Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2019

Sami Koskinen

WEB-SOVELLUSTEN TIETOTURVASTANDARDIN TESTAAMINEN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 36

Sami Koskinen

WEB-SOVELLUSTEN TIETOTURVASTANDARDIN TESTAAMINEN

Tietoturva on nousemassa yhä tärkeämmäksi osaksi web-sovellusten kehitystä ja myyntiä. Euroopan unionin vuonna 2018 voimaanastuneen yleisen tietoturva-asetuksen myötä web-sovellusten tilaajat voivat saada konkreettisia rangaistuksia tietoturvallisuuden laiminlyömisestä. Näin ollen web-sovellusten tietoturva pitää pystyä varmistamaan ja todistamaan nykyisille sekä mahdollisille tuleville asiakkaille.

Työn tavoitteena on kehittää tehokas ja kohtuullisella työmäärällä toteutettava testaustapa jo olemassa olevan tietoturvastandardin testaamiselle. Tietoturvastandardi pitää sisällään OWASPin, eli Open Web Application Security Projectin, 10 kriittisintä haavoittuvuutta. Tavoitteena on myös luoda valmius tietoturvaraportin luomiselle.

Työn kehitysvaiheessa etsittiin ensin sopivat työkalut testauksen toteuttamiselle, minkä jälkeen tutustuttiin työkaluihin ja testausmenetelmiin. Testaus toteutettiin aluksi manuaalisesti OWASPin omaa web-sovellusten penetraatiotestaustyökalulla, Zed Attack Proxyllä. Manuaalisen testauksen ja tulosten raportoinnin jälkeen manuaalinen testaus todettiin toimivaksi, mutta liian paljon työtä vaativaksi prosessiksi. Toisessa vaiheessa testausta pyrittiin jatkokehittämään kevyemmäksi, automatisoiduksi prosessiksi. Automatisointiin käytettiin versionhallinnan pilvipalvelua, Bitbuckettia, sekä CircleCI-työkalua, joka on jatkuvan integraation työkalu.

Projektin lopputuloksena syntyi kaksi eri testausprosessia. Ensimmäinen lopputulos oli manuaalinen testausprosessi, jonka avulla voidaan olla yhä varmempia web-sovelluksen tietoturvallisuudesta. Toisena lopputuloksena syntyi automatisoitu tietoturvastandardin testausprosessi, joka onnistuttiin sulauttamaan mukaan web-sovelluksen kehitysvaiheeseen kohtuullisella työmäärällä. Manuaalinen testaus on huomattavasti automatisoitua testausta hitaampi, mutta on sitäkin tarkempi. Automatisoitutestaus soveltuu suurimpien haavoittuvuuksien testaamiseen.

Projektissa päästiin työn tilaajan asettamiin tavoitteisiin. Kehitettävää automatisoidulle testaukselle jäi, sillä ajan puutteen vuoksi web-sovelluksiin tunnistautumista ei ehditty automatisoimaan, jolloin tunnistautumisen takana olevat sivustot jäävät testaamatta. Automatisoitua testausprosessia pystyy kuitenkin hyödyntämään suurimmassa osassa web-sovelluksia.

ASIASANAT:

Tietoturva, penetraatiotestaus, tietoturvastandardi, web-sovellukset

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and communication technology

2019 | 36

Sami Koskinen

TESTING A WEB-APPLICATION SECURITY STANDARD

Information security is continuously becoming a important part of web-application development and sales. After the inception of European Union's General Data Protection Regulations in 2018 web-applications owners can face concrete punishments for neglecting security in web-applications. This means that customers are more interested in information security but without knowledge in the technology behind web-application security, it has become more important for the web-application provider to test and prove the security of their applications.

The goal of this project was to develop an efficient method for testing and proving an existing security standard while maintaining a reasonable workload. The security standard includes all of the Open Web Application Security Projects Top 10 most critical web-application vulnerabilities. A secondary goal for the project was to create a method for efficiently reporting the results of web-application security testing.

In the first part of the development phase of the project, sufficient tools were researched for penetration testing the web-applications. The penetration testing was originally conducted manually using OWASP's own web-application penetration testing tool, Zed Attack Proxy. After manually testing the security of the web-application based on the security standard, and reporting the results, the manual penetration testing was concluded to be too inefficient for being integrated into web-application development workflow. In the second part of the development phase the goal was to automate the testing that was conducted manually in part one. The automation was implemented using a version control tool, BitBucket, and a continuous integration tool, CircleCI.

The outcome of the project was two separate testing methods. The first one was a manual testing method. The manual testing method is conducted by hand and requires the attention of a penetration tester. The manual testing method is an accurate and extensive testing method for web-applications. The second method was a completely automated testing method for web-applications that could be integrated into web-application development workflow. The automated testing method is less accurate, than the manual one but requires no work from the penetration tester after the initial configuration.

The project goals were reached after developing the automated testing method. Further development is recommended for the automated penetration testing, since authentication could not be automated in the limited time that the project was being worked on. This means that parts of the web-application being tested might remain untested if no authentication is automated. The automated testing method will however test all parts of the web-application that do not require authentication.

KEYWORDS:

Information security, penetration testing, web-application, security standard

SISÄLTÖ

KUVAT

TAULUKOT

KÄYTETYT LYHENTEET

HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
OWASP	Open Web Application Security Project, Avoin web-sovellusten tietoturvaprojekti
SQL	Structured Query Language, standardoitu kyselykieli
ZAP	Zed Attack Proxy, Penetraatiotestaustyökalu

1 JOHDANTO

Opinnäytetyön aiheena on web-sovellusten tietoturvastandardin testaaminen. Opinnäytetyöprojektin toimeksiantajana toimii alustatalouksiin erikoistunut turkulainen ohjelmistoyritys. Toimeksiantajayrityksen asiakkaina toimivat yritykset, jotka pyrkivät saamaan lisäarvoa yritykselleen web-sovelluksilla tai alustoilla.

Tietoturvan tärkeys on kasvanut web-sovelluksissa asiakkaan näkökulmasta huomattavasti viimeisen vuoden aikana. Syynä tietoturvan saamaan huomioon on Euroopan Unionin vuonna 2018 voimaanastunut yleinen tietosuoja-asetus. Tietosuoja-asetuksen myötä työn tilaajat voivat saada rangaistuksia, jos laiminlyövät web-sovelluksen ja henkilötietojen tietoturvaa. Tästä syystä asiakkaat kaipaavat selkeää näyttöä web-sovelluksien tietoturvan huomioimisesta sekä riskien minimoinnista.

Projektin aloitushetkellä projektin toimeksiantajayrityksellä oli jo olemassa tietoturvastandardi. Tietoturvastandardin mukaan tilaajayritys pyrkii suojaamaan ohjelmistonsa vähintään OWASPin 10 kriittisintä haavoittuvuutta vastaan. Yritykseltä puuttui tapa, jolla voi testata tietoturvastandardin pitävyyttä ja osoittaa näin web-sovellustensa tietoturvallisuus. Lisäksi tietoturva tulisi ottaa paremmin huomioon jo web-sovellusten kehitysvaiheessa. Tietoturvan testaamiseen ei kuitenkaan haluta käyttää merkittäviä määriä aikaa, jolloin testauksen tulisi olla toteutettavissa kohtuullisella työmäärällä.

Näin ollen työn tavoitteeksi muodostui kehittää olemassa olevan tieturvastandardin testausmenetelmä, jota voidaan käyttää ohjelmistokehitysprojektien kehitysvaiheessa, sekä jo olemassa olevien web-sovellusten testaamiseen kohtuullisella työmäärällä. Testauksen ja sen raportoinnin tavoitteena on varmistaa web-sovellusten tietoturvallisuus sekä tarjota olemassa oleville sekä mahdollisille tuleville asiakkaille näyttöä web-sovellusten tietoturvallisuudesta. Tietoturvastandardi pitää sisällään kaikki OWASPin 10 kriittisintä haavoittuvuutta.

Opinnäytetyössä tutkitaan web-sovelluksen tietoturvastandardin testaamista kohtuullisella työmäärällä. Opinnäytetyön aikana ensimmäisessä vaiheessa tutkitaan tarkemmin toimeksiantajayrityksen tietoturvastandardiin sisältyviä haavoittuvuuksia sekä riskejä. Tämän jälkeen määritetään mittarit web-sovelluksen tietoturvan mittaukseen ja arvioimiseen, jonka jälkeen perehdytään testausmenetelmiin. Käytännön osuudessa tutkimus konkretisoituu OWASPin penetraatiotestaustyökalulla tehtävään web-sovelluksen

tietoturvakartoitukseen sekä penetraatiotestauksen automatisointiin. Lopussa tutkitaan tietoturvastandardin testauksen sulauttamista web-sovelluksen kehitysprosessiin.

2 TIETOTURVALLISUUS WEB-SOVELLUKSISSA

Web-sovellusten tietoturvan tärkeys on kasvanut asiakkaan näkökulmasta viime aikoina huomattavasti, sillä Euroopan unionin uuden yleisen tietosuoja-asetuksen myötä yritykset kohtaavat konkreettisia rangaistuksia tietoturvan laiminlyömisestä. Näin ollen tietoturvallisuudesta on tullut web-sovellusten myynnissä ja kehityksessä tärkeämpää. Web-sovellusten tilaajat ovat yhä enemmän kiinnostuneita kehitettävien web-sovellusten tietoturvallisuudesta.

Yleisen tietosuoja-asetuksen mukaan vastuu henkilötietojen käsittelystä on rekisterinpitäjällä, eli yrityksellä joka web-sovelluksen omistaa. Asiakastoteutuksissa tämä tarkoittaa käytännössä itse asiakasta. Asiakkaan vastuulla on myös varmistaa, että tarvittavat tekniset toimenpiteet on toteutettava, jotta voidaan osoittaa yleisen tietosuoja-asetuksen noudattamista web-sovelluksessa. Tämä tarkoittaa myös sitä, että asiakas on vastuussa web-sovelluksen tietoturvasta. Asiakas ei kuitenkaan usein osaa arvioida web-sovelluksen tietoturvallisuutta, jolloin tarvitaan helpompi ja yksinkertaisempi tapa osoittaa web-sovelluksen tietoturvallisuus. (Lex-Europa 2016.)

