käyttämään monia kryptografian menetelmiä Java-sovelluksia toteutettaessa. Tässä luvussa on esitelty Javan tarjoamia rajapintoja kryptografian menetelmien käyttöön osana sovellusta.

5.1 Tiedon tiivistys

Tiedon tiivistys (*engl. message digest*) on usein käytetty kryptografian menetelmä. Siinä annettu vaihtelevan kokoinen viesti tai muu tieto tiivistetään määrämittaiseksi tiivisteeksi. Näin voidaan muodostaan ns. digitaalinen sormenjälki, jonka perusteella vastaanottaja voi varmistua tiedon oikeellisuudesta. Käytännössä lähettäjä lähettää sekä alkuperäisen, että tiivistetyn viestin vastaanottajalle. Vastaanottaja muodostaa alkuperäisestä viestistä tiivisteen ja vertaa vastaanottamaansa sekä itse tekemäänsä tiivistettä toisiinsa. Jos ne ovat samat, on viesti aito.

Tällä menetelmällä pystytään huomaamaan se, jos joku on muuttanut alkuperäistä viestiä tai tiivistettä siirron aikana. Menetelmä on tehoton tilanteessa, jossa joku on muuttanut sekä viestiä että tiivistettä. Tämän estämiseksi tiiviste pitäisi salakirjoittaa lähettäjän salaisella avoimella, mutta tällöin kyseessä on digitaalinen allekirjoitus, joka katsotaan jo eri menetelmäksi. Tiivistysmenetelmä poikkeaa muista salakirjoitusmenetelmissä eniten juuri siinä, että salakirjoituksen yhteydessä käytössä ei ole minkäänlaista avainta.

Javassa on `MessageDigest`-niminen rajapinta tiedon tiivistys -menetelmän käyttöön. Rajapinnan kautta on mahdollista toteuttaa tiiviste ja tarkistaa tiivisteiden identtisyys. Kuten rajapintojen käytössä yleensäkin, `MessageDigest`in käyttäjän ei tarvitse tietää mitään varsinaisesta viestintiivistys algoritmista, jota rajapinnan kautta käytetään. Luokka sisältää seuraavat julkiset metodit [20]:

- `getInstance()` palauttaa luokan instanssin, joka toteuttaa parametrina annetun algoritmin. Parametreina voi olla myös algoritmin lisäksi valmistaja, jolloin palautetaan annetun valmistajan toteuttama algoritmi.
- `update()` lisää annetun datan tiivisteeseen. Mahdollisia lisäysvaihtoehtoja ovat yksi tavu, taulukollinen tavuja ja osajoukko taulukon tavuista.
- `digest()` laskee tiivisteen parametrina annetusta datasta, joka palautetaan tavutaulukossa. Parametriksi voidaan antaa myös tavutaulukko, jolloin se lisätään algoritmin syötteeseen ennen tiivisteen laskua.
- `isEqual()` vertaa, ovatko kaksi tiivistettä identtisesti samanlaisia.
- `reset()` tyhjentää tiivisteluokan instanssin vastaamaan tilaa, joka on uudella vastaluodulla instanssilla.
- `getAlgorithm()` palauttaa merkkijonona käytössä olevan tiivistealgoritmin nimen.
- `clone()` palauttaa kopion `MessageDigest`-instanssista. Tätä tarvitaan, koska joidenkin tiivisteen sisäisten operaatioiden täytyy kutsua `digest()`-metodia, joka tyhjentää instanssin sisällön.
- `getDigestLength()` palauttaa tavutaulukon koon.

Näiden metodien avulla voidaan toteuttaa helposti tiivisteen käsittelyssä tarvittavat operaatiot sekä lähettäjän että vastaanottajan päässä. Menetelmän luotettavuutta voidaan nostaa helposti ottamalla käyttöön jokin salalause. Tällöin lähettäjällä ja vastaanottajalla on käytössä sama salainen lause, jota käytetään tiivistyksen yhteydessä.

Salalausetta käytettäessä lähettäjä lisää `MessageDigest`-instanssiin tiivistettävän viestin ja salaisen lauseen. Tämän jälkeen suoritetaan tiivistys. Seuraavaksi instanssiin lisätään salalause ja äsken muodostettu tiiviste. Tämä yhdistelmä tiivistetään jälleen ja

lähetetään vastaanottajalle. Kun lähettäjä tekee vastaavat toimenpiteet ja vertaa tiivisteiden identtisyyden, on viesti saatu luotettavasti perille. Tämän menetelmän turvallisuus riippuu osapuolien yhteisestä salalauseesta: jos se pysyy salassa, ei kukaan pysty muuttamaan viestin sisältöä siirron yhteydessä osapuolten siitä tietämättä.

Pelkkä `MessageDigest`-luokka mahdollistaa tiivistettävän datan sisäänoton ainoastaan yksittäisinä tavuina. Jotta tiivisteiden muodostaminen olisi helppoa myös tiedostosta sijaitsevasta datasta tai muusta tietovirrasta, on Javassa erityiset `DigestInputStream`- ja `DigestOutputStream`-luokat, joiden avulla tietovirran sisältö voidaan lukea tiivisteeksi ja vastaavasti kirjoittaa tietovirtaan.

Myös `MessageDigest`-luokasta voidaan kirjoittaa oma toteutus. Tällöin voidaan toteuttaa itse tiivisteen tuottava hajautusalgoritmi. Kun algoritmi tehdään itse, saadaan tiivisteen toteuttamisesta kaikkialla toimiva ratkaisu, joka ei ole riippuvainen tarvittavan tiivistealgoritmin löytymisestä kyseisessä käyttöympäristössä.

5.2 Avaimet

Avaimet ovat tärkeitä komponentteja useiden salakirjoitusmenetelmien käytössä. Näistä yleisesti käytössä ovat ns. julkiset ja yksityiset avaimet. Näiden avaimien käyttöön on Javan standardi API:ssa mukana useita luokkia ja rajapintoja. Näistä tärkeimmät ovat `Key`-rajapinta sekä `KeyPairGenerator`- ja `KeyFactory`-luokat. Kyseisiä luokkia käytetään tavallisimpiin avainten käsittelytehtäviin kuten avainparien muodostamiseen.

`Key`-rajapinta muodostaa tavan mallintaa varsinaista avainta. Tämän lisäksi käytettävissä on erikseen julkiselle ja yksityisille avaimille `Key`:stä perityt omat rajapintansa. Myös Javan kanssa usein käytetyille DSA-menetelmällä muodostetuille avaimille on oma rajapintansa.

`KeyPairGenerator`-luokka mahdollistaa avainparien muodostamisen, mutta se on hyvin hidas operaatio. Tätä ei kuitenkaan tarvitse tehdä usein, koska yleensä avaimet otetaan avainten hallintajärjestelmästä. Tämän luokan voi kirjoittaa myös itse, jolloin käytettävän algoritminkin voi toteuttaa itse tai vaihtoehtoisesti käyttää valmiita algoritmeja.

`KeyFactory`-luokka tarjoaa tarvittavat operaatiot avainten lähettämiseen ja vastaanottamiseen. Nämä toimenpiteet käsittävät avain-olion muuttamisen ulkoiseen muotoon, joka on sopiva lähetettäväksi ja vastaavasti ulkoisessa muodossa olevan avaimen muuttamisen takaisin avain-olioksi. Myös tästä luokasta voi muodostaa oman toteutuksen.

Java Cryptography Extension (*JCE*) sisältää salaisen avaimen menetelmään käytettävät luokat. Salaisen avaimen menetelmässä on käytössä vain yksi ja sama avain molemmilla osapuolilla. Näitä `Key`stä perittyjä `SecretKey`-rajapinnan yhteydessä käytettäviä luokkia ovat `KeyGenerator`- ja `SecretKeyFactory`-luokat. Näistä ensimmäisen avulla voidaan muodostaa uusi salainen avain ja jälkimmäisen käyttötarkoitus on avaimen muuntaminen eri esitysmuotoihin vastaavasti kuin `KeyFactory`-luokkakin tekee julkisten/yksityisten avaimien kanssa.

5.3 Sertifikaatit

Sertifikaattien, joita on eri kokoisia ja eri muotoisia, tarkoituksena on todentaa esimerkiksi julkisen avaimen kuuluvan sille, jolle sen väitetäänkin kuuluvan. Tämä tapahtuu ns. luotetun kolmannen osapuolen avulla, jota kutsutaan sertifikaatin todentajaksi (*engl. certificate authority, CA*). Sertifikaatti kuitenkin vakuuttaa vain avaimen kuuluvan määrätylle taholle, mutta se ei takaa sitä, että avaimen omistaja olisi luotettava. Myös sertifikaattien luotettavuustasoja on useita. Alhaisimmilla tasoilla osapuolen todentamista ei tehdä kovinkaan perusteellisesti, jolloin sertifikaattiin ei voi täysin luottaa. Ylemmillä tasoilla tehdään sen sijaan hyvinkin tarkkoja tarkistuksia todennettavan tahon identiteetistä.

Sertifikaatit koostuvat kolmesta eri tiedosta. Ensimmäisenä on tahon nimi, jolle kyseinen sertifikaatti on julkaistu. Seuraavana on varsinainen todennettava kohde, joka voi olla esimerkiksi julkinen avain tai Java-sovelma. Viimeisenä osana on digitaalinen allekirjoitus, jolla sertifikaatin sisältö todennetaan. Näiden lisäksi täytyy jollakin tavalla saada sertifikaatin todentajan julkinen avain luotettavasti. Tietenkin olisi mahdollista lähettää sertifikaatti, joka sen sisältää, mutta kuka tällöin todentaa kyseisen sertifikaatin? Useimmissa Javaa tukevissa selaimissa ongelma on ratkaistu niin, että selain sisältää tunnetuimpien sertifikaattien todentajien julkiset avaimet.

Tunnettuja sertifikaattien todentajia on monia, josta aiheutuu myös ongelmia. Lähetetty sertifikaatti ei välttämättä kuulukaan vastaanottajan hyväksymien sertifikaattien joukkoon. Tästä syystä joudutaan usein muodostamaan sertifikaattiketjuja, jossa tietty lähettäjän käyttämä sertifikaatti on todennettu toisella vastaanottajan tukemalla sertifikaatilla. Sertifikaatteja käytetään WWW-selaimissa, salatuissa sähköposti-standardeissa, sähköisen kaupan protokollissa ja koodin allekirjoituksen yhteydessä (JAR-tiedostot).

Sertifikaatteja joudutaan joskus myös poistamaan, jos ne on esimerkiksi muodostettu väärennetyillä tiedoilla tai niiden kohteena oleva taho on muuttunut epäluotettavaksi. Tällöin sertifikaatin todentaja julkaisee sertifikaattien poistolistan (*engl. Certificate Revocaton List, CRL*). Javassa käytetään sertifikaattien yhteydessä `Certificate`-luokkaa, joka tarjoaa yleisimmät sertifikaattien käsittelyssä tarpeelliset toimenpiteet. Näitä metodeja ovat seuraavat [20]:

- `getEncoded()` palauttaa sertifikaatin tavutaulukkona. Jokaisen sertifikaatin täytyy pystyä muodostamaan kyseinen esitysmuoto.
- `verify()` tarkastaa sertifikaatin laillisuuden. Parametrina voidaan antaa myös julkinen avain.
- `getPublicKey()` palauttaa sertifikaatin julkisen avaimen.

Monista sertifikaattimuodoista yleisimmin käytetty on X509, joka on kansainvälisen ITU:n (*International Telecommunication Union*) standardi. Java API tukee tämän sertifikaatin kolmatta versiota. API:ssa on `X509` niminen luokka, joka tarjoaa useita metodeja sertifikaatin käyttöön. Muita suosittuja sertifikaatteja ovat PGP (*Pretty Good Privacy*) ja SDSI (*Simple Distributed Security Infrastructure*). Jos näitä sertifikaattityyppejä halutaan käyttää Javassa, täytyy niille tehdä oma `Certificate`-luokan aliluokka ja tarvittavat metodit. [35]

5.4 Avainten hallinta

Javan turvamallissa avaimilla on suuri merkitys digitaalisten allekirjoitusten käytön yhteydessä. Avainten hallintajärjestelmä huolehtii yksityisen avaimen välittämisestä

allekirjoitusta muodostettaessa ja vastaavasti julkisen avaimen välityksestä allekirjoitusta todennettaessa. Tähän järjestelmään kuuluu kolme eri osaa, joita ovat avaimet, sertifikaatit ja avaimen omaava taho, esimerkiksi henkilö tai yritys.

Javassa on versiosta 1.1 lähtien ollut mukana `javakey`-apuohjelma avaintenhallintaan. Sen heikkoutena on julkisten ja yksityisten avaimien sijoittaminen samaan suojaamattomaan tiedostoon, josta avaimien päättely on helppoa. Versiosta 1.2 lähtien mukana ollut `keytool` tarjoaa turvallisemman hallinnan avaimille, sillä siinä avain-tiedosto on suojattu salasanalla.

Javan avaimiin liittyvissä luokissa ei ole mitään tietoa niiden omistajasta. Avaimien omistajat määräytyvät usean luokan käytön mukaan. Näistä yksi on `Principal`-rajapinta, jota käytetään avaimen omistajan nimen kapselointiin. Nimi esitetään usein X.500 DN (Distinguished Name) -muodossa, joka käsittää usean eri tason nimikentän, joilla taho voidaan erottaa luotettavasti muista.

Tämän lisäksi käytössä on `Identity`-luokka, joka sisältää mm. julkisen avaimen ja sertifikaattilistan, joka todistaa identiteetin. Luokka käyttää `Principal`-rajapintaa, jolloin sen käytössä on myös nimi. Näin muodostetaan identiteetti, joka esittää, kenelle tietty julkinen avain kuuluu. Julkisen avaimen sisältää `Signer`-luokka, joka on peritty `Identity`-luokasta. Näin sen kapseloimana on myös saman tahon salainen avain.

Varsinainen avaintenhallinta tapahtuu `keytool`-apuohjelmalla. Apuohjelman toiminta perustuu tiedoston käyttöön, johon julkiset ja salaiset avaimet sekä sertifikaatit on tallennettu. Tämän tiedoston käyttöä varten Java API sisältää `KeyStore`-luokan, joka sisältää tarvittavat metodit avaintiedoston käyttöä varten.

5.5 Digitaaliset allekirjoitukset

Digitaalista allekirjoitusta käytetään Javassa eniten allekirjoitettujen luokkien toteuttamiseen. Varsinaisen digitaalisen allekirjoituksen käyttöä varten Java API:ssa on `Signature`-luokka, johon on koottu allekirjoituksissa tarvittavat operaatiot. Allekirjoitus toteutetaan tavallisesti `jarsigner`-apuohjelmalla, joka muodostaa allekirjoituksen luokat

sisältävään JAR-tiedostoon. Allekirjoitettuun JAR-tiedostoon kuuluu kolme erityistä elementtiä, jotka ovat manifesti-, allekirjoitus- ja lohkotiedosto.

Manifesti sisältää listan kaikista JAR-tiedostossa olevista allekirjoitetuista tiedostoista sekä jokaisesta tiedostosta muodostetun tiivisteen. Allekirjoitustiedosto sisältää varsinaisen allekirjoitustiedon. Tämä tiedosto on muodostettu manifestissa olevista tiivisteistä. Lohkotiedosto sisältää allekirjoitusdatan niin sanotussa PKCS7-muodossa. [34]

6 Turvalliset tietoliikenneyhteydet ja hajautetut Java-arkkitehtuurit

Javan turvaominaisuudet on suunniteltu suojelemaan käyttäjän työasemaa ja sen resursseja edellisissä luvuissa esitellyillä tekniikoilla. Tämän lisäksi Java tarjoaa hyvät mahdollisuudet rakentaa verkossa toimivia monikerros-sovelluksia. Tällaisten sovelluksien periaatteena on, että paikallisella koneella oleva sovellus voi käyttää hyväksi myös muiden etäkoneiden tarjoamia palveluita. Yleensä paikallisen sovelluksen ei tarvitse edes tietää, missä etäkone fyysisesti sijaitsee, vaan se voi käyttää etäkoneen palveluita aivan kuin ne olisivat paikallisella koneella. Tyypillinen esimerkki tästä on tietokantaa käyttävät sovellukset, joissa tietokanta sijaitsee eri koneella kuin varsinainen käyttäjän käyttämä sovellus (käyttöliittymä).

Sovelluksien monikerros-rakenteella voidaan parantaa myös tietoturvaa. Jos esimerkiksi käyttöliittymäkerros sijaitsee web-palvelimella ja sovellus- sekä tietokantakerros omalla palvelimellaan, ovat tietokantakerroksen arkaluontoiset tiedot melko hyvässä suojassa web-palvelimelle kohdistuvilta hyökkäyksiltä. Jotta turvallisuus maksimoituu, ei käyttöliittymän koodiin saa jättää tiedostonimiä, salasanoja ja palvelimien IP-osoitteita. [29]

Useimmat hajautetut sovellukset ovat oliopohjaisia. Tämä tarkoittaa sitä, että asiakas-sovellus pääsee palvelimen palveluihin olioiden rajapintojen kautta. Ainoastaan rajapinnat näkyvät asiakkaille ja varsinainen toiminnallisuus on palvelimella piilotettuna. Tämän takia hajautetut oliot jaetaan yleisesti kahteen luokkaan: 1) ns. tynkä- (*engl. stub*) olioihin, jotka ovat asiakaan puolella ja 2) ns. luuranko- (*engl. skeleton*) olioihin, jotka ovat palvelimen puolella ja sisältävät varsinaisen palvelun toteutuksen. Hajautettu olioarkkitehtuuri välittää metodikutsut näiden kahden eri paikoissa olevien olioiden välillä. Hajautetun sovelluksen käyttäjä ei yleensä tiedä, missä ja miten hänen käyttämänsä palvelu on toteutettu.

Javassa hajautettujen sovellukset toteutetaan RMI:tä (*Remote Method Invocation*), CORBA (*Common Object Request Broker Architecture*) -arkkitehtuureja ja uusimpana Enterprise JavaBeans -mallia käyttäen.

6.1 Turvalliset tietoliikenneyhteydet

Internetissä toimivissa hajautetuissa sovelluksissa tapahtuu luonnollisesti paljon verkkoliikennettä, jonka täytyy olla salattua, jottei mahdollinen verkkoliikennettä monitoroiva taho saa selville siirretyn datan sisältöä. IP Security Protocol Working Group (*IPSEC*) on ollut vaikuttamassa useiden suojattujen tietoliikenneprotokollien kehitykseen.

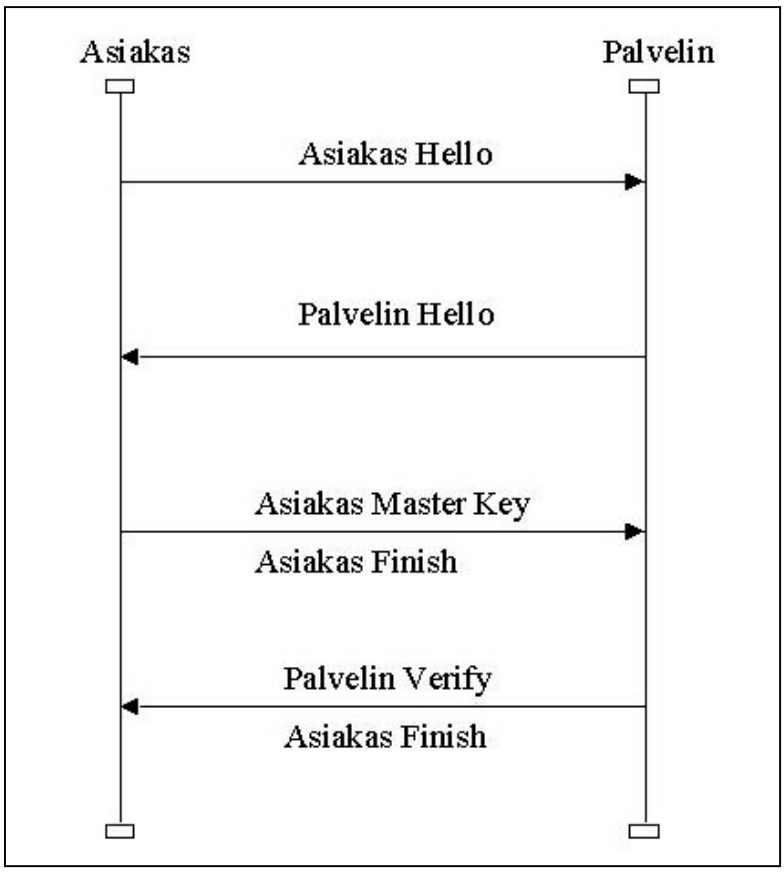
Turvalliseen kommunikointiin verkon välityksellä kuuluu kolme toisistaan erottuvaa vaatimusta. Ensinnäkin osapuolten välinen data on salakirjoitettava, jotta ulkopuoliset eivät pääse selville sen sisällöstä. Toiseksi salakirjoitusavaimet on hajautettava niin, että ne ovat datan vastaanottajien saatavilla. Kolmanneksi vastaanottaja on todennettava oikeaksi.

Näille vaatimuksille löytyy verrattain suoraviivaisesti vastaavat osa-alueet IETF:n määrittelemistä tietoturvan osa-alueista. Ensimmäinen liittyy selvästi tiedon luottamuksellisuuteen, koska välitettävä tieto ei saa olla sivullisten käytettäviessä. Toinen vaatimus kuuluu sekä luottamuksellisuuteen että eheyteen, koska tällä pyritään varmistamaan, että tieto on vain niille tarkoitettujen tahojen käytössä ja tieto pysyy muuttumattomana verkon yli siirron ajan. Kolmas osa-alue liittyy suoraan todennukseen. Seuraavaksi on kerrottu tarkemmin muutaman yleisimmän turvallisen verkkoliikenteen mahdollistavan protokollan ominaisuuksista. [31]

6.1.1 SSL

SSL on monissa hajautetuissa sovelluksissa käytetty protokolla verkkoliikennöintiin. Se tarjoaa turvallisen liikennöinnin lähetyskerroksen (TCP-protokolla) ja ylemmän tason sovelluskerroksen protokollien (HTTP, FTP) välillä. Sen perusominaisuuksiin kuuluu varmistaa, että SSL:ää käyttävät asiakkaat ovat todella yhteydessä siihen palvelimeen kuin luulevatkin olevansa. Se estää todentamattomien asiakkaiden yhteyden muodostamisen palvelimeen. Lisäksi SSL tunnistaa palvelimelle tulevan ja sieltä lähtevän datan muuttamisen siirron aikana.

SSL-protokollassa on kolme tärkeää elementtiä, jotka ovat tietuekerros, kättelyprotokolla ja hälytys. Tietueet ovat SSL:n käyttämä kokonaisuus, joihin kaikki lähetettävä data kapseloidaan. Tietue koostuu otsikosta (*engl. header*) ja varsinaisesta



Kuva 7: Asiakkaan ja palvelimen kättely SSL-protokollassa. [19]

datasta. Kaikki tietueet pakataan pakkausalgoritmilla ja suojataan käyttäen salakirjoitus- ja tietueen todennus (*engl. Message Authentication Code, MAC*) -algoritmeja.

Kättely koostuu kuvassa 7 esitetyistä vaiheista. Kättelyn aikana asiakas ja palvelin päättävät käytettävästä protokollaversiosta, istunnon ID-numerosta, salakirjoituksesta ja pakkausmenetelmästä. Ensimmäisessä vaiheessa asiakas lähettää HTTPS-muotoisen pyynnön palvelimelle. Palvelin vastaa tähän lähettämällä digitaalisen sertifikaatin, jotta asiakas voi varmistua palvelimen identiteetistä. Seuraavaksi asiakas todentaa palvelimen, luo istuntoavaimen ja lähettää sen palvelimelle julkisella avaimella salakirjoitettuna. Palvelin tulkitsee istuntoavaimen ja lähettää asiakkaan pyytämän objektin istuntoavaimella salakirjoitettuna.

Hälytys on eräs tietuekerroksella käytetty viestityyppi, jota käytetään ilmaisemaan tärkeää viestiä ja sen kuvausta. Tarvittaessa hälytysviestillä pystytään ilmoittamaan välitön

tarve katkaista asiakkaan ja palvelimen välinen istunto. Kuten muutkin viestit, myös hälytykset salakirjoitetaan ja pakataan. [7]

Java API:ssa on SSL:ää varten oma paketti `javax.net.ssl`, joka sisältää luokkia ja rajapintoja SSL-yhteyksien ja -istuntojen luomiseen. Nämä perustuvat normaaleihin yhteydenmuodostusluokkiin, mutta niihin on lisätty SSL-protokollan käyttö salausta varten. Nämä luokat löytyvät `sun.security.ssl` -paketista, jossa siis on kyseisten luokkien varsinainen toiminnallinen toteutus.

6.1.2 S-HTTP

S-HTTP on HTTP:n kanssa käytettävä turvallinen tietoliikenneprotokolla. Se sijoittuu OSI-mallin yläosaan sovellus- ja esitystapakerroksien tasolle. S-HTTP:n yhteydessä voidaan käyttää salaisen avaimen menetelmää ja viestit pystytään allekirjoittamaan. Todennus tapahtuu laskemalla MAC (*Message Authentication Code*) todennuskoodi. S-HTTP-viestit muodostetaan varsinaisesta viestistä ja lähettäjän sekä vastaanottajan vaatimista salakirjoitusmenetelmistä ja -avaimista.

Vastaanottaja tutkii S-HTTP-viestin otsikkokenttää, josta se saa selville käytetyt salakirjoitusmenetelmät. Sitten se muodostaa käytettyjen menetelmien mukaisen viestin purun takaisin selväkieliseksi tekstiksi. [31]

6.1.3 JSAFE/BSAFE

JSAFE on RSA:n kehittämä kaupallinen salakirjoitustyöväline elektronisen kaupankäynnin sovelluksiin käytettäväksi. Se on suunniteltu käytettäväksi Java Security API:n kautta. Sen käyttötarkoituksena on kehittää turvallisia Java-pohjaisia elektronisen kaupankäynnin sovelluksia Internetiin. JSAFE kuuluu nykyisin yhtenä osana BSAFE:en, joka on laaja salakirjoitusominaisuuksien toteuttamiseen tarkoitettu työkalu.

BSAFE koostuu RSA:n salakirjoitusalgoritmeista, joita käytetään useissa sovelluksien turvastandardeissa kuten SSL, TLS, SET ja S/MIME. Sen avulla Java-kehittäjä voi tehdä turvallisempia sovelmia, jotka tukevat julkisen avaimen salakirjoitusta mahdollistaen sekä eheyden että todennuksen. [26]

6.1.4 SKIP

Simple Key Management for Internet Protocols (*SKIP*) on OSI-mallin verkkokerroksella toimiva salausprotokolla, jonka toiminta perustuu lähettyjen pakettien salaukseen. SKIP mahdollistaa IP-pakettien lähettämisen toiselle koneelle salattuna ilman yhteydenmuodostamista etukäteen. Niinpä SKIP ei tunnista käyttäjiään eikä muodosta istuntoa pakettien siirron ajaksi.

Java-teknologian yhteydessä SKIP:iä voidaan käyttää sovelmien turvalliseen hajautukseen sisäisissä ja julkisissa verkoissa. SKIP:n avulla Java-sovelmat voidaan hajauttaa verkkoon todennetuissa ja salatuissa paketeissa. Koska sovelmien paketit voidaan todentaa, voi vastaanottaja luottaa siihen, että paketit ovat tulleet oikealta lähettäjältä ja ne eivät ole muuttuneet siirron aikana. [3]

6.2 Java RMI

RMI on Javan sisältämä tekniikka hajautettujen oliopohjaisten sovelluksien tekoon. RMI:n avulla JVM:ssä oleva olio voi kutsua myös toisessa virtuaalikoneessa olevan olion metodeja. RMI perustuu Javan tarjoamaan olioiden serialisointiin. Serialisoinnilla tarkoitetaan sitä, että olioita voidaan kirjoittaa tietovirtaan ja lukea ne takaisin olioiksi tietovirrasta. RMI:n käyttö on helppoa, mutta koska se on Javan oma arkkitehtuuri, ei sen käyttö palvelujen tarjoajana ole yhtä laajaa kuin esimerkiksi CORBA:n.

RMI:tä käytettäessä RMI-palvelimesta muodostetaan olio ja rekisteröidään se nimipalveluun, joka RMI:n tapauksessa on `rmiregistry`-sovellus. Tämän jälkeen asiakas voi kutsua palvelimen etärajapinnassa olevia metodeja. RMI:n välityksellä asiakas saa käyttöönsä referenssin tai kopion etäoliosta. Olion kopio välitetään serialisoimalla olio bittivirraksi ja muuttamalla se takaisin olioksi serialisoinnin avulla.

RMI:ssä on alunperin käytetty omaa protokollaa kutsujen välitykseen. Uusimmissa RMI:n versioissa voidaan käyttää CORBA:n IIOP-protokollaa, jolloin mahdollistuu ohjelmointikieliriippumaton oliokutsujen määrittely. Näin Java-oliot pystyvät kutsumaan CORBA-olioiden palveluja riippumatta siitä, mitä ohjelmointikieltä CORBA-olioiden toteutuksessa on käytetty. Näiden erona on tosin se, että CORBA-oliot sijaitsevat aina sillä

koneella, jossa ne on luotu, ja vain viittaukset välitetään verkon yli. Normaaleja RMI:n mukaisia olioita puolestaan voidaan siirtää parametreina verkon yli.

RMI on Internet-käytössä hyvin turvaton. Merkittävimpiä RMI:n turvattomuuteen vaikuttavia seikkoja ovat seuraavat:

- RMI ei todenna asiakkaita. Asiakas kutsuu referenssiä palvelimen etärajapintaan ja palvelin lähettää sen. Tämän jälkeisen kommunikoinnin oletetaan tulevan samalta asiakkaalta, joten tämä tarjoaa monia mahdollisuuksia erilaisiin hyökkäyksiin.
- RMI:n välittämillä olioilla ei ole minkäänlaisia pääsytarkistuksia.
- Rajapintojen oletetaan olevan yhtenäisiä luokkien toteutuksen kanssa. Tämän takia on tärkeää, että rajapintojen ja luokkien toteutuksen versionhallinta on ajan tasalla ja ne vastaavat toisiaan.
- RMI:n tapa muodostaa yhteys asiakkaan ja palvelimen välille on yksinkertainen. Oliot serialisoidaan ja lähetetään sellaisenaan ilman salausta verkkoon, joten kuka tahansa voi lukea kaiken lähetetyn datan ja tehdä siihen muutoksia.

JDK 1.2:sta alkaen pahimmat turva-aukot voidaan paikata käyttämällä SSL:ää salaamaan asiakkaan ja palvelimen välisen liikennöinnin. Tällöin välitetyt oliot voidaan todentaa. Yleisesti RMI:n käyttö on kuitenkin liian turvatonta Internetissä. Intraneteissä RMI:n avulla saadaan sen sijaan toteutettua käyttökelpoisia hajautettuja sovelluksia, koska ne toimivat silloin turvallisessa suljetussa verkossa.

6.3 Java ja CORBA

CORBA (*Common Object Request Broker Architecture*) on yleisin standardi hajautettujen sovellusten arkkitehtuurina. Sen on kehittänyt Object Management Group (*OMG*). Suurimpana erona RMI:hin verrattuna CORBA:ssa on kieliriippumattomuus ja sen tarjoamien palvelujen laajuus. Niinpä Java-ohjelmasta voidaan kutsua luokkia, jotka on tehty esimerkiksi C++:lla. Toinen merkittävä ero on, että CORBA:ssa olioita ei kuljeteta verkon yli vaan ainoastaan viittaukset niihin siirretään.

6.3.1 CORBA:n rakenne

CORBA-sovellusten kieliriippumattomuus on toteutettu lisäämällä olioiden rajapinnalle erillinen määrittelykieli, IDL (*Interface Descriprion Language*). CORBA-oliot kommunikoivat keskenään oliovälittäjien (*engl. Object Request Broker, ORB*) avulla, jotka välittävät etäproseduurien kutsut verkon yli. ORB:it puolestaan kommunikoivat keskenään Internet Inter-Orb Protocol (*IIOP*) -protokollan avulla.

IDL-rajapinnan rakenne on hyvin samanlainen kuin Java-rajapintojenkin. Niinpä niiden välisen kommunikoinnin rakentaminen on helppoa, eikä vaadi mitään erityisiä valmisteluja Java-luokkiin. Yhdessä tekniikat tarjoavat hyvän ympäristön hajautettujen sovelluksien toteuttamiseen: Java mahdollistaa olioiden kehittämisen ja hajauttamisen, kun taas CORBA mahdollistaa niiden keskinäisen kommunikoinnin toteuttamisen ja liittämisen muuhun ympäristöön. [21]

6.3.2 CORBA:n turvallisuus

CORBA:n turvallisuuteen vaikuttaa TCB (*Trusted Computing Base*), jolla varmistetaan, etteivät asiattomat tahot voi muokata turvamekanismia ja että turvapolitiikka pysyy asetettuna. CORBA:an on sisällytetty useita turvapiirteitä. Kaikki osapuolet yksilöidään ja todennetaan sekä valtuutetaan pääsynvalvonnan yhteydessä tekemään ennalta määrättyjä toimenpiteitä.

CORBA tarjoaa rajapinnat useiden kryptografisten menetelmien käyttöön, joiden avulla eri olioiden välinen liikennöinti voidaan salata. Käytettävä turvapolitiikka, eli mitkä toimenpiteet ovat milläkin taholla sallittuja, voidaan määritellä itse sovelluksissa tai ORB:ien avulla. CORBA:n yhteydessä käytettävät turvakomponentit ovat toisistaan riippumattomia, joten eri osia voidaan korvata toisilla ilman kaikkien komponenttien korvaamista. [8]

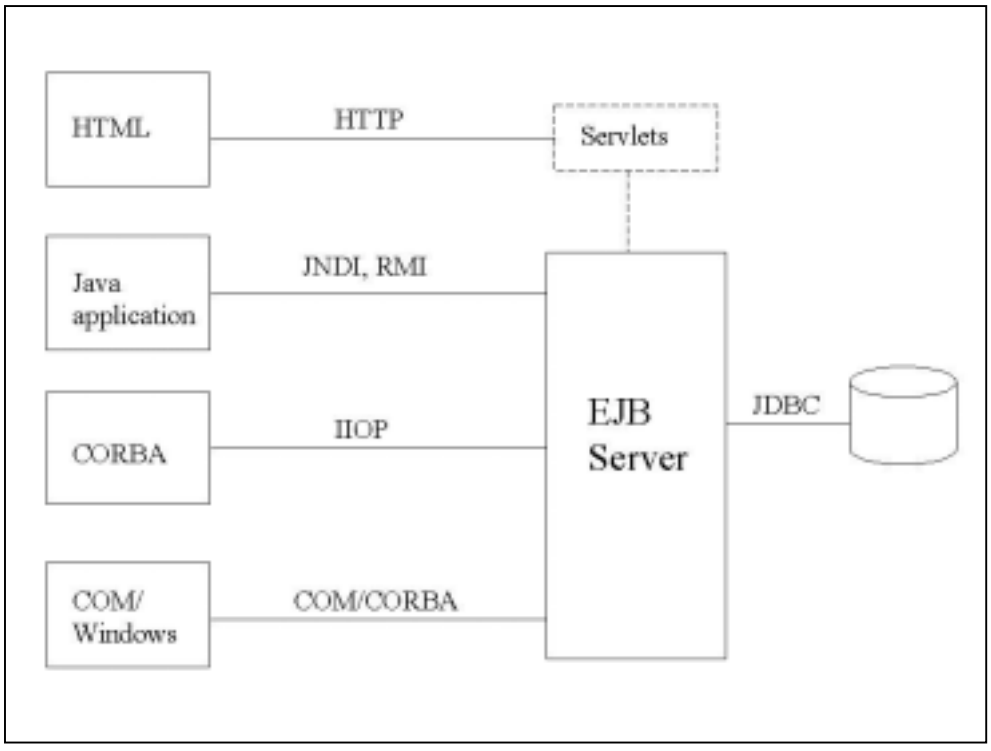
6.4 Enterprise Java Beans

Enterprise Java Beans (*EJB*) on Javan komponenttiarkkitehtuuri palvelimella vastaavasti kuin Java-pavut muodostavat asiakkaan puolen arkkitehtuurin. EJB mahdollistaa uudelleen käytettävien komponenttien tekemisen palvelimelle. EJB-mallin

tarkoituksena on koota useat matalan tason palvelut helposti käytettäväksi kokonaisuudeksi, jolloin ohjelmoijan ei tarvitse huolehtia niiden kaikista yksityiskohdista. EJB on yksi tapa toteuttaa Javan Middleware-tyyppistä monikerrosarkkitehtuuria.

6.4.1 EJB:n rakenne

EJB toimii erityisen sovelluspalvelimen (*engl. Application Server*) alaisuudessa. Eri valmistajien sovelluspalvelimissa saattaa toiminnallisuus ja turvaominaisuudet vaihdella. Tässä on esitelty Sunin J2EE ympäristön tarjoamia palveluita, joiden pitäisi pääpiirteissään sisältyä myös muiden valmistajien palvelimiin. EJB:t sijaitsevat palvelimen tarjoamassa säiliöluokassa (*engl. Container*), joka julkaisee EJB:n palvelurajapinnan. Säiliöluokka sisältää monia käytännön toimintoja, kuten yhteydenpidosta huolehtimisen asiakkaisiin, olioiden elinkaaren hallinnan, suojauksen, tapahtumanhallinnan ja tietokantayhteyksien hoitamisen. [24]



Kuva 8: EJB arkkitehtuuri. [9]

EJB:n toteutukseen liittyy kolme erillistä määrittelyä. Yksi niistä on kotirajapinta, jonka avulla säiliöluokka pystyy hallitsemaan papua. Toinen on etärajapinta, joka määrittelee EJB:n tarjoamat palvelut ja jonka kautta asiakkaat kommunikoivat EJB:n kanssa. Kolmantena on varsinainen toteutusluokka, joka toteuttaa etä- ja kotirajapinnoissa määritellyt metodit.

Koska sovelluspalvelin huolehtii yleisistä toiminnoista, ei EJB voi käsitellä näitä suoraan, esimerkiksi aloittaa uusia säikeitä. Asiakkaan kutsuväylänä käytetään oletuksena RMI-protokollaa. EJB tukee lisäksi CORBA:a, joten myös CORBA-asiakkaat voivat käyttää EJB:n palveluita. Lisäksi muita mahdollisia yhteysprotokollia on kuvassa 8, jossa on esitetty EJB-arkkitehtuurin rakennetta ja käytettäviä protokollia. [9]

6.4.2 EJB:n turvallisuus

Turvaominaisuuksista EJB tarjoaa todennuksen ja pääsynvalvonnan. (Tämä tarkastelu pätee J2EE JDK:hon. Muissa EJB toteutuksissa saattaa olla pieniä eroja käytännön toteutuksissa.) J2EE:ssä samaa todennuspolitiikkaa käyttävät käyttäjät kuuluvat samaan piiriin (*engl. realm*), joita ovat sertifikaatti- ja oletuspiirit. Sertifikaatteja käytetään HTTPS-protokollan kanssa tunnistamaan selainasiakkaat. Kaikkien muiden asiakkaiden todennukseen käytetään oletuspiiriä. Samaan oletuspiiriin kuuluvat käyttäjät voidaan jakaa edelleen ryhmiin, jolloin suurien käyttäjämäärien pääsynvalvonta helpottuu.

J2EE tarjoaa todennuspalvelun kaikille papujen asiakastyypeille. J2EE-sovelluksen käynnistyessä säiliöluokka avaa sisäänkirjautumisikkunan, joka tarkistaa, että käyttäjätunnus ja salasana kuuluvat oletusalueeseen. Todennuksen jälkeen kaikki J2EE-palvelimella olevat EJB:t, joihin kyseisellä käyttäjällä on oikeus, ovat hänen käytettävissään.

Asiakkaina toimivat ”stand-alone” tyyppiset Java-ohjelmat eivät kirjaudu sisään kutsuessaan EJB-komponentteja, joten niitä J2EE-ympäristö käsittelee tunnistamattomina ja tarjoaa komponenteista sellaiset palvelut, joita tunnistamattomille asiakkaille on määritelty. Myös muuntyyppiset asiakkaat, kuten web-selaimet, voivat päästä J2EE-palvelimelle ilman todennusta. Onhan hyvin yleistä, että esimerkiksi WWW-sivustoilla

olevissa sähköisissä kauppapalveluissa asiakkaat pääsevät selailemaan tuotteita ilman sisäänkirjautumista ja vasta tilausta tehdessään käyttäjä todennetaan.

Web-komponenttia tehdessä täytyy määritellä, onko se suojattu vai ei. Jos komponentti ei ole suojattu, pääsevät siihen käsiksi kaikki käyttäjät selaimellaan. Jos kyseinen komponentti kutsuu EJB:tä, käsitellään kutsut tunnistamattoman käyttäjän toimintatavalla. Suojatun komponentin tapauksessa todennukseen on käytettävissä kolme eri tapaa: perustodennus, lomakkeen (*engl. form*) tai sertifikaatin käyttö.

Perustodennuksessa palvelin pyytää käyttäjätunnuksen ja salasanan selaimen kautta. Lomaketta käytettäessä määritellään html- tai jsp-tiedosto, jonka välityksellä tunnus ja salasana pyydetään. Sertifikaatti-todennuksessa selaimelta pyydetään sertifikaatti, joka todentaa käyttäjän. Jos web-komponentti on todennettu jollakin näistä kolmesta tavasta, käsitellään komponentin kutsut EJB:lle todennettuna. [8, 32]

6.5 Java servletit

Servletit ovat Javan tapa muodostaa web-palvelinpään toiminnallisuus. Ne ovat Javalla kirjoitettuja komponentteja, jotka toimivat web-palvelimissa. Niitä varten palvelimen yhteydessä täytyy olla servlettimoottori (*engl. servlet engine*), joka huolehtii servlettien ajamisesta. Servlettien käyttötarkoituksena on tarjota turvallinen web-pohjainen pääsy dataan ja sen esitys ja muokkaus WWW-sivuilta käsin. Niille voidaan lähettää dataa WWW-sivujen lomake-elementiltä HTTP:n get- ja post-tyyppisillä pyynnöillä (*engl. request*). Servletit ovat hyvä vaihtoehto transaktio-tyyppiseen asiakkaiden pyyntöjen käsittelyyn ja niillä on nykyisin usein korvattu perinteisien CGI (Common Gateway Interface) -skriptien käyttö. Seuraavassa on lueteltu muutamia servlettien hyödyllisiä ominaisuuksia:

- Jokaisen pyynnön käsittelyyn muodostetaan oma instanssi servletti-luokasta. Tämä uusi instanssi ajetaan uudessa säikeessä, jolloin sekä servletti-luokan instanssi että säie ovat olemassa vain niin kauan kuin pyynnön käsittely kestää. Säikeen muodostaminen on kevyt operaatio verrattuna uuden prosessin muodostamiseen niin kuin CGI-ohjelmissa täytyy tehdä.

- Servletit toteutetaan standardi Javalla ja ne ovat siten ympäristö- ja käyttöjärjestelmäriippumattomia. Useimmat web-palvelimet tarjoavat tuen Servlet API:lle tai erillisen laajennusosan, jonka avulla servlettejä voidaan suorittaa.
- Servlettien tarjoamat turvallisuusominaisuudet ovat samat kuin muissakin Java-ohjelmissa. Esimerkkinä puskurin ylivuodon suoja, joka on aiheuttanut CGI-ohjelmissa vakavia turva-aukkoja.
- Servletti voi lähettää pyynnön edelleen toisille palvelimille. Näin voidaan tasata saman sisällön tarjoavien palvelimien kuormitusta tai jakaa yksi looginen palvelu useammalle eri palvelimelle.

6.5.1 Servletin ja asiakkaan välinen kommunikointi

Kun asiakas ottaa yhteyden palvelimella toimivalle servletille, täytyy turvallisuutta vaativissa sovelluksissa toteuttaa käyttäjän todennus. Servlettejä käytettäessä tähän on kolme perusmahdollisuutta. Ensimmäinen ja yleisin tapa näistä on web-palvelimen suorittama todennus. Koska kaikissa servlettien käyttöön pystyvissä palvelimissa on jonkinlainen mekanismi rajoittamaan käyttäjien pääsyä palvelimella oleviin tiedostoihin, voidaan todennukseen käyttää palvelimen omaa käyttäjätunnistusta.

Toinen tapa on käyttää servlettimoottorin tarjoamaa todennusta. Se voidaan tehdä neljällä eri tavalla, joita ovat: perustodennus, tiivistetodennus, HTTPS-asiakastodennus ja lomakkeeseen perustuva todennus. Perustodennus (*eng. Basic authentication*) on standardoitu käytettäväksi käyttäjien todennukseen HTTP:n kautta. Tätä käytettäessä selain avaa käyttäjälle sisäänkirjautumis-ikkunan, jossa kysytään käyttäjätunnusta ja salasanaa. Nämä tunnistetiedot koodataan HTTP-pyynnön `AUTHORIZATION`-otsikkoon. Sitä ei ole salattu millään salakirjoitusmenetelmällä lukuun ottamatta `Base64`-muotoon koodausta. Siksi tämän menetelmän yhteydessä on käytettävä esim. SSL:llä salattua verkkoyhteyttä.

Tiivistetodennus (*engl. Digest authentication*) on muuten samanlainen kuin perustodennusmenetelmä, mutta siinä salasana salakirjoitetaan MD5-menetelmällä selaimessa ennen lähettämistä. Mikään yleinen selain ei tue tätä ominaisuutta, joten sen

käyttö on hyvin vähäistä. Myös tällöin verkkoyhteys on salattava, koska tiivistetty salasana voidaan napata ja käyttää myöhemmin.

Myös asiakas voidaan pyytää todentamaan itsensä. Tähän käytetään SSL-protokollan mukana tulevaa digitaalista sertifikaattia. Kaikki yleisimmät selaimet ja web-palvelimet mahdollistavat todennukset digitaalisen sertifikaatin avulla. Servletit tarjoavat lisäksi formeihin perustuvan todennustavan. Sitä käytettäessä ohjelmoijan ei tarvitse erikseen lisätä todennusmekanismia sovelluksiinsa, vaan hän voi käyttää servlettimoottorin todennustietoja.

Kolmas tapa käyttäjän todennukseen servleteissä on toteuttaa todennusmekanismi itse sovellukseen. Tämä toteutetaan tavallisimmin lomakkeiden avulla. Tämän tyyppistä todennusta käytetään silloin, kun palvelin, jolle sovellus sijoitetaan, ei ole täysin ohjelmoijan hallinnassa. Jotta tiedon eheys säilyisi, servletit mahdollistavat tietoliikenteen salauksen servletin ja asiakkaan välillä salausmenetelmillä kuten SSL. Salausta käyttäen voidaan asiakkaan ja servletin välinen liikenne toteuttaa turvallisesti. [2]

6.5.2 Servlettien turvallisuus

Koska servletit ovat Javalla tehtyjä, on niillä myös yleiset Java-sovelluksien edut. Vääriin muistialueisiin viittaukset ja muut epävakautta aiheuttavat virheet tai tahalliset hyökkäykset eivät ole mahdollisia. Näin servletti ei pysty vahingoittamaan itse palvelinta, kuten esimerkiksi C-pohjaisilla palvelimella toimivilla skripteillä on mahdollista tehdä.

Servlettejä suoritetaan palvelimella olevalla laajennuksella. Käytettävät servletit voivat sijaita palvelimella tai ne voidaan ladata etäkoneelta. Servlettien lataus noudattaa normaalia Javan luokkalataustapaa. Koska luokkia voidaan ladata etäkoneilta, on servleteillä käytössä turvaohjaaja kuten sovelmillakin.

Oletuksena servletit ovat epäluotettavia, jolloin niillä ei ole pääsyä tiedostopalveluihin, verkkoyhteyksiin tai muihin mahdollista vahinkoa aiheuttaviin toimenpiteisiin. Paikallisella palvelimella oleville tai digitaalisella allekirjoituksella varustettuihin JAR-tiedostoihin pakatuille servleteille turvaohjaaja voi sen sijaan myöntää enemmän oikeuksia. Näin luotettujen servlettien avulla voidaan toteuttaa monipuolisia palveluja tarjoavia palvelimen laajennuksia.

Normaalikäytössä HTTP-protokollan kautta asiakkaan ja servletin välinen yhteys on turvaton. Tähän on ratkaisuna HTTPS-protokollan mukainen yhteys, jolla verkkoliikenne saadaan salattua. Edellä esiteltyjä todennusmenetelmiä ja salattua yhteyttä käytettäessä servletit ovat hyvin turvallinen tapa toteuttaa palvelimen laajennuksia verrattuna perinteisiin CGI-skripteihin.

7 Esimerkkisovellus

Tässä luvussa esitellään esimerkkisovelluksen toteutus alkaen määrittelystä ja päätyen lopussa oleviin huomioihin saavutetuista ominaisuuksista ja mahdollisista heikoista kohdista.

7.1 Järjestelmän määrittely

Esimerkkisovelluksena rakennetaan järjestelmä, joka hoitaa käyttäjien henkilötietojen ja oikeuksien ylläpitoa tietokannassa. Järjestelmä toteutetaan Javalla ja tietokannan käsittely SQL:llä. Järjestelmää käytetään WWW-selaimen välityksellä. Järjestelmä pystyy toimimaan myös Internetissä, joten sen täytyy sisältää turvaominaisuuksia, mutta toisaalta sovelluksen täytyy olla helposti käytettävä ja siirrettävä, joten ylimitoitettuja turvajärjestelyitä ei sallita.

Järjestelmän käyttäjät täytyy todentaa käyttäjätunnuksien ja salasanojen avulla. Kaikki verkkoliikenne on salattava, jottei kukaan pääse monitoroimaan salasanoja verkosta. Tietokantapalvelin ja WWW-palvelin toimivat tässä samalla koneella, mutta tietokanta voi sijaita myös toisella koneella, jolloin tämäkin väli täytyy salata.

Järjestelmän toteutuksessa käytetään Java Servlet Pages (*JSP*) -tekniikkaa sekä palvelimella olevia Java-papuja. JSP:n avulla muodostetaan sovelluksen käyttöliittymä, joka on käytettävissä WWW-selaimella. Java-pavut hoitavat sovelluksen toimintalogiikan ja yhteydet tietokantaan. Tietokantayhteys muodostetaan JDBC-ODBC-tekniikan avulla. JDBC (*Java Database Connectivity*) on Javan tietosilta erityyppisiin tietokantoihin, joka sisältää useita rajapintoja tietokannan käsittelyyn. ODBC (*Open Database Connectivity*) on yleinen käyttöjärjestelmien tukema tietosilta tietokantoihin.

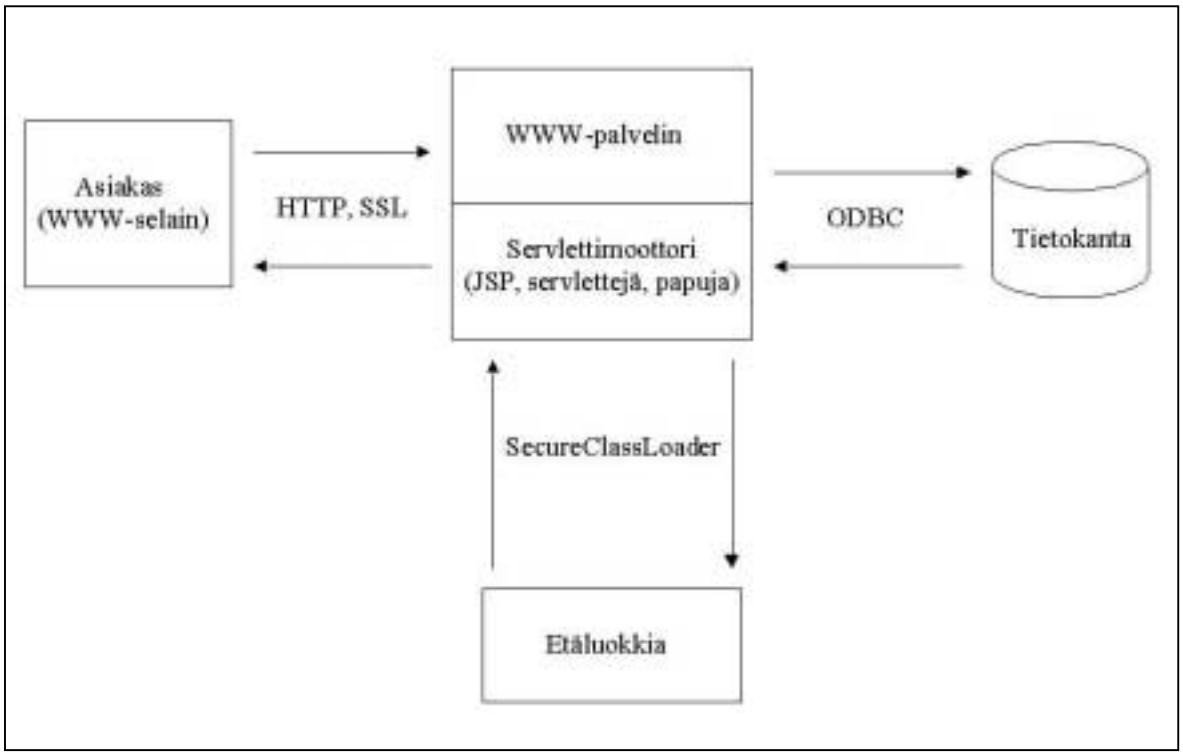
Järjestelmän toteutuksessa käytetään JDK:n 1.3 versiota. Servlettimoottorina on Tomcatin versio 3.2, joka sisältää lisäksi yksinkertaisen HTTP-palvelimen ominaisuudet ja toimii näin myös esimerkkijärjestelmän HTTP-palvelimena. Tomcat mahdollistaa SSL-salauksen käytön erillisen laajennuksen avulla, jolloin liikennöinti selaimen ja palvelimen välillä saadaan salattua. Järjestelmän tietokantana käytetään Microsoftin Accessia.

Järjestelmä toteutetaan standardi Javaa käyttäen, jotta se olisi mahdollisimman siirrettävä ympäristöstä toiseen. Samoin tietokannan käsittelyyn käytetään standardi SQL:ää ilman tietokantojen erikoisominaisuuksia, jotta järjestelmä voidaan siirtää toimimaan eri tietokantojen yhteydessä mahdollisimman helposti.

7.2 Järjestelmän suunnittelu

Käyttöliittymänä toimii WWW-selaimella. Järjestelmä on kolmikerroksinen, jossa ensimmäinen kerroksen muodostaa käyttäjän WWW-selaimella toimiva käyttöliittymä, toisen WWW-palvelimen yhteydessä olevalla servlettimoottorilla sijaitsevat JSP-sivut ja pavut sekä kolmantena kerroksena tietokanta. JSP-sivut käännetään niiden kutsun yhteydessä servleteiksi, joilla tuotetaan varsinainen selaimella näkyvä HTML-sivu. Lisäksi yksi papujen käyttämä luokka on esimerkin vuoksi sijoitettu etäkoneelle, josta se ladataan ohjelman suorituksen aikana.

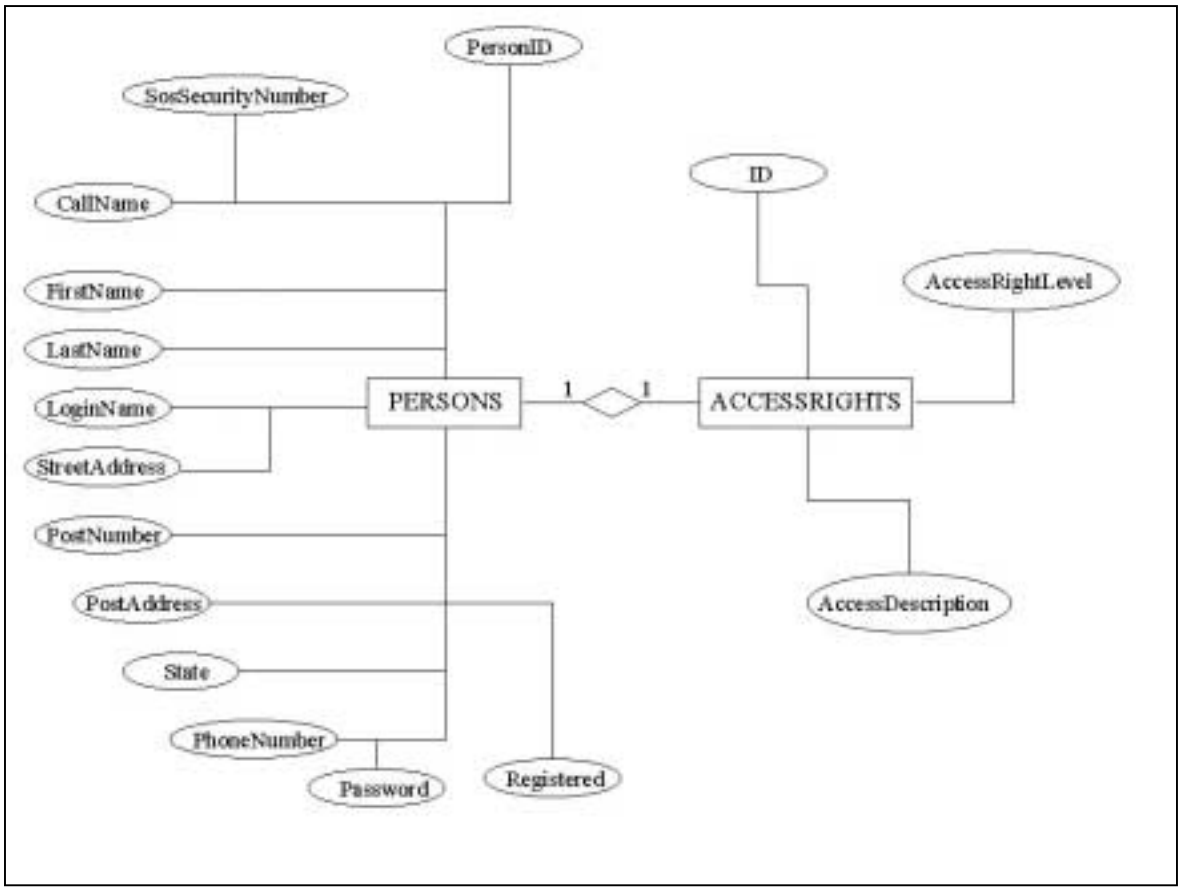
Järjestelmän käyttöliittymän muodostavat JSP-sivut, joita on erikseen pääkäyttäjälle ja tavallisille käyttäjille. Pääkäyttäjä voi selata kaikkia järjestelmään rekisteröityneitä



Kuva 9: Järjestelmän arkkitehtuuri.

käyttäjiä ja muokata heidän tietojaan erillisen JSP-sivun avulla. Hän voi myös lisätä uusia käyttäjiä ja poistaa olemassa olevia käyttäjiä. Tavallisella käyttäjällä on käytössä rekisteröitymissivu, jolta hän voi rekisteröityä käyttäjäksi sekä muokata omia tietojaan. Tavallinen käyttäjä ei voi muokata käyttäjäoikeustasoaan, vaan ainoastaan pääkäyttäjä voi muuttaa hänen oikeuksiensa tasoa.

Asiakkaan ja palvelimen välinen kommunikointi tapahtuu normaaliin JSP-sivujen tapaan, mutta HTTP-protokollaa käytetään SSL:n läpi (tätä menetelmää kutsutaan HTTPS:n käytöksi). Näin saadaan asiakkaan ja palvelimen välinen kommunikointi salattua, eivätkä sivulliset pysty saamaan verkossa liikkuvaa tietoa käyttöönsä. Järjestelmä sisältää myös yhden etäluokan syöttötietojen tarkistusta varten, joka ladataan suorituksen aikana JVM:ään `SecureClassLoader`-luokan avulla. Kyseinen luokka sijoitetaan etäpalvelimelle JAR-pakettiin, joka allekirjoitetaan digitaalisella allekirjoituksella. Näin voidaan todentaa luokan alkuperä ja muuttumattomuus latauksen aikana.



Kuva 10: ER-kaavio järjestelmän tietokannasta.

Asiakkaan ja tietokannan välinen toimintalogiikka ohjelmoidaan JSP-sivujen yhteydessä toimiviin Java-papuihin. Yhteys papujen ja tietokannan välillä tapahtuu JDBC-ODBC-tekniikalla. Jos tietokanta sijaitsee etäkoneella, on sen ja Java-papujen välinen liikennöinti salattava esimerkiksi SSH-yhteyden avulla. Yhteydenotot tietokantaan todennetaan normaaliin tapaan käyttäjätunnuksella ja salasanalla. Kyseinen käyttäjätunnus ja salasana ovat siis vain ohjelman käyttämiä Java-papujen muodostaessa yhteyden ODBC:n kautta, ja ne eivät vaikuta mitenkään varsinaisen käyttäjän todennukseen.

Kunkin käyttäjän todennus tapahtuu heidän omilla tunnuksilla ja salasanoillaan, jotka he itse valitsevat rekisteröinnin yhteydessä. Tietokannan rakenne on esitetty kuvassa 10. Käyttäjien tiedot siis tallennetaan PERSONS-tauluun ja mahdolliset käyttäjätasot on tallennettu ACCESSRIGHTS-tauluun.

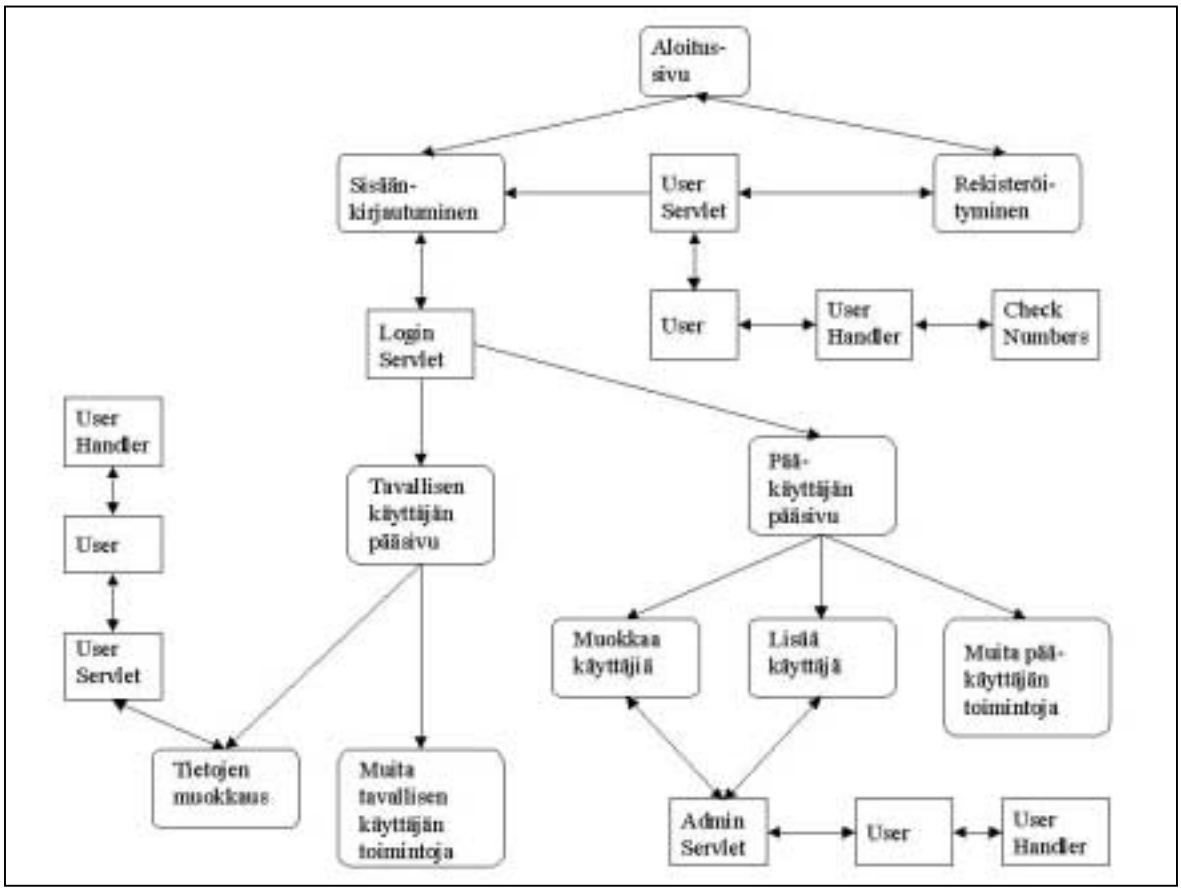
Jokaisen JSP-sivun yhteyteen liitetään tarkistus siitä, onko sivun käyttäjällä oikeudet kyseisellä sivulla olevan toiminnon käyttöön. Käyttäjän järjestelmään kirjautumisen jälkeen hänen käyttäjätietonsa pidetään tallessa Java-pavussa, joka on toiminnassa koko käyttäjän istunnon (*engl. session*) ajan. Istunnosta saadaan eri sivuille tultaessa selville mm. käyttäjän oikeustaso.

7.3 Järjestelmän toteutus

Järjestelmän toteutuksessa tavallisille käyttäjille on omat sivunsa, joissa he voivat muokata ja selata omia tietojaan. Pääkäyttäjille on erillinen oma sivustonsa, joilla he voivat selata järjestelmään rekisteröityneiden käyttäjien tietoja, muokata niitä sekä lisätä uusia ja poistaa vanhoja käyttäjiä. Käyttäjän oikeudet sivun käyttöön tarkistetaan aina JSP-sivulle tultaessa, jotta asiattomat eivät pääse tietoihin käsiksi.

Uuden käyttäjän rekisteröityessä järjestelmään, `UserServlet` hoitaa käyttäjän tietojen rekisteröimisen käyttäen `User`-pavun palveluita. `User`-papu käyttää `UserHandler`-luokan palveluita tietokannan käsittelyyn liittyvissä tehtävissä. `UserHandler` käyttää edelleen käyttäjien lisäämisen yhteydessä ladattavaa `CheckNumbers`-etäluokkaa lisättävän käyttäjän henkilötunnuksen ja postinumeron muodon tarkastukseen. Sisäänkirjautumisen yhteydessä kirjautumisesta vastaava

`LoginServlet` ohjaa käyttäjän hänen oikeuksiensa mukaisille sivuille. Tieto käyttäjästä liikkuu JSP-sivujen yhteydessä olevassa `User`-pavussa, johon tiedot haetaan tietokannasta



Kuva 11: Järjestelmän tietovirtakaavio.

käyttäjän kirjautuessa järjestelmään ja joka on käytössä koko käyttäjän istunnon ajan.

Tavallisen käyttäjän sivuilla `UserServlet` vastaa käyttäjän tietojen päivittämisestä. Pääkäyttäjän sivuilla `AdminServlet` hoitaa pääkäyttäjän tekemien toimenpiteiden toteuttamisen. Myös se käyttää User-pavun palveluita hyväkseen toimenpiteiden suoritukseen. Tiedot käyttäjän selaimen ja palvelimen välillä salataan SSL-protokollan avulla.

Esimerkin vuoksi käyttäjän rekisteröitymisen yhteydessä annetuista syötteistä tarkastetaan postinumeron pituus ja henkilötunnuksen muoto (päivät, kuukaudet ja vuodet järjellisiä lukuja), jotta tietokantaa ei pystytä täyttämään täysin järjettömillä tiedoilla.

Tämän toiminnon toteutukseen käytetään eri palvelimelta ladattavaa luokkaa, jonka lataus suoritetaan `SecureClassLoader`-luokasta laajennetun luokkalataajan avulla. Kyseinen luokkalataajatyyppi on valittu, koska se mahdollistaa digitaalisen allekirjoituksen käytön luokkien latauksen yhteydessä ja näin voidaan varmistua luokkien alkuperästä.

7.4 Huomioita esimerkkijärjestelmästä

Järjestelmän suorituksen aikana Javan turvamallin kaikki komponentit ovat käytössä. Paikalliset luokat ladataan JVM:ään oletusluokkalataajalla ja etäluokan lataukseen käytetään `SecureClassLoader`-luokasta perittyä luokkalataajaa. Ennen luokkien suoritusta JVM käyttää normaalisti tavukoodin tarkastajaa luokkien rakenteellisen virheettömyyden tarkistamiseen ja turvaohjaaja huolehtii, ettei järjestelmässä käytettävä etäluokka yritä laittomia operaatioita.

Käyttäjän WWW-selaimen ja WWW-palvelimen välinen liikenne on salattu SSL:n avulla, joten tämän välin liikenteen sisältöä ei pysty monitoroimalla selvittämään. SSL-yhteyttä muodostettaessa palvelin ja asiakas muodostavat yhteyden kuvassa 7 esitetyllä tavalla, joten palvelimen identiteetistä voidaan varmistua. Käyttäjä todennetaan hänen käyttäjätunnuksensa ja salasanansa avulla. Näin ollen selaimen ja palvelimen välissä olevat turva-aukot liittyvät inhimillisiin tekijöihin.

Palvelimen ja tietokannan välinen kommunikointi hoidetaan tarvittaessa SSH:lla salattuna, jolloin tältäkään väliltä ei liikennöintiä voida salakuunnella. Tietokannan turvallisuus saattaa vaarantua, jos mahdollinen tunkeutuja saa tietokannan käyttäjätunnuksen ja salasanan luettua pavun luokkatiedostosta, joihin ne on sijoitettu. Nämä kannattaisi sijoittaa erilliseen tiedostoon, joka hieman hidastaa tunkeutujaa, mutta ei estä salasanojen selvittämisessä.

Etäkoneelta ladattavan luokan alkuperä todennetaan `SecureClassLoader`-luokkalataajan käytön yhteydessä JAR-paketin yhteydessä olevalla digitaalisella allekirjoituksella. Näin luokan alkuperä pystytään todentamaan ja varmistumaan, ettei luokka ole muuttunut siirron aikana. Tässä täytyy luottaa etäkoneen ylläpitäjään siinä suhteessa, että sieltä ladattava luokka on aito oikean toteutuksen omaava luokka. Tämä

sisältää pienen riskin, mutta latauksen aikainen luokan muuttumattomuus pystytään varmistamaan.

JSP-sivujen yhteydessä käyttäjän selaimen osoitekenttään tulee osoite, josta käytettävät JSP-sivut ja servletit löytyvät. Tämän lisäksi HTTP:n get-tyyppisissä pyynnöissä tulevat kaikki sivuille/servleteille välitettävät parametrit näkyviin selaimen osoitekenttään. Tämä antaa mahdolliselle tunkeutujalla paljon tietoa mahdollisista hakemistoista, joista mahdollisia turvapuutteita kannattaisi etsiä. Turvallisuutta voisi hieman parantaa käyttämällä get-pyynnön sijaan post-tyyppistä pyyntöä, jolloin välitettävät parametrit eivät näkyisi osoitekentässä, vaan ne lähetettäisiin erillisessä tietovirrassa.

8 Yhteenveto

Internetissä ajettavien hajautettujen sovelluksien tietoturvaan liittyy lukuisia asioita, joista tässä työssä keskityttiin Javan tarjoamiin turvapiirteisiin sekä muutamiin yleisiin tekniikoihin, joita myös Javan kanssa käytetään. Vaikka Java sisältää tietoturvan kannalta useita kehittyneitä piirteitä, ei avoimessa verkkoympäristössä tapahtuva ohjelmien ajo voi koskaan olla täysin turvallista. Java-ympäristö toimii eri käyttöjärjestelmissä ja sovelmat ajetaan WWW-selaimissa. Vaikka itse Java-ympäristö olisikin turvallinen, ei tämä välttämättä tarkoita sitä, että ympäristö, jossa JVM:ää ajetaan, olisi turvallinen. Ajettaessa ohjelmia Internetissä on olemassa monia vaaratekijöitä, jotka voivat Javan turvamallista huolimatta vaarantaa tietoturvaa.

8.1 Yleisiä Internetin turvaongelmia

Yleisimmät turvaongelmat pohjautuvat sähköpostiin, salasanoihin ja luotettuihin isäntäkoneisiin. Salasanojen suojaus onkin hyvin tärkeää turvallisuuden varmistamiseen, koska ne muodostavat osan käyttäjän tunnistus- ja todennus-prosessia. Turvallisuusongelmia saattaa tulla, jos käyttäjätunnus, salasana ja koneen IP-osoite lähetetään Internetin läpi ilman minkäänlaista suojausta.

Salasanojen kaappausta varten on olemassa lukuisia pakettien monitorointiohjelmia. Myös salasanat sisältävään tiedostoon saattaa olla pääsy ilman todennusta, jolloin sopiva ohjelma löytää ne helposti ja tietoturva vaarantuu. Käyttäjien hyväuskoisuutta hyödyntäen hyökkääjällä on myös mahdollista saada selville salasanoja. Hyökkääjä voi esimerkiksi käyttää Java-sovelmaa, joka pyytää käyttäjää syöttämään WWW-sivulla oman verkkoon liitetyn koneensa käyttäjätunnuksen ja salasanansa turvallisuutensa takia. Jotkut ajattelemattomat käyttäjät saattavat syöttää salasanansa mitään epäilemättä.

Tunnetut isäntäkoneet ovat UNIX:issa käytetty mekanismi, joka perustuu IP-osoitteen todennukseen. Tämän tyyppinen todennus on hyvin haavoittuvainen IP-osoitteiden vakoilulle, koska luotettuja IP-osoitteita säilytetään tiedostossa. Jos hyökkääjä pääsee murtautumaan jollekin luotetulle koneelle, on hänellä silloin pääsy myös kaikille koneille jotka tiedostossa on lueteltu.

8.2 Internet ja sovelmat

Vaikka Javassa on edellä esitelty tehokas turvamalli, ei se pysty takaamaan täydellistä turvallisuutta. Aina ajan kuluessa ja tekniikoiden kehittyessä jostain löytyy turva-aukkoja, joita taitava tunkeutuja pystyy käyttämään hyväkseen hyökkäyksessään. Kun käyttäjä sitten lataa pahaa aavistamattomana sovelman WWW-sivun mukana, saattaa se ollakin hyökkäämässä käyttäjän koneelle. Erityyppisiä sovelmien avulla tehtäviä hyökkäyksiä voidaan jakaa neljään osaan niiden vakavuuden mukaan [17]:

- Järjestelmää tai resursseja muokkaamaan pyrkivät hyökkäykset.
- Käyttäjän yksityisyyttä loukkaavat hyökkäykset.
- Järjestelmän resurssien normaalin käytön estävät hyökkäykset.
- Käyttäjää ärsyttävät tai muuten kiusaavat hyökkäykset.

Ensimmäistä tyyppiä näistä sanotaan hyökkääviksi sovelmiksi niiden aiheuttaman uhan vakavuuden takia. Niiden tavoitteena on muokata järjestelmää. Kolmeen jälkimmäiseen ryhmään kuuluvia sanotaan ilkeämielisiksi (*engl. malicious*) sovelmiksi niiden lievemmän vakavuuden johdosta. Tällaisten sovelmien huomattavimpia aikaansaannoksia voivat olla koneen uudelleenkäynnistys tai selainohjelman sulkeminen.

Hyökkäävien sovelmien murtautumisia ei ole julkisesti ilmaantunut. Vaikka niitä ei olekaan ollut, ei se tarkoita sitä että sellaiset olisivat mahdottomia. Useiden yliopistojen turvahenkilöt ovat löytäneet puutteita JDK:n ensimmäisistä versioista, joita hyökkäävät sovelmat voisivat käyttää hyväksi. Uudemmissa JDK:n versioissa havaitut puutteet on korjattu, joten turvallisuuden maksimoimiseksi kannattaa käyttää aina mahdollisimman uutta JDK:n versiota. [18]

Ilkeämielisiä sovelmia on tullut julkisuuteen muutamia. Tyypillinen tilanne on WWW-sivun mukana ladattu sovelma, joka yrittää tunkeutua käyttäjän koneelle. Sovelma saattaa yrittää lähettää sähköpostia, soittaa äänitiedostoja tai yrittää jotakin muuta käyttäjää häiritsevää. Se, mitä sovelmat pääsevät tekemään, riippuu selaimen asetuksista. Tuntemattomille sivuille mennessä sovelmien suorituksen voi estää kokonaan, kun taas luotetuilla sivuilla sovelmille voidaan antaa täydet suoritusoikeudet.

Sovelmien ajon turvallisuus riippuu pitkälti Javan ajoympäristöstä. Esimerkiksi Microsoftin Internet Explorerin Java virtuaalikoneen eri versioissa on ilmennyt puutteita. Siinä tavukoodin tarkastaja salli laittomia tyyppimuunnoksia Javan eri tietotyyppien välillä tavukoodissa. Tämä on mahdollistanut ilkeämielisten sovelmien teon, jotka pystyvät tekemään lähes mitä tahansa käyttäjänsä koneella. Myös Sunin omassa JVM:ssä ja Netscapen eri versioissa on ollut turvallisuutta vaarantavia virheitä, mutta uusissa versioissa nämä havaitut ongelmat on korjattu. Esimerkiksi Netscapen selainversioiden 4.05 – 4.74 JVM:ssä oli vika, joka mahdollisti käyttäjän koneen tiedostoja lukevan sovelman teon ilman, että käyttäjä tätä huomasi. [11]

Koneen resursseja varaava sovelma on perusesimerkki harmillisesta sovelmasta. Tällainen sovelma voi avata suuren määrän ikkunoita, jolloin näppäimistön ja hiiren käyttö lopulta estyy. Yksinkertainen tapa tällaisen toteutukseen on tehdä sovelma, jonka prioriteetti asetetaan suurimmaksi mahdolliseksi ja pysäytysmetodi määritellään uudelleen arvoon `null`. Kun tällainen sovelma asetetaan näyttämään esimerkiksi kuvaa ja käynnistetään, jää sen suoritus ikuiseen silmukkaan. [8]

CLASSPATHiin itse sijoitettujen luokkien alkuperästä ja toiminnasta kannattaa varmistua, koska ne käsitellään luotettuina luokkina, eikä niiden toiminnalle oletuksena aseteta rajoituksia. Säikeissä on muutama metodi, joita käyttämällä on helppo tehdä ilkeämielinen sovelma, joka jää pyörimään ikuiseen silmukkaan. Tällaisia ovat säikeen `stop()`-, `suspend()`- ja `resume()`-metodit. `Stop()`-metodin käytöllä on mahdollista saada sovelman käyttämä olio ns. epäyhtenäiseen tilaan. Jos muutkin sovelmat käyttävät samaa oliota, voi niiden toiminta olla mielivaltaista.

`Stop()`-metodin sijasta onkin suositeltavaa käyttää muuttujaa, joka ilmaisee, jatketaanko sovelman suoritusta. Tämän muuttujan tilaa sitten vain tarkkaillaan säännöllisin väliajoin sovelmaa ajettaessa. `Suspend()`- ja `resume()`-metodien käytöllä saadaan helposti aikaan lukkotilanne (*engl. deadlock*), jolloin mikään säie ei voi tehdä yhtään mitään.

8.3 Huomioita

Turvamallista huolimatta Java-ohjelmien ajo Internetissä sisältää riskejä. Turvallisuutta ei voi koskaan täysin taata, kun ollaan tekemisissä avoimessa verkossa suoritettavien sovelluksien kanssa. Jotta tehtävästä ohjelmasta saataisiin mahdollisimman turvallinen, täytyy suunnittelussa kartoittaa huolellisesti kaikki tunnetut turva-aukot ja suunnitella, kuinka ne voidaan suojata.

Javan tarjoamat turvaominaisuudet mahdollistavat kohtuullisen turvallisen ajoympäristön hajautetuille sovelluksille. Vaikka virheitä ja turva-aukkoja löytyy eri selainvalmistajien Java-ajoympäristöissä ja myös Sunin omasta JVM:ssä, on turvallisuuteen suuresti vaikuttava tekijä itse käyttäjä. Selainvalmistajat pyrkivät korjaamaan havaitut viat mahdollisimman nopeasti ja jakelemaan päivityksiä WWW-sivuillaan. Todellisuudessa käyttäjät eivät aina vaivaudu päivittämään selainversioitaan jokaisen pienen päivityksen jälkeen, jolloin turvattomat ohjelmat pysyvät käytössä.

JDK 1.0 – 1.1 versioiden alkuperäistä turvamallia rajoitti niiden pääsynvalvonnan jäykkyys, koska kaikki eri toimenpiteiden teon sallimiseen liittyvät asiat olivat kiinteästi ohjelmakoodissa, eikä niihin pystynyt vaikuttamaan muokkaamatta koodia. Sen sijaan uudempien versioiden pääsynvalvonta on hyvin käyttökelpoinen helposti muokattavine policy-tiedostoineen. Kun käyttäjä pystyy itse muokkaamaan käytettävän pääsynvalvonnan yksityiskohtia ilman koodin uudelleen kääntämistä, on pääsynvalvonnan käyttökelpoisuus aivan eri tasolla. Huonona puolena tässä tosin on, että käyttäjä saattaa myöntää liian paljon oikeuksia tuntemattomille luokille, jolloin turvallisuus voi kärsiä käyttäjän varomattomuuden takia.

Laajoja hajautettuja sovelluksia toteutettaessa Javan EJB-arkkitehtuuri on käyttökelpoinen ja turvallinen tapa toteuttaa suuriakin kokonaisuuksia. Myös CORBA on hyvät turvaominaisuudet sisältävä tekniikkaa hajautettujen sovellusten toteutukseen Javalla ja koska se on myös esimerkiksi C++ yhteensopiva, voidaan samaan sovellukseen lisätä C++:lla tehtyjä komponentteja.

Java-kielen ja ajoympäristön kehittyessä yhä turvallisemmaksi avoimessa verkossa täytyy aina uusien parannustenkin jälkeen muistaa tietoturvan perusperiaate: ”Vaikka

tiettyyn järjestelmään ei olisikaan murtauduttu, ei se tarkoita, että järjestelmä olisi murtamaton ja täysin turvallinen.”

9 Lähdeluettelo

[1] Amoroso Edward G., Fundamentals of Computer Security Technology, Prentice-Hall, 1994

[2] Avedal Karl, Profession JSP, Wrox Press, Birmingham, 2000

[3] Aziz Ashar, Simple Key-Management for Internet Protocols (SKIP), <URL: http://www.skip-vpn.org/inet-95.html>, luettu 3.2.2001

[4] Bruce Glen, Security in Distributed Computing, Hewlett-Packar Professional Books, New Jersey, 1997

[5] Common Criteria, Common Criteria for Information Technology Security Evaluation Parts 1 – 3, 1999, saatavilla osoitteesta <URL: http://csrc.nist.gov/cc/ccv20/ccv2list.htm>, luettu 19.5.2001

[6] Flanagan David, Java in a Nutshell, O'Reilly, 1997

[7] Freier Alan O., The SSL Protocol Version 3.0, Transport Layer Security Working Group, 1996, <URL: http://home.netscape.com/eng/ssl3/draft302.txt>, luettu 3.2.2001

[8] Gritzalis Stefanos, Distributed Component Software Security Issues on Deploying a Secure Electronic Marketplace, Information Management & Computer Security, 1/2000, s. 5-13

[9] Hemrajani Alan, The state of Java middleware, Part 2: Enterprise JavaBeans, JavaWorld April 1999, <URL: http://www.javaworld.com/javaworld/jw-04-1999/jw-04-middleware_p.html>, luettu 5.12.2000

[10] IETF, "Active IETF Working Groups, Security Area", <URL: http://www.ietf.org/html.charters/wg-dir.html#Security_Area>, luettu 3.11.2000

[11] Johnston Stuart J., Java Lets Hackers Attack Your Browser, PC World, 11/2000, s. 47

[12] Järvinen Petteri, Sinulle on sähköpostia, Teknolit, 2000

[13] Kerttula Esa, Tietoverkkojen tietoturva, Edita, Helsinki, 1999

[14] Levy Elias, Executing Java Programs Securely, Network Computing, 8/2000, s. 114-116

[15] MacGregor R, Java Network Security, Prentice Hall, 1998

[16] Majander Olli, Waara Waanii Werkossa – salaus Internetissä, Tietokone, 3/2001, s. 64-65

[17] McGraw Gary, Java Security – Hostile Applets, Holes and Antidotes, Wiley, 1996

[18] McGraw Gary, Securing Java – Getting Down to Business with Mobile Code, Wiley, 1998

[19] McGregor Tony, Secure Socket Layer, 1995, <URL: http://byerley.cs.waikato.ac.nz/~tonym/articles/ssl/ssl.html> , luettu 3.2.2001

[20] Oaks Scott, Java Security, O'Reilly, Sebastobol, 1998

[21] Orfali Robert, Client/Server Programming with Java and CORBA, Wiley, New York, 1997

[22] Paavilainen Juhani, Tietoturva, Suomen Atk-keskus, 1998

[23] Pfleeger Charles P, Security in Computing, Prentice Hall, 1997

[24] Roman Ed, Mastering Enterprise JavaBeans and the Java2 Platform Enterprise Edition, Wiley, Toronto, 1999

[25] RSA, "RSA Laboratories' Frequently Asked Questions About Today's Cryptography", <URL: http://www.rsasecurity.com/rsalabs/faq/>, luettu 3.11.2000

[26] RSA, "RSA BSAFE", <URL: http://www.rsasecurity.com/products/bsafe/index.html>, luettu 1.2.2001

[27] Rubin Aviel D, Web Security Sourcebook, Wiley, 1997

[28] Saarimäki Mikko, Diskreettiä ja äärellistä matematiikkaa, ER-Paino/Yliopistopaino, Jyväskylä, 1997

[29] Sibakov Marko, Kerroksilla tietoturvaa, Tietokone, 4/2001, s. 109-110

[30] Spinellis D., Security Requirements, Risks and Recommendations for Small Enterprise and Home-office Environments, Information Management & Computer Security, 3/1999, s. 121-128

[31] Sun, Implementing Java Security Student Guide – kurssimateriaali, Sun Microsystems, 1998

[32] Sun, J2EE(tm) Developer's Guide, <URL: http://java.sun.com/j2ee/j2sdkee/techdocs/guides/ejb/html/Security2.html>, luettu 16.12.2000

[33] Sun, Secure Computing with Java: Now and the Future, <URL: http://java.sun.com/marketing/collateral/security.html>, luettu 18.12.2000

[34] Sun, Security Features Overview, <URL: http://java.sun.com/docs/books/tutorial/security1.2/overview/index.html>, luettu, 16.12.2000.

[35] Sundsted Todd, Construct secure networked applications with certificates – Part 1, JavaWorld January 2001, <URL: http://www.javaworld.com/javaworld/jw-01-2001/jw-0112-howto.html>, luettu 2.2.2001

[36] Tiveke (Liikenneministeriön kansallinen tietoverkkojen kehittämisohjelma), ”Tiveken tietoturvasivut”, <URL: http://www.tieke.fi/arkisto/tiveke/turva.htm>, luettu 3.11.2000

Liitteet

Liite 1: Työssä käytettyjen suomenkielisten termien englanninkieliset vastineet

Asiakas	*Client*
Hiekkalaatikkomalli	*Java Sandbox*
Istukka	*Socket*
Isäntäkone	*Host*
Java papu	*Java Bean*
Java virtuaalikone	*Java Virtual Machine*
Java ydin API	*Java Core API*
Koodin lähde	*Code source*
Luokkalataaja	*Classloader*
Oikeudet	*Permissions*
Ominaisuus	*Property*
Otsikko	*Header*
Palvelin	*Server*
Piiri	*Realm*
Politiikka	*Policy*
Pääsynvalvoja	*Access Controller*
Roskien keruu	*Garbage Collection*
Servlettimoottori	*Servlet Engine*
Sovelluspalvelin	*Application Server*
Säiliöluokka	*Container*
Sovelma	*Applet*
Suojattu alue	*Protection Domain*
Tiedon tiivistys	*Message Digest*
Turvaohjaaja	*Security Manager*

Liite 2: Luokan allekirjoituksen toteuttaminen

Esimerkkisovelluksen etäluokan allekirjoitus toteutetaan jarsigner-apuohjelman avulla. Käytettävät avaimet muodostetaan keytool-apuohjelmalla. Seuraavassa on esitetty lyhyesti allekirjoituksen toteuttamisen vaiheet.

1. Muodostetaan GraduEsim.jar-niminen jar-paketti, joka sisältää valmiiksi käännetyn CheckNumber.class-luokan: `jar cvf GraduEsim.jar CheckNumbers.class`
2. Muodostetaan salainen ja julkinen avain: `keytool -genkey -alias janneFiles -keystore senderstore -keypass kpi135 -dname "cn=janne" -storepass ab987c`
3. Allekirjoitetaan GraduEsim.jar, jolloin saadaan uusi SGraduEsim.jar-niminen allekirjoitettu jar-paketti: `jarsigner -keystore senderstore -storepass ab987c -keypass kpi135 -signedjar SGraduEsim.jar GraduEsim.jar janneFiles`
4. Otetaan omasta avaintietokannasta julkisen avaimen sertifikaatti, joka toimitetaan vastaanottajalle: `keytool -export -keystore senderstore -storepass ab987c -alias janneFiles -file JanneCer.cer`
5. Vastaanottaja luo oman avaintietokannan ja sijoittaa vastaanotetun sertifikaatin sinne luotettuna sertifikaattina: `keytool -import -alias jannenluokat -file JanneCer.cer -keystore receiverstore -storepass abcdefgh`

Liite 3: JSP-sivuilla oleva käyttöoikeuksien tarkistus

Sisäänkirjautumiseen käytettävällä JSP-sivulla käyttäjälle muodostetaan istunto, jonka aikana käyttäjän tiedot ovat käytettävissä User-pavusta käsin. Istunto muodostetaan JSP-sivuilla seuraavalla ohjelmakoodilla, joka täytyy olla kaikkien JSP-sivujen alussa joilla käyttäjän tietoja tarvitaan:

```
<jsp:useBean id="user" scope="session" class="gradu.beans.User"/>
```

Kun JSP-sivulla haetaan käyttäjän tiedot sisäänkirjautumisen yhteydessä User-papuun, pysyvät ne siellä koko istunnon ajan. Jos tietokannasta ei löydy syötettyä käyttäjää syötetyllä salasanalla, sisäänkirjautumisservletti avaa sisäänkirjautumissivun uudelleen ja ilmoittaa sisäänkirjautumisen epäonnistumisesta. Jos sisäänkirjautuminen onnistuu, servletti avaa käyttäjän oikeustason mukaisen aloitussivun.

Jokaisen sivun avauksen yhteydessä tarkastetaan istunnossa olevan käyttäjän oikeustaso, joten käyttäjä ei pysty aukaisemaan sivuja, joihin hänellä ei ole oikeutta. Jos istuntoa ei ole ollenkaan eli käyttäjä ei ole kirjautunut järjestelmään, ei sivuja aukaista. Käyttäjän oikeustason tarkistus JSP-sivuilla tapahtuu yksinkertaisesti seuraavalla periaatteella:

```
<% if ("1".equals(user.getAccessRightLevel())) {
%> // Tänne toiminta mitä käyttäjä voi tehdä
<% } else { %>
<html><h2>Ei käyttöoikeuksi sivulle</h2></html>
// ja tänne ilmoitus ettei käyttöoikeuksia ole
<% } %>
```

Eli verrataan istuntossa olevan käyttäjän oikeustasoa kyseisellä sivulla tarvittavaan oikeustasoon.

Liite 4: Esimerkkisovelluksen Java-luokat ja JSP-sivut

LoginServlet-luokka

```
import gradu.beans.User;
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Sisäänkirjautumisesta huolehtiva servletti.
 *
 * @author  Janne Laitinen
 */

public class LoginServlet extends HttpServlet
{
    public void doGet( HttpServletRequest req, HttpServletResponse res )
            throws IOException, ServletException
    {
        User user = new User();
        user.setLoginName( req.getParameter( "loginname" ) );
        user.setPassword( req.getParameter( "password" ) );
        user.login();

        // Jos käyttäjätunnus ja salasana oikein...
        if ( user.getPersonID() != null )
        {
            // Laitetaan käyttäjän ID talteen istuntoon, josta se voidaan ottaa
            // käyttöön JSP-sivuilla.
            HttpSession session = req.getSession( true );
            session.setAttribute( "USERID", user.getPersonID() );

            // Jos käyttäjän oikeustaso 1, avataan pääkäyttäjän aloitussivu.
            if ( "1".equals( user.getAccessRightLevel() ) )
            {
                String address = "/jsp/graduJSP/AdminView.jsp";
                getServletConfig().getServletContext().getRequestDispatcher(
                        address ).forward( req, res );
            }
            // Muuten avataan tavallisen käyttäjän aloitussivu.
            else
            {
                String address = "/jsp/graduJSP/NormalUser.jsp";
                getServletConfig().getServletContext().getRequestDispatcher(
                        address).forward( req, res );
            }
        }
        else
        {
            // Jos käyttäjätunnus/salasana väärin, palataan login-sivulle.
            String address = "/jsp/graduJSP/LoginPage.jsp?fail=true";
            getServletConfig().getServletContext().getRequestDispatcher(
                    address ).forward( req, res );
        }
    }
}
```

UserServlet-luokka

```
import gradu.beans.User;
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Servletti joka huolehtii uuden käyttäjän rekisteröitymisestä ja normaali-
 * käyttäjän oikeudet omaavan käyttäjän tietojen päivityksen.
 *
 * @author  Janne Laitinen
 */

public class UserServlet extends HttpServlet
{
    public void doGet( HttpServletRequest req, HttpServletResponse res )
            throws IOException, ServletException
    {
        User user = new User();
        user.setAccessRightLevel( req.getParameter( "accessRightLevel" ) );
        user.setPersonID( req.getParameter( "personID" ) );
        user.setSotu( req.getParameter( "sotu" ) );
        user.setFirstName( req.getParameter( "firstName" ) );
        user.setLastName( req.getParameter( "lastName" ) );
        user.setCallName( req.getParameter( "callName" ) );
        user.setLoginName( req.getParameter( "login" ) );
        user.setStreetAddress( req.getParameter( "streetAddress" ) );
        user.setPostNumber( req.getParameter( "postNumber" ) );
        user.setPostAddress( req.getParameter( "postAddress" ) );
        user.setState( req.getParameter( "state" ) );
        user.setPhoneNumber( req.getParameter( "phoneNumber" ) );
        user.setPassword( req.getParameter( "password" ) );

        // Jos requestissa parametri "edit", ollaan tulossa käyttäjätietojen
        // muokkaussivulta -> käyttäjän tiedot päivitetään.
        if ( req.getParameter( "edit" ) != null )
        {
            user.updateUser();
            getServletConfig().getServletContext().getRequestDispatcher( "/jsp/"
                    + "graduJSP/YouUpdated.jsp" ).forward( req, res );
        }
        else // Muuten request tulee käyttäjän lisäyssivulta -> lisätään
             // käyttäjä.
        {
            user.setPostNumber( req.getParameter( "postNumber" ) );
            user.setSotu( req.getParameter( "sotu" ) );
            user.setPersonID( req.getParameter( "personID" ) );
            user.setFirstName( req.getParameter( "firstName") );
            user.setLastName( req.getParameter( "lastName" ) );
            user.setCallName( req.getParameter( "callName" ) );
            user.setLoginName( req.getParameter( "login" ) );
            user.setStreetAddress( req.getParameter( "streetAddress" ) );
            user.setPostNumber( req.getParameter( "postNumber" ) );
            user.setPostAddress( req.getParameter( "postAddress" ) );
            user.setState( req.getParameter( "state" ) );
            user.setPhoneNumber( req.getParameter( "phoneNumber" ) );
            user.setPassword( req.getParameter( "password" ) );
            user.setAccessRightLevel( req.getParameter( "accessRights" ) );
            user.setRegistered( req.getParameter( "registered" ) );
```

```
            // Jos numerotiedot oikeassa muodossa -> käyttäjä lisätään.
            if ( user.checkNumber() )
            {
                user.addUser();
                getServletConfig().getServletContext().getRequestDispatcher(
                        "/jsp/graduJSP/YouAdded.jsp" ).forward( req, res );
            }
            // Muuten palataan rekisteröintisivulle.
            else
            {
                String address = "/jsp/graduJSP/Register.jsp?failed=true";
                getServletConfig().getServletContext().getRequestDispatcher(
                        address ).forward( req, res );
            }
        }
    }
}
```

AdminServlet-luokka

```
import gradu.beans.User;
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Servletti joka huolehtii pääkäyttäjän oikeudet omaavan käyttäjän tekemistä
 * toimenpiteistä.
 *
 * @author  Janne Laitinen
 */

public class AdminServlet extends HttpServlet
{
    public void doGet( HttpServletRequest req, HttpServletResponse res )
            throws IOException, ServletException
    {
        User user = new User();
        user.setPersonID( req.getParameter( "personID" ) );
        user.setSotu( req.getParameter( "sotu" ) );
        user.setFirstName( req.getParameter( "firstName" ) );
        user.setLastName( req.getParameter( "lastName" ) );
        user.setCallName( req.getParameter( "callName" ) );
        user.setLoginName( req.getParameter( "login" ) );
        user.setStreetAddress( req.getParameter( "streetAddress" ) );
        user.setPostNumber( req.getParameter( "postNumber" ) );
        user.setPostAddress( req.getParameter( "postAddress" ) );
        user.setState( req.getParameter( "state" ) );
        user.setPhoneNumber( req.getParameter( "phoneNumber" ) );
        user.setPassword( req.getParameter( "password" ) );
        user.setAccessRightLevel( req.getParameter( "accessRights" ) );
        user.setRegistered( req.getParameter( "registered" ) );

        // Jos tullaan käyttäjän tietojen muokkaussivulta, päivitetään käyttäjän
        // tiedot tietokantaan.
        if ( req.getParameter( "edit" ) != null && req.getParameter( "remove" )
            == null )
        {
            user.updateUser();
            getServletConfig().getServletContext().getRequestDispatcher(
                    "/jsp/graduJSP/UserUpdated.jsp" ).forward( req, res );
        }
        // Jos käyttäjän tiedot on merkitty poistettavaksi, poistetaan käyttäjä
        // tietokannasta.
        else if ( req.getParameter( "remove" ) != null )
        {
            user.deleteUser();
            getServletConfig().getServletContext().getRequestDispatcher(
                    "/jsp/graduJSP/UserDeleted.jsp" ).forward( req, res );
        }
        // Jos tullaan käyttäjän lisäyssivulta ja tiedot ovat oikeassa muodossa,
        // lisätään käyttäjä tietokantaan.
        else
        {
            if ( user.checkNumber() )
            {
                user.addUser();
```

```
                getServletConfig().getServletContext().getRequestDispatcher(
                        "/jsp/graduJSP/UserAdded.jsp" ).forward( req, res );
            }
            else
            {
                String address = "/jsp/graduJSP/AddUser.jsp?failed=true";
                getServletConfig().getServletContext().getRequestDispatcher(
                        address ).forward( req, res );
            }
        }
    }
}
```

User-luokka

```
package gradu.beans;

import gradu.*;
import gradu.handlers.UserHandler;
import java.util.Vector;

/**
 * Luokka joka toimii JSP-sivujen yhteydessä toimivana papuna kapseloiden kai-
 * ken käyttäjiin liittyvän tiedon itseensä. Sisältää metodeja, joiden avulla
 * kutsutaan tarvittavia tietokannan käsittelyyn liittyviä metodeja UserHandler
 * luokasta.
 *
 * @author  Janne Laitinen
 */

public class User
{
    public User()
    {
    }

    private String personID = "";
    private String sotu = "";
    private String callName = "";
    private String firstName = "";
    private String lastName = "";
    private String loginName = "";
    private String registered = "";
    private String streetAddress = "";
    private String postNumber = "";
    private String postAddress = "";
    private String state = "";
    private String phoneNumber = "";
    private String password = "";
    private String accessRightLevel = "";

    public String getSotu()
    {
        return sotu;
    }

    public String getCallName()
    {
        return callName;
    }

    public String getPersonID()
    {
        return personID;
    }

    public String getFirstName()
    {
        return firstName;
    }

    public String getLastName()
    {
        return lastName;
```

```
}

public String getLoginName()
{
    return loginName;
}

public String getRegistered()
{
    return registered;
}

public String getStreetAddress()
{
    return streetAddress;
}

public String getPostNumber()
{
    return postNumber;
}

public String getPostAddress()
{
    return postAddress;
}

public String getState()
{
    return state;
}

public String getPhoneNumber()
{
    return phoneNumber;
}

public String getPassword()
{
    return password;
}

public String getAccessRightLevel()
{
    return accessRightLevel;
}

public void setSotu( String newValue )
{
    sotu = newValue;
}

public void setCallName( String newValue )
{
    callName = newValue;
}

public void setPersonID( String newValue )
{
    personID = newValue;
}

public void setFirstName( String newValue )
```

```
{
    firstName = newValue;
}

public void setLastName( String newValue )
{
    lastName = newValue;
}

public void setLoginName( String newValue )
{
    loginName = newValue;
}

public void setRegistered( String newValue )
{
    registered = newValue;
}

public void setStreetAddress( String newValue )
{
    streetAddress = newValue;
}

public void setPostNumber( String newValue )
{
    postNumber = newValue;
}

public void setPostAddress( String newValue )
{
    postAddress = newValue;
}

public void setState( String newValue )
{
    state = newValue;
}

public void setPhoneNumber( String newValue )
{
    phoneNumber = newValue;
}

public void setPassword( String newValue )
{
    password = newValue;
}

public void setAccessRightLevel( String newValue )
{
    accessRightLevel = newValue;
}

public Vector getUsers()
{
    UserHandler UH = new UserHandler();
    return UH.getUsers();
}

public User getUser( String personID )
{
    UserHandler UH = new UserHandler();
```

```
        return UH.getUser( personID );
    }

    public void setUserInfo( String ID )
    {
        UserHandler UH = new UserHandler();
        UH.setUserInfo( ID, this );
    }

    public void addUser()
    {
        UserHandler UH = new UserHandler();
        UH.addUser( this );
    }

    public void updateUser()
    {
        UserHandler UH = new UserHandler();
        UH.updateUser( this );
    }

    public void login()
    {
        UserHandler UH = new UserHandler();
        UH.login( this );
    }

    public boolean checkNumber()
    {
        UserHandler UH = new UserHandler();
        return UH.checkNumber( this );
    }

    public void deleteUser()
    {
        UserHandler UH = new UserHandler();
        UH.deleteUser( this );
    }
}
```

UserHandler-luokka

```
package gradu.handlers;

import gradu.*;
import gradu.beans.User;
import java.io.*;
import java.lang.reflect.*;
import java.net.*;
import java.security.*;
import java.security.cert.Certificate;
import java.sql.*;
import java.util.*;
import java.util.jar.*;

/**
 * Luokka, joka suorittaa tietokannan käsittelytoimenpiteitä ja käytetyn
 * etäluokan latauksen.
 *
 * @author  Janne Laitinen
 */

public class UserHandler extends SecureClassLoader
{
    /**
     * Rakentaja
     */
    public UserHandler()
    {
    }

    /**
     * Metodi hakee kaikki tietokannassa olevat henkilöt.
     */
    public Vector getUsers()
    {
        Connection con;
        String query = "select * from persons";
        Statement stmt;
        ResultSet rs;
        Vector result = new Vector();
        User user = null;

        try
        {
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
        }
        catch ( java.lang.ClassNotFoundException e )
        {
            System.err.println( "ClassNotFoundException: " );
            System.err.println( e.getMessage() );
        }

        // Muodostetaan tietokantayhteys.
        try
        {
            con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                    "jannel" );
            stmt = con.createStatement();
            rs = stmt.executeQuery( query );
            while ( rs.next() )
```

```
        {
            user = new User();
            user.setPersonID( rs.getString( 1 ) );
            user.setSotu( rs.getString( 2 ) );
            user.setFirstName( rs.getString( 4 ) );
            user.setLastName( rs.getString( 5 ) );
            user.setStreetAddress( rs.getString( 6 ) );
            user.setPostNumber( rs.getString( 7 ) );
            user.setPostAddress( rs.getString( 8 ) );
            result.addElement( user );
        }
        rs.close();
        stmt.close();
        con.close();
    }
    catch ( SQLException ex )
    {
        System.err.println( "SQLException: " + ex.getMessage() );
    }
    return result;
}

/**
 * Metodi hakee yksittäisen käyttäjän tietokannasta.
 */
public User getUser( String personID )
{
    Connection con;
    Statement stmt;
    ResultSet rs;
    Vector result = new Vector();
    User user = null;
    String query = "select * from persons where personID = '"
            + personID + "'";
    try
    {
        Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
    }
    catch( java.lang.ClassNotFoundException e )
    {
        System.err.print( "ClassNotFoundException: " );
        System.err.println( e.getMessage() ) ;
    }

    try
    {
        con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                "jannel");
        stmt = con.createStatement();
        rs = stmt.executeQuery( query );
        while ( rs.next() )
        {
            user = new User();
            user.setPersonID( rs.getString( 1 ) );
            user.setSotu( rs.getString( 2 ) );
            user.setCallName( rs.getString( 3 ) );
            user.setFirstName( rs.getString( 4 ) );
            user.setLastName( rs.getString( 5 ) );
            user.setLoginName( rs.getString( 6 ) );
            user.setStreetAddress( rs.getString( 7 ) );
            user.setPostNumber( rs.getString( 8 ) );
            user.setPostAddress( rs.getString( 9 ) );
            user.setState( rs.getString( 10 ) );
```

```
            user.setPhoneNumber( rs.getString( 11 ) );
            user.setPassword( rs.getString( 12 ) );
            user.setRegistered( rs.getString( 13 ) );
            user.setAccessRightLevel( rs.getString( 14 ) );
        }
        rs.close();
        stmt.close();
        con.close();
    }
    catch ( SQLException ex )
    {
        System.err.println( "SQLException: " + ex.getMessage() );
    }
    return user;
}

/**
 * Metodi hakee yksittäisen käyttäjän tiedot ja sijoittaa ne parametrina
 * tulevaan User:iin.
 */
public void setUserInfo( String personID, User user )
{
    Connection con;
    Statement stmt;
    ResultSet rs;
    Vector result = new Vector();
    String query = "select * from persons where personID = '"
            + personID + "'";
    try
    {
        Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
    }
    catch ( java.lang.ClassNotFoundException e )
    {
        System.err.print( "ClassNotFoundException: " );
        System.err.println( e.getMessage() );
    }

    try
    {
        con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                "jannel" );
        stmt = con.createStatement();
        rs = stmt.executeQuery( query );
        while ( rs.next() )
        {
            user.setPersonID( rs.getString( 1 ) );
            user.setSotu( rs.getString( 2 ) );
            user.setCallName( rs.getString( 3 ) );
            user.setFirstName( rs.getString( 4 ) );
            user.setLastName( rs.getString( 5 ) );
            user.setLoginName( rs.getString( 6 ) );
            user.setStreetAddress( rs.getString( 7 ) );
            user.setPostNumber( rs.getString( 8 ) );
            user.setPostAddress( rs.getString( 9 ) );
            user.setState( rs.getString( 10 ) );
            user.setPhoneNumber( rs.getString( 11 ) );
            user.setPassword( rs.getString( 12 ) );
            user.setRegistered( rs.getString( 13 ) );
            user.setAccessRightLevel( rs.getString( 14 ) );
        }
        rs.close();
        stmt.close();
```

```
            con.close();
        }
        catch( SQLException ex )
        {
            System.err.println( "SQLException: " + ex.getMessage() );
        }
    }

    /**
     * Metodi lisää yksittäisen käyttäjän tietokantaan.
     */
    public void addUser( User user )
    {
        Connection con;
        Statement stmt;
        ResultSet rs;
        String count = null;
        String query = "select personcount from count where id = '1'";
        System.out.println( "Query: " + query );
        try
        {
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
        }
        catch( java.lang.ClassNotFoundException e )
        {
            System.err.print( "ClassNotFoundException: " );
            System.err.println( e.getMessage() );
        }

        try
        {
            con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                    "jannel" );
            stmt = con.createStatement();
            rs = stmt.executeQuery( query );
            if ( rs.next() )
            {
                count = Integer.toString( Integer.parseInt
                        ( rs.getString( 1 ) ) + 1 );
            }
            if ( count != null )
            {
                query = "insert into persons values ('" + count + "', '"
                + user.getSotu() + "', '" + user.getCallName() + "', '"
                + user.getFirstName() + "', '" + user.getLastName() + "', '"
                + user.getLoginName() + "', '" + user.getStreetAddress() + "','"
                + user.getPostNumber() + "', '" + user.getPostAddress() + "', '"
                + user.getState()+ "', '" + user.getPhoneNumber() + "', '"
                + user.getPassword() + "', '" + user.getRegistered() + "', '"
                + user.getAccessRightLevel() + "')" ;

                stmt.executeUpdate( query );

                query = "update count set personcount = '"+count+"' "
                        + "where id = '"+1+"'";
                stmt.executeUpdate( query );
            }
            stmt.close();
            con.close();
        }
        catch ( SQLException ex )
        {
            System.err.println( "SQLException: " + ex.getMessage() );
```

```
    }

}

/**
 * Metodi päivittää yksittäisen käyttäjän tiedot tietokantaan.
 */
public void updateUser( User user )
{
    Connection con;
    Statement stmt;
    ResultSet rs;
    String query = null;
    try
    {
        Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
    }
    catch( java.lang.ClassNotFoundException e )
    {
        System.err.print( "ClassNotFoundException: " );
        System.err.println( e.getMessage() );
    }

    try
    {
        con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                "jannel" );
        stmt = con.createStatement();
        query = "update persons set sotu = '" + user.getSotu()
                + "', CallName = '" + user.getCallName()
                + "', FirstName = '" + user.getFirstName()
                + "', LastName = '" + user.getLastName()
                + "', LoginName = '" + user.getLoginName()
                + "', StreetAddress = '" + user.getStreetAddress()
                + "', PostNumber = '" + user.getPostNumber()
                + "', PostAddress = '" + user.getPostAddress()
                + "', State = '" + user.getState()
                + "', PhoneNumber = '" + user.getPhoneNumber()
                + "', Password = '" + user.getPassword()
                + "', AccessRightLevel = '" +user.getAccessRightLevel()
                + "' where PersonID = '" + user.getPersonID() + "'";

        stmt.executeUpdate( query );
        stmt.close();
        con.close();
    }
    catch ( SQLException ex )
    {
        System.err.println( "SQLException: " + ex.getMessage() );
    }
}

/**
 * Metodi poistaa yksittäisen käyttäjän tietokannasta.
 */
public void deleteUser( User user )
{
    Connection con;
    Statement stmt;
    ResultSet rs;
    String query = null;
    try
    {
```

```
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
        }
        catch( java.lang.ClassNotFoundException e )
        {
            System.err.print( "ClassNotFoundException: " );
            System.err.println( e.getMessage() );
        }

        try
        {
            con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                    "jannel" );
            stmt = con.createStatement();
            query = "delete from persons where PersonID = '"
                    + user.getPersonID() + "'";
            stmt.executeUpdate( query );
            stmt.close();
            con.close();
        }
        catch ( SQLException ex )
        {
            System.err.println( "SQLException: " + ex.getMessage() );
        }
    }

    /**
     * Metodi tarkastaa käyttäjätunnuksen ja salasanan tietokannasta ja hakee
     * käyttäjän tiedot parametrina olevaan User:iin, jos käyttäjä ja salasana
     * ovat oikeita. Jos ei löydy, asetetaan personID arvoon null.
     */
    public void login( User user )
    {
        Connection con;
        Statement stmt;
        ResultSet rs;
        Vector result = new Vector();
        String query = "select * from persons where loginname = '"
                + user.getLoginName() + "' and password = '"
                + user.getPassword() + "'";
        try
        {
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
        }
        catch ( java.lang.ClassNotFoundException e )
        {
            System.err.print( "ClassNotFoundException: " );
            System.err.println( e.getMessage() );
        }

        try
        {
            con = DriverManager.getConnection( "jdbc:odbc:graduEsim", "jannel",
                    "jannel" );
            stmt = con.createStatement();
            rs = stmt.executeQuery( query );
            if ( rs.next() )
            {
                user.setPersonID( rs.getString( 1 ) );
                user.setSotu( rs.getString( 2 ) );
                user.setCallName( rs.getString( 3 ) );
                user.setFirstName( rs.getString( 4 ) );
                user.setLastName( rs.getString( 5 ) );
                user.setLoginName( rs.getString( 6 ) );
```

```
            user.setStreetAddress( rs.getString( 7 ) );
            user.setPostNumber( rs.getString( 8 ) );
            user.setPostAddress( rs.getString( 9 ) );
            user.setState( rs.getString( 10 ) );
            user.setPhoneNumber( rs.getString( 11 ) );
            user.setPassword( rs.getString( 12 ) );
            user.setRegistered( rs.getString( 13) );
            user.setAccessRightLevel( rs.getString( 14 ) );
        }
        else
        {
            user.setPersonID( null );
        }
        rs.close();
        stmt.close();
        con.close();
    }
    catch ( SQLException ex )
    {
        System.err.println( "SQLException: " + ex.getMessage() );
    }
}

/**
 * Käyttäjän postinumeron ja henkilötunnuksen muodon tarkastava metodi.
 * Toiminta tapahtuu lataamalla JAR-paketissa erillisellä palvelimella
 * oleva luokka. Kyseinen luokka sisältää varsinaiset tarkistusmetodit,
 * joita käytetään numeroiden tarkastamiseen.
 */
public boolean checkNumber( User user )
{
    boolean returnvalue = true;
    byte buf[] = null;
    Class cl = null;
    Class cl2 = null;
    CodeSource cs = null;

    readJarFile( "SGraduEsim.jar" );

    buf = ( byte[] ) classArrays.get( "CheckNumbers" );
    try
    {
        if ( buf != null )
        {
            Certificate certs[] = ( Certificate[] )
                    classCerts.get( "CheckNumbers" );
            cs = new CodeSource( new URL( "http://jannela:8080/examples/"
                    + "jsp/graduJSP/remoteClass/" ), certs );
            cl = defineClass( "CheckNumbers", buf, 0, buf.length, cs );

        }
    }
    catch ( Exception e )
    {
        e.printStackTrace();
    }

    buf = ( byte[] ) classArrays.get( "CheckNumbers$1" );
    try
    {
        if ( buf != null )
        {
            Certificate certs[] = ( Certificate[] )
```

```
                    classCerts.get( "CheckNumbers$1" );
                cs = new CodeSource( new URL( "http://jannela:8080/examples/"
                        + "jsp/graduJSP/remoteClass/" ), certs );
                cl2 = defineClass( "CheckNumbers$1", buf, 0, buf.length, cs );

            }
        }
        catch ( Exception e )
        {
            e.printStackTrace();
        }

        try
        {
            Class argList[] = new Class[]{ String.class };
            Method postnumberTest = cl.getMethod( "testPostNumber", argList );
            Object args[] = new Object[1];
            args[0] = user.getPostNumber();
            Boolean isLegal = ( Boolean ) postnumberTest.invoke(
                    cl.newInstance(), args );

            // Jos postinumero ei ole oikeassa muodossa, käyttäjää ei lisätä.
            if ( !isLegal.booleanValue() )
            {
                returnvalue = false;
            }

            Method sotuTest = cl.getMethod( "testSotu", argList );
            args[0] = user.getSotu();
            System.out.println("SOTU: " + user.getSotu());
            isLegal = ( Boolean ) sotuTest.invoke( cl.newInstance(), args );

            // Jos sotu ei ole oikeassa muodossa, käyttäjää ei lisätä.
            if ( !isLegal.booleanValue() )
            {
                returnvalue = false;
            }

            // Jos luokkaa ei ole allekirjoitettu ei siihen voi luottaa ja siten
            // käyttäjää ei lisätä.
            if ( cl.getSigners() == null )
            {
                returnvalue = false;
            }
        }
        catch ( Exception e )
        {
            returnvalue = false;
        }
        return returnvalue;
    }

    /**
     * Jar-tiedoston etäkoneelta lataava metodi.
     */
    private void readJarFile( String name )
    {
        JarInputStream jis;
        JarEntry je;
        URL jarUrl = null;

        try
        {
```

```
            jarUrl = new URL( "http://jannela:8080/examples/jsp/graduJSP/"
                    + "remoteClass/" + name );
        }
        catch ( MalformedURLException mue )
        {
            System.out.println( "Tunnistamaton jar-tiedosto: " + name );
            return;
        }

        try
        {
            jis = new JarInputStream( jarUrl.openConnection().
                    getInputStream() );
        }
        catch ( IOException ioe )
        {
            System.out.println( "Jar-tiedostoa " + jarUrl + " ei voi avata" );
            return;
        }

        try
        {
            while ( ( je = jis.getNextJarEntry() ) != null )
            {
                String jarName = je.getName();
                System.out.println( "Jarname: " + jarName );
                if ( jarName.endsWith( ".class" ) )
                {
                    loadClassBytes( jis, jarName, je );
                }
                jis.closeEntry();
            }
        }
        catch ( IOException ioe )
        {
            System.out.println( "Jar-tiedosto ei ole oikeassa muodossa" );
        }
    }

    /**
     * JAR-tiedostossa olevan luokan lukeva metodi.
     */
    private void loadClassBytes( JarInputStream jis,
                                 String jarName,
                                 JarEntry je )
    {
        System.out.println( "\t" + jarName );
        BufferedInputStream jarBuf = new BufferedInputStream( jis );
        ByteArrayOutputStream jarOut = new ByteArrayOutputStream();
        String className = null;
        int b;
        try
        {
            while ( ( b = jarBuf.read() ) != -1 )
            {
                jarOut.write( b );
            }
            className = jarName.substring( 0, jarName.length() - 6 );
            classArrays.put( className, jarOut.toByteArray() );
            System.out.println( "Luokan nimi hashtablessa: " + className );
            System.out.println( "ClassName: " + className );
            Object certs[] = je.getCertificates();
            if ( certs != null )
```

```
            {
                classCerts.put( className, certs );
            }
        }
        catch ( IOException iot )
        {
            System.out.println( "Virhe tiedoston luvussa " + jarName );
        }
    }

    protected Class findLocalClass( String name )
    {
        return ( Class )classArrays.get( name );

    }

    // Hashtable, johon sijoitetaan JAR-paketista luetut luokat.
    private Hashtable classArrays = new Hashtable();

    // Hashtable, johon sijoitetaan JAR-paketista luetut sertifikaatit.
    private Hashtable classCerts = new Hashtable();
}
```

