Pyry Kärki

SQL-injektio: toiminta ja ehkäisy

Tietotekniikan kandidaatintutkielma

29. huhtikuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pyry Kärki

Yhteystiedot: pyry.p.karki@student.jyu.fi

Ohjaaja: Rossi Tuomo

Työn nimi: SQL-injektio: toiminta ja ehkäisy

Title in English: SQL injections and prevention techniques

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 21+0

Tiivistelmä: Tietokannat ovat useiden sovellusohjelmien käytössä olevia määriteltyjä kokonaisuuksia, joita voidaan hyödyntää reaalimaailmassa. Sovellusohjelmat mahdollistavat tietokantojen datan hakemisen ja visualisoinnin kyselykielillä, joista käytetyin on SQL. Kyselykielen väärinkäyttö sovellusohjelmassa mahdollistaa tietokannan arkaluonteisen datan hakemisen ja muokkamisen. Ilmiötä kutsutaan SQL-injektioksi. Kirjallisuuskatsauksen tarkoituksena on selvittää kuinka SQL-injektio toteutetaan ja kuinka sen toimintaa voidaan ehkäistä.

Avainsanat: SQL-injektio, tietokanta, SQL, web-sovellukset, tietoturva

Abstract: Databases are a collection of defined data, frequently used by application programs. Application programs use and visualize data of the databases by query languages, which the most noted is the SQL language. Misuse of query language in an application programs allows exploitation of delicate data and unauthorized editing of the database. One of these types of misuse is known as SQL injection. This literature review seeks to identify technical aspects of an SQL injection and how to prevent it.

Keywords: SQL injection, SQL, database, web application security

Kuviot

Sisällys

1 Johdanto

Tietokannat ovat muodostuneet kriittisiksi komponenteiksi nykypäivän yhteiskunnassa. Abstraktisti tietokantojen voidaan sanoa olevan kokonaisuus dataa jostain reaalimaailman ilmiöstä. Tietomallit määrittelevät tietokannan rakennetta sekä toiminnallisuutta, joista yleisin on Codd 1970:n esittelemä relaatiotietomalli. Relaatiotietokantojen hallinnoimiseksi luotiin IBM:n toimesta SQL-kyselykieli, joka lopulta standardoitiin. Standardoitua SQL-kieltä on jatkossa hyödynnetty useisiin eri tietokannanhallintajärjestelmiin.

Tietokantoja halutaan usein väärinkäyttää niiden arkaluontoisen ja arvokkaan datan takia. Tietokantojen väärinkäytöllä tarkoitetaan datan luvatonta hakemista, poistamista tai muokkaamista tietokannanhallintajärjestelmän kyselykielen avulla. Vaikka tietokantaa hallitaankin kyselykielellä, väärinkäyttö toteutetaan usein datan esittävän sovellusohjelman haavoittuvuuden kautta. Tätä haavoittuvuutta kutsutaan SQL-injektioksi.

SQL-injektiot mahdollistavat tietokannan datan väärinkäytön sovellusohjelmien haavoittuvuuksien avulla. Ensimmäisen kerran SQL-injektiosta mainittiin vuonna 1998 Phrachk-verkkolehdessä (Clarke-Salt 2012). SQL-injektion tarkoituksena on hyödyntää sovellusohjelman haavoittuvuutta, jossa hyökkääjän on mahdollista lähettää ohjelmalle haitallista SQL-koodia. Vaikka kyseiset haavoittuvuudet ovat olleet tiedossa jo pitkään, niiden käyttö on lisääntynyt automaation ja laitteiston kehityksen myötä. Injektoiden avulla voidaan aiheuttaa taloudellisten vahinkojen lisäksi myös henkilövahinkoja varastetun arkaluontoisen datan julkaisemisella tai tuhoamisella (Tang ym. 2020).

Tämän kandidaatintutkielman tarkoituksena on selvittää SQL-injektioiden toimintaperiaatteita aikaisemman kirjallisuuden avulla. Tiedostaessa injektoiden mahdollistavat haavoittuvuudet, voidaan myös pohtia ehkäisykeinoja näiden estämiseksi. Koska injektioita on hyödynnetty usealla vuosikymmenellä, tutkitaan yleisimpien torjuntametodien lisäksi myös uusia metodeja ehkäistä tietomurtoja.

Tutkielma koostuu teoriaosuudesta, tutkimusongelman käsittelyosuudesta, sekä pohdintaosuudesta. Luvussa 2 käsitellään aiheet tietokanta, SQL, sekä lyhyesti sovellusohjelma. Luvussa 3 käsitellään SQL-injektion toimintaa yksinkertaisten esimerkkien avulla, sekä kä-

sitellään eri injektiometodeja. Luvussa 4 hyödynnetään aikaisempien kappaleiden tiedoilla ehkäisymetodeja injektioiden torjumiseksi, sekä pohditaan ohjelmistokehittäjien vaikutusta injektoiden ilmenevyyteen. Lopuksi luku 5 sisältää yhteenvedon johtopäätöksineen.