Yleinen tietosuoja-asetus käsittelee nimenomaan henkilöiden yksityisyydensuojaa ja OWASPin, eli Open Web Application Security Projectin, Top 10 Privacy Risks -projektin mukaan suurin uhka yksityisyyttä kohtaan ovat web-sovelluksissa nimenomaan web-sovelluksen tietoturvahaavoittuvuudet. Penetraatiotestauksella pyritään testaamaan ja ennaltaehkäisemään juuri näitä haavoittuvuuksia. (OWASP 2019.)

2.1 Tietoturvastandardi

Tietoturvastandardi on toimeksiantajayrityksen määritelmä siitä, miten tietoturvallisuus tulisi yrityksen sisällä hoitaa. Tässä tapauksessa kyseessä on nimenomaan web-sovellusten tietoturvastandardi. Tietoturvastandardin mukaan yrityksen kehittämien web-sovellusten on vastattava tietoturvallisuudeltaan tietoturvastandardin määrittelemää tasoa, eli web-sovelluksen on oltava parhaan mukaan suojattuna OWASPin määrittämältä 10:ltä kriittisimmältä haavoittuvuudelta. Tietoturvastandardi on määritetty ohjaamaan sekä toimeksiantajayrityksen sisäistä ohjelmistokehitystä, sekä todistamaan tuleville ja olemassa oleville asiakkaille web-sovellusten tietoturvallisuus.

Tietoturvastandardilla ei kuitenkaan ole merkitystä, jos sitä ei pystytä todentamaan ja osoittamaan sen noudattamista. Näin ollen tietoturvastandardin testaaminen on tärkeä osa tietoturvastandardia. Tässä tietoturvastandardissa olennaisimmassa osassa on OWASPin 10 kriittisintä haavoittuvuutta, eli tietoturvastandardia testatessa täytyy pystyä testaamaan kaikki OWASPin 10 kriittisintä haavoittuvuutta parhaan mukaan. Web-sovellusten haavoittuvuuksien testaamiseen on olemassa muutamia menetelmiä. Näitä vertailtiin pääasiassa vertailemalla muiden web-sovellusten tietoturvatestaajien kokemuksia, sekä perehtymällä molempien testausmenetelmien vahvuuksiin ja heikkouksiin.

Koodikatselmointi on yksi tapa tutkia web-sovelluksen tietoturvallisuutta. Koodikatselmoinnissa tarkastellaan koodia konkreettisesti ja pyritään löytämään mahdollisia haavoittuvuuksia sitä kautta. Koodikatselmointi on tehokas menetelmä tiettyjen koodiin liittyvien haavoittuvuuksien testaamiseen, mutta se ei ole testausmenetelmänä kovinkaan tehokas, sillä se vie hyvin paljon aikaa. Lisäksi mahdollisten lisäosien ja käytettyjen kirjastojen koodi pitäisi katselmoida myös läpi, jos haluaa tarkastaa tietoturvallisuuden myös niiden osalta. Koodikatselmointi on kuitenkin yhdessä muiden testausmenetelmien kanssa hyvinkin tehokas tapa testata ja varmistua sovelluksen tietoturvallisuudesta, silloin kuin koodikatselmointia ei tehdä koko web-sovellukselle, vaan tarkkaillaan esimerkiksi ainoastaan uusimpia muutoksia nopeaan tahtiin. Ongelmana koodikatselmoinnissa on myös se, että katselmoinnin suorittajana ei voi olla yksi henkilö, tai yhden henkilön pitäisi osata todella montaa eri ohjelmointikieltä, sekä olla tutustunut mahdollisimman moneen eri web-sovellusten toteutusteknologiaan. Todella suuren työmääränsä vuoksi koodikatselmointi ei ole varteenotettava tietoturvastandardin testaustapa tässä projektissa, sillä se ei täytä kohtuullisen työmäärän tavoitteita. (OWASP 2010a)

Toinen ja tunnetuin tietoturvastandardin testausmenetelmä on penetraatiotestaus. Penetraatiotestauksessa pyritään ensin ohjelmallisesti tutkimaan ja sen jälkeen hyökkäämään testattavaan kohteeseen. Penetraatiotestaus voi olla hyvinkin nopea prosessi, jossa testataan koneellisesti mahdollisimman monta haavoittuvuutta. Tämä tarkoittaa sitä, että penetraatiotestaus on sopiva testausmenetelmä tähän projektiin. Penetraatiotestauksen vahvuuksiin kuuluu sen tehokkuus ja mahdollinen automatisointi, pienienkin haavoittuvuuksien havainnointi sekä menetelmän sopivuus kaikille eri web-applikaation toteutusteknologioille, eli testaajan ei tarvitse osata ja ymmärtää kaikkia web-applikaation toteutukseen käytettyjä teknologioita. Heikkouksiakin penetraatiotestaukselta kuitenkin löytyy. Väärin toteutettu penetraatiotestaus saattaa antaa väärän turvallisuuden tunteen, jolloin haavoittuvuudet jäävät huomaamatta ja etenevät jopa tuotantoon asti.

Esimerkiksi huomattavasti tuotantoympäristöstä poikkeavassa ympäristöstä testaaminen saattaa antaa eri tulokset, kuin tuotantoympäristössä testaaminen. (IT Governance 2018)

Tietoturvastandardin noudattaminen ja testaus pitää kuitenkin pystyä todistamaan myös web-sovelluksen tilaajalle, kuin myös tietoturvastandardin valvojalle, eli toimeksiantajayritykselle. Näin ollen myös testauksen tuloksista on luotava raportti. Raportissa on käytävä ilmi testauksen lopputulokset ja testauksessa käytetyt menetelmät. Näin web-sovelluksen tilaaja, sekä toimeksiantajayritys näkevät konkreettisesti web-sovelluksen tietoturvan tason.

3 TIETOTURVALLISUUDEN MITTAAMINEN

Web-sovelluksen tietoturvallisuutta on vaikea mitata. Uusia haavoittuvuuksia löydetään jatkuvasti ja jokaiselta haavoittuvuudelta suojautuminen on mahdotonta. Kuitenkin yksi selkä mittari, erityisesti web-sovellusten kohdalla on tilaajayrityksen tietoturvastandardin käyttämä OWASPin 10 kriittisintä haavoittuvuutta lista. OWASP on vuonna 2001 perustettu avoin voittoa tavoittelematon yhteisö, jonka tavoitteena on auttaa yrityksiä ja yhteisöjä löytämään tietoturvariskejä sekä edesauttaa tietoturvan yleistä kehitystä. (OWASP 2019b.) OWASP ylläpitää vekkosivullansa listaa web-sovellusten 10:stä suurimmasta tietoturvariskistä ja suosittelee varmistamaan sovellusten turvallisuuden minimoimalla listalla olevat riskit (OWASP 2019c).

3.1 Injektio

Injektio on OWASPin 10 kriittisimmän haavoittuvuuden listalla sijalla yksi, sillä ne ovat todella yleisiä web-sovelluksissa ja injektiohyökkäyksen seuraukset voivat olla erittäin vakavat. Injektio on haavoittuvuus, jossa käyttäjän on mahdollista syöttää järjestelmään haitallista informaatiota. Haavoittuvuuden mahdollistaa useasti käyttäjän syöttämän tiedon virheellinen käsittely ohjelmistossa. Virheellinen käsittely tapahtuu usein käyttäjän syöttämän datan virheellisessä siivoamisessa, jossa esimerkiksi haitallista koodinpätkää ei siivota syötön yhteydessä, vaan se syötetään eteenpäin ohjelmointikielen tulkille, jolloin järjestelmä suorittaa haitallisen koodinpätkän muuttaen ohjelman suunniteltua kulkua. Injektion riski on web-sovelluksissa jokaisessa paikassa, toiminnossa ja ominaisuudessa, joka käsittelee käyttäjän syöttämää tietoa. Käyttäjä voi syöttää haitallista informaatiota esimerkiksi syöttökenttiin, kuten käyttäjänimekseen sivulle rekisteröityessä, osoiteriville parametrina tai vaikkapa keskusteluviestinä sovelluksessa. Tämä tekee injektioilta suojautumisen hyvin haasteelliseksi, sillä hyökkäyspinta-ala on injektiohyökkäyksissä todella suuri erityisesti web-sovelluksissa, joissa käsitellään käyttäjän syöttämiä tietoja paljon. (OWASP 2013.)

Injektiohyökkäyksiä on useita erilaisia ja seuraukset vaihtelevat hyökkäyksien tyyppien mukaan. Esimerkiksi SQL-Injektiot ovat yksi yleisimmistä ja vaarallisimmista injektioista web-sovelluksissa. SQL-injektiossa käyttäjä syöttää järjestelmään esimerkiksi syöttökentän kautta SQL-komentoja, jotka järjestelmä suorittaa. SQL-injektioiden avulla

hyökkääjä pääsee käsiksi osaan tai koko tietokantaan saaden näin haltuunsa sille kuulumatonta tietoa. SQL-komennoilla voi myös aiheuttaa merkittävää haittaa sovellukselle, kuten esimerkiksi poistaa kaikki järjestelmään tallennetut tiedot.

Injektioilta suojautuminen on usein todella vaikeaa ja huomioon otettavia asioita on paljon. Paras tapa varmistua injektioilta suojautumiselta on tarkistaa web-sovelluksen lähdekoodista käyttäjän syöttämien tietojen käsittely, sekä minimoida ohjelmointikielen tulkkien käyttöä. Testauksessa ei kuitenkaan ole mahdollista tarkistella itse koodia, joten koodin laatua ei voida käyttää luotettavana mittarina. Näin ollen tehokkain tapa mitata ohjelmiston suojautumista injektioilta, on testata mahdollisimman moneen syöttökenttään mahdollisimman monta eri injektiota ja tarkistettava niiden toimivuus. Mitä useampi eri injektiotapa ja -lauseke testataan, sitä varmempia voidaan olla järjestelmän turvallisuudesta injektioita vastaan. (OWASP 2016.)

3.2 Heikko tunnistautuminen

Heikko tunnistautuminen on OWASPin mukaan toiseksi kriittisin haavoittuvaisuus. Heikossa tunnistautumisessa on kyse siitä, että järjestelmän tunnistautuminen on puutteellinen tai liian helposti kierrettävissä. Heikoksi tunnistautumiseksi luokitellaan esimerkiksi tapaukset, joissa käyttäjä tai järjestelmänvalvoja on asettanut itselleen web-sovellukseen helposti murrettavan salasanan. Heikossa tunnistautumisessa voi olla kyse myös käyttäjän istunnon kaappaamisesta, tai virheellisestä istunnon luonnista.

Heikolta tunnistautumiselta suojautuakseen voi esimerkiksi pakottaa käyttäjien salasanat vaikeammin murrettaviksi, määrittää lyhyemmän istuntoajan tai käyttää nykyään suosiota kasvattavaa kahden vaiheen tunnistautumista, jossa käyttäjän on tunnistauduttava salasanan ja käyttäjänimen lisäksi esimerkiksi puhelimellaan tai sähköpostitse. Heikon tunnistautumisen haavoittuvuuksien mittaus ja testaus tapahtuu testaamalla esimerkiksi salasanoja tai istuntoavaimia arvailemalla. Jos penetraatiotestauksessa ei päästä kirjautumaan järjestelmään ilman ennalta määritettyjä tunnuksia, voidaan tunnistautumisen todeta olevan vähintään hyvällä tasolla. (OWASP 2010b; OWASP 2017.)

3.3 Arkaluontoisen tiedon paljastaminen

Arkaluontoisen tiedon paljastaminen on hyvin itseselitteinen haavoittuvuus. Haavoittuvuudessa kyse on uhasta, jossa käyttäjä saa tavalla tai toisella käsiinsä arkaluontoista tietoa, joka hänelle ei kuulu. Web-sovelluksissa arkaluontoiset tiedot eivät rajoitu ainoastaan henkilötietoihin, vaan siihen kuuluu myös esimerkiksi järjestelmässä käytetyt salasanat, teknologiat ja työkalut. Arkaluontoisten tietojen paljastuminen voi tapahtua monella eri tavalla.

Arkaluontoisen tiedon paljastamisen mittaaminen on melko yksinkertaista. Jos järjestelmää käyttäessä tai muita haavoittuvuuksia testatessa järjestelmä ei anna itsestään tai käyttäjistään mitään arkaluontoista tietoa, voidaan järjestelmää pitää turvallisena. Kuitenkin jo yksi vuodettu arkaluontoinen tieto on liikaa ja sillä saattaa olla tuhoisat seuraukset. Näin ollen arkaluontoista tietoa vuotavaa järjestelmää voidaan pitää erittäin haavoittuvaisena. Lisäksi Euroopan unionin yleisen tietosuoja-asetuksen mukaan henkilötietoja ei saa luovuttaa ilman lupaa ja käsittelyssä tulee noudattaa erityistä varovaisuutta. (OWASP 2017; Lex-Europa 2016.)

3.4 XML External Entities

XML-haavoittuvuudessa huonosti asetetut tai päivittämättömät XML prosessorit prosessoivat XML-tiedostoja tietoturvattomasti, jolloin XML-tiedoston sisällöllä voidaan vaikuttaa järjestelmään esimerkiksi injektoimalla koodia XML-tiedoston sisällä. Tämä haavoittuvuus koskee ainoastaan järjestelmiä, jossa käsitellään XML tietoja. XML-tiedostojen haavoittuvuudet ovat usein kriittisiä, jolloin yhtään XML-tiedosto haavoittuvuutta ei saa löytyä. (OWASP 2017.)

3.5 Rikkinäiset käyttöoikeudet

Rikkinäisten käyttöoikeuksien haavoittuvuuksissa kyse on siitä, että tunnistautunut tai tunnistautumaton käyttäjä pääsee web-sovelluksessa käsiksi sisältöön, johon hänellä ei ole tai pitäisi olla oikeutta. Esimerkkinä rikkinäisten käyttöoikeuksien haavoittuvuudesta toimii yksinkertaisimmillaan osoiteriville salatun osoitteen kirjoittaminen. Käyttäjä saattaa päästä sovelluksen hallintanäkymään käsiksi kirjoittamalla osoitteen osoiteriville, jos

hallintanäkymässä ei tarkisteta käyttäjän oikeuksia oikein. Rikkinäisten käyttöoikeuksien haavoittuvuudet johtuvat usein väärin määritellyistä käyttöoikeuksista tai suojauksista.

Turvallisten käyttöoikeuksien mittaaminen on melko yksinkertaista. Määritetään ennalta sivut, toiminnot ja tiedot, joille tietyn käyttäjän tai tunnistautumattoman käyttäjän ei pitäisi päästä ja testataan, pääseekö käyttäjä tai tunnistautumaton käyttäjä käsiksi niihin. Jokainen rikkinäisen käyttöoikeuden haavoittuvuus ei ole välttämättä vakava, mutta esimerkiksi ylläpitosivun haavoittuvuus aiheuttaisi merkittäviä haittoja, joten turvallisuus määritellään haavoittuvuuskohtaisesti. (OWASP 2017.)

3.6 Virheelliset tietoturva-asetukset

Virheellisissä tietoturva-asetuksissa on kyse tietoturva-asetuksista, jotka ovat joko unohtuneet oletusasetuksille ja ovat näin ollen alttiita hyökkäyksille, tai väärin asetetuista asetuksista, joissa esimerkiksi web-sovelluksen palvelin on unohtunut salasanasuojata. Virheellisten tietoturva-asetusten uhkaan kuuluu myös käyttöjärjestelmien päivitykset, sillä vanhentuneissa järjestelmissä on olemassa haavoittuvuuksia, joita hyökkääjä voi hyödyntää. (OWASP 2017.)

3.7 Cross-Site Scripting

Cross-Site Scripting on injektio, jossa hyökkääjä onnistuu syöttämään web-sovellukseen omaa koodiaan niin, että sovellus esittää sen niin kuin se olisi sen omaa koodia, jolloin loppukäyttäjän laite suorittaa koodin aavistamatta sen vaarallisuutta. Cross-Site Scripting poikkeaa aikaisemmin mainituista injektiohaavoittuvuuksista siten, että koodi suoritetaan loppukäyttäjän päässä. Koska loppukäyttäjän selain luulee, että hyökkääjän syöttämän koodi onkin luotettavan web-sovelluksen omaa koodia, ei se osaa estää koodin suorittamista. Cross-Site Scriptingillä pystyy aiheuttamaan vakavaa haittaa loppukäyttäjille, eli on syytä pitää vakavana haavoittuvuutena. Cross-Site Scripting haavoittuvuuksia ei pitäisi löytyä luotettavasta web-sovelluksesta ollenkaan. (OWASP 2017.)

3.8 Vaarallinen deserialisointi

Yksinkertaisimmillaan ohjelmistoissa serialisoinnilla tarkoitetaan datan muuttamista tietyn muotoisesta toisen muotoiseksi. Esimerkiksi käyttäjän antamat komennot tai tekstit voidaan tallentaa web-sovelluksen tiedostoihin tai muistiin bitteinä, jolloin käyttäjän syöttämät komennot serialisoidaan biteiksi. Deserialisoinnissa on kyse serialisoidun datan muuttamisesta takaisin luettavaan tai ohjelmiston suoritettavaan muotoon. Vaarallisessa deserialisoinnissa on kyse juuri deserialisoinnissa tapahtuvasta hyökkäyksestä. Haavoittuvuudessa web-sovelluksen järjestelmä serialisoi käyttäjän syöttämän haitallisen datan. Tämän jälkeen, kun järjestelmä haluaa päästä käsiksi käyttäjän syöttämään dataa, se deserialisoidaan. Deserialisointi suoritetaan web-sovelluksen järjestelmässä, jossa deserialisoidun datan käyttämisessä täytyy noudattaa varovaisuutta. Useissa ohjelmointikielissä deserialisointi on tehty valmiiksi, jolloin siihen ei itse ohjelmoija vaikuttaa, mutta usein mahdolliset haavoittuvuudet päivitetään nopeasti. Hyökkäys onnistuu, kun ohjelmiston deserialisointi käsittelee käyttäjän syöttämän haitallisen datan virheellisesti, ja suorittaa sen. Vaarallisella deserialisointi -hyökkäyksellä käyttäjä saattaa saada pääsyn koko järjestelmään ja koko järjestelmä on silloin uhattuna. Vaarallisessa deserialisoinnissa lievimmät hyökkäykset saattavat kaataa järjestelmän tai estää käyttäjiltä pääsyn sinne. Vaarallinen deserialisointi on aina kriittinen haavoittuvuus, eikä niitä saa löytyä web-sovelluksesta ollenkaan. (OWASP 2017; Acunetix 2017.)

3.9 Vaarallisten komponenttien käyttö

Haavoittuvaisten komponenttien käytössä on kyse joko vanhentuneitten tai vaarallisten komponenttien käytöstä. Komponentit saattavat olla tässä tapauksessa kirjastoja, paketteja, lisäosia tai moduuleita, joita ohjelmistossa käytetään. Haavoittuvuudelta suojaudutaan pitämällä komponentit ajan tasalla ja tekemällä taustatutkimusta aina uusien komponenttien hankkimisen yhteydessä. Haavoittuvaiset komponentit näkyvät testeissä muina haavoittuvuuksina ja niiden vakavuus vaihtelee komponenttikohtaisesti. Yksi vaarallinen komponentti ei tee välttämättä koko web-sovelluksesta hyökkäysaltista, joten vaarallisten komponenttien uhka arvioidaan haavoittuvuuskohtaisesti. (OWASP 2017.)

3.10 Puutteellinen loki ja valvonta

Puutteellisessa lokissa ja valvonnassa järjestelmän lokin kirjaaminen ja valvonta on joko puutteellinen tai puuttuu kokonaan. Web-sovelluksissa hyökkäyksien havaitseminen on todella oleellinen osa web-sovelluksen tietoturvallisuutta. Hyökkäykset saattavat kestää hyvinkin pitkiä aikoja, ja haavoittuvuudet tulisi korjata mahdollisimman nopeasti. Lisäksi on tärkeää tietää, mitä järjestelmälle on tehty, onko jotain tietoja viety ja mahdollisesti myös mistä hyökkäys on tullut. (OWASP 2017.)

