Tampereen yliopisto

Ville Aarnio

INJEKTIOT OSANA VERKKOPALVELUIDEN TIETOTURVAA

Informaatioteknologian ja viestinnän tiedekunta

Kandidaatintyö

Syyskuu 2019

TIIVISTELMÄ

Ville Aarnio: Injektiot osana verkkopalveluiden tietoturvaa
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan kandidaatin tutkinto-ohjelma
Syyskuu 2019

Tämä työ käsittelee injektioita osana verkkopalveluiden tietoturvaa. Työn pääpaino on SQL-injektiossa sekä Cross Site Scriptingissä. Näiden lisäksi työ hieman sivuaa myös muita injektioita.

Injektiot ovat todella oleellinen osa tietoturvaa ja ne voivat olla vaarallinen uhka verkkopalvelun tietoturvalle. Tästä syystä jokaisen verkkopalvelun tulisi olla tietoinen tästä, sekä suojautua sitä vastaan mahdollisimman hyvin. Tässä työssä käsitellään tapoja käyttää injektioita sekä kerrotaan mahdollisia tapoja niiden suojautumiseen.

Työn päätavoitteet ovat kertoa injektioiden toiminnasta ja yhteydestä verkkopalveluihin. Näiden lisäksi tavoite on kertoa injektioilta suojautumisesta. Työssä havaittiin, että injektioilta suojautumiseen ei ole yhtä oikeaa ratkaisua, vaan se on erinäisten keinojen yhdistämistä siten, että käytettävä aika sekä budjetti riittävät toteutukseen kuitenkin siten, ettei tietoturva kärsi.

Tässä työssä käsitellään SQL-injektiota ja Cross Site Scriptingiä. Nämä ovat tunnetuimpia ja yleisimmin käytettyjä injektioita. SQL-injektiossa haitallista koodia syötetään verkkopalvelun osiin, joka on tekemisissä SQL-tietokannan kanssa ja Cross Site Scriptingissä osiin, jotka ovat tekemisissä HTML-koodin kanssa.

Näiltä injektioilta varmin suojautumistapa on manuaalinen syötteiden tarkistus, sillä täydellisesti tehtynä tämä estää injektioiden käytön kokonaan. Täydellinen ei kuitenkaan lähes koskaan ole mahdollinen, joten tapahtuneen hyökkäyksen aiheuttamia haittoja tulee mitätöidä mahdollisimman hyvin.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

SQL = Structured Query Language

XML = Extensible Markup Language

XPath = XML Path Language

URL = Uniform Resource Locators

1. JOHDANTO

The Open Web Application Security Project (OWASP) on tehnyt vuonna 2013 ja 2017 listan kymmenestä yleisimmästä sovellusten tietoturvaongelmasta. Molempina vuosina injektiot ovat olleet listan sijalla yksi, mikä tarkoittaa, että nämä ovat kaikkein tietoturvakriittisin osa verkkopalveluissa. [1]

Työn aiheena ovat injektiot osana tietoturvaa. Injektiot liittyvät vahvasti verkkopalveluihin, kuten internetsivuihin ja tarkemmin niiden tietokantoihin. Injektiota käytettäessä hyökkääjä käyttää hyväkseen verkkopalvelun visuaalisella puolella olevia tekstinsyöttökenttiä. Näiden kenttien avulla hän pyrkii saamaan aikaan tietokannassa tapahtumia, joita hänen ei kuuluisi pystyä tekemään. Tämä voi pahimmassa tapauksessa tarkoittaa esimerkiksi käyttäjätietojen ja salasanojen vuotamista tai tietokannan tietojen tuhoamista.

Työn tutkimusongelma on injektioiden käyttö osana tietoturvahyökkäystä tietokantoihin. Työn pääpaino on SQL-injektioissa sekä Cross Site Scriptingissä ja työn konkreettiset esimerkit keskittyvät näiden käyttöön. Työssä myös esitellään lyhyesti muita mahdollisia injektioita. Työn tulisi vastata kysymyksiin mikä on injektio, mitä eri injektioita on olemassa, miten näitä käytetään, mitä hyökkääjä näistä hyötyy ja miten niiltä suojaudutaan.

Työ alkaa luvusta injektiot, jossa määritellään injektiot yleisesti ja eritellään yleisimpiä injektioita. Tämän jälkeen työssä ovat omat lukunsa SQL-injektioille sekä Cross Site scriptingille, sillä ne ovat työn pääaihe. Luvussa 3 kerrotaan yleisesti SQL:stä sekä SQL-injektioista ja kerrotaan esimerkkien kautta, miten SQL-injektiota voidaan käyttää. Luvussa 4 käsitellään Cross Site scriptingiä, sen yhteyttä HTML-sekä Javascript-koodiin sekä sen käyttöä. Luvussa 5 käsitellään Injektioilta suojautumista erinäisillä metodeilla. Tämä luku käsittelee SQL-injektion sekä Cross Site Scriptingin.

2. INJEKTIOT

Luvussa määritellään injektio yleisellä tasolla, kerrotaan tarkemmin mitä tarkoittaa SQL-injektio, sekä eritellään muita mahdollisia injektioita. Kappaleessa 2.2 kerrotaan myös yleistä tietoa SQL:stä ja sen käytöstä.

2.1 Määritelmä

Injektiohyökkäyksillä tarkoitetaan hyökkäyksiä, joissa hyökkääjä onnistuu injektoimaan haitallista dataa ohjelmaan tai verkkopalveluun. Tällainen data voi olla koodia, jota syötetään verkkopalvelun syötteeseen ja joka suoritetaan verkkopalvelun backend-osassa komentona. Backend-puolella tarkoitetaan verkkosovelluksen tietokantapuolta, joka ei ole suoraan käyttäjälle näkyvissä.

Injektioiden tarkoitus on saada muutoksia ohjelman tai verkkopalvelun toiminnassa. Tällaisia muutoksia voi olla esimerkiksi datan häviäminen, datan varastaminen, palvelun esto tai jopa koko järjestelmän kaappaus. [3]

2.2 Erinäisiä injektioita

SQL-injektio on yksi kuuluisimmista injektioista, mutta ei kuitenkaan ainoa. Tässä työssä käsitellään suurimmaksi osaksi juurikin SQL-injektiota, jossa hyökkäys kohdistuu tietokantaa vasten. Tätä injektiota käsitellään luvussa kolme. Tämä ei kuitenkaan ole ainoa tapa käyttää injektioita ja kaikki injektiot eivät kohdistu tietokantoihin.