2 Taustatiedot

SQL-injektio hyödyntää SQL-kyselykieltä, sekä sovellusohjelman käyttöliittymää suorittaakseen tietomurron tietokantaan. Tämä kappale käsittelee edellämainittuja termejä, joiden avulla ilmiötä selitetään. Kappaleen ja sen jälkeisissä esimerkeissä käytetään MySQL-tietokannanhallintajärjestelmää, joka noudattaa relaatiotietomallia sekä SQL-standardia.

2.1 Tietokanta

Tietokanta (database) on looginen kokoelma toisiinsa liittyvää määritettyä informaatiota (dataa) reaalimaailman ilmiöstä (universe of discourse, UoD), jonka muutokset peilaantuvat tietokantaan. Tietokannalle tärkeitä ominaisuuksia ovat sen ennalta määritelty rakenne, käyttötarkoitus sekä käyttäjäkunta (Elmasri 2013).

Tietokannoille on olemassa tietomalleja, jotka määrittävät tietokannan rakenteen ja toimintaperiaatteen abstraktiivisesti. Tunnetuin ja yleisimmin käytetty tietokantamalli on relaatiomalli (DB Engines 2021b), jonka teorian Edgar F. Codd loi vuonna 1970. Relaatiomallissa data esitetään matriiseina, jonka rivit muodostavat monikon (tuple). Monikot puolestaan muodostuvat attribuuteista, jotka ovat määritelty matriisin ylimmässä sarakkeessa eli otsakkeessa (header). Monikoitten muodostamat matriisit moudostavat relaation. Relaation monikoiden on oltava erotettavissa toisistaan. Tätä varten relaatiosta valitaan yksi attribuutti, jota kutsutaan perusavaimeksi (primary key). Perusavain on aina yksilöivä ja se ei voi olla tyhjäarvo. Perusavaimet mahdollistavat viittamisen myös muihin relaatioihin (Codd 1970).

Tietokannan hallinnoimiseksi käytetään tietokannanhallintajärjestelmää, eli DBMS:ää (Database Management System). Tietokannanhallintajärjestelmät ovat joukko ohjelmia, joiden avulla luodaan ja ylläpidetään tietokantoja, mahdollistaen myös niiden suojaamisen ja jakamisen sovellusohjelmaan. Tietokannanhallintajärjestelmät määrittävät tietokannassa käytettävän datan tyypityksen, rakenteen sekä rajoitukset tiettyä tietokantatyyppiä noudattaen. DBMS sisältää kyselykielen (query language), jonka avulla varsinaista tietokantaa käytetään. Suosituimpia tietokannanhallintajärjestelmiä ovat Oracle, MySQL ja Microsoft SQL Server (DB Engines 2021a). DBMS:n sisältämä kyselykieli on standardoitua, sisältäen yleensä vain

pieniä eriäväisyyksiä toisiinsa.

2.2 SQL

SQL eli Structured Query Language on IBM-yhtiön kehittämä kyselykieli, jolla hallinnoidaan relaatiotietokantoja. Toiminta perustuu lauseisiin, joiden avulla voidaan muokata, lisätä, hakea tai poistaa dataa tietokannasta. SQL julkistettiin ensimmäisen kerran vuonna 1974 nimellä SEQUEL (Structured English Query Language), joka pian lyhennettiin muotoon SQL. Ensimmäiset standartoinnit luotiin vuonna 1986 ANSI:n (American National Standards Institute) ja kansanvälisesti vuonna 1987 ISO:n (International Organization for Standardization).

Useat ohjelmointikielet, kuten Python tai C tukevat SQL-kielellä muodostettuja lauseita esikääntäjän avulla, mahdollistaen relaatiotietokantojen käytön isäntäkielessä (host language). Käytäntöä kuvataan yleisesti termillä *embedded SQL* (Jan L. 2003). Kyselyitä voidaan muodostaa sekä staattisesti, että dynaamisesti käyttötarkoituksesta riippumatta. Dynaamisesti muodostetuissa kyselyissä hyödynnetään usein käyttäjän antamia rajoitteita halutun datan hakemiseksi. Toiminnan havainnollistamiseksi alla oleva esimerkki esittää yksinkertaisen SQL-lauseen, joka hakee tietokannassa olevasta relaatiosta dataa.

```
1 SELECT productname
2 FROM products
3 WHERE productname LIKE 'k%';
```

Ensimmäisen rivin SELECT-käsky määrittää DBMS:n hakevan tietokannasta kaikki *productname* attribuutin sisältävät arvot. Rivillä kaksi käsky FROM määrittää relaation, josta attribuutti halutaan hakea. Kolmas rivi määrittää lisäehdon, joka rajaa hakulausetta. Lisäehtojen määrä ei ole rajattu ja niillä voidaan tarvittaessa viitata myös toisiin relaatioihin. Kun lause suoritetaan, tietokannasta haettu data palauttaa kaikki k-kirjaimella alkavien attribuuttien arvot.