4 TYÖKALUT

Projektin aloitusvaiheessa tutkittiin mahdollisia työkaluja web-sovellusten tietoturvastandardin testaamiseen. Penetraatiotestausvaiheeseen löytyi useita sopivia työkaluja, joiden avulla työ olisi toteutettavissa. Merkittävimmiksi penetraatiotestaustyökaluiksi osoittautuivat PortSwiggerin Burp Suite, Tenablen Nessus sekä OWASPin Zed Attack Proxy. Näistä kolmesta vaihtoehdosta ainoastaan Burp Suite sekä Zed Attack Proxy päätyivät testausvaiheeseen. Burp Suiten sekä Zed Attack Proxyn todettiin alussa tehdyissä kevyissä testiajoissa olevan hyvin samanlaisia. Merkittävin ero penetraatiotestaustyökalujen välillä oli hinta, sillä Zed Attack Proxy on täysin ilmainen penetraatiotestaustyökalu. Lisäksi Zed Attack Proxystä minulla oli jo aikaisempaa kokemusta. Tämän vuoksi Zed Attack Proxy valikoitui projektin penetraatiotestaustyökaluksi.

Lisäksi testauksen aikana testausympäristö, sekä testattava web-sovellus erotettiin erillisiin virtuaaliympäristöihin. Virtuaaliympäristöjen hajauttamiseen käytettiin virtuaaliympäristöjen hallintaa helpottavaa työkalua.

4.1 OWASP Zed Attack Proxy

OWASPin ZAP eli Zed Attack Proxy on OWASPin kehittämä ja ylläpitämä penetraatiotestaustyökalu. ZAP on lisäksi yksi maailman suosituimmista ilmaisista penetraatiotestaustyökaluista. ZAPissa on olemassa useita eri automatisoituja työkaluja web-sovellusten tietoturvan testaamiseen. Valitsin ZAPin käytettäväksi työkaluksi sen useiden helppokäyttöisten ominaisuuksien, sekä ilmaisen hinnan vuoksi. Lisäksi tutkimustyöni perusteella ZAPin käyttöä on helppo automatisoida ja sillä on olemassa omat työkalut komentorivikomentojen käyttämiseen testauksessa. ZAP täyttää myös OWASP 10 yleisimmän haavoittuvuuden listan haavoittuvuudet kohtuullisesti. Testaustyökaluun on myös mahdollista asentaa lisäosia, joiden avulla yhä useampi haavoittuvuus on testattavissa.

Ensimmäinen ZAPin ominaisuus, jota testauksessa käytetään, on robotti, jonka avulla kartoitetaan ensin testauksen kohdetta ja sen eri osioita. ZAPissa robotti on nimetty Spideriksi. Robotti ryömii verkkosivuja eteenpäin hyperlinkkien avulla ja kartoittaa kaikki web-sovelluksen osoitteet, osiot ja rajapintakutsut, joihin sillä on pääsy. Robotti tarvitsee toimiakseen ensin yhden URL-osoitteen, josta se aloittaa ryömimisen. Robotin ryömimisen lopputuloksena käyttäjällä on lista kaikista saatavilla olevista URL-osoitteista sekä

parametreistä, joita URL-osoitteisiin on syötetty. Listaa hyödynnetään ZAPin muissa työkaluissa. Tätä työkalua käytetään tiedusteluvaiheessa. (OWASP 2017b.) Mikäli robotti ei onnistuisi ryömimään web-sovelluksen sisältöä eteenpäin eikä kartoittamaan sivustoa, voi URL-osoitteet ja niiden parametrit syöttää manuaalisesti itse ZAPiin. ZAPissa on myös työkalu, jolla se voi koittaa arvata osoitteita massiivisen listan avulla, jossa on listattuna suosituimmat tai tunnetut eri järjestelmien URL-osoitteet. (OWASP 2015a.) Lisäksi testauksessa käytetään myös Spiderin kaltaista kahlaajarobottia, joka tarkkailee HTML-sisällön sijaan sivuston JavaScript sisältöä. Työkalun nimi on Ajax-Spider.

Seuraava olennainen ominaisuus on Aktiivinen skannaus. Aktiivinen skannaus on kaikista olennaisin työkalu haavoittuvuuksia etsittäessä. Aktiivisessa skannauksessa ZAP pyrkii löytämään automaattisesti mahdolliset haavoittuvuudet. ZAP hyödyntää jo aikaisemmin löydettyjä ja tiedettyjä haavoittuvuuksia ja testaa niitä Spider-robotin kartoittamiin URL-osoitteisiin sekä niiden parametreihin. Kaikkia haavoittuvuuksia ei voi kuitenkaan automatisoidulla aktiivisella skannauksella löytää. Skannaus löytää ainoastaan tietyn tyyppiset haavoittuvuudet, kuten esimerkiksi injektiot, cross-site scriptingit sekä arkaluontoisen tiedon paljastamisen. Aktiivinen skannaus ei löydä ns. loogisia haavoittuvuuksia, kuten esimerkiksi rikkinäisiä käyttöoikeuksia, sillä skannauksella ei ole tietoa siitä, että kenellä pitäisi olla oikeudet millekin sivulle. (OWASP 2015b.)

OWASPin itsensä mukaan ZAPin avulla voidaan tarkistaa kaikki OWASP 10 yleisimmän haavoittuvuudet (Taulukko 1).

Taulukko 1. Taulukko ZAPin työkaluista haavoittuvuuksien testaamiseen.

Haavoittuvuus	Työkalut
A1 Injektio	Active Scan SQLMAP
A2 Heikko tunnistautuminen	Spider Access Control Testing (Lisäosa)
A3 Arkaluontoisen tiedon paljastaminen	Active Scan
A4 XML External Entities	Active Scan Beta (Lisäosa)
A5 Rikkinäiset käyttöoikeudet	Active Scan
A6 Virheelliset tietoturva-asetukset	Spider Access Control Testing Ajax Spider
A7 Cross-Site Scripting	Active Scan
A8 Vaarallinen deserialisointi	jsonview (lisäosa) viewstate (lisäosa)
A9 Vaarallisten komponenttien käyttö	Passive Scan Alpha (Lisäosa)
A10 Puutteellinen loki ja valvonta	Spider Active Scan

ZAPiin on kuitenkin asennettava lisäosia, jotta kaikki OWASPin 10 kriittisintä haavoittuvuutta saadaan testattua riittävällä tasolla. ZAPiin on tässä projektissa asennettu kaikki listassa mainitut lisäosat. (OWASP 2018)

4.2 SQLMAP

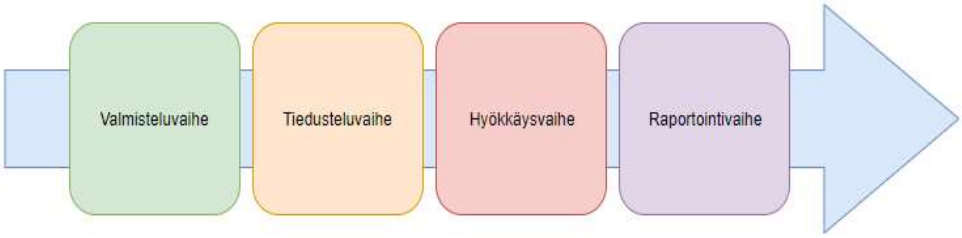
Sqlmap-työkalu on ilmainen työkalu SQL-injektioiden testaamiseen. Sqlmap on erikoistunut ainoastaan SQL-injektioiden havaitsemiseen ja on erittäin tärkeä työkalu niiden havaitsemisessa. OWASP ZAPissa SQL-injektioiden testaus on hyvin rajoittunut ja se testaa vain yleisen tason injektioita. Sqlmap testaa tietokantakohtaisesti erilaisia haavoittuvuuksia ja saattaa näin löytää sivustolla SQL-injektiolle haavoittuvaisia kenttiä paremmin kuin ZAP. Projektissa sqlmap-työkalua käytetään mahdollisten SQL-injektiohaavoittuvuuksien löytämiseen.

4.3 Vagrant

Vagrant on helppokäyttöinen virtuaaliympäristöjen hallintatyökalu. Vagrantin avulla käyttäjä saa helposti hallittua ja käytettyä useaa eri virtuaaliympäristöä yhdellä tietokoneella. Projektissa Vagrantia käytetään virtuaaliympäristöjen eristämiseen ja testaustyökalujen pyörittämiseen. Virtuaaliympäristöjen erottaminen toisistaan on testauksen kannalta tärkeää, sillä web-sovelluksen ympäristön on oltava mahdollisimman samankaltainen, kuin itse tuotantoympäristö. Vagrantin avulla web-sovellus pyörii Ubuntun 16.04 -version ympäristössä, johon on asennettu ainoastaan tarpeelliset paketit. Penetraatiotestaustyökalu pyörii myös omassa ympäristössään, jossa käyttöjärjestelmänä toimii Kali Linux, joka on Linux-pohjainen käyttöjärjestelmä, johon on asennettu ennalta tarvittavat penetraatiotestaustyökalut.

5 PENETRAATIOTESTAUS

Penetraatiotestaukseen kuuluu neljä eri vaihetta (Kuva 1). Aluksi tutustutaan penetraatiotestattavaan kohteeseen, eli tässä tapauksessa web-sovellukseen. Tämän jälkeen tiedustellaan web-sovellusta ja tutustutaan hyökkäyspinta-alaan, jonka jälkeen suoritetaan penetraatiotestauksella hyökkäykset. Lopuksi raportoidaan penetraatiotestauksen tulokset.



Kuva 1. Penetraatiotestauksen eri vaiheet.

5.1 Valmisteluvaihe

Testauksen valmistelu on olennainen osa testausprosessia. Testauksen valmisteluvaiheessa hankitaan testaukseen tarvittavat luvat, sekä määritetään yhdessä testauksen tilaajan kanssa tavoitteet. Tavoitteiden määrittelyn jälkeen suunnitellaan testausmenetelmät ja päätetään käytettävät työkalut. Penetraatiotestaus ilman web-sovelluksen omistajan lupaa on laitonta.

