HAAGA-HELIA
ammattikorkeakoulu

Valmisohjelmiston sovellusturvallisuuden arviointi OWASP-metodologian avulla

Eino Liimatta

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2013

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

17.9.2013

Tekijä tai tekijät Eino Liimatta	**Ryhmätunnus tai aloitusvuosi** 2011
Raportin nimi Valmisohjelmiston sovellusturvallisuuden arviointi OWASP-metodologian avulla	**Sivu- ja liitesivumäärä** 46 + 13
Opettajat tai ohjaajat Titta Ahlberg	

Suurin osa yrityksistä ja julkisista organisaatioista hankkii käyttämänsä sovellukset räätälöitävinä valmisohjelmistoina, jolloin niiden sovellusturvallisuuden arviointi on vaikeaa. Tämän opinnäytetyön tarkoituksena on selvittää valmisohjelmistojen sovellusturvallisuuden arvioinnin mahdollisuuksia ja rajoitteita käyttäen Open Web Application Security Project (OWASP) -säätiön tuottamaa, julkista sovellusturvallisuuden arvioimisen ja testaamisen metodologiaa sekä avoimen lähdekoodin verkkoturvallisuustyökaluja.

Opinnäytetyön tavoitteena oli muodostaa kriittinen kokonaiskuva OWASP -metodologian soveltamisesta käytännössä valmisohjelmistojen sovellusturvallisuuden arviointiin ja testaamiseen. OWASP-metodologiaa tutkittiin sekä teoreettisesti että käytännön tapaustutkimuksena IBM InfoSphere MDM Collaboration Server -ohjelmistotuotteen sovellusturvallisuuden arvioimiseksi.

OWASP-metodologia rajattiin tutkimuksessa dokumentteihin OWASP Testing Guide 2008 v3.0 ja OWASP Application Security Verification Standard 2009. Tapaustutkimus suoritettiin suljetussa virtuaaliympäristössä ja sovellusturvallisuuden testaamiseen käytettiin Kali Linux -distribuutiota ja sen sisältämiä avoimen lähdekoodin verkkoturvallisuusohjelmistoja. Testaaminen suoritettiin mustalaatikko-menetelmää käyttäen etsimällä haavoittuvuuksia, joita voidaan havaita ja joita voidaan hyödyntää järjestelmän ulkopuolelta.

Tutkimuksen tulosten perusteella OWASP-metodologia antaa hyvän teknisen perustan sovellusturvallisuuden arvioinnille. OWASP-metodologian soveltuvuus valmisohjelmistojen arviointiin ei osoittautunut täysin kattavaksi, koska OWASP-dokumentaation laatu on vaihtelevaa eri sovellusturvallisuuden osa-alueiden kohdalla. OWASP -metodologiassa on puutteita riskienhallinnan osalta. OWASP-metodologian heikkouksia kuitenkin kompensoi laajempien organisaation tietoturvan hallinnan viitekehysten käyttäminen sen rinnalla. Valmisohjelmiston kohdalla korostuu myös tietojärjestelmässä käsiteltävän informaation organisaatiolle tulevan substanssiarvon tunnistaminen, joka auttaa tietoturvan hallinnan kohdistamisessa kriittisiin osa-alueisiin.

Asiasanat
ohjelmistoala, tietoturva, avoin lähdekoodi

HAAGA-HELIA
University of Applied Sciences

Abstract

17th September 2013

Degree Programme in Information Technology

Authors Eino Liimatta	**Group or year of entry** 2011
The title of thesis Assessing the application security of commercial off-the-self software using the OWASP methodology	**Number of report pages and attachment pages** 46 + 13
Advisor(s) Titta Ahlberg	

Many companies and public organizations acquire their software applications as customizable commercial off-the-self software (COTS), which makes the application security assessment difficult. The purpose of this thesis was to investigate possibilities and challenges of COTS application security assessment using the publicly available methodology of the Open Web Application Security Project (OWASP) Foundation and open source network security software tools.

The main goal of the thesis was to produce critical overview of applying the OWASP methodology to assess and test the application security of the COTS software in practice. The OWASP methodology was investigated theoretically and also as practical case study of a software product, IBM InfoSphere MDM Collaboration Server.

The OWASP methodology used in this thesis was confined to two online documents, OWASP Testing Guide 2008 v3.0 and OWASP Application Security Verification Standard 2009. The case study was conducted in a closed virtualization environment. Application security testing was done using open source network security tools included in the Kali Linux distribution. Black box testing methodology was used to find possible vulnerabilities outside of the target system.

Based on the results of this thesis, the OWASP methodology seems to provide a solid technical framework for application security assessment in general. Nevertheless the OWASP methodology was found to be slightly inadequate for COTS software assessment, because the variability of coverage on different sectors of the application security. The OWASP methodology description of risk management is particularly insufficient. The use of a broader organizational information security framework in parallel with the OWASP methodology would compensate its shortcomings. In case of the COTS software, identifying the asset value of information processed in the information system allows to focus on the most critical areas of information security management.

Key words
software industry, information security, open source

Sisällys

1 Johdanto

Yhä useammat yritykset ja organisaatiot käyttävät websovelluksia toimintansa tukemiseen ja vuorovaikutukseen erilaisten sidosryhmiensä kanssa. Websovellusten mahdollistama uudistunut liiketoiminnan malli asettaa entistä suurempia vaatimuksia sovellusturvallisuudelle, koska websovelluksia käytetään julkisen Internet-verkon kautta. Suurin osa yrityksistä hankkii tarvitsemansa sovellukset räätälöitävinä valmisohjelmistoina, jolloin niiden sovellusturvallisuuden arviointi on varsin vaikeaa. Websovellusten tietoturvan haasteisiin alettiin havahtua 1990-luvulla, kun Internetin käyttö kasvoi ja organisaatiot alkoivat hyödyntää websovelluksia oman toimintansa tukemiseksi. Tietoturvatutkijat kiinnittivät huomiota verkkoliikenteen de facto -standardiksi muodostuneen TCP/IP-protokollapinon ja tietokoneohjelmistojen yleisiin haavoittuvuuksiin. Sovellusturvallisuuden tutkimuksen pioneerityötä tehtiin Yhdysvalloissa USA:n liittovaltion NIST-viraston informaatioteknologian laboratoriossa sekä vuonna 1989 perustetussa yksityisessä SANS-instituutissa.

Internetin käytön leviäminen ympäri maailman on luonut uudenlaisia, informaatiotekniikkaan perustuvia yhteisöllisyyden muotoja. Tietoturva-alalla muutos on näkynyt tutkimuksen ja koulutuksen painopisteen siirtymisessä perinteisten akateemisten organisaatioiden ja valtiollisien turvallisuusviranomaisten piiristä kohti kansainvälisiä, avoimia verkkoyhteisöjä, jotka yhdistävät alan ammattilaisia, opiskelijoita ja muita asianharrastajia ennen näkemättömällä tavalla. Sovellusturvallisuuden tutkimuksen parissa uuden verkkoyhteisöihin perustuvan organisaatiomallin näkyvimpiä edustajia on ollut 2000-luvun alussa Yhdysvalloissa perustettu Open Web Application Security Project (OWASP), joka on keskittynyt sovellusturvallisuuden edistämiseen ohjelmistokehityksessä. OWASP:n lukuisat yhteisöprojektit ovat 12 vuoden aikana tuottaneet runsaasti vapaasti saatavilla olevia artikkeleita, metodologisia oppaita, työkaluja ja teknologiaa aiheeseen liittyen.

Tämän opinnäytetyön tarkoituksena on selvittää valmisohjelmistojen sovellusturvallisuuden arvioinnin mahdollisuuksia ja rajoitteita käyttäen Open Web Application Security Project (OWASP) -säätiön tuottamaa, julkista sovellusturvallisuuden arvioimisen ja testaamisen metodologiaa sekä avoimen lähdekoodin verkkoturvallisuustyökaluja.

Opinnäytetyö rakentuu teoreettisesta osuudesta, jossa ensiksi tutustutaan tarkemmin OWASP-metodologian sisältöön, sekä käytännön tapaustutkimuksesta, jossa OWASP-metodologiaa pyritään soveltamaan laajan, Java-teknologiaan pohjautuvan websovelluksen sovellusturvallisuuden arviointiin. Tapaustutkimus koostuu useista mustalaatikko-menetelmällä suoritetuista testeistä, joissa käytetään Linux-käyttöjärjestelmässä toimivia työkaluohjelmia virtualisoidussa testiympäristössä. Opinnäytetyön johtopäätöksissä analysoidaan tutkimusprosessissa kerääntynyttä tietoa OWASP-metodologian soveltamisesta valmisohjelmistojen sovellusturvallisuuden arviointiin.

1.1 Opinnäytetyön tavoitteet

Tutkimuksen päätavoitteena on muodostaa kriittinen kokonaiskuva OWASP-sovellusturvallisuuden arviointimetodologian soveltamisesta käytännössä valmisohjelmistojen sovellusturvallisuuden arviointiin ja testaamiseen. OWASP-metodologiaa tutkitaan sekä teoreettisesti analysoimalla OWASP-metodologian sisältöä että käytännön tapaustutkimuksena proof of concept -näkökulmasta OWASP-metodologian käyttämisestä IBM InfoSphere MDM Collaboration Server -ohjelmistotuotteen sovellusturvallisuuden arvioimiseksi. OWASP-metodologian kokonaiskuvan lisäksi tavoitteena on pohtia sen soveltamista yritysten ja muiden organisaatioiden IT-toimintoihin, erityisesti sen sopivuutta tilaajan ja toimittajan välisten neuvotteluiden työkaluksi valmisohjelmistoihin perustuvien tietojärjestelmien hankinta- ja toimitusprojekteihin sekä vaatimustenhallintaan.

Tutkimuksen tuloksena syntyy kokonaisarvio OWASP-sovellusturvallisuuden arviointimetodologian soveltamisesta käytännössä valmisohjelmistoihin perustuviin tietojärjestelmiin. Projektissa saatujen kokemusten perusteella voidaan esittää parhaita käytäntöjä yleiseen sovellusturvallisuuden arviointiin. Tapaustutkimuksen konkreettisena tuloksena syntyy raportti IBM InfoSphere MDM Collaboration Server -tietojärjestelmän sovellusturvallisuuden tilasta.

Projektin oppimistavoitteena on tekijän tietoturvaosaamisen syventäminen erityisesti sovellusturvallisuuden arvioinnin ja avoimen lähdekoodin verkkoturvallisuusohjelmis-

tojen käyttämisen osalta. Tavoitteena on myös oppia yleisestä tietoturvan hallinnan ja kehittämisen konsultoinnista.

1.2 Tutkimusongelmat ja aiheen rajaus

Tutkimustehtävänä on selvittää, kuinka OWASP-sovellusturvallisuuden arviointimetodologiaa ja avoimen lähdekoodin verkkoturvallisuusohjelmistoja käyttämällä voidaan arvioida kaupallisen, suljetun lähdekoodin valmisohjelmiston sovellusturvallisuutta. Tutkimusmenetelminä käytetään laadullista tapaustutkimusta ja konstruktiivista tutkimusta (Ojasalo, Moilanen & Ritalahti, 52, 65). Tapaustutkimuksen kohteeksi valittiin J2EE-tekniikalla toteutettu IBM InfoSphere MDM Collaboration Server -ohjelmistotuote. Tapaustutkimus suoritetaan suljetussa virtuaaliympäristössä ja sovellusturvallisuuden testaamiseen käytetään Kali Linux -distribuutiota ja sen sisältämiä avoimen lähdekoodin verkkoturvallisuusohjelmistoja.

OWASP-sovellusturvallisuuden arviointimetodologia rajataan tutkimuksessa dokumentteihin OWASP Testing Guide 2008 v3.0 ja OWASP Application Security Verification Standard 2009, jotka ovat julkisesti saatavilla OWASP-säätiön verkkosivuilta. Tutkimuksessa käsiteltävä valmisohjelmiston sovellusturvallisuus rajataan yksittäiseen ohjelmistotuotteen toimintaan. Käyttöjärjestelmän, väliohjelmistojen kuten sovellus- ja tietokantapalvelinten sekä tietoverkkojen tietoturvaa ei erikseen arvioida ja testata. Sovellusturvallisuuden testaamisessa käytetään vain Linux-alustalla ajettavia avoimen lähdekoodin verkkoturvallisuusohjelmistoja. Tutkimusta varten ei kehitetä omia ohjelmistoja. Testaaminen suoritetaan mustalaatikko-menetelmällä, jolloin etsitään järjestelmän ulkopuolelta havaittavia ja hyödynnettävissä olevia haavoittuvuuksia. Tutkittavan ohjelmistotuotteen lähdekoodia ei yritetä purkaa takaisinmallinnustekniikoilla (engl. reverse engineering).

1.3 Käsitteet

Avoin lähdekoodi, engl. open source, ohjelmistojen lisensointimalli, joka mahdollistaa ohjelmistojen vapaan käytön, muokkaamisen ja levittämisen. Avoimen lähdekoodin ohjelmistojen ovat yleensä ilmaisia.
Eettinen hakkerointi, engl. ethical hackering, ks. mustalaatikkotestaus.

Haavoittuvuus, engl. vulnerability, virhe tai heikkous tietojärjestelmän suunnittelussa, toteutuksessa, operoinnissa tai hallinnassa, jota voidaan käyttää hyväksi tietojärjestelmän tai sovelluksen tietoturvan loukkaamiseen.

Harmaalaatikkotestaus, engl. grey box testing, mustalaatikko- ja valkoinenlaatikkotestauksen yhdistelmä, jossa testaaja käyttää sekä penetraatiotestauksen menetelmiä että osittaista tietoa sovelluksen sisäisestä rakenteesta.

Hyökkäys, engl. attack, tietoinen yritys hankkia luvaton pääsy tietojärjestelmään ja käyttää luvattomasti sen resursseja. Luvaton käyttö voi olla tietojen varastamista, tuhoamista, muuttamista tai paljastamista. Hyökkäys voi häiritä tai vaurioittaa tietojärjestelmää tai estää sen käyttöä.

Mustalaatikkotestaus, engl. black box testing, tuotantoympäristössä tai sitä vastaavassa testiympäristössä ajettavan sovelluksen tietoturvan ulkopuolinen testaaminen käyttäen simuloituja hyökkäysmenetelmiä tarkoituksen löytää hyödynnettäviä haavoittuvuuksia. Testaajalla ei ole ennakkotietoa sovelluksen sisäisestä rakenteesta ja toimintalogiikasta.

Penetraatiotestaus, engl. penetration testing, ks. mustalaatikkotestaus.

Sovellusturvallisuus, engl. application security, sovelluksen turvallisuutta suojaavat toimenpiteet, joilla pyritään estämään tietoturvaloukkaukset ja haavoittuvuuksien mahdollistavat hyökkäykset sovelluksen elinkaaren kaikissa vaiheissa. Sovellusturvallisuuteen kuuluu uhkien kartoittaminen, tietojärjestelmien kaikkien komponenttien varmistaminen ja tietoturvallisen ohjelmistokehitysprosessin kehittäminen.

Uhka, engl. threat, potentiaalinen haavoittuvuutta hyväksi käyttävä hyökkäys, joka vaarantaa sovelluksen hallinnassa olevien resurssien kuten tietokantojen tai tiedostojen eheyden, luottamuksellisuuden ja saatavuuden.

Valkoinenlaatikkotestaus, engl. white box testing, sovelluksen sisäisen tietoturvan testaaminen. Valkoinenlaatikkotestauksessa analysoidaan ohjelmistokehitysvaiheessa sovelluksen lähdekoodia, joista etsitään haavoittuvuuksia tai suunnitteluvirheitä.

Valmisohjelmisto, engl. commercial off-the-self software, toimittajan tai kolmannen osapuolen valmiiksi rakentama ohjelmisto tai ohjelmistokomponentti, jota myydään ja lisensoidaan julkisesti useille asiakkaille. Valmisohjelmistojen lähdekoodi on yleensä suljettu, jolloin ohjelmiston myytävään lisenssiin kuuluu vain ohjelmiston käyttöoikeus.

2 Sovellusturvallisuuden arvioinnin menetelmät OWASP-metodologiassa

2.1 Mikä on OWASP?

Open Web Application Security Project (OWASP) on avoin ja kansainvälinen sovellusturvallisuuden edistämiseen keskittynyt julkinen organisaatio. OWASP:n toiminta käynnistyi Yhdysvalloissa vuonna 2001 ja vuonna 2004 perustettiin OWASP-säätiö, joka on Yhdysvaltojen lainsäädännön mukainen voittoa tavoittelematon yleishyödyllinen yhteisö. OWASP-säätiön tarkoituksena on tietokoneohjelmistojen sovellusturvallisuuden edistäminen ja siitä tiedottaminen. OWASP-säätiön arvoja ovat avoimuus toiminnassa, innovointi uusien tietoturvaratkaisujen kehittämiseksi, kansainvälisyyteen perustuva toimintamalli sekä lahjomattomuus rehellisenä ja avoimena yhteisönä, joka ei ole sidottu kaupallisiin intresseihin. (OWASP 2013.)

OWASP:n toiminta perustuu kansainväliseen katto-organisaatioon sekä ympäri maailmaa perustettuihin paikallisyhdistyksiin[1], jotka järjestävät säännöllisesti avoimia yleisötilaisuuksia sovellusturvallisuudesta. OWASP-säätiö järjestää vuosittain kansainvälisiä AppSec-konferensseja, jotka kokoavat tietoturvan ammattilaisia keskustelemaan sovellusturvallisuuden ajankohtaisista teemoista.[2] OWASP-organisaatioon kuuluu myös lukuisia projekteja[3], jotka tuottavat apuvälineitä tietoturvalliseen ohjelmistokehitykseen, tietoturvan testaamiseen soveltuvia työkaluja sekä sovellusturvallisuuteen liittyvää dokumentaatiota. Kaikki OWASP-säätiön ja sen sisällä toimivien projektien tuottamat materiaalit julkaistaan avoimen lähdekoodin lisensseillä, jotka mahdollistavat niiden ilmaisen ja vapaan hyödyntämisen. (OWASP 2013a.)

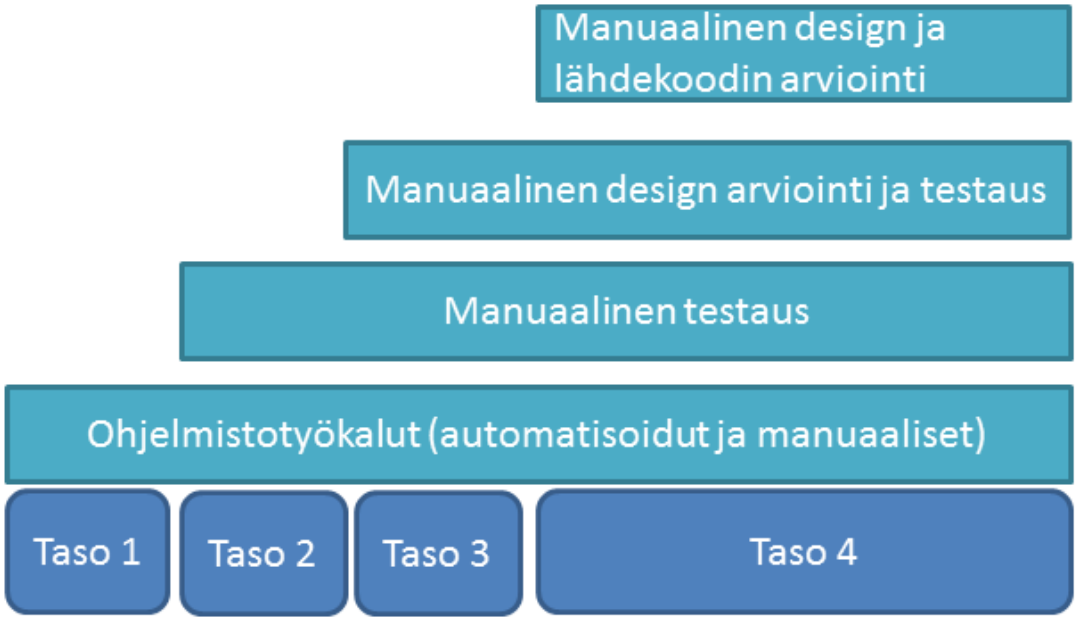
[1] Suomessa toimii OWASP Helsinki Chapter, joka on järjestänyt avoimia kokoontumisia vuodesta 2006 lähtien. OWASP Helsinki Chapter esittely: https://www.owasp.org/index.php/Helsinki.
[2] OWASP AppSec -konferensseja järjestetään Pohjois-Amerikassa, Etelä-Amerikassa, Aasiassa ja Euroopassa. AppSec-konferenssien ajankohdat ja teemat: https://www.owasp.org/index.php/Conferences.
[3] Vuoden 2013 aktiiviset projektit on kuvattu OWASP Projects Handbook 2013 -oppaassa: https://www.owasp.org/images/6/6a/OWASP_Projects_Handbook_2013.pdf.

2.2 OWASP Application Security Verification Standard – sovellusturvallisuuden mittaristo

OWASP Application Security Verification Standard (ASVS) -projekti käynnistettiin vuonna 2008. Projektin tarkoituksena on luoda avoin standardi sovellusturvallisuuden tarkastamiseen, joka pyrkii tasoittamaan sovellusturvallisuuden arviointiin käytettävissä olevien lukuisien ohjelmistotyökalujen ja manuaalisen testaamisen tekniikoiden väliset erot kattavuudessa ja tarkkuudessa. Standardi tarjoaa perustan ohjelmistojen ja niiden suorituksen aikaisten ympäristöjen teknisten tietoturvakontrollien[4] testaamiseen ja niiden kykyyn suojautua tunnetuilta haavoittuvuuksilta[5]. Testattavien ohjelmistojen turvallisuuden luottamustaso voidaan määrittää ASVS-standardin pohjalta. ASVS-standardia voidaan käyttää sovellusturvallisuuden mittaristona ohjelmistokehityksessä, ohjeena sovellusturvallisuuden vaatimukset täyttävien tietoturvakontrollien kehittämiseksi sekä ohjelmistojen hankintaprosessissa sovellusturvallisuuden vaatimusten määrittämiseen tilaajan ja toimittajan välisissä sopimuksissa. (OWASP 2012; OWASP 2009a.)

OWASP ASVS -projektin tuloksena on syntynyt sovellusturvallisuuden arviointistandardin dokumentaatio, joka on julkaistu OWASP-säätiön nettisivuilla (OWASP 2009b). Sovellusturvallisuuden arviointistandardissa kuvataan neljästä hierarkkisesti sovellusturvallisuuden tasosta rakentuva sovellusturvallisuuden arvioinnin viitekehys (kuvio 1). Tasot rakentuvat toistensa päälle niin että ylemmän tason läpäisemiseksi sovelluksen on hyväksyttävästi täytettävä alempien tasojen vaatimukset. Sovellusturvallisuuden arviointimenetelminä käytetään penetraatiotestausta (mustalaatikko-menetelmä) sekä sovelluksen lähdekoodin arviointia (valkoinenlaatikko-menetelmä). Työmenetelminä käytetään sekä automatisoituja tarkistus- ja testaustyökaluja sekä asiantuntijan manuaalisesti suorittamaa arviointia ja testaamista. (OWASP 2009a; OWASP 2009b, 1.)

[4] Tietoturvakontrollilla tarkoitetaan ohjelmiston osaa tai komponenttia, joka turvaa tiedon luottamuksellisuutta, eheyttä ja saatavuutta. Tekninen kontrolli voi olla esim. käyttöoikeuksien tarkistaminen tai audit-lokin kirjaaminen.
[5] Tunnetuilla haavoittuvuuksilla tarkoitetaan tässä yleisiä haavoittuvuustyyppejä kuten SQL-injektiota.



Kuvio 1. OWASP ASVS-sovellusturvallisuuden tasojen arviointi. Lähde: OWASP ASVS 2009.

OWASP ASVS-sovellusturvallisuuden tasot koostuvat määriteltyjen vaatimusten joukosta. Vaatimuksia on kolmenlaisia: ylätason abstraktit tietoturvavaatimukset, yksityiskohtaiset sovelluksen tietoturvakontrolleihin liittyvät vaatimukset ja raportoinnin vaatimukset. Ylemmillä tasoilla testattavien vaatimusten määrä kasvaa sekä testaamisessa vaaditaan suurempaa tarkkuutta. Kaikilla tasoilla arvioinnissa voidaan käyttää ohjelmistotyökaluja, joilla saadut tulokset on aina tarkistettava manuaalisesti. Sovellusturvallisuuden arvioinnin viitekehys on kuitenkin ohjeellinen, jolloin arvioijan vastuulle jää määritellä mille tasolla sovelluksen tietoturva sijoittuu. Arvioinnissa voidaan ottaa huomioon myös organisaation omia, lainsäädäntöön tai liiketoimintaan perustuvia tietoturvavaatimuksia. (OWASP 2009b, 4.)

Tasolla 1 sovellusturvallisuuden tarkistus suoritetaan automatisoiduilla ohjelmistotyökaluilla. Sovellukselta vaaditaan tietoturvakontrollien oikeanlaista käyttöä, niin että sovelluksen tietoturvallisuuden perustasoon voidaan luottaa. Riskien arvioinnissa keskitytään yleisiin viruksiin tai matoihin, jotka pyrkivät saastuttamaan heikoimmin suojattuja sovelluksia. Sovellusturvallisuuden arviointi rajataan sovelluksen rakentamiseksi kehitet-

tyyn ohjelmakoodiin. Taso 1 jakaantuu vielä sisäisesti tasoon 1A ja tasoon 1B. Taso 1A on automatisoidun penetraatiotestauksen[6] suorittaminen ja taso 1B sovelluksen lähdekoodin tarkistaminen[7] automatisoiduilla ohjelmistotyökaluilla. Tason 1 raportointivaatimuksiin kuuluu sovelluksen arkkitehtuuriin kuuluvien eri komponenttien tunnistaminen ja kuvaaminen yleisellä tasolla. (OWASP 2009b, 4-7.)

Tasolla 2 sovellusturvallisuus tarkistetaan manuaalisesti. Tason 2 sovellusturvallisuuden tarkistus sopii transaktioita käsitteleville sovelluksille, jotka käsittelevät liiketoimintadataa, luottotietoja tai henkilötietoja. Sovellukselta vaaditaan tietoturvakontrollien oikeaa käyttöä sekä varmistusta niiden toimivuudesta. Riskien arvioinnissa keskitytään virusten ja matojen lisäksi kokemattomiin, satunnaisiin hyökkääjiin. Sovellusturvallisuuden arvioinnin rajauksessa otetaan huomioon sovellusta varten kehitetyn ohjelmakoodin lisäksi tietoturvatoimintojen toteuttamisessa käytetyt kolmannen osapuolen ohjelmistokomponentit. Taso 2 jakaantuu vielä sisäisesti tasoon 2A ja tasoon 2B. Taso 2A on manuaalinen penetraatiotestaus sovelluksen tietoturvakontrollien tarkistamiseksi ja taso 2B sovelluksen manuaalinen lähdekoodin tarkistaminen tietoturvakontrollien varmistamiseksi. Tason 2 raportointivaatimuksena on sovelluksen loogisen arkkitehtuurin kuvaaminen esimerkiksi MVC-mallin[8] mukaisesti sekä loppukäyttäjälle avoinna olevien tulokohtien kuvaaminen. (OWASP 2009b, 7-10.)

Tasolla 3 tarkistetaan manuaalisesti ja arvioidaan sovelluksen arkkitehtuuria. Tason 3 tarkistus sopii sovelluksille, jotka käsittelevät merkittävää tietoa kuten terveydenhoidon informaatiota, liiketoiminnalle kriittistä tai muuta sensitiivistä aineistoa. Riskien arvioinnissa otetaan huomioon taitavan hyökkääjän suorittaman kohdistetun hyökkäyksen mahdollisuus. Sovellusturvallisuuden arvioinnin rajaukseen kuuluu nyt sovellusta varten kehitetyn ohjelmakoodin lisäksi kaikki sovelluksessa käytetyt kolmannen osapuolen ohjelmistokomponentit. Sovellukselta edellytetään tietoturvakontrollien oikeaa käyt-

[6] Automatisoidusta penetraatiotestauksesta käytetään myös nimitystä dynaaminen skannaus tai sovelluksen haavoittuusskannaus.
[7] Lähdekoodin tarkistuksesta käytetään nimitystä staattinen analyysi. Tarkistuksessa käytetyt ohjelmistotyökalut hakevat lähdekoodista yleisiä toimintalogiikan malleja, jotka aiheuttavat haavoittuvuuksia sovelluksessa.
[8] Model-View-Controller on yleinen ohjelmistokehityksen arkkitehtuuri, jossa tiedon esittäminen ja käyttäjän vuorovaikutus on erotettu toisistaan.

töä sekä niiden käyttöä kaikkialle sovelluksessa sen tietoturvakäytäntöjen toteuttamiseksi. Tason 3 raportointivaatimuksena on sovelluksen turvallisuusarkkitehtuurin kuvaaminen, niin että loogisen arkkitehtuurin kuvaukseen on yhdistetty uhkamallinnuksella[9] tunnistetut potentiaaliset hyökkääjät ja hyökkäysmenetelmät, sekä kaikki sovelluksen loogisen arkkitehtuurin mukaiset tulokohdat on tunnistettu. (OWASP 2009b, 10-12.)

Tasolla 4 arvioidaan sovelluksen sisäinen rakenne kokonaisuudessaan. Tason 4 tarkistus sopii kriittisille sovelluksille, jotka suojaavat ihmishenkeä sekä yleistä turvallisuutta tai niiden toimintaa liittyy kriittiseen infrastruktuuriin tai maanpuolustukseen. Jossain tapauksissa myös sensitiivistä aineistoa käsitteleville sovelluksille voidaan suorittaa tason 4 tarkistus. Riskien arvioinnissa huomioidaan todennäköinen taitavan hyökkääjän kohdistettu hyökkäys. Sovellukselta edellytetään sen tietoturvakäytäntöjen mukaista tietoturvakontrollien käyttöä kaikkialla sovelluksessa sekä sovelluksen lähdekoodissa noudatettavaa tietoturvallista ohjelmointitapaa. (OWASP 2009b, 13-15.)

Sovellusturvallisuuden arvioinnin rajaukseen tasolla 4 sisältyy kaikki sovelluksessa käytetty sekä sen kehittämisessä käytettyjen ohjelmistojen lähdekoodi. Rajauksen ulkopuolella jätetään vain sovelluksen alustaan liittyvä ohjelmistojen lähdekoodi, kuten sovelluspalvelimet, tietokannat, käyttöjärjestelmät ja virtuaalikoneet. Tason 4 raportointivaatimuksissa on kuvattava sovelluksen turvallisuusarkkitehtuuri kokonaisuutena, jossa myös kaikki sovelluksen liittyvä, tarkastamaton lähdekoodi on tunnistettu ja kuvattu. (OWASP 2009b, 13-15.)

OWASP ASVS -standardissa on sovelluksen neljän tietoturvatason lisäksi eritelty 14 yksityiskohtaisten tietoturvavaatimusten kategoriaa, joiden sisällä jokaiselle neljän tasolle on määritelty niiden läpäisemiseen tarvittava yksityiskohtaisten tietoturvavaatimusten joukko. Yksityiskohtaisten tietoturvavaatimusten kategoriat ovat sovelluksen turvallisuusarkkitehtuuri (V1), autentikointi (V2), istuntojenhallinta (V3), pääsynvalvonta (V4), syötteiden validointi (V5), tulosteiden merkistökoodaus (V6), salaustekniikka (V7), vir-

[9] OWASP-säätiö suosittelee erityisesti Microsoftin kehittämien uhkamallinnusmenetelmien käyttämistä. Lisätietoa uhkamallinnuksesta on OWASP-sivuston artikkelissa "Threat Risk Modeling": https://www.owasp.org/index.php/Threat_Risk_Modeling.

heenkäsittely ja lokit (V8), tiedon suojaaminen (V9), tietoliikenteen turvallisuus (V10), HTTP-liikenteen turvallisuus (V11), sovelluksen tietoturva-asetukset (V12), haitallisen lähdekoodin havaitseminen (V13) ja sovelluksen sisäinen turvallisuus (V14). Ylätasoihin liittyvät vaatimuksiin ei välttämättä kuulu vaatimuksia kaikista kategorioista. Tason 1 sovellusturvallisuuden tarkistuksessa ei ole vaatimuksia salaustekniikasta (V7), sovellukseen tietoturva-asetuksista (V12), haitallisen lähdekoodin havaitsemisesta (V13) ja sovelluksen sisäisestä turvallisuudesta (V14). (OWASP 2009b, 16, 24, 29-31.)

OWASP ASVS -standardissa on määritelty myös sovellusturvallisuuden tarkistuksesta annettavan raportin rakenne. Raportin johdanto-osassa tulee antaa tunnistettava kuvaus tarkastetusta sovelluksesta, yleisarvio sen tietoturvallisuuden tasosta, sovelluksen käyttöön liittyvät keskeiset liiketoiminnan riskit sekä kuvata tarkastuksessa käytetyt toimintatavat. Johdannon jälkeen seuraa kuvaus sovelluksesta, jonka avulla sen toiminta ja toimintaympäristö on ymmärrettävissä. Raportin kolmantena osiona on sovelluksen turvallisuusarkkitehtuurin kuvaus, jonka tehtävänä on esittää tarkastuksen konteksti sekä vakuuttaa lukija tarkistuksen kattavuudesta ja tarkkuudesta. Sovelluksen turvallisuusarkkitehtuurin kuvauksen yksityiskohdat määräytyvät tarkastuksessa käytetyn ylätason mukaisesti. (OWASP 2009b, 32.)

Raportin viimeisessä osiossa esitetään sovellusturvallisuuden tarkistuksen tulokset tarkastuksessa käytetyn ylätason mukaisesti. Tulokset esitetään yksityiskohtaisten vaatimusten mukaan ja jokaisesta tarkistetusta vaatimuksesta annetaan arvio (täyttää vaatimuksen/ei täytä vaatimusta). Tuloksien esittämisen yhteydessä on kuvattava tarkastuksessa käytetyt ohjelmistotyökalut ja niissä käytetyt asetukset. Sovelluksessa tietoturvassa havaituista puutteista on esitettävä tarkka sijainti (URL-osoitteet parametreineen, sovelluksen lähdekoodi-tiedostoista rivien tarkkuudella), haavoittuvuuden kuvaus sekä arvio riskin vakavuudesta. Ylätasojen 2-4 tarkistusten yhteydessä on lisäksi esitettävä, millä perusteilla arvioija katsoo tarkastetun vaatimuksen täytetyksi. Sovelluksen haavoittuvuuksista on kuvattava haavoittuvuuteen johtava sovelluksen komponenttien toiminta ja sen aikaansaamiseksi tarvittavat, toistettavissa olevat toimenpiteet. (OWASP 2009b, 33-34.)

2.3 OWASP Testing Guide – sovellusturvallisuuden testaamisen viitekehys

OWASP Testing Guide -projekti käynnistettiin vuonna 2003. Projektin tarkoituksena on tuottaa websovellusten tietoturvatestauksen yleinen viitekehys, joka koostuu alan parhaista käytännöistä sekä yksityiskohtainen kuvaus penetraatiotestauksen menetelmistä ja tekniikoista. Projektin tuloksena on syntynyt OWASP Testing Guide -dokumentaatio, josta on vuoteen 2013 mennessä julkaista kolme versiota. Vuonna 2008 julkaistu versio 3.0 on viimeisin OWASP-säätiön tarkastama, "virallisen" aseman saavuttanut versio. OWASP Testing Guide -dokumentin neljättä versiota ollaan parhaillaan kehittämässä ja sen draft-versio (OWASP 2013c) on luettavissa projektin wikisivustolta. (OWASP 2008, 14; OWASP 2013b.)

OWASP Testing Guide 3.0 -dokumentti jakaantuu neljään osioon. Johdannossa (luku 2) esitetään websovellusten tietoturvatestauksen yleiset periaatteet, erityyppiset testaustekniikat sekä testauksen lähtökohtana olevien tietoturvavaatimusten määritteleminen. OWASP Testing Guide -projektissa kehitetty websovellusten tietoturvatestauksen yleinen viitekehys kuvataan omassa osiossaan (luku 3). Tietoturvatestauksen viitekehys on jäsennetty ohjelmistokehityksen elinkaarimallin (engl. Software Development Life Cycle, SDLC) vaiheiden mukaisesti alkaen suunnitteluvaiheesta jatkuen aina valmiin ohjelmiston käyttöönottoon ja ylläpitovaiheeseen asti. Viitekehyksessä varsinainen penetraatiotestaus kuuluu valmiin ohjelmiston käyttöönotto-vaiheeseen (engl. deployment). Sovellusten haavoittuvuuksien lisäksi penetraatiotestaukseen sisältyy myös sovelluksen asetusten testaaminen. (OWASP 2008, 15, 41, 44-45.)

Websovelluksen penetraatiotestausta ja sen osa-alueita käsitellään yksityiskohtaisesti OWASP Testing Guide 3.0 -dokumentin luvussa 4. Penetraatiotestauksen metodologian lähtökohtana on mustalaatikkotestaus, jossa testaajalla ei ole ennakkotietoa sovelluksen sisäisestä rakenteesta. Testauksen malli koostuu testaajasta, työkaluista ja testimenetelmistä sekä testattavasta sovelluksesta. Testaaminen jakaantuu passiiviseen ja aktiiviseen vaiheeseen. Passiivisessa vaiheessa testaaja kerää tietoja sovelluksesta pyrkien ymmärtämään sovelluksen toimintaa ja tunnistamaan sovelluksen avoimet rajapinnat. Aktiivisessa vaiheessa testataan sovelluksen tietoturvakontrollien toimivuutta ennalta määriteltyjen tietoturvavaatimusten mukaisesti. Aktiivisen vaiheen testit ovat jao-

teltu yhdeksään kategoriaan. Testattavia tietoturvakontrolleja on yhteensä 66 kappaletta. (OWASP 2008, 47-48.)

OWASP Testing Guide 3.0 -dokumentissa määritellyt testattavien tietoturvakontrollien kategoriat ovat sovelluksen asetustenhallinta, sovelluslogiikan testaus, autentikointi, pääsynvalvonta, istuntojenhallinta, syötteiden validointi, palvelunestohyökkäykset, sovelluksen tarjoamien Web Service -palvelujen testaaminen ja sovelluksessa toteutuksessa käytetyn AJAX-tekniikan testaaminen. Testattavat tietoturvakontrollit vastaavat pääpiirteissään edellä mainittuja OWASP ASVS -standardin yksityiskohtaisten tietoturvavaatimusten kategorioita (luku 2.2). Testattavien tietoturvakontrollien kuvaamisessa käytetään mallia, joka koostuu lyhyestä yleistajuisesta johdannosta, lyhyestä teknisestä kuvauksesta, mustalaatikko- ja harmaalaatikkotestaus-esimerkeistä (testauksessa suoritettavat toimenpiteet ja odotetut testitulokset) sekä lähdeviittauksista aihetta käsitteleviin julkaisuihin ja työkaluohjelmistoihin. (OWASP 2008, 48; Meucci 2010, 22.)

OWASP Testing Guide 3.0 -dokumentin viidennessä luvussa kuvataan websovelluksen penetraatiotestauksen raportointia, jonka olennainen osa on testituloksiin liittyvien todellisten riskien arviointi ja kuvaus. Riskien arvioinnin lähtökohtana on yleisesti käytetty malli[10], jonka mukaan riski = todennäköisyys x seuraus. Riskien arviointi jakaantuu kuuteen vaiheeseen: 1. riskin tunnistaminen, 2. todennäköisyystekijöiden arviointi, 3. liiketoimintaan vaikuttavien seurauksien arviointi, 4. riskin vakavuuden arviointi, 5. tarvittavien tietoturvakorjausten priorisointi ja 6. riskien arviointimallin sovittaminen oman organisaation tarpeisiin. (OWASP 2008, 327.)

Riskin todennäköisyys ilmaistaan yleisellä asteikolla vähäinen, kohtalainen ja suuri. Todennäköisyys jaetaan tarkemmassa arvioinnissa asteikolla 0-9 pisteytettyihin osatekijöihin. Asteikon arvoilla on jokaiselle osatekijälle yksilöity sanallinen kuvaus. Riskin todennäköisyyden osatekijät jaetaan kahteen joukkoon: hyökkääjään liittyviin osatekijöihin sekä haavoittuvuuksiin liittyviin osatekijöihin (taulukko 1). Riskien seuraus ilmais-

[10] Tietoturvariskien arvioimista on kuvattu laajasti valtionhallinnon tietoturvallisuusohjeessa VAHTI 7/2003, joka on ladattavissa osoitteesta
http://www.vm.fi/vm/fi/04_julkaisut_ja_asiakirjat/01_julkaisut/05_valtionhallinnon_tietoturvallisuus/53828/53827_fi.pdf.

taan myös samanlaisella yleisellä asteikolla ja jaotellaan osatekijöihin kuin riskin todennäköisyys. Riskien seurausten osatekijät jakaantuvat teknisiin seurauksiin ja liiketoimintaan vaikuttaviin seurauksiin (taulukko 2). (OWASP 2008, 329-330.)

Taulukko 1. Riskin todennäköisyyden osatekijät (asteikko 0-9).

Hyökkääjän liittyvät osatekijät				Haavoittuvuuteen liittyvät osatekijät			
Taitotaso	Motiivi	Mahdollisuus	Henkilömäärä	Löytämisen helppous	Hyödyntämisen helppous	Tunnettavuus	Tunkeutumisen havaitseminen
Riskin yleinen todennäköisyys (osatekijöiden keskiarvo) = 0-2 (vähäinen), 3-5 (keskimääräinen), 6-9 (korkea)							

Lähde: OWASP 2008, 331.

Taulukko 2. Riskin seurauksen osatekijät (asteikko 0-9).

Teknisten seurausten osatekijät				Liiketoimintaan vaikuttavat osatekijät			
Tiedon luottamuksellisuuden menetys	Tiedon eheyden menetys	Tiedon saatavuuden menetys	Tiedon jäljitettävyyden menetys	Taloudelliset menetykset	Maineen menettäminen	Lainsäädännölliset rikkomukset	Yksityisyydensuojan rikkomukset
Riskin yleisen seurauksen vakavuus (osatekijöiden keskiarvo) = 0-2 (vähäinen), 3-5 (kohtalainen), 6-9 (vakava)							

Lähde: OWASP 2008, 331.

Riskien vakavuuden arvioinnissa edellä kuvattujen riskin osatekijät suhteutetaan yleiseen asteikkoon, niin että arvot 0-2 vastaavat vähäistä riskiä, 3-5 kohtalaista riskiä ja 6-9 korkeaa riskiä. Riskien todennäköisyys ja seuraus arvioidaan laskemalla riskin osatekijöiden joukon keskiarvo, jonka pohjalta tehdään lopullinen arvio riskin vakavuudesta. Osatekijöitä voidaan käsitellä eri tavalla painotettuina kokonaisuuksina, jos halutaan arvioida riskiä organisaation omien tarpeiden mukaisesti. Yleisessä riskin vakavuuden arvioinnissa käytetään riskin todennäköisyyden ja seurauksen kerrointa (taulukko 3). (OWASP 2008, 330-331.)

Taulukko 3. OWASP Testing Guide 3.0 mukainen riskien vakavuuden arviointi.

Riskin vakavuus				
Riskin seuraus	Vakava	Kohtalainen	Merkittävä	Kriittinen
	Kohtalainen	Vähäinen	Kohtalainen	Merkittävä
	Vähäinen	Merkityksetön	Vähäinen	Kohtalainen
		Vähäinen	Keskimääräinen	Korkea
	Riskin todennäköisyys			

Lähde: OWASP 2008, 331.

