Tommi Prusti

Injektiot verkkosovellusten tietoturvauhkana

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Tommi Prusti

Yhteystiedot: tommi.t.j.prusti@student.jyu.fi

Ohjaaja: Tuomo Rossi

Työn nimi: Injektiot verkkosovellusten tietoturvauhkana

Title in English: Injections as security threat for web applications

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 20+0

Tiivistelmä: Digitalisaation seurauksena entistä useammat yritykset ovat alkaneet käyttää erilaisia verkkosovelluksia osana omaa liiketoimintaansa. Nykyään näissä sovelluksissa käsitellään myös paljon arkaluonteisia tietoja, joten ne ovat erityisen arvokkaita kohteita tietoturvahyökkäyksille. Tässä tutkielmassa käsitellään yleisesti verkkosovelluksia, niihin kohdistuvia tietoturvauhkia ja kehittäjien keinoja tietoturvan parantamiseen. Tietoturvauhista keskitytään koodi-injektioihin, erityisesti SQL- ja skripti-injektioihin. Kehittäjien keinoista verkkosovellusten suojaamiseksi käsitellään ohjelmistoanalyysejä sekä penetraatiotestausta.

Avainsanat: kandidaatintutkielma, kirjallisuuskartoitus, verkkosovellusten tietoturva, SQL-injektio, Skripti-injektio, ohjelmistoanalyysi, penetraatiotestaus

Abstract: As a result of digitalization, more companies have started using various web applications as part of their business. Nowadays, these applications also deal with a lot of sensitive data so they are particularly valuable targets for cybersecurity attacks. In this bachelor's thesis, I deal with web applications, their cybersecurity threats, and ways to protect them. From cybersecurity threats I focus on code injections, specifically focusing on SQL and script injections. I deal with program analysis and penetration testing as a way to protect web applications.

Keywords: Bachelor's Thesis, book review, web application security, SQL-injection, Cross-site scripting, program analysis, penetration testing

Kuviot

Sisällys

1 Johdanto

Digitalisaation myötä ihmisten työpaikoilla ja vapaa-ajalla päivittäin käyttämät ohjelmistot ovat siirtyneet entistä enemmän verkkopohjaisiin toteutuksiin, eli ne tarvitsevat verkkoyhteyttä toimiakseen. Verkon asiakas-palvelin arkkitehtuuri tarjoaa tehokkaan alustan, jossa verkkoselain toimii käyttöliittymänä sovelluksille (Jazayeri 2007). Tämä parantaa esimerkiksi sovellusten laiteriippumattomuutta, mutta kuitenkin altistaa ne myös erilaisille tietoturvahyökkäyksille.

Nykyään on käytössä paljon verkon välityksellä toimivia arkaluonteista tietoa käsitteleviä verkkosovelluksia. Ne ovat erityisen arvokkaita kohteita tietoturvahyökkäyksille, koska niissä voidaan käsitellä esimerkiksi ihmisten henkilötietoja tai yritysten yksityisiä tietoja. Jos verkkosovellus käyttää tällaista tietoa sisältävää tietokantaa toiminnoissaan, voi se mahdollistaa tietojen joutumisen ulkopuolisten haltuun, mikäli sovelluksen tietoturvasta ei ole huolehdittu oikein. Tämä voi johtaa vakaviin taloudellisiin tappioihin, sekä eettisiin ja oikeudellisiin seurauksiin (Li ja Xue 2011). Tästä syystä verkkosovellusten suojaaminen on erityisen tärkeää ja yleensä yritykset ovat valmiita investoimaan tietoturvaan. Tietoturvahyökkäyksessä käytetään hyväksi tietojärjestelmässä olevaa haavoittuvuutta. Haavoittuvuus määritellään järjestelmän ohjelmointivirheenä, takaporttina, heikkoutena tai vikana, jota luvaton käyttäjä voi hyödyntää suorittaakseen jonkin hyökkäyksen tai päästäkseen käsiksi järjestelmään tallennettuihin tietoihin (Gupta, Govil ja Singh 2014).