Testauksen tavoitteet määriteltiin yhdessä työn tilaajayrityksen kanssa. Testauksen tavoitteeksi muodostui tarve minimoida web-sovelluksen tietoturvahaavoittuvuudet ja vahvistaa järjestelmän turvallisuus OWASPin 10:tä kriittisintä haavoittuvuutta vastaan kohtuullisella työmäärällä. Lisäksi tavoitteena on varmistaa eri käyttäjätyyppien mahdollisuudet häiritä web-sovelluksen suunniteltua toimintaa. Testauksessa pyritään mahdollisimman kattavaan testaukseen, eli testejä ei tehdä täysin sokeasti vaan ennakkotietona testaukseen saadaan järjestelmän osoitteet, normaalin käyttäjän tunnukset sekä pääsy lähdekoodiin ja versionhallintaan mahdollisten vanhentuneiden pakettien haavoittuvuuksien tarkistamiseksi.

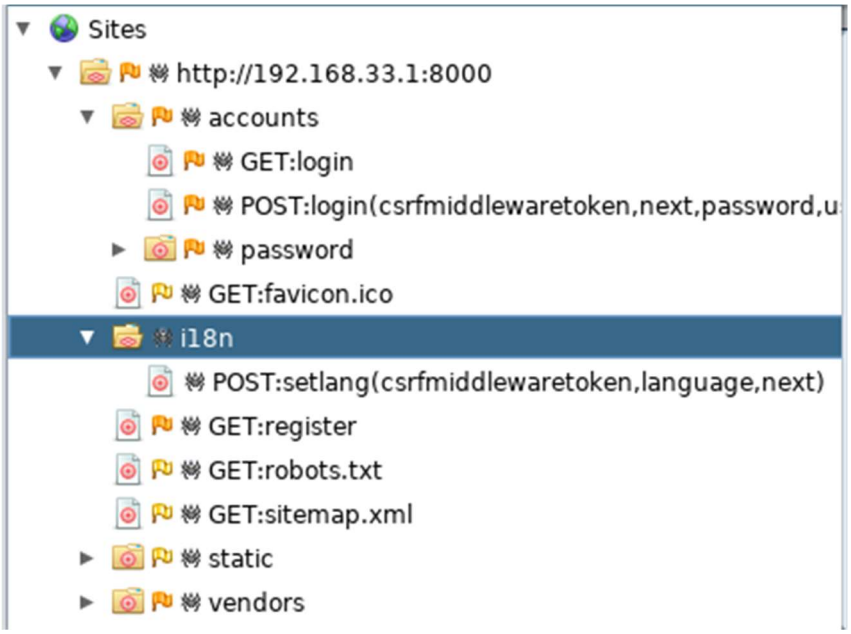
Koska haavoittuvuustestauksessa testataan aivan oikeita haavoittuvuuksia, on ympäristön syytä olla eristetty sekä irrallinen tuotantoympäristöstä. Tässä testauksessa sekä kohde web-sovellus sekä penetraatiotestaustyökalut pyörivät samalla tietokoneella erillisissä virtuaaliympäristöissä. Virtuaaliympäristöt on yhdistetty toisiinsa sisäisen osoitteen kautta ja testaustyökalulta on estetty pääsy ulkoverkkoon. Näin varmistetaan se, että esimerkiksi Spider-robotti ei ryömi sivustolta ulos esimerkiksi hyperlinkkien tai muiden osoitteiden avulla. Testaus on rajoitettu tarkasti ainoastaan testattavaan web-sovellukseen.

5.2 Tiedusteluvaihe

Penetraatiotestauksen ensimmäinen vaihe on tiedusteluvaihe. Tiedusteluvaiheen aikana on tavoitteena kerätä web-sovelluksesta mahdollisimman paljon tietoa. OWASP ZAP-työkalulla tiedustelu hoidetaan Spider-robotin avulla täysin automatisoidusti. Lisäksi tiedusteluvaiheeseen kuuluu tässä projektissa myös web-sovelluksessa käytettävien kirjastojen ja lisäosien tarkastelu ja tautatutkimustyö, sillä meillä on pääsy ohjelmiston lähdekoodiin ja yksi OWASPin 10 yleisimmästä haavoittuvuudesta koskee nimenomaan vanhentuneita paketteja ja kirjastoja.

Tiedustelu aloitetaan tarkastelemalla pakettien ja kirjastojen versioita. Näin saadaan jo etukäteen tietoon mahdolliset haavoittuvuudet ja testauksen kohteet, joihin on kiinnitettävä enemmän huomiota. Kirjastojen ja pakettien haavoittuvuuksien tarkastelu suoritetaan Googlaamalla pakettien nimet ja versiot ja etsimällä niistä raportoituja haavoittuvuuksia. Lisäksi haavoittuvuuskirjastosivut käydään läpi vanhentuneiden pakettien osalta. Haavoittuvuuskirjastosivuina käytetään https://www.exploit-db.com/ sekä https://www.cvedetails.com/. Projektissa on käytössä useita eri paketteja, joten tässä ei käydä läpi jokaiselle paketille löytyneitä haavoittuvuuksia. Tärkeimpänä löytönä on koko projektin pohjana käytettävän python-kirjaston Djangon tietoturvahaavoittuvuudet. Projektissa on käytössä Djangon version 1.8.18, joka on julkaistu vuonna 2015. Haavoittuvuuksien etsimisen jälkeen korkeimmalle nousee Djangon itsensä julkaisema päivitys ja tiedote, jossa ilmoitetaan nimenomaan Djangon 1.9 vanhemmissa versioissa olevan vakava Cross-Site Scripting -haavoittuvuus. Cross-Site Scriptingeihin on siis kiinnitettävä enemmän huomiota. Muita vakavia haavoittuvuuksia web-sovelluksessa käytettävistä kirjastoista ja paketeista ei löytynyt.

Pakettien tarkistelun jälkeen siirrytään OWASP ZAP -työkalun käyttöön. Ensimmäiseksi käynnistetään OWASP ZAPista Spider-robotti ja annetaan sille aloitusosoite, josta ryömiä sivustoa eteenpäin. Kun robotti on lopettanut sivun ryömimisen, niin se listaa kaikki löydetyt sivut ja niiden mukana nousseet varoitukset (Kuva 2).



Kuva 2. Spiderin listaamat sivut.

Jo pelkästään Spiderin avulla on löydetty pieniä tietoturvahuomautuksia, jotka eivät kuitenkaan ole varsinaisia haavoittuvuuksia, vaan pikemminkin kehotuksia tehdä pieniä toimenpiteitä ennalta ehkäistäkseen tietoturvaongelmia. Spiderin tuloksista nähdään myös, että sivustolla on sisäänkirjautuminen osoitteessa http://192.168.33.1:8000/login/, jossa lähetetään palvelimelle käyttäjätunnus sekä salasana POST-pyyntönä. Tästä voimme päätellä, että koko web-sovelluksen testaamiseen tarvitaan toimivat käyttäjätunnukset. Spiderille voi antaa olemassa olevan session käyttöönsä, jolloin se pääsee käsiksi kirjautuneiden käyttäjien sisältöön. Kohdesovelluksen sessiohallinta tallentaa sessioavaimen käyttäjän muistiin, jolloin se tunnistaa käyttäjän sessioavaimen perusteella, eikä käyttäjän tarvitse kirjautua jokaisella sivulla uudestaan. Session voi antaa ZAPille käyttöön avaamalla asettamalla selaimen välityspalvelimeksi ZAPin oman osoitteen eli localhost:8080. Tämän jälkeen ZAP seuraa itse HTTP sessioita ja tallentaa niiden sessioavaimet. Seuraavaksi kirjaudutaan kohdesivustolle, jolloin ZAP kaappaa session avaimen. Tämän jälkeen asetetaan ZAPin HTTP Sessions -työkalusta sessio aktiiviseksi ja näin Spider osaa käyttää olemassa olevaa sessiota ryömimiseen. Tämän jälkeen ajetaan

Spider-työkalu uudestaan, jolloin saadaan kartoitettua myös tunnistautumisen takana olevat sivut.

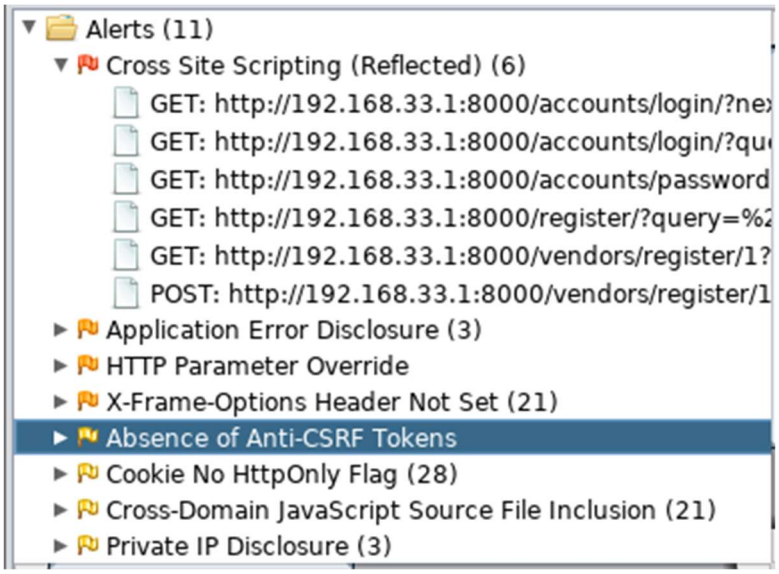
5.3 Hyökkäysvaihe

Hyökkäysvaiheessa pyritään vaikuttamaan web-sovelluksen järjestelmään jollakin tapaa kokeilemalla mahdollisimman monta hyökkäystä. Hyökkäysvaiheen päätyökaluna toimii ZAPin Active Scan -työkalu. Annetaan Active Scanille Spiderin tavoin aloituspisteeksi web-sovelluksen etusivu. Tämän jälkeen Active Scan testaa jokaiseen Spiderin kartoittamaan sivuun mahdollisimman montaa hyökkäystä. Koska Active Scan kokeilee todella useaa eri hyökkäystä, sen ajaminen kestää myös huomattavasti Spideria kauemmin. Active Scanin testaamia haavoittuvuuksia on laajennettu projektissa asentamalla projektiin Active Scanin beta ja alpha säännöt. Näiden avulla saadaan testattua yhä useampi haavoittuvuus, mutta riskinä on myös väärät varoitukset, sillä sääntöjä ei ole vielä virallisesti hyväksytty tuotantokäyttöön. Tämän jälkeen ajetaan web-sovellusta vasten vielä ZAPin Ajax-Spider-työkalu, jotta myös mahdolliset JavaScript-ongelmat ja haavoittuvuudet saadaan testattua.

