SQL-Injektio ja siltä suojautuminen

Lalli Nuorteva

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 17. helmikuuta 2015

Sisältö

1 Johdanto

Tämän kandidaatintutkielman päätavoite on esitellä selvittää kuinka SQL-injektiolta voidaan suojautua mahdollisimman tehokkaasti. Ensimmäisissä luvuissa esitellään mikä on SQL-injektio. SQL-Injektio esitellään ensin pääpiirteissään. Sen jälkeen esitellään miten se voidaan toteuttaa käytännössä, sekä millaisiin eri alatyyppeihin SQL-injektiot jaotellaan. Kun SQL-Injektion käsite ja lähestymistavat on esitelty, esitellään käytänteet joilla SQL-injektioita pyritään estämään.

Aihe on tärkeä koska verkossa käsitellään jatkuvasti entistä enemmän arkaluontoisia tietoja. Verkossa hoidetaan asioita kuten laskujen maksaminen, hotellien varaus ja henkilökohtaisten viestien vaihtaminen. Yleensä tiedot tallennetaan relaatiotietokantoihin. Juuri tällaiset sovellukset voivat olla haavoittuvaisia SQL-Injektioille, ellei asiaa ole otettu huomioon. Lienee siis itsestäänselvää, että jokaisen tietokantasovelluksia ohjelmoivan on ymmärettävä mikä on SQL-injektio ja kuinka suojautua siltä.

SQL-Injektio on yleisin tietoturva-aukko web-sovelluksissa [5]. Yleisyytensä lisäksi SQL-injektio on myös hyvin vaarallinen. Onnistuneen SQL-injektion avulla hyökkääjä voi tehdä tietokannalle mitä tahansa operaatioita. SQL-Injektiolla on vuosien aikana tehty lukuisia murtoja. Yksi suurimmista murroista tehtiin SQL-Injektiolla Guess.com:ille. Hyökkääjä sai käsinsä 200 000 ihmisen nimet ja luottokorttitiedot. Sen lisäksi että SQL-injektio on yleisin tietoturva-aukko, se on myös helppoa toteuttaa ilman suurempaa ymmärrystä sen toiminnasta. SQL-Injektioiden tekemiseen löytyy valmiita työkaluja, jotka ilmoittavat sivuston heikkouksista lähes tulkoon napin painalluksella. Tästä esimerkkinä "sqlmap"työkalu joka on tehty tunkeutumistestaamista varten *(engl. penetration testing)*

2 Käsitteitä

-SQL - SQL Injection out of band inband

3 SQL-Injektio

Tavallisesti relaatiotietokantoja käyttävissä sovelluksissa tietokanta on oma erillinen palvelimensa. Sovelluksella on tietokantaan omat tunnuksensa. Sovelluksen tietokantatunnuksien oikeuksia voidaan rajata halutulla tavalla. Esimerkiksi sovelluksen tietokantatunnuksilla on harvoin tarpeellista pystyä poistamaan tietokantatauluja. Sovellus kommunikoi tietokantapalvelimen kanssa käyttäen SQL (*Structured Query Language*) kyselykieltä.

SQL-Injektiossa hyökkääjä pääsee suorittamaan sovelluksen tietokannassa itse kirjoittamiaan käskyjä. Mikäli sovelluksen tietokantakäyttäjän oikeuksia ei ole rajattu, hyökkääjä voi onnistuneen sql-injektion seurauksena suorittaa mitä tahansa tietokantapalvelimen tukemia SQL-kyselyitä. Tällöin hyökkääjän on mahdollista esimerkiksi lukea, muuttaa, lisätä tai poistaa mitä tahansa tietokannan tietoja. Jotkut tietokantapalvelimet myös sallivat käyttöjärjestelmätason komentojen suorittamisen. Tällöin hyökkääjän on mahdollista suorittaa myös muunlaisia hyökkäyksiä.

SQL-Injektio on mahdollinen vain silloin kun käyttäjältä tulevaa tietoa käytetään osana tietokantapalvelimelle tehtävää kyselyä. Tämä on kuitenkin varsin tavallinen tarve sovelluksissa. Tietokannassa voidaan säilyttää esimerkiksi käyttäjätunnuksia ja salasanoja. Näin ollen kirjautuessa järjestelmään käyttäjän antamaa syötettä käytetään osana SQL-kyselyä.

3.1 SQL-Injektio käytännössä

Anleyn artikkelin "Advancen SQL Injections in SQL Server Applications"mukaan SQL-Injektio hyökkäys esiintyy sillon, kun hyökkääjä pääsee muuttamaan käskyn logiikkaa, semantiikkaa tai syntaksia. Tämä tapahtuu lisäämällä alkuperäiseen kyselyyn uusia SQL-avainsanoja tai operaattoreita [1]. Esimerkiksi tuotteiden etsimiseen liittyvä sql-käsky voidaan rakentaa sovelluksessa seuraavalla tavalla:

```
sql = "SELECT * FROM tuotteet
WHERE nimi ='" + params[:tuotenimi]
```

Mikäli käyttäjä antaa nimekseen ”; DROP TABLE tuottet;’. Valmis kysely tietokannalle näyttää seuraavalta:

```
sql = "SELECT * FROM tuotteet
WHERE nimi ='’; DROP TABLE tuotteet
```

Kyseinen kysely ensin etsii kaikki tuotteet joiden nimi on tyhjä. Seuraavaksi suoritetaan komento "DROP TABLE tuotteet", joka poistaa koko tuotteet taulun, mikäli sovelluksen tietokantatunnuksilla on siihen oikeudet.

"DROP TABLE tuotteet"tilalla olisi voinut olla mikä tahansa muukin SQL käsky. Esimerkiksi kaikkien tuotteiden listaamiseen hyökkääjä olisi voinut käyttää tuotenimeä joka sisältää jonkin tautologian esimerkiksi "; OR 1=!".

3.2 SQL-Injektion alatyypit

Artikkelin "SQL-Injection is still alive"mukaan SQL-injektioita on kolmea eri päätyyppiä[4].

Inband injektio
Kun SQL-injektion tuloste saadaan samaa reittiä kun se on syötetty, on kyseessä inband injektio. Esimerkiksi jos sovelluksessa on mahdollista hakea lista tuotteista jotka ovat maasta jonka käyttäjä antaa kyselyssä.

Out-of-band injektio
Kun SQL-Injektion tuloste saadaan eri reittiä kun se on syötetty, on kyseesä out-of-band injektio. Web-sovellus saattaa esimerkiksi tallettaa tietokantaansa millä selaimilla sitä on käytetty. Selaintiedot haetaan HTTP pyynnön "User-Agent"kentästä. Hyökkääjä voi asettaa SQL-injektion User-Agent kenttäänsä. Todennäköisesti hyökkääjä ei näe kyselyn tulosta kyselyn palauttamalla sivulla. Hyökkääjän voi ohjata tulokset itselleen esimerkiksi suorituttamalla injektiossa haun:

```
utl_http('http://www.hyokkaajansivu.fi/
    injections/' ||
SELECT password
FROM User
WHERE username = 'admin'
)
```

Injektion onnistuessa hyökkääjän palvelimen logeissa näkyy esimerkiksi:

```
GET "/injections/admininpassword", 200
```

Tällöin hyökkääjä saa selville käyttäjän admin salasanan.

Sokea injektio
Sokea injektio: Hyökkääjä ei saa minkäänlaista palautetta sovellukselta. Hyökkääjä voi kuitenkin kokeilla muokata sovelluksen tietoja ja tarkastella vaikuttaako se sovellukseen. Hyökkääjä voi käyttää apunaan sitä kuinka nopeasti sivu latautuu. Lisäämällä injektoituun sql-käskyyn komennon "waitfor delay 0:0:5", tietokanta odottaa 5 sekuntia ennen kuin

se palauttaa tuloksen. Tästä voidaan päätellä injektion onnistuneen. [6]

4 SQL-Injektiolta suojautuminen

SQL-Injektiolta suojautumiseen on kehitetty useita keinoja. Keinot voidaan jakaa kolmeen päätyyppiin: Hyvät ohjelmointikäytänteet, SQL-injektion havaitseminen ja SQL-injektion ajonaikainen ehkäisy[5].

4.1 Hyvät ohjelmointikäntänteet

Parametrisoidut kyselyt

Parametrisoiduissa kyselyssä luodaan SQL-kyselystä pohja, johon lisätään paikanpitäjät *(engl. placeholder)*. Paikanpitäjät korvataan myöhemmin varsinaisilla arvoilla. Parametrisoitu kysely annetaan tietokannalle. Tietokanta kääntää ja optimoi kyselyn pohjan vain kerran. Tietokanta ei kuitenkaan vielä suorita varsinaista kyselyä, koska varsinaiset arvot puuttuvat. Tällainen toimintatapa parantaa tietoturvan lisäksi myös suorituskykyä.