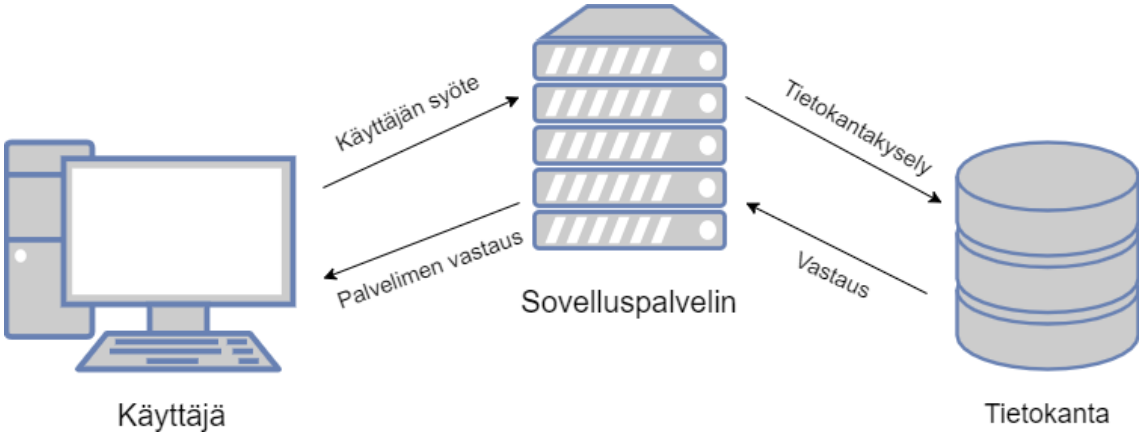
Tässä kirjallisuuskartoituksessa tarkastellaan verkkosovelluksia ja niiden tietoturvaa. The Open Web Application Security Project (OWASP) on julkaissut vuosina 2013 ja 2017 listauksen kymmenestä yleisimmästä verkkosovellusten tietoturvauhasta. Molemmissa listoissa injektiot ovat olleet sijalla yksi, joten ne ovat olleet jo pitkään kaikista suurin tietoturvauhka verkkosovelluksille (Wichers ja Williams 2017). Tutkielman luvussa kaksi käsitellään yleisesti verkkosovellusta, sekä sen rakennetta ja toimintaa. Luvussa kolme keskitytään niihin kohdistuviin SQL- ja skripti-injektio tietoturvahyökkäyksiin, sekä niiden syihin ja seurauksiin. Luvussa neljä käsitellään verkkosovellusten suojaamista erilaisten ohjelmistoanalyysien ja penetraatiotestauksen avulla.

2 Verkkosovellus

Verkkosovellus (engl. *Web application*) on sovellus, jota käyttäjä käyttää verkkoselaimella internetin välityksellä. Jazayeri (2007) kuvaa verkkosovellusten tyypillisesti koostuvan käyttäjälle näytettävistä HTML-dokumenteista, asiakaspuolen skripteistä, jotka ajetaan käyttäjän selaimessa ja ne voivat vuorovaikuttaa käyttäjän kanssa, sekä palvelinpuolen skripteistä, jotka tyypillisesti hoitavat esimerkiksi vuorovaikutuksen sovelluksen taustalla olevien tietokantojen kanssa. Tässä luvussa käsitellään verkkosovellusten toiminnan perustaa, yleisellä tasolla niiden tyypillistä rakennetta, sekä mihin niiden kehitys ja suosio perustuu.

Verkkosovellusten toiminta internetin välityksellä perustuu World Wide Webin eli WWW:n toimintaan. Joka puolestaan perustuu asiakas-palvelin arkkitehtuuriin, jossa palvelimet ja asiakkaat kommunikoivat HTTP-protokollan avulla keskenään. Palvelintietokoneelle voi olla tallennettuna esimerkiksi HTML-dokumentti, jonka asiakkaan selain voi URL-osoitteen ja protokollan GET-metodin avulla hakea nähtäväksi. Nykyään palvelimilla on usein kuitenkin muutakin kuin pelkkiä staattisia HTML-dokumentteja. Palvelin voi esimerkiksi hakea tietoa tietokannasta, muotoilla sen perusteella HTML-dokumentin ja lähettää tämän ohjelmallisesti luodun verkkosivun asiakkaan selaimeen (ks. kuvio 1). (Jazayeri 2007)

Verkkosovellus rakentuu tavallisesti jonkinlaisesta käyttöliittymästä eli käyttäjälle näkyvistä HTML-dokumenteista, sekä asiakas- ja palvelinpuolen skripteistä. HTML-dokumentit määrittävät miltä sovelluksen käyttöliittymä näyttää käyttäjän selaimessa. Asiakaspuolen skriptit ovat HTML-dokumentteihin upotettua koodia, yleensä JavaScriptiä ja ne ajetaan käyttäjän selaimessa, kun sivu latautuu. Asiakaspuolen ohjelmointiin on olemassa olio-suuntautunut ohjelmointikielen ECMAScript standardi (ECMA-262), joka perustuu esimerkiksi JavaScriptiin (Jazayeri 2007). Niiden avulla sovellus voi reagoida esimerkiksi käyttäjän hiiren klikkauksiin. Palvelinpuolen skriptit puolestaan hoitavat sovelluksen taustalla tapahtuvia asioita, kuten esimerkiksi tietokantakyselyitä ja tiedostojen käsittelyä (Jazayeri 2007). Palvelinpuolen ohjelmointi voidaan toteuttaa monilla eri ohjelmointikielillä, joita ovat esimerkiksi Java, PHP tai Python.



Kuvio 1. Verkkosovelluksen tyypillinen järjestelmäarkkitehtuuri, mukailtu lähteestä (Su ja Wassermann 2006)

Jazayeri (2007) toteaa World Wide Webin käytettävyyden parantumisen vuoden 1995 jälkeen tehneen siitä suositun alustan suurelle määrälle entistä kehittyneempiä ja innovatiivisempia verkkosovelluksia. Siten siitä kehittyi vuosikymmenessä tehokas alusta sovellusten kehittämiseen ja käyttöönottoon. Verkkoteknologian kehittyminen on ollut todella nopeaa, mikä on johtanut verkkosovellusten määrän räjähdysmäiseen kasvuun. Nykyään lähes kaikki käyttävät päivittäin erilaisia verkkosovelluksia, joten niiden kehitys tulee varmasti jatkumaan myös lähitulevaisuudessa. Yritysten mielenkiinto verkkosovelluksiin on kasvanut paljon, kun mobiililaitteiden kehitys on mennyt eteenpäin. Nykyään on paljon yrityksiä joiden liiketoiminta nojautuu pitkälti verkko- ja mobiilisovelluksiin. Tällaisesta yrityksestä hyvä esimerkki on Uber, joka tarjoaa henkilökuljetuspalvelua.

3 Koodi-injektiot

Wichersin ja Williamsin (2017) listauksen mukaan verkkosovellusten suurin tietoturvauhka on injektiot. Koodi-injektio (engl. *Code injection*) on tietoturvahyökkäys, jossa hyökkääjä pyrkii jonkin haavoittuvuuden, esimerkiksi tarkistamattoman syötteen kautta syöttämään eli injektoimaan haitallista koodia verkkosovellukseen. Usein käyttäjät antavat syötteitä verkkosovellukselle, jonkin toiminnon suorittamiseksi. Kehittäjät voivat unohtaa palvelinpuolen koodissa tarkastaa yhdenkin käyttäjän antaman syötteen, minkä jälkeen sovellus voi olla haavoittuvainen hyökkäykselle. Yleensä injektoidun koodin on tarkoitus ohittaa tai muuttaa ohjelman alkuperäisesti suunniteltua toimintaa (Lin ja Chen 2007). Monissa verkkosovelluksissa on vielä nykyäänkin tämäntyyppisiä haavoittuvuuksia, vaikka tietoturvariskit ovat yleensä kehittäjien tiedossa, niin yksikin inhimillinen unohdus saattaa avata tien hyökkääjälle.

Kappaleen aliluvuissa käsitellään kahden tyyppisiä injektioita: SQL- ja skripti-injektioita. Nämä ovat kaksi hyvin yleistä injektiohyökkäystä ja niitä tarkastellaan myös esimerkkien kautta. Viimeisessä aliluvussa käydään yleisesti läpi mihin hyökkääjä voi injektiohyökkäyksellä pyrkiä, sekä mahdollisia seurauksia.

3.1 SQL-injektio

SQL-injektio on tietoturvahyökkäys verkkosovelluksen tietokantaan. Siinä hyökkääjä syöttää Structured Query Language (SQL) koodia sovelluksessa olevaan syötekenttään (Kindy ja Pathan 2011). SQL-injektio on hyvin yleinen hyökkäystyyli, joka johtuu tarkistamattomista syötteistä. Yleisyys johtuu siitä, että verkkosovelluksissa on herkästi puutteita juuri syötteiden tarkistuksissa. Tällainen haavoittuvuus mahdollistaa SQL-injektion avulla pääsyn sovelluksen tietokantaan tallennettuun dataan.

Käydään seuraavaksi SQL-injektiosta esimerkkinä AlKhurafin ja AlAhmadin (2015) esittämä hyökkäys. Verkkosovelluksen kirjautumissivulla käyttäjää pyydetään syöttämään käyttäjätunnus ja salasana tunnistautuakseen. Syötettyjen tietojen perusteella sovellus tekee esimerkiksi tietokantakyselyn (ks. kuvio 2).

```
SELECT *
FROM Käyttäjät
WHERE käyttäjänimi = 'pääkäyttäjä' AND salasana = 'pääkäyttäjä123'
```

Kuvio 2. Sovelluksen muodostama tietokantakysely (AlKhurafi ja AlAhmad 2015)

Tautologiaan perustuvalla SQL-injektiolla hyökkääjä voi ohittaa tunnistautumisen syöttämällä ehdon kyselyn WHERE-osaan, joka on aina tosi. Hyökkääjä voi siis manipuloida tietokantakyselyä syöttämällä käyttäjätunnus kenttään ’ OR 1 = 1 – ja salasanaksi voi syöttää mitä vain. Tämän syötteen jälkeen sovellus tekeekin erilaisen tietokantakyselyn (ks. kuvio 3).

```
SELECT *
FROM Käyttäjät
WHERE käyttäjänimi = ' ' OR  1 = 1 -- ' AND salasana = 'jotain'
```

Kuvio 3. Sovelluksen muodostama injektoitu tietokantakysely (AlKhurafi ja AlAhmad 2015)

Yksittäinen lainausmerkki (’) ennen OR-operaattoria lopettaa merkkijonon ja ehdon jälkeistä kahta viivaa (–) käytetään tekemään kyselyn loppu osasta kommentti, eli tietokantapalvelin ei suorita sitä kyselyn osana. Täten tietokantakyselyn syntaksi on oikeellinen ja tietokantapalvelin suorittaa tautologian osana kyselyä ja suorittaa tunnistautumisen sovellukseen ilman että edes tarkastetaan syötettyä salasanaa. Näin hyökkääjää saa tunnistauduttua sovellukseen, vaikka hänellä ei olisi tunnuksia.

3.2 Skripti-injektio

Skripti-injektio on tietoturvahyökkäys, josta käytetään usein verkkosovellusten yhteydessä termiä Cross-site scripting (XSS). Tässä hyökkäyksessä hyökkääjä onnistuu haavoittuvuuden kautta lähettämään haitallisia skriptejä käyttäjien selaimiin ja sitä tapahtuu kun verkkosovellus viittaa HTML-sivulla käyttäjän antamiin syötteisiin ilman niiden oikeellisuuden tarkistamista (Shar ja Tan 2012). Kun hyökkääjä saa onnistuneesti upotettua oman skriptinsä verkkosovelluksen HTML-sivulle, niin selaimet suorittavat haitallisen skriptin normaalis-

ti osana sovellusta. Tällaisella hyökkäyksellä pyritään yleensä varastamaan käyttäjän selaimesta sessiokohtaisia tietoja eli evästeitä. Esimerkiksi haitallinen skripti voi lähettää suoraan hyökkääjälle evästeen, jolla verkkosovelluksen palvelin tunnistaa kirjautuneen käyttäjän. Tämän jälkeen hyökkääjä pystyy esiintymään palvelimelle uhrin nimissä. Hyökkäys on onnistunut, kun käyttäjä vierailee sovelluksen verkkosivulla, jossa haitallinen skripti on upotettuna (Shar ja Tan 2012).

Johari ja Sharma (2012) jakavat skripti-injektiohyökkäykset kahteen päätyyppiin: tallennettuihin ja heijastuviin hyökkäyksiin. Tyyppien erona on käytetäänkö hyökkäyksessä palvelin- vai asiakaspuolen haavoittuvuutta. Tallennetuissa hyökkäyksissä haitalliset skriptit on tallennettuna esimerkiksi verkkosovelluksen tietokantaan ja ne ajetaan aina kun sivustolla käydään. Heijastuvissa hyökkäyksissä puolestaan yritetään saada kohdekäyttäjä käyttämään haitallisen skriptin sisältävää URL-osoitetta siirtyessään sovellukseen. Käyttäjä siirtyy normaalisti sivustolle, mutta haitallinen skripti ajetaan samalla ja siten hyökkääjä pääsee käsiksi esimerkiksi uhrin selaimen evästetietoihin. (Johari ja Sharma 2012)

Käyttäjän sessiokohtaisten tietojen vuotaminen hyökkääjälle onnistuu, koska haitalliset skriptit ajetaan suoraan asiakaspuolella eli käyttäjän selaimessa. Koska selain on yhteydessä luotettavana pidettyyn palvelimeen, myös haitalliset skriptit pystyvät lukemaan, muuttamaan ja lähettämään kaikkia tietoja, joihin selaimella on pääsy (Johari ja Sharma 2012). Tästä syystä skripti-injektiolla on mahdollista kaapata käyttäjätilejä, varastaa tietoja, varastaa ja muokata evästeitä, manipuloida sivuston sisältöä, sekä edesauttaa palvelunestohyökkäystä (Shar ja Tan 2012). Esimerkiksi hyökkääjä voi kaapata käyttäjätilin evästeen avulla, uudelleen ohjata uhrin toiselle verkkosivulle, jossa häneen voidaan kohdistaa myös toisenlaisia hyökkäyksiä, kuten tietojenkalastelua (Johari ja Sharma 2012). Hyökkääjä voi kaapatulla käyttäjätilillä päästä esimerkiksi lukemaan uhrin sovellukseen antamia tietoja, sekä tekemään mahdollisia toimintoja hänen nimissään.

Su ja Wassermann (2006) esittelee seuraavan esimerkin skripti-injektiosta: huutokauppasivusto, joka on haavoittuvainen skripti-injektioille. Sivusto esittää listan tuotenumeroita, jotka sisältää URL-osoitteen, jonka kautta käyttäjä voi tehdä tarjouksen. Hyökkääjä syöttääkin tuotteena skriptin (ks. kuvio 4).

```
><script>document.location= 'http://www.xss.com/cgi-
bin/cookie.cgi?'%20+document.cookie</script
```

Kuvio 4. Esimerkki skripti-injektiosta, ei ole validia HTML:ää (Su ja Wassermann 2006)

Kun käyttäjä painaa hyökkääjän tuotteen linkkiä, selain jäsentää URL-osoitteen tekstin ja tulkitsee sen JavaScriptinä. Esimerkki skripti lähettää uhrin evästetiedot hyökkääjän omalle sivustolle (ks. kuvio 4).

3.3 Seuraukset

SQL- ja skripti-injektioiden seuraukset voivat olla moninaisia. Tietenkin hyökkäyksen mahdollisiin seurauksiin vaikuttaa verkkosovellus johon hyökätään. Esimerkiksi jos sovelluksen taustalla olevasta tietokannasta löytyy selkokielisesti tallennettuna kaikkien sovelluksen käyttäjien tunnukset ja salasanat, on SQL-injektion seuraukset paljon vakavammat kuin jos ne olisi salattu jollakin algoritmilla. Myös se mitä hyökkääjä haluaa varastamillaan tiedoilla tehdä vaikuttaa seurauksiin. Hyökkääjä saattaa kiusatakseen julkaista kaikki tunnukset vaikka tor-verkkoon, sen sijaan että pitäisi tiedot itsellään ja yrittäisi kiristää niillä yritystä, joka oli hyökkäyksen uhri. Yleensä onnistuneella hyökkäyksellä on aina jonkinlaisia seurauksia. Verkkosovelluksen joutuminen hyökkäyksen kohteeksi aiheuttaa vähintään haittaa siitä vastaavan yrityksen maineelle ja siksi mikään yritys ei halua joutua hyökkäyksen uhriksi. Tästä syystä yritykset haluavat panostaa omien verkkosovellustensa suojaamiseen. Verkkosovellusten suojaamista käsitellään seuraavassa luvussa tarkemmin.