Hyökkäysvaiheessa todetaan jo, että ZAP on nostanut varoituksen mahdollisesta HTTP-parametrin ylikirjoittamisesta, jota se ei pysty kuitenkaan varmentamaan. Injektioiden testaamiseen erikoistunut työkalu SQLMAP pystyy tarkastelemaan injektioiden mahdollisuutta tarkemmin. Parametrin ylikirjoitus -varoitus on tullut osoitteesta http://192.168.33.1:8000/vendors/register/1, jossa käyttäjä lähettää lomakkeen web-sovelluksen backendiin. SQLMAP ajetaan komentoriviltä sqlmap-komennolla, ja annetaan SQLMAPille komento, joka kokeilee eri SQL-injektioita osoitteen parametriin: sqlmap -u http://192.168.33.1:8000/vendors/register/1* .

5.4 Raportointivaihe

Lopputuloksena Active Scan löysi web-sovelluksesta useita Cross-Site Scripting -haavoittuvuuksia. Lisäksi työkalulla löydettiin web-sovelluksesta useita pieniä huomioita, jotka eivät vaadi välitöntä huomiota. Kaikkiaan huomioita löytyi yksi suuren riskin haavoittuvuus, kolme keskitason huomiota, sekä viisi matalan tasan huomiota (Kuva 3). Web-sovellus ei siis ole suojattu OWASPin 10 yleisimmältä haavoittuvuudelta, sillä vähintään yksi vakavahaavoittuvuus on löydetty.



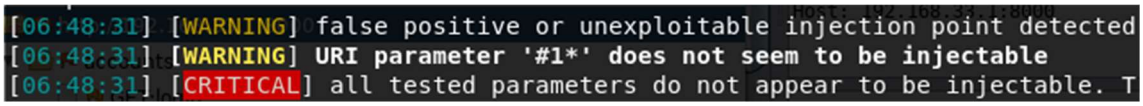
Kuva 3. Spiderin nostamat varoitukset.

Työkalun löytämä vakava haavoittuvuus on tyypiltään ei-pysyvä Cross Site Scripting. Haavoittuvuudessa käyttäjä voi syöttää sivustolle JavaScriptiä, jonka sivusto esittää sivunlatauksen jälkeen omana koodinaan. Haavoittuvuus on vakava, sillä sen avulla voidaan aiheuttaa vakavaa ja pysyvää haittaa web-sovelluksen loppukäyttäjille. Itse järjestelmään ei penetraatiotestauksen perusteella pääse käsiksi, eikä sen toimintaan voi vaikuttaa. Sama vakava haavoittuvuus on löytynyt useasta eri paikasta sivustolta. Haavoittuvuudessa ZAP-työkalu on lisännyt web-sovelluksen osoitteeseen pätkän JavaScriptiä, jonka sivusto suorittaa. Esimerkiksi kirjautumissivustolle on syötetty URL-enkoodattua JavaScriptiä näin: http://192.168.33.1:8000/accounts/login/?query=%22%3E%3Cscript%3alert%281%29%3B%3C%2Fscript%3E. Web-sovellus kääntää koodin muotoon '"><script>alert(1)</script>'", joka suoritetaan loppukäyttäjän selaimessa. Kyseinen koodinpätkä on todiste haavoittuvuuden toimivuudesta, eikä aiheuta käyttäjälle minkäänlaista haittaa.

Lisäksi sivustolta on löytynyt kolme keskiluokan haavoittuvuutta. Ensimmäinen on web-sovelluksen virheiden esittäminen käyttäjälle. Haavoittuvuus kuuluu OWASPin 10 kriittisimmän haavoittuvuuden listassa arkaluontoisten tiedon paljastamiseen. Web-sovellus antaa itsestään käyttäjälle kuulumattomia tietoja virheiden käsittelyn yhteydessä. Tässä tapauksessa virhesivu antaa käyttäjälle tietoja projektin kansionrakenteesta, asennetuista paketeista sekä järjestelmänvalvojien sähköpostiosoitteet. Vaikka ZAP luokittelee tämä haavoittuvuuden keskivakavaksi, niin manuaalisen tutkimisen jälkeen voidaan

todeta tämän olevan vakava haavoittuvuus ja näin ollen web-sovellus ei läpäise arkaluontoisten tietojen paljastamisen testiä.

Toinen keskiluokan varoitus on tullut HTTP-parametrin ylikirjoitusmahdollisuudesta. Tämä havaittiin jo hyökkäysvaiheessa ja varoitusta tutkittiin lisää SQLMAP-työkalun avulla. SQLMAP työkalu löysi saman haavoittuvuuden kuin ZAP, eli SQLMAPin mukaan parametri on haavoittuvainen Cross-Site Scriptingille, mutta SQL-injektiotestauksessa parametri aiheutti väärän hälytyksen, eli se ei ollut haavoittuvainen SQL-injektiolle, vaikka tunnustelun mukaan se olisi saattanut olla mahdollista (Kuva 4).



Kuva 4. SQLMAP-testauksen tulokset.

Kolmas keskitason varoitus on X-Frame-Options header asettamatta. X-Frame-Options headeria käytetään suojaamaan käyttäjiä klikkauksen kaappaus -hyökkäyksiltä. Klikkauksen kaappaus hyökkäys on hyökkäys, jossa käyttäjä huijataan klikkaamaan sivustolla jotain, jonka käyttäjä kuvittelee olevan luotettava tai tärkeä. Klikkaus kuitenkin kaapataan, jolloin klikkaus tekeekin jotain muuta, kuin odotettu. X-Frame-Options headerilla voidaan estää web-sovelluksen käyttö muilla sivustoilla, jolloin kyseistä sivustoa ei voi käyttää klikkausten kaappaamiseen muilla sivustoilla. (OWASP 2017c.) Matalan tason varoitukset eivät ole tässä tapauksessa varsinaisia haavoittuvuuksia, vaan ennemminkin hyviä käytäntöjä, joita tulisi kehityksessä noudattaa.

5.5 Raportointi

Penetraatiotestauksen raportointi on oleellinen osa itse työtä. Työn tilaajalle on toimitettava tieto mahdollisista löydöksistä, tai niiden puutteesta. Tässä tapauksessa työn tilaaja on tekninen henkilö ja testauksen tavoitteena oli varmistaa OWASPin 10 kriittisimmän haavoittuvuuden turvallisuus web-sovelluksessa. OWASP ZAPissa on itsessään raportointityökalut. ZAPin raportissa kerrotaan jokaisen varoituksen kohdalla tarkka kuvaus mistä varoitus johtuu, kohde johon varoitus kohdistuu, sekä todisteet varoituksen mainitsemasta haavoittuvuudesta. Näiden lisäksi raportissa kerrotaan myös miten mahdolliset haavoittuvuudet tulisi korjata. Raportin ja aikaisempien havaintojen perusteella voidaan

penetraatiotestauksesta koostaa yhteenveto läpäistyistä haavoittuvuustesteistä (Taulukko 2).

Taulukko 2. Taulukko testauksen lopputuloksista.

Haavoittuvuus	Arvosana
A1 Injektio	Hyväksytty
A2 Heikko tunnistautuminen	Hyväksytty
A3 Arkaluontoisen tiedon paljastaminen	Hylätty
A4 XML External Entities	Hyväksytty
A5 Rikkinäiset käyttöoikeudet	Hyväksytty
A6 Security Misconfiguration	Hyväksytty
A7 Cross-Site Scripting	Hylätty
A8 Insecure Deserialization	Hyväksytty
A9 Using components with Known Vulnerabilities	Hylätty
A10 Insufficient Logging and Monitoring	Hyväksytty

Tämän testauksen raportin lopputuloksena on hylätty arvosana, sillä projekti ei suojannut kaikkia OWASPin 10 yleisintä haavoittuvuutta penetraatiotestauksessa.

6 TESTAUKSEN AUTOMATISOINTI KEHITYSPROSESSIIN

Penetraatiotestauksen suorittaminen manuaalisesti osoittautui raskaaksi prosessiksi, jonka suorittaminen jokaisessa projektissa erikseen ei ole kannattavaa. Näin ollen OWASPin 10 yleisimmän haavoittuvuuden testaus tulisi automatisoida prosessin nopeuttamiseksi. OWASPin oman kehittämän raportin avulla saadaan selville suurin osa haavoittuvuuksista, joten sitä pidettiin riittävänä lopputuloksena tarkistelulle. Toimeksiantajayrityksen projektit käyttävät jo versionhallintaa ja CircleCI-työkalua testien ajamiseen, joten penetraatiotestauksen lisääminen testausprosessiin on oletettavasti nopea prosessi.

Koska manuaalinen skannaus oli pitkä prosessi, oli manuaaliselle skannaukselle keksittävä automatisoitu korvike. ZAPin omassa dokumentaatiossa kerrotaan ZAPin docker imagen sopivan parhaiten jatkuvaan integraatiojärjestelmään. Docker on virtualisointiohjelmisto, joka suorittaa virtualisoinnin käyttöjärjestelmän tasolla. Docker ajaa ohjelmistopaketteja tai imageja omissa säiliöissään. Säiliöt voivat kuitenkin keskustella keskenään. Säiliöinti on kevyempää, kuin perinteinen virtuaaliympäristöjen pyörittäminen. Säiliöiden sisältö luodaan imagejen avulla, jotka pitävät sisällään säiliöiden tarkat sisällöt.

Ensimmäinen vaihe automatisoidun prosessin kehittämisessä oli testata ZAPin docker-imagea paikallisesti ja testata sen käyttö sekä toimivuus. Dockerin asennuksen jälkeen on ensin ladattava ja asennettava OWASP ZAPin docker-image. ZAPin docker imagen asennus onnistuu komennolla “docker pull owasp/zap2docker-stable”. Tämän jälkeen käynnistetään docker-säiliö komennolla ”docker run -u zap -p 8080:8080 –name owasp-zap -d owasp/zap2docker-stable zap-x.sh -daemon -addoninstallall”. Komennolla käynnistetään ZAP docker-säiliössä portissa 8080, annetaan säiliölle nimeksi owasp-zap ja ”-addoninstallall”-vivulla asennetaan kaikki ZAPiin saatavat lisäosat, sillä kuten manuaalisessa testauksessa jo todettiin, lisäosat tarvitaan ZAPin 10 yleisimmän haavoittuvuuden testaamiseen. Tämän jälkeen käytetään testaamiseen ZAPin omaa rajapintaa. Rajapinta toimii siten, että ZAPin rajapintaan tehdään pyyntöjä, joiden perusteella ZAP suorittaa erilaisia toimenpiteitä. Manuaalisen skannauksen perusteella ensimmäinen toimenpide on Spider-robotin ajaminen, sen jälkeen Active Scan ja viimeiseksi raportin luonti. Linuxissa kutsuminen tapahtuu curl-komennolla, joka hakee verkkosivun sisällön ja tulostaa sen konsoliin. Spideria kutsutaan osoitteesta

”http://zap:8080/JSON/Spider/action/scan/?zapapiformat=JSON&formMethod=GET&url=http://192.168.33.1:8000&maxChildren=&recuse=true&contextName=&subtreeOnly=true”. Komennolla käynnistetään ZAPin Spider-työkalu ja annetaan sille kohde url-parametrissä. Kohteen lisäksi url-parametreissä annetaan myös maxChildren-parametrissä Spiderin ryömintäsyvyys, eli kuinka syvälle Spider ryömii ryömimisen aikana. Tyhjä arvo tarkoittaa ääretöntä. Recurse-parametrissä määritetään se, jatkaako Spider alkuperäisen kohdesivun alasivuille. Tämän arvoksi on annettu tosi, sillä haluamme käydä läpi koko web-sovelluksen sisällön. Lisäksi Spiderille voi antaa kontekstin, mutta sitä emme tarvitse, sillä ainoa kohde mitä tarkastellaan on alkuperäinen web-sovellus. subtreeOnly-parametrillä määritetään se, että halutaanko Spiderin pysyvän sivustolla vaikka se löytäisi tien ryömiä sivustolta ulos. Tämä määritetään todeksi, sillä kohteemme rajoittuu ainoastaan web-sovelluksen sisältöön. Spider-robotin onnistuneen käynnistyksen jälkeen rajapinta palauttaa vastauksessaan skannauksen numeron.

Tämän jälkeen odotetaan Spiderin valmistumista jonkin aikaa. Esimerkiksi minuutti on hyvä aika odottaa näin pintapuolisessa tarkistuksessa. Pienin odotuksen jälkeen käynnistetään Active Scan. Active Scanin käynnistäminen toimii Spiderin käynnistystä vastaavalla tavalla kutsumalla ZAPin rajapintaosoitetta. Active Scanin käynnistämiseksi käytetään seuraavaa URL-osoitetta: http://zap:8080/JSON/ascan/action/scan/?zapapiformat=JSON&formMethod=GET&url=http://192.168.33.1:8000&maxChildren=&recurse=true&context-Name=&subtreeOnly=true. Kutsussa annetaan samat parametrit, kuin Spideria käynnistäessä. Active Scanin käynnistyksen onnistuessa rajapinta palauttaa skannauksen numeron. Active Scan kestää huomattavasti Spideria kauemmin, joten sen käynnistämisen jälkeen on odotettava pidempään.

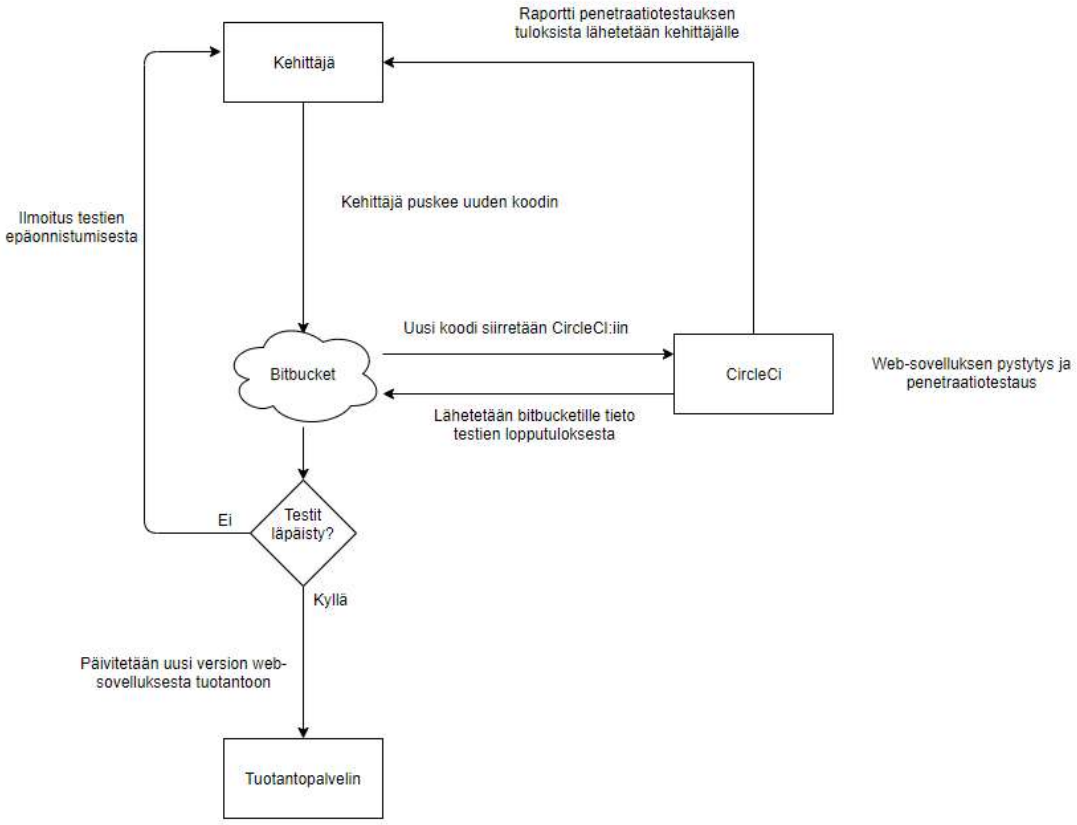
Active Scanin valmistumisen jälkeen generoidaan skannauksesta HTML-raportti ja tallennetaan se tiedostoon. Raportti generoidaan kutsumalla ZAPin rajapinnan osoitetta http://zap:8080/OTHER/core/other/htmlreport/?formMethod=GET. Raportin tallentaminen tapahtuu curl komennolla. Koko HTML raportin haku ja tallennus komennoksi muodostuu ”curl -K http://zap:8080/OTHER/core/other/htmlreport/?formMethod=GET -o raportti.html”. Raportti on rakenteeltaan samanlainen, kuin manuaalisessa testauksessa generoitu raportti. Raportin mukaan automaattinen skannaus on löytänyt samat varoitukset, kuin manuaalinen skannaus, eli automatisoitua skannausta voidaan pitää pätevänä OWASPin 10 kriittisimmän haavoittuvuuden testaamiseen.

6.1 Versionhallinta

Web-sovellus projektien kehitystyössä käytetään aina versionhallintaa. Versionhallinta helpottaa ohjelmiston hallintaa sekä kehityksen seurantaa. Versionhallinta on järjestelmä, joka ylläpitää ohjelmiston tiedostoihin tehtyjä muutoksia ja versioita ja tallentaa ne esimerkiksi ulkoiseen tiedostoon. Yksinkertaisuudessaan versionhallinta voi toimia niin, että aina kun tiedostoon tehdään muutoksia, se tallennetaan eri nimellä esimerkiksi tiedosto-versio1, tiedosto-versio2, tiedosto-versio3 jne. Tähän on kuitenkin luotu automatisoituja ratkaisuja, joista toimeksiantajayrityksen käytössä on Git. Git on hajautettu versionhallintajärjestelmä, joka tarkoittaa sitä, että aina kun tiedostot haetaan Gitin avulla, niin se ei hae ainoastaan uusinta versiota vaan käytännössä noutaa aina täydellisen varmuuskopion kaikesta datasta. Koska useat koodarit saattavat työstää samaa projektia samaan aikaan, on tarvittu ratkaisu projektin tiedostojen ja versionhallinnan jakamiseen. Tässä on käytetty BitBucketia.

Bitbucket on verkkopohjainen versionhallintavarasto, jossa projektin tiedostot ja versionhallinta ovat tallessa. BitBucket toimii siten, että aina kun käyttäjä päivittää versionhallinnan alaista dataa ja lähettää sen Gitin avulla eteenpäin, niin uudet tiedostot ja versiot päivittyvät BitBucketiin. BitBucketin suurin hyöty on se, että kaikki tiedostot ja versiot ovat tallessa verkossa, eikä niitä tarvitse siirtää projektin työntekijöiden kesken manuaalisesti esimerkiksi muistitikulla. (Bitbucket.org) Bitbucketissa jokaisella projektilla on oma pakettivarasto, jossa koodia säilytetään.

Lisäksi toimeksiantajalla on automatisoitu ohjelmiston päivittäminen tuotantopalvelimelle sekä testien ajo. Nämä automatisoinnit ovat tehty CircleCI-työkalun avulla. CircleCI on jatkuvan integraation työkalu. CircleCI:n avulla aina kun jonkin ohjelmiston Bitbucketissa olevaan pakettivarastoon päivitetään sisältöä, niin ajetaan myös ohjelmistoon kuuluvat testit sekä testien onnistuessa päivitetään myös tuotantopalvelimelle uusin versio ohjelmistosta. Testien epäonnistuessa CircleCI ilmoittaa projektitiimille testien epäonnistuneen, eikä päivitä tuotantopalvelinta. Penetraatiotestauksen automatisoinnin kannalta kiinnostavin on CircleCI:n testausvaihe. Testausvaiheessa CircleCI pystyttää pilvipalvelimelle virtuaaliympäristön, jossa se ensin asentaa web-sovelluksen tiedostot, sitten pystyttää web-sovelluksen paikalliseen ympäristöön ja ajaa sen jälkeen testit pystytettyä ympäristöä vasten prosessikaavion mukaisesti (Kuva 5).



