Opinnäytetyö (AMK)

Tietojenkäsittely

Tietoverkot ja tietoturva

2016

Sami Marttinen

WEB-SOVELLUSTEN MURTAUTUMISTESTAUS

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietoverkot ja tietoturva

2016 | 35 sivua

Sami Marttinen

WEB-SOVELLUSTEN MURTAUTUMISTESTAUS

Tämän opinnäytetyön tarkoituksena on kartoittaa Whitestone Oy:n web-sovelluksien tietoturvaa. Toimeksiannon syynä on yrityksen jatkuva halu kehittää tuotteidensa laatua ja tietoturvaa sekä parantaa sisäisiä tietoturvan testauksen prosesseja. Kyseessä on yritys, joka tuottaa monipuolisesti IT- ja digitalisaatiopalveluja. Täten yrityksen tuottamien töiden tietoturva on yritykselle elintärkeää sen toiminnan sekä imagon kannalta. Työ on erittäin ajankohtainen sillä tietoturva on viime vuosina ollut suuri puheenaihe tietotekniikan alalla, ja monet yritykset ovatkin markkina-arvonsa kasvattamiseksi panostaneet entistä enemmän tietoturvaan ja sen mainostamiseen.

Opinnäytetyön teoriaosuus muodostuu web-sovelluksiin kohdistuvien tietoturvauhkien tutkimisesta. Samalla selvitetään hieman uhkien toimintaa ja yleisiä ehkäisytapoja. Käytännön osuus koostuu testauksen suunnittelusta, testausprosessista ja sen tuloksien dokumentoinnista ja käytettyjen metodien selvittämisestä. Työssä tullaan käyttämään pääasiallisesti avoimen lähdekoodin työkaluja sekä alustoja. Hyökkäysalustana käytetään Kali-Linux käyttöjärjestelmää ja Open Web Application Security Project (OWASP) Zed Attack Proxy (ZAP) -turvallisuusskanneria.

Lopputuloksena on tietoa nykyisten sovellusten tietoturvasta toimeksiantajalle sekä yleisellä tasolla toistettava testausprosessi. Tuloksien pohjalta kehitetään raporttipohja, jonka avulla pystytään selittämään mahdolliset uhat ja toimenpiteet järkevästi niin johtoportaalle kuin IT-henkilöstölle.

ASIASANAT:

Injektio, Javascript, PHP, SQL, tietoturva, web-sovellus, tietoturvahaavoittuvuus

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Business Data Communications and Information Security

2016 | 35 pages

Sami Marttinen

WEB APPLICATION PENETRATION TESTING

The purpose of this thesis was to map the security of the Whitestone company web applications. The reason for this commission is the company's continued will to improve the quality and information security of their products and testing processes. The company in question produces a wide array of IT and digitalization services. Because of this, the information security of their products is vitally important for the company operations and image. The result of the thesis, is a generally repeatable testing process. The results of the tests will be used to create a report template. The template can be used so that the threats found can be explained to the management and IT personnel.

The subject of the thesis is very current. For the past few years information security has been a huge global subject in the IT industry. Many companies have been putting more resources into bolstering their information security and the marketing of it.

The theoretical part of thesis provides a literature review of security threats that target web applications, and the common prevention methods for these threats. The actual practical part of the thesis consists of planning the security testing, its process and the documentation of the results. In this thesis we mainly used open-source tools and platforms. As the attacker platform we used the Kali-Linux operating system and the Open Web Application Security Project (OWASP) Zed Attack Proxy (Zap) security scanner. The end result was information about the current application information security for the commissioner of the thesis and suggestions on fixing any potential security threats.

KEYWORDS:

Injection, Javascript, PHP, SQL, Information security, web application, vulnerability

SISÄLTÖ

KUVAT

KUVIOT

KÄYTETYT LYHENTEET TAI SANASTO

Autentikointi	Käyttäjän tunnistaminen ja todentaminen.
Avoin Lähdekoodi	Tapa kehittää ja jakaa tietokoneohjelmistoja ilmaiseksi.
BIOS	Basic Input/Output System
DOM	Dokumenttioliomalli
Eväste/keksi (cookie)	Dataa, jota web-palvelin tallentaa käyttäjän koneelle. Selain käyttää tätä tietoa tunnistamaan käyttäjät.
Isäntänimi	Nimi, jolla kone tunnistaa itsensä verkossa.
HTTP	Hypertext Transfer Protocol
JavaScript	Selaimessa ajettava komentosarjakieli.
MySQL	Avoimen lähdekoodin relaatiokanta.
NSA	National Security Agency
OWASP	The Open Web Application Security Project, voittoa tavoittelemation organisaatio, joka keskittyy web-sovellusten tietoturvaan.
Palvelin	Tietokone joka tarjoaa palveluita toisille tietokoneille kuten esimerkiksi sähköpostia ja tiedostonjakoa.
PHP	Hypetertext Preprocessor on komentosarjakieli, yleisesti käytössä web-palvelimissa vuorovaikutteisten sekä dynaamisten web-sivustojen luonnissa.
Ping-toiminto	TCP/IP -protokollien työkalu, jolla tutkitaan asetetun IP-osoitteen omaavan laitteen saatavuutta.
Virtualisointi	Käyttöjärjestelmien ja ohjelmien näennäinen asentaminen tietokoneelle, piilottaen resurssit toisiltaan.

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on suorittaa murtautumistestausta toimeksiantajalle Whitestone Oy. Tavoitteena on löytää annetusta sivustosta ja sen sisältämästä sisällönhallintasovelluksesta mahdollisia paikattavia haavoittuvuuksia. Testin tulosten raportoimiseen luodaan oma raporttipohja, jota voidaan myös käyttää tulevaisuuden testeihin. Testeissä käytetään pääasiallisesti OWASP ZAP -tietoturvatestaustyökalua.

Tietoturva on ollut viimeisten vuosien ajan maailmanlaajuisesti suosittu puheenaihe. Vuonna 2013 Edward Snowden julkaisi NSA:n salaisia dokumentteja, mikä sai ihmiset ja yritykset kyseenalaistamaan oman yksityisyytensä ja tietoturvansa. Nyt vuonna 2016 erityisesti mobiilisovelluksien tietoturva puhuttaa mediassa. Applen ja FBI:n välisen tietoturvakiistan seurauksena viestintäsovellusten tietoturva ja käyttäjien liikenteen salaaminen nähdään markkina-arvoa nostattavana tekijänä. Tämän seurauksena yritykset haluavat parantaa omien sovelluksien ja prosessien tietoturvaa. (Cordero, 2016.)

Työn lähdeaineisto koostuu monesta verkkolähteestä sekä osin alaan liittyvästä kirjallisuudesta. Perusteluna tähän on tietoturvan standardien, uudenlaisten uhkien ja hyökkäystapojen jatkuva kehitys. Painetusta materiaalista voi löytää vanhaa tietoa, joka ei enää pidä paikkaansa standardien muuttuessa. Verkon tietolähteet kuten OWASP (Open Application Security Project), pidetään ajan tasalla ja jatkuvasti päivitettyinä. Näistä syistä käytän tässä työssä suurimmaksi osaksi verkkolähteitä.

Tavoitteena on auttaa toimeksiantajaa löytämään mahdollisia tietoturvauhkia annetussa web-sovelluksessa. Testien pohjalta on tarkoitus antaa ehdotuksia niiden korjaamiseen sekä tuottaa testeistä raporttipohja, joka voidaan esittää niin IT-henkilöstölle kuin myös johtoryhmälle.

Opinnäytetyön tutkimusmenetelmänä toimii tapaustutkimus, kvalitatiivinen ja osaksi toimintatutkimuksellinen lähestyminen.

Tämän opinnäytetyön toimeksiantajana toimii digitoimisto Whitestone Oy. Whitestonella on toimisto Jyväskylässä, pääkaupunkiseudulla sekä Turussa. Yrityksen palveluihin kuuluu digitalisaatio, verkkosivujen luonti, verkkokaupat, räätälöidyt sovellukset sekä verkkomarkkinointi (Whitestone Oy, 2016). Koska yritys tarjoaa näin laajasti IT-alan palveluita, on tietoturva yksi yrityksen korkeimmista prioriteeteista. Tietoturvan parantamiseksi

ja yritysten sisäisten prosessien jatkuvan kehityksen takia, sain toimeksiannon tehdä murtautumistestausta yrityksen omalle sivulle ja sen sisältämään sovellukseen.

Sivustolla oleva sovellus on yrityksen itse kehittämä wManage-julkaisujärjestelmä. Kyseessä on PHP-pohjainen sovellus, jota yritys käyttää verkkosivujen kehitykseen. Sovellukseen annetaan myös asiakkaille tunnukset sivuston omaa muokkaamista varten. Testausta varten annetaan sivuston tuotantokäytössä olevasta versiosta kopio virtuaaliympäristöön asennettavaksi. Vaikka yrityksen prosesseihin kuuluu tietoturvan testaus jokaisen projektin kehityksen aikana, halutaan kuitenkin varmistaa, että mitään mahdollisia aukkoja ei löytyisi esimerkiksi skannereilla ja yleisesti käytetyillä hyökkäysohjelmilla. Tätä varten minulle annettiin toimeksiannoksi yrittää löytää tietoturva-aukkoja sivustosta ja sovelluksesta käyttäen ilmaisia skannereita ja ohjelmia. Testausvaiheessa toimeksiantaja on pyytänyt keskittymään erityisesti mahdollisiin Cross-Site Scripting (XSS) ja MySQL-injektio -haavoittuvuuksiin.