Toinen kuuluisa injektio on Cross site scripting (XSS). Tässä hyökkääjä voi syöttää haitallisen scriptin suoraan verkkopalveluun. Hyökkääjä ei siis hyökkää suoraan verkkopalvelun yksittäistä käyttäjää vasten vaan hyökkäys kohdistuu uhrin käyttämään verkkopalveluun. Uhri voi saastunutta verkkopalvelua käyttämällä saada haitallisen scriptin omaan selaimeensa. SQL-injektion tavoin myös tässä tapauksessa verkkopalvelussa täytyy olla jonkinlainen tekstinsyöttökenttä, johon hyökkääjä kirjoittaa haitallisen scriptin. [4] [5]

Tunkeutumisessa (koodi-injektiossa) hyökkääjä voi syöttää haitallista koodia haavoittuvaan ohjelmaan ja tämän avulla saada ohjelman tekemään jotain muuta, kuin mitä sen on tarkoitus tehdä. Tämä voi pahimmassa tapauksessa tarkoittaa sitä, että hyökkääjä pystyy suorittamaan käyttöjärjestelmän komentoja tietokoneen käyttäjän oikeuksilla. Tästä voi seurata koko käyttöjärjestelmän vaarantuminen hyökkääjän haltuun. [3]

XPath-injektio on SQL-injektion kaltainen hyökkäys. Tässä tapauksessa verkkopalvelu ottaa käyttäjän syötteitä ja muodostaa niistä XML dataa XPath queryn avulla. Tätä hyökkääjä voi hyödyntää lähettämällä haitallista informaatiota verkkopalvelulle ja löytää miten XML-data on rakennettu. Tässä hyökkääjällä on myös mahdollista päästä käsiksi dataan, johon hänellä ei ole oikeuksia. [3]

3. SQL-INJEKTIO

Tässä luvussa tarkastellaan SQL-injektiota. Luku alkaa määrittelemällä SQL sekä SQL-injektio. Tämän jälkeen esitellään injektion käyttöä konkreettisten esimerkkien kautta.

3.1 Määritelmä

Moni verkkosovellus tarvitsee tietokannan, johon tallennetaan tietoa. Näitä tietokantoja käsitellään sovelluksen backend-puolella tietokantapalvelulta saadulla toiminnallisuudella. Yksi tällainen tietokantapalvelu on SQL, jonka avulla käyttäjä voi kirjoittaa lauseita muun muassa tiedon tallennukseen, päivitykseen ja käsittelyyn tietokannassa. SQL-tietokantaa käsitellään SQL-lauseilla, joiden avulla tietokannasta voidaan lukea, muokata tai poistaa tietoa. SQL-tietokannat ovat tulleet todella suosituiksi ja tästä syystä ne ovat houkuttelevia kohteita hyökkääjälle. [4]

SQL-tietokanta on niin sanottu relaatiotietokanta. Tällaisessa tietokannassa tallennetut tiedot ovat taulukoitu loogisiksi kokonaisuuksiksi. Tietokannassa voisi olla esimerkiksi taulukko, joka koostuu käyttäjän tiedoista. Sarakkeita voisivat olla käyttäjän etunimi, sukunimi ja ikä. Taulukon riveille tulee siis kaikki tieto yhdestä käyttäjästä. Yksi esimerkki tällaisesta taulukosta tietokannassa voisi olla taulukko, johon tallennetaan käyttäjän kirjautumistiedot. Tässä taulukossa yhdessä sarakkeessa olisi sähköpostiosoite ja toisessa sarakkeessa salasana. Tällainen taulukko voisi olla taulukon 1 mukainen:

Taulukko 1. *SQL-tietokannan users relaatio*

ID	email	password
1	admin@yritys.fi	5EtTTGcA2g
2	teemu.teekkari@yritys.fi	4pwgk8QtcZ
3	maija.mehilainen@yritys.fi	pjLEtLy5CU

Seuraavanlaisella SQL-kyselyllä haetaan taulukosta 1 kaikki kirjautumistiedot käyttäjälle admin@yritys.fi:

```
SELECT * FROM users WHERE email = “admin@yritys.fi”
```

SELECT käsky on siis tietokantakysely, jolla tietokannasta luetaan tietoa. Tämä kysely palauttaa taulukosta koko rivin, jossa sähköpostiosoite vastaa kyselyyn sijoitettua sähköpostia. Palautukseen kuuluu siis ID 1, sähköpostiosoite admin@yritys.fi ja salasana 5EtTTGcA2g. Salasanaa ei tulisi koskaan tallentaa tietokantaan selväkielisenä niin kuin yllä olevassa esimerkissä, vaan se tulisi salata. Tätä käsitellään enemmän luvussa neljä.

SQL-injektio on hyökkäys, joka tapahtuu SQL-tietokantaa vasten. Tällaisessa tapauksessa hyökkääjä pyrkii injektoimaan väärää dataa sivuston tekstikenttään. Kun tällainen väärä data suoritetaan, tapahtuu luvaton pääsy tietokantaan. Tätä hyökkääjä voi käyttää hyväkseen esimerkiksi ohittaakseen autentikoinnin eli käyttäjän sisäänkirjautumisen tai saamaan tietokannan tulostamaan tietoa, mitä sen ei normaalisti tulisi näyttää käyttäjälle. [2]

SQL-injektion käyttöön on olemassa useita tekniikoita. Näistä yleisimmät viisi tekniikkaa ovat yhdistysoperaatio, boolean, Errorin hyödyntäminen, Out-of-band sekä ajan viivästyttäminen. Näitä tekniikoita voidaan myös tarpeen mukaan yhdistellä. [7]

Yhdistystä voidaan käyttää, kun injektio on mahdollista kyselyn SELECT osassa. Tässä kaksi kyselyä yhdistetään yhdeksi siten, että saadaan kaksi tulosta. Toinen näistä voi olla sallittu kysely, minkä verkkosivu tekisi muutenkin. Tämä kysely yhdistetään haitalliseen kyselyyn UNION käskyllä ja mahdollisesti saadaan tietoa taulusta, johon ei tulisi olla oikeuksia. [7]