OWASP Testing Guide 3.0 -dokumentissa annetaan suositus websovelluksen penetraatiotestauksen loppuraportin rakenteesta. Raportin tulisi huomioida sekä organisaation johdon että teknisten asiantuntijoiden tarpeet. Raportti jakaantuu neljään osioon. Ensimmäinen osa on yleistajuinen tiivistelmä testituloksista, niiden mahdollisista seurauksista ja tarvittavista toimenpiteistä. Tämä osio on suunnattu erityisesti organisaation liiketoiminnan johdon edustajille. Toinen osio on tekninen tiivistelmä testituloksista, joka on suunnattu organisaation tekniselle johdolle. Kolmannessa osiossa esitetään varsinaiset testitulokset yksityiskohtaisella teknisellä tasolla, niin että organisaation tekniset asiantuntijat ymmärtävät löydyttyjen haavoittuvuuksien toiminnan ja pystyvät tekemään tarvittavat korjaukset websovellukseen. Testitulosten esittämisessä voidaan käyttää taulukkomuotoa, jonka sarakkeina ovat testin viitenumero, testattu komponentti, tekninen kuvaus haavoittuvuudesta, tekninen kuvaus korjaavasta toimenpiteestä sekä arvio riskin vakavuudesta. Raportin neljännessä osiossa kuvataan testaamisessa käytetyt kaupalliset ja avoimen lähdekoodin ohjelmistotyökalut sekä testaajan kehittämät skriptit tai ohjel-

mistotyökalut. Ohjelmistotyökalujen kuvaaminen helpottaa testauksen tarkkuuden arvioimista. (OWASP 2008, 333-334,337.)

OWASP Testing Guide 3.0 -dokumentin liitteissä on kokoava luettelo testattavien tietoturvakontrollien yhteydessä esitellyistä avoimen lähdekoodin ja kaupallisista ohjelmistotyökaluista (Appendix A), kirjallisuusluettelo websovellusten tietoturvaan liittyvistä julkaisuista (Appendix B), ohjeita syötteiden validoinnin fuzz-testaukseen[11] (Appendix C) sekä ohjeita websovelluksen merkistökoodauksen[12] käsittelyn testaamiseen (Appendix D). (OWASP 2008, 338, 341, 342, 347.)

[11] Fuzz-testaus (engl. fuzz testing, fuzzing) on mustalaatikko-menetelmä, jossa sovellukselle syötetään automatisoidusti väärin muotoiltua, osittain satunnaista dataa. Tarkoituksena on etsiä sovelluksen syötteiden käsittelystä ohjelmointivirheitä ja niiden aiheuttamia haavoittuvuuksia kuten puskuriylivuotoja tai palvelunestoja. Lisätietoa aiheesta osoitteessa https://www.owasp.org/index.php/Fuzzing.
[12] Tässä yhteydessä käsitellään lähinnä HTTP-protokollan merkistökoodaukseen liittyviä haavoittuvuuksia.

3 OWASP-metodologian soveltaminen valmisohjelmiston sovellusturvallisuuden arviointiin ja testaamiseen

3.1 Case: IBM InfoSphere MDM Collaboration Server -ohjelmisto

IBM InfoSphere MDM Collaboration Server (MDM CS) on yrityskäyttöön tarkoitettu tuotetietojen hallinnan (engl. Product Information Management, PIM) tietojärjestelmä, jonka keskeisiä sovelluskohteita ovat vähittäis- ja tukkukaupan alan liiketoiminta ja valmistavan teollisuuden yritysten tuotetietojen hallinta. MDM CS toimii väliohjelmistona (engl. middleware), jonka kautta yrityksen liiketoiminnassa käytettäviä tuotetietoja hallitaan keskitetysti yhdessä tietojärjestelmässä. Tuotetietojen hallinnan tietojärjestelmällä avulla yritys voi kehittää ja tehostaa tuotteisiin liittyviä liiketoimintaprosesseja. MDM CS -järjestelmän käyttökohteita ovat muun muassa toimittaja- ja yritysasiakastietojen hallinta (engl. trading-partner management), tilaustoimitusketjun hallinta (engl. supply-chain management), integroituminen kansallisiin tai globaaleihin tuotetietopankkeihin[13] sekä verkkokauppasovellukset[14] (engl. e-commerce). (IBM 2013a.)

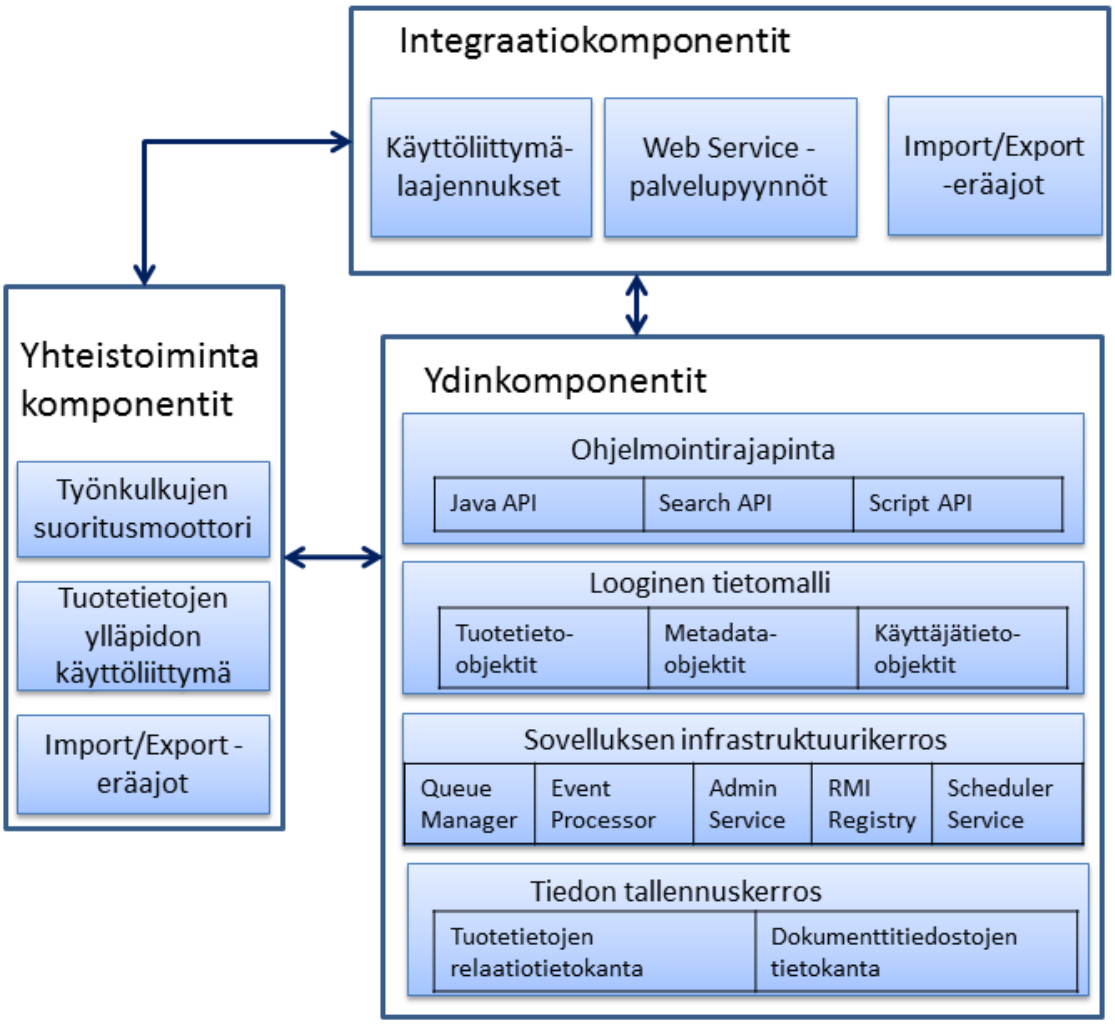
MSM CS -järjestelmä on Java-kielellä ohjelmoitu, J2EE-tyyppinen websovellus, joka tarvitsee alustakseen Java-sovelluspalvelimen ja relaatiotietokannan. Sovelluspalvelimena voidaan käyttää IBM WebSphere Application Server tai Oracle/BEA WebLogic Server -tuotteita. Tuettuja tietokantapalvelimia ovat IBM DB2 Enterprise Server Edition ja Oracle Database Enterprise Edition. MDM CS -järjestelmä tarvitsee toimiakseen myös avoimen lähdekoodin Perl-komentotulkin. Käyttöjärjestelmäarkkitehtuureista tuettuja ovat AIX POWER System, Solaris ja Linux. (IBM 2013b.)

MDM CS -järjestelmän sisäinen arkkitehtuuri on komponenttilähtöinen (kuvio 2). Järjestelmä rakentuu kolmeen pääryhmään kuuluvista ohjelmistokomponenteista. Pää-

[13] Esimerkiksi Suomessa on käytössä kansanvälisen GS1-organisaation ylläpitämä WS1 Sinfos -tuotepankki, jonka käyttäjiä ovat mm. suurimmat päivittäistavara-alan kaupparyhmät. Lisätietoja osoitteessa http://www.gs1.fi/sinfos-tuotetietopankki.

[14] Tässä yhteydessä tarkoitetaan ensisijaisesti yritysten välistä B2B-kaupankäyntiä.

ryhmät ovat ydinkomponentit, integraatiokomponentit sekä yhteistyötoimintojen[15] (engl. collaboration) komponentit. Järjestelmän ydinkomponentit jakaantuvat neljään sovelluskerrokseen: ohjelmointirajapinta, loogisen tietomalli, sovelluksen infrastruktuurikerros ja tiedon tallennuskerros. Yhteistyökomponentteja ovat työnkulkujen suoritusmoottori, tuotetietojen ylläpidon käyttöliittymä sekä työnkulkuihin liittyvät import/export -eräajot. Järjestelmän integraatiokomponentteja ovat portaalirajapinta, käyttöliittymälaajennukset, ajastetut import/export -eräajot sekä Web Service -palvelupyynnöt. (IBM 2013c.)



Kuvio 2. InfoSphere MDM Collaboration Server -järjestelmän arkkitehtuuri. Lähde: IBM 2013a.

[15] Yhteistyötoiminnoilla tarkoitetaan MDM CS -järjestelmässä liiketoimintaprosessien mukaan mallinnettuja työnkulkuja (engl. workflow), joissa useat käyttäjät voivat muokata tuotetietoja.

MDM CS -järjestelmää voidaan hahmottaa kolmitasoisen[16] (engl. Three-Tier Architecture) client/server -ohjelmistoarkkitehtuurin mukaisesti. Järjestelmän ydinkomponentit tarjoavat järjestelmän loogisen kerroksen (engl. logic tier) ja tietokerroksen (engl. data tier). Ydinkomponenttien välityksellä tapahtuu tuotetietojen loogisen tietomallin ylläpitäminen ja tietojen tallentaminen tietokantaan. MDM CS -ohjelmiston yhteistoimintakomponentit muodostavat järjestelmän esityskerroksen (engl. presentation tier), jossa loppukäyttäjät voivat osallistua tuotetietojen ylläpidon käyttöliittymän kautta tuotetietojen ylläpitoprosesseihin. Järjestelmän integraatiokomponenteilla voidaan integroida MDM CS -järjestelmä toisiin tietojärjestelmiin palvelulähtöisen arkkitehtuurin[17] (engl. Service-Oriented Architecture, SOA) suunnittelumallin mukaisesti. (IBM 2013c.)

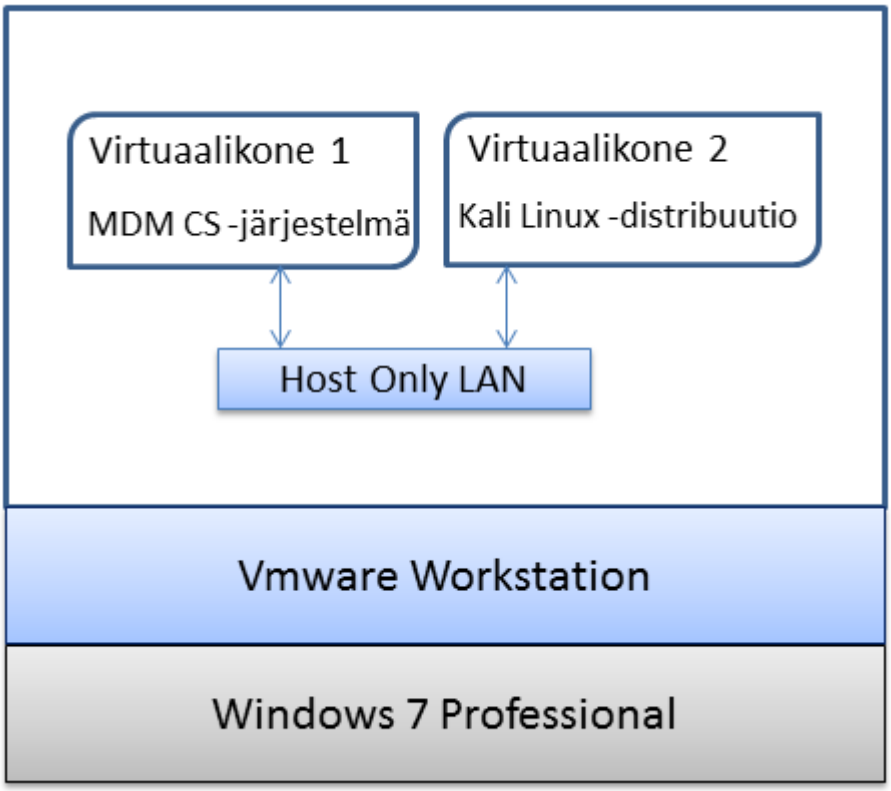
3.2 Virtualisoidun testausympäristön rakentaminen

Opinnäytetyössä käytetty virtualisoitu testausympäristö (kuvio 3) rakennettiin Lenovo W520-kannettavalle tietokoneella, jonka kokoonpanoon kuului neljällä suoritinytimellä varustettu Intel Core i7-2670QM 2,20 GHz -prosessori, 8 Gt keskusmuistia ja 500 Gt kiintolevy (liite 1). Lenovo W520-kannettava tietokoneen käyttöjärjestelmänä oli Windows 7 Professional SP1 64-bittinen versio. Virtualisointikerroksena käytettiin VMware Workstation 8 -ohjelmistoa. Testiympäristö koostui kahdesta VMware Workstation -virtuaalikoneesta, jotka olivat kytkettyinä samaan suljettuun virtuaaliseen lähiverkkoon[18].

[16] Monitasoisessa (engl. n-tier) ohjelmistoarkkitehtuurissa ohjelmistojen samantyyppiset toiminnot ryhmitellään erillisiin loogisiin kokonaisuuksiin (engl. layer), jotka toteutetaan toisistaan erillään fyysisen infrastruktuurin tasolla (engl. tier). Lisätietoja monitasoarkkitehtuurista on Microsoft Application Architecture Guide -verkkojulkaisussa osoitteessa http://msdn.microsoft.com/en-us/library/ee658109.aspx.

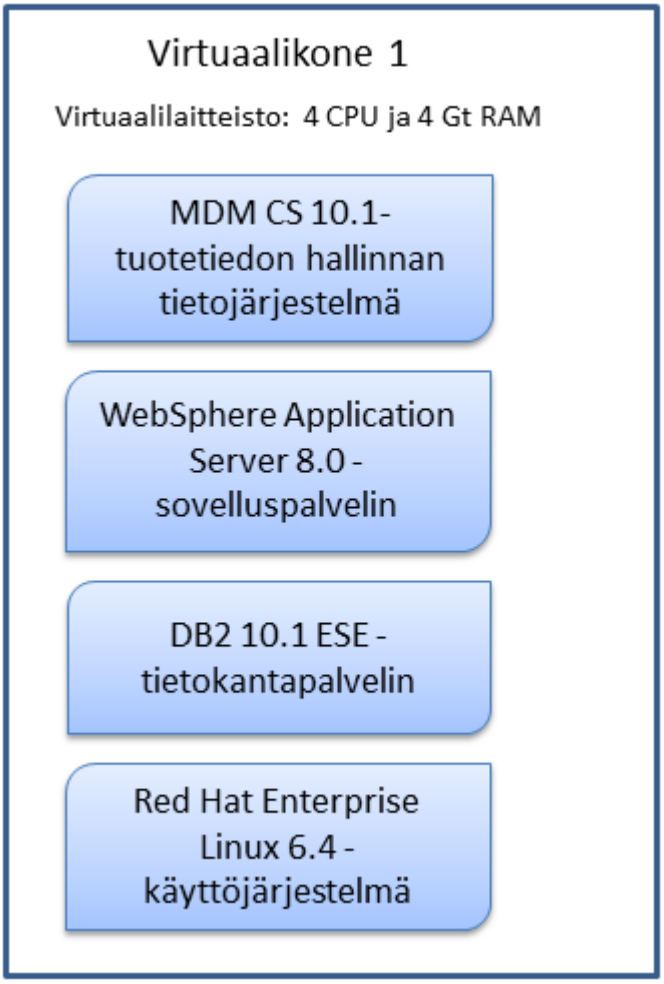
[17] Palvelukeskeisessä arkkitehtuurissa erilaiset tietojärjestelmät ovat vuorovaikutuksessa avoimien ja joustavien palvelujen välityksellä. MDM CS -järjestelmä tukee SOA-ratkaisuissa yleisesti käytettyjä integraatiotekniikoita kuten Web Service -palvelupyyntöjä. Lisätietoja palvelukeskeisestä arkkitehtuurista on OASIS-konsortion SOA-arkkitehtuurin referenssimalli -julkaisussa osoitteessa http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html.

[18] VMware Workstation -ohjelmiston Host Only -verkkoasetus, jolla virtuaalikone kytketään suljettuun lähiverkkoon. Virtuaalikone voi olla yhteydessä vain toisiin samaan lähiverkkoon kytkettyyn virtuaalikoneeseen ja VMware Workstation -isäntäkoneeseen. Suljettu lähiverkko on eristetty isäntäkoneen fyysisestä lähiverkosta ja Internetverkosta. Lisätietoa VMware-verkkoasetuksista osoitteessa http://kb.vmware.com/kb/1006480.



Kuvio 3. Virtualisoidun testiympäristön arkkitehtuurin kuvaus.

Ensimmäiseen virtuaalikoneeseen (kuvio 1) oli asennettuna MDM CS -järjestelmä, jota käytettiin sovellusturvallisuuden testauksessa kohdejärjestelmänä. Virtuaalikone oli tehty aikaisemmin VMware ESXi -palvelimella ja muunnettu VMware Workstation -tyyppiseksi virtuaalikoneeksi VMware vCenter Converter -työkaluohjelmistolla. Virtuaalikoneeseen oli asennettu Red Hat Enterprise Linux 6.4 -käyttöjärjestelmä, WebSphere Application Server 8.0 -sovelluspalvelin, DB2 10.1 Enterprise Server Edition -tietokantapalvelin sekä tuotetietojen hallinnan tietojärjestelmä MDM CS 10.1.



Kuvio 4. Virtuaalikone 1 komponenttien kuvaus.

Toinen virtuaalikone (kuvio 2) oli valmiiksi asennettu Kali Linux 1.0.3 -distribuutio, jonka sisältämiä verkkoturvallisuusohjelmistoja käytettiin sovellusturvallisuuden testaustyökaluina. Kali Linux distribuution valmiiksi asennettu VMware Workstation -virtuaalikone[19] ladattiin Kali Linux -projektin websivustolta ja se purettiin Lenovo W520 -tietokoneen kiintolevylle. Virtuaalikone otettiin käyttöön avaamalla sen VMX-asetustiedosto VMware Workstation -ohjelmistossa.

[19] VMware-virtuaalikone on pakattu tiedostoon kali-linux-1.0-i386-gnome-vm.tar.gz. Paketin purkamisessa tarvitaan gzip- ja tar-työkaluohjelmia (Linux) tai 7-Zip -ohjelmaa (Windows). Kali Linux on saatavilla myös ISO-tiedostona (live-CD), josta se voidaan asentaa kiintolevylle tai USB-muistitikulle. Kali Linux ladattavat tiedostot löytyvät osoitteesta http://www.kali.org/downloads.

Kuvio 5. Virtuaalikone 2 komponenttien kuvaus.

3.3 Avoimen lähdekoodin verkkoturvallisuusohjelmistot testaamisen välineinä

Opinnäytetyöprojektissa käytettiin sovellusturvallisuuden testaamisessa avoimen lähdekoodin verkkoturvallisuusohjelmistoja, koska ne ovat saatavilla ilmaiseksi Internet-verkon välityksellä ja ne ovat tekniseltä laadultaan korkeatasoisia verrattuna vastaaviin kaupallisiin ohjelmistotuotteisiin. Monia avoimen lähdekoodin verkkoturvallisuusohjelmistoja päivitetään hyvin aktiivisesti ja niiden kehittäjinä on usein kokeneita tietoturva-alan konsultteja, jotka seuraavat tiiviisti alan kehitystä ja uusimpia trendejä.

Yleensä avoimen lähdekoodin verkkoturvallisuusohjelmistot soveltuvat parhaiten rajatun ohjelmistotekniikan alueen tai haavoittuvuustyypin tutkimiseen[20]. Saatavilla on myös laajoja ja monipuolisia penetraatiotestausohjelmistoja, joilla voidaan suorittaa useita erityyppisiä testejä sekä automatisoituja koko websovelluksen haavoittuvuusskannauksia[21]. Automatisoitu testaus perustuu yleisten haavoittuvuuksien tuntomerkkien etsimiseen ja muistuttaa hieman PC-työaseman virustorjuntaohjelmiston virustarkistusta.