Parametrisoitujen kyselyiden avulla erotetaan kysely ja siihen liittyvä data. Kun kysely on valmiiksi käännettynä, varsinaisia arvoja ei enää käännetä SQL:läksi. Tästä syystä on mahdotonta, että hyökkääjä voisi suorittaa omia SQL-käskyään syötteensä avulla. Jos hyökkääjä esimerkiksi asettaa käyttäjänimekseen ”OR 1=1’", tietokannasta haetaan käyttäjää jonka käyttäjänimi on ”OR1=1".

Parametrisoidut kyselyt ovat tuettuina lähes kaikissa yleisimmissä ohjelmointikielissä.

Korvaaminen

Korvaaminen *engl. escaping* on toimenpide jossa käyttäjän syötteestä parsitaan vaaralliset merkit pois. Esimerkiksi ’ merkit voidaan muuttaa \merkeiksi. Syötteen korvaamisissa on kuitenkin tietokantakohtaisia eroja. Siksi kullekkin tietokannalle on olemassa omat korvaamisfunktionsa. Esimerkiksi PHP:ssa käytetään mySQL:lää varten "my_sql_real_ecape()"funktiota.

Datan validointi

Datan validointi ei takaa suojaa SQL-injektiolta, mutta se tekee hyökkäyksestä vaikeampaa. Esimerkiksi jos kyseessä on puhelinnumerokenttä, voidaan tarkistaa että syötteessä on vain numeroita. Käytössä voi olla myös luotettujen lista *(engl. whitelist)*, jonne on listattu kaikki syötteelle sallitut arvot.

4.2 Penetraatiotestaus

Ei voida luottaa että edellä esiteltyjä meneltemiä muistettaisiin aina käyttää. Tämän takia sovellusta on tietoturvatestattava. Penetraatiotestaus tarjoaa automatisoidun tavan etsiä sovelluksesta tietoturva-aukkoja, ilman että ohjelmoijan tarvitsee käsin tehdä lukuisa testajä jokaisen koodimuutoksen jälkeen. Penetraatiotestaus voidaan jakaa valkolaatikko- *(engl. white-box testing)* ja musta laatikkotestaukseen *(engl. black-box testing)*. Musta laatikko testauksessa ei nähdä itse ohjelman koodia. Ohjelmalle annetaan erilaisia syötteitä ja tutkitaan tulostetta. Tulosteesta päätellään tässä tapauksessa onko SQL-injektio tapahtunut vai ei. Valkolaatikko testaukseen

4.2.1 Musta laatikko penetraatiotestaus

Haixia edottaa artikkelissaan "A database security testing scheme of web application"[2] seuraavanlaista testausmallia.

Ensiksi etsitään kaikki mahdolliset paikat sovelluksesta, joista käyttäjä voi syöttää dataa. Tämä onnistuu leveyssuuntaista hakua (*engl. Breadth-first search*) käyttämällä. Algoritmi toimii seuraavasti:

1. Alustetaan lista L jossa on ainoana jäsenenä etusivun URL. Etusivu merkataan käsittelemättökäks.

2. Käydään listalta L läpi kaikki käsittelemättömiksi merkatut sivut ja merkataan ne käsitellyiksi. Jokaiselta sivulta kirjataan kaikki paikat joista käyttäjä voi syöttää dataa. Lopuksi etsitään sivulta kaikki linkit. Ne linkit jotka eivät vielä ole listalla L, lisätään listalle.

3. Mikäli listalla on käsittelemättömiä linkkejä, palataan vaiheeseen 2.

Tämän jälkeen luodaan mahdollisimman kattava lista erilaisista haitallisista syötteistä. Kaikkiin mahdollisiin paikkoihin joista voi syöttää dataa sisään kokeillaan kaikkia haitallisia syötteitä. Tietokannan palauttamasta arvosta voidaan päätellä onko injektio onnistunut vai ei. Esimerkiksi jos vastauksen HTTP statuskoodi on 200, kyseessä on haavoittuvuus.

4.2.2 Valkolaatikko penetraatiotestaus

- static code analyzers

4.3 AMNESIA Menetelmä

Halfonding ja Orson artikkelissa [3] esitellään SQL-injektioiden torjumiseksi AMNESIA tekniikkaa. AMNESIA on lyhenne sanoille "Analysis and Monitoring for NEutralizing SQL-Injection Attacks". Tekniikka koostuu neljästä osasta.