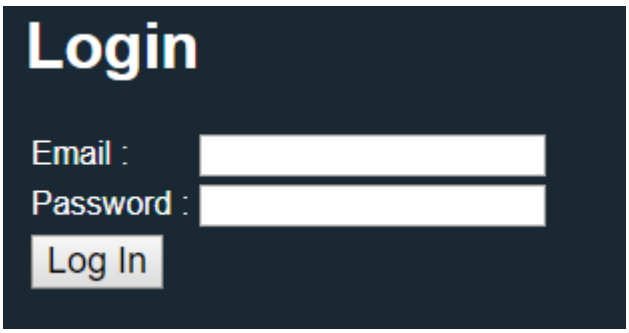
Boolean tekniikalla voidaan käyttää boolean operaattoreita (True tai False) osoittamaan, että jokin tilanne on true tai false. Errorin hyödyntäminen SQL-injektiossa tarkoittaa tekniikkaa, jolla pyritään tietokantaa antamaan errorin, jonka avulla pystytään muokkaamaan SQL-kyselyä sopivaksi. Esimerkiksi jos verkkosivustolla on tekstinsyöttökenttä, joka antaa virheilmoituksen, kun syötetään väärää tekstiä, voidaan pyrkiä siihen, että virheilmoituksen sijasta verkkosivusto antaa error-ilmoituksen, joka tulee SQL tietokannasta. Tässä tilanteessa on mahdollista, että tietokanta on haavoittuvainen ja haavoittuvuus on löydetty. Tämän jälkeen kysely tarvitsee vain muokata toimivaksi. [7]

Out-of-band tekniikalla pyritään suorittamaan SQL-injektio käyttäen eri kanavoita. Näitä kanavoita voi olla esimerkiksi DNS tai HTTP request, jolla SQL-injektion mukainen data syötetään tietokantaan. Ajan viivästyttäminen tarkoittaa sitä, että käytetään tietokantaan komentoja, mitkä viivästyttävät tietokannan tapahtumia. Näitä on esimerkiksi sleep-komento, joka jättää tietokannan odottamaan tietyn hetken. Tällä voidaan saada tietoa tietokannan vastauksesta. [7]

SQL-injektioita käyttäen hyökkääjä voi pahimmassa tilanteessa onnistua sivuuttamaan palvelun autentikoinnin. Jos sivusto käyttää SQL-tietokantoja käyttäjän autentikointiin, niin kaikkien uusien rekisteröityjen käyttäjien kirjautumistiedot ja salasanat tallentuvat SQL-tietokantaan. Täältä hyökkääjä voi saada salasanat onnistuneen SQL-injektion seurauksena. [6]

3.2 SQL-injektion käyttö

Luvussa 3.1 esiteltiin taulukko 1 ja siihen toimiva SQL-kysely. Tässä kyselyssä käsiteltiin tietokannan relaatiota, johon oli tallennettu sähköpostiosoitteita sekä salasanoja. Kyselyssä haettiin tietylle sähköpostiosoitteelle salasanaa käyttäen SELECT- lausetta. Kyselyssä tehty SQL-lause perustui siihen, että kohta email = ”admin@yritys.fi” on tosi, sillä relaatiosta löytyy kyseinen sähköpostiosoite. Tämän ollessa tosi palautettiin koko rivi, jolla sähköpostiosoite oli. Tämä rivi palautettiin kokonaan, sillä kyseisessä kyselyssä SELECT käskyn jälkeen oli symboli ”*”, joka tarkoittaa kaikkea. Tämä kyseinen tilanne voisi näyttää verkkopalvelussa seuraavanlaiselta:



Kuva 1. *Verkkopalvelun kirjautumisnäkymä*

Kuvassa 1 käyttäjä syöttää ”Email” kohtaan sähköpostiosoitteen ja ”Password” kohtaan salasanan. Tämän jälkeen ohjelma vertailee sähköpostiosoitetta ja salasanaa taulukon 1 mukaan. Jos nämä täsmäävät, tapahtuu sisäänkirjautuminen. Tämä voisi näyttää SQL-kyselynä seuraavanlaiselta:

SELECT * FROM Users WHERE Email = ‘$email’ AND Password = ‘$password’

Tällä tavalla voidaan toteuttaa verkkopalveluun kirjautuminen. Jos tähän kyseiseen tilanteeseen verkkopalvelu ei ole tehnyt käyttäjän syötetarkasteluja, voi hyökkääjä syöttää tekstikenttiin erikoismerkkejä ja täten saada aikaan kirjautumistietojen virheellisen käytön. Jos kyseinen kysely palauttaa arvon, tarkoittaa tämä sitä, että käyttäjä on syöttänyt sähköpostiosoitteen ja sitä vastaavan salasanan. Jos kysely ei palauta arvoa, käyttäjällä ei ole pääsyä palveluun. Jos kuitenkin syöttäisimme kirjautumistietoihin seuraavanlaiset tiedot:

$email = 1’ or ’1’ = ’1

$password = 1’ or ’1’ = ’1

kyselyksi tulisi:

SELECT * FROM Users WHERE Email = ‘1’ OR ‘1’ = ‘1’ AND Password = ‘1’ OR ‘1’ = ‘1’

Tästä huomaisimme, että kysely palauttaa arvon, sillä kohdat OR ’1’ = ’1’ ovat aina tosia. Tässä tapauksessa kysely ei tiedä, millä sähköpostiosoitteella hyökkääjä on autentikoinut. Joissain tapauksissa tämä kysely palauttaisi taulukon ensimmäisen käyttäjän, joka taulukossa 1 olisi admin@yritys.fi. Tässä tapauksessa hyökkääjä olisi päässyt kirjautumaan adminina palveluun. [7]

Tämä kyseinen esimerkki injektion käytöstä on melko yksinkertaistettu ja todellisuudessa onnistunut hyökkäys vaatii hyökkääjältä paljon muita toimenpiteitä. Hyökkääjän tulee tietää ennen injektion käyttöä, mitä SQL-järjestelmää verkkopalvelu käyttää, minkä nimisiä relaatioita on käytössä ja mitä relaatioista etsiä. Tämän lisäksi useissa tapauksissa injektio ei toimi näin yksinkertaisesti, sillä se on yleisesti tiedostettu hyökkäys ja sitä vastaan voidaan suojautua. Tähän palataan kappaleessa 4.

4. CROSS SITE SCRIPTING

Tässä luvussa käsitellään Cross Site Scriptingiä. Luku alkaa tämän määritelmällä, tämän jälkeen käsitellään HTML ja JavaScript ja viimeisessä alaluvussa käsitellään Cross Site Scriptingin käyttö.

4.1 Määritelmä

XSS eli Cross Site Scripting on SQL-injektion ohella toinen todella tunnettu hyökkäystekniikka. Se perustuu siihen, että hyökkääjä pystyy piilottamaan scriptejä verkkopalvelun URL:n tai nettisivun taakse. Myös tässä hyökkäystavassa hyökkääjä syöttää verkkopalveluun erikoismerkkejä, joiden avulla hyökkääjä saa verkkopalveluun scriptejä, jotka vaikuttavat javascript-tulkkiin. [4]

SQL-injektio vaikutti verkkopalvelun tekstinsyöttökenttiin ja niiden avulla kohtiin koodissa, joissa verkkopalvelu oli tekemisissä tietokannan kanssa. XSS taas kohdennetaan koodissa HTML-funktioon, joka lähettää dataa selaimelle. Hyökkääjä voi siis onnistuneessa XSS hyökkäyksessä sisällyttää verkkopalvelun tekstinsyöttökenttään <script> tagin, joka tarkoittaa alkanutta scriptiä. Jos verkkopalvelu ei tee tässä kohdassa syötetarkasteluja, voi tällainen scripti vaikuttaa verkkopalvelun javascript-kääntäjään. Onnistunut XSS-hyökkäys voi johtaa käyttäjien kaappaukseen, palvelunestohyökkäykseen tai verkkopalvelun sisällön manipulointiin. [8][9]

4.2 HTML ja JavaScript

Jotta Cross Site Scriptingin käyttö voidaan esimerkkien kautta avata, pitää ensin tietää mitä on HTML ja siinä olevat scriptit. HTML eli hypertext markup language on kuvauskieli, jolla voidaan kuvata tekstiä sekä tekstin rakennetta. HTML tunnetaan varsinkin internetsivujen kirjoituskielenä [9, s. 1]. HTML dokumentti tai internetsivu koostuu HTML-elementeistä. Nämä elementit ovat tunnisteiden sisällä. Jokaisella elementillä tulee olla alkutunniste sekä lopputunniste. HTML-dokumentti kirjoitetaan html tunnisteen sisään. Tällainen tunniste alkaa alkutunnisteella <html> ja loppuu lopputunnisteeseen </html>. Kaikki muut tunnisteet näiden alku- ja lopputunnisteiden sisällä muodostavat HTML-dokumentin. Muita tunnisteita voi olla esimerkiksi <title>, jonka sisään tulee otsikko [9, s. 6].

Cross Site scriptingissä oleellinen HTML-elementti on script. Tämä tulee HTML dokumentissa tunnisteen <script> sisään. Tätä elementtiä käytetään määrittelemään internetsivulla oleva scripti. Nämä scriptit kirjoitetaan HTML-dokumenttiin JavaScriptillä. JavaScriptillä saadaan aikaan internetsivulle esimerkiksi dynaamisia muutoksia, joita ei pelkällä HTML:llä saada aikaan [10].

Kuvassa 2 on yksinkertainen esimerkki HTML-dokumentista, joka sisältää script-elementin.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p>JavaScript can change the content of an HTML element:</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo">This is a demonstration.</p>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
</script>

</body>
</html>
```

Kuva 2. *Esimerkki scriptin käytöstä HTML-koodissa [11]*

Kuvassa 2 ensimmäisellä rivillä oleva <!DOCTYPE html> kertoo internet selaimelle HTML version, jolla sivu on kirjoitettu [12]. Tämän jälkeen tulee varsinaiset HTML tunnisteet. Jotta script voidaan selventää tämän esimerkin avulla, täytyy hieman avata button sekä p elementtejä. Button elementti lisää internetsivulle yksinkertaisen painikkeen, jota käyttäjä voi painaa. Tämän <button> tunnisteen sisällä on ominaisuuksia, joita tälle painikkeelle voidaan määrittää. Ominaisuus type=”button” tulee olla jokaisella painikkeella, jotta painikkeelle ei mene väärää tyyppiä. Seuraavaksi painikkeelle on asetettu ominaisuus onclick. Tämä tarkoittaa painettaessa, eli mitä tapahtuu, kun painiketta painetaan [13]. Tässä esimerkissä se on asetettu myFunction() arvoon. Tämä funktio tullaan määrittelemään script-osiossa. Seuraavaksi on tunniste <p>, joka tarkoittaa tekstikappaletta. Sille on asetettu ominaisuus id, jota tarvitaan myös script-osiossa.

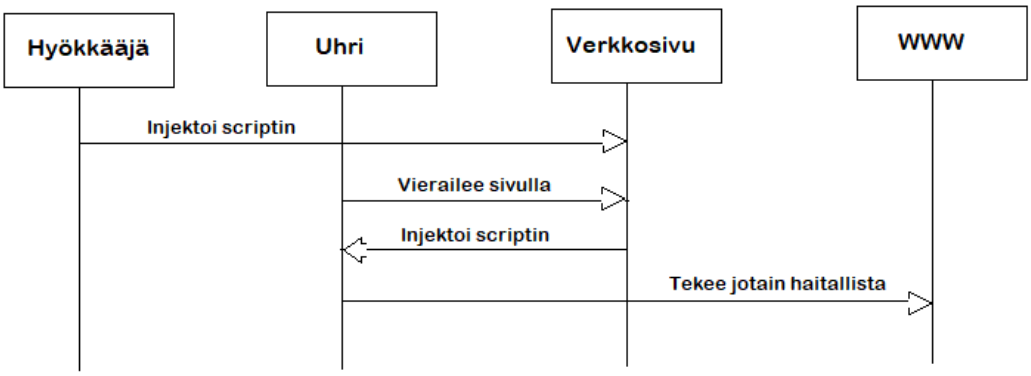
Elementin p jälkeen kuvassa 2 tulee tunniste <script>. Tämän tunnisteen sisälle on kirjoitettu kaksi riviä JavaScript-koodia. Ensimmäinen rivi määrittelee function nimeltään myFunction. Edellä nähtiin tämä funktio button-elementin ominaisuutena, mikä tarkoittaa, että painike käyttää tämän funktion sisältöä sitä painettaessa. Seuraavalla rivillä <script> tunnisteen sisällä on varsinainen JavaScript toteutus. Rivi alkaa document.getElementById(”demo”) käskyllä. Tämä on JavaScriptissä valmiina oleva metodi, joka palauttaa sen elementin, jota sulkujen sisällä oleva id vastaa. Tässä tapauksessa JavaScript ottaa tämän elementin käsittelyyn [14]. Edellä koodissa nähtiin, että <p> tunnisteen ominaisuutena oli juurikin tämä id, eli JavaScript ottaa tämän osan HTML koodia käsittelyyn. Tämän jälkeen tulee kohta innerHTML, joka yksinkertaisuudessaan muuttaa edellä määritellyn elementin arvoa. Tämän jälkeen tulee =-merkki, jonka jälkeen on tekstiä.

Kuvan 2 mukainen verkkosivu toimii siis siten, että sivulla on painike, jonka alapuolella lukee teksti ”This is a demonstration”. Painiketta painamalla teksti muuttuu ”Hello JavaScript!” tekstiksi. Tätä tekstin dynaamista muutosta ei pystytä toteuttamaan ilman <script> tunnistetta ja juurikin tällaisia muutoksia varten se on olemassa HTML koodissa.

4.3 Cross Site Scriptingin käyttö

Jotta XSS hyökkäys verkkosivulle on mahdollinen, täytyy sivulla olla jonkinlainen käyttäjän tekstinsyöttökenttä. SQL-injektion lailla hyökkääjä syöttää tähän kenttään tekstiä, joka tulkitaan koodina uhrin selaimessa. Tällainen teksti on useimmiten pätkä JavaScript koodia, joka on sijoitettu <script> tunnisteiden sisään. Jos verkkosivu ei tee tässä vaiheessa käyttäjän syötetarkasteluja, voi sivu tulkita scriptin osana verkkosivua. Onnistunut XSS hyökkäys siis ajaa käyttäjän syöttämän scriptin verkkosivulla [5].

Kuvassa 3 nähdään yleinen tapa toteuttaa XSS hyökkäys. Ensimmäisessä vaiheessa hyökkääjä edellisen kappaleen mukaisella tavalla syöttää verkkosivulle haitallisen scriptin. Tämän jälkeen uhri käy verkkosivulla. Hyökkääjän asettama scripti injektoi hyökkääjän selaimen, jonka jälkeen tämä scripti tekee jotain haitallista uhrin selaimella. Tämä voi olla esimerkiksi evästeiden kaappausta, jolloin hyökkääjä voi esiintyä uhrina.



Kuva 3. *XSS hyökkäys. Muokattu lähteestä [15]*

Käytännössä siis XSS hyökkäys tarvitsee kolme elementtiä toimiakseen; verkkosivun, uhrin sekä hyökkääjän. Edellisessä kappaleessa mainittiin hyökkäys, jossa hyökkääjä varastaa uhrin evästeet, jotta hän voi esiintyä uhrina. Kuvassa 4 on esitetty yksi tapa toteuttaa kyseinen hyökkäys. Hyökkääjä syöttää tämän scriptin verkkosivulle, josta se kuvan 3 mukaisesti injektoidaan uhrin selaimeen. [5]

```
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
```

Kuva 4. *Evästeiden varastaminen [5]*

Kuvassa 4 olevassa JavaScript koodissa window.location lähettää uhrin evästeet hyökkääjän serverille, joka sijaitsee käskyssä olevassa URL:ssä. [5] Tämä esimerkki on reflected (heijastettu) XSS hyökkäys. [8]

Toinen mahdollinen XSS hyökkäys on stored (tallennettu). Tällainen hyökkäys on mahdollinen tilanteissa, joissa verkkosivu tallentaa käyttäjän syötteet tietokantaan ja esittää ne myöhemmin verkkosivulla. Tällainen voi olla esimerkiksi sähköposti tai verkkoforum. Kun haitallista koodia näytetään verkkosivulla, jossa käyttäjä vierailee, niin kyseinen koodi suoritetaan käyttäjän selaimeen. [8][4]

Stored XSS hyökkäyksestä tunnettu esimerkki on myspace.com verkkosivulla ollut mato. Tämä mato vaikutti miljoonaan käyttäjää vuonna 2005. Tässä tapauksessa Myspace käyttäjä nimeltään ”samy” asetti JavaScript koodia omalle profiilisivulleen. Kun joku muu käyttäjä vierailu Samyn profiilissa, JavaScript koodi suoritettiin ja Samy asetettiin automaattisesti vierailleen käyttäjän kaverilistalle. Tämän jälkeen kyseinen JavaScript koodi kopioitui automaattisesti sivulla vierailleen käyttäjän profiiliin, jonka ansiosta Samy lisättiin myös heidän sivuillaan vierailleiden kaveriksi ja taas koodi kopioitiin heidän profiileihinsa. [16] Vaikka kyseinen tapaus ei aiheuttanut suurta vahinkoa kenellekään, on se silti todella merkittävä tapaus XSS hyökkäysten tietoisuuden sekä suojautumisen kannalta.

5. INJEKTIOILTA SUOJAUTUMINEN

Yleisesti ohjelma tai palvelu on haavoittuva injektioille, jos käyttäjän syöttämää tietoa ei ole tarkastettu erikseen ennen sen syöttämistä. Lähdekoodin arviointi on paras tapa tutkia, onko palvelussa heikkouksia, jotka voivat johtaa injektoiden käyttöön. [1]

5.1 SQL-injektiolta Suojautuminen

Yleisin, varmin sekä suoraviivaisin tapa suojautua SQL-injektiolta on turvallinen ohjelmointi. Tämän lisäksi on olemassa monia erilaisia skannereita, jotka löytävät verkkosivustolta SQL-injektio heikkouksia. [18]

SQL kysely voi olla dynaaminen tai parametrisoitu. Dynaamisessa kyselyssä käytetään käyttäjän syötteitä suoraan muuttamattomana. Tässä tapauksessa kysely siis hyväksyy käyttäjän syötteen sellaisenaan, oli se haitallinen tai ei. Parametrisoidussa kyselyssä käytetään koodissa erikseen määritettyjä parametreja. Näihin parametreihin tallennetaan käyttäjän syöte silloin, kun koodissa esitetyt ehdot täyttyvät. Esimerkiksi jos käyttäjän halutaan syöttävän numeroarvo, tarkistetaan, onko syöte numero ennen kuin se tallennetaan parametriin. Jos käyttäjän syöte on numeroarvo, syötetään se parametriin ja tätä parametriä käytetään SQL-kyselyssä. Tällä tavoin käyttäjän ei ole mahdollista syöttää haitallista syötettä kyselyyn, eikä täten SQL-injektio ole mahdollinen. [17] [18]

Edellisessä kappaleessa käytetty datan tyypin validointi voidaan tehdä kahdella tavalla. Ensimmäisessä tavassa hylätään erikseen määritellyt SQL-injektion näkökulmasta haitalliset merkit kuten ’ ja :. Toisessa tavassa määritellään erikseen kaikki sallitut merkit. Näistä jälkimmäinen on yleisesti parempi, sillä ensimmäisessä tavassa ohjelmoijan tulee huomioida kaikki mahdolliset haitalliset merkit. Tämä voi olla vaikeaa ja työlästä. [18] Turvallisen ohjelmoinnin toteutuksen jälkeen verkkopalvelua tulee testata, ettei jokin heikkous jää huomioimatta. Yleisesti testaajan tulee testata jokainen tekstinsyöttökenttä, joka on tekemisissä SQL-tietokannan kanssa. Tällainen testaus voidaan suorittaa manuaalisesti luvun 3.2 esittämillä periaatteilla tai automaattisesti erinäisillä tähän tarkoitetuilla työkaluilla. [7]

Välillä tulee kuitenkin tilanteita, kun parameterized queryt eivät ole mahdollisia. Tämä voi johtua esimerkiksi käytettävästä SQL- versiosta [20]. Tällaisissa tapauksissa on hyvä käyttää escaping metodia. Tällaisessa tapauksessa SQL-kyselyssä määritellään haitalliset merkit ja jos näitä esiintyy kyselyssä, tämä keskeytetään (escape).

Koska SQL-injektion suojautuminen koodi tasolla on ihmisen tekemää manuaalista syötteiden tarkistusta, on se hyvin riskialtista. Näissä tarkistuksissa on hyvin mahdollista, että jokin merkki jää määrittelemättä ja tästä syystä injektio voi olla mahdollinen. Pelkkä injektiolta suojautuminen ei siis usein riitä, vaan tapahtuneen injektion haittoja tulee mitätöidä.

Tietoturvan näkökulmasta verkkopalvelun käyttäjien salasanat ovat usein kriittisin osa tietokantaa. Tästä syystä näitä salasanoja ei koskaan tulisi säilyttää tietokannassa salaamattomina. Salasana tulisi aina tallentaa tietokantaa salatussa muodossa. Yleisin tapa salata salasanat ovat hash-salaukset. Hash algoritmi muuttaa salasanan vakiopituiseksi merkkijonoksi. Jokainen salasana tuottaa siis saman pituisen hashin, jos siinä käytetään samaa hash-algoritmia. Samasta salasanasta tulee näillä algoritmeilla aina sama hash ja tästä syystä se on hyvä salasanojen suojaamiseen. [22]

Hash-salattua salasanaa ei voi purkaa, vaan ainoa tapa saada siitä alkuperäinen salasana on arvata salasana, tehdä siitä hash ja verrata kahta hashia toisiinsa. Mahdotonta tämä ei kuitenkaan ole, sillä hyökkääjillä on usein käytössään ohjelmia, joiden avulla salasanoja arvataan esimerkiksi sanakirjojen avulla, muutetaan hasheiksi ja verrataan alkuperäiseen salasanaan todella nopeasti. [22]

Salasanan salauksesta saa tehokkaamman lisäämällä siihen niin sanotun suolan. Suola on pätkä tekstiä, joka on satunnaisesti generoitu erillisellä algoritmilla. Tämä teksti lisätään selkokielisen salasanan perään, jonka jälkeen tästä syntynyt merkkijono hashataan. Jos hyökkääjä saa käsiinsä tämän hashin ja onnistuu arvaamaan merkkijonon sen takana, hän ei saa käyttäjän oikeaa salasanaan vaan salasanan ja suolan yhdistelmän. Suola tulee kuitenkin olla eri jokaiselle käyttäjälle, sillä hyökkääjä voi saada haltuunsa monta salasanaa kerrallaan ja onnistua arvaamaan usean näistä. Jos tässä tilanteessa suola on sama useammassa salasanassa, on hyökkääjän helppo keksiä se ja näin saada varsinaiset salasanat haltuun.