2 WEB-SOVELLUKSEN YLEISIMPIÄ HAAVOITTUVUUKSIA

Tässä luvussa käydään läpi web-sovelluksien yleisimpiä uhkia pääasiallisesti OWASPin 2013 top 10 -listan avustuksella. Syy tämän kyseisen listan käyttämiseen on sen yleinen käyttö tietotekniikan alalla sekä sen tämänkin hetkinen ajantasaisuus, vaikka lista on vuodelta 2013. Lisäksi toimeksiantaja käyttää OWASP top 10 -listaa tarkistaessaan omien tuotteidensa tietoturvaa.

Koska näihin haavoittuvuuksiin löytyy monta lähestymistapaa, rajoitan tässä kappaleessa olevat esimerkit yleisimpiin tapoihin. OWASP Application Security Verification Standard 3.0 -dokumentti on erinomainen lähde suositelluille turvallisuuskäytännöille. (OWASP Foundation 2015.)

Tätä top 10 -listaa hyödynnettäessä pitää kuitenkin ottaa huomioon tietoturvan laajuus käytännössä. Vaikka lista toimiikin hyvänä pohjana testauksessa, sitä ei kuitenkaan ole tarkoitus käyttää tietoturvan testauksessa rajoittavana tekijänä. Haavoittuvuuksia on monenlaisia, monissa eri muodoissa ja tämä lista ei kata niistä läheskään kaikkia. (OWASP Foundation 2013a.)

2.1 Injektio

Injektiohyökkäyksessä hyökkääjän tavoitteena on löytää tapa syöttää tietoa sivustolle niin, että sivusto tulkitsee sen osaksi nettisivun pohjalla olevaa ohjelmaa. Tämän haavoittuvuuden mahdollistaa virheellinen käyttäjän syöttämän tiedon käsittely. Haavoittuvuuden vakavuus piilee sen suurista sivunmuokkausmahdollisuuksista sekä tietojen, kuten esimerkiksi käyttäjätilien nimien ja salasanojen, antamisessa hyökkääjälle. Pahimmassa tapauksessa injektiohyökkäys voi antaa hyökkääjälle mahdollisuuden antaa komentoja kohdesivuston palvelimelle ja johtaa koko sivuston kaappaamiseen. Tärkein sääntö injektion estämiseen on olla luottamatta käyttäjän syötteeseen. (Weidman 2014, 319.)

2.1.1 SQL-injektio

Yleisin ja tunnetuin injektiotapa on SQL-injektio (Structured Query Language), jonka tavoitteena on murtautua sivustossa olevaan tietokantaan ja sen sisältämään dataan. SQL-injektiossa pyritään ujuttamaan SQL-komentoja käyttäjän antamaan syötteeseen. On erittäin tapauskohtaista, minkälaista syötettä sivustolle pitää antaa tuloksia saadakseen. Hyökkääjän pitääkin yleisesti ottaen arvailun ja kokeilun kautta muodostaa syöte, jolla saadaan aikaan haluttu vaikutus. Jos tavan kuitenkin löytää, niin se voi antaa hyökkääjälle mahdollisuuden muokata tietokantaa ja kaikkia sen sisältämiä tietueita.

Seuraavassa esimerkissä oletetaan sivustolla olevan MySQL-tietokanta ja käyttäjää ei estetä antamasta haitallista syöttöä. Palvelimella on seuraavanlainen koodinpätkä, jossa käyttäjän syötettä ”Id” tekstikenttään käytetään hakemaan tietoa ”Users” -tietokannasta:

```
txtUserId=getRequestString("UserId");

txtSQL="SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Hyökkääjä voisi antaa seuraavan syötteen:

```
105 or 1=1
```

Ylempi syöte siirtyisi palvelimelle muodossa:

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

Tässä tilanteessa se palauttaisi hyökkääjälle kaikki rivit ”Users” -taulusta. (W3schools 2016.)

Yleisin ja helpoin tapa estää SQL-injektioita on käyttää parametrisoituja kyselyjä (Parameterized Queries). Parametrisoiduilla kyselyillä saavutetaan lopputulos, jossa syötteeseen ujutettu koodi muutetaan muotoon, missä se ei pääse tietokantaan asti. Parametrisoitu kysely toimii niin, että kehittäjä määrittää kaiken SQL-koodin ja lähettää sen vasta sen jälkeen eteenpäin. Tällöin ohjelma voi tunnistaa ja ohittaa syötteeseen piilotetut komennot.

Esimerkki parametrisoidusta kyselystä PHP:ssa:

```
$stmt = dbh->prepare(”INSERT INTO REGISTRY (name, value) VALUES (:name, :value”);

$stmt->bindParam(“’ :name’, $name);

$stmt->bindParam(‘:name’, $name);

$stmt->bindParam(‘:value’, $value);
```

(Manico ym. 2014).

2.1.2 Cross-Site-Scripting (XSS)

XSS-haavoittuvuus on myös eräänlainen injektio, jossa hyödynnetään kaikkia mahdollisia HTTP-pyyntöön liittyviä syötteitä. Suurin ero XSS:n ja SQL-injektion välillä on pääasiallinen kohde. XSS-haavoittuvuus kohdennetaan sivuston käyttäjään, eikä niinkään itse palveluun. Kyseessä on erittäin yleinen haavoittuvuus, jota on löytynyt monien tunnettujen yritysten, kuten Twitterin ja Facebookin palveluista. Yleisen ja laajalle levittyneen haavoittuvuudesta tekee sen ominaisuus tulla hyödynnetyksi monissa yleisimmissä ohjelmointikielissä.

Hyökkääjän tavoitteena on syöttää haavoittuvuuden kautta haitallisia ohjelmascriptejä, jotka suoritetaan käyttäjän selaimessa. Joskus nämä scriptit suoritetaan sivuston taustalla ja käyttäjä ei ole niistä millään tavalla tietoinen. Hyökkääjän kohteena on päästä käsiksi käyttäjälle luotuun sessioon, sekä evästeihin. Näiden tietojen avulla voidaan kohdepalveluun päästä käsiksi toisen käyttäjän nimellä ja tiedoilla ilman kirjautumista. Hyökkääjä voi pystyä haavoittuvuuden kautta jopa uudelleenkirjoittamaan HTML-sivuston sisältöä. (OWASP Foundation, 2016a.)

XSS-hyökkäykset voidaan jakaa kolmeen kategoriaan: 1. 2. ei-pysyvään (reflected) ja 3. DOM-hyökkäykseen (Document Object Model).

- pysyvään (stored)
- ei pysyvään (reflected)
- DOM-hyökkäykseen (Document Object Model)

Pysyvällä hyökkäyksessä tarkoitetaan hyökkäystä, jossa haavoittuvuudesta on tehty pysyvä osa esimerkiksi nettisivun tietokantaa. Tällöin jokainen käyttäjä, joka tekee tietokantaan kyselyjä lataa tietämättään haitallista koodia selaimeen.

Ei-pysyvässä hyökkäyksessä hyökkäys kohdistuu yksittäisiin käyttäjiin esimerkiksi lisäämällä haitallista koodia linkkiin, joka vie sivustolle, missä haavoittuvuus sijaitsee. Koska tämä tapa tarvitsee vain haitallisen linkin, sen levittäminen on helppoa vaikka sähköpostin tai minkä tahansa chat-palvelun kautta.

DOM-hyökkäys on hyvin samankaltainen ei-pysyvän hyökkäyksen kanssa. Siinäkin käytetään hyödyksi haavoittuvalle sivustolle vieviä saastuneita linkkejä. Haavoittuvuus käy toteen, jos sivun osoitteessa olevia parametreja käytetään kirjoittamaan sivustolle sisältöä, mutta ennen tämän tiedon lisäämistä, ei tarkisteta siihen mahdollisesti sisällytettyä HTML-koodia. Näissä tilanteissa parametrina annettu HTML lisätään sivulle sellaisenaan, milloin se voidaan suorittaa käyttäjän selaimessa. (OWASP Foundation, 2013b.)

XSS:n estämiseen löytyy monia menetelmiä riippuen käytetyistä koodikielistä ja alustoista. Yleisiä menetelmiä ovat muun muassa:

Whitelistin käyttö, jolloin rajoitetaan minkälaista syötettä käyttäjä voi antaa sivustolle rajaamalla listan avulla käytettäviä merkkejä vain sellaisiin arvoihin mitä oletamme käyttäjältä kyseiseen kenttään. Tämänkaltaisia rajauksia ei voi käyttää kaikkialla, mutta esimerkiksi ”käyttäjänimi” -kentissä sitä voidaan käyttää, koska käyttäjänimen voi yleisesti rajoittaa kirjaimiin ja numeroihin ilman käyttäjälle aiheutuvaa vaivaa. (OWASP 2016a.)

Syötteen puhdistaminen on toinen tapa estää ei-haluttujen komentojen suorittamista sivustoilla. Puhdistamisessa muunnetaan esimerkiksi lomakkeisiin annettua syöttöä tavalla, joka ”siivoaa” sen ei-halutuista merkeistä ennen kuin se lähetetään eteenpäin. Käytännössä tämä tarkoittaa sitä, että jos käyttäjältä odotetaan sähköpostia, sovelluksella ei ole syytä hyväksyä muita syötteitä kuin kirjaimia, numeroita ja @-merkkiä. (OWASP 2016a.)

Yksi helpoimmista tavoista on käyttää ohjelmakirjastoa (esimerkiksi Sequelize), jonka avulla puhdistaminen voidaan suorittaa sen sisälle rakennettujen toimintojen kautta.

2.2 Haavoittuva todennus ja sessionhallinta

Web-sovelluksissa todennusta käytetään todistamaan käyttäjän identiteetti. Todentamiseen käytetään esimerkiksi keksejä (cookies) tai tokeneita. Sovelluksesta riippuen nämä arvot annetaan sivulle saapuessa ja erikseen onnistuneen kirjautumisen yhteydessä, jonka jälkeen käyttäjälle luodaan sessio. Koska kirjautumista käytetään antamaan käyttäjälle enemmän toiminnollisuuksia sovelluksessa, on sovelluksen tarkoitus pitää kaikki käyttäjiin liittyvät todennustiedot salassa ulkopuolisilta.

Jotta turvallinen todennus ja sessionhallinta toteutuisivat, on sovelluksen noudatettava tietynlaista toimintamallia sekä erilaisia käytäntöjä pitääkseen tiedot salassa ja piilotettuna muilta kuin tarkoitetuilta käyttäjiltä. Nykyajan nettisivuilla käyttäjän tietoja tallennetaan jo ennen kirjautumista keksien avulla. Keksit ovat yleisimmin käytetty selaimen tapa tallentaa käyttäjäkohtaisia tietoja käyttäjän koneelle ennen kirjautumista. Tähän on syynä tarve pitää tallessa tiettyjä valintoja, kuten sivuston kieli tai väriteema, mihin ei ole tarvetta pyytää käyttäjältä todennusta erikseen. (Siles 2016.)

Kuviossa 1. näytetään todennuksen (Authentication) ja sessionhallinan (Session Management) periaatteinen toiminta.

Kuvio 1. Session luonti- ja lopetusprosessi. (Siles 2016).

Käyttäjä yrittää kirjautua sisään, jolloin sivu kysyy todennusta. Todennuksen onnistuessa käyttäjälle luodaan hallittava sessio. Jos tämän tiedon suojauksessa on virheitä, saattaa mahdollinen hyökkääjä pystyä kaappaamaan toiselle käyttäjälle tarkoitetun session. Jos kyseessä olisi esimerkiksi verkkokauppa, hyökkääjä voisi tilata tuotteita toisen käyttäjän nimissä, käyttäen tilille tallennettuja tietoja. Pääsynhallinta (Access Control) tarkoittaa käyttäjän oikeuksien rajoittamista sovelluksessa. Sillä määritetään, mitä toimintoja sovellus sallii käyttäjän käyttävän. Tämä on yksi tapa rajoittaa mahdollista hyökkäysrajapintaa. (Siles 2016.)

Session ID tai token on usein sovelluksen kehittäjän määrittämä ja siksi onkin tärkeää, että sen määrittely suoritetaan tavalla, jonka tuloksena olisi turvallinen kokonaisuus. Yleisimpiä keinoja tämän saavuttamiseksi on:

- Itse ID:n nimeäminen jollain yleisellä nimellä. Nimeäminen ”PHP_ID:ksi” ei ole suositeltavaa, sillä se kertoo hyökkääjälle, millä kielellä ID luodaan ja saattaa helpottaa murtautumista sivustolle. Tämän sijaan on parempi nimetä ID esimerkiksi pelkäksi id:ksi.
- ID:n pituuden pitäisi olla tarpeeksi pitkä kestääkseen ”brute-force” -hyökkäyksiä. Vähintään 128-bitin pituus on suositeltavaa.
- Kaikkien ID-tietojen lähetys tulee hoitaa salattujen yhteyksien yli, jotta niitä ei voida varastaa lähetyksen yhteydessä.
- ID:n luonti ei saa olla ennalta-arvattavaa. Satunnaisen ID:n luontiin kannattaa käyttää hyvää pseudo-satunnaislukugeneraattoria saavuttaakseen turvallisia ID-arvoja.
- Session ID ei saa sisältää mitään tärkeää tietoa kuten luottokorttien tietoja tai salasanoja. Tarkoituksena on, että ID toimii pelkkänä tunnisteena käyttäjälle.
- Mitään tärkeää tietoa ei tallenneta sivuston URL-osoitteeseen. (Keary 2016.)

2.3 Vääränlaiset turvallisuuskonfiguraatiot

Vaikka itse Web-sovelluksen tietoturvasta puhutaan paljon, kuuluu myös ottaa huomioon sovellusta ylläpitämän palvelimen tietoturva ja konfiguraatiot. Palvelimen tietoturvan kannalta on tärkeää, että kaikki ohjelmistot ja alustat ovat päivitettyjä ja ajan tasalla. Vanhemmissa versioissa olevat tietoturva-aukot aiheuttavat uhkia. Turhien ohjelmien olemassaolo voi myös aiheuttaa ongelmia, sillä nekin voivat olla haavoittuvaisia. Tämän takia on suositeltavaa, että palvelimella pidetään vain ne ohjelmat, joita tarvitaan sovelluksen toimintaan ja kaikki ylimääräiset poistetaan. (Osborne 2006, 284.)

Palvelimen antamat virheilmoitukset eivät myöskään saisi paljastaa sitä, mikä aiheuttaa virhettä käyttäjälle. Tämänlaisen virheen kautta hyökkääjä pystyy näkemään millä alustalla tai mitä sovelluksia sivuston taustalla on ja millä palvelimella. Vaikka tämä ei ole välttämättä ole suora haavoittuvuus, se voi helpottaa hyökkääjää löytämään haavoittuvuuksia. Tähän tietoturvan alueeseen kuuluu myös olemassa olevien asennusten turvallisuus. Ohjelmien asennuksen jälkeen pitää aina tarkistaa, etteivät oletussalasanat ja konfiguraatiot ole käytössä. (OWASP Foundation, 2013c.)

Helpoin tapa vähentää virheitä palvelimen konfiguraatioissa on tuottaa automatisoitu prosessi niiden luontiin. Tämä tapahtuu luomalla yksi turvallinen instanssi palvelimesta ja sen konfiguraatioiden pohjalta luoduilla asennusskripteillä luodaan käyttövalmis pohja. Tämänkaltaisessa pohjassa on myös muuttuvia arvoja, kuten esimerkiksi salasanat, jotka tulee luoda aina erikseen. Turvalliseksi todetun palvelimen pohjalta tehdyt skriptit voidaan käyttää uudelleen seuraavan palvelimen luontiin. Näin uusien palvelimien asennus tulee olemaan identtinen edellisen kanssa. Tällä käytännöllä mahdolliset tietoturva-aukot voidaan helposti korjata kaikilla palvelimilla yhtä aikaa korjaamalla aukko palvelin ja muokkaamalla skriptiä sen estämiseksi. Lisäksi asennusprosessi nopeutuu ja yksinkertaistuu. Tietyin aikavälein on myös suositeltavaa suorittaa tietoturva-auditointeja. (OWASP Foundation, 2013c.)

2.4 Tiedon salaus

Tämä otsikko kattaa erittäin laajan kokonaisuuden, joten kappaleessa keskitytään kaikista olennaisempiin osiin. Mikä tahansa herkkä tieto, jota liikutetaan sovelluksessa ja joka voi aiheuttaa haittaa tietoturvalle, tulee jollain tavalla salata. Jos liikennettä ei salata, se altistuu vakoilulle. Tämä tieto voi olla palvelimen ja web-sovelluksen välillä liikkuva tietoa, tai asiakkaan ja web-sovelluksen välinen liikenne.

liikenteen salaus voidaan toteuttaa Secure-Sockets Layerillä (SSL). Tämä on yleisesti käytetty tapa luoda ja kryptata linkki selaimen ja palvelimen välille. Tärkeää on pitää huoli siitä, että mitään tällaista tietoa ei lähetetä tai varastoida niin sanotussa ”clear-text” -formaatissa, jolloin liikkuva teksti on vapaasti luettavissa kenelle tahansa liikennettä seuraavalle. Tilanteessa, jossa tätä formaattia käytetään salaamattomassa liikenteessä (esimerkiksi session kekseissä), on hyökkääjän helppo seurata sivuston liikennettä ja varastaa käyttäjän session keksi. Tämän jälkeen hyökkääjä voi kaapata session omaan käyttöönsä. Myös kaikki väliaikainen data tulee poistaa heti, kun sitä ei tarvita.

Herkkien tietojen salaamiseen on käytettävä tarpeeksi vahvoja salaustekniikoita, jotta niiden murtaminen ei olisi hyökkääjälle helppoa. Yleisesti luotettuihin tekniikkoihin kuuluvat Advanced Encryption Standard (AES) ja RSA julkisten avainten kryptaamiseen, sekä SHA-256 hashaamiseen. Kaikkien käyttäjien avainten ja salasanojen uusiminen tietyin väliajoin on myös tärkeää. (OWASP Testing Guide 4.0, 16.)

2.5 Cross-site Request Forgery (CSRF)

CSRF-hyökkäyksessä hyödynnetään nettisivun luottoa käyttäjän selaimeen. Käyttäjä on haavoittuvainen ollessaan kirjautuneena johonkin selaimen sovellukseen. Samalla kun käyttäjä on kirjautuneena sovellukseen, saattaa hän avata saastuneen linkin, joka suorittaa komentoja selaimen taustalla. Komentojen avulla hyökkääjä voi käyttää kirjautuneelle käyttäjälle tarkoitettuja ominaisuuksia, esimerkiksi pankkisivustolla rahan siirron käyttäjän tililtä hyökkääjän tilille. Itse käyttäjä ei todennäköisesti ole millään tavalla tietoinen hyökkäyksestä. Tämä hyökkäys voidaan suorittaa niin kauan kuin käyttäjän sessio on avoinna ja hyökkääjän käytettävissä. (Weidman 2014, 335.)

3 TESTAUSYMPÄRISTÖ

Testauksen alustaminen ja suunnittelu ovat tärkeä osa tietoturvatestausprosessia. Suunnitteluvaiheessa mietitään, mitä tapoja käytetään ympäristön tiedonkeräykseen sekä itse testauksen suorittamiseen. Koska kyseessä on testaus, jossa on monta muuttujaa, voi tämä suunnitelma vaihtaa muotoaan testauksen edetessä riippuen tuloksista ja valittujen ohjelmien toimivuudesta.