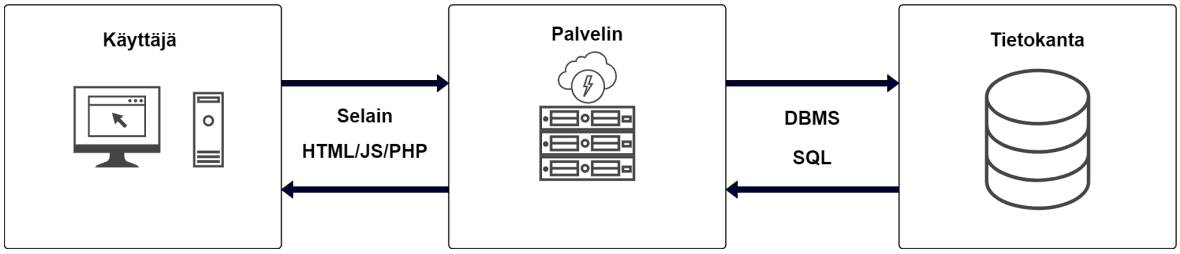
Koska SQL on kyselykieli, datan esitystä varten toteutetaan sovellusohjelma joka toimii rajapintana tietokantaa varten. Datan esityksen lisäksi sovellusohjelmalla määritetään, voivatko

käyttäjät lukea tai muokata tietokannan dataa ja millä tavalla.

2.3 Sovellusohjelma

Sovellusohjelma on käyttäjiä varten suunniteltu ohjelma, jonka avulla voidaan hallita laitteiston toimintaa tai suorittaa prosesseja. Usein sovellusohjelmilla halutaan ratkoa tietty ongelma tai lyhentää sen suoritusaikaa.

Tietokantoja käyttävien sovellusohjelmien tarkoituksena on mahdollistaa halutun datan esitys, hallinta tai molemmat. Ohjelmiston suunnitteluvaiheessa määritetään, kuinka ja ketkä voivat käyttää tietokannan dataa. Yleisesti käyttäjät voivat vain hakea ja tutkia ennaltamääriteltyjä osioita tietokannan datasta ja järjestelmänvalvoja hallinnoi tietokantaan kohdistuvat muokkaukset ja lisäykset. Tietokantoja hyödyntävissä sovellusohjelmissa data haetaan usein dynaamisilla SQL-lauseilla, jossa käyttäjä määrittää lauseelle lisäehtoja. Toiminnot tapahtuvat suurilta osin käyttäjän näkemättä välivaiheita, näyttäen ainoastaan lopputulokset. Kun käyttäjä lähettää pyynnön hakea tietokannasta dataa, ohjelman ohjelmointikieli muodostaa dynaamisesti SQL-kyselyn halutuilla määritteillä. Kysely jatketaan eteenpäin palvelimelle, joka siirtää haun eteenpäin DBMS:lle. DBMS suorittaa varsinaisen haun tietokantaan ja palauttaa tuloksesta saadun datan. Sovellusohjelma muotoilee DBMS:n palauttaman datan käyttäjälle haluttuun esitysmuotoon (Jan L. 2003). Alla oleva toimintakaavio esittää yksinkertaistettuna web-pohjaisen sovellusohjelman toiminnan.



Kuvio 1. Toimintakaavio tietokantaa hyödyntävästä web-pohjaisesta sovellusohjelmasta

3 SQL-injektio

SQL-injektio (SQLI) on tekniikka, jota hyödynnetään tietoturva-aukon sisältämässä sovellusohjelmassa tietomurron toteuttamiseksi. SQL-injektion tarkoituksena on lisätä tai muuttaa tietokantaan kohdistettavia käskyjä, joiden avulla voidaan hakea tai muokata tietokannan dataa tai kirjautua järjestelmään ilman oikeanlaisia tunnistetietoja. Vaikka valtaosa SQL-injektioista suoritetaan web-pohjaisille sovelluksille, voidaan tekniikkaa käyttää mihin tahansa SQL-kyselyjä käyttävään sovellusohjelmaan. Perusperiaatteena injektiot käyttävät aina hyväkseen dynaamisesti muodostettu kyselyjä käyttäjän lisäämillä ehdoilla (Morgan 2006).

SQL-injektiot voidaan jakaa eri tyyppeihin toimintaperiaatteidensa mukaan (Ma ym. 2019). Injektiotyyppien käyttö vaihtelee hyökkääjän tietämyksestä, mitä tietokannanhallintajärjestelmää ohjelma käyttää sekä mitä tietoturva-aukkoja sovellusohjelma sisältää. Hyökkääjän ei kuitenkaan tarvitse olla tietoinen ohjelman täydellisestä toiminnasta: onnistuneen injektion toteuttamiseksi vaaditaan yleensä useita kokeiluja haavoittuvuuksien todentamiseksi ja hyödyntämiseksi. Haavoittuvuuksia voidaan myös etsiä erilaisten työkalujen kautta (Clarke-Salt 2012).

3.1 Klassinen SQL-injektio

Klassinen SQL-injektio (classic SQLI) on yleisin ja helppokäyttöisin injektiotyyppi (Clarke-Salt 2012). Kyseisessä injektiometodissa on tarkoituksena lisätä dynaamisen SQL-lauseen WHERE-operandille lisäehtoja ja parametrejä, joilla lausekkeiden logiikkaa muutetaan. Lauseille voidaan antaa myös UNION-operandi, jonka avulla kyselyyn voidaan lisätä kokonaan uusia SQL-lauseita, mahdollistaen tietokannan relaatioiden muokkaamisen tai lisäämisen ja poistamisen. WHERE-lausekkeiden tyypillisin käyttötarkoitus on järjestelmiin kirjautuminen ilman todenmukaisia tunnistetietoja (Yunus ym. 2018). Tätä havainnollistaakseen alla on esimerkki yksinkertaisesta klassisesta SQL-injektiototeutuksesta, jossa järjestelmään kirjaudutaan ilman tiedossa olevia tunnuksia.

Kirjautumissivustolla hyökkääjä syöttää käyttäjätunnukseksi ja salasanaksi merkkijonon *' OR '1' = '1'*. Sovellusohjelma muodostaa dynaamisesti käyttäjän täydentämän SQL-lauseen.

```
SELECT *
FROM users
WHERE username = '' OR '1'='1'
AND password = '' OR '1'='1';
```

Lauseen alkuperäisenä tarkoituksena on tarkastaa, löytyykö tietokannasta käyttäjää annetuilla parametreillä. Hyökkääjän antama parametri lisää lauseelle lisäehdon, joka tarkistaa tunnusten lisäksi onko 1 sama kuin 1. Tämä muuttaa lauseen logiikkaa: SQL:n ei tarvitse todentaa, onko käyttäjänimi sekä salasana oikein, koska lisäehto palauttaa aina arvon tosi. Hyökkääjän on siis mahdollista nyt kirjautua järjestelmään ilman oikeanlaisia tunnuksia, joita tavallisella toiminnalla vaadittaisiin. Samankaltaista tekniikkaa voidaan soveltaa kirjautumislomakkeiden lisäksi myös muissa lomakkeissa, jossa ohjelma vastaanottaa käyttäjän syöttämää tekstiä josta muodostetaan SQL-lause. Web-sovellukset, joissa sivun osoite (URL) sisältää SQL-parametrejä, voidaan hyödyntää samalla tavalla klassisen injektion toteuttamiseksi. WHERE-lausekkeen muokkaamisella voidaan vaikuttaa ainoastaan siihen relaatioon, mihin se on alunperinkin tarkoitettu, mutta UNION-operandin lisäyksellä voidaan viitata myös toisiin relaatioihin. UNION-operandi sallii useamman SQL-lausekkeen yhdistämisen, joka ajetaan yhtenäisenä palvelimelle. Haasteena kyseisen operandin käytössä on koko lausekkeen muodostama rakenne, jonka on oltava samankaltainen lausekkeen jokaiselta osalta jotta lause olisi syntaktisesti oikein.

Klassisen SQL-injektion toteutuksessa on usein tärkeää, kuinka lausekkeelle muodostetaan suojausmerkki (escape character). Merkistön toimintaperiaatteet vaihtelevat DBMS:n mukaan. SQL:n muodostamissa kyselyissä merkkijonot rajoitetaan heittomerkkien sisälle, jonka avulla voidaan erottaa parametrit sekä operandit. Jos aikaisemmin mainitusta esimerkistä *' ' OR '1' = '1'* poistettaisiin keskimmäiset heittomerkit, SQL lukisi sen yhtenä merkkijonona. Heittomerkkien lisäksi voidaan hyödyntää myös puolipistettä, joka päättää SQL-lauseen. Suojausmerkkien avulla voidaan siis vaikuttaa kyselyjen rakenteeseen lisäämällä lausekkeita kyselyjen lauseisiin (Ma ym. 2019).

SQL-lauseiden ollessa syntaktisesti puutteellinen, SQL palauttaa virheilmoituksen. Ohjelman toteutuksesta riippuen virheilmoitus voidaan palauttaa myös suoraan käyttäjälle. Hyök-

kääjä voi hyödyntää virheilmoituksia injektion toteutuksessa. Virheilmoitusten avulla hyökkääjän on mahdollista päätellä tietokannan rakennetta ja sen relaatioita. Tämä samalla lyhentää tietomurron toteutukseen kuluvaa aikaa.

3.2 Sokea SQL-injektio

Sokea SQL-injektio (blind SQL injection) eroaa klassisesta injektiotyypistä siten, että hyökkääjä ei voi hyödyntää SQL:n antamia virheilmoituksia, eikä tietokannasta palautettua dataa palauteta suoraan käyttäjälle. Sokealle injektiolle on tyypillistä, että hyökkääjän virheelliset kyselylauseet aiheuttavat ohjelmassa geneerisen virheilmoituksen ilman spefisejä tietoja, tai toimivan lauseen suorittamat muutokset näkyvät sovellusohjelmassa joko heikosti tai ei ollenkaan (Clarke-Salt 2012).

Vaikka ohjelma ei näyttäisi käyttäjälle SQL-lauseiden virheilmoituksia, voidaan injektiota silti hyödyntää haavoittuvaisessa ohjelmassa. Dynaamisesti muodostetuille SQL-lauseille voidaan syöttää lisäehtoja samalla tavalla kuin klassisessa injektiotyypissä. Ohjelman palauttaessa vastauksen alkuperäisesti suunnitellun toimintatavan mukaan hyökkääjä voi todeta ohjelman hyväksyneen antamansa lisäehdon. Tietokannan rakennetta voidaan siis tutkia antamalla ohjelmalle tosi-epätosi ehtoja, jotta hyökkääjä pystyisi päättelemään relaatioiden otsakkeita ja attribuutteja (Spett 2003). Sokea injektiotyyppi on hyökkääjän kannalta huomattavasti vaativampi kuin klassinen injektiotyppi, vaatien useampien haitallisten SQL-lauseiden lähettämistä injektion hyödyntämiseksi.

Sokeassa injektiotyypissä voidaan hyödyntää myös ajoittimia, joiden avulla höykkääjä voi arvioida, onko ohjelma haavoittuvainen injektiolle (Morgan 2006). Useimmat tietokannanhallintajärjestelmät sisältävät ajoitustoimintoja, joiden avulla voidaan optimoida tietokantojen suorituskykyä. Esimerkkinä injektiossa voidaan hyödyntää MySQL:n benchmark-funktiota.

```
1 BENCHMARK(7000000,ENCODE('STRNG','KSTRNG'))
```

Lauseke suorittaisi saman merkkijonon salauksen useita kertoja kuormittaakseen tietokantaa, jotta ohjelma DBMS palauttaisi vastauksen viiveellä. Viiveen mittaamisella hyökkääjä voi päätellä injektion toimivuutta. Ajoittimia käyttäessä on otettava huomioon myös vakaa

yhteys tietokantaan sekä asetettava viiveen pituus, jotta tietokanta ei katkaise yhteyttä.

Sokean SQL-injektion toteutus on yleensä pitkäaikaista (Ma ym. 2019). Suoritusaikaa lyhentääkseen voidaan hyödyntää automaatiota erilaisilla ohjelmilla, joita käytetään tietokantojen tutkimiseen ja testaamiseen. Yleisimpiä testausohjelmia ovat mm. Absinthe, BSQL, Marathon Tool ja Havji (Singh ym. 2016).

3.3 Haavoittuvuudet DBMS:ä sekä yhdistetyt injektiot

Toisinaan itse tietokannanhallintajärjestelmissä voi ilmeentyä tietoturva-aukkoja, jotka mahdollistavat injektion toteuttamisen. Esimerkiksi MySQL sisältää *mysql_real_escape_string()*-funktion, jolla voidaan määrittää erikoismerkkejä merkkijonossa. Hyökkääjä voi käyttää funktiota injektion toteutuksessa, pakottaen haitallisia lausekkeita tietokantakyselyihin. Haavoittuvuus on kuitenkin tunnettu ja monet kyseistä funktiota käyttävät ohjelmointikielet ovat rajoittaneet sen käyttöä sekä tilalle on luotu vaihtotehtoisia, turvallisempia funktioita (Ma ym. 2019).

SQL-injektiota voidaan käyttää myös muiden haavoittuvuuksien rinnalla. XSS eli cross site scripting on web-pohjaisissa ohjelmissa esiintyvä haavoittuvuus, jossa hyökkääjä voi syöttää ohjelmalle komentoja, ns. skriptejä tunnistetietojen varastamiseen tai sivuston pääsyoikeuksien kiertämiseen (Rodríguez ym. 2020). SQL-injektion ja XSS:n yhdistelmällä hyökkääjällä on suurempi mahdollisuus tietomurron toteutukseen, sekä mahdollisuus hyödyntää muita ohjelmassa olevia haavoittuvuuksia tehden hyökkäyksestä vakavamman. Tällöin myös kohteeksi joutuneen osapuolen vahingoista toipuminen vaikeutuu (Clarke-Salt 2012).

4 Ehkäiseminen

SQL-injektio on laajasti tunnistettu tietoturva-aukko, jonka ehkäisemiseksi on olemassa useita eri metodeja. Tietomurron estämiseksi ohjelmistojen suunnittelu- ja toteutusvaiheissa voidaan ennaltaehkäistä suurin osa haavoittuvuuksista. Tietomurron toteutuessa on tärkeää tutkia, millaista dataa tietokannasta on kyetty varastamaan ja kuinka tietomurto on toteutettu, jotta vastaisuudessa samankaltaista toimintaa voidaan ehkäistä (Clarke-Salt 2012).

4.1 Haavoittuvuuksien tunnistaminen sovellusohjelmissa

Sovellusohjelmien turvallisuuden kannalta lähes välttämätöntä tarkistaa käyttäjän antamia syötteitä, jotta järjestelmälle ei annettaisi haitallisia komentoja. Yleisin torjuntamenetelmä SQL-injektioiden estämiseksi on parametrisoitujen SQL-lauseiden käyttö. Parametrisoiduilla lauseilla käyttäjän syöte liitetään staattiseen SQL-lauseeseen. Tällöin käyttäjän syötteen sisältämät suojausmerkit eivät muokkaa lauseen rakennetta tai logiikkaa. Turvallisuuden lisäksi parametrisoidut lauseet parantavat ohjelman suorituskykyä (Elmasri 2013). Injektioita voidaan estää myös säännöllisillä lauseilla (regular expression). Ohjelmien käyttäessä säännöllisiä lauseita käyttäjän antamaa syötettä voidaan muotoilla validimmaksi. Säännöllisillä lauseilla tarkoitetaan yleistämistä, jossa parametria muokataan yleisesti hyväksyttyyn muotoon. Esimerkiksi sähköpostiosoitteilla tai salasanoilla on usein säännöllinen muoto. Säännöllisesti muotoiltu dynaaminen SQL-lause voidaan joko hyväksyä palvelimelle jatkettavaksi, tai hylätä kokonaan antaen virheilmoituksen virheellisestä syötteestä (Soewito ym. 2018).

Sovellusohjelmoihin voidaan myös lisätä erilaisia tarkastimia, jotka tarkastavat dynaamisesti muodostettavia lauseita. Tarkistimet sisältävät merkkijonoja, joiden injektioiden tiedetään käytettävän ja poistavat niiden avulla dynaamisista SQL-lauseista haavoittuvuuksia sisältävät osat. Tarkistimien kannalta on tärkeää, että merkkijonot käyttävät aina samaa merkistökoodausta, kuten ASCII-muotoa (Balasundaram ja Ramaraj 2012).

SQL-injektioiden tunnistamiseksi voidaan hyödyntää myös koneoppimista. Valeur, Mutz ja Vigna 2005 ehdottavat vertailumetodia, jossa käyttäjien lähettämät SQL-kyselyt tallennetaan. Ohjelmalle annetaan myös malleja SQL-lauseista jotka mallintavat ohjelman alkupe-

räistä käyttötarkoitusta. Tietokannalle saapuvia uusia kyselyitä mallinnetaan ohjelman annettuihin malleihin sekä tietokannan vanhoihin kyselyihin. Eroavaisuuksien perusteella haitalliset kyselyt voidaan erottaa käyttötarkoituksen mukaisista kyselyistä estäen injektoiden toteutumista.

Injektioiden estoissa voidaan hyödyntää myös neuroverkkoja (Tang ym. 2020). Neuroverkkojen etuutena injektiontorjunnassa dataa voidaan käsitellä sekä vastaanottaa suuria määriä. Datan määrän kasvaessa neuroverkon tehokkuus kasvaa. Tämä johtaa vielä tarkempaan injektiotunnistukseen kuin koneoppiminen, pienentäen tietomurron riskiä. Koneoppimisen ja neuroverkkojen haittapuolena voidaan pitää tietokannan suorityskyvyn heikentyneisyyttä, joka modernilla laitteistolla kuitenkin on lähes marginaalista.

Web-pohjaisille sovelluksille on kehitetty myös palomuureja (WAF eli web application firewall). Niiden tarkoituksena on tunnistaa ja ehkäistä ohjelmalle suunnattuja hyökkäyksiä. WAF:t tarvitsevat kuitenkin usein muutoksia toimiakseen käyttötarkoituksensa mukaan ohjelmistoissa, jonka takia niiden käyttöönotto voi olla hankalaa (Heiderich ym. 2011).

4.2 Tietokannanhallintajärjestelmien tietoturva

Tietokannanhallintajärjestelmät sisältävät useita toimintoja, joiden avulla voidaan ehkäistä sekä vähentää tietomurron vahingollisuutta. Tietokannan käyttäjille voidaan valtuuttaa sekä evätä luku- ja kirjoitusoikeuksia DBMS:n avulla. Tavalliselta käyttäjältä voidaan siis evätä pääsy sensitiivisen datan tutkimiseen ja muokkaamiseen. Yleisesti suotava malli on sallia käyttäjälle minimaaliset käyttöoikeudet ohjelman oikeanlaisen toiminnallisuuden puitteissa (Elmasri 2013).

Tietokannan datan eheydellä varmistuksella sekä virheidenhallinnalla voidaan estää korruptoituneen datan ja haitallisten kyselyjen ajo tietokannalle. SQL sisältää START-, COMMIT- sekä ROLLBACK-komentoja tapahtumien hallitsemiseen (Jan L. 2003). Esimerkiksi havaittaessa virheen, voidaan tapahtuma peruuttaa ROLLBACK-komennolla. Datan attribuuttien tyypityksellä voidaan vähentää korruptoituneen datan ilmenevyyttä (Clarke-Salt 2012).

4.3 Ohjelmistokehittäjien vastuu

Ohjelmistokehittäjillä sekä suunnittelijoilla on pääsääntöinen vastuu ohjelman toimivuudesta sekä tietoturvasta. Haavoittuvuuksien tiedostaminen ja ehkäisy ennen ohjelman julkaisua vähentävät injektoiden kohteeksi päätymistä.

Ohjelmiston suunnitteluvaiheessa tietokannan rakenteen määrittely sekä normalisointi helpottavat varsinaista toteutusta. Tietokannan normalisoinnilla tarkoitetaan loogisten virheiden sekä toisteisuuden minimointia, optimoiden tietokannan suorituskykyä ja toimintavarmuutta ohjelmistossa (Codd 1990). Hyökkääjän varastaman arkaluontaisen tiedon hyväksikäyttöä voidaan estää myös tietokantojen datan salaamisella. Parhaimmassa tapauksessa varastetun datan salausta ei saada purettua, tehden tietomurrosta hyödyttömän.