CheckNumbers-luokka

```
import java.io.*;
import java.security.*;

/**
 * Luokka jolla suoritetaan postinumeron ja henkilötunnuksen muodon tarkastus.
 * Tämä luokka ladataan sovelluksen suorituksen aikanaa etäkoneelta allekirjoi-
 * tettuna Jar-paketissa.
 *
 * @author  Janne Laitinen
 */

public class CheckNumbers
{
    public CheckNumbers()
    {
    }

    /**
     * Metodi joka testaa parametrina tulevan postinumeron pituuden (5 merkkiä).
     * (Tarkistuksen menetelmää olisi hyvää kehittää kattavammaksi.)
     */
    public boolean testPostNumber( String number )
    {
        boolean returnvalue = true;

        if ( number != null )
        {
            if ( number.length() != 5 )
            {
                returnvalue = false;
            }
        }
        else
        {
            returnvalue = false;
        }

        if ( returnvalue )
        {
            writeLog( number + " kelvollinen" );
        }
        else
        {
            writeLog( number + " epäkelvollinen" );
        }

        return returnvalue;
    }

    /**
     * Metodi joka tarkastaa, että parametrina tulleen henkilötunnuksen muoto on
     * oikea päivän, kuukauden ja vuoden osalta.
     */
    public boolean testSotu( String sotu )
    {
        boolean returnvalue = true;

        if ( sotu != null )
```

```
        {
            String day = sotu.substring( 0, 2 );
            String month = sotu.substring( 2, 4 );
            String year = sotu.substring( 4, 6 );
            System.out.println("Day: " + day + ", month: " + month + ", year: "
                    + year);

            if ( Integer.parseInt( day ) > 31 || Integer.parseInt( day ) == 0 )
            {
                returnvalue = false;
            }
            if ( Integer.parseInt( month )> 12 ||
                    Integer.parseInt( month ) == 0 )
            {
                returnvalue = false;
            }
        }
        else
        {
            returnvalue = false;
        }

        if ( returnvalue )
        {
            writeLog( sotu + " kelvollinen" );
        }
        else
        {
            writeLog( sotu + " epäkelvollinen" );
        }

        return returnvalue;
    }

    /**
     * Metodi kirjoittaa lokiin parametrina tulevan merkkijonon.
     */
    private void writeLog( String log )
    {
        try
        {
            FileWriter fos = new FileWriter( "log.txt" );
            fos.write( log, 0, log.length());
            fos.close();
            System.out.println( "Lokin kirjoitus onnistui : " + "log.txt" );
        }
        catch ( Exception e )
        {
            System.out.println( "Lokin kirjoitus epäonnistui : " + "log.txt" );
            e.printStackTrace();
        }
    }
}
```

MainPage.jsp (Aloitussivu)

```
<html>
<h1 align="center">MainPage</h1>
<p align="center"><a href="/examples/jsp/graduJSP/Register.jsp">
            Rekisteröidy</a></p>
<p align="center"><a href="/examples/jsp/graduJSP/LoginPage.jsp">
            Login-sivulle</a></p>
</html>
```

LoginPage.jsp (Sisäänkirjautuminen)

```
<html>
<h1 align="center">Sisäänkirjautuminen</h1>
<form action="/examples/servlet/LoginServlet">
  <% if ( request.getParameter( "fail" ) != null ) {%>
    <p>Tunnistus epäonnistui, yritä uudelleen.</p>
  <% } %>
  <table align="center">
    <tr><td>Login:</td><td><input type="text" name="loginname"></td></tr>
    <tr><td>Password:</td><td><input type="passwd" name="password"></td></tr>
    <tr><td><input type="submit" name="add" value="Login"></td><td>
      <input type="reset" value="Tyhjennä kentät"></td></tr>
    <tr><td><a href="/examples/jsp/graduJSP/MainPage.jsp">Takaisin pääsivulle
      </a></td></tr>
  </table>
</form>
</html>
```

Register.jsp (Käyttäjäksi rekisteröityminen)

```
<html>
<h1 align="center">Käyttäjäksi rekisteröityminen</h1>
<% if ( request.getParameter( "failed" ) != null) { %>
  <h2>Syöttämäsi tiedot eivät ole kelvollisia!</h2>
<% } %>

<form action="/examples/servlet/UserServlet" >
  <table align="center">
    <tr><td>Sotu:</td><td><input type="text" name="sotu"></td></tr>
    <tr><td>Etunimi:</td><td><input type="text" name="firstName"></td></tr>
    <tr><td>Sukunimi:</td><td><input type="text" name="lastName"></td></tr>
    <tr><td>Kutsumanimi:</td><td><input type="text" name="callName"></td></tr>
    <tr><td>Login:</td><td><input type="text" name="login"></td></tr>
    <tr><td>Lähiosoite:</td><td><input type="text"
      name="streetAddress"></td></tr>
    <tr><td>Postinumero:</td><td><input type="text" name="postNumber"></td></tr>
    <tr><td>Postiosoite:</td><td><input type="text"
      name="postAddress"></td></tr>
    <tr><td>Valtio:</td><td><input type="text" name="state"></td></tr>
    <tr><td>Puhelinnumero:</td><td><input type="text"
      name="phoneNumber"></td></tr>
    <tr><td>Password:</td><td><input type="text" name="password"></td></tr>
    <tr><td>Oikeustaso:</td><td><input type="text"
      name="accessRights"></td></tr>
    <tr><td>Rekisteröity:</td><td><input type="text" name="registered"</td></tr>
    <tr><td><input type="submit" name="add" value="Rekisteröidy"></td><td>
```

```
      <input type="reset" value="Tyhjennä kentät"></td></tr>
    <tr><td><a href="MainPage.jsp">Takaisin pääsivulle</a></td></tr>
  </table>
</form>
</html>
```

YouAdded.jsp (Käyttäjäksi rekisteröityminen onnistui)

```
<html>
<h1 align="center">Tiedot lisätty</h1>
<p align="center">Tietosi on lisätty, voit nyt kirjautua sisään järjestelmään
  antamallasi salasanalla</p>
<p align="center"><a href="/examples/jsp/graduJSP/LoginPage.jsp">
  Login-sivulle</a></p>
</html>
```

Adminview.jsp (Pääkäyttäjän pääsivu)

```
<%@ page import = "gradu.beans.User" %>
<%@ page import = "java.util.Vector" %>

<jsp:useBean id="user" scope="session" class="gradu.beans.User" />

<html>
<%
  user.setUserInfo( ( String )request.getSession().getAttribute( "USERID" ) );
  if ( "1".equals( user.getAccessRightLevel() ) ) {
%>
<h1 align="center">Pääkäyttäjän sivu</h1>
<p align="center">Valitse henkilö, jonka tietoja haluat muokata tai katsoa.
  Voit myös lisätä uusita henkilöitä ja poistaa olemassaolevia.</p>
<form>
  <table align="center">
  <%
    User user2=new User();
    Vector userVector=user.getUsers();
    if (userVector != null)
    {
      for ( int i = 0; i < userVector.size(); i++)
      {
        user2 = ( User )userVector.elementAt( i );
        %>
        <tr><td>Sotu:</td><td><input type="text" name="sotu"
          value="<%=user2.getSotu()%>"></td>
        <td>Etunimi:</td><td><input type="text" name="nimi"
          value="<%=user2.getFirstName()%>"></td>
        <td>Sukunimi:</td><td><input type="text name="sukunimi"
          value="<%=user2.getLastName()%>"></td>
        <td>
          <a href="/examples/jsp/graduJSP/EditUsers.jsp?id=<%=user2.getPersonID()
          %>">Muokkaa tietoja</a></td>
          </tr>
      <%}
      }
    %>
  </table>
</form>
<a href="/examples/jsp/graduJSP/AddUser.jsp">Lisää käyttäjä</a>
</html>

<% } else { %>
  <h1> Sinulla ei ole oikeuksia käyttää tätä sivua! </h1>
  </html>
<%}%>
```

NormalUser.jsp (Tavallisen käyttäjän pääsivu)

```
<%@ page import = "gradu.beans.User" %>
<jsp:useBean id="user" scope="session" class="gradu.beans.User" />

<html>
<%
  String userID = request.getParameter("id");
  user.setUserInfo( ( String )request.getSession().getAttribute("USERID") );
  if ( "2".equals( user.getAccessRightLevel() ) ) {
  %>

  <h1 align="center">Tervetuloa <%=user.getFirstName()%>!</h1>
  <p align="center">Voit muokata tietojasi alla olevasta linkistä</p>
  <p align="center"><a href="/examples/jsp/graduJSP/EditUser.jsp">
    Tietojen muokkaussivulle</a></p>
</html>

<% } else { %>
  <h1> Sinulla ei ole oikeuksia käyttää tätä sivua! </h1>
  </html>
<%}%>
```

EditUsers.jsp (Pääkäyttäjän muokkaussivu yksittäisille käyttäjille)

```
<%@ page import = "gradu.beans.User" %>

<jsp:useBean id="user" scope="session" class="gradu.beans.User" />

<html>
<%
if ( "1".equals( user.getAccessRightLevel() ) ) {
%>
  <h1 align="center">Käyttäjän tietojen muokkaus</h1>
  <p align="center">Voi muokata alla olevan henkilön tietoja.</p>
  <form action="/examples/servlet/AdminServlet" method="get">
  <input type=hidden name="edit" value="true">
  <input type=hidden name="personID" value="<%=request.getParameter( "id" )%>">
  <%User user2 = new User();
  String userID = request.getParameter( "id" );
  if ( userID != null) {
    user2=user2.getUser( userID );
    if ( user2 != null ) { %>
      <table align="center">
        <tr><td>Sotu:</td><td><input type="text" name="sotu"
          value="<%=user2.getSotu()%>"></td></tr>
        <tr><td>Etunimi:</td><td><input type="text" name="firstName"
          value="<%=user2.getFirstName()%>"></td></tr>
        <tr><td>Sukunimi:</td><td><input type="text" name="lastName"
          value="<%=user2.getLastName()%>"></td></tr>
        <tr><td>Kutsumanimi:</td><td><input type="text" name="callName"
          value="<%=user2.getCallName()%>"</td></tr>
        <tr><td>Login:</td><td><input type="text" name="login"
          value="<%=user2.getLoginName()%>"</td></tr>
        <tr><td>Lähiosoite:</td><td><input type="text" name="streetAddress"
          value="<%=user2.getStreetAddress()%>"</td></tr>
        <tr><td>Postinumero:</td><td><input type="text" name="postNumber"
          value="<%=user2.getPostNumber()%>"</td></tr>
        <tr><td>Postiosoite:</td><td><input type="text" name="postAddress"
```

```
          value="<%=user2.getPostAddress()%>"</td></tr>
        <tr><td>Valtio:</td><td><input type="text" name="state"
          value="<%=user2.getState()%>"</td></tr>
        <tr><td>Puhelinnumero:</td><td><input type="text" name="phoneNumber"
          value="<%=user2.getPhoneNumber()%>"</td></tr>
        <tr><td>Password:</td><td><input type="text" name="password"
          value="<%=user2.getPassword()%>"</td></tr>
        <tr><td>Oikeustaso:</td><td><input type="text" name="accessRights"
          value="<%=user2.getAccessRightLevel()%>"</td></tr>
        <tr><td>Rekisteröity:</td><td><%=user.getRegistered()%></td></tr>
        <tr><td><input type="submit" name="add" value="Päivitä tiedot"></td>
        <td><input type="reset" value="Tyhjennä kentät"></td></tr>
        <tr><td>Poista käyttäjä</td><td><input type="checkbox"
             name="remove"</td></tr>
        <tr><td><a href="AdminView.jsp">Takaisin pääkäyttäjän sivulle</a>
          </td></tr>
      </table>
    <%}
  }
%>
</form>
</html>

<% } else { %>
  <h1> Sinulla ei ole oikeuksia käyttää tätä sivua! </h1>
  </html>
<%}%>
```

AddUser.jsp (Käyttäjien lisäyssivu (pääkäyttäjän käytössä))

```
<h1 align="center">Lisää käyttäjä</h1>
<html>
<% if (request.getParameter( "failed" ) != null ) { %>
  <h2>Syöttämäsi tiedot eivät ole kelvollisia!</h2>
<% } %>

<form action="/examples/servlet/AdminServlet">
  <table align="center">
    <tr><td>Sotu:</td><td><input type="text" name="sotu"></td></tr>
    <tr><td>Etunimi:</td><td><input type="text" name="firstName"></td></tr>
    <tr><td>Sukunimi:</td><td><input type="text" name="lastName"></td></tr>
    <tr><td>Kutsumanimi:</td><td><input type="text" name="callName"></td></tr>
    <tr><td>Login:</td><td><input type="text" name="login"></td></tr>
    <tr><td>Lähiosoite:</td><td><input type="text"
      name="streetAddress"></td></tr>
    <tr><td>Postinumero:</td><td><input type="text" name="postNumber"></td></tr>
    <tr><td>Postiosoite:</td><td><input type="text"
      name="postAddress"></td></tr>
    <tr><td>Valtio:</td><td><input type="text" name="state"></td></tr>
    <tr><td>Puhelinnumero:</td><td>
      <input type="text" name="phoneNumber"></td></tr>
    <tr><td>Password:</td><td><input type="text" name="password"></td></tr>
    <tr><td>Oikeustaso:</td><td><input type="text"
      name="accessRights"></td></tr>
    <tr><td>Rekisteröity:</td><td><input type="text" name="registered"</td></tr>
    <tr><td><a href="AdminView.jsp">Takaisin pääkäyttäjän sivulle</a></td></tr>
    <tr><td><input type="submit" name="add" value="Lisää tiedot"></td><td>
      <input type="reset" value="Tyhjennä kentät"></td></tr>
  </table>
</form>
</html>
```

UserAdded.jsp (Käyttäjä lisätty)

```
<html>
<h1 align="center">UserAdded</h1>
<p align="center">Käyttäjän tiedot on päivitetty</p>
<p align="center"><a href="/examples/jsp/graduJSP/AdminView.jsp">
  Takaisin AdminViewiin</a></p>
</html>
```

UserUpdated.jsp (Käyttäjän tiedot päivitetty)

```
<html>
<h1 align="center">Käyttäjän tiedot on päivitetty</h1>
<p align="center"><a href="/examples/jsp/graduJSP/AdminView.jsp">
  Takaisin pääkäyttäjän sivulle</a></p>
</html>
```

UserDeleted.jsp (Käyttäjä poistettu)

```
<html>
<h1 align="center">Käyttäjän poistettu</h1>
<p align="center"><a href="/examples/jsp/graduJSP/AdminView.jsp">
  Takaisin pääkäyttäjän sivulle</a></p>
</html>
```

EditUser.jsp (Käyttäjän omien tietojen muokkaus (tavallisen käyttäjän käytössä))

```
<%@ page import = "gradu.beans.User" %>
<jsp:useBean id="user" scope="session" class="gradu.beans.User" />

<html>
<h1 align="center">Tietojen muokkaus</h1>
<p align="center">Voi muokata tietojasi tällä sivulla</p>
<form action="/examples/servlet/UserServlet">
  <input type=hidden name="edit" value="true">
  <input type=hidden name="normaluser" value="true">
  <input type=hidden name="personID" value="<%=user.getPersonID()%>">
  <input type=hidden name="accessRightLevel"
    value="<%=user.getAccessRightLevel()%>">
  <table align="center">
    <tr><td>Sotu:</td><td><input type="text" name="sotu"
      value="<%=user.getSotu()%>"></td></tr>
    <tr><td>Etunimi:</td><td><input type="text" name="firstName"
      value="<%=user.getFirstName()%>"></td></tr>
    <tr><td>Sukunimi:</td><td><input type="text" name="lastName"
      value="<%=user.getLastName()%>"></td></tr>
    <tr><td>Kutsumanimi:</td><td><input type="text" name="callName"
      value="<%=user.getCallName()%>"</td></tr>
    <tr><td>Login:</td><td><input type="text" name="login"
      value="<%=user.getLoginName()%>"</td></tr>
    <tr><td>Lähiosoite:</td><td><input type="text"
      name="streetAddress" value="<%=user.getStreetAddress()%>"</td></tr>
    <tr><td>Postinumero:</td><td><input type="text"
      name="postNumber" value="<%=user.getPostNumber()%>"</td></tr>
    <tr><td>Postiosoite:</td><td><input type="text"
      name="postAddress" value="<%=user.getPostAddress()%>"</td></tr>
    <tr><td>Valtio:</td><td><input type="text"
      name="state" value="<%=user.getState()%>"</td></tr>
    <tr><td>Puhelinnumero:</td><td><input type="text"
      name="phoneNumber" value="<%=user.getPhoneNumber()%>"</td></tr>
    <tr><td>Password:</td><td><input type="text" name="password"
      value="<%=user.getPassword()%>"</td></tr>
    <tr><td>Oikeustaso:</td><td><%=user.getAccessRightLevel()%></td></tr>
    <tr><td>Rekisteröity:</td><td><%=user.getRegistered()%></td></tr>
    <tr><td><input type="submit" name="add" value="Päivitä tiedot"></td><td>
      <input type="reset" value="Tyhjennä kentät"></td></tr>
    <tr><td><a href="AdminView.jsp">Takaisin pääsivulle</a></td></tr>
  </table>
</form>
</html>
```

YouUpdated.jsp (Käyttäjän tiedot päivitetty)

```
<html>
  <h1 align="center">Tietosi on päivitetty</h1>
  <p align="center">
    <a href="/examples/jsp/graduJSP/NormalUser.jsp">Pääsivulle</a></p>
</html>
```