5.2 XSS-injektiolta suojautuminen

Cross site scriptingin suojautumiseen käytetään hyvin samanlaisia tapoja kuin SQL-injektion suojautumiseen. Paras tapa on turvallinen ohjelmointi ja tämän lisäksi on olemassa valmiita ohjelmia muun muassa XSS-heikkouksien havaitsemiseen sekä testaukseen. [21]

Turvallisessa ohjelmoinnissa korvaaminen ja poistaminen (black list) etsii haitallisia merkkejä ja joko korvaa ne sallituilla merkeillä tai poistaa ne. Rajoittaminen (white list) nimensä mukaisesti rajoittaa syötteet vain sallittuihin syötteisiin. Tässä siis on erikseen määritellyt sallitut merkit, joita verrataan syötteeseen ja muita merkkejä ei sallita. Tämä tapa siis käyttää samanlaista tekniikkaa kuin SQL-injektion suojautumisessa ja myös tässä rajoittaminen on parempi tapa, sillä kaikkien haitallisten merkkien tunnistaminen on työlästä ja virhealtista. Toisaalta tämä tapa voi myös rajoittaa joitakin haluttuja tilanteita, jos kaikkia sallittuja merkkejä ei ole määritelty. [21] Teoriaosuus turvallisesta ohjelmoinnista luvussa 5.3.

Niin kuin SQL-injektion suojautumisessa, myös tässä, pelkkä turvallinen ohjelmointi voi poistaa XSS-heikkouden kokonaan. Tämä on kuitenkin todella työlästä ja virhealtista. Tästä syystä testaus on myös tässä tärkeässä osassa. SQL-injektion testauksessa oli tarpeellista testata vain tekstinsyöttökentät, jotka olivat tekemisissä tietokannan kanssa. XSS testauksessa tulee testata kaikki tekstinsyöttökentät, jotka ovat tekemisissä HTML-koodin kanssa. Näitä voi olla verkkopalvelussa todella paljon ja osa näistä voi olla melko huomaamattomia kenttiä.

Cross site scriptingin testauksella voidaan löytää heikkouksia suoraan verkkopalvelusta. Tämä voidaan tehdä manuaalisesti syöttämällä haitallisia syötteitä tekstinsyöttökenttiin. Tämä ei kuitenkaan yleensä anna realistista kuvaa mahdollisista haavoittuvuuksista, sillä osa tapauksista jää pakosti testaamatta. Yksi tapa aloittaa tällainen testaus olisi tutkia vastaako verkkopalvelun serveri scriptien syöttämiseen HTTP- vastauksella. Jos vastaus tulee, on mahdollista, että haavoittuvuus löytyy. [23]

XSS testausta varten on olemassa myös automaattisia työkaluja. Yksi tällainen työkalu löytyy sivulta pentest-tools.com. Tämä työkalu ensin yrittää tunnistaa kaikki sivut sekä sivujen syötekentät verkkopalvelussa. Tämän jälkeen se injektoi yksinkertaisen merkkijonon ja tarkastelee verkkopalvelun vastausta tähän. Jos työkalu saa oikeanlaisen vastauksen, syöttää se pätkän javascript-koodia samaan syötekenttään. Jos koodi palautetaan samanlaisena vastauksena kuin aiempi merkkijono, sivun ja syötekohtan voidaan olettaa sisältävän heikkouden. [24]

5.3 Turvallinen ohjelmointi

Edellisissä luvuissa turvallinen ohjelmointi nousi tärkeäksi injektioilta suojautumisen tavoiksi. Pelkkä turvallinen ohjelmointi ei kuitenkaan välttämättä riitä injektioilta suojautumiseen ja tätä varten koko ohjelmistokehitysprosessi tulee tehdä turvalliseksi. Tällä tavoin voidaan varmistaa, että tietoturva tulee mukana koko kehitysvaiheessa eikä se jää vain toissijaiseksi asiaksi.

Tätä varten tehty Secure Developement Lifecycle (SDL) on prosessi, joka yhdistetään osaksi ohjelmistokehitystä. Tämä ei ole pelkästään injektioita vasten tehty prosessi vaan tämä kattaa myös muut tietoturvaheikkoudet sekä tietoturvakriittiset osat ohjelmistokehityksessä. Microsoft on luonut oman SDL prosessinsa, jota se käyttää ja jakaa. [25]

Microsoftin SDL rakenne on melko laaja ja se koostuu monesta osasta. Näitä osia ovat muun muassa tietoturvakoulutukset, uhkakuvaukset, tietoturvatestaukset sekä tapahtuneen tietoturvaonnettomuuden raportointi. Nämä ovat tärkeitä kokonaisuudessa ohjelmiston tietoturvaa, mutta myös injektioiden ehkäisyssä. Tietoturvakoulutukset ovat iso osa turvallista ohjelmointia ja pelkästään tällä voidaan saada ohjelmistosta todella turvallisen. Ihmiset kuitenkin tekevät virheitä, minkä takia muun muassa tietoturvatestaus sekä onnettomuuksien raportointi on tärkeä olla olemassa. [25]

Toinen tapa sisällyttää tietoturva osaksi ohjelmistokehitystä on niin sanottu DevSecOps. Tässä käytetään DevOps viitekehystä ja siihen lisätään mukaan tietoturva. DevOps on tapa kehittää ohjelmistoa siten, että ohjelmistokehitys, testaus ja ylläpito pyritään automatisoimaan yhdeksi kokonaisuudeksi. DevSecOps lisää tähän tietoturvan SDL:n tapaan siten, että se on osana koko ohjelmistokehitystä. Tätä ennen tietoturva on saatettu lisätä vasta valmiiseen ohjelmistoon, mikä saattaa aiheuttaa ongelmatilanteita yhteensopivuudessa eikä tämä tapa tuo parasta tietoturvaa. [26]

DevSecOps kehyksessä on tietoturvaosasto, joka tarjoaa ohjelmistokehityksessä työskenteleville henkilöille koulutusta tietoturvasta sekä työkaluja, jolla tietoturvan implementointi ja testaus saataisiin mahdollisimman hyvin automatisoitua. [26]

Injektioilta suojautumisen pääasia on se, että yksittäinen ohjelmoija osaa ja tekee omasta koodistaan turvallisen. Tämä ei kuitenkaan riitä suurissa ohjelmistoprojekteissa, sillä tietoturva ei ole yleensä pääasiassa yksittäisen ohjelmoijan työssä. Turvallisessa ohjelmoinnissa pitää nähdä koko ohjelman turvallisuus alusta lähtien ja sitä pitää monitoroida koko ohjelmiston elinkaaren ajan.

6. YHTEENVETO

Tässä työssä tutkittiin SQL-injektiota sekä Cross Site Scriptingiä. Injektiot ovat tärkeä osa verkkopalvelujen tietoturvaa ja yksi tietoturvakriittisimmistä heikkouksista. Nykyisin injektiot ovat hyvin tiedettyjä tietoturvariskejä ja ne tulee ottaa huomioon verkkopalveluja toteuttaessa. Muita tunnettuja injektioita ovat muun muassa koodi-injektiot ja XPath-injektio.