[20] Esimerkiksi SQL-injektioiden testaamiseen, salasanatiivisteiden murtamiseen ja fuzz-testaukseen on saatavilla pitkälle kehitettyjä erikoistyökaluohjelmia. OWASP Testing Guide 3.0 -dokumentin liitteessä Appendix A on lueteltuja penetraatiotestauksessa yleisesti käytettyjä ohjelmistoja (OWASP 2009b, 338).
[21] Laajat penetraatiotestausohjelmistot rakentuvat monista testauskomponenteista, jotka ovat integroituja samaan käyttöliittymään. Ohjelmistojen perustoiminnallisuuksiin kuuluu kaappaava HTTP proxy -palvelin, jonka avulla websovelluksen sisään ja ulos menevää HTTP-liikennettä pystytään manipuloimaan reaaliaikaisesti. Laajoja penetraatiotestausohjelmistoja ovat esimerkiksi Burp Suite ja OWASP Zed Attack Proxy.

Opinnäytetyöprojektissa verkkoturvallisuusohjelmistojen testausalustaksi valittiin tietoturva-alan asiantuntija- ja koulutuspalveluita tarjoavan Offensive Security[22] -yrityksen konsulttien kehittämä Kali Linux[23] -distribuutio, joka sisältää lukuisia avoimen lähdekoodin verkkoturvallisuusohjelmistoja. Kali Linux perustuu standardiin Debian GNU/Linux -distribuution ja se toimii kuten normaali Linux-käyttöjärjestelmä. Kali Linux voidaan asentaa tietokoneen kiintolevylle, USB-muistitikulle tai virtuaalikoneeseen. Sitä voidaan käyttää myös live-CD:ltä ilman kiinteää asennusta. (Offensive Security 2013).

OWASP-sovellusturvallisuuden arviointi- ja testausmetodologia on lähtökohtaisesti teknologiasta riippumatonta. Sekä kaupallisia että avoimen lähdekoodin ohjelmistoja voidaan käyttää OWASP-metodologian mukaisessa sovellusturvallisuuden testauksessa. Käytettävien ohjelmistojen lisensointimallia tärkeämpää on suoritettavan testauksen kattavuus ja tarkkuus. Avoimen lähdekoodin verkkoturvallisuusohjelmistot soveltuvat hyvin OWASP-metodologian hyödyntämiseen, koska kaikki OWASP-projektit jakavat avoimen lähdekoodin yleiset periaatteet: avoimuuden, ilmaisuuden ja yhteisöllisyyden. Avoimen lähdekoodin ohjelmistojen käyttämistä tukee myös niiden esiintyminen useissa OWASP Testing Guide 3.0 -dokumentin käytännön testausesimerkeissä. (OWASP 2008, 13, 18-19).

[22] Offensive Security -yritys tarjoaa penetraatiotestauspalveluita ja -konsultointia yritysasiakkaille sekä järjestää tietoturva-alan ammattilaisille suunnattuja maksullisia kursseja penetraatiotestauksen tekniikoista. Kursseja järjestetään sekä verkkototeutuksina että ohjaajan vetämänä lähiopetuksena. Kursseihin kuuluu aiheeseen liittyvän Offensive Security -sertifikaatin suorittaminen, joka voidaan liittää osaksi tietoturva-alan yleisesti tunnettuja SSCP- tai CISSP-sertifikaattien opintoja. Offensive Security:n toimintaan kuuluu myös julkisia yhteisöprojekteja kuten haavoittuvuustietokannat ExploitDB ja Google Hacking Database, avoin verkkokurssi Metasploit Unleashed haavoittuvuuksien ohjelmoinnista Metasploit-työkalulla sekä käytännön penetraatiotestausohjelmistoja sisältävä Kali Linux -distribuutio. Offensive Security -yrityksen websivusto on osoitteessa http://www.offensive-security.com.

[23] Kali Linux on digitaaliseen rikosteknisen tutkimuksen ja penetraatiotestauksen käyttötarpeisiin suunniteltu Linux-distribuutio. Kali Linux julkaistiin virallisesti Black Hat Europe 2013 -tietoturvakonferenssin yhteydessä maaliskuussa 2013. Se on uudistettu versio Offensive Security:n aikaisemmasta BackTrack Linux -distribuutiosta. Kali Linux -distribuution websivusto on osoitteessa http://www.kali.org ja lisätietoa Kali Linux:n mukana toimitettavista verkkoturvallisuusohjelmistoista on Kali Linux GIT-versionhallinnan koodivarastoissa osoitteessa http://git.kali.org/gitweb.

3.4 Testitapausten suunnitteleminen OWASP-metodologian mukaisesti

Opinnäytetyössä suoritettavan sovellusturvallisuuden testauksen lähtökohdaksi otettiin OWASP ASVS -standardin tason 1A arviointiin kuuluvat yksityiskohtaiset tietoturvavaatimukset, koska ne soveltuivat parhaiten opinnäytetyön kokonaisuuteen. Pitemmälle menevä testaus ei olisi ollut mahdollista opinnäytetyöprojektin puitteissa. Testaamisessa ei käytetty valkoinenlaatikko-menetelmiä, koska opinnäytetyössä testatun MDM CS -ohjelmiston lähdekoodia ei ole saatavilla. Testaamisen helpottamiseksi käytettiin kuitenkin tarvittaessa apuna ohjelmiston julkista dokumentaatiota. OWASP ASVS ja Testing Guide -dokumentaatioita käytettiin soveltaen testitapausten suunnittelun ja testien suorittamisen taustamateriaalina.

OWASP ASVS -standardin tason 1A tietoturvavaatimukset jakaantuvat seuraaviin kategorioihin: sovelluksen tietoturva-arkkitehtuurin dokumentointi (V1), autentikointimekanismien toiminta (V2), istuntojenhallinta (V3), pääsynvalvonta (V4), syötteiden validointi (V5), virheiden käsittely (V8), tiedon suojaaminen (V9), tietoliikenteen turvallisuus (V10) ja HTTP-liikenteen turvallisuus (V11). Tietoliikenteen turvallisuus (V10) kategoria jätettiin kokonaan pois opinnäytetyöprojektiin kuuluvasta testauksesta, koska se liittyy suojattujen verkkoyhteyksien SSL-sertifikaattien testaamiseen ja testattava ohjelmisto ei käytä HTTPS-protokollan mukaista suojattua verkkoyhteyttä.

Testikategorioiden pääsynvalvonta (V4), syötteiden validointi (V5), virheiden käsittely (V8), tiedon suojaaminen (V9) ja HTTP-liikenteen turvallisuuden (V11) testitapaukset suoritettiin vain osittain tai ne jäivät kokonaan testauksen ulkopuolelle. Kaikkien testien suorittamista ei katsottu tarpeelliseksi, koska tutkimuksen ensisijaisena tavoitteena oli proof of concept -tyyppisen selvityksen tekeminen. Taulukossa 4 on kuvattu tutkimuksen suunnittelussa käytetyt testitapaukset, jotka on numeroitu OWASP ASVS -testikategorioiden mukaisessa järjestyksessä. Testauksen ulkopuolelle jääneet testitapaukset on merkitty taulukossa yliviivattuina.

Taulukko 4. OWASP ASVS -standardin tason 1A mukaiset testitapaukset.

Sovellusturvallisuuden testitapaukset		
Tunniste	Testitapauksen kuvaus	Testikategoria
1.1	Sovelluksen komponenttien tunnistaminen ja ryhmittely yleisellä tasolla	Sovelluksen tietoturva-arkkitehtuuri
2.1	Sovelluksen kaikkien resurssien (websivut) käyttäminen tulisi vaatia käyttäjän autentikointia	Autentikointimekanismit
2.2	Kirjautumisnäkymän salasanakenttään syötettäviä merkkejä ei tule näyttää käyttöliittymässä ja salasanakentän sisällön tallentaminen webselaimen välimuistiin tulee estää	Autentikointimekanismit
2.3	Käyttäjätunnus lukitaan väliaikaisesti useiden epäonnistuneiden kirjautumisyritysten jälkeen	Autentikointimekanismit
3.1	Sovellus käyttää ohjelmakirjaston tarjoamaa valmista istuntojenhallinnan kontrollimekanismia	Istuntojenhallinta
3.2	Sovellus tuhoaa käytetyn istunnon kun käyttäjä kirjautuu ulos	Istuntojenhallinta
3.3	Käyttämättömät istunnot poistetaan tietyn ajan kuluttua, jos käyttäjä ei ole vuorovaikutuksessa sovelluksen kanssa	Istuntojenhallinta
3.4	Kaikilla autentikointia vaativilla sovelluksen resursseilla (websivuilla) tulee olla uloskirjautumisen toiminnallisuus	Istuntojenhallinta
~~4.1~~	~~Vain määrätyn käyttöoikeustason omaavilla käyttäjillä on pääsy sovelluksen suojattuihin toimintoihin~~	~~Pääsynvalvonta~~
~~4.2~~	~~Käyttäjällä on pääsy vain hänen käyttöoikeuksiensa edellyttämiin URL-~~	~~Pääsynvalvonta~~

	~~osoitteisiin~~	
~~4.3~~	~~Käyttäjällä on pääsy vain hänen käyttöoikeuksiensa edellyttämiin datatiedostoihin~~	~~Pääsynvalvonta~~
~~4.4~~	~~Suorat viittaukset sovelluksen objekteihin on suojattu käyttöoikeustasojen mukaisesti~~	~~Pääsynvalvonta~~
4.5	Websovelluksen hakemistopolun selaaminen on estetty	Pääsynvalvonta
5.1	Sovellus ei ole haavoittuvainen puskuriylivuodoille	Syötteiden validointi
~~5.2~~	~~Syötteiden validoinnissa käytetään sallittujen syötteiden suodatusta (whitelist)~~	~~Syötteiden validointi~~
~~5.3~~	~~Syötteiden validoinnin virhetilanteissa virheellinen syöte hylätään tai tehdään vaarattomaksi~~	~~Syötteiden validointi~~
~~8.1~~	~~Sovellus ei paljasta virheilmoituksissa arkaluonteista tai hyökkäyksiä helpottavaa tietoa kuten käyttäjän henkilötietoja tai istuntojen tunnisteita~~	~~Virheiden käsittely~~
~~9.1~~	~~Sovelluksen käyttöliittymä estää arkaluonteista tietoa sisältävien lomakekenttien tallentamisen webselaimen välimuistiin~~	~~Tiedon suojaaminen~~
~~11.1~~	~~URL-osoitteiden uudelleenohjauksessa ei käytetä validoimatonta dataa~~	~~HTTP-liikenteen turvallisuus~~
11.2	Sovellus hyväksyy vain yleisesti käytetyt HTTP-pyynnöt kuten GET ja POST -metodit	HTTP-liikenteen turvallisuus
11.3	Sovelluksen HTTP-vastausten otsakkeissa määritellään turvallinen merkistökoodaus kuten UTF-8	HTTP-liikenteen turvallisuus

Sovelluksen tietoturva-arkkitehtuurin dokumentointiin kuuluu sovelluksen sisäisten komponenttien tunnistaminen ja ryhmittely yleisellä tasolla (OWASP 2009b, 6,17). Autentikointimekanismien testauksella pyritään varmistamaan autentikoinnin toiminta sovelluksessa, niin että se on käytössä kaikki käyttäjän tunnistamista tarvitsevilla käyttöliittymän resursseilla. Käyttöliittymän kirjautumisnäkymästä testataan salasanakenttien[24] toiminta sekä brute force[25] -tyyppiseen hyökkäykseen varautuminen. (OWASP 2009b, 18). Istunnonhallinnan testaamisessa selvitetään sovelluksen käyttämä istuntojenhallinnan kontrollimekanismi[26], istuntojen tuhoaminen uloskirjautumisen yhteydessä ja käyttämättömien istuntojen automaattinen sulkeminen. (OWASP 2009b, 19.)

Pääsynvalvonnan testauksella tarkistetaan, että käyttäjillä on pääsy vain heidän käyttöoikeustasonsa edellyttämiin sovelluksen resursseihin ja toiminnallisuuksiin[27]. (OWASP 2009b, 21). Syötteiden validoinnin osalta testataan sovelluksen haavoittuvuus puskuriylivuotohyökkäyksille[28] (engl. buffer overflow) ja virheellisten syötteiden käsittelymekanismit. (OWASP 2009b, 22). Virheiden käsittelystä tarkistetaan, että sovellus ei paljasta liikaa sisäistä informaatiota tulostamissaan virheilmoituksissa. (OWASP 2009b, 25). Tiedon suojaamisen testauksessa varmistetaan, että websovellus estää käyttäjän syötettämän arkaluontoisen tiedon tallentamisen webselaimen välimuistiin. (OWASP 2009b, 19). HTTP-liikenteen turvallisuuteen kuuluu URL-uudelleenohjauksessa käytettävien osoitteiden sisältämän datan validointi, epätavallisten HTTP-pyyntöjen estämi-

[24]Tietoturvallisen ohjelmointikäytännön mukaan websovelluksen kirjautumisikkunassa salasanakenttään syötetty teksti tulee piilottaa käyttäjältä ja estää sen tallentaminen webselaimessa automaattisen täydentämisen toiminnallisuuden kautta.

[25] Brute force -hyökkäyksessä käyttäjätunnus yritetään murtaa syöttämällä kaikki mahdolliset salasanayhdistelmät. Brute force -hyökkäyksiä varten on tehty työkaluohjelmia kuten THC Hydra ja Brutus, joilla salasanayhdistelmien generoiminen ja salasanojen syöttäminen käyttöliittymään on automatisoitu. Websovelluksen tulisi varautua brute force -hyökkäykseen esim. lukitsemalla käyttäjätunnus väliaikaisesti epäonnistuneiden kirjautumisyritysten jälkeen.

[26] Ohjelmistokehityksessä käytetään yleisesti valitulle kehitysympäristölle suunnattuja laajoja ohjelmointikirjastoja (engl. framework, esim. MS.NET tai yleiset Java-kirjastot kuten Spring, Hibernate jne.), jotka sisältävät myös istuntojenhallinnan kontrollitoiminnallisuudet. Websovelluksessa tulisi käyttää ensisijaisesti ohjelmointikirjaston tarjoamia valmiita istuntojenhallinnan kontrollimekanismeja.

[27] Sovelluksen resursseilla tarkoitetaan tässä käyttöliittymän osia, toimintoja ja dataa. Suorat viittaukset sovelluksen objekteihin kuten URL-osoitteisiin tulee olla myös suojattuja, jotta käyttäjä ei voi manuaalisesti ohittaa sovelluksen tietoturvakontrolleja.

[28] Puskuriylivuoto johtuu sovelluksen ohjelmointivirheestä, joka mahdollistaa tietokoneen keskusmuistin ylikirjoittamisen. Esimerkiksi kun sovellukselle annetaan syötteenä liian paljon dataa ja sovelluksen tietokoneen keskusmuistista varaama muistialue loppuu, niin virheellisesti ohjelmoitu sovellus ylikirjoittaa keskusmuistia oman muistialueensa ulkopuolella. Puskuriylivuotohyökkäyksessä sovelluksen muistinkäsittelyn virheellisyyttä pyritään hyödyntämään haitallisen ohjelmakoodin suorittamiseen. Lisätietoja aiheesta löytyy osoitteesta https://www.owasp.org/index.php/Buffer_Overflow.

nen ja turvallisen merkistökoodauksen ilmoittaminen HTTP-vastauksissa. (OWASP 2009b, 28.)

3.5 Testauksen tarkempi rajaus ja testijärjestelyjen kuvaus

Testijärjestelyt suunniteltiin opinnäytetyöprojektia varten rakennetun testiympäristön mukaisesti. Testattava InfoSphere MDM Collaboration Server -ohjelmisto oli asennettu IBM:n tuotedokumentaation mukaisesti (IBM 2013b). WebSphere-sovelluspalvelin ja DB2-tietokanta oli asennettu samaan Linux-virtuaalikoneeseen niiden tuotedokumentaation[29] sekä MDM CS -dokumentaation mukaisesti. MDM CS -ohjelmistoa ja muita IBM-ohjelmistoja ei konfiguroitu erityisen tietoturvallisesti, vaan pääsääntöisesti käytettiin IBM-dokumentaatiossa annettuja yleisiä oletusarvoja. Testattavan MDM CS -ohjelmiston alustana toimiva Linux-virtuaalikoneen käyttöjärjestelmä Red Hat Enterprise Linux 6.4 asennettiin ja konfiguroitiin myös oletusasetusten mukaisesti. Poikkeuksena tavanomaiseen asennukseen graafinen käyttöliittymä X Windows ja siihen kuuluvat komponentit jätettiin asentamatta sekä Linuxin tilallinen palomuuri iptables otettiin pois käytöstä.

Sovellusturvallisuuden testaaminen suoritettiin virtualisoidussa testiympäristössä, niin että testauksessa käytetyt virtuaalikoneet olivat käynnistettyinä ja virtuaalikoneita käytettiin VMware Workstation -ohjelmiston virtuaalikoneiden konsolikäyttöliittymästä. Mustalaatikko-menetelmään pohjautuva penetraatiotestaus suoritettiin Kali Linux -virtuaalikoneesta käsin (kuvio 3). Testaamisessa käytettiin myös MDM CS -järjestelmän alustana toimivaa Red Hat Linux -virtuaalikonetta sovelluksen tietoturva-arkkitehtuurin kartoittamiseen sekä MDM CS -järjestelmän teknisten tietojen selvittämiseen (kuvio 4).

[29] WAS 8.0 ND Info Center -dokumentaatio osoitteessa http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/welcome_ndmp.html ja DB2 10.1 Info Center -dokumentaatio osoitteessa http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp.

Taulukko 5. Testauksessa käytetyt työkaluohjelmat ja testien tarkempi rajaus.

Testitapaus	Käytetty työkaluohjelma	Testin tarkempi rajaus
1.1	–	Tietoturva-arkkitehtuurin kartoittamisessa käytettiin MDM CS Info Center -dokumentaatiota
2.1	OWASP ZAP	Testaus rajoittui yksittäiseen esimerkkitapaukseen autentikointia vaativasta sovelluksen resurssista. Testattava HTTP-pyyntö lähettiin manuaalisesti sovellukselle OWASP ZAP -työkalulla.
2.2	OWASP ZAP, IceWeasel-selain	Sovelluksen kirjautumisikkunaa testattiin vain yhdellä selaimella (IceWeasel). Kirjautumisikkunan HTML-koodia tutkittiin OWAS ZAP -työkalussa.
2.3	Burp Suite (Intruder-työkalu)	Testauksessa tehtiin simuloitu brute force -hyökkäys Burp Suite ohjelmiston Intruder-työkalulla. Simuloitu brute force -hyökkäys rajoitettiin hyvin lyhyeksi (78 kirjautumisyritystä) ja merkkijonojen generoinnissa käytettiin vain alfanumeerisia merkkejä.
3.1	w3af	Sovelluksen istuntojenhallinnan kontrollimekanismin selvittäminen tehtiin vain yhdellä haavoittuvuusskannausohjelmistolla (w3af).
3.2	OWASP ZAP	Käytettyjen istuntojen sulkemista testattiin käyttämällä OWASP ZAP -ohjelmiston HTTP Sessions -toimintoa.
3.3	OWASP ZAP	Käyttämättömien istuntojen sulkemista testattiin lähettämällä HTTP-pyyntöjä sovellukselle OWASP ZAP -ohjelmistosta. MDM CS -asetuksia muutettiin ennen testiä, jotta testaaminen voidaan suorittaa nopeammin.
3.4	IceWeasel-selain	Sovelluksen uloskirjautumisen toimintoa testattiin vain käyttämällä sovellusta selaimella (IceWeasel). Uloskirjautumismekanismin toimintaa ei tutkittu yksi-

		tyiskohtaisella tasolla.
~~4.1~~	–	Sovelluksen resurssien autorisointimekanismin toimintaan liittyviä testejä ei suoritettu, koska testien suorittaminen mustalaatikko-menetelmällä ei vaikuttanut tarkoituksen mukaiselta. Testien suorittaminen olisi vaatinut käytännössä MDM CS -tietomallin laajentamista sekä testaamista järjestelmään kirjautuneella käyttäjällä.
~~4.2~~	–	
~~4.3~~	–	
~~4.4~~	–	
4.5	OpenVAS	Sovelluksen hakemistopolun selaamista testattiin OpenVAS-ohjelmiston Full and very deep ultimate -asetusten mukaisella haavoittuvuusskannauksella.
5.1	Burp Suite (Intruder-työkalu)	Sovelluksen haavoittuvuutta puskuriylivuodoille tutkittiin Burp Suite -ohjelmiston Intruder-työkalulla lähettämällä sovellukseen sisään kirjautumista yrittäviä HTTP-pyyntöjä, joiden parametriksi annettiin ylipitkiä merkkijonoja. Puskuriylivuotoja ei tutkittu muualla sovelluksen käyttöliittymässä.
~~5.2~~	–	Sallittujen syötteiden suodatukseen ja virheellisten syötteiden käsittelyyn liittyviä testejä ei suoritettu, koska testien suorittaminen mustalaatikko -menetelmällä ei vaikuttanut tarkoituksen mukaiselta. Valmisohjelmiston syötteiden käsittelymekanismia voidaan tutkia vain epäsuorasti. Testaaminen olisi käytännössä pitänyt tehdä järjestelmään kirjautuneena käyttäjänä.
~~5.3~~	–	
~~8.1~~	–	Virheiden käsittelyn testaamista ei suoritettu, koska testien suorittaminen mustalaatikko-menetelmällä ei vaikuttanut tarkoituksen mukaiselta. Virheiden käsittelyn kunnollinen testaaminen olisi käytännössä pitänyt tehdä järjestelmään kirjautuneena käyttäjänä.
~~9.1~~	–	Virheiden käsittelyn testaamista ei suoritettu, koska testien suorittaminen mustalaatikko-menetelmällä ei

		vaikuttanut tarkoituksen mukaiselta. Testitapauksessa 2.1 tutkittiin kirjautumissivun salasanakentän suojaamista. Muuten testaaminen olisi vaatinut sisään kirjautumista järjestelmään.
~~11.1~~	–	URL-osoitteiden uudelleenohjausta ei testattu, koska OWASP-dokumentaatiossa ei ollut selkeää kuvausta miten testaaminen pitäisi suorittaa. Testaaminen olisi edellyttänyt kirjautumista sisään järjestelmään. URL-uudelleenohjausten tutkimisessa olisi myös tarvittu runsaasti sovelluksen HTTP-liikenteen manuaalista tutkimista OWASP ZAP tai Burp Suite -ohjelmistoilla.
11.2	w3af, OpenVAS	Sallittujen HTTP-metodien käyttöä tutkittiin w3af ja OpenVAS -ohjelmistoilla suoritetuilla haavoittuvuusskannauksilla.
11.3	Burp Suite (Proxy-työkalu)	Sovelluksen HTTP-otsakkeiden merkistökoodausta tutkittiin Burp Suite -ohjelmiston Proxy-työkalulla.