1. Etsi suorituspaikat

   Ensin ohjelman koodi skannataan. Skannauksessa etsitään koodista ne paikat, joissa tietokantakyselyjä suoritetaan. Näihin paikkoihin viitataan tässä tutkielmassa sanalla "suorituspaikka"*(engl. hotspot)*. Esimerkiksi Javan tapauksessa etsitään koodista paikat joissa kutsutaan "java.sql.Statement.execute(String)" metodia.

2. Rakenna sql query mallit

   Seuraavaksi rakennetaan jokaiselle edellisessä kohdassa löydetylle suorituspaikalle oma mallinsa. Tämä onnistuu siten, että AMNESIA simuloi sovelluksen toimintaa Java String Analysis (JSA) kirjaston avulla. JSA Luo analyysin tuloksena epädetermistisen ääreellisen automaatin (NDFA), joka tunnistaa kaikki mahdolliset merkkijonot, jotka kysely voi saada arvokseen. Esimerkiksi allaoleva koodipätkä voi saada arvokseen joko: "SELECT info FROM kayttajat WHERE kayttajanimi = β" tai "SELECT info FROM kayttajat WHERE kayttajanimi='vieras'". Käyttäjän syötettä merkataan symbolilla β.

```
query = "SELECT info FROM users WHERE"
if (!kayttajanimi.empty) {
  query += "kayttajanimi ='" +
     kayttajanimi + "'"
} else {
  query += "kayttajanimi = vieras"
}
```

3. Instrument application

   Seuraavaksi lisätään jokaiseen vaiheessa 1. löydettyyn suorituspaikkaan monitori. Monitori suoritetaan aina ennen itse tietokantakyselyä. Monitori ottaa parametriksi suorituspaikan uniikin ID:n ja merkkijonon jota ollaan suorittamassa. ID:n avulla monitori etsii kyseistä suorituspaikkaa vastaavan mallin. Alla sama koodi esimerkkinä:

```
if (monitor.hyvaksyy(<suortuspaikan id>,
   kysely)) {
```

```
        return db.suorita(kysely);
    }
```

4. Ajonaikainen monitorointi

   Ajonaikana ohjelma toimii normalisti kunnes se törmää suorituspaikkaan. Suorituspaikkaan törmättyään se antaa tarvittavat parametrit monitorille. Ensin monitori käsittelee kyselyn samalla tapaa kuin tietokanta sen käsittelisi. Tämän asiosta esimerkiksi erikoismerkit evaluoituvat niiden oikeaan arvoonsa. Tämä estää SQL-avainsanojen piilottamisen erikoismerkeillä. Kun kysely on käsitelty, tarkastetaan tunnistaako malli sen. Mikäli malli hyväksyy kyselyn se suoritetaan, muulloin malli tunnistaa sen SQL-injektioksi.

   Oletetaan että kyselymme olisi "SELECT info FROM users WHERE kayttajanimi=" OR 1=1. Vaiheessa 1. kuvatun koodipätkän automaatti jakautuisi kahtia. Koska automaatti tunnistaa vain kielet jotka loppuvat merkkiin """, kyseinen kysely huomataan SQL-injektioksi.

   [tähän äskösen automaattikuva]

5 Riskin minimointi

6 Yhteenveto

- yksinkertaista suojauta - kuitenkin muitakin riskejä kuten xss yms. -

Lähteet

[1] Anley, C.: *Advanced SQL Injection In SQL Server Applications.* Teoksessa *Next Generation Security Software Ltd. White Paper, 2002.*

[2] Haixia, Yang ja Zhihong, Nan: *A database security testing scheme of web application.* Teoksessa *Computer Science Education, 2009. ICCSE '09. 4th International Conference on*, sivut 953–955, July 2009.

[3] Halfond, William G.J. ja Orso, Alessandro: *AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks.* Teoksessa *Georgia Institute Of Technology.*

[4] Sadeghian, A., Zamani, M. ja Ibrahim, S.: *SQL Injection Is Still Alive: A Study on SQL Injection Signature Evasion Techniques.* Sept 2013.

[5] Sadeghian, A., Zamani, M. ja Manaf, A.A.: *A Taxonomy of SQL Injection Detection and Prevention Techniques.* Sept 2013.

[6] Tajpour, A., Massrum, M. ja Heydari, M.Z.: *Comparison of SQL injection detection and prevention techniques.* Teoksessa *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, nide 5, sivut V5–174–V5–179, June 2010.