Tampereen yliopisto

Ilari Mattsson

TAVALLISIMMAT SYÖTTEIDENKÄSITTELYYN LIITTYVÄT TIETOTURVAVIRHEET

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Joulukuu 2019

TIIVISTELMÄ

Ilari Mattsson: Tavallisimmat syötteidenkäsittelyyn liittyvät tietoturvavirheet (Common information security mistakes in input handling)
Kandidaatin tutkinto
Tampereen yliopisto
Tieto- ja sähkötekniikka, TkK
Joulukuu 2019

Tässä kandidaatin tutkielmassa käsitellään yleisimpiä syötteidenkäsittelyyn liittyviä virheitä, niiden aiheuttamia uhkia sekä keinoja, joilla käsiteltyjen virheiden syntymistä voidaan ehkäistä.

Tutkielmassa käsitellään laajempia ja suppeampia syötteidenkäsittelyyn liittyviä virhetyyppejä. Käsiteltäviä laajempia virhetyyppejä ovat puutteellinen syötteiden tarkastus ja puutteellinen syötteiden puhdistus. Tarkemmin käsiteltyjä, suppeampia virhetyyppejä ovat SQL-injektio, cross-site scritping ja puutteellinen ohjelman toiminnan rajoittaminen käsiteltävän muuttujan muistialueelle.

Kustakin käsiteltävästä virhetