Reko Anton Loisti

Injektioiden turvallisuusuhat ja parhaat keinot niitä vastaan

Tietotekniikan kandidaatintutkielma

29. huhtikuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Reko Anton Loisti

Yhteystiedot: rekantlo@student.jyu.fi

Ohjaaja: Tuomo Rossi

Työn nimi: Injektioiden turvallisuusuhat ja parhaat keinot niitä vastaan

Title in English: Security dangers of injections and how to prevent them

Työ: Kandidaatintutkielma

Opintosuunta: Tietoteknikka

Sivumäärä: 24+0

Tiivistelmä: Injektiot ovat olleet uhkana kaikenlaisille sovelluksille jo pitkään ja niiden ehkäisyyn on esitetty todella monia eri keinoja. Vaikka injektioiden ehkäisyä on tutkittu paljon, ovat inkjektiot vieläkin yksi yleisimmistä tietoturvauhkista. Tämän tutkimuksen tarkoituksena on kartoittaa parhaimmat ehkäisykeinot SQL-, XSS-, ja komentorivi-injektiolle ja yrittää löytää niiden ehkäisystä yleisiä teemoja.

Avainsanat: Kandidaatintutkielma, Injektiot, SQL-injektiot, XSS-injektio, Komentorivi-injektio, Ohjelmistoturvallisuus

Abstract: Injections have been a threat to application level security for a long time and there have been many ways presented to stop them. Although injections have been widely studied they are still one of the biggest threats to application security. This study aims to find the best ways to combat SQL-, XSS, and Shell-injections and to find common themes from their prevention.

Keywords: Bachelor's Theses, Injections, SQl-injection, XSS-injection, Shell-injection, Application security

Kuviot

Sisällys

1 Johdanto

Erilaiset injektiot ovat todella vakava uhka sovellustason turvallisuudelle (Pietraszek ja Berghe 2006). Injektiot ovat myös eräitä yleisimmistä tietoturvahyökkyksistä. SQL-injektioden avulla hyökkäjä voi saada käsiinsä arkaluontoista dataa, kuten pankkitunnuksia ja salasanoja (Halfond, Viegas, Orso ym. 2006). XSS-injektiot hyökkäävät suoraan verkkosivujen käyttäjiin mahdollisesti asentaen haittaohjelmia (Shema 2012). Komentorivi-injektio on vastuussa ShellShock-haavoittuvuudesta, joka on yksi vaarallisimmista ja kuuluisimmista tietoturvaheikkouksista (Stasinopoulos, Ntantogian ja Xenakis 2015). Injektioiden vaarallisuuden ja yleisyyden takia onkin mielekästä tuntea yleisimmät injektiot ja niiden ehkäisykeinot. Tutkielma tarjoaa yleiskuvan näistä kolmesta injektiosta ja niiden ehkäisykeinoista. Kontribuutiona on injektioiden ja niiden ehkäisykeinojen yleistäminen. Tarkoituksena on kuvata injektioiden ehkäisykeinoissa toistuvia teemoja, joiden avulla voidaan ymmärtää injektioita paremmin.

Tämän tutkimuksen luvussa kaksi tutustutaan injektion käsitteeseen. Luvussa kaksi esitellään myös kolme yleisintä injektiota ja selvitetään niiden toimintaperiaatteet, vaikutukset ja käyttöympäristöt. Luvussa kolme käsitellään kolmen tutkielmassa esitellyn injektion ehkäisykeinoja ja tutustutaan injektioiden ehkäisyn yleisiin teemoihin. Luku neljä on yhteenveto, jossa tehdään myös tutkielman johtopäätökset.

2 Injektio

Injektiot ovat eräitä yleisimmistä tietoturvahökkäyksistä. Injektioissa käytetään hyväksi jotain ohjelmassa olevaa heikkoutta, jotta ohjelmaan saadaan syötettyä eli injektoitua haitallista koodia. Koodin injektoinnin jälkeen ohjelma suorittaa injektoidun koodin (Hu ym. 2006). Injektioita on monia erilaisia, mutta niiden karkeat toimintaperiaatteet ovat usein samat (Halfond, Viegas, Orso ym. 2006). Injektioiden lopputulos on riippuvainen ohjelmasta, johon hyökätään, mutta yleisesti lopputulos on tietovuoto, oikeuksien korottaminen tai mielivaltaisten komentojen suorittaminen (Pietraszek ja Berghe 2006). Tietovuoto voi olla esimerkiksi tietokannassa olevien tietojen, kuten pankkitunnuksien tai yrityssalaisuuksien vuotaminen. Oikeuksien korottamisella tarkoitetaan esimerkiksi kirjautumista ilman oikeita tunnuksia, jolloin hyökkääjä korottaa oikeuksiensa määrää sisäänkirjautuneen käyttäjän tasolle. Mielivaltaisten komentojen suorittaminen tarkoittaa puolestaan sitä, että hyökkääjä voi suorittaa mitä tahansa komentoja. Suoritettavia komentoja kuitenkin rajoittaa ympäristö, johon hyökkääjä pääsee käsiksi. Injektion kohteen ollessa tietokanta, hyökkääjä voi käyttää vain komentoja, joita tietokanta ymmärtää.

Tässä tutkielmassa käsitellään kolmen tyyppisiä injektioita: SQL-, skripti- ja komentorivi-injektioita. On tärkeää tiedostaa, että nämä kolme injektiota eivät edusta kaikkia mahdollisia injektioita. Tähän tutkimukseen on valittu nämä kolme injektiotyyppiä, sillä ne ovat kolme yleisintä injektiota (Pietraszek ja Berghe 2006). Koska injektioden toimintaperiaatteet ovat suurinpiirtein samoja, näitä kolmea injektiota käsittelemällä saa hyvän yleiskuvan injektioista.

2.1 SQL-injektio

SQL-injektio on injektiotyyppi, jossa ohjelmassa oleva heikkous antaa hyökkääjälle mahdollisuuden muokata ohjelman muodostamaa SQL-kielistä kyselyä. Muokattu SQL-kysely lähetetään tietokannalle, jolloin kysely tekee jotain mitä kehittäjä ei ennakoinut sen tekevän. Vaikka SQL-injektiot mielletään usein verkkosovelluksien uhkaksi, ne ovat uhkana mille tahansa sovellukselle, joka muodostaa dynaamisia SQL-kyselyitä käyttäjän syötteen perusteel-

la (Clarke 2012). SQL-injektiot ovat erityisen vaarallisia, sillä ne kohdistuvat tietokantaan, joka voi sisältää esimerkiksi asiakkaiden salasanoja, pankkitietoja ja henkilötietoja (Halfond, Viegas, Orso ym. 2006). Tätä ajatusta tukien Open Web Appplication Security Project (OWASP) on asettanut SQL-injektion vaarallisimmaksi uhkaksi verkkopohjaisille sovelluksille yli vuosikymmenen ajan (Alenezi, Nadeem ja Asif 2020).