3.6 Testaamisen toteutuksen kuvaus

Testit suoritettiin niin, että molemmat virtuaalikoneet olivat käynnistettyinä. Virtuaalikoneessa 1 (MDM CS -järjestelmä, kuvio 3) kaikki IBM-ohjelmistot olivat myös käynnistettyinä ja Linuxin tilallinen palomuuri iptables oli pois päältä. Virtuaalikoneesta 2 (Kali Linux, kuvio 4) oli käynnistettynä graafiseen käyttöliittymään (Gnome/X Windows) ja testauksessa käytetyt verkkoturvallisuusohjelmistot käynnistettiin komentoriviltä (Terminal) tai graafisesta ohjelmavalikosta Applications → Kali Linux. Yleisenä testausparametrina käytettiin kohdejärjestelmänä virtuaalikoneen 1 (MDM CS -järjestelmä) IP-osoitetta.

Testaaminen aloitettiin MDM CS -järjestelmän tietoturva-arkkitehtuurin kartoittamisella (Taulukko 5, testitapaus 1.1). Testausta suoritettiin määriteltyjen testitapausten mukaisessa järjestyksessä. Testauksen tulokset tallennettiin tapauskohtaisesti joko tekstimuotoisiin tiedostoihin verkkoturvallisuusohjelmien tulosteista tai virtuaalikoneesta

otettuihin kuvankaappauksiin (liite 2). Testaustyökaluina käytettiin autentikointimekanismien (Taulukko 5, testitapaukset 2.1 ja 2.2) testaamisessa OWASP Zed Attack Proxy[30] -ohjelmistoa, joka toimii HTTP-liikenteen kaappaavana proxy-palvelimena. Testauksessa Kali Linuxin webselain Iceweasel[31] 18.0.1 asetettiin käyttämään OWASP ZAP -ohjelmiston proxy-palvelinta localhost osoitteen portissa 8080 ja webselaimeen asennettiin OWASP ZAP -ohjelmiston generoima SSL-sertifikaatti. MDM CS -sovellusta käytettiin Iceweasel-webselaimessa, jolloin kaikki HTTP-liikenne selaimen ja sovelluksen välillä kulki OWASP ZAP -ohjelmiston kautta.

Sovelluksen autentikointimekanismien toimintaa brute force -tyyppisen hyökkäyksen (Taulukko 5, testitapaus 2.3) aikana testattiin Burp Suite[32] -ohjelmiston Intruder-työkalulla. Burp Suite toimii HTTP-liikenteen kaappaavana proxy-palvelimena samalla tavalla OWAS ZAP -ohjelmisto. Kirjautumisen välittävä HTTP-pyyntö kaapattiin Intruder-työkalun käyttöön. Intruder-työkalu konfiguroitiin tekemään brute force -hyökkäys, jossa kaikkia alfanumeerisia merkkiyhdistelmiä kokeiltiin HTTP-pyynnön salasanakenttään ja MDM CS -sovellukselle lähettiin useita kymmeniä erilaisilla salasanakentän arvoilla varustettuja HTTP-pyyntöjä.

Sovelluksen istuntojenhallinnan kontrollimekanismia tutkittiin w3af-ohjelmiston[33] haavoittuvuusskannauksen avulla. Istuntojenhallintaa tutkittiin myös OWASP ZAP ja Burp Suite -ohjelmistoilla. Sovelluksen pääsynvalvontaa testattiin hakemistopolun selaamisen osalta käyttäen OpenVAS[34] -ohjelmiston haavoittuvuusskannausta, joka käyt-

---

[30] OWASP Zed Attack Proxy on integroitu penetraatiotestausohjelmisto, jolla voidaan suorittaa automaattista ja manuaalista sovellusturvallisuuden testausta useilla eri työkaluilla. OWAS ZAP -projektin nettisivut ovat osoitteessa https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[31] Iceweasel on Debian-projektin julkaisema versio Mozilla Firefox selaimesta, josta on poistettu Mozilla-säätiön tekijänoikeuksilla suojatut tavaramerkit. Iceweasel julkaistiin Debian-projektin ja Mozilla-säätiön toisistaan poikkeavien lisenssikäytäntöjen aiheuttaman kiistan takia vuonna 2006. Lisätietoja Iceweasel-selaimesta löytyy osoitteesta http://wiki.debian.org/Iceweasel.

[32] Burp Suite on integroitu penetraatiotestausohjelmisto, joka soveltuu automaattiseen ja manuaaliseen sovellusturvallisuuden testaamiseen. Burp Suite Professional on kaupallinen ohjelmisto, josta on myös saatavilla ilmaisversio Free Edition. Opinnäytetyöprojektissa käytettiin ohjelmiston ilmaisversiota. Burp Suite -ohjelmiston nettisivut ovat osoitteessa http://www.portswigger.net/burp.

[33] w3af on websovellusten haavoittuvuusskannaukseen tarkoitettu ohjelmisto, jota voi käyttää myös kaappaavana HTTP proxy-palvelimena ja sillä voidaan myös käynnistää hyökkäyksiä löydettyjä haavoittuvuuksia vastaan. w3af-ohjelmiston nettisivut ovat osoitteessa http://w3af.org.

[34] OpenVAS on avoimen lähdekoodin haavoittuvuusskannaukseen tarkoitettu ohjelmisto, joka on kehitetty tunnetun Nessus-ohjelmiston pohjalta. OpenVAS-ohjelmiston kehitystä tukee saksalainen yritys Greenbone Networks Gbmh. OpenVAS-ohjelmiston nettisivut löytyvät osoitteesta http://www.openvas.org.

tää useita kolmannen osapuolen skannausohjelmia[35] perusteellisen haavoittuvuusskannauksen suorittamiseen. Sovelluksen toimintaa puskuriylivuotohyökkäyksen aikana testattiin Burp Suite -ohjelmiston Intruder-työkalulla. Sovelluksen sallimia HTTP-pyyntöjä testattiin w3af ja OpenVAS -ohjelmistojen haavoittuvuusskannauksilla.

Testaamisessa ei käyty systemaattisesti läpi kaikki sovelluksen resursseja[36] kuten aidossa tuotantojärjestelmän penetraatiotestauksessa, vaan testejä suoritettiin valikoidusti yksittäisille sovelluksen websivuille ja muille resursseille. Testauksessa ei tarkastettu kaikkia OWASP ASVS -tason 1A yksityiskohtaisia tietoturvavaatimuksia.[37] Tarkoituksena oli kuvata testien suorittamisen käytäntöä proof of concept -näkökulmasta. Testauksen aikana otetuista kuvankaappauksista piilotettiin osa sovelluksen käyttöliittymässä näkyvää informaatiota tietoturvasyiden takia.

3.7 Testitulosten raportointi

Sovellusturvallisuuden testauksen lopputulokset on esitetty liitteessä 2. Testauksen lopputulokset on käyty läpi testitapaus kerrallaan. Testauksen suorittaminen ja saadut tulokset on kuvattu lyhyesti jokaisen testitapauksen osalta. Testitapauksiin on myös liitetty testauksen aikana otetut kuvakaappaukset Kali Linux -virtuaalikoneesta (liite 2). Testien suorittamista ja niissä käytettyjä ohjelmistoja on kuvattu alaluvuissa 3.5. ja 3.6. Sovellusturvallisuuden testauksen loppuraportti on esitetty taulukossa 6.

[35] OpenVAS koostuu useista moduuleista, jotka kutsuvat mm. nikto, wapiti ja arachni -skannausohjelmia.
[36] Esimerkiksi testitapauksessa 2.1 käytettiin vain yhtä sovelluksen sisäistä URL-osoitetta autentikointimekanismin testaamiseen. Tuotantoympäristön penetraatiotestauksessa testauksessa olisi pitänyt kartoittaa kaikki sovelluksen sisäiset resurssit ja testata yksitellen jokaisella autentikointimekanismin toiminta.
[37] Sovelluksen pääsynvalvontaa, syötteiden validointia ja HTTP-liikenteen turvallisuutta testattiin vain osittain. Virheiden käsittelyn ja tiedon suojaamisen vaatimuksia ei testattu lainkaan, koska niille ei löytynyt sopivaa testitapausta mustalaatikkotestauksen näkökulmasta.

Taulukko 6. MDM CS -järjestelmän sovellusturvallisuuden testauksen raportti.

Testitapaus	Arvio sovellusturvallisuuden vaatimuksesta			Perustelut	Testitapauksen lisätiedot
	Täytetty	Osittain täytetty	Ei täytetty		
1.1 sovelluksen tietoturva-arkkitehtuuri	☑			Sovelluksen tietoturva-arkkitehtuuri pystytään kuvaamaan riittävällä tasolla	MDM CS -sovelluksen dokumentaatiossa oli kattava ja selkeä arkkitehtuurikuvaus.
2.1 sovelluksen resurssien autentikointi	☑			Sovelluksen resurssien autentikointia ei pystytä ohittamaan suoralla viittauksella URL-osoitteisiin	Testausta ei tehty täydessä laajuudessaan, koska siihen olisi tarvittu huomattavasti enemmän manuaalista työtä ja aikaa.
2.2 kirjautumisikkunan turvallisuus		☑		Kirjautumisikkunan salasanakentän sisältöä ei näytetä käyttäjälle. Salasanakentän sisällön tallentamista selaimen välimuistiin ei estetä.	
2.3 käyttäjätunnuksen väliaikainen lukitseminen			☑	MDM CS -sovellus ei lukitse käyttäjätunnusta lukuisien epäonnistuneiden kirjautumisyritysten	Testauksessa suoritettu simuloitu brute force -hyökkäys oli

				jälkeen.	hyvin pienimuotoinen, mutta kuitenkin riittävä tuloksen aikaansaamiseksi.
3.1 istuntojenhallinnan kontrollimekanismi	☑			MDM CS -sovellus käyttää istuntojenhallintaan Apache Tomcat -tyyppistä Java-toteutusta.	
3.2 sovelluksen käytettyjen istuntojen tuhoaminen	☑			MDM CS -sovellus tuhoaa istunnot uloskirjautumisen yhteydessä. Vanhan istuntotunnuksen hyväksi käyttäminen ei ole mahdollista.	
3.3 käyttämättömien istuntojen sulkeminen	☑			MDM CS -sovellus sulkee käyttämättömät istunnot asetuksissa määritellyn ajan kuluttua ja tuhoaa istunnot.	Testijärjestelyissä käytettiin apuna MDM CS -dokumentaatiota ja sovelluksen asetuksia muutettiin testaamisen helpottamiseksi.
3.4 sovelluksen uloskirjautumisen toiminnallisuus	☑			MDM CS -sovelluksen käyttöliittymässä on koko ajan selkeästi näkyvissä Log out -linkki.	Testaamisessa käytettiin vain manuaalisesti selainta. Testin kattavuutta ja tarkkuutta on vaikeaa arvioida.
4.5 websovel-				MDM CS -sovelluksen	

luksen hakemistopolun selaaminen			☑	hakemistopolkua pystyttiin selaamaan haavoittuvuusskannauksen yhteydessä.	
5.1 sovelluksen haavoittuvuus puskuriylivuodoille	☑			MDM CS -sovelluksen toiminta ei häiriintynyt simuloidussa puskuriylivuotohyökkäyksessä.	Testauksessa käytettiin vain sovelluksen sisään kirjautumisessa käytettäviä HTTP- pyyntöjä ja niiden parametreja. Testaus oli teknisesti yksinkertainen ja kattavuudeltaan hyvin suppea.
11.2 sovelluksen sallitut HTTP-pyynnöt			☑	MDM CS sovellus käsittelee ei-standardin mukaisia HTTP-metodeja GET-metodeina ja HTTP-metodi TRACE on sallittu.	
11.3. sovelluksen HTTP-otsakkeiden merkistökoodaus	☑			MDM CS -sovelluksen palauttamien HTTP-vastausten otsakkeissa on määritelty UTF-8 merkistökoodaus.	

Opinnäytetyön tapaustutkimuksen raportoinnin esitystavassa seurattiin pääpiirteissään OWASP ASVS -standardin mukaista raportointimallia, jonka mukaan testattavaa sovellusta ja sen toimintaympäristöä kuvattiin ensin yleisellä tasolla ja sen jälkeen tarkasteltiin

sovelluksen tietoturva-arkkitehtuuria yksityiskohtaisella tasolla. Sovelluksen tietoturvan tasoa arvioitiin OWASP ASVS -tason 1A mukaisten tietoturvavaatimusten kannalta.

OWASP-metodologian painotukset ovat ensisijaisesti sovellusturvallisuuden teknisissä lähtökohdissa ja raportointiohjeet ovat selkeästi vähäisemmässä asemassa molemmissa opinnäytetyössä tutkituissa OWASP-dokumenteissa. OWASP ASVS -standardin raportointiohje on hyvin suppea (37 sivun pituisesta dokumentista vain noin 2,5 sivua) ja esimerkiksi sovelluksen käyttöön liittyvien liiketoiminnallisten riskien arviointiin ei anneta tarkempia ohjeita (OWASP 2009b, 32). Raportoinnin tärkeimpänä osana pidetään yksityiskohtaisten tietoturvavaatimusten arviointia (OWASP 2009b, 33). Raportointiohjeen sijoittaminen OWASP ASVS -dokumentin loppuun jättää sen hieman irralliseksi osioksi, joka ei tue parhaimmalla tavalla OWASP ASVS -dokumentin käyttämistä liiketoiminnan kontekstissa.

OWASP-metodologian raportointiohjeissa kiinnittää huomiota niiden vaihtelevuus. OWASP Testing Guide -dokumentissa raportointi on OWASP ASVS -dokumentin tapaan sijoitettu epäedullisesti dokumentin loppuun (OWASP 2008, 333). Raportointiohjeen tarkemman sisällölliset painotukset ovat kuitenkin erilaisia OWASP Testing Guide -dokumentissa. Raportointiohje on siinä rakennettu enemmän liiketoiminnan lähtökohdasta, niin että johdanto-osuus on suunnattu liiketoiminnan johdolle sekä tekniselle johdolle ja vasta sitten siirrytään asiantuntijoille tarkoitettuun tekniseen osuuteen (OWASP 2008, 333). Raportin rakenteen malli on parempi kuin OWASP ASVS -standardissa, jossa johdanto-osuuden sisältöä ja kohderyhmää ei ole määritelty kovin tarkasti (OWASP 2009b, 32). OWASP Testing Guide -raportointiohjeen teknisen osuuden taulukkomalli (OWASP 2008, 333) on onnistuneempi verrattuna OWASP ASVS -standardin samankaltaiseen taulukkomalliin (OWASP 2009b, 32.)

OWASP-metodologian erilaiset raportointiohjeet luovat pulmallisen tilanteen sovellusturvallisuuden arviointiin. Kummassakin raportointimallissa on omat hyvät puolensa ja puutteensa. Kahden raportointimallin käyttäminen rinnakkain on kuitenkin työlästä ja hankalaa, joten sovellusturvallisuuden arvioinnissa on helpompaa käyttää vain yhtä raportointimallia. Raportointiohjeiden poikkeavuus toisistaan kärjistyy riskien hallinnan kohdalla. OWASP ASVS -standardissa riskien hallintaa ei käsitellä juuri lainkaan ja

OWASP Testing Guide -dokumentissa riskien arviointiin löytyy perusteelliset ohjeet (OWASP 2008, 326-332). Opinnäytetyön tapaustutkimuksen pohjalta havaittiin, että riskien hallinta on avainasemassa valmisohjelmistojen sovellusturvallisuuden arvioinnissa. OWASP-metodologian kehityshaasteena näyttäisivät olevan nimenomaan riskien hallinnan ja arvioinnin osa-alueet, jotka eivät tule parhaimmalla tavalla esiin OWASP-metodologian nykyisessä kokonaisuudessa.

4 Johtopäätökset

4.1 OWASP-metodologian soveltuvuus valmisohjelmiston sovellusturvallisuuden arvioimiseen

Opinnäytetyön päätavoitteena oli muodostaa kriittinen kokonaiskuva OWASP-metodologian soveltamisesta valmisohjelmiston sovellusturvallisuuden arvioimiseen ja testaamiseen. OWASP-metodologian soveltamista tutkittiin käytännössä laadullisen tapaustutkimuksen ja konstruktiivisen tutkimuksen menetelmillä. Tutkimuksen kohteena oli IBM InfoSphere MDM Collaboration Server -ohjelmisto, jonka sovellusturvallisuutta tutkittiin virtualisoidussa testiympäristössä käyttäen Kali Linux -distribuution avoimen lähdekoodin verkkoturvallisuusohjelmistoja.

OWASP-metodologiassa tutkimisessa keskityttiin kahden OWASP-projektin, OWASP Application Security Verification Standard sekä OWASP Testing Guide, tuottamaan kirjalliseen dokumentaatioon. Laadullisen tapaustutkimuksen viitekehyksenä toimi OWASP ASVS -standardin tason 1A mukainen sovellusturvallisuuden arviointi. Sovellusturvallisuuden testaamisen tukimateriaalina käytettiin OWASP Testing Guide -dokumentaatiota sekä muiden OWASP-projektien tuottamaa kirjallista materiaalia.

OWASP-metodologia tarjoaa selkeän yleisen viitekehyksen sovellusturvallisuuden arviointiin, jossa otetaan huomioon myös yksityiskohtaiset tekniset tietoturvavaatimukset. Opinnäytetyössä OWASP-metodologian avulla pystyttiin suorittamaan yleisellä tasolla teknisesti monimutkaisen kaupallisen valmisohjelmiston sovellusturvallisuuden arviointia ilman aikaisempaa kokemusta sovellusten tekniseen tietoturvaan liittyvistä konsultointitehtävistä. Opinnäytetyöprojektin asettamissa rajoissa sovellusturvallisuuden arviointi jäi suhteellisen pintapuoliseksi ja se ei vastaa täysin OWASP ASVS -standardin tason 1A arvioimiseen kuuluvia vaatimuksia.

OWASP-metodologian dokumentaatio on kielellisesti sekä teknisten yksityiskohtien tasolla selkeästi ja ytimekkäästi ilmaistua. Sovellusturvallisuuden opiskeluun OWASP ASVS ja Testing Guide -dokumentaatiot ovat monissa kohdissa liian suppeita ja niiden

lukija joutuu etsimään lisätietoja muista lähteistä, joita on annettu valmiiksi myös Testing Guide -dokumentaatiossa. OWASP-metodologian dokumentaatiot soveltuvat parhaiten tietoturva-alan ammattilaiselle, jolla on aikaisempaa kokemusta ohjelmistokehityksestä ja tietoturvaan liittyvästä ohjelmistotekniikasta. OWASP-metodologian avulla voidaan saavuttaa laadullisesti parempia tuloksia sovellusturvallisuuden arvioinnissa, koska OWASP-metodologia toimii yleisenä sovellusturvallisuuden mittaristona ja sen avulla sovellusturvallisuuden testaamisen prosessin kattavuutta ja tarkkuutta voidaan kehittää.

OWASP-metodologian vahvuutena on sen looginen selkeys ja pitkälle menevä teknisen yksityiskohtaisuus, joka on silti riippumatonta testaamisessa käytetyistä ohjelmistoista ja teknisistä ratkaisuista. OWASP-metodologian avulla voidaan myös arvioida testaamisessa käytettäviä ohjelmia ja niillä saavutettavia tuloksia. OWASP-metodologiaa voidaan soveltaa hyvin vapaasti ja sen ohessa voidaan käyttää laajempia tietoturvan hallinnan viitekehyksiä kuten Yhdysvaltojen valtionhallinnon NIST SP 800 -julkaisusarjaa, ISO/IEC 27000 -sarjan standardeja tai Suomen valtionhallinnon VAHTI-suosituksia. OWASP-metodologia keskittyy sovellusturvallisuuden arviointimenetelmien teknisiin yksityiskohtiin, joten se ei yksinään riitä organisaatioiden tietoturvapolitiikan ja siihen liittyvien vaatimusten määrittämiseen.

OWASP-metodologian vapaamuotoisuus on osittain sen heikkous, koska OWASP-metodologia antaa soveltajalleen suuren vapauden menetelmien valinnassa ja niiden käyttämisessä. Sovellusturvallisuuden arvioinnista OWASP-metodologian mukaisesti on arvioija itse vastuussa, koska hänellä on täysi vapaus soveltaa OWASP-metodologiaa omien näkemystensä mukaisesti. Organisaatioiden tulisikin määrittää tarkasti sisäinen tietoturvapolitiikka ja käyttää sen tukena laajempia tietoturvan hallinnan viitekehyksiä, ennen kuin tuotantojärjestelmien sovellusturvallisuutta aletaan arvioida vakavasti. OWASP-metodologian hyödyntämisessä korostuu arvioinnin ja testaamisen läpinäkyvyydestä kiinnipitäminen, niin että sisäisen tai ulkoisen arvioinnin tilaaja saa raportoinnin kautta selkeän käsityksen testaamiseen kattavuudesta, tarkkuudesta sekä käytetyistä menetelmistä ja ohjelmistoista. Raportoinnissa on myös selkeästi esitettävä testaamisen objektiiviset lopputulokset ja arvioijan omat tulkinnat perusteluineen, jotta testauksen luotettavuutta pystyttäisiin paremmin arvioimaan.