Kuva 5. Prosessikaavio automatisoidusta testausympäristöstä.

Automatisoitu penetraatiotestaus on siis ajettava testien ajon yhteydessä samassa virtuaaliympäristössä. CircleCI ajaa virtuaaliympäristössä sille sen asetustiedostossa annetut komennot. Näin ollen paikallisessa ympäristössä testattu automatisoitu penetraatiotestaus voidaan suorittaa komento kerrallaan CircleCI:n testausympäristössä samalla tavalla, kuin paikallisessa ympäristössä. Komentojen suorittamiseksi ne täytyy vain lisätä CircleCI:n asetustiedostoon YAML-muotoisena ja muuttaa kohteeksi paikallisesti pyörivän web-sovelluksen osoitteeksi.

CircleCI:n asetustiedoston päivittämisen jälkeen web-sovelluksen uusi versio päivitetään BitBucketissa olevaan pakettivarastoon, jolloin CircleCI aloittaa uuden version testaamisen automaattisesti. CircleCI:n testauksen etenemistä voidaan seurata verkossa CircleCI:n verkkopalvelussa. CircleCI:n suoritettua komennot on myös HTML raportti generoitu CircleCI:n virtuaaliympäristöön. Tämän jälkeen HTML raportti lähetetään CircleCI:n

virtuaaliympäristöstä toimeksiantajayrityksen palvelimelle, jolloin raportti jää jokaisesta testauksesta talteen, eikä häviä CircleCI:n virtuaalipalvelimen mukana.

6.2 Lopputulema

Automatisoitu testaus toimii lähes yhtä tehokkaasti, kuin manuaalinen testaus, jonka on todettu kattavan koko OWASP 10 kriittisintä haavoittuvuutta. Automatisoidulla testauksella ei kuitenkaan saada testattua kaikkia niitä haavoittuvuuksia ja ongelmia, jotka manuaalisen testauksen yhteydessä olisi voitu huomata. Esimerkiksi sivut, joille ei kirjautumattomalla käyttäjällä pitäisi olla pääsyä, voivat näkyä suojausten ulkopuolelle eikä automatisoitu testaus tiedä, kuuluisiko sivulle päästä. Automaattinen testaus kuitenkin kattaa OWASPin 10 kriittisintä haavoittuvuutta pääpiirteiltään kohtuullisella työmäärällä. Automatisoitu skannaus on huomattavasti nopeampi prosessi sekä vaatii itse työntekijältä huomattavasti vähemmän vaivaa, koska se on automatisoitu kokonaisuudessaan.

Suurin puute automatisoidussa skannauksessa on kuitenkin käyttäjän autentikoinnin puute. Automaattiseen testaukseen pystyy automatisoimaan sisäänkirjautumisen ainoastaan joissain tapauksissa. Manuaalisessa testauksessa käytetty kirjautumistapa vaatii olemassa olevan session, jota ei CircleCI:n ympäristössä ole. Session voi kuitenkin luoda, esimerkiksi automatisoimalla selaimen sisään kirjautumisen esimerkiksi Selenium-työkalulla. Selenium on työkalu, joka on tarkoitettu nimenomaan selainten automatisointiin. Tällöin Seleniumilla luotu kirjautunut käyttäjäsessio voitaisiin antaa ZAPille, jolloin kirjautuminen onnistuisi. Haasteena tässä on kuitenkin se, että kirjautuminen on aina web-sovelluskohtaista, jolloin Seleniumilla tehty kirjautuminen pitäisi tehdä jokaiseen projektiin erikseen.

Automatisoitu testaus kattaa kuitenkin toimeksiantajayrityksen tietoturvastandardin tason, sillä se testaa OWASPin 10 kriittisintä haavoittuvuutta. Tietoturvastandardi on näin vahvistettu. Myös manuaalinen testaus kattaa tietoturvastandardin asettaman tason, mutta se ei täytä toimeksiantajayrityksen asettamaa kohtuullisen työmäärän tavoitettu, jolloin sitä ei voi sulauttaa web-sovellusten kehitysprosessiin. Lisäksi niin manuaalisesta, kuin automaattisestakin testauksesta lopputuloksena on penetraatiotestaustyökalun itse generoima raportti, joka kattaa kokonaisuudessaan kaikki tietoturvastandardin raportoinnissa asetetut tavoitteet. Raportissa kuvataan testausmenetelmät, löydetyt haavoittuvuudet sekä niille mahdolliset korjausmenetelmät.

7 YHTEENVETO

Opinnäytetyössä tutkittiin web-sovelluksien testaamista tietoturvastandardin tasolla sekä testauksen sulauttamista kehitysprosessiin. Projektin tavoitteena oli kehittää tilaajayritykselle toimiva testausprosessi, joka testaa kaikki OWASPin 10 kriittisintä haavoittuvuutta kohtuullisella työmäärällä.

Aluksi tutkittiin tietoturvastandardin testausmenetelmiä, joista lopulliseksi testausmenetelmäksi valikoitui penetraatiotestaus sen tehokkuuden ja joustavuuden vuoksi. Toinen mahdollinen vaihtoehto testaukseen oli koodikatselmointi. Lisäksi asetettiin selkeät tavoitteet tietoturvastandardin testauksen lopputuloksille. Yksittäinen haavoittuvuus ei välttämättä tarkoita sovelluksen olevan tietoturvaton tietoturvastandardin tasolla.

Tietoturvastandardin testaus toteutettiin OWASPin omalla testaustyökalulla, Zed Attack Proxyllä. Manuaalisessa testauksessa tietoturvan testauksen tavoitteisiin päästiin, sillä kaikki OWASPin 10 kriittisintä haavoittuvuutta testattiin riittävällä tasolla, mutta kohtuullisen työmäärän tavoite ei ollut manuaalisella työllä saavutettavissa. Manuaalinen testaus kesti useita tunteja, jolloin useiden projektien testaukseen menisi liikaa aikaa. Yhden projektin tarkempaan testaukseen manuaalinen testaus kuitenkin soveltui, mutta kehitysprosessiin sulauttaminen veisi web-sovelluksen kehitystyöstä useita tunteja jokaisessa kehitysprojektissa. Näin ollen testaus oli automatisoitava. Testauksen automatisointiin käytettiin CircleCI-työkalua, joka oli käytössä tilaajayrityksellä jo ennen projektin alkua. Automatisoinnissa testaus toteutettiin Zed Attack Proxyn rajapinnan kautta automatisoiduilla komennoilla, jolloin itse testaajalle työksi jäi ainoastaan komentojen kertalaatuinen syöttäminen CircleCI:n asetustiedostoon. Näin testaajan työmäärä pieneni huomattavasti ja saavutti työn tilaajan asettaman kohtuullisen työmäärän tavoitteen.

Automatisoitu testaus ei kuitenkaan ollut yhtä kattava, kuin manuaalinen testaus. Näin ollen ajan puutteen vuoksi automatisoidusta testauksesta jouduttiin jättämään pois websovellukseen kirjautuminen, jolloin tunnistautumisen takana olevat osiot jäävät testaamatta kokonaan. Jatkokehityksenä projektille tunnistautumisen toteutus selaimen automatisoinnilla, esimerkiksi Selenium-työkalulla, sekä selaimen session käyttämisessä Zed Attack Proxyssä olisi järjestelmään tunnistautuminenkin automatisoitavissa.

Työn lopputuloksena kehitetyn testausprosessin avulla tilaajayritys pystyy testaamaan sisäisesti kehittämiensä web-sovellusten tietoturvallisuutta sekä todistamaan

konkreettisesti testauksen lopputuloksena syntyvän raportin avulla olemassa oleville sekä tuleville asiakkaille kehittämiensä web-sovellusten tietoturvallisuuden oman tietoturvastandardinsa tasolla.

LÄHTEET

Acutenix 2017. What is insecure deserialization? Viitattu 20.4.2019 https://www.acunetix.com/blog/articles/what-is-insecure-deserialization/.

EUR-Lex, Access to European Union law; Euroopan parlamentin ja neuvoston asetus (EU) 2016/679. Viitattu 20.4.2019 https://eur-lex.europa.eu/legal-content/FI/TXT/PDF/?uri=CELEX:02016R0679-20160504&from=FI.

IT Governance 2018. Viitattu 20.5.2019. https://www.itgovernance.eu/blog/en/pros-and-cons-of-penetration-testing.

OWASP 2010a. Code Review Introduction. Viitattu 20.5.2019. https://www.owasp.org/index.php/Code_Review_Introduction.

OWASP 2010b. Broken Authentication and Session Management. Viitattu 16.4.2019 https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management.

OWASP 2013. Top 10 2007-Injection Flaws. Viitattu 15.4.2019 https://www.owasp.org/index.php/Top_10_2007-Injection_Flaws.

OWASP 2015a. Forced Browse. Viitattu 18.4.2019 https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsBruteforceConcepts.

OWASP 2015b. Active Scan. Viitattu 18.4.2019 https://github.com/zaproxy/zap-core-help/wiki/HelpStartConceptsAscan.

OWASP 2016. SQL-injection. Viitattu 15.4.2019 https://www.owasp.org/index.php/SQL_Injection.

OWASP 2017a. OWASP Top 10-2017. Viitattu 16.4.2019 https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

OWASP 2017b. Spider. Viitattu 18.4.2019 https://github.com/zaproxy/zap-core-help/wiki/HelpStartConceptsSpider.

OWASP 2017c. Clickjacking. Viitattu 24.4.2019 https://www.owasp.org/index.php/Clickjacking.

OWASP 2018. ZAPping the OWASP top 10. Viitattu 22.4.2019 https://www.owasp.org/index.php/ZAPpingTheTop10.

OWASP 2019a. Top 10 Privacy Risks Project. Viitattu 10.5.2019 https://www.owasp.org/index.php/OWASP_Top_10_Privacy_Risks_Project.

OWASP 2019b. About The Open Web Application Security Project. Viitattu 12.4.2019 https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project.

OWASP 2019c. OWASP Top Ten Project. Viitattu 12.4.2019 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Main .