SQL-injektion havainnollistamiseksi tutkitaan yksinkertaista esimerkkiä SQL-kyselystä ja siitä, miten hyökkääjä muodostaa SQL-injektion. Kuviossa 1 on SQL-kysely, joka voisi muodostua, kun käyttäjä yrittää kirjautua sisään johonkin palveluun. Tässä tapauksessa käyttäjä on käyttänyt käyttäjänimeä ”Username” ja salasanaa ”password”. Esitetty SQL-kysely hakee tietokannan ”users” taulusta kaikki käyttäjät, joiden userID on ”Username” ja password on ”password”.

Kuvio 1. SQL esimerkki mukailtu lähteestä (Alenezi, Nadeem ja Asif 2020).

```
1 SELECT * from users
2 WHERE userID = "Username" and password = "password"
```

Tälläisessä tapauksessa hyökkääjä voisi muodostaa seuraavanlaisen SQL-injektion (ks. kuvio 2). Hyökkääjä muokkaa SQL-kyselyä kommentoimalla salasanan tarkastuksen pois kyselystä käyttäen kommentin aloitusmerkkiä ”/*” ja kommentin lopetusmerkkiä ”*/- -”. Tässä SQL-kyselyssä kysytään tietokannalta, onko ”users” taulussa käyttäjää, jonka userID on ” tai onko 1 = 1. Luonnollisesti yksi on yhtäsuuri kuin yksi, joten kysely menee läpi ja hyökkääjä pääsee kirjautumaan ilman oikeita tunnuksia. Erilaisessa tilanteessa ja erilaisella komennolla hyökkääjä voisi saada tietokannasta tietoa, jonka kehittäjät olettivat olevan saavuttamatonta (Alenezi, Nadeem ja Asif 2020).

Kuvio 2. SQL-injektio esimerkki mukailtu lähteestä (Alenezi, Nadeem ja Asif 2020).

```
1 SELECT * from users
2 WHERE userID = " OR 1 = 1; /* and password = ''*/--''
```

Tämä esimerkki on hyvin yksinkertainen ja SQL-injektiot voivat olla hyvinkin monimutkaisempia. Halfond, Viegas, Orso ym. (2006) jakavat SQL-injektiot moneen eri kategoriaan,

joilla jokaisella on monia variaatiota. Eri injektiotyyppejä käytetään yhdessä tai tarkasti mietityssä järjestyksessä, riippuen hyökkääjän tavoitteista. Yhdistelemällä näitä kategorioita ja niiden variaatiotia hyökkääjät voivat muodostaa pitkiä ja tarkasti harkittuja SQL-injektioita. SQL-injektiot ovat siis harvoin yhtä yksinkertaisia kuin tässä esimerkissä.

2.2 XSS-injektio

Shema (2012) Määrittelee XSS-injektion (engl. Cross Site Scripting) olevan injektiotyyppi, joka muokkaa verkkosivun HTML-lähdettä tai suorittaa mielivaltaista JavaScriptiä verkkosovelluksessa. Näin voi tapahtua, kun verkkosovellus pyytää käyttäjältä jonkin syötteen, kuten hakusanan, sähköpostin tai käyttäjänimen ja näyttää tämän syötteen verkkosivulla. Verkkosovelluksen turvallisuuden ollessa heikko, siihen voidaan kohdistaa XSS-injektio käyttämällä tarkasti muotoiltua syötettä (Shema 2012). Koska XSS-injektio muokkaa verkkosivun rakennetta tai suorittaa JavaScriptiä, se on luonnollinen uhka verkkosovelluksille, sillä JavaScript on verkkosovelluksien yleinen ohjelmointikieli ja se toimii upotettuna verkkosivun lähteeseen. Shema (2012) huomauttaa XSS-injektiota usein kutsuttavan nimellä HTML-injektio. Tämä viittaa siihen, että kyseinen injektio tarvitsee vain HTML:ää onnistuneeseen hyökkäyksessä. Vaikka XSS-injektiot kohdistuvat verkkosovelluksiin, sen hyökkäyskohde ei yleensä ole itse verkkosovellus vaan sen käyttäjät. Verkkosovellus on usein injektiolle kuljetusväline, jonka avulla hyökätään verkkosivun käyttäjiin (Shema 2012).

Kuuluisin esimerkki XSS-injektiosta, tapahtui vuonna 2005 Myspace verkkosivulla. Eräs käyttäjä katsoi ”Samy” nimisen käyttäjän profiilia, jolloin selain suoritti JavaScriptiä, jonka Samy oli injektoinut. Tämänkin käyttäjän profiili päivittyi automaattisesti ilmoittamaan ”Samy is my hero”. 24 tuntia myöhemmin Samylla oli yli miljoona kannattajaa ja Myspacen palvelimet eivät kykeneet enää toimimaan. Samy oli luonut XSS-injektion ja havainnostanut esimerkillisesti XSS-injektion voiman (Shema 2012).

”Samy is my hero”-esimerkissä XSS-injektio ei ollut vaarallinen tai haitallinen palvelun käyttäjille, mutta se oli haitallinen palvelun ylläpitäjille. Tämä ei ole kuitenkaan tyypillistä. XSS-injektioita voidaan käyttää esimerkiksi lataamaan verkkosivun käyttäjien tietokoneille ohjelmia, jotka seuraavat käyttäjän jokaista näppäimen painallusta samalla varastaen

salasanoja ja käyttäjätunnuksia (Shema 2012). On helppo kuvitella vahingon määrä tilanteessa, jossa Samy olisi asentanut miljoonien ihmisten tietokoneelle viruksen tai onnistunut vuotamaan heidän salasanansa.

XSS-injektion havainnollistamiseksi tarkastellaan yksinkertaista esimerkkiä (ks. Kuvio 3). Kuvitellaan tilanne, jossa käyttäjä etsii verkkokaupasta elokuvia jollain hakuehdolla. Esimerkissä käyttäjä käyttää hakusanaa ”Vares”. Käyttäjän hakuehto voi esiintyä useaan otteeseen haun jälkeisen verkkosivun HTML-rakenteessa. Tässä tapauksessa hakuehto esiintyy ”span” elementin sisällä, joka kertoo käyttäjälle millä termillä haku on suoritettu.

Kuvio 3. XSS-esimerkki 1. Mukailtu lähteestä (Shema 2012).

```
1 <div>matches for=''<span id=''Search=''>Vares</span>''</div>
```

XSS-injektiouhka syntyy, jos nettisivu liimaa HTML:n sekaan käyttäjän hakuehdon, välittämättä siitä, mitä se sisältää. Hyökkääjä voisi kirjoittaa hakulaatikkoon vapaamuotoista HTML:ää esimerkiksi sijoittaen sinne ”script”-elementin. ”script”-elementin sisällä voidaan kirjoittaa normaalia JavaScriptiä. JavaScriptin avulla voitaisiin esimerkiksi ladata käyttäjän tietokoneelle haittaohjelmia, avata välilehti vaaralliselle sivulle tai muokata verkkosivun sisältöä huijaten käyttäjää luovuttamaan arkaluontoista tietoa. Kuviossa 4 on esimerkki, jossa hyökkääjä tekee ”script”-elementin sijoituksen. Kuvion 4 tapauksessa verkkosivu näyttäisi ponnahdusikkunan, jossa lukee ”XSS-injektio”.

Kuvio 4. XSS-esimerkki 2. Mukailtu lähteestä (Shema 2012).

```
1 <div>matches for=''<span id=''Search=''>Vares<script>alert(''XSS−injektio'')</script></span>''</div>
```

Sen sijaan, että selain näkisi ”script”-elementin tekstinä, kuten se näkee sanan ”Vares”, selain käsittelee elementin niin kuin se käsittelisi minkä tahansa muun ”script”-elementin. Hyökkääjä voi siis vapaasti lisätä verkkosivulle ”script”-elementtejä ja mitä tahansa muuta haluavaansa. Tässä esimerkissä XSS-injektio luotiin luomalla uusi ”script”-elementti. XSS-injektio voi tapahtua myös monella muulla tavalla, kuten muokkaamalla elementin attribuuttia tai muokkaamalla valmiin ”script”-elementin sisältöä (Shema 2012).

2.3 Komentorivi-injektio

Komentorivi-injektion (engl. Shell injection) tavoitteena on suorittaa komentorivillä jokin järjestelmälle haitallinen komento (Alnabulsi, Islam ja Talukder 2018). Komentorivi-injektion onnistuessa seuraukset vaihtelevat arkaluontoisen datan vuotamisesta mielivaltaisten komentojen suorittamiseen (Stasinopoulos, Ntantogian ja Xenakis 2015). Komentorivi-injektiot ovat mahdollisia ohjelman käyttäessä jotain komentorivikomentoa, liittäen siihen käyttäjän syötettä. Tälläinen komentorivikomentojen käyttö on ongelma, jos käyttäjän syöte liitetään komentoon pohtimatta sen sisältöä (Bravenboer, Dolstra ja Visser 2010). Tälläisessa tilanteessa käyttäjän komennot ovat mielivaltaisia ja injektiouhka on syntynyt. Stasinopoulos, Ntantogian ja Xenakis (2015) kertovat komentorivi-injektiouhkia löytyneen verkkosovelluksista, reitittimistä, tulostimista ja erityisesti IoT-laitteista (engl. Internet of Things). Komentorivi-injektiot ovat hyvin vaarallisia, sillä komentorivikomennoilla on mahdollista vaikuttaa koko järjestelmään. Hyökkääjä voi esimerkiksi poistaa järjestelmälle olennaisia tiedostoja tai luoda järjestelmälle haitallisia tiedostoja.

Tarkastellaan esimerkkiä komentorivi-injektiosta (ks. kuvio 5). Tässä esimerkissä ohjelma kysyy käyttäjältä kansion nimen, jonka sisällön ohjelma tulostaa. Tämä esimerkki on toteutettu C-ohjelmointikielellä (Uitto ym. 2015).

Kuvio 5. Komentorivi-injektio esimerkki. Mukailtu lähteestä (Uitto ym. 2015).

```
1 char command[100] = "ls−l"
2 char *user_input;
3
4 /* Haetaan tiedostonimi ja asetetaan se user_input muuttujan arvoksi*/
5 user_input = ";cat /hakemisto/salasanat";
6
7 strcat(command, user_input); /*Asetetaan tiedostonimi ls −l komennon jatkeeksi*/
8 /* Tulee siis ''ls −l; cat /hakemisto/salasanat'' */
9 system(command); /*Suoritetaan komentorivikomentona*/
```

Esimerkin ohjelma ei ollenkaan tarkista mitä käyttäjä syöttää. Käyttäjä voi siis syöttää täysin mitä haluaa. Hyökkääjän antaessa esimerkiksi syötteen ";cat /hakemisto/salasanat", tulostaisi ohjelma salasana hakemiston tai tiedoston sisällön (Uitto ym. 2015). Edellisen "ls-

l”-komennon oletettu toiminta estetään käyttämällä ”;”-merkkiä, jonka jälkeen pyydetään tulostamaan salasanat tiedoston sisältö. Komentorivillä voidaan tähän tapaan putkittaa, eli ketjuttaa montakin komentoa, jotka suorittavat jatkuvasti monimutkaisempia asioita.

3 Injektioiden ehkäisy

Erilaiset injektiot ovat olleet vaarallinen uhka sovellusturvallisuudelle jo pitkään. Näin ollen onkin luonnollista, että niitä varten on kehitetty useitakin erilaisia ehkäisykeinoja. Esimerkiksi Hu ym. (2006) ja Pietraszek ja Berghe (2006) ovat esittäneet yleisiä keinoja injektioiden estämiseen. Koska erilaisia injektiota ja niiden alalajeja on niin monia erilaisia, ehkäisykeinojen kehittäminen on suuri haaste (Halfond, Viegas, Orso ym. 2006).

Injektioiden ehkäisyn lähtökohtana on usein etsiä yleinen syy tai lähtokohta injektiolle. Sen tarkoituksena on yleistää injektiota, jotta ehkäisevän keinon kehittäminen olisi helpompaa. Hu ym. (2006) toteavat, että injektiouhan olemassaoloon tarvitaan vain kaksi seuraavaa tekijää. Sovelluksen täytyy lähettää komentoja johonkin ulkoiseen sovellukseen käyttäen dataa, johon liimataan muuttujia ja käyttäjän syöte vaikuttaa tämän lähetetyn komennon sisältöön. Pietraszek ja Berghe (2006) toteavat käyttäjän syötteen olevan aina injektion lähtökohtana ja ehdottavat keinoa, jossa kaikki ohjelman ulkopuolelle lähettävä syöte enkryptataan, eli salataan. Käyttäjän syöte on siis injektiouhkan yhteinen tekijä.

Seuraavissa alaluvuissa käsitellään kolmen tutkimuksessa käsiteltyjen injektioiden ehkäisykeinoja ja vertaillaan niitä. Luvun lopussa tutkimme kaikkiin injektioihin yleistettäviä keinoja. Tarkoituksena on tutkia jokaiselle injektiolle esitettyjä keinoja, löytää niistä parhaimpia, antaa hyviä lähtökohtia injektioden ehkäisyyn ja tutkia löytyykö ehkäisystä jotain yleisiä teemoja.

3.1 SQL-injektion ehkäisy

SQL-injektiot ovat olleet olemassa niin pitkään kuin SQL-tietokantoja on liitetty verkkosovelluksiin. SQL-injektioiden julkisuuteeen tuonti tapahtui vuonna 1998, kun Rain Forest Puppy kirjoitti aiheesta artikkelin (Clarke 2012). Vaikka SQL-injektiot ovat olleet uhkana jo kauan, on niiden ehkäisy silti huomattavan alatasoista. Gartner Groupin tutkimus testasi yli 300 verkkosivua ja suurin osa niistä oli alttiita SQL-injektiolle (Halfond, Viegas, Orso ym. 2006).

Alenezi, Nadeem ja Asif (2020) totesivat, että yhtä kaikkea kattavaa tekniikkaa SQL-injektioita vastaan ei ole olemassa. Clarke (2012) tukee tätä ajatusta ja suosittelee monen eri keinon käyttämistä samaan aikaan. Kaikkia tekniikoita ei kuitenkaan ole välttämättä mahdollista käyttää jokaisessa sovelluksessa. Tämän takia on tärkeää valita vaihtoehtoiset tekniikat oikein, jotta saavutetaan vaadittu suojaustaso (Clarke 2012).

SQL-injektioiden syy on yksinkertainen ja laajasti tiedossa: käyttäjän syötteen puutteellinen tarkistus (Halfond, Viegas, Orso ym. 2006). Siihen onkin tarjottu erilaisia ratkisuja. Halfond, Viegas, Orso ym. (2006) jakaa ratkaisut karkeasti kahteen kategoriaan: ohjelmointikäytänteisiin ja tunnistus- ja ehkäisytekniikoihin.

3.1.1 Ohjelmointikäytänteet

Ohjelmointikäytänteitä on useita ja niiden toteutus riippuu täysin kehittäjästä. Toisin sanoen kehittäjän pitää olla tietoinen näistä keinoista ja kehittäjän täytyy myös osata toteuttaa ne oikein. Keinot täytyy muistaa toteuttaa kaikkialla sovelluksen kohdissa, joissa SQL-injektiouhka on todellinen. Ohjelmointikäytänteitä ovat syötteen tyypin tarkistus, tuotteen enkoodaus, oikeantyyppisten syötteiden etsintä ja jokaisen käyttäjän syötteen lähteen tarkistus (Halfond, Viegas, Orso ym. 2006). Myös Clarke (2012) esittelee useita ohjelmointikäytänteitä, joita hän kutsuu kooditason ehkäisykeinoiksi. Näitä käytänteitä ovat datan standardointi, parametrisoidut SQL-kutsut, syötteen tarkistus, tuotteen enkoodaus ja syötteen tyypin tarkistus. Voidaan huomata useitakin samanlaisuuksia näiden kahden listauksen välillä. Halfond, Viegas, Orso ym. (2006) ja Clarke (2012) esittelevät syötteen tyypin tarkistuksen ja tuotteen enkoodauksen tärkeänä käytäntönä. Samanlaisuuksia löytyy myös vahvasta painotuksesta tarkastaa käyttäjän syötteet hyvin. Vaikka lähteet esittelevät samoja käytänteitä, ne esittelevät myös toisistaan eriäviä käytänteitä.

Halfond, Viegas, Orso ym. (2006) huomauttavat, että näiden keinojen systemaattinen ja tarkka käyttö on tehokas ratkaisu SQL-injektioita vastaan, mutta käytännössä nämä keinot perustuvat ihmisiin ja ovat siten virhealttiita. Vaikka ohjelmointikäytänteet ovat virhealtiita, niiden toteuttamista ei kannata unohtaa. Sen sijaan niitä pitäisi yrittää toteuttaa aina kun kehittää sovellusta, jossa SQL-injektio on riski. Ohjelmointikäytänteet nostavat sovelluksen

turvallisuustasoa, vaikka niitä toteuttaessa tapahtuisikin virheitä (Clarke 2012).

3.1.2 Automaattinen ehkäisy

Vaikka paras tapa korjata injektiouhka on korjata injektiouhkan aiheuttama koodi, tämä ei ole aina mahdollista, käytännöllistä tai kustannustehokasta. Automaattiset ehkäisykeinot ovat tälläisissä tilanteissa varteenotettava keino (Clarke 2012). Clarke (2012) tarkentaa, että vaikka koodin korjaaminen olisi mahdollista, automaattisen ehkäisyn keinot voivat silti olla todella hyödyllinen keino ehkäistä SQL-injektioita. Ohjelmassa olevaa injektiouhkaa ei ole välttämättä vielä löydetty tai hyökkääjät hyödyntävät uudenlaista hyökkäystapaa, johon kehittäjät eivät ole osanneet varautua. On myös täysin mahdollista, että kehittäjät ovat epäonnistunet SQL-injektion ehkäisyssä (Clarke 2012).

SQL-injektioiden automaattiseen ehkäisyyn on kehitetty vuosien saatossa monia automaattisia tekniikoita. Tekniikoiden toimintaperiaate ja käyttövaatimukset vaihtelevat laajasti. Tiettyjä tekniikoita käyttäessä on pakollista tehdä suuriakin muutoksia alkuperäiseen ohjelmaan (Halfond, Viegas, Orso ym. 2006). Halfond, Viegas, Orso ym. (2006) esittelevät automaattisen ehkäisyn tekniikoita, joita he kutsuvat tunnistus- ja ehkäisytekniikoiksi. Tunnistustekniikat tunnistavat injektion sen tapahtuessa ja ehkäisytekniikat tunnistavat injektion etukäteen analysoimalla koodia. Tekniikoita vertailtiin selvittäen, miten ne pärjäsivät kolmessa eri kategoriassa: tehokkuus yleisiä hyökkäyskeinoja vastaan, käyttövaatimukset ja yleisimpien kehittäjävirheiden automaattikorjaus. Tunnistustekniikoista ”AMNESIA”, ”SQLCheck” ja ”SQLGUARD” suoriutuivat mainiosti, estäen kaikki paitsi yhdentyyppisen injektion. Ehkäisytekniikoista sen sijaan ”Safe Query Objects” ja ”SQL DOM” pääsivät samalle tasolle. Näistä tekniikoista pienin käyttövaatimus oli AMNNESIAlla. Muut tekniikat vaativat muutoksia alkuperäiseen ohjelmaan ja mahdollisesti ylimääräistä infrastruktuuria, kuten kehittäjien koulutusta (Halfond, Viegas, Orso ym. 2006).

WebSSARI oli ainoa tekniikka, joka tarjosi suojan jokaista testattua hyökkäystyyppiä vastaan. Kyseisellä tekniikalla ei ole myöskään mitään käyttövaatimuksia tai ylimääräisiä infrastruktuurivatimuksia. WebSSARI tunnistaa ”likaisia” syötteitä ennaltamäärättyjen suodattimien pohjalta ja ehdottaa puhdistusfunktiota, jotka voidaan lisätä ohjelmaan automaattisesti.

Tekniikan vaatimuksena on, että jokainen käyttäjän syötteen lähde tunnistetaan etukäteen. Tämä voi olla hyvinkin vaikea tehtävä laajoissa ja modulaarisissa verkkosovelluksissa, jolloin tekniikan käytettävyys myös heikkenee (Halfond, Viegas, Orso ym. 2006).

Myös Clarke (2012) esittelee automaattisia ehkäisytekniikoita SQL-injektioita vastaan, jotka ovat suorituksenaikaisia. Suorituksenaikaiseksi tekniikaksi lasketaan mikä tahansa tekniikka, jota voidaan käyttää ilman ohjelman koodin uudelleenajamista, eli ohjelman uudelleenkäynnistystä (Clarke 2012). Suorituksenaikaisista tekniikoista parhaiten tunnettu on verkkosovelluspalomuuri (engl. Web application firewall, WAF), josta tunnetuin toteutus on ModSecurity. WAF on helposti käyttöönotettava tekniikka, eikä se muokkaa itse verkkosovellusta (Clarke 2012). Clarke (2012) esittelee WAF:n käyttämiä tekniikoita SQL-injektioiden estämiseen. WAF estää SQL-injektioita vertaamalla syötteitä tyypillisimpiin hyökkäyksiin (engl. blacklist) tai oikeanlaisiin syötteisiin (engl. whitelist) ja normalisoiden syötettä. Clarke (2012) korostaa, että WAF tekee muutakin kuin estää SQL-injektioita. Sen avulla voi helposti korjata ohjelmassa olevia SQL-injektiouhkia, tunnistaa hyökkäysyrityksiä ja salata SQL-virheviestejä, jotka voisivat mahdollistaa itse injektion.

Automaattisen ehkäisyn keinot eivät ole kuitenkaan täydellisiä ja niiden käyttämistä kannatta harkita tarkasti. Sovelluksen toimintanopeus tulee heikkenemään riippuen käytetystä keinosta. Keinoa valittaessa kannattaa myös miettiä tarkkaan, onko keinon käyttö sovelluksessa työläämpää verrattuna koodin korjaamiseen. Keinot voivat olla hyvin monimutkaisia ja niiden käyttöönotto hankalaa. Paras tapa saada eniten hyötyä automaaattisisista keinoista on oppia niiden rajoitteet ja arvioida mikä keino on jokaisessa tilanteessa sopivin (Clarke 2012).

3.1.3 Tietokannan turvaaminen

Koska SQL-injektion hyökkäyskohde on usein tietokanta. Kannattaa harkita myös tietokannan turvaamista. Hyökkääjä voi yrittää hyökätä tietokannassa oleviin tietoihin tai hyödyntää tietokantaa hyökätäkseen siihin liittyviin verkkoihin. Tälläisiä verkkoja voi olla esimerkiksi yrityksen sisäinen verkko (Clarke 2012). Clarke (2012) keskittää keinot hyökkääjän toiminnan rajoittamiseksi tietokannan sisällä, joko turvaamalla dataa tai palvelinta. Datan

turvaamisen keinoja ovat käyttäjän kirjaaminen tietokantaan mahdollisimman vähäisillä oikeuksilla, toiminnallisuuksien rajoittaminen ja datan vahva suojaaminen (Clarke 2012). Oikeuksien vähentäminen ja toiminnallisuuksien rajoittaminen estää hyökkääjää pääsemästä käsiksi suurempaan määrää dataa kuin alkuperäisessä hyökkäysessä on mahdollista. Datan vahva suojaus esimerkiksi enkryptaamalla estää hyökkääjän näkemän datan käytön, ilman enkryptoinnin murtamista. Palvelimen suojaamiseen sisältyy komentojen käytön rajoittaminen, salasanojen vahvistaminen ja tietokannan ohjelmiston päivittäminen (Clarke 2012).

3.2 XSS-injektion ehkäisy

XSS-injektiot eroavat muista injektiotyypeistä. XSS-injektiot tarvitsevat verkkosovelluksen, jota injektio käyttää kulkuvälineenä hyökätäkseen verkkosivun käyttäjiin verkkoselaimen läpi. Tämä takia on tärkeää toteuttaa ehkäisykeinoja verkkosovelluksen ja verkkoselaimen puolella (Shema 2012). Shema (2012) huomauttaa verkkoselainten käyttäjien olevan yleisin XSS-injektion hyökkäyskohde, mutta verkkoselaimen suojaus on täysin verkkoselainten tarjoajien varassa. Paras tapa estää XSS-injektio on kuitenkin verkkosovelluksen puolella. HTML:n, JavaSriptin ja maailman eri kielien tukeminen tekee tästä haasteen kokeneimallekkin ohjelmoijalle (Shema 2012). XSS-injektion ehkäisy on usein todella hankalaa, koska verkkoselaimet eivät ole vastuussa turvallisuuudesta. Niiden on pakko sokeasti lukea, käyttää ja näyttää niille annettua dataa, jotta internet toimisi kuten se toimii nyt. (Grossman ym. 2007).

3.2.1 Verkkosovelluksien keinot

Shema (2012) ja Grossman ym. (2007) esittävät keinoja, joita verkkosovelluksien kehittäjät voivat käyttää estääkseen XSS-injektioita. Luonnollisin ratkaisu on tutkia käyttäjän syötteitä ja jollain tavalla suodattaa, puhdistaa tai enkoodata niitä. Käyttäjän syötteen suodatuksella on kuitenkin useita huonoja puolia. Syötteen suodatus on hankalaa, koska kehittäjän täytyy tietää jokainen konteksti, jossa syötettä tullaan käsittelemään ja kehittäjän täytyy tarkasti suodattaa jokainen haitallinen merkki pois. On kuitenkin mahdollista rakentaa syötteen suodatin tilanteelle, jossa syöte näytetään vain yhdessä kontekstissa. Ongelma syntyy, jos tulevaisuudessa halutaankin käyttää syötettä jossain uudessa kontekstissa. Silloin pitäisi ottaa huomion

myös tämä uusi konteksti ja siihen liittyvät erityismerkit (Grossman ym. 2007). Grossman ym. (2007) huomauttaa, että syötteen suodatuksessa on myös hyötynsä. Pienimuotoisissa verkkosovelluksissa, jossa on vain yksi syöte ja yksi ympäristö, suodatus voi olla hyvinkin tehokasta. Syötteen suodattaminen voi olla myös mielekästä, jos sille täytyy tehdä erilainen suodatus. Syöte voidaan haluta tallentaa tietokantaan, mutta sitä ennen se pitäisi kuitenkin suodattaa SQL-injektion takia. Samalla voisi hyvin suodattaa syöte XSS-injektiota vastaan.

Käyttäjän syöte ei kuitenkaan voi aiheuttaa XSS-injektiota, ennen kuin sitä käytetään verkkosivun lähteessä ja verkkoselain käsittelee sen. Voidaan siis suodattaa pelkästään käyttäjän syötteestä muodostettua tulostetta (engl. output). Grossman ym. (2007) toteaa tulosteen suodatuksella olevan muutamia selviä etuja. Tulostetta voidaan suodattaa eri tavoin riippuen tulosteen käyttötarkoituksesta ja kontekstista. Tulosteen suodatuskaan ei ole täydellinen keino. Tulostuksen suodatus pitää suorittaa jokaisessa paikassa, missä kyseistä tulostetta käytetään. Tämä on kuitenkin hyvin helppo tehdä väärin. Tulostuksen suodatuksessa tulee helposti virheitä, jos kehittäjä ei tiedä missä kaikissa konteksteissa tulostetta käytetään (Grossman ym. 2007). Tälläinen virhe voi tapahtua erityisesti suurissa sovelluksissa, joissa kontekseja on kymmeniä. Tällöin kehittäjä ei suodata kyseisen kontekstin suhteen ja syntyy tietoturvauhka. Konteksti voi olla myös kehittäjälle vieras, jonka takia jotkin erikoismerkit jäävät suodattamatta. Tulosteen suodatus voidaan esimerkiksi tehdä enkoodaamalla kontekstiin liittyvät erikoismerkit (Shema 2012). Tämä tarkoittaa esimerkiksi HTML:n kontekstissa elementin aloitusmerkin ”<” ja elementin lopetusmerkin ”/>” enkoodausta.

Shema (2012) esittelee muitakin keinoja XSS-injektion estämiseen. Näitä ovat esimerkiksi verkkosovelluksen merkkikoodauksen standardisointi, sisällön tyypin standardisointi ja JavaScriptin hiekkalaatikot. Standardisointi pyrkii rajoittaa erilaisia merkkejä mitä hyökkääjä voi käyttää, ja estää selainta ymmärtämästä syötteen väärän tyyppisenä. Jos syöte ymmärretään väärän tyyppisenä, se rikkoo aikaisemmin tehdyt ehkäisykeinot. Nämä keinot rajoittavat hyökkääjän mahdollisuuksia. JavaScriptin hiekkalaatikko (engl. sandbox) on keino, jossa tehdään tietystä verkkosovelluksen osasta hiekkalaatikko. Hiekkalaatikosta ei näe muualle sovellukseen, rajoittaen käytössä olevia toimintoja. Tämä estää hyökkääjän mahdollisuuksia XSS-injektion onnistuessa.

3.2.2 Käyttäjän toimenpiteet

XSS-injektion kohdistuessa käyttäjään ja niiden ollesssa uhka normaalien verkkosivujen selaamisen aikana, käyttäjä voi miettiä, millä toimenpiteillä hän voi parantaa omaa turvallisuuttansa. Shema (2012) toteaa käyttäjän hyödyllisten toimenpiteiden olevan vähäisiä, mutta turvallisuutta voi lisätä pitämällä verkkoselaimensa aina uusimmassa versiossa. Grossman ym. (2007) kertovat verkkoselainten turvallisuuden olevan todella heikkoa ja käyttäjän toimenpiteiden vähäisiä. XSS-injektion uhriksi joutuminen on hyvin helppoa, sillä kaikki joutuvat käyttämään verkkoselainta lähes päivittäin. Itseään voi kuitenkin suojata esimerkiksi verkkoselaimen valinnalla, asentamalla hyödyllisiä lisäosia, selaimen toiminnallisuuksien poistamisella, pitkien linkkien varomisella ja vain tunnettujen linkkien klikkaamisella (Grossman ym. 2007).

3.3 Komentorivi-injektion ehkäisy

Komentorivi-injektioiden ehkäisyä ei ole tutkittu niin laajasti kuin XSS -ja SQL-injektioiden ehkäisyä. Sitä varten ei ole myöskään kehitetty monia automaattisia työkaluja ja kehitetyt työkalut ovat puutteellisia (Alnabulsi, Islam ja Talukder 2018). Yksi syy komentorivi-injektion ehkäisyn vähäiseen tutkimukseen voisi olla komentorivikomentojen vähäinen käyttö sovelluksissa. Usein on paljon helpompaa ja fiksumpaa toteuttaa itse tai käyttää valmiiksi tehtyä koodia, joka tekee saman asian mitä haluaa tehdä komentorivikomennon avulla.