4.2 Sovellusturvallisuuden testaamisen luotettavuus

Opinnäytetyön tavoitteena oli myös arvioida yleisesti sovellusturvallisuuden testaamisen luotettavuutta osana OWASP-metodologian kriittisen kokonaiskuvan muodostamista. Laadullisessa tapaustutkimuksessa tutkittiin IBM InfoSphere MDM Collaboration Server -ohjelmiston sovellusturvallisuuden testaamista virtualisoidussa testiympäristössä. Tapaustutkimuksen tuloksia voidaan pitää objektiivisena siinä mielessä, että testijärjestelyt ovat toistettavissa ja niistä saadaan samat tulokset. Testaamisessa käytettiin useita automaattisia haavoittuvuusskannausohjelmia, joiden tulokset olivat samansuuntaisia ja yksittäisien ohjelmien tulokset olivat samanlaisia kun niiden haavoittuvuusskannauksia ajettiin useamman kerran.

OWASP-metodologiassa esitetyt testijärjestelyt ovat kokonaisuutena hyvin monipuolisia ja kattavia. Niiden pohjalta on mahdollista suorittaa perusteellinen sovellusturvallisuuden arviointi. Yksittäisten tietoturvakontrollien testauksen kuvauksessa OWASP -metodologian dokumentaation taso kuitenkin vaihtelee ja testijärjestelyjen kuvaus voi jäädä epäselväksi tai tulkinnallisesti avoimeksi. OWASP-metodologian heikkoutena on testitulosten tulkinnan määrittely ja arviointi, joka on paljolti testaamista suorittavan tietoturva-ammattilaisen vastuulla. Tämä johtuu osittain sovellusturvallisuuden testaamisen luonteesta. Sovelluksia ja niiden käyttöympäristöjä on tutkittava yksittäisinä tapauksia ottaen huomioon kaikki kokonaisuuteen vaikuttavat tekijät kuten organisaation prosessien toiminta, käytössä olevat tietojärjestelmät ja IT-infrastruktuuri. Sovellusturvallisuuden testaamisessa on kyettävä soveltamaan tapauskohtaisesti yleisiä sovellusturvallisuuden periaatteita ja hyviä käytäntöjä, niin että testaaminen on riittävän kattavaa ja tarkkaa sekä kohdistuu suurimpien tietoturvariskien alueelle.

Sovellusturvallisuuden luotettavuutta arvioitaessa on otettava huomioon, että testaukseen käytettävät resurssit ovat usein rajallisia ja testauksen tarkka rajaus on käytännössä välttämätöntä jo kustannussyistä. Haavoittuvuuksia etsivät ja käyttävät todelliset hyökkääjät voivat puolestaan käyttää halutessaan huomattavasti enemmän aikaa kohteeksi valitun uhrin tutkimiseen ja hyökkäyksien suunnitteluun kuin normaalia sovellusturvallisuuden testausta suorittava tietoturva-alan ammattilainen. Näiden syiden vuoksi sovellusturvallisuuden testauksen suunnittelussa on tärkeää potentiaalisten riskien kartoitta-

minen ja keskittyminen suurimpien riskien huomioimiseen testaamisessa. Kaikkien mahdollisten tapausten yksityiskohtainen selvittely ei ole välttämättä tarkoituksenmukaista.

Opinnäytetyöhön kuulunut laadullinen tapaustutkimus osoitti, että sovellusturvallisuuden testaamisessa tarvitaan testausohjelmistoihin liittyvän teknisen tietämyksen lisäksi ymmärrystä tietoturvan hallinnan kokonaisuudesta sekä erityisesti riskien kartoittamisen ja arvioinnin osaamista. OWASP-metodologian antaa hyvän kokonaiskuvan testaamiseen liittyvistä teknisistä kysymyksistä ja testausohjelmistojen käyttämisestä. OWASP -metodologia ei kuitenkaan muodosta täysin yhtenäistä kokonaisuutta ja sen sisällä dokumentaation taso vaihtelee, koska OWASP-metodologiaa kehitetään useissa avoimissa yhteisöprojekteissa vapaaehtoistyönä. Yhteisöprojektit ovat täysin riippuvaisia vapaaehtoisten henkilöiden aktiivisesta osallistumisesta. Luonnollisesti työskentelyä ohjaavat projektien aktiivijäsenten omat mielenkiinnon kohteet ja osaaminen.

Organisaatioiden tietoturvan hallinta ja riskien arviointi ovat OWASP-metodologian dokumentaatiossa vähäisemmässä roolissa. Tämä johtunee siitä, että pääosa OWASP -projekteissa vaikuttavista henkilöistä on tietoturva-alan konsultteja, jotka ovat keskittyneet enemmän tekniseen tietoturvan testaamiseen ja ohjelmistokehitykseen liittyviin tehtäviin. OWASP-metodologiassa on myös selkeä painotus ohjelmistokehitysprosessin tietoturvallisuuden kehittämiseen, jota on kuvattu perusteellisesti OWASP Testing Guide -dokumentaatiossa. Opinnäytetyön lähtökohtana oli OWASP-metodologian soveltaminen valmisohjelmistoja käyttävän organisaation näkökulmasta. OWASP -metodologia ei tarjoa tällaiseen soveltamiseen yhtä kattavasti tukea kuin ohjelmistokehitykseen.

Laadullisessa tapaustutkimuksessa käsitellystä InfoSphere MDM Collaboration Server -ohjelmistosta ei löytynyt testauksessa kriittiseksi arvioituja haavoittuvuuksia. Tulos oli odotettavissa, koska kyseistä ohjelmistoa on kehitetty kymmenisen vuotta ja sen ohjelmistotekninen J2EE-arkkitehtuuri on websovelluksen alustana tunnetusti vakaa ja laadukas. InfoSphere MDM Collaboration Server -ohjelmiston sovellusturvallisuuden arvioinnissa paljastui kuitenkin useita pienempiä puutteita. Testausjärjestelyt, joiden mukaan testaus suoritettiin, ovat suhteellisen helposti toistettavissa, ja testitulokset ovat

teknisessä mielessä objektiivisia. Löydettyjen sovellusturvallisuuden puutteiden aiheuttaman todellisen uhkan ja riskien arviointi on kuitenkin vaikeaa. Tarkemmalla manuaalisella arvioinnilla olisi mahdollista selvittää paremmin riskien vakavuutta, mutta se lisäisi huomattavasti testauksen työmäärää ja on laajuudeltaan opinnäytetyöprojektin mahdollisuuksien ulkopuolella. Tarkemmassa arvioinnissa tulisi myös käyttää OWASP ASVS -standardin ylempiä sovellusturvallisuuden tasoja.

Yleisesti ottaen OWASP-metodologia tarjoaa hyvän teknisen perustan sovellusturvallisuuden arviointiin. Valmisohjelmistojen sovellusturvallisuuden arviointiin OWASP -metodologian soveltaminen on haasteellisempaa kuin tyypilliseen ohjelmistokehitysprojektiin, jossa lähtökohtana on uuden ohjelmistotuotteen suunnittelu ja toteuttaminen. Valmisohjelmiston kohdalla korostuu enemmän tietojärjestelmässä käsiteltävän informaation organisaatiolle tulevan substanssiarvon tunnistaminen, jonka avulla sovellusturvallisuuden testausta voidaan kohdistaa kriittisiin osa-alueisiin ja varmistaa tietoturvakontrollien toiminta sovelluksen turvallisuuspolitiikan valvomiseksi.

Opinnäytetyönprojektin tulosten pohjalta hyvänä käytäntönä voidaan pitää OWASP -metodologian käyttämistä jonkin laajemman organisaatioiden tietoturvan hallinnan viitekehyksen yhteydessä. Näin voidaan kompensoida OWASP-metodologian osittaisia puutteita ja parantaa erityisesti organisaation kykyä riskien arviointiin ja hallintaan. Laajemman tietoturvan hallinnan viitekehyksen avulla OWASP-metodologiaa pystytään hyödyntämään tuloksellisesti ja tehokkaasti. Käyttökelpoisia tietoturvan hallinnan viitekehyksiä voisivat olla esimerkiksi NIST SP-800 -julkaisusarja, ISO/IEC 27000 -sarjan standardit tai Suomen valtionhallinnon VAHTI-suositukset. Organisaatioiden tietoturvassa tulee myös huomioida lainsäädännön asettamat rajoitukset, jotka voivat asettaa lisävaatimuksia tietoturvan hallintaan. Esimerkiksi Yhdysvaltojen liittovaltion nykyiseen lainsäädäntöön kuuluvat Sarbanes-Oxley Act vuodelta 2002 ja Gramm-Leach-Bliley Act vuodelta 1999 vaikuttavat laajasti myös kansainväliseen liiketoimintaan.

4.3 Kehittämis- ja jatkotutkimusehdotukset

Tässä opinnäytetyössä selvitettiin valmisohjelmiston sovellusturvallisuuden arvioinnin mahdollisuuksia ja haasteita yleisellä tasolla. Jatkotutkimuksessa olisi hedelmällistä sy-

ventää tarkastelua sovellusturvallisuuden kokonaisuuden sisällä rajattuihin osa-alueisiin yksityiskohtaisemmalla tasolla. Sovellusturvallisuuden testaaminen mustalaatikkomenetelmillä on haastava tehtävä, jonka suorittaminen ja tulosten analysointi on sellaisenaan aikaa vievä prosessi. Testauksen kohteen selkeä rajaaminen lisää testaamisen tuloksellisuutta ja tehokkuutta. Jotta testaamisesta saadaan mahdollisimman paljon hyötyä, on tulosten analysointiin syytä varata riittävästi aikaa. Testaamisen suorittaminen mekaanisesti tuottaa vain teknistä dataa, joka tulee jalostaa riskien arviointia tukevaksi informaatioksi.

Sovellusturvallisuuden tutkimusta voidaan tarkemman rajauksen lisäksi kehittää tutkimuksen tavoitteiden selkeämmällä määrittelyllä. Tässä opinnäytetyöprojektissa havaittiin, että tietojärjestelmien sisältämän informaation substanssiarvo ja riskienhallinta ovat sovellusturvallisuuden kannalta tärkeitä tekijöitä teknisten tietoturvakontrollien ohella. Sovellusturvallisuus muodostaa laajan ja monimutkaisen kokonaisuuden, jonka määrittelystä on monenlaisia näkemyksiä. Tämän vuoksi tutkimuksen lähtökohdaksi valittu näkökulma vaikuttaa ratkaisevasti tutkimuksessa saataviin tuloksiin. Tutkimuksessa voidaan saavuttaa parempi laadullinen ja tuloksellinen taso, kun valittu näkökulman, kohteen rajaus ja tutkimuksen tavoitteet muodostavat selkeästi määritellyn, eheän kokonaisuuden.

Sovellusturvallisuuden testauksessa on tärkeää huomioida testattavan sovelluksen ohjelmistotekniseen arkkitehtuurin ominaisuuspiirteet. Sovelluksen kehittämisessä käytetyt ohjelmointikielet sekä palvelinympäristön erilaiset komponentit vaikuttavat testaukseen suunnitteluun. Testaamisessa tulisi ottaa huomioon sovelluksen ohjelmistotekninen arkkitehtuuri, niin että testaus kohdistuisi arkkitehtuurin tunnettuihin heikkouksiin. Tässä opinnäytetyössä tutkittiin J2EE-arkkitehtuurilla toteutettua websovellusta. Opinnäytetyöprojektissa havaittiin, että OWASP-projektien materiaaleista sekä yleisistä Internet-lähteistä ei löydy kovin paljoa yksityiskohtaisempaa tietoa J2EE-arkkitehtuurin sovellusturvallisuudesta. Jatkotutkimuksessa olisi erityisen arvokasta J2EE-arkkitehtuurin sovellusturvallisuuden tarkempi selvittely, joka voisi herättää laajempaa ammatillista mielenkiintoa.

4.4 Opinnäytetyöprosessin ja oman oppimisen arvioiminen

Tämän opinnäytetyöprojektin henkilökohtaisena oppimistavoitteena oli tekijän tietoturvaosaamisen syventäminen erityisesti sovellusturvallisuuden ja avoimen lähdekoodin verkkoturvallisuusohjelmien käyttämisen osa-alueilla. Lisäksi tavoitteena oli oppia yleistä tietoturvan hallintaa ja konsultointia. Omasta näkökulmastani opinnäytetyöprosessi oli kokonaisuutena onnistunut ja henkilökohtaiset oppimistavoitteet tulivat täytetyiksi. Ennakko-odotuksia vastaten, työskentely opinnäytetyön parissa oli mielenkiintoista ja haasteellista. Opinnäytetyön aiheen rajauksesta käytiin paljon keskustelua ohjausryhmässä, joka auttoi opinnäytetyöprojektin eteenpäin viemistä.

Opinnäytetyön aiheen valinta tuki henkilökohtaisia oppimistavoitteita, sillä keskittyminen OWASP-metodologiaan tuki sovellusturvallisuuden hahmottamista kokonaisuutena ja samalla ohjasi avoimen lähdekoodin verkkoturvallisuusohjelmistojen käyttämisen opettelua, joka olisi ollut vaikeampaa ilman OWASP-metodologian tarjoamaa teoreettista viitekehystä. OWASP-metodologiaan ja sen soveltamiseen keskittyminen auttoi myös ymmärtämään yleisemmällä tasolla tietoturvan hallintaan ja kehittämiseen liittyviä haasteita.

Kriittisesti arvioituna opinnäytetyön aiheen rajausta olisi voinut kehittää vieläkin tarkemmaksi ja keskittyä tarkemmin johonkin sovellusturvallisuuden osa-alueeseen. Tällä tavoin opinnäytetyössä olisi voitu saavuttaa enemmän merkityksellisiä tuloksia. Myös keskittyminen sovelluksen ohjelmistoteknisen arkkitehtuurin ominaispiirteisiin tietoturvan näkökulmasta olisi voinut tuottaa hyödyllistä lisäarvoa opinnäytetyön tuloksiin. Opinnäytetyön nykyiselle rajaukselle on kuitenkin hyvä perustelu, koska sovellusturvallisuuden arvioinnista on vähän aikaisempaa tutkimusta ja kriittisen kokonaiskuvan hahmottaminen helpottaa yksityiskohtaisempiin osa-alueisiin kohdistuvaa jatkotutkimusta. Oman oppimisen kannalta opinnäytetyöprosessi konkretisoi henkilökohtaisella tasolla yleisen toteamuksen, jonka mukaan tietoturva on pikemmin prosessi kuin yksittäinen teknologia tai tuote. Opinnäytetyöprojektin kautta sain hyvät valmiudet jatkaa oppimisprosessia sovellusturvallisuuden tutkimuksessa ja soveltamisessa sekä teoreettisella tasolla että käytännön tasolla organisaatioiden tietoturvan hallinnassa ja kehittämisessä.

Lähteet

IBM 2013a. IBM Software Product Compatibility Reports. InfoSphere Master Data Management 10.1 Luettavissa: http://pic.dhe.ibm.com/infocenter/prodguid/v1r0/clarity-reports/report/html/softwareReqsForProductByComponent?deliverableId=1315594748846&duComponent=Server_385083F00BF511E28BB5885E0925FE36. Luettu 16.6.2013.

IBM 2013b. Collaborative authoring with InfoSphere MDM Collaboration Server. Luettavissa: http://pic.dhe.ibm.com/infocenter/mdm/v10r1/topic/com.ibm.pim.ovr.doc/pim_ref_overview.html. Luettu 16.6.2013.

IBM 2013c. Architecture of InfoSphere MDM Collaboration Server Luettavissa: http://pic.dhe.ibm.com/infocenter/mdm/v10r1/topic/com.ibm.pim.ovr.doc/pim_con_archfo.html. Luettu 18.6.2013.

Meucci, M. 2010. OWASP Testing Guide v3: training. OWASP London 28th May. Luettavissa: http://www.owasp.org/images/7/7d/OWASP_London_Training_Course_2010_-_Testing_Guide_v3.pdf. Luettu 8.6.2013.

Offensive Security 2013. Kali Linux Official Documentation. Luettavissa: http://www.kali.org/official-documentation. Luettu 20.6.2013.

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2009. Kehittämistyön menetelmät. Uudenlaista osaamista liiketoimintaan. WSOYpro Oy. Helsinki.

OWASP 2013a. About The Open Web Application Security Project. Luettavissa: https://www.owasp.org/index.php/About_OWASP. Luettu 25.3.2013.

OWASP 2013b. OWASP Testing Project. Luettavissa: https://www.owasp.org/index.php/OWASP_Testing_Project. Luettu 3.6.2013.

OWASP 2013c. OWASP Testing Guide v4 Table Of Contents. Luettavissa: https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents. Luettu 26.8.2013.

OWASP 2012. OWASP Application Security Verification Standard Project. Luettavissa:https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project. Luettu 26.3.2013.

OWASP 2009a. About ASVS. Luettavissa: http://www.owasp.org/images/7/71/About_OWASP_ASVS.ppt. Luettu 26.3.2013.

OWASP 2009b. OWASP Application Security Verification Standard 2009. Web Application Standard Release. Luettavissa: http://www.owasp.org/images/4/4e/OWASP_ASVS_2009_Web_App_Std_Release.pdf. Luettu: 29.5.2013.

OWASP 2008. OWASP Testing Guide 2008 V3.0. Luettavissa: https://www.owasp.org/images/8/89/OWASP_Testing_Guide_V3.pdf. Luettu: 3.6.2013.

Liitteet

Liite 1. Virtualisoidun testiympäristön kuvaus

Opinnäytetyöprojektissa käytettiin kannettavalla tietokoneella ajettavaa, virtualisoitua testiympäristöä, joka koostui kahdesta VMware Workstation -virtuaalikoneesta. Virtualisointikerroksen alustana oli Windows 7 Professional 64-Bit -käyttöjärjestelmä.

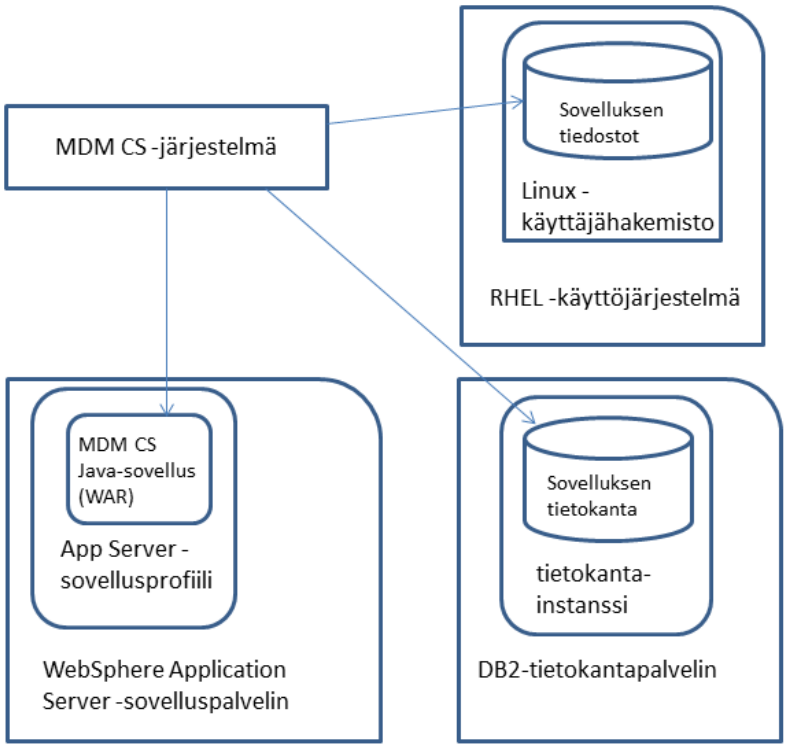
Kannettavan tietokoneen kokoonpano	
Valmistaja ja malli:	Lenovo W520
Prosessori:	Intel Core i7-2670QM 2,20 GHz
Keskusmuisti:	8 Gt
Kiintolevy:	500 Gt
Käyttöjärjestelmä:	Windows 7 Professional 64-bit

Virtualisointikerroksen kokoonpano	
Virtualisointiohjelmisto:	VMware Workstation 8.0.6 for Windows
Virtuaalikoneiden määrä:	2
Virtuaalikoneiden verkkoasetus:	Host Only
IP-aliverkko	192.168.126.0/24

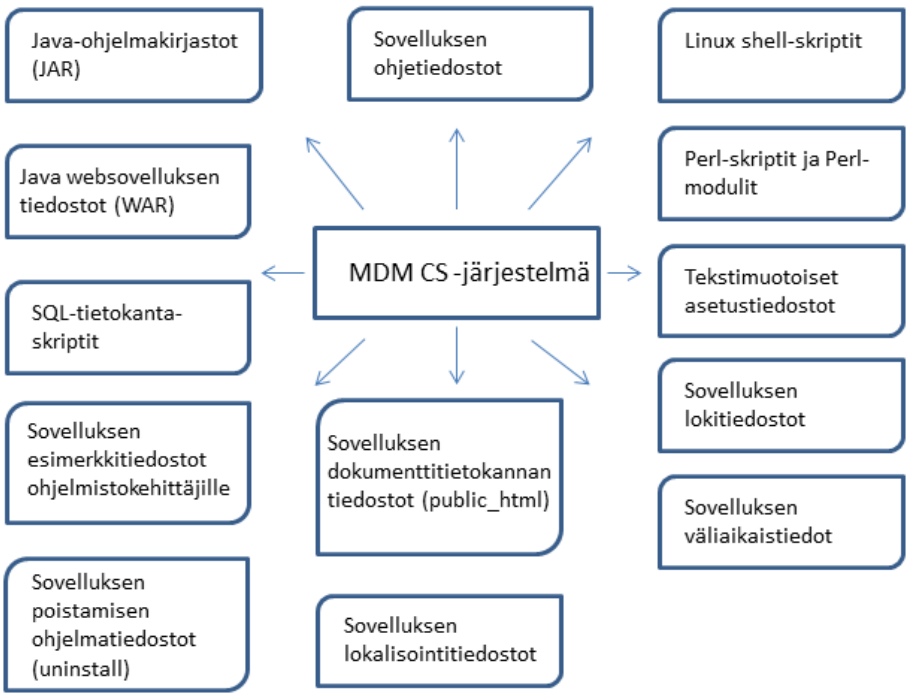
Virtuaalikone 1 (MDM CS -ohjelmisto)	
Prosessori:	4 sockets, 1 CPU per socket
Muisti:	4 Gt
Levytila:	40 Gt
IP-osoite:	192.168.126.121 (staattinen)
Käyttöjärjestelmä:	Red Hat Enterprise Linux 6.4 64-bit
Virtualisointiajurit:	VMware Tools for Linux
IBM ohjelmistot:	DB2 10.1 ESE WebSphere Application Server 8.0 InfoSphere MDM CS 10.1 FP1

Virtuaalikone 2 (Kali Linux -distribuutio, Offensive Securityn paketoima VMware Workstation -virtuaalikone)	
Prosessori:	1 socket, 1 CPU per socket
Muisti:	768 Mt
Levytila:	30 Gt
IP-osoite:	dynaaminen 192.168.126.0/24 aliverkosta
Käyttöjärjestelmä:	Kali Linux 1.0.3 32-bit
Virtualisointiajurit:	Avoimen lähdekoodin open-vm-tools

Testitapaus 1.1, sovelluksen tietoturva-arkkitehtuuri



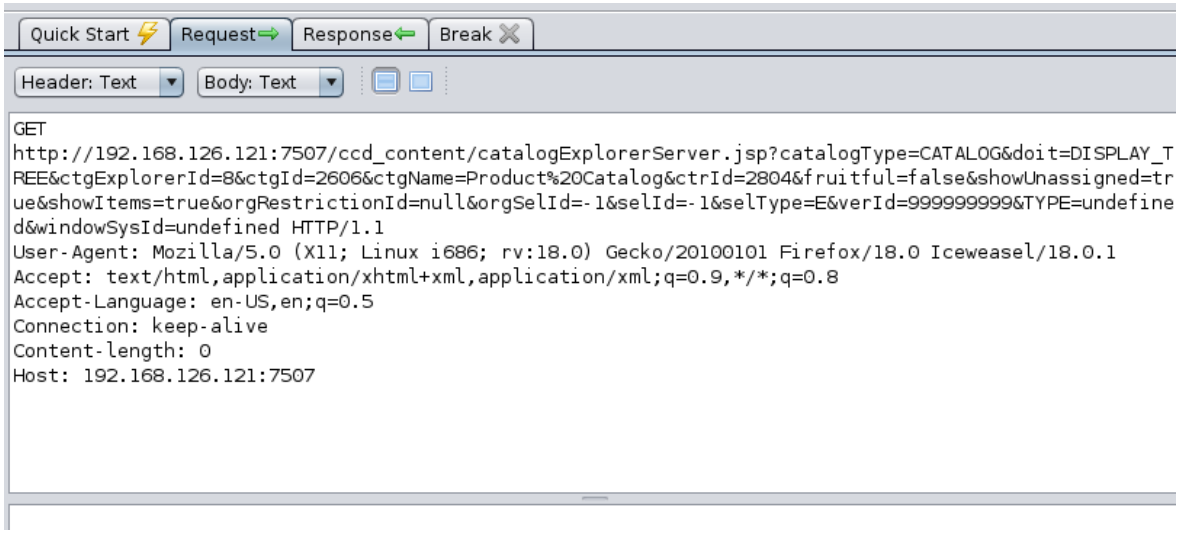
Kuvio 1. Sovelluksen tietoturva-arkkitehtuuri.