Ohjelmistojen kehitystiimin on tärkeä olla tietoinen mahdollisista tietoturva-aukoista, joita toteutettavassa ohjelmistossa on aikaisemminkin havaittu. Kehitystiimin koulutus turvallisen ohjelmoinnin suhteen voi olla ratkaisevassa asemassa injektioiden torjumiseen. Tietomurron toteutuessa kehittäjillä tulisi myös olla asianmukaista tietotaitoa tietomurrosta elpymiseen.

Ohjelmistojen toteutukseen käytettävistä työkaluista voi myös olla hyötyä. Aikaisemmin mainittuja tietokantojen testaustyökaluja voidaan käyttää tietomurron ehkäisevässä toiminnassa paljastaen toteutuksessa tapahtuneita haavoittuvuuksia. Ohjelmointiympäristöt sisältävät usein virheentunnistustyökaluja, jotka ilmoittavat syntaktisesti virheellisestä koodista. Kehittyneemmillä virheentunnistuksilla voitaisiin estää myös syntaktisesti validi, mutta haavoittuvaiseksi todettu koodi. Kehitysvaiheessa olisi siis tärkeää käyttää aikaa testaamiseen injektioiden estämiseksi.

5 Yhteenveto

Kandidaatitutkielmassa selvitettiin SQL-injektion toiminta sekä ehdotettiin metodeja niiden ehkäisemiseksi. Injektioiden toteutustavoille ei ole määritetty oikeaa tai väärää menetelmää, sillä toteutustapa on riippuvainen tietokannan rakenteesta, tietokannanhallintajärjestelmästä, sekä sovellusohjelmasta. Hyökkääjälle on usein tärkeää tiedostaa ohjelman toiminta sekä tietokannan rakenne, jolloin injektio on helpompi toteuttaa. SQL-injektioita voidaan myös automatisoida erilaisilla ohjelmilla. Tutkielmassa tuotiin esille kaksi päämenetelmää injektion toteuttamiseksi: klassinen- ja sokea SQL-injektio, joiden sisäisissä toimintamenetelmissä on eroavaisuuksia. Tutkielmassa todettiin myös, kuinka eri haavoittuvuuksien yhdistelmällä voidaan manipuloida ohjelmiston ja tietokannan toimintaa hyökkääjän puolesta.

Vaikka SQL-injektio on yleisesti tunnettu haavoittuvuus, ohjelmistot usein silti sisältävät sen mahdollistavia tietoturva-aukkoja. Koska hyökkääjä käyttää sovellusohjelmaa saman rajapinnan kautta kuin tavallinenkin käyttäjä, on injektiota usein vaikea huomata reaaliajassa ennen kuin tietomurto on onnistuneesti suoritettu. Injektiolle haavoittuvaisen ohjelmiston ylläpitäjän tulisi valvoa tietokannalle annettavia käskyjä huomatakseen tietomurron reaaliajassa.

SQL-injektion ehkäisemiseksi todettiin useita eri käytäntöjä. Useimmat käytännöistä liittyivät sovellusohjelmiston syötteenhallintaan. Käyttäjän antamien syötteiden tarkistus todettiin ensiaikaisen tärkeäksi injektioiden torjumisessa. Syötteen tarkistamiseen voidaan käyttää useita eri metodeja, sekä sitä voidaan myös automatisoida. Tietokannanhallintajärjestelmien sekä sovellusohjelman ohjelmointikielen tietoturvan varmistavien ominaisuuksien oikeanlainen käyttö mahdollistaa tietomurtojen eston. Ohjelmistokehittäjien koulutus turvallisista koodausmenetelmistä pienentävät tietomurtojen riskiä.

SQL-injektiot ovat edelleen yksi yleisimmistä tietoturvauhista ohjelmistoille. Datan varastamisella tai muokkaamisella voidaan aiheuttaa mittavia vahinkoja niin yrityksiin, yksityishenkilöihin, kuin tietojärjestelmiin. Onnistuneita tietomurtoja suoritetaan usein ja niiden vahingot ovat usein kustannuksilta moninkertaisesti suuremmat kuin kehitysvaiheessa ehkäisevään toimintaan käytetty aika ja toimenpiteet. SQL-injektioiden ehkäisevässä toiminnassa

on kuitenkin edetty pitkälle ja uusia ehkäisymetodeja luodaan jatkuvasti. Nykyaikaisten torjuntametodien käyttöönottovastuulle jääkin ohjelmistojen kehitystiimit, ohjelmistokehittäjät sekä suunnittelijat.

Lähteet

Balasundaram, Indrani, ja E. Ramaraj. 2012. “An Efficient Technique for Detection and Prevention of SQL Injection Attack using ASCII Based String Matching”. International Conference on Communication Technology and System Design 2011, *Procedia Engineering* 30:183–190. ISSN: 1877-7058. https://doi.org/https://doi.org/10.1016/j.proeng.2012.01.850. https://www.sciencedirect.com/science/article/pii/S1877705812008600.