SQL-injektiolla on vaikutusta verkkosovelluksen datan luottamuksellisuuteen ja eheyteen, sekä ongelmiin käyttäjien tunnistamisessa ja käyttöoikeuksissa (Johari ja Sharma 2012). Datan luottamuksellisuuden ja eheyden ongelmat johtuvat siitä, että nämä injektiot kohdistuvat tietokantoihin. Niihin voi olla tallennettuna luottamuksellista tietoa, kuten yksityisiä henkilötietoja ja siksi SQL-injektion avulla ulkopuolinen henkilö voi päästä tietoihin käsiksi, sekä mahdollisesti muokata tai poistaa niitä, jolloin tiedon luottamuksellisuus sekä eheys kärsii. Pahimmillaan hyökkääjä voi tuhota koko tietokannan (Li ja Xue 2011).

Käyttäjien käyttöoikeuksien ongelmat johtuvat, siitä että hyökkääjä voi SQL-injektiolla myös

muuttaa sovelluksen käyttäjien oikeuksia ja näin saada itselle korotettuja käyttöoikeuksia, mikäli käyttöoikeustiedot ovat myös tallennettuina haavoittuvaan tietokantaan (Johari ja Sharma 2012). Korotetuilla käyttöoikeuksillaan hyökkääjä voi päästä näkemään tietoja, joita ulkopuolisten ei kuuluisi nähdä. Käyttäjien tunnistamisen ongelma on käyty esimerkin avulla läpi aliluvussa 3.1. SQL-injektiot ovat hyvin vaikeita havaita, koska usein hyökkääjän verkkosovelluksessa tekemä luvaton toiminta tehdään oikeudet omistavilla käyttäjätunnuksilla (Johari ja Sharma 2012).

Skripti-injektioiden vaikutukset ovat vaihtelevia. Johari ja Sharma (2012) toteavat, että onnistuneen hyökkäyksen seuraukset vaihtelevat pelkästä ärsyttävyydestä käyttäjätunnusten menettämiseen, ja ne riippuvat pitkälti hyökkääjän saamista tiedoista. Usein skripti-injektioita kuitenkin käytetään osana suurempia hyökkäyksiä, joissa pyritään uudelleenohjamaan luotettavan verkkosovelluksen käyttäjiä hyökkääjien omille haitallisille sivustoille, joilla pyritään esimerkiksi levittämään haittaohjelmia tai tekemään petoksia (Johari ja Sharma 2012).

4 Verkkosovellusten suojaaminen

Gupta, Govil ja Singh (2014) toteavat että monista vanhemmista, jo käyttöön otetuista verkkosovelluksista löytyy vakavia tietoturvaongelmia ja ne ovat herkästi haavoittuvaisia hyökkäyksille, koska ne on ohjelmoitu puutteellisella tietämyksellä mahdollisista turvallisuusuhista tai kehittäjillä on ollut puutteelliset keinot niiden estämiseen. Koska jo käytössä olevissa sovelluksissa on paljon haavoittuvuuksia, niin uusien sovellusten koko kehitysprosessin aikana tietoturvasta on tärkeää huolehtia alusta loppuun. Suurin syy verkkosovellusten haavoittuvuuksiin on heikkoudet sovellusten lähdekoodeissa (Gupta, Govil ja Singh 2014). Koodi-injektiot usein johtuvat tarkastamattomista syötteistä, joten verkkosovellusten kehityksessä on erityisen tärkeää kiinnittää huomioita koodin kohtiin, joissa sovellus käsittelee käyttäjän syöttämää dataa. Heikkouksien löytäminen sovelluksen kehitysvaiheessa säästää rahaa ja auttaa välttämään ohjelmistohäiriöt (Gupta, Govil ja Singh 2014).

Verkkosovellusten suojaamiseen on useita eri tapoja. Kehittäjä voi esimerkiksi varmistaa syötteiden tarkistukset ja välttää tallentamasta sovelluksen toiminnan kannalta tarpeetonta dataa itse sovellukseen tai sen tietokantaan (Atashzar ym. 2011). Kappaleen aliluvuissa käsitellään kahta mahdollista tapaa verkkosovellusten suojaamiseen: ohjelmistoanalyysiä, sekä penetraatiotestausta. Niillä verkkosovellus pyritään suojaamaan varmistamalla sen oikeellinen toiminta kaikissa tilanteissa.

4.1 Ohjelmistoanalyysi

Ohjelmistoanalyyseillä (engl. Program analysis) etsitään verkkosovelluksen mahdollisia haavoittuvuuksia analysoimalla esimerkiksi sovelluksen lähdekoodeja. Ohjelmistoanalyysitekniikoiden avulla pyritään tunnistamaan mahdollisesti epäluotettavan tiedon kulku sovelluksessa (Li ja Xue 2011). Tällaista tietoa on yleensä esimerkiksi käyttäjien antamat syötteet. Lähtökohtaisesti verkkosovelluksen tulisi aina käsitellä epäluotettavana informaationa kaikki tieto, jota käyttäjät sovellukselle antavat syötteiksi, etenkin jos syöte sisältää tekstiä. Syötetty teksti saatetaan sovelluksen koodeissa käsitellä merkkijonona niin että syötekenttää väärin käyttämällä pystytään toteuttamaan SQL-injektio. Asiaa käsiteltiin aiemmin kolmannessa

luvussa.

4.1.1 Erilaiset analyysit

Li ja Xue (2011) esittävät kolme erilaista ohjelmistoanalyysitekniikkaa: staattisen, dynaamisen ja hybridianalyysin. Staattisessa analyysissä voidaan esimerkiksi analysoida verkkosovelluksen lähdekoodeja sellaisenaan ilman, että niitä ajetaan. Staattisiin analyyseihin kuuluu useita eri tekniikoita, esimerkiksi tietovirtojen analyysi, osoitinanalyysi ja merkkijonoanalyysi (Li ja Xue 2011). Näillä analyyseillä voidaan tunnistaa epäluotettavan tiedon kulkua verkkosovelluksessa. Staattisia analyysejä rajoittaa niiden kyky mallintaa ohjelmointikielten, kuten JavaScriptin dynaamisia ominaisuuksia, esimerkiksi oliosuuntautunutta koodia (Li ja Xue 2011). Esimerkiksi WebSSARI (Web application Security Analysis and Runtime Inspection) on staattisen ohjelmistoanalyysin tekemiseen tarkoitettu työkalu.

Dynaamisessa analyysissä seurataan verkkosovelluksen käyttäjän syötteiden kulkua koodien suorituksen aikana instrumentoimalla (Li ja Xue 2011). Esimerkiksi sovelluksen lähdekoodien sekaan voidaan kirjoittaa erillistä koodia, jonka avulla pystytään seuraamaan käyttäjän syötteen kulkua tai muuten seuraamaan sovelluksen suoritusta. Li ja Xue (2011) toteavat, että dynaamisen analyysin ero verrattuna staattiseen on se, ettei dynaaminen vaadi monimutkaisten lähdekoodien analysointia, jotta analyysin tarkkuutta saadaan parannettua, mutta kuitenkin sovelluksen toiminnan seurannan takia lisätyt koodit voivat vaikuttaa negatiivisesti sovelluksen suorituskykyyn ja vakauteen, eikä analyysin täydellisyyttä voida taata. Staattisella analyysillä tunnistetaan yleensä kaikki mahdolliset haavoittuvuudet, mutta se löytää herkästi myös virheellisiä (Li ja Xue 2011). Dynaamisella analyysillä puolestaan voidaan olla varmoja, että tietyistä koodin kohdista tunnistetut haavoittuvuudet ovat oikeellisia, mutta kuitenkaan ei voida taata kaikkien haavoittuvuuksien löytyneen (Li ja Xue 2011). Hybridianalyysi on nimensä mukaan kahden edellä mainitun analyysitekniikan yhdistelmä ja se yhdistää sekä staattisen, että dynaamisen analyysin vahvuudet, jotta analyysin tarkkuutta saadaan parannettua (Li ja Xue 2011).