Kuvio 2. Sovelluksen komponenttien ryhmittely.

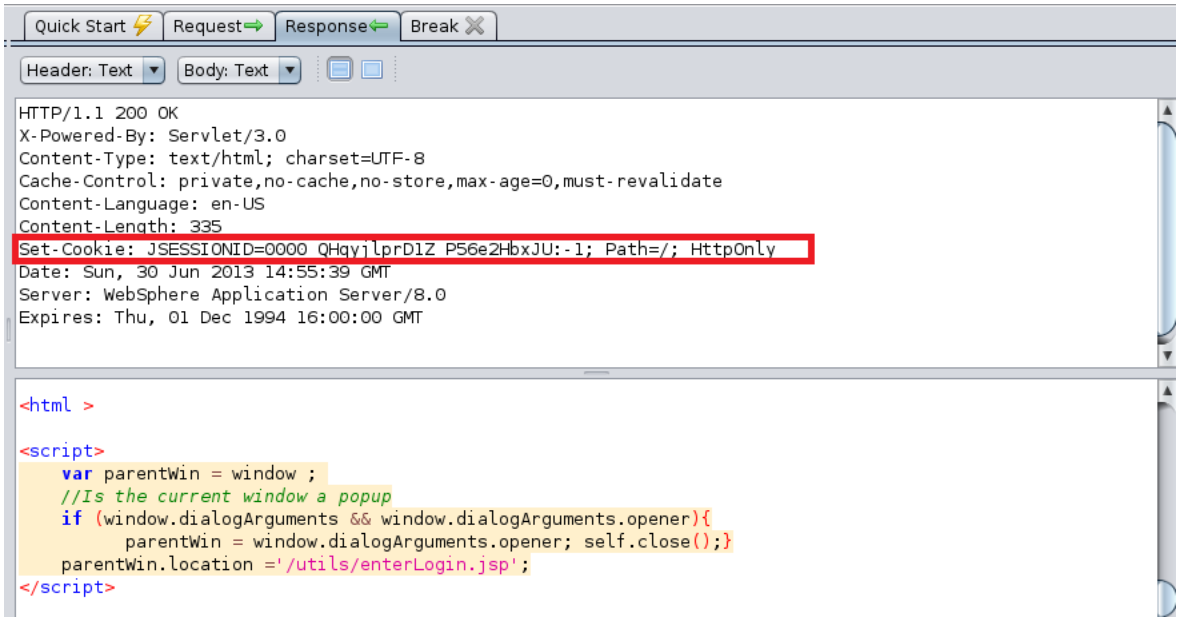
Testitapaus 2.1, sovelluksen resurssien autentikointi

Testauksessa yritettiin ohittaa autentikointi suoralla viittauksella sovelluksen resursseihin. HTTP-pyynnöllä ei ole JSESSIONID-istuntotunnistetta.



Kuva 1. OWASP Zed Attack Proxy -kuvakaappaus HTTP GET -pyynnöstä.

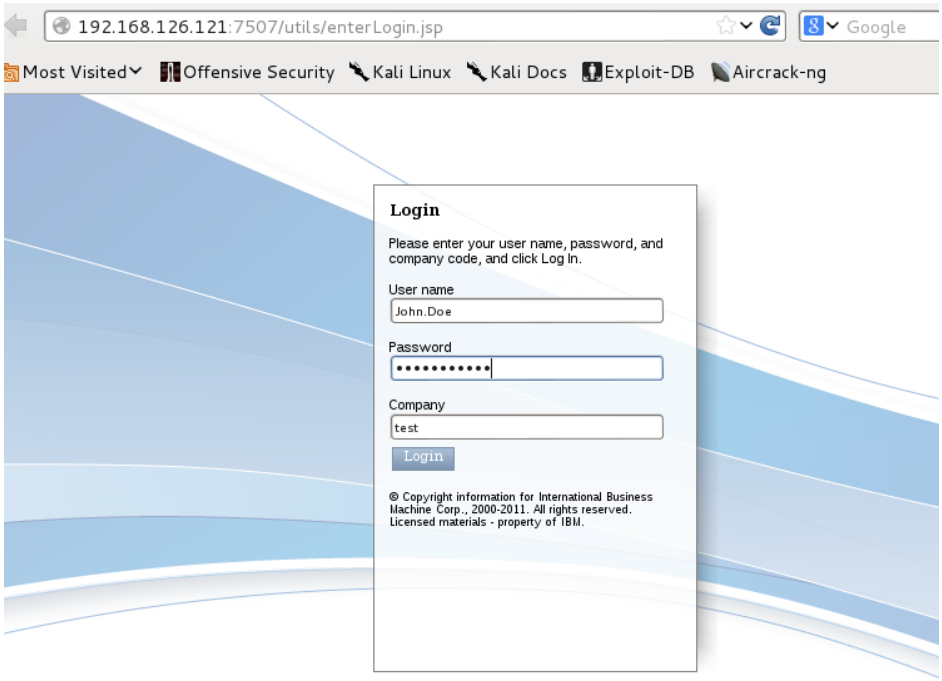
Sovellus vastaa HTTP-vastauksella, joka asettaa JSESSIONID-istuntotunnisteen ja tekee URL-uudelleenohjauksen JavaScript-koodilla sovelluksen kirjautumissivulle.



Kuva 2. OWASP ZAP -kuvakaappaus HTTP-vastauksesta.

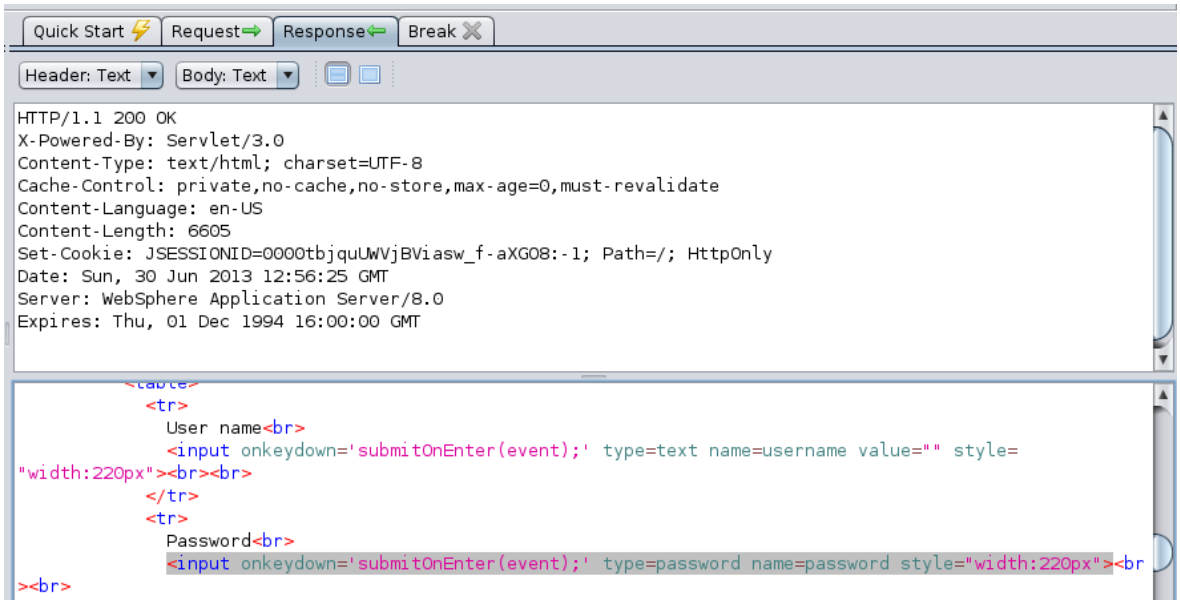
Testitapaus 2.2, kirjautumisikkunan turvallisuus

Testauksessa käytettiin MDM CS järjestelmän kirjautumisikkunaa Iceweasel-selaimella. Sovelluksen kirjautumisikkuna ei näytä salasanakentän sisältöä käyttäjälle.



Kuva 3. Kuvakaappaus MDM CS -sovelluksen kirjautumisikkunasta.

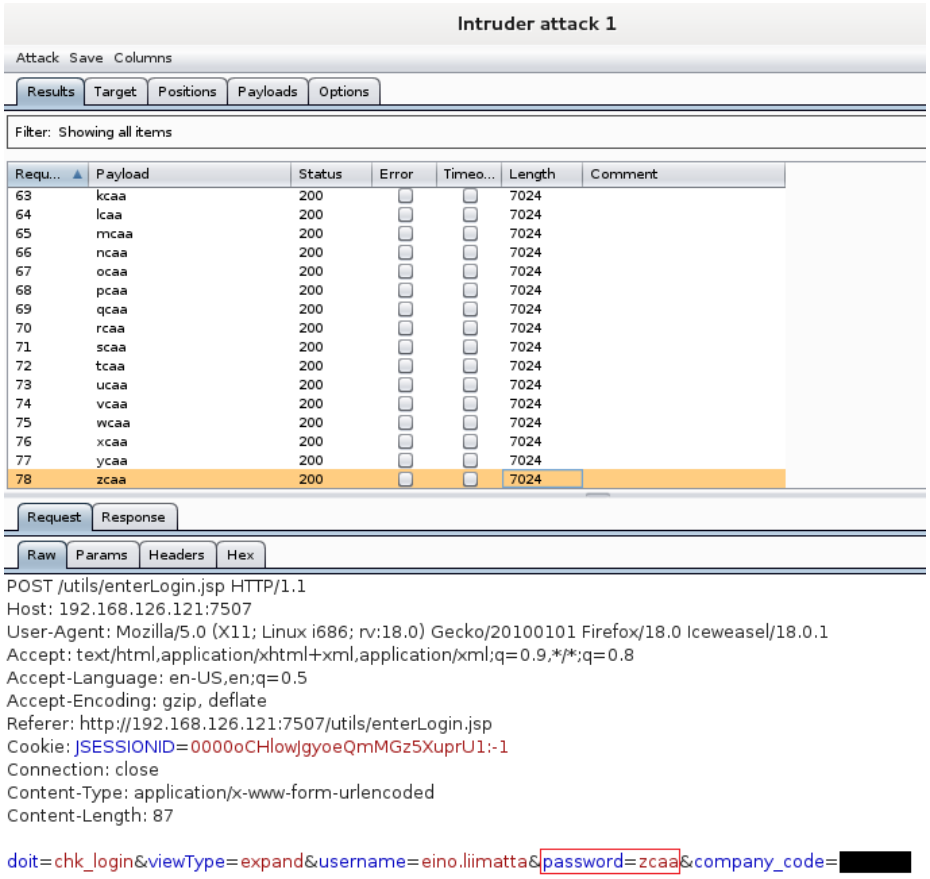
MDM CS -sovellus ei estä salasanakentän tallentamista selaimen välimuistiin. HTML-lomakkeen salasanakentässä ei käytetä autocomplete=off -attribuuttia.



Kuva 4. OWASP ZAP -kuvakaappaus MDM CS -sovelluksen kirjautumisikkunasta.

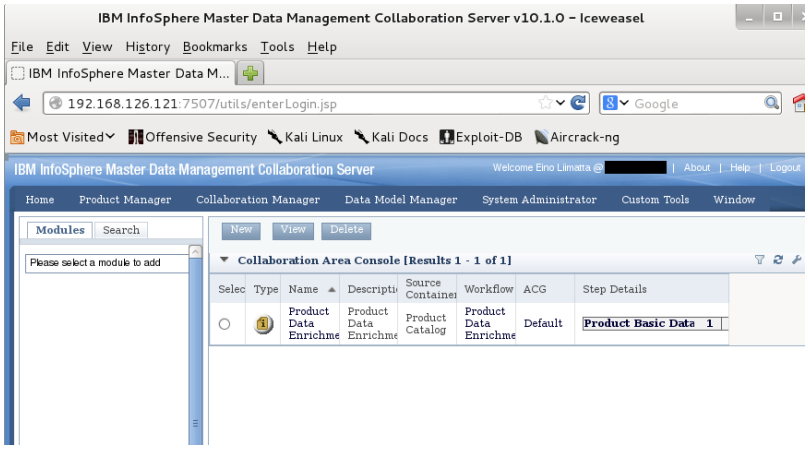
Testitapaus 2.3, käyttäjätunnuksen väliaikainen lukitseminen

Testauksessa simuloitiin brute force -hyökkäystä sovellusta vastaan. MDM CS -järjestelmään tehtiin olemassa olevalla käyttäjätunnuksella 78 kirjautumisyritystä Burp Suite Intruder-työkalulla, joka vaihtaa salasanakentän sisällön jokaiseen HTTP-pyyntöön.



Kuva 5. Kuvakaappaus Burp Suite -ohjelmiston Intruder-työkalusta.

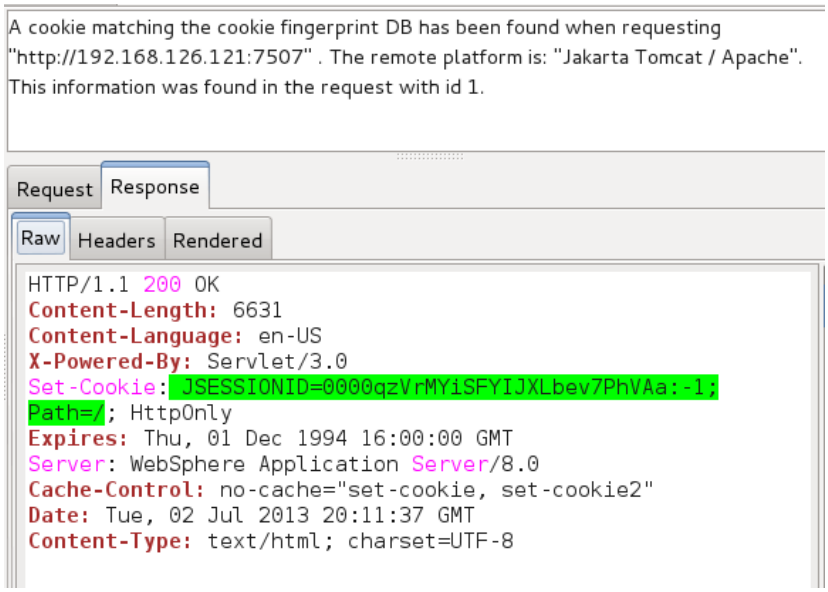
MDM CS -järjestelmässä ei ole käyttäjätunnuksen väliaikaista lukitusta, koska kirjautuminen oikealla salasanalla onnistuu välittömästi kun brute force -hyökkäys keskeytetään.



Kuva 6. Kuvakaappaus MDM CS -järjestelmän onnistuneesta kirjautumisesta brute force -hyökkäyksen jälkeen.

Testitapaus 3.1., sovelluksen istuntojenhallinnan kontrollimekanismi

MDM CS -järjestelmän istuntohallinnan kontrollimekanismi tunnistettiin Apache Tomcat -tyyppiseksi Java-toteutukseksi. Testaus suoritettiin w3af-ohjelmiston haavoittuvuusskannauksella.

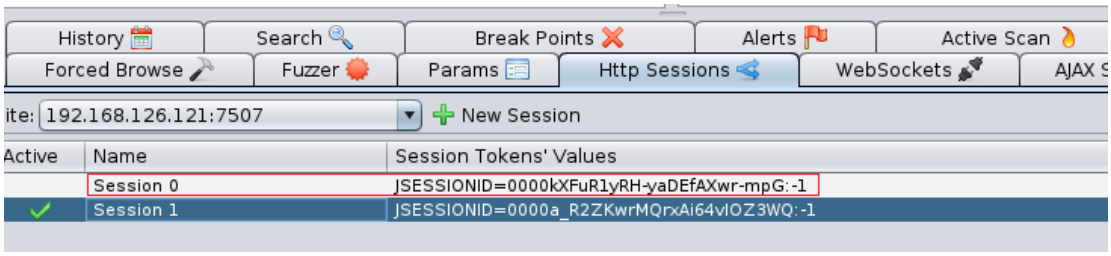


Kuva 7. Kuvakaappaus w3af-ohjelmiston haavoittuvuusskannauksen tuloksista.

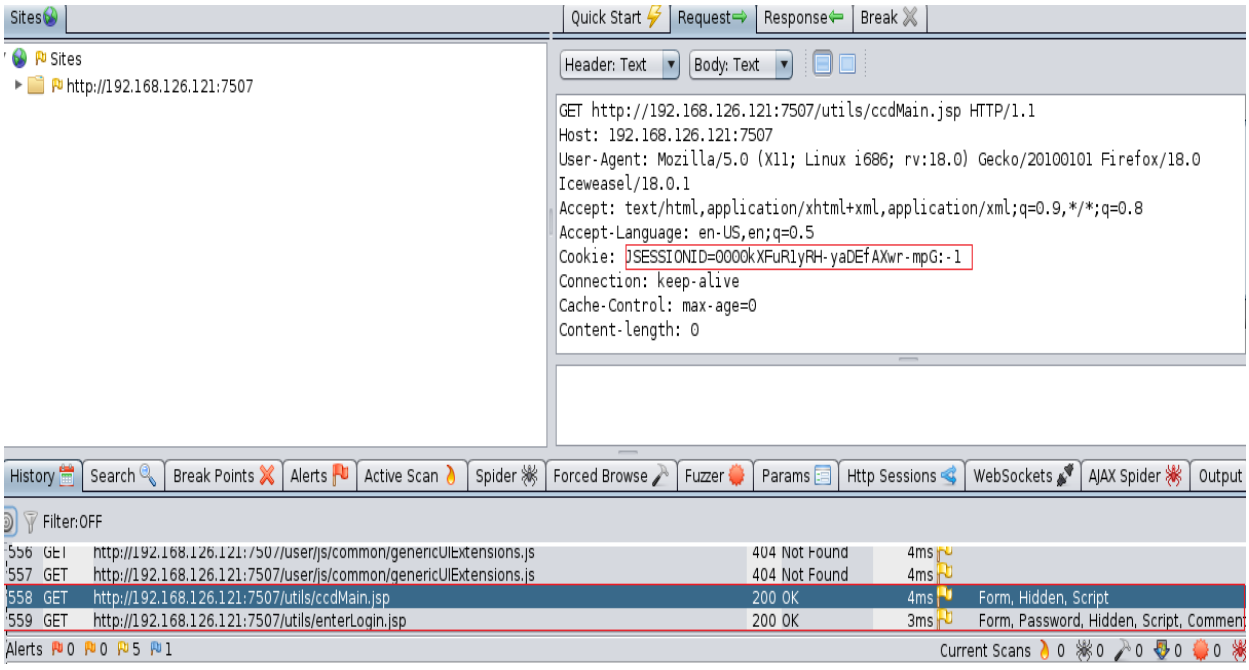
Testitapaus 3.2, sovelluksessa käytettyjen istuntojen tuhoaminen

Sovelluksen istuntojen tuhoamista testattiin OWAS ZAP -ohjelmistolla, jossa voidaan luoda useita yhtäaikaisia HTTP-istuntoja. Testi suoritettiin kirjautumalla MDM CS -sovellukseen, joka alusti uuden HTTP-istunnon. OWASP ZAP -ohjelmistolla lisättiin

toinen HTTP-istunto välilehdellä Sessions → New Session. Tämän jälkeen kirjauduttiin ulos ensimmäisestä HTTP-istunnosta ja vaihdettiin OWASP ZAP -ohjelmistossa toinen istunto aktiiviseksi. MDM CS -järjestelmään kirjauduttiin uudestaan, jolloin OWASP ZAP -ohjelmistolla oli kaksi HTTP-istuntoa tallennettuna. Ensimmäinen istunto vaihdettiin nyt aktiiviseksi ja yritettiin päästä MDM CS -järjestelmän sisälle kirjoittamalla webselaimeen sovelluksen sisäinen URL-osoite. MDM CS -järjestelmä esti vanhan HTTP-istunnon käyttämisen ja teki URL-uudelleenohjauksen kirjautumissivulle.