Komentorivi-injektion ehkäisyn vähäinen tutkinta on harmillistä, sillä komentorivi-injektio on vastuullinen useastakin tunnetusta tietoturvaheikkoudesta, joista tunnetuin on Shellshock. Shellshock oli löytämishetkellä todella suuri uhka miljoonille laitteille (Stasinopoulos, Ntantogian ja Xenakis 2015). Capobianco ym. (2019) kertovat ShellShockin kohdistuvan bash kuoreen ja sen avulla hyökkääjä voi poistaa, lisätä tai muokata tiedostoja ja suorittaa mielivaltaisia komentoja. Bash kuori on laajasti käytössä Linux ja MacOS käyttöjärjestelmissä. Linux on hyvin suosittu käyttöjärjestelmä varsinkin palvelimissa, eli palvelua isännöivissä tietokoneissa. ShellShockin avulla hyökkääjät pystyivät siis saamaan haltuunsa palvelimia hyökäten joko palvelimeen tai sitä käyttäviin henkilöihin.

Komentorivi-injektio ei ole jäänyt kuitenkaan huomiotta. Stasinopoulos, Ntantogian ja Xe-

nakis (2015) Esittelevät kaksi tärkeää ohjelmointikäytännettä. Ohjelmointikäytänteitä ovat syötteen tarkistus ja syötetystä datasta pakeneminen (engl. escaping input data). Syötteen tarkistus tarkoittaa syötteen tarkkaa käsittelyä, jonka tarkoituksena on poistaa syötteestä kaikki vaaralliset merkit. Pakeneminen tarkoittaa syötteessä olevien vaarallisten merkkien lukemista kirjaimina, eikä järjestelmän ymmärtäminä erikoismerkkeinä (Stasinopoulos, Ntantogian ja Xenakis 2015). Stasinopoulos, Ntantogian ja Xenakis (2015) tarkentavat, että syötteen tarkistus pitäisi toteuttaa sallittujen syötteiden listan ja kiellettyjen syötteiden listan avulla. Pakeneminen pitäisi toteuttaa ohjelmointikielien tarjoamilla työkaluilla. Stasinopoulos, Ntantogian ja Xenakis (2015) kertovat komentorivi-injektioiden automaattisen ehkäisyn tuokalujen olevan vähäisiä tai puutteellisia. Tämän takia Stasinopoulos, Ntantogian ja Xenakis (2015) esittelevät oman automaattisen ehkäisyn metodin nimeltään COMMIX. Myös Alnabulsi, Islam ja Talukder (2018) esittelevät keinon nimeltään GMSA, joka toimii komentorivi-injektioiden ehkäisykeinona. Komentorivi-injektion ehkäisyyn on olemassa myös muitakin automaattisia keinoja, mutta niiden väärien positiivisten tuloksien määrä on todella korkea (Alnabulsi, Islam ja Talukder 2018). Väärä positiivinen tarkoittaa sovelluksen tekemää virhettä, jossa injektiouhka tunnistetaan, vaikka sitä ei ole olemassa.

3.4 Injektioiden ehkäisyn yleisiä teemoja

Vaikka injektiot ovat hyvinkin erilaisia toimintaympristöltään ja hyökkäyskeinoiltaan, niistä voidaan silti löytää paljonkin yhteistä. Injektioiden ehkäisyn yleistäminen on hyvä tapa ymmärtää injektioita ja niiden ehkäisyä paremmin. Aikasemmissa kappaleissa käydyistä ehkäisykeinoista voidaan löytää kolme yleistä teemaa: ohjelmointikäytänteet, automaattiset keinot ja onnistunutta hyökkäystä rajoittavat keinot. Ohjelmointikäytänteitä esiteltiin jokaiselle injektiolle, mutta niissä painotettiin eri asioita. Yhteisiä teemoja olivat tuotteen enkoodaus tai suodatus, syötteen tyypin tarkistus ja käyttäjän syötteen hyvin tarkka käsittely. Automaattisia keinoja olivat injektiotyypistä riippuen esitelty enemmän tai vähemmän, ja niistä löytyi luonnollisia eroja. Tärkeintä on valita keino tarkasti ja tietää mitä se tarkalleen tekee sekä vaatii. Onnistuneen hyökkäyksen rajoittavia keinoja olivat esimerkiksi JavaScriptin hiekkalaatikot ja SQL-tietokannan turvaamiseen liittyvät keinot. Komentorivi-injektion yhteydessä tälläistä toimenpidettä ei mainittu, mutta se on mahdollista toteuttaa. Komentorivin avulla voidan

rajoittaa tiettyihin tiedoistoihin liittyviä oikeuksia ja estää tiettyjen komentojen käyttäminen. Hyökkääjän rajoittamisessa korostui onnistuneen hyökkääjän toimenpiteiden ja näkyvyyden rajoittuminen sekä näkyvillä olevan datan salaaminen.

4 Yhteenveto

Tutkielmassa esiteltiin kolmea yleisintä injektiota, joita ovat SQL-, XSS-, ja komentorivi-injektio, ja tutustuttiin niiden ehkäisykeinoihin. Injektiot ovat hyvin vaarallisia ja suuressa määrässä sovelluksista on injektoihin liittyviä tietoturvauhkia. Injektioiden avulla hyökkääjät voivat esimerkiksi suorittaa mielivaltaisia komentoja tai päästä käsiksi arkaluontoiseen dataan, kuten salasanoihin. Erilaisten injektioiden ehkäisy on monimutkaista ja jokaista injektiota varten on kehitetty niihin erikoistuneita ehkäisykeinoja. Vaikka ehkäisykeinot ovat eri injektioille erilaisia, on löydetty myös yleistettäviä tekjöitä. Injektioiden lähtökohdan todettiin olevan käyttäjän syöte, jota käytetään muodostamaan jotain komentoja. Injektioiden estäminen lähteekin usein liikkeelle syötteen tarkasta käsittelystä. Injektioiden ehkäisy on monitasoista ja vain yhden keinon käyttäminen ei yleensä riitä. Ehkäisykeinot ovat yleisesti automaattisia, kehittäjän toteutuksen varassa olevia ohjelmointikäytänteitä tai onnistuneen hyökkääjän rajoittamiseen liittyviä keinoja.

SQL-injektioiden ehkäisyyn käytettäviä keinoja ovat ohjelmointikäytänteet, automaattisen ehkäisyn keinot ja tietokannan turvaaminen. SQL-injektion ehkäisyssä käytettäviä ohjelmointikäytänteitä ovat syötteen tyypin tarkistus, tuotteen enkoodaus ja käyttäjän syötteen tarkka käsittely. Automaattisen ehkäisyyn sisältyvistä tunnistus- ja ehkäisytekniikoista nousi esille esimerkiksi ”WEBSSARI”, ”AMNESIA”, ”Safe Query Objects” ja ”SQL DOM”. SQL-injektion automaattisen ehkäisyn keinoksi esiteltiin myös verkkosovelluspalomuuri. Tietokannan turvaamisessa korostuivat tietokannassa olevan datan turvaaminen ja palvelimen suojaaminen. XSS-injektioiden ehkäisyyn käytettäviä keinoja ovat verkkosovelluksen keinot ja käyttäjän toimenpiteet. Verkkosovelluksen keinoja ovat syöttteen suodatus, tulosteen suodatus, merkkikoodauksen standardisointi, sisällön tyypin standardisointi ja JavaScriptin hiekkalaatikot. Käyttäjän toimenpiteitä ovat selaimen päivittäminen, hyödyllisten lisäosien asentaminen, selaimen toiminnallisuuksien poistaminen, pitkien linkkien varominen ja vain tunnettujen linkkien klikkaaminen. Komentorivi-injektion ehkäisykeinoja ovat ohjelmointikäytänteet ja automaattiset keinot. Komentorivi-injektion ehkäisyssä käytettäviä ohjelmointikäytänteitä ovat syötteen tarkistus ja syötetystä datasta pakeneminen. Komentorivi-injektio automaattisia ehkäisykeinoja ovat ”COMMIX” ja ”GMSA”.

Tämä tutkielma on toteutettu kirjallisuuskatsauksena, joten mainittuja keinoja ei ole konkreettisesti testattu tutkielman laatimisen aikana. Keinot ovat suurimmaksi osaksi testattu lähteissä, mutta näissäkin testeissä on omia rajoitteita. Tutkimuksessa ei siis voitu konkreettisesti testata keinojen yhteentoimivuutta, niiden vaikutusta tietoturvaan tai onko monta keinoa parempi kuin yksi. Käsitelty kirjallisuus kuitenkin kannustaa useamman keinon käyttämiseen. Tutkimuksessa esitellyt keinot antavat suuntaa injektioiden ehkäisyä varten ja perustuvat aikaisemman kirjallisuuden esitteleviin tai kehittämiin keinoihin. Täten on mahdollista, että keinot eivät toimi yhdessä, ne ovat vanhentuneet tai niiden yhdistelmä ei anna täydellistä turvaa. On myös mahdollista, että hyökkääjät kehittävät uudenlaisia keinoja, joihin tässä tutkimuksessa mainitut keinot eivät toimi.

Lähteet

Alenezi, Mamdouh, Muhammad Nadeem ja Raja Asif. 2020. “SQL Injection Attacks Countermeasures Assessments”. *Indonesian Journal of Electrical Engineering and Computer Science* 21 (lokakuu). https://doi.org/10.11591/ijeecs.v21.i2.pp1121-1131.

Alnabulsi, Hussein, Rafiqul Islam ja Majharul Talukder. 2018. “GMSA: Gathering Multiple Signatures Approach to Defend Against Code Injection Attacks”. *IEEE Access* 6:77829–77840. https://doi.org/10.1109/ACCESS.2018.2884201.

Bravenboer, Martin, Eelco Dolstra ja Eelco Visser. 2010. “Preventing injection attacks with syntax embeddings”. *Science of Computer Programming* 75 (7): 473–495.

Capobianco, Frank, Rahul George, Kaiming Huang, Trent Jaeger, Srikanth Krishnamurthy, Zhiyun Qian, Mathias Payer ja Paul Yu. 2019. “Employing Attack Graphs for Intrusion Detection”. Teoksessa *Proceedings of the New Security Paradigms Workshop,* 16–30. NSPW ’19. San Carlos, Costa Rica: Association for Computing Machinery. ISBN: 9781450376471. https://doi.org/10.1145/3368860.3368862. https://doi.org/10.1145/3368860.3368862.

Clarke, Justin. 2012. *SQL injection attacks and defense.* Elsevier.

Grossman, Jeremiah, Seth Fogie, Robert Hansen, Anton Rager ja Petko D Petkov. 2007. *XSS attacks: cross site scripting exploits and defense.* Syngress.

Halfond, William G, Jeremy Viegas, Alessandro Orso ym. 2006. “A classification of SQL-injection attacks and countermeasures”. Teoksessa *Proceedings of the IEEE international symposium on secure software engineering,* 1:13–15. IEEE.

Hu, Wei, Jason Hiser, Daniel Williams, Adrian Filipi, Jack Davidson, David Evans, John Knight, Anh Nguyen-tuong ja Jonathan Rowanhill. 2006. “Secure and practical defense against code-injection attacks using software dynamic translation”, 2006:2–12. Kesäkuu. https://doi.org/10.1145/1134760.1134764.

Pietraszek, Tadeusz, ja Chris Vanden Berghe. 2006. “Defending Against Injection Attacks Through Context-Sensitive String Evaluation”. Teoksessa *Recent Advances in Intrusion Detection,* toimittanut Alfonso Valdes ja Diego Zamboni, 124–145. Berlin, Heidelberg: Springer Berlin Heidelberg.

Shema, Mike. 2012. *Hacking Web Apps : Detecting and Preventing Web Application Security Problems.* Syngress. ISBN: 9781597499514.

Stasinopoulos, Anastasios, Christoforos Ntantogian ja Christos Xenakis. 2015. “Commix: Detecting and exploiting command injection flaws”. *Dept. Digit. Syst., Univ. Piraeus, Piraeus, Greece, White Paper.*

Uitto, Joni, Sampsa Rauti, Jari-Matti Mäkelä ja Ville Leppänen. 2015. “Preventing malicious attacks by diversifying Linux shell commands.” Teoksessa *SPLST.*