Tiedonkeräyksessä käydään läpi kohdepalvelimen konfiguraatiota sekä katsotaan, mitä tietoja ulkopuolinen voisi siitä nähdä. Virtuaaliympäristö toimii tässä rajoittavana tekijänä.

Tässä työssä suoritettavat testit tehdään suljetussa virtuaaliympäristössä. Ympäristössä on asennettuna testausta suorittava kone sekä toimeksiantajan asetuksia käyttävä palvelin, joka on konfiguroitu olemaan identtinen tuotantokäytössä olevan palvelimen kanssa. Palvelin sisältää nettisivun, jossa on myös erillinen sisällönhallinta web-sovellus. Sekä nettisivu että sen sisältämä sovellus toimivat molemmat testauskohteina. Luvussa käydään osaksi läpi testausympäristön rakentamisen prosessia. Väliin jätetään itse käyttöjärjestelmän asennus, koska prosessi on virtuaalikoneelle sama kuin normaalille koneelle ja on monesti dokumentoituna lukuisissa lähteissä.

3.1 Virtuaaliympäristön suunnittelu

Virtuaaliympäristön pohjana päätettiin käyttää Microsoftin omaa virtualisointiohjelmaa Hyper-V:tä. Ohjelman käyttöön tarvitaan Pro tai Enterprise versio 7, 8, tai 10 Windowsista tai 2008 tai uudempi Windows Server. Tähän perusteluna on toimeksiantajan ohjelman käyttö omassa työympäristössä. Suljettu virtuaaliympäristö koostuu kahdesta koneesta: testauksen kohteena toimivasta palvelinkoneesta ja hyökkääjänä toimivasta toisesta koneesta.

Palvelinkone toimii CentOS 7 Linux -käyttöjärjestelmällä (Community Enterprise Operating System). CentOS on Red Hat Linux -käyttöjärjestelmään pohjautuva ilmainen versio, jota kehitetään oman yhteisön avuin täysin erillisenä Red Hatista. Kone asennetaan ilman graafista käyttöliittymää, kuten on yleisesti tapana palvelinkoneissa. (The CentOS Project 2016.)

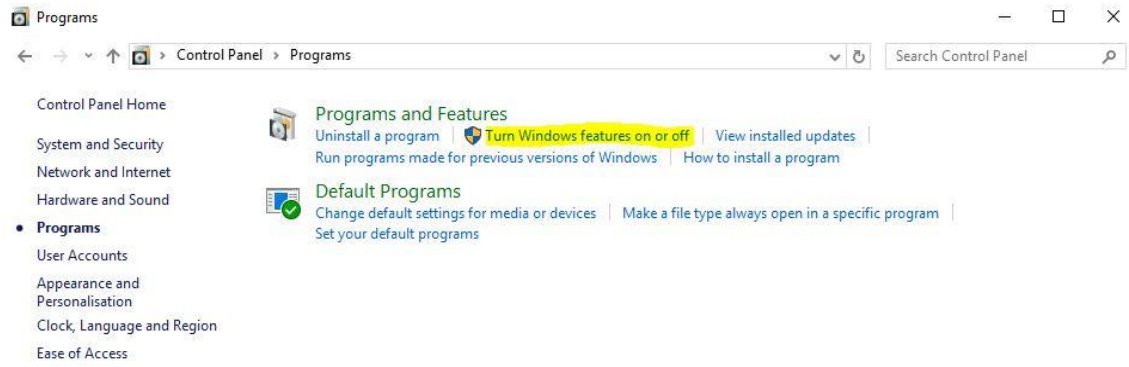
Hyökkääjäkoneeseen asennetaan Debianiin pohjautuva Kali-Linux käyttöjärjestelmä. Kyseessä on murtautumistestaukseen luotu, avoimen lähdekoodin käyttöjärjestelmä, josta löytyy valmiiksi asennettuna yli 600 työkalua murtautumistestaukseen (Offensive Security 2016). Kuvassa 1 näkyy Kali-Linuxen työkalut -näkymä.

Kuva 1. Kali-Linux työkalut -näkymä.

Hyper-V:n virtuaalikoneen luonti on aika yksinkertainen prosessi. Ensimmäisenä pitää kuitenkin varmistaa, että koneessa on virtualisointi päällä. Monissa koneissa asetus on oletuksena pois päältä.

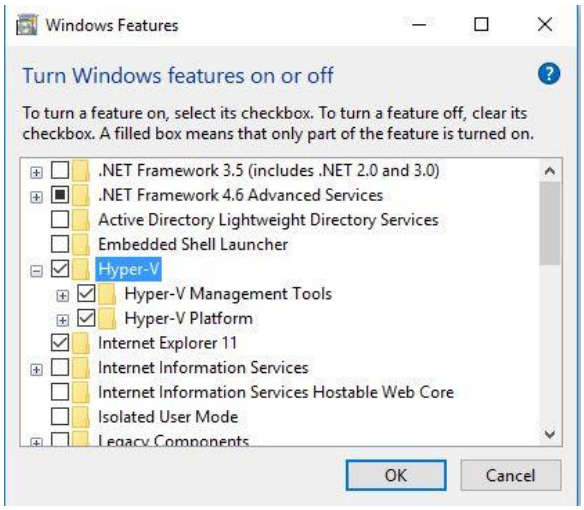
3.1.1 Virtualisoinnin aktivointi

Virtualisointi aktivoidaan ensin BIOS:n kautta prosessorin valikosta. Prosessorista riippuen pitää aktivoida joko Intel Virtualization tai AMD-V. Tämän jälkeen virtualisointi pitää vielä aktivoida Windowsin asetuksista: Control Panel -> Programs and Features-valikosta avaamalla Turn Windows features on or off (kuva 2).



Kuva 2. Turn Windows features on or off -valikon sijainti.

Tästä avautuvasta valikosta rastitetaan virtualisointi päälle (kuva 3)

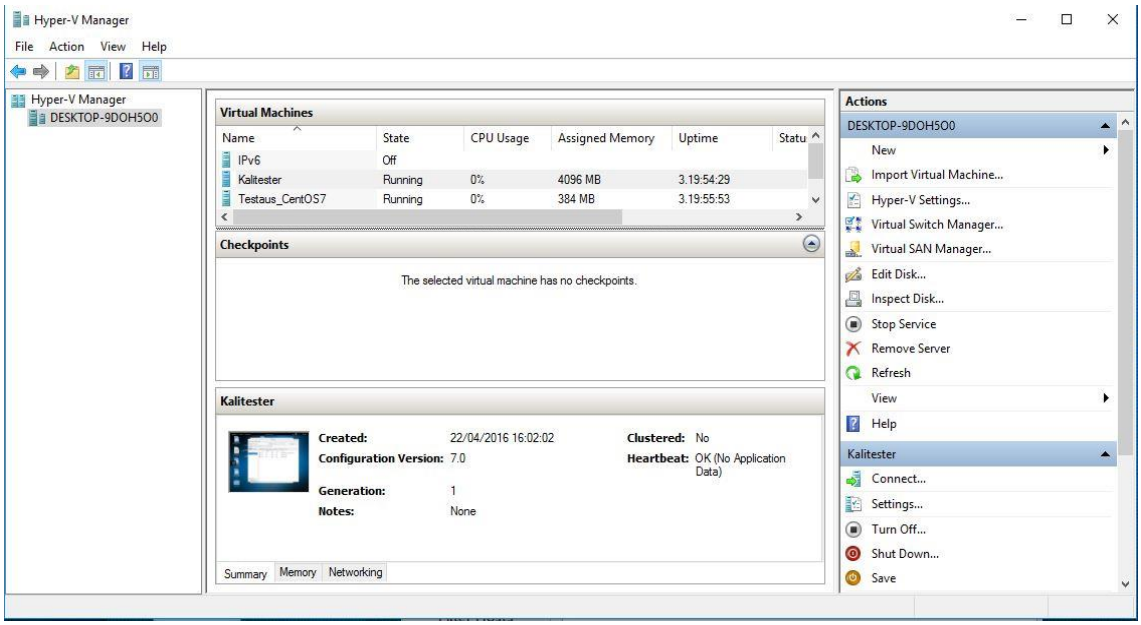


Kuva 3. Hyper-V:n aktivointi Windowsissa.

Tämän jälkeen painetaan OK, käynnistetään kone uudelleen, jonka jälkeen kone on valmis luomaan virtuaalikoneita Hyper-V:llä. (Microsoft 2016a.)

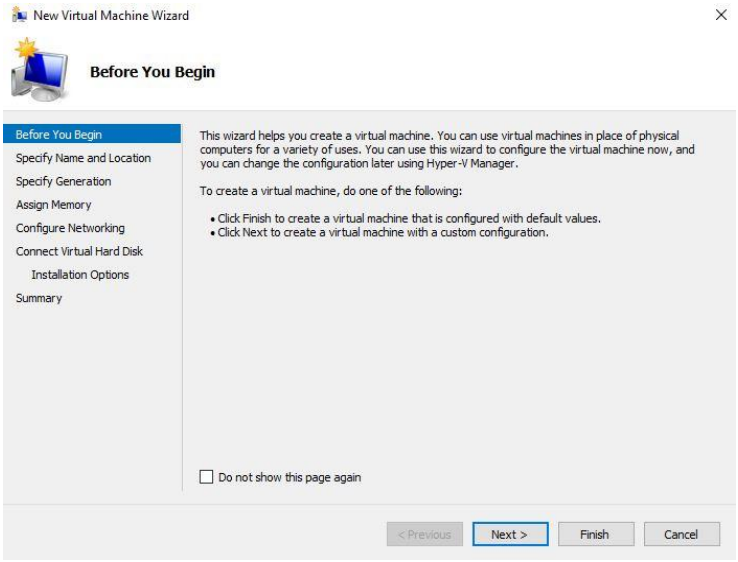
3.1.2 Virtuaalikoneen luominen Hyper-V:llä

Aloitetaan virtuaalikoneen asennus avaamalla Hyper-V Manager (kuva 4).



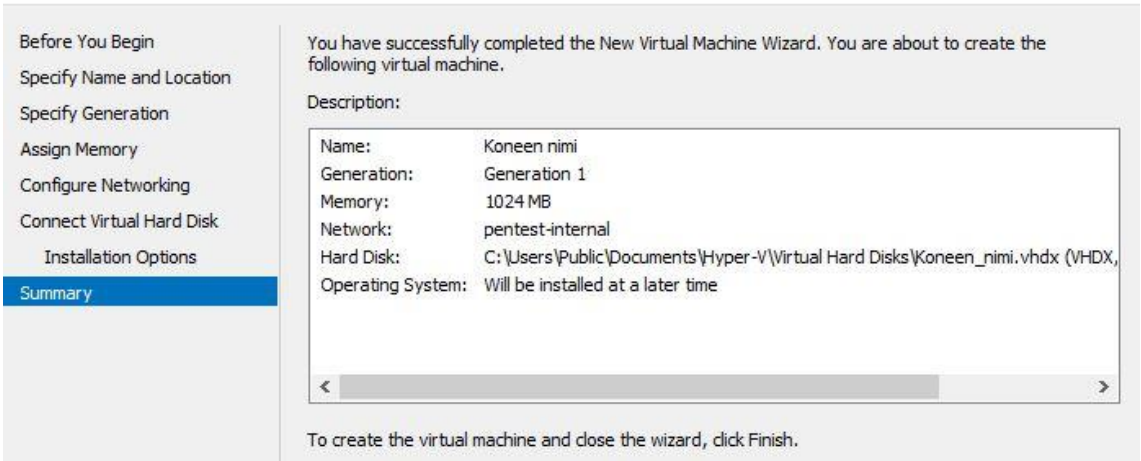
Kuva 4. Hyper-V:n avautumisnäkymä.

Kuvan 4 näkymässä näkyy asennetut virtuaalikoneet, niiden hallinnointi sekä uusien koneiden luonti ovat näkyvissä. Jos halutaan luoda uusi virtuaalikone, painetaan Actions -valikosta New ja valitaan Virtual Machine. Tämän jälkeen avautuu virtuaalikoneen asennusikkuna (kuva 5).



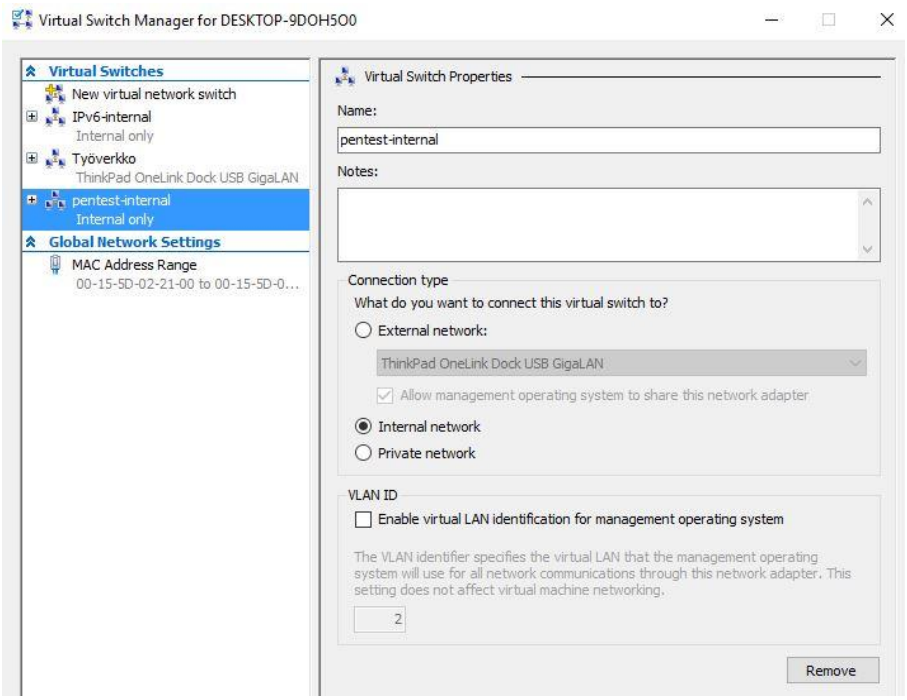
Kuva 5. Virtuaalikoneen asennusikkuna.

Asennuksessa virtuaalikoneelle määritellään nimi, sukupolvi (generation), muisti, verkko ja kovalevy. Sukupolvea valitessa pitää ottaa huomioon mitä käyttöjärjestelmää ollaan asentamassa. Windowsin sivustolla on listaus vieraskäyttöjärjestelmistä ja niiden sukupolvituista (Microsoft 2016b). Sukupolvea ei pysty vaihtamaan sen valinnan jälkeen. Jos aloittaa täysin tyhjältä pohjalta, verkkosovitinta ei todennäköisesti ole vielä asetettuna, joten tässä tilanteessa verkkosovittimen valinnan voi jättää tyhjäksi. Jos käyttöjärjestelmän asennustiedostoa ei ole vielä ladattu voi valita käyttöjärjestelmän asentamisen myöhemmin. Lopuksi asennus näyttää yhteenvedon asetuksista (kuva 6).



Kuva 6. Tiivistelmä asennettavan virtuaalikoneen asetuksista.

Verkkosovittimen luonti aloitetaan kuvassa 4 olevassa ikkunassa avaamalla Virtual Switch Manager. Avautuvassa näkymässä (kuva 7) voidaan valita kolmenlaisesta virtuaalikytkintyypistä External (ulkoinen), Internal (sisäinen) tai Private (yksityinen). Luodaan sisäinen verkko, koska tavoitteenamme on kahden virtuaalikoneen välinen yhteys, joka ei yhdisty fyysisen koneen verkkoon. Tätä verkkoa tullaan käyttämään molemmissa virtuaalikoneissa. Luonnin jälkeen voimme nimetä verkon, hyväksyä asetukset ja jatkaa eteenpäin.



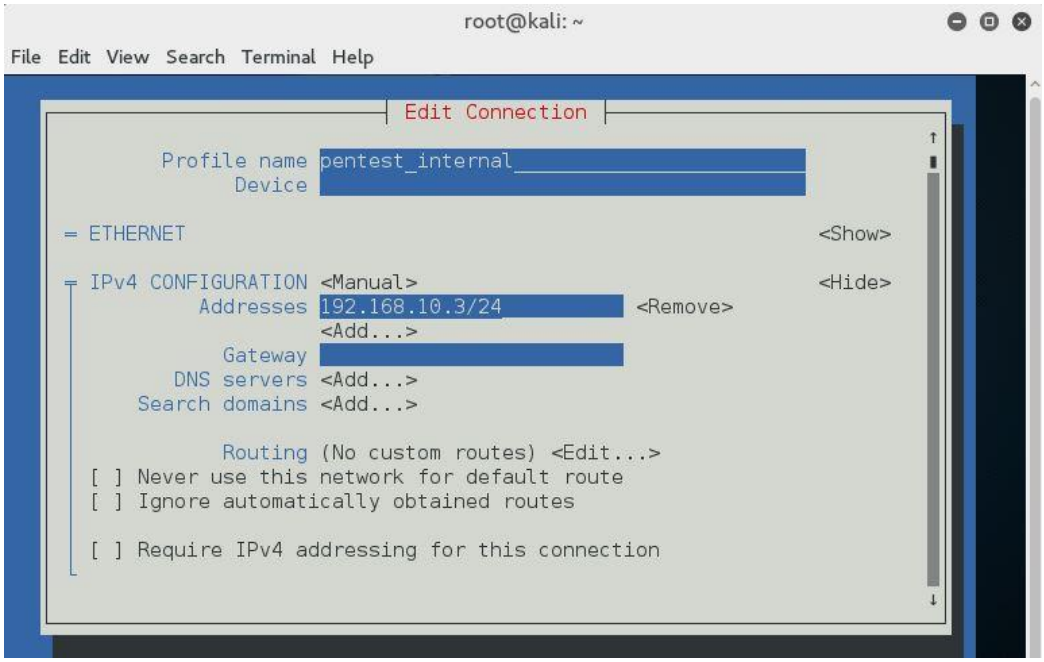
Kuva 7. Luotu ja käyttövalmis sisäinen virtuaalikytkin.

Kun tällä tavoin on asennettu kaksi virtuaalikonetta, voidaan niihin asentaa käyttöjärjestelmät ja konfiguroida koneet samaan verkkoon. Verkossa on kaksi Linux-pohjaista konetta: testausta suorittava kone, jossa on käytössä Kali-Linux, sekä testauksen kohde, jossa on asennettuna CentOS 7. Molemmissa koneissa verkon konfigurointi onnistuu parhaiten käyttämällä komentorivissä nmtui -komentoa (NetworkManager text user interface). Täältä voi myös asettaa koneelle isäntänimen. Avautuvassa näkymässä siirrytään paikkaan Edit a connection. Avautuvassa ikkunassa määritetään koneiden IP-osoitteet. IP-osoitteet määritettiin seuraavasti:

CentOS 7: 192.168.10.2

Kali-Linux: 192.168.10.3

nmtui -näkymässä Kali-Linux -koneen asetukset näyttävät Kuvan 8 osoittamalla tavalla.



Kuva 8. Kali-Linuxin asetukset nmtui -näkymässä.

Voi olla tarpeellista avata ja sulkea yhteys Activate connection -näkymästä asetusten käyttöönoton jälkeen molemmissa koneissa. Tämän jälkeen koneiden pitäisi pystyä näkemään toisensa sisäisessä verkossa. Tämän voi todentaa pingaamalla molemmilla koneilla toisen koneen IP-osoitetta. Jos haluaa varmistaa koneen olevan yhteydessä vain sisäiseen verkkoon, voi koittaa pingata Googlen DNS-palvelimia osoitteessa ”8.8.8.8”. Kun koneet eivät saa ulkoverkkoon yhteyttä ja näkevät toisensa verkossa, on luotu toimiva testiympäristö.

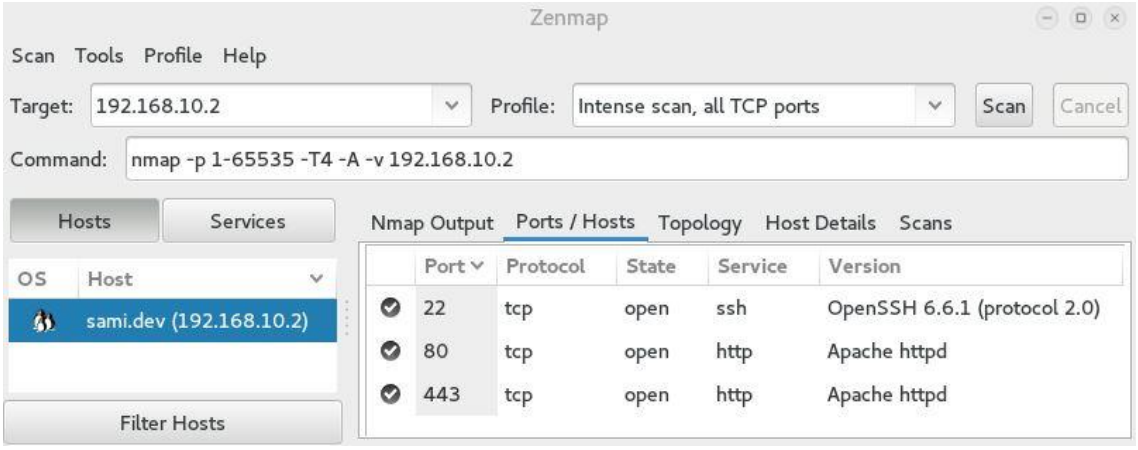
4 WHITESTONEN WEB-SOVELLUKSEN SKANNAUS

Tässä kappaleessa käydään läpi käytetyt ohjelmat ja testausmetodit, joilla kerättiin tietoa kohteista ja niiden ympäristöstä. Skannausvaiheessa yksi tavoitteista on varmistaa, että testauksesta ei koidu haittaa toimeksiantajalle, asiakkaille tai verkkoympäristölle.

4.1 Tiedonkeräys

Tietoturvallisuuden kannalta tiedonkeräykseen on monta tapaa. Tässä tapauksessa käytetään Network Mapperia (Nmap) testaamaan mitä tietoja hyökkääjä voi nähdä palvelimen porttiasetuksista ja näiden porttien takana olevista ohjelmista. Tämän jälkeen käydään läpi mitä ohjelmia ja asetuksia testaajana tiedämme palvelimella olevan.

Nmappia käytetään kohteen porttien skannaamiseen ja niiden tilan tarkistamiseen. Avoimet portit, joista ei olla tietoisia, voivat toimia mahdollisina hyökkäysalustoina. Testissä käytetään Zenmappia, joka on graafisen käyttöliittymän omaava versio Nmapista (kuva 9). Kohteeksi asetetaan palvelinkoneemme ja profiiliksi valitsemme ”Intense scan, all TCP ports”, joka skannaa kaikki palvelinkoneen TCP portit väliltä 1-65535.



Kuva 9. Zenmap porttiskannauksen tulokset.

Kuten kuvasta 9 näkyy, löytyi kolme auki olevaa porttia. Kyseiset portit 22, 80 ja 443 ovat kuitenkin yleisesti palvelimissa auki olevia portteja koska ne ovat http- ja ssh-palveluja

varten tarvittavia portteja. Tässä tilanteessa tämä ei kuitenkaan anna meille hyökkäysrajapintaa, koska kohdepalvelimen palomuuri on asetettu hyväksymään vain tietyt IP-osoitteet ja torjumaan muut yhteyspyynnöt palvelimelle.

4.2 OWASP Zed Attack Proxy

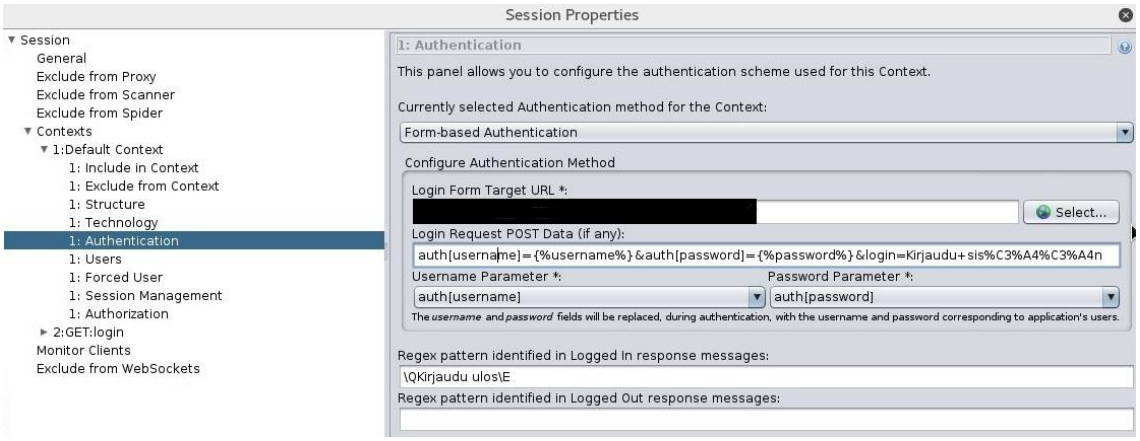
OWASP ZAP on ilmainen turvallisuustestaustyökalu. Se on suunniteltu olemaan hyödyllinen tietoturvan aloittelijoille sekä myös kokeneemmalle testaajalle. Se on hyvin kattava työkalu automatisoituun sekä manuaaliseen testaukseen. OWASP ZAPia käytettiin pääasiallisena työkaluna tämän työn testausprosessissa. Ohjelman avaamisen jälkeen (Kuva 10) ohjelmalle annetaan testattavan web-sovelluksen osoite.

Kuva 10. OWASP ZAPin aloitusruutu.

Testauksen voi aloittaa painamalla Attack painiketta. Kuvan 10 vasemmassa yläreunassa voimme valita skannauksen moodin. Testaamisen valittiin Standard Mode, joka skannaa sivuston ja yrittää löytää sekä todentaa siitä löytyviä haavoittuvuuksia. Skannaus alkaa Spider -työkalun skannauksella. Spiderin tavoitteena on kartoittaa verkkosivun kokonaisuus tarkistamalla kaikki sivustot ja osoitteet, mitä se löytää. (OWASP Testing Guide 4.0, 52.)

Kun Spider on kartoittanut sivuston, ohjelma aloittaa näille sivustoille Active Scan -vaiheen. Tässä vaiheessa ohjelma yrittää löytää yleisesti käytetyillä metodeilla haavoittuvuuksia sivustosta. Testattavat haavoittuvuudet voidaan määrittää erikseen ohjelman asetuksista. Tätä testiä varten pidettiin oletusasetukset, jotta skannaus olisi kattava.

Kohdesivuston web-sovelluksen testausta varten pitää ohjelmaan antaa enemmän parametreja Session Properties -> Context -> Authentication -ikkunassa (kuva 11). Ohjelmalle pitää määritellä sovellukseen kirjautumiseen tarvittavat tiedot. Testausta varten ohjelmaan luotiin testitunnukset, joilla ohjelma kirjautui sisään. Spider onnistui itse löytämään kirjautumiseen käytettävät parametrit. Tämän jälkeen tarvitsi vain määrittää tarkemmin kirjautumiseen käytettävät tiedot. Tärkeää on myös määrittää tapa, jolla ohjelma tietää onnistuneen kirjautumisen. Yleisesti ottaen tämä voi olla jonkinlainen tekstinpätkä tai tiedosto kirjautumisen jälkeisellä sivulla. Testattavan sovelluksen tapauksessa sivustolla määritettiin lukevan ”Kirjaudu ulos”. Ohjelma siis määrittää kirjautumisen onnistuneen, kun se löytää sivustolta ”Kirjaudu ulos” -tekstin. Tämän jälkeen annamme vielä käyttäjänimi ja salasanaparametriin annettavat tiedot ”Users” ikkunassa.



Kuva 11. Sovellukseen kirjautumisen asetukset.

Testin aikana OWASP ZAP listaa epäillyt haavoittuvuudet, niiden todennustavan ja korjausehdotukset. Skannereita käytettäessä pitää kuitenkin ottaa huomioon, että listatut haavoittuvuudet eivät välttämättä ole tarkkoja. Testaajan työnä on tarkistaa, ovatko skannerin löytämät haavoittuvuudet oikeita vai eivät.

4.3 Jatkotestaus

Aikaisempien testien tueksi kohde tarkistetaan myös käyttäen Metasploit sekä SQLmap -työkaluja. Mitä enemmän työkaluja käytetään testaukseen, sen luotettavimpia ovat myös sen tulokset. Metasploit on yksi maailman käytetyimmistä murtautumistestausso-velluksista. Metasploittia ei käytetä työssä pääasiallisena työkaluna, koska ohjelmasta on ilmainen, sekä maksullinen versio. Ilmainen versio toimii ainoastaan käyttöjärjestel-män, sekä tietokannan skannaukseen. OWASP ZAPissa saa käyttöön kaikki ohjelman työkalut ilmaiseksi.

Metasploitissa hyödynnetään sen sisälle rakennettua WMAP web-skanneria, joka oli alun perin SQLmappiin suunniteltu ominaisuus. SQLmap on erityisesti SQL-injektio haa-voittuvuuksiin erikoistuva työkalu. Tässä testauksessa SQLmappia käytetään vahvista-maan testauksen tarkkuutta SQL-injektio haavoittuvuuksien osalta. Testissä hyödynne-tään SQLmappiin löydettyä automatisoitua testiä.

4.3.1 Metasploit

Metasploitin käytön aloittamiseksi, tarvitaan Kali-Linux koneelle käyntiin Metasploitin hyödyntämä PostgreSQL-tietokanta. Tämä suoritetaan komennolla:

```
service postgresql start
```

Kun tietokanta on toiminnassa, luodaan vielä msf-tietokanta ja käynnistetään Metasploit komennoilla:

```
msfdb init
```

```
msfconsole
```

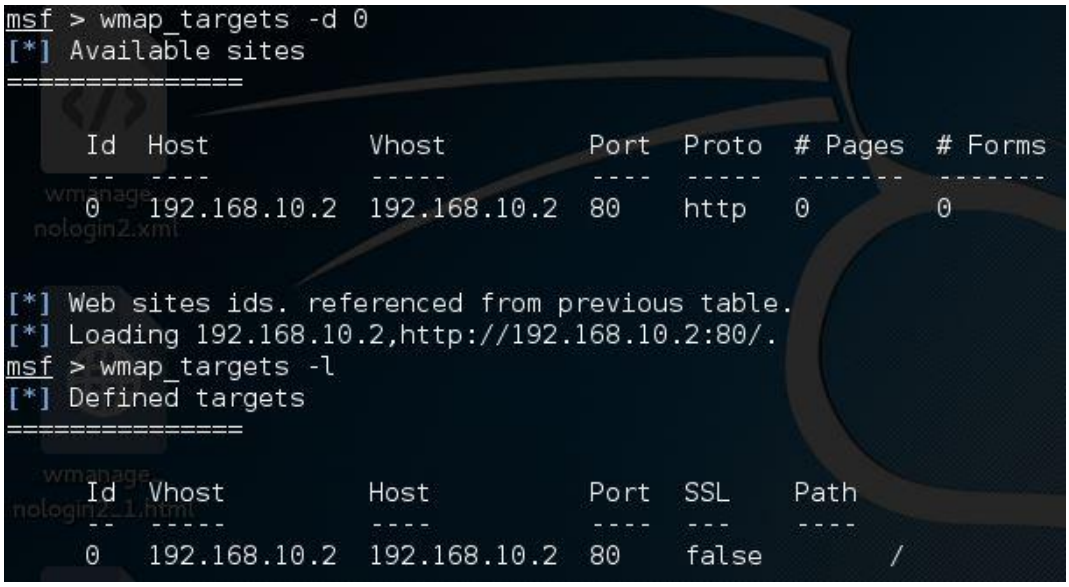
Tämän jälkeen ohjelman käytön voi aloittaa. Aloitetaan lisäämällä kohdesivusto ohjel-maan (kuva 12).

```
msf > wmap_sites -a http://whitestone.sami.dev/
[*] Site created.
```

Kuva 12. Kohdesivun lisääminen Metasploittiin.

Seuraavaksi tarkistetaan kohteen olevan oikea komennolla:

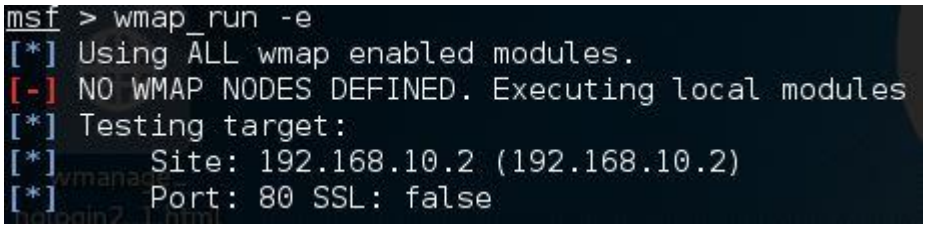
`wmap_targets -d 0` (kuva 13)



Kuva 13. Metasploitin kohteiden tarkistaminen.

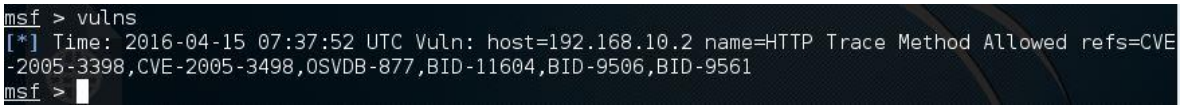
Kohteen varmistuksen jälkeen suoritetaan skannaus kaikilla ohjelman sisältämillä moduuleilla. Moduulit komennolla:

`wmap_run -e` (kuva 14)



Kuva 14. Metasploitin WMAP skannauksen suorittaminen.

Tämän jälkeen ohjelma etsii sivustosta mahdollisia haavoittuvuuksia määritetyillä moduuleilla. Kun testi on valmis Metasploit ilmoittaa löytyikö mahdollisia haavoittuvuuksia. Havaitut mahdolliset haavoittuvuudet saadaan esille ohjelmassa vuln -komennolla (kuva 15).



Kuva 15. Metasploitin löytämä mahdollinen haavoittuvuus.

Metasploit löysi ainakin yhden mahdollisen haavoittuvuuden. Haavoittuvuus todettiin kuitenkin vääräksi todennusvaiheessa. Metasploit ei siis löytänyt uusia haavoittuvuuksia.

4.3.2 SQLmap

OWASP ZAPin kanssa tehtyjen testien jälkeen mahdollisia SQL-injektioita käydään läpi manuaalisesti. Koska näitä kenttiä oli kuitenkin paljon, halusin vielä manuaalisen testauksen lisäksi koittaa OWASPIN verkkosivulta löytämääni, mahdollisimman automatisoitua SQLmap skannausta, joka näkyy kokonaisuudessaan kuvassa 16 (OWASP Foundation, 2016c.)

Kuva 16. Automatisoitu SQLmap skannaus.

Tämä automatisoitu testi käyttää kaikkia SQLmappiin asetettuja hyökkäyksiä asetettuun verkkosivuun järjestyksessä, sekä tuottaa skannauksen kulusta lokitiedoston, josta voidaan tarkemmin nähdä skannauksen kulku. Tarkemmin tähän komentoon asetetuista ehdoista voi lukea OWASPin verkkosivuilla (OWASP Foundation, 2016c). Kun ohjelma on suorittanut kaikki hyökkäykset, se listaa kaikki löydetyt mahdolliset haavoittuvuudet testaajalle. Tässä tapauksessa saimme kuitenkin vain ilmoituksen siitä, että testi ei löytänyt mitään asetetuilla ehdoilla.

5 LÖYDETYT ONGELMAT JA SUOSITELTAVAT TOIMENPITEET

Skannereiden suorittamien testien jälkeen, ohjelmien löytämät tietoturvauhat tarkistettiin, ja niiden olemassaolo todennettiin. Tässä kappaleessa ei käydä läpi jokaista skannerin raportoimaa uhkaa, koska tuloksia oli paljon, ja monet saaduista tuloksista eivät olleet todellisia uhkia. Kappaleessa Keskitytään siis lähinnä uhkiin, jotka ovat potentiaalisesti eniten haitallisia.

Ei-pysyvä XSS-haavoittuvuus löytyi sivustolla olevasta ”Ota yhteyttä” -lomakkeesta. Lomakkeeseen pystyi syöttämään Javascriptiä ja selain suoritti komennot selaimen sisällä. Haavoittuvuus tuli todennettua syöttämällä lomakkeeseen seuraava Javascript koodi:

```
</textarea><script>alert('hello');</script>.
```

Lomakkeen lähetyksessä koodi suoritetaan ja käyttäjä näkee ”hello” -tekstin hälytysikkunassa. Tätä haavoittuvuutta voi mahdollisesti käyttää levittämällä saastunutta linkkiä. Kun linkki avataan, se käyttää lomaketta suorittaakseen Javascript koodia käyttäjän selaimessa. Hyökkääjä voi luoda valesivuston, joka käyttää kohdesivustoa, mutta muokkaa sen ulkonäköä paikallisesti käyttäjän selaimessa. Tämä teoria tuli testattua ja todennettua myöhemmässä vaiheessa, luomalla oma testiskripti, jonka kautta käyttäjä saatin vietyä lievästi muunnettuun versioon sivustosta.

Haavoittuvuuden hyväksikäyttö vaatii kuitenkin paljon vaivannäköä ja tuuria hyökkääjän osalta, jotta hyökkääjä voisi hyötyä sen olemassaolosta. Haavoittuvuus nojaa pääosin siihen, että joku painaa todennäköisesti epäilyttävän oloista linkkiä. Tämän haavoittuvuuden voi korjata sanitoimalla lomakkeen tekstin lähetyksen yhteydessä. Siistimällä mahdolliset Javascriptiä suorittavat osat tekstistä, voidaan myös vaikeuttaa XSS:n käyttöä. Tämä voi olla monimutkaista, jos käyttäjän syötettä halutaan rajoittaa mahdollisimman vähän.

Sivustolta löytyi myös monta osaa, jossa kekseihin ei ollu asetettu HttpOnly-ominaisuutta. Ilman tätä ominaisuutta keksiin voi mahdollisesti päästä käsiksi Javascriptillä. Tämä mahdollistaa keksin siirtämisen ulkoiselle sivulle, jonka kautta käyttäjän session voidaan varastaa. Helppo korjaus tähän haavoittuvuuteen, on lisätä jokaiseen jaettuun keksiin HttpOnly-ominaisuus.

Sovelluksen kirjautumislomakkeessa on Autocomplete-attribuutti. Yleisesti ottaen tätä käytäntöä ei suositella. Autocomplete tallentaa käyttäjän selaimelle kirjautumistietoja, jotka mahdollinen hyökkääjä voi saada käyttöönsä, jos hänellä on pääsy käyttäjän tietokoneseen tai selaimeen. Tästä syystä esimerkiksi pankkien ei pitäisi ikinä käyttää Autocompletea verkkosivuissaan, tai sovelluksissaan. tämän attribuutin käyttö on jokaisen kehittäjän oma valinta, eikä mitenkään harvasti käytetty ominaisuus verkkosivustoissa. Autocomplete-attribuutin poistaminen kirjautumislomakkeesta korjaa tämän mahdollisen haavoittuvuuden.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Testauksessa OWASP ZAP antoi parhaita tuloksia. Vaikka kaikki löydetyt uhat eivät olleet tosia, seassa oli kuitenkin oikeita haavoittuvuuksia. Tämä tulos oli kylläkin hieman oletettavissa, sillä SQLmap ja Metasploit soveltuvat enemmän testaamaan jo todennettuun haavoittuvuuteen hyökkäämistä. Tämän opinnäytetyön pohjalta saadun kokemukseni kanssa, näen selvästi käyttämieni ohjelmien potentiaalin toimia tietoturvatestauksessa tehokkaina työkaluina. Yleisesti ottaen tässä työssä tehdyn testauksen tuloksista löydetyt haavoittuvuudet olivat erittäin pienimuotoisia. Korjauksiin ei tarvitse tehdä suuria toimenpiteitä.

6.1 Testauksen yleinen kulku ja ongelmat

Testauksen aikataulutus ja eteneminen saatiin toimimaan annetussa aikamääreissä. Itse ohjelmien kanssa suoritettu testaus ei aiheuttanut suurempaa päänvaivaa. Suurin ongelma oli joidenkin skannauksien vaatima aika. Joskus sivuston tai web-sovelluksen skannaus saattoi kestää jopa päälle kahdeksan tuntia. Kyseessä oli kuitenkin työpaikalla käytettävä tietokone, joten sen pystyi jättämään yön yli suorittamaan skannauksia. Syinä tähän saattoi olla palvelimen asetukset, jotka pystyivät hidastamaan skannereiden tuottamaa verkkoliikennettä jollain tavoin. En kuitenkaan halunnut muuttaa palvelimeen asetettuja konfiguraatioita, sillä se olisi voinut vaikuttaa lopputulosten uskottavuuteen negatiivisella tavalla. Loppujen lopuksi tulokset saatiin ajoissa, eikä muita suuria ongelmia tullut vastaan.

Yhdelle henkilölle tämän sivuston ja sovelluksen tietoturvan tarkastaminen oli suurempi työtaakka, kuin mitä alun perin ajattelin sen olevan. Tämä saattaa johtua sivuston suuruudesta ja sovelluksen monista ominaisuuksista. Manuaalisen testauksen suunnitteleminen tarkemmin olisi myös auttanut tarkempien tulosten saavuttamisessa. Suuremmassa projektissa tarkempi suunnitelma ja testauksen vaiheiden tarkempi läpikäyminen olisi suositeltavaa.

Henkilökohtaisesti uskon, että jos oma osaamiseni olisi koodauksen osalta parempi, olisin mahdollisesti voinut käyttää haavoittuvuuksia manuaalisessa testauksessa enemmän hyödyksi. Pystyin kuitenkin nykyisellä osaamisella todentamaan haavoittuvuuden ja ymmärtämään niiden periaatteellisen toiminnan. Tämä tarkoittaa minulle koodaamisen opiskelua parantaakseni suoritusta mahdollisissa tulevissa tietoturvaprojekteissa.

6.2 Loppumietteet

Tämä työ voi toimia osaksi myös suuntaa antavana ohjekirjana tulevissa testeissä. Ohjeistus antaa helpon ja nopeasti kopioitavan prosessin testiympäristön luomiselle. Luodulle raporttipohjalle uskon myös tulevan käyttöä jatkotestauksissa. Opinnäytetyön aikana sain paljon kokemusta valittujen tietoturvaohjelmien käytöstä sekä pääsin syventämään tietämystäni web-ympäristön tietoturvasta ja sitä ympäröivistä aiheista.

Työn aikana saatiin haluttuja tuloksia ja testauksen yleinen kulku saatiin dokumentoitua suunnitelman mukaisesti. Itse testaus meni mielestäni yleisesti ottaen hyvin, ja haluttuihin lopputuloksiin päästiin. Jatkotestauksesta voisi olla vielä toimeksiantajalle hyötyä, jos halutaan vielä syvempiä tuloksia. Tämän työn aikana sain omasta mielestäni luotua hyvän kokonaisuuden, joka kattaa käsiteltävän aiheen ja tehdyt toimenpiteet hyvällä tasolla.

LÄHTEET

Cordero. C. 2016. Information Security is a Marketing Responsibility. Viitattu 26.5.2016 http://www.convergnce.com/thoughts-marketing-maturity/2016/2/26/marketing-risks-information-security-infosec-is-a-marketing-responsibility

Keary. R; Manico. J; Goosen T; Krawczyk. P; Neuhaus. S & Morales. M. 2016. Authentication Cheat Sheet. Viitattu 20.4.2016 https://www.owasp.org/index.php/Authentication_Cheat_Sheet.

Manico J; Wichers D & Matalal Neil 2014. Viitattu 23.4.2016 https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet.

Microsoft 2016a. Run Virtual machines on Windows 8.1 with Client Hyper-V. Viitattu 30.4.2016 http://windows.microsoft.com/en-us/windows-8/hyper-v-run-virtual-machines.

Microsoft 2016b. Should I Create a generation 1 or 2 virtual machine in Hyper-V? Viitattu 30.4.2016 https://technet.microsoft.com/library/Dn770158.aspx#BKMK_OS.

Offensive Security 2016. What is Kali Linux? Viitattu 28.4.2016 http://docs.kali.org/introduction/what-is-kali-linux.

Osborne, M 2006. How to Cheat: How to Cheat at Managing Information Security, Syngress, Rockland, US. Viitattu 25.4.2016 Saatavilla: ProQuest ebrary.

OWASP Foundation 2013a. Top 10 2013-introduction. Viitattu 10.4.2016 https://www.owasp.org/index.php/Top_10_2013-Introduction.

OWASP Foundation 2013b. Types of Cross-site Scripting. Viitattu 24.4.2016 https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting.

OWASP Foundation 2013c.Top 10 2013-A5-Security Misconfiguration. Viitattu 20.4.2016 https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration.

OWASP Foundation 2015. Application Security Verification Standard 3.0. Viitattu 10.4.2016 https://www.owasp.org/images/6/67/OWASPApplicationSecurityVerificationStandard3.0.pdf.

OWASP Foundation 2016a. Cross-site Scripting (XSS). Viitattu 23.4.2016 https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

OWASP Foundation 2016b. OWASP Testing Guide 2016 Viitattu 29.4.2016 Saatavilla: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf.

OWASP Foundation 2016c, Automated Audit using SQLmap. Viitattu 5.6.2016 https://www.owasp.org/index.php/Automated_Audit_using_SQLMap.

Siles, R. 2016. Session Management Cheat Sheet. Viitattu 10.4.2016 https://www.owasp.org/index.php/Session_Management_Cheat_Sheet.

The CentOS Project 2016. CentOS Linux. Viitattu 28.4.2016 https://www.centos.org/about/.

Weidman, G 2014, Penetration Testing: A Hands-On Introduction to Hacking, No Starch Press, San Francisco, CA, USA. Viitattu 25.4.2016 Saatavilla: ProQuest ebrary.

Whitestone Oy 2016. Viitattu 26.4.2016 http://www.whitestone.fi/fi/etusivu.

Wichers, D; Manico J. & Seil M. 2016. SQL Injection Prevention Cheat Sheet. Viitattu 10.4.2016 https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet.

W3schools 2016. SQL Injection. Viitattu 15.4.2016 http://www.w3schools.com/sql/sql_injection.asp.