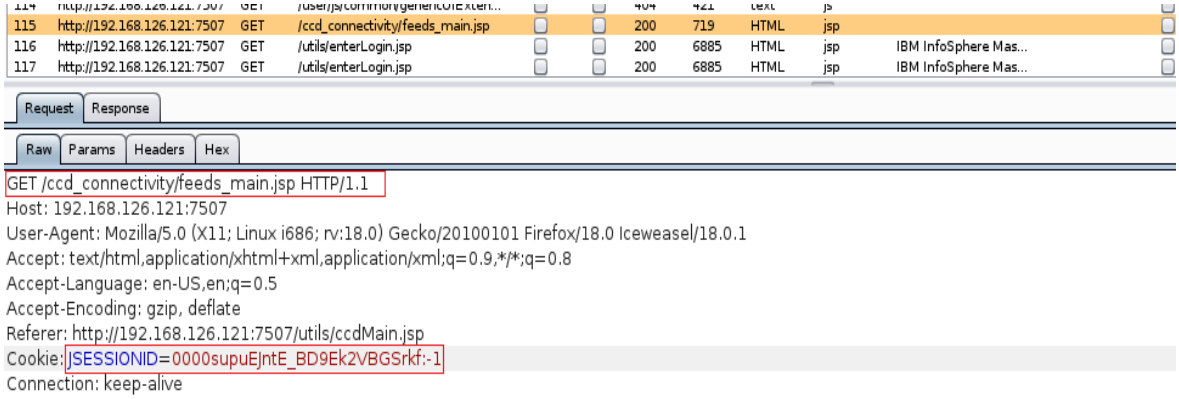
Kuva 8. Kuvakaappaus OWAS ZAP -ohjelmistosta. Kuvassa Session 0 on ensimmäinen HTTP-istunto.



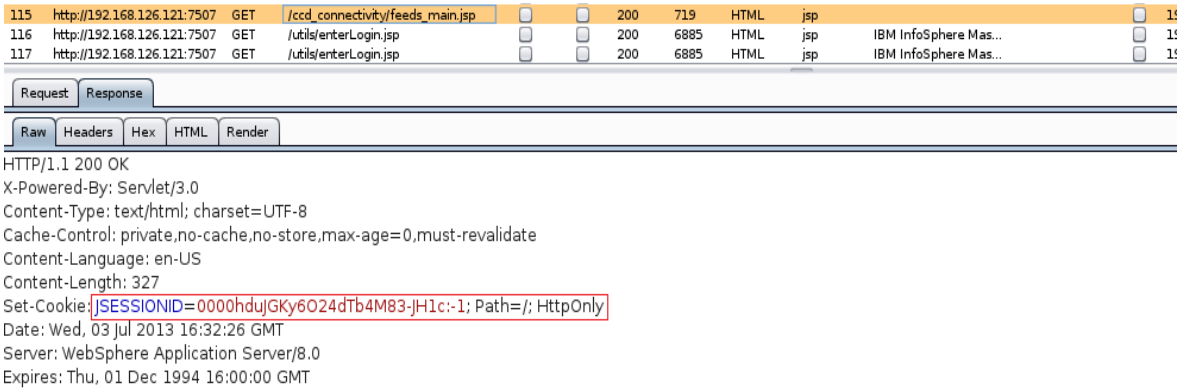
Kuva 9. Kuvakaappaus OWASP ZAP -ohjelmistossa, kun yritetään käyttää suljettua HTTP-istuntoa uudestaan.

Testitapaus 3.3., käyttämättömien istuntojen sulkeminen

MDM CS järjestelmässä on max_inactive_interval -parametri[38], jolla voidaan määrittää kuinka kauan HTTP-istunto voi olla käyttämättömänä. Oletusarvona on 1800 sekuntia (30 minuuttia). Käyttämättömien istuntojenhallintaa testattiin muuttamalla max_inactive_interval -parametrin arvoksi 120 sekuntia (2 minuuttia). Sen jälkeen kirjauduttiin MDM CS -järjestelmään ja testattiin sovelluksen toimintaa kun se oli käyttämättömänä yli 2 minuuttia. MDM CS -järjestelmä teki URL-uudelleenohjauksen kirjautumissivulle ja tuhosi käytössä olleen HTTP-istunnon.

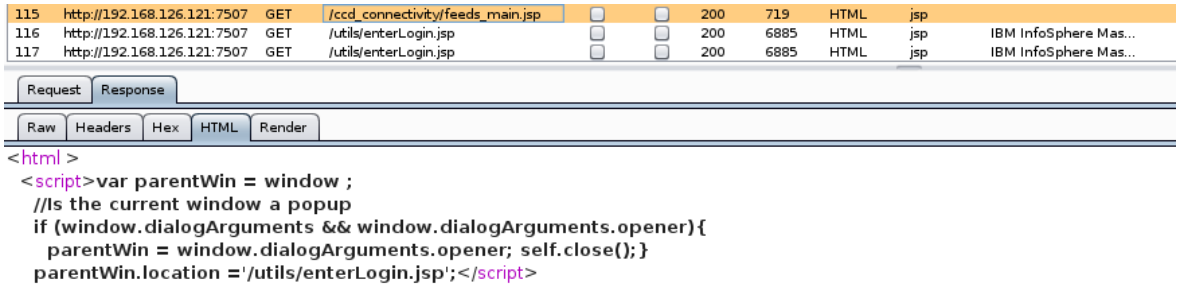


Kuva 10. Kuvakaappaus Burp Suite -ohjelmiston Proxy-työkalusta. MDM CS -sovellusta yritetään käyttää kun HTTP-istunto on vanhentunut.



Kuva 11. Kuvakaappaus Burp Suite -ohjelmiston Proxy-työkalusta. MDM CS -sovelluksen HTTP-vastauksen otsakkeessa istunnon tunniste on vaihtunut.

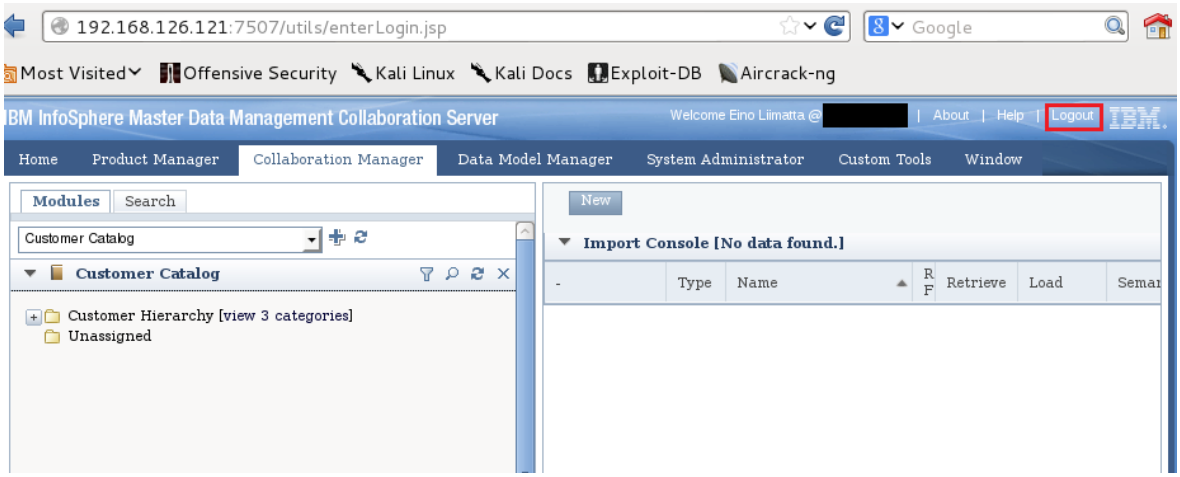
[38] max_inactive_interval -parametri on kuvattu MDM CS Info Center -dokumentaatiossa, ks. http://pic.dhe.ibm.com/infocenter/mdm/v10r1/topic/com.ibm.pim.cof.doc/properties/pim_ref_cp_maxinactiveinterval.html.



Kuva 12. Kuvakaappaus Burp Suite -ohjelmiston Proxy-työkalusta. MDM CS -sovelluksen HTTP-vastauksen HTML-koodissa on uudelleenohjaus kirjautumissivulle.

Testitapaus 3.4, sovelluksen uloskirjautumisen toiminnallisuus

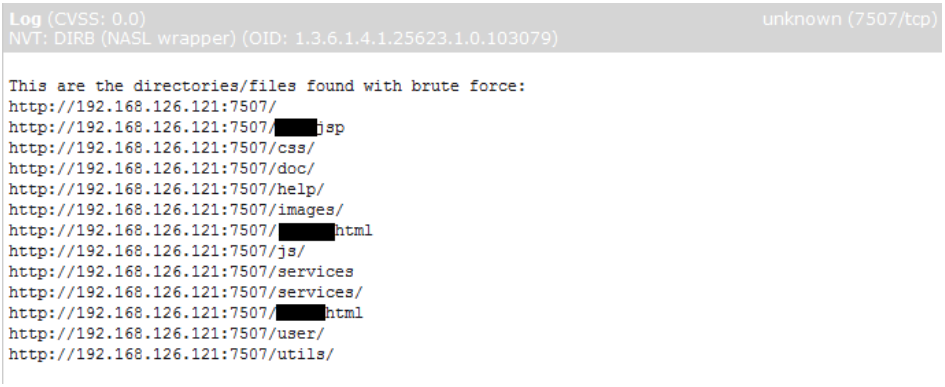
Testauksessa sovellukseen kirjauduttiin Iceweasel-selaimella. MDM CS -järjestelmässä webkäyttöliittymässä on selkeä Log out -linkki, joka on koko ajan näkyvissä HTML-sivun oikeassa ylälaidassa.



Kuva 13. Kuvakaappaus MDM CS -järjestelmän käyttöliittymästä.

Testitapaus 4.5, sovelluksen hakemistopolun selaaminen

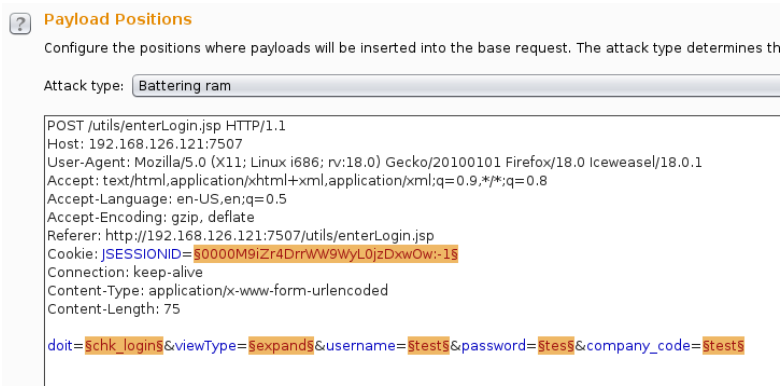
Sovelluksen hakemistopolun selaamista testattiin OpenVAS-ohjelmiston Full and very deep ultimate -asetusten mukaisella haavoittuvuusskannauksella. OpenVAS-ohjelmiston suorittama skannaus paljasti kolme websivua ja muutaman hakemiston (mm. JavaScript- ja kuvatiedostojen hakemistot).



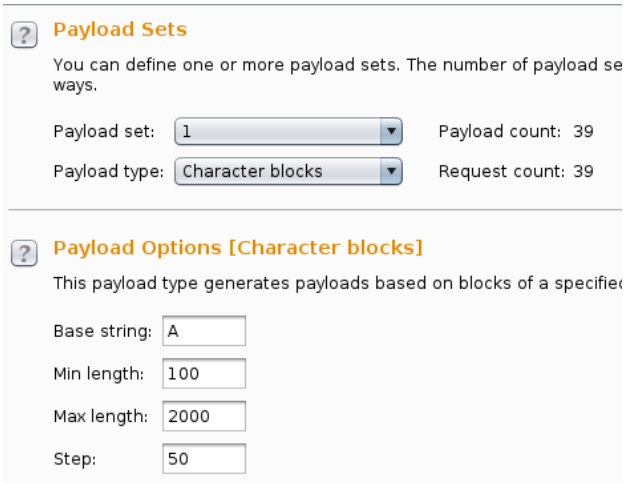
Kuva 14. Kuvakaappaus OpenVAS-ohjelmiston HTML-raportista.

Testitapaus 5.1, sovelluksen haavoittuvuus puskuriylivuodoille

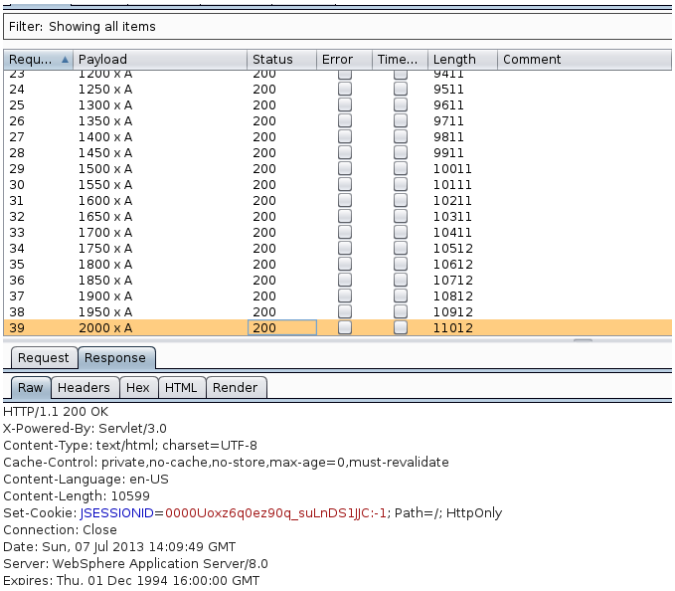
Sovelluksen haavoittuvuutta puskuri ylivuodoille testattiin Burp Suite -ohjelmiston Intruder-työkalulla lähettämällä MDM CS -sovellukselle HTTP-pyyntöjä, joissa oli ylipitkiä merkkijonoparametreja. Intruder-työkalun asetuksissa käytettiin Battering ram -tyyppistä hyökkäystä ja Character blocks -tyyppistä hyökkäyskuormaa 100-2000 merkin pituisilla merkkijonoilla. Puskuriylivuotohyökkäyksellä ei ollut vaikutusta MDM CS -sovelluksen toimintaan.



Kuva 15. Kuvakaappaus Burp Suite -ohjelmiston Intruder-työkalun asetuksista.



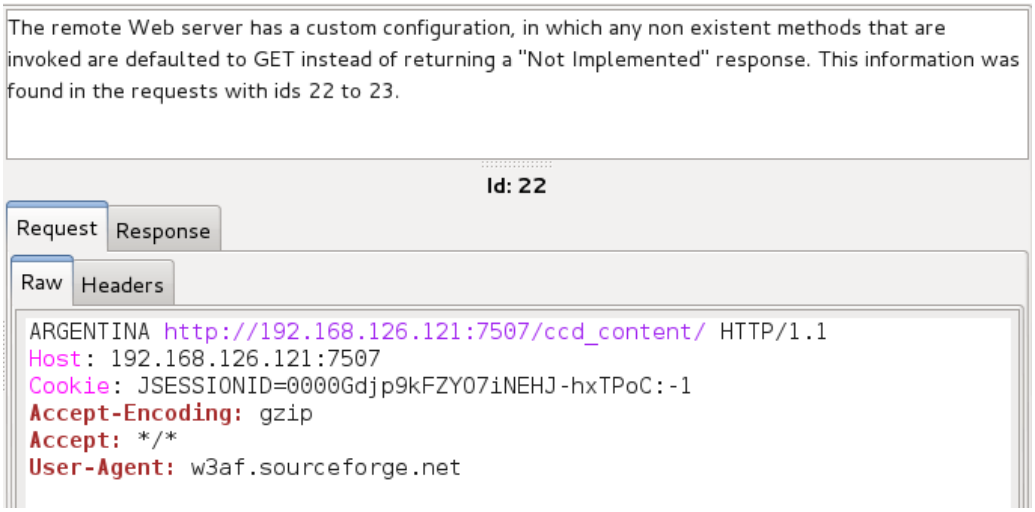
Kuva 16. Kuvakaappaus Burp Suite -ohjelmiston Intruder-työkalun asetuksista.



Kuva 17. Kuvakaappaus Burp Suite -ohjelmiston Intruder-työkalun hyökkäyksestä.

Testitapaus 11.2, sovelluksen sallitut HTTP-pyynnöt

MDM CS -järjestelmän sallimia HTTP-pyyntöjä testattiin w3af-ohjelmiston haavoittuvuusskannauksella. Tuloksista havaittiin, että MDM CS -järjestelmä tulkitsee ei-standardinmukaiset HTTP-pyynnöt GET-metodiksi. OpenVAS-ohjelmiston skannauksen tuloksista havaittiin, että palvelin vastaa TRACE-metodiin.



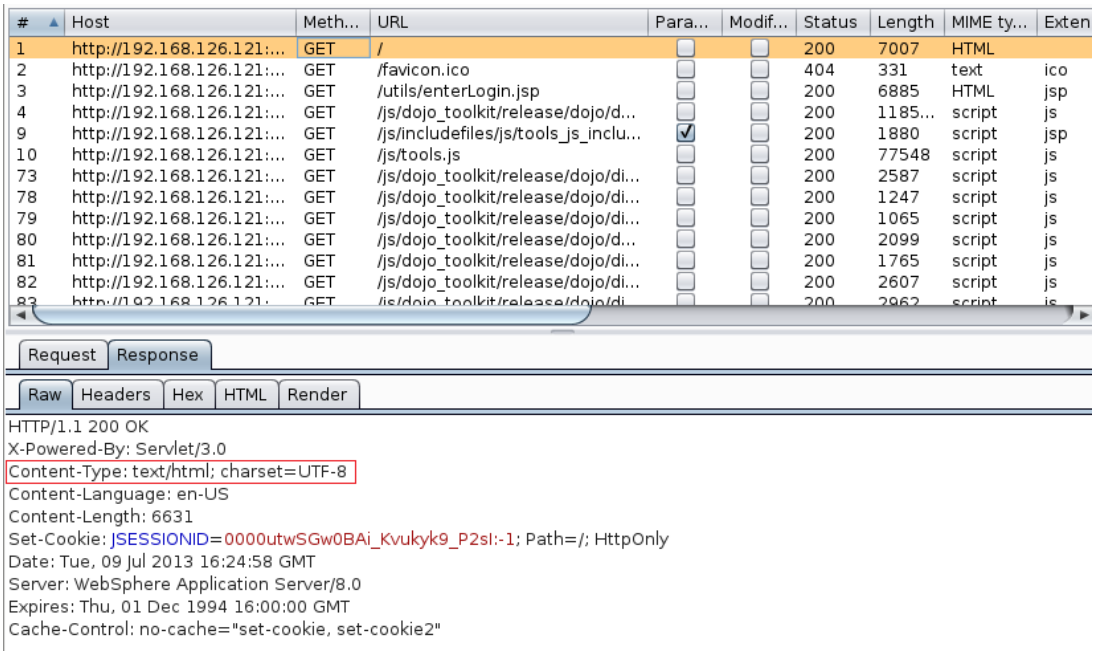
Kuva 18. Kuvakaappaus w3af-ohjelmiston haavoittuvuusskannauksen tuloksista.

```
[+] [1] The TRACE HTTP method is enabled.
[~] ~~~~~~~~~~~~~~~~~~~~
[~] ID Hash:  3edee880490c7c07ebf7e1e088efb912d80657d2615544300caa24748610b7c5
[~] Severity: Medium
[~] URL:      http://192.168.126.121:7507/
[~] Element:  server
[~] Method:   TRACE
[~] Tags:     xst, methods, trace, server
[~] Description:
[~] This type of attack can occur when the there
    is an XSS vulnerability and the server supports HTTP TRACE.
[~] CWE: http://cwe.mitre.org/data/definitions/693.html
[~] Requires manual verification?: false
[~] References:
[~]   CAPEC - http://capec.mitre.org/data/definitions/107.html
[~]   OWASP - http://www.owasp.org/index.php/Cross_Site_Tracing
[*] Variations
[~] ----------
[~] Variation 1:
[~] URL: http://192.168.126.121:7507/
[~] Regular expression:
```

Kuva 19. Kuvakaappaus OpenVAS-ohjelmiston HTML-raportista.

Testitapaus 11.3, sovelluksen HTTP-otsakkeiden merkistökoodaus

MDM CS -järjestelmän HTTP-vastauksia tutkittiin Burp Suite -ohjelmiston Proxy-työkalulla, jonka History-välilehdeltä on luettavissa kaikki sovellukselle vastaanottamat HTTP-pyynnöt ja takaisinpäin lähetetyt HTTP-vastaukset. MDM CS -järjestelmän palauttamissa HTTP-otsakkeissa merkistökoodaus on asetettu UTF-8-muotoiseksi.



Kuva 20. Kuvakaappaus Burp Suite -ohjelmiston Proxy-työkalusta.