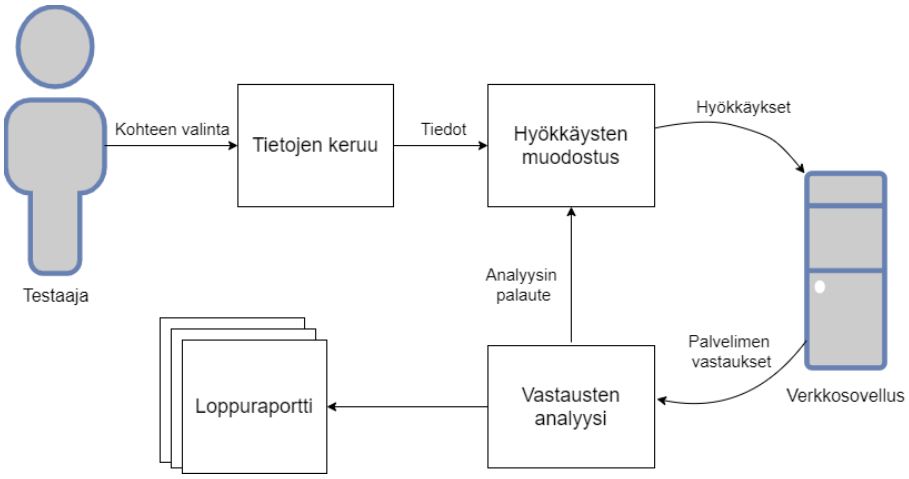
4.2 Penetraatiotestaus

Penetraatiotestaus (engl. Penetration testing) on laajalti käytetty tekniikka, jolla pyritään varmistamaan verkkosovellusten tietoturvaa ja siinä testaajat pyrkivät kartoittamaan mahdollisia haavoittuvuuksia simuloimalla pahantahtoisen käyttäjän tekemiä hyökkäyksiä kohdesovellukseen (Halfond, Choudhary ja Orso 2011). Testauksen suorittavat tietoturva-alan ammattilaiset, joista joskus käytetään myös nimitystä valkohattuhakkerit. Testaajat pyrkivät näillä simuloiduilla hyökkäyksillä esimerkiksi varastamaan verkkosovelluksen tietokannassa olevia tietoja, saamaan luvattomia käyttöoikeuksia tai muuta vastaavaa vahinkoa aikaan sovellukselle. Vaikka hyökkäyksiä käytännössä toteutetaan, niin kuitenkin ammattilaisten tekemänä penetraatiotestaus ei vaaranna oikeasti sovelluksen toimintaa tai sen tietokantoja. Koko prosessin tavoitteena on muodostaa testauksesta raportti, johon raportoidaan onnistuneet hyökkäykset ja kuinka ne toteutettiin. Usein eri yritykset ja organisaatiot tilaavat testauksen palveluna ulkopuoliselta palveluntarjoajalta. Testauksen tuloksena saadun raportin perusteella yritys voi tehdä tarvittavia korjauksia verkkosovellukseensa, jotta löydetyt haavoittuvuudet saadaan korjattua ja täten sovelluksesta turvallinen. Näin yritys pystyy välttämään todellisen hyökkäyksen uhriksi joutumisen seuraukset ja mahdolliset kustannusket.

Halfond, Choudhary ja Orso (2011) toteavat ettei penetraatiotestaus pysty takaamaan, että kaikki haavoittuvuudet löytyvät, mutta silti se on kehittäjien suosiossa useista syistä. Testauksen avulla löydetään yleensä vain oikeellisia haavoittuvuuksia, koska ne etsitään hyödyntämällä niitä simuloidussa hyökkäyksessä. Testaaminen tehdään käytössä olevaan sovellukseen, joten on mahdollista löytää haavoittuvuudet, jotka johtuvat todellisesta asennusympäristöstä. Penetraatiotestauksella saadaan usein sellaista tietoa, jonka avulla verkkosovelluksen kehittäjien on helppoa tehdä tarvittavia korjauksia koodeihin, koska testauksessa raportoidaan hyökkäyksissä käytetyt konkreettiset syötteet. (Halfond, Choudhary ja Orso 2011)

Halfond, Choudhary ja Orso (2011) esittelevät penetraatiotestausprosessia ja sen eri vaiheita. Testaajilla on paljon erilaisia tehtäviä, joita he tekevät testausporsessin aikana, kuitenkin yleisesti se voidaan jakaa kolmeen eri vaiheeseen. Ensimmäisessä vaiheessa testaajat valitsevat verkkosovelluksen testattavaksi ja hankkivat tietoja siitä eri tekniikoilla, esimerkiksi automaattisella skannauksella. Saatujen tietojen perusteella testaajat voivat siirtyä toiseen

vaiheeseen, joka on hyökkäysten muodostus eli siinä kehitetään hyökkäykset, joilla verkkosovellusta testataan. Tämä vaihe on monesti automatisoitavissa esimerkiksi muokkaamalla yleisesti tunnettuja hyökkäyksiä tai käyttämällä automaattisia hyökkäysskriptejä. Hyökkäysten jälkeen testaajat siirtyvät kolmanteen vaiheeseen, joka on palvelimelta saatujen vastausten analyysi. Siinä analysoidaan sovelluspalvelimelta saadut vastaukset, joista selvitetään onnistuivatko hyökkäykset ja sen perusteella laaditaan lopullinen raportti löydetyistä haavoittuvuuksista. Kuvio 5 havainnollistaa prosessin. (Halfond, Choudhary ja Orso 2011)



Kuvio 5. Kuvaus penetraatiotestausprosessista, mukailtu lähteestä (Halfond, Choudhary ja Orso 2011)

4.2.1 Black box- ja white box-testaus

Black box- ja white box-testaus ovat yleisesti tunnetut termit, joita näkee monesti käytettävän sovellusten tietoturvatestaamisen yhteydessä. Liu ja Kuan Tan (2009) kuvaavat black box-testauksen olevan funktionaalinen testaustekniikka, joka muodostaa testitapaukset määrittelyjen pohjalta, kun taas white box-testaus on rakenteellinen testaustekniikka, jossa testitapaukset muodostetaan lähdekoodista saatujen tietojen perusteella. Näitä kahta testaustyyliä pidetään toisiaan täydentävinä.

Testaus tyylien eroina on tiedot, joita testaajalla on sovelluksesta, kun testaus suoritetaan. Black box-testauksessa testaajalla ei tulisi olla pääsyä sovelluksen lähdekoodeihin, kun taas

white box-testauksessa testaajalla on pääsy niihin. Black box-testauksessa testin kohteen voi ajatella olevan laatikko, johon testaaja syöttää dataa ja saa jotain dataa takaisin, eikä testaaja tunne tarkemmin sovelluksen toteutusta. White box-testauksessa puolestaan testattavan sovelluksen voi ajatella olevan lasikuutiossa, jolloin testaajalla on pääsy kaikkiin sovelluksen lähdekoodeihin ja täten niitä analysoimalla testaajalla on ymmärrys sovelluksen toimintaperiaatteesta. Yleensä penetraatiotestaajat olettavat, että hyökkääjillä on pääsy ainakin johonkin lähdekoodin versioon (Halfond, Choudhary ja Orso 2011). Kuitenkin siitä huolimatta penetraatiotestaus on yleensä black box-testausta, koska siinä testaaja on käytännössä samassa tilanteessa kuin oikeasti mahdollinen hyökkääjä ja täten testauksessa saadaan todennäköisemmin oikeellista informaatiota verkkosovelluksen mahdollisesta haavoittuvuudesta.