SQL-injektio on verkkosivun tietokantaan kohdistettu hyökkäys, jolla hyökkääjä pyrkii saamaan tietokannassa muutoksia, joihin hänellä ei tulisi olla oikeuksia. Näitä ovat esimerkiksi tietokannan näkeminen niiltä osin, mihin oikeuksia ei tulisi olla, tiedon muuttaminen sekä poistaminen. SQL-injektiolta suojaudutaan turvallisen ohjelmoinnin sekä tietoturvatestauksen keinoin.

Cross Site Scripting on injektio, jossa hyökkääjä syöttää verkkopalvelun tekstinsyöttökenttään haitallisia scriptejä, jotka selain tulkistee osana sen HTML-koodia ja täten ajaa ne verkkosivulla käyvälle vieraalle. Tällä saadaan aikaiseksi muutoksia uhrin selaimessa, kuten tietojen lähettäminen hyökkääjälle tai esimerkiksi evästeiden varastaminen. Cross Site Scriptingiltä suojaudutaan myös turvallisella ohjelmoinnilla sekä tietoturvatestauksella.

Turvallinen ohjelmointi on injektioiden suojauksessa tärkeää. Parhaiten toteutettuna turvallinen ohjelmointi on osana koko ohjelmistokehityksen elinkaarta alusta loppuun. Jokaisen ohjelmistokehitykseen osallistuvan henkilön tulee tuntea tietoturva osana ohjelmointia ja näitä keinoja tulee toteuttaa jatkuvasti.

Työn tutkimusongelmiin onnistuttiin vastaamaan hyvin, vaikkakin injektioilta suojautuminen on monimutkaista, eikä siihen ole yksiselitteistä vastausta. Työ kuitenkin antoi myös tähän ongelmaan mahdollisia vaihtoehtoja. Tämän voisi viedä eteenpäin tutkimalla konkreettista järjestelmää ja luomalla tälle järjestelmälle suojautumiskeinon injektioita vastaan.

7. LÄHTEET

[1] The OWASP Foundation, OWASP Top 10 - 2017. Saatavissa (viitattu 24.2.2018): https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

[2] G. Weidman, Advanced Computing and Systems for Security, pp 49-64, No Starch Press, 2014. Saatavissa (viitattu 24.2.2018): https://link-springer-com.libproxy.tut.fi/chapter/10.1007%2F978-81-322-2650-5_4

[3] I. Muscat, What are injection attacks? Acunetix 2017. Saatavissa (viitattu 16.3.2018): https://www.acunetix.com/blog/articles/injection-attacks/

[4] B. Nagpal, N, Chauhan, N. Singh, SECSIX: security engine for CSRF, SQL injection and XSS attacks, 2017. Saatavissa (viitattu 19.3.2018): https://link-springer-com.libproxy.tut.fi/article/10.1007%2Fs13198-016-0489-0

[5] Acunetix, Cross-site Scripting (XSS) Attack, 2018. Saatavissa (viitattu 18.3.2018): https://www.acunetix.com/websitesecurity/cross-site-scripting/

[6] Acunetix, SQL Injection (SQLi), 2018. Saatavissa (viitattu 8.4.2018): https://www.acunetix.com/websitesecurity/sql-injection/

[7] The OWASP Foundation, Testing for SQL Injection (OTG-INPVAL-005), Saatavissa (viitattu 19.3.2018): https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)

[8] L. K. Shar, H. B. K. Tang, Institute of Electrical and Electronics Engineers, Defending against Cross-Site Scripting Attacks. Saatavissa (viitattu: 15.4.2018): https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=5999631

[9] I. Yusof, A. K. Pathan, Institute of Electrical and Electronics Engineers, Mitigating Cross-Site Scripting Attacks with a Content Security Policy. Saatavissa (viitattu 15.4.2018) https://ieeexplore-ieee-org.libproxy.tuni.fi/document/7433336

[10] W3Schools, HTML JavaScript. Saatavissa (viitattu 20.4.2018) https://www.w3schools.com/html/html_scripts.asp

[11] W3Schools, Saatavissa (viitattu 20.4.2018) https://www.w3schools.com/html/tryit.asp?filename=tryhtml_script_html

[12] W3Schools, HTML <!DOCTYPE> Declaration. Saatavissa (viitattu 20.4.2018) https://www.w3schools.com/tags/tag_doctype.asp

[13] W3Schools, HTML <button> Tag. Saatavissa (viitattu 20.4.2018) https://www.w3schools.com/tags/tag_button.asp

[14] W3Schools, HTML DOM getElementById() method. Saatavissa (viitattu 20.4.2018) https://www.w3schools.com/jsref/met_document_getelementbyid.asp

[15] ALECU, Felician Oeconomics of Knowledge, Cross Site Scripting (XSS) in Action. Saatavissa (viitattu 22.4.2018) https://search-proquest-com.libproxy.tut.fi/docview/1220444266?pq-origsite=summon

[16] P. Laborge, SecurityFocus, XSS worm hits myspace.com. Saatavissa (viitattu 22.4.2018) https://www.securityfocus.com/brief/18

[17] Infosec, Parameterized SQL Query over Dynamic SQL Query, saatavissa (viitattu 22.4.2018) https://resources.infosecinstitute.com/parameterized-sql-query-dynamic-sql-query/#gref

[18] L. K. Star, H. B. K. Tan, Institute of Electrical and Electronics Engineers, Defeating SQL Injection. Saatavissa (viitattu 23.11.2018) https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=6265060

[19] The OWASP foundation, Cheat Sheet Series Project, Saatavissa (viitattu 23.11.2018) https://github.com/OWASP/CheatSheetSeries

[20] R. Gracia, Microsoft, Dynamic SQL & SQL injection, Saatavissa (viitattu 23.11.2018) https://blogs.msdn.microsoft.com/raulga/2007/01/04/dynamic-sql-sql-injection/

[21] L. K. Shar, H. B. K. Tan, Institute of Electrical and Electronics Engineers, Defending against Cross-site Scripting Attacks. Saatavissa (viitattu 23.11.2018) https://ieeexplore-ieee-org.libproxy.tut.fi/document/5999631/?part=1

[22] S. Gigoyan, MSSQLTips, Storing passwords in a secure way in SQL Server Database. Saatavilla (viitattu 24.11.2018) https://www.mssqltips.com/sqlservertip/4037/storing-passwords-in-a-secure-way-in-a-sql-server-database/

[23] The OWASP foundation, Testing for Cross site scripting. Saatavissa (viitattu 24.11.2018) https://www.owasp.org/index.php/Testing_for_Cross_site_scripting

[24] Pentest-Tools, XSS Scanner. Saatavilla (viitattu 24.11.2018) https://pentest-tools.com/website-vulnerability-scanning/xss-scanner-online

[25] Microsoft, What are the Microsoft SDL practices? Saatavilla (viitattu 2.5.2018) https://www.microsoft.com/en-us/securityengineering/sdl/practices

[26] K. Carter, Institute of Electrical and Electronics Engineers, Francois Raynad on DevSecOps. Saatavilla (viitattu 2.5.2018) https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8048652