Clarke-Salt, Justin. 2012. *SQL Injection Attacks and Defense.* Saint Louis, UNITED STATES: Elsevier Science & Technology Books. ISBN: 9781597499736. http://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=953188.

Codd, E. F. 1970. “A Relational Model of Data for Large Shared Data Banks”. *Commun. ACM* (New York, NY, USA) 13, numero 6 (kesäkuu): 377–387. ISSN: 0001-0782. https://doi.org/10.1145/362384.362685. https://doi.org/10.1145/362384.362685.

———. 1990. *The Relational Model for Database Management: Version 2.* USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201141922.

DB Engines. 2021a. “DB-Engines Ranking of Relational DBMS”. Viitattu 17. maaliskuuta 2021. https://db-engines.com/en/ranking/relational+dbms.

———. 2021b. “DB-Engines Ranking per database model category”. Viitattu 8. maaliskuuta 2021. https://db-engines.com/en/ranking_categories.

Elmasri, Ramez. 2013. *Fundamentals of Database Systems: Pearson New International Edition.* Pearson Education UK. ISBN: 9781292038032. http://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=5248269.

Heiderich, Mario, Eduardo Alberto Vela Nava, Gareth Heyes ja David Lindsay. 2011. “Chapter 8 - Web application firewalls and client-side filters”. Teoksessa *Web Application Obfuscation,* toimittanut Mario Heiderich, Eduardo Alberto Vela Nava, Gareth Heyes ja David Lindsay, 199–216. Boston: Syngress. ISBN: 978-1-59749-604-9. https://doi.org/https://doi.org/10.1016/B978-1-59749-604-9.00008-X. https://www.sciencedirect.com/science/article/pii/B9781597496049000%2008X.

Jan L., Harrington. 2003. *SQL Clearly Explained.* Nide 2nd ed. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann. ISBN: 9781558608764. http://search.ebscohost.com.ezproxy.jyu.fi/login.aspx?direct=true&db=nlebk&AN=209356&site=ehost-live.

Ma, L., D. Zhao, Y. Gao ja C. Zhao. 2019. "Research on SQL Injection Attack and Prevention Technology Based on Web". Teoksessa *2019 International Conference on Computer Network,Electronic and Automation (ICCNEA),* 176–179. https://doi.org/10.1109/ICCNEA.2019.00042.

Morgan, David. 2006. "Web application security – SQL injection attacks". *Network Security* 2006 (4): 4–5. ISSN: 1353-4858. https://doi.org/https://doi.org/10.1016/S1353-4858(06)70353-1. https://www.sciencedirect.com/science/article/pii/S1353485806703531.

Rodríguez, Germán E., Jenny G. Torres, Pamela Flores ja Diego E. Benavides. 2020. "Cross-site scripting (XSS) attacks and mitigation: A survey". *Computer Networks* 166:106960. ISSN: 1389-1286. https://doi.org/https://doi.org/10.1016/j.comnet.2019.106960. https://www.sciencedirect.com/science/article/pii/S1389128619311247.

Singh, N., M. Dayal, R. S. Raw ja S. Kumar. 2016. "SQL injection: Types, methodology, attack queries and prevention". Teoksessa *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom),* 2872–2876.

Soewito, Benfano, Fergyanto E. Gunawan, Hirzi ja Frumentius. 2018. "Prevention Structured Query Language Injection Using Regular Expression and Escape String". The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life, *Procedia Computer Science* 135:678–687. ISSN: 1877-0509. https://doi.org/https://doi.org/10.1016/j.procs.2018.08.218. https://www.sciencedirect.com/science/article/pii/S1877050918315114.

Spett, Kevin. 2003. *Blind sql injection.* Tekninen raportti. Technical report, SPI Dynamics.

Tang, Peng, Weidong Qiu, Zheng Huang, Huijuan Lian ja Guozhen Liu. 2020. "Detection of SQL injection based on artificial neural network". *Knowledge-Based Systems* 190:105528. ISSN: 0950-7051. https://doi.org/https://doi.org/10.1016/j.knosys.2020.105528. https://www.sciencedirect.com/science/article/pii/S0950705120300332.

Valeur, Fredrik, Darren Mutz ja Giovanni Vigna. 2005. “A Learning-Based Approach to the Detection of SQL Attacks”. Teoksessa *Detection of Intrusions and Malware, and Vulnerability Assessment,* toimittanut Klaus Julisch ja Christopher Kruegel, 123–140. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-31645-9.

Yunus, Mohd Amin Mohd, Muhammad Zainulariff Brohan, Nazri Nawi, Ely Mat Surin, Nurhakimah Azwani Md Najib ja Chan Liang. 2018. “Review of SQL Injection : Problems and Prevention”. *JOIV : International Journal on Informatics Visualization* 2 (3-2): 215–219. ISSN: 2549-9904. https://doi.org/10.30630/joiv.2.3-2.144. http://joiv.org/index.php/joiv/article/view/144.