5 Yhteenveto

Kirjallisuuskartoituksessa käsiteltiin yleisesti verkkosovelluksia, niiden tietoturvauhkia, keskittyen kahteen hyvin yleiseen koodi-injektioon ja lopuksi keskityttiin verkkosovellusten suojaamiseen ohjelmistoanalyysien ja penetraatiotestaamisen avulla. Nykyään on vielä paljon verkkosovelluksia, jotka ovat haavoittuvaisia koodi-injektioille. Tutkielmassa käsitellyistä SQL- ja skripti-injektioista todettiin, että ne johtuvat yleensä haavoittuvuuksista verkkosovellusten syötteiden tarkistuksissa. Näillä kahdella hyökkäystyylillä hyökkääjät pyrkivät esimerkiksi varastamaan arkaluonteisia tietoja sovelluksen tietokannoista, saamaan luvattomia käyttöoikeuksia ja kokonaan ohittamaan tunnistautumisen. Koska nykyään verkkosovellukset käsittelevät usein arkaluonteista tietoa, niiden suojaamisen tärkeys on korostunut entistä enemmän. Käsitellyillä suojaamistekniikoilla pyritään löytämään verkkosovelluksesta mahdolliset haavoittuvuudet, jotta kehittäjät voivat korjata niitä aiheuttavat ohjelmointivirheet. Näin saadaan parannettua verkkosovellusten tietoturvaa, estettyä mahdolliset tietovuodot ja turvattua yksityishenkilöiden, sekä yritysten ja organisaatioiden yksityiset tiedot.

Lähteet

AlKhurafi, O. B., ja M. A. AlAhmad. 2015. “Survey of Web Application Vulnerability Attacks”. Teoksessa *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT),* 154–158. https://doi.org/10.1109/ACSAT.2015.46.

Atashzar, H., A. Torkaman, M. Bahrololum ja M. H. Tadayon. 2011. “A survey on web application vulnerabilities and countermeasures”. Teoksessa *2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT),* 647–652.

Gupta, M. K., M. C. Govil ja G. Singh. 2014. “Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey”. Teoksessa *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014),* 1–5. https://doi.org/10.1109/ICRAIE.2014.6909173.

Halfond, William G. J., Shauvik Roy Choudhary ja Alessandro Orso. 2011. “Improving penetration testing through static and dynamic analysis”. *Software Testing, Verification and Reliability* 21 (3): 195–214. https://doi.org/https://doi.org/10.1002/stvr.450. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.450. https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.450.

Jazayeri, Mehdi. 2007. “Some trends in web application development”. Teoksessa *Future of Software Engineering (FOSE’07),* 199–213. IEEE.

Johari, R., ja P. Sharma. 2012. “A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection”. Teoksessa *2012 International Conference on Communication Systems and Network Technologies,* 453–458. https://doi.org/10.1109/CSNT.2012.104.

Kindy, D. A., ja A. K. Pathan. 2011. “A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques”. Teoksessa *2011 IEEE 15th International Symposium on Consumer Electronics (ISCE),* 468–471. https://doi.org/10.1109/ISCE.2011.5973873.

Li, Xiaowei, ja Yuan Xue. 2011. “A survey on web application security”. *Nashville, TN USA* 25 (5): 1–14.

Lin, J., ja J. Chen. 2007. “The Automatic Defense Mechanism for Malicious Injection Attack”. Teoksessa *7th IEEE International Conference on Computer and Information Technology (CIT 2007),* 709–714. https://doi.org/10.1109/CIT.2007.21.

Liu, Hui, ja Hee Beng Kuan Tan. 2009. “Covering code behavior on input validation in functional testing”. *Information and Software Technology* 51 (2): 546–553. ISSN: 0950-5849. https://doi.org/https://doi.org/10.1016/j.infsof.2008.07.001. https://www.sciencedirect.com/science/article/pii/S0950584908000955.

Shar, Lwin Khin, ja Hee Beng Kuan Tan. 2012. “Automated removal of cross site scripting vulnerabilities in web applications”. *Information and Software Technology* 54 (5): 467–478. ISSN: 0950-5849. https://doi.org/https://doi.org/10.1016/j.infsof.2011.12.006. https://www.sciencedirect.com/science/article/pii/S0950584911002503.

Su, Zhendong, ja Gary Wassermann. 2006. “The Essence of Command Injection Attacks in Web Applications”. *SIGPLAN Not.* (New York, NY, USA) 41, numero 1 (tammikuu): 372–382. ISSN: 0362-1340. https://doi.org/10.1145/1111320.1111070. https://doi-org.ezproxy.jyu.fi/10.1145/1111320.1111070.

Wichers, Dave, ja Jeff Williams. 2017. *Owasp top-10 2017.* Saatavilla WWW-muodossa, https://owasp.org/www-project-top-ten/2017/, viitattu 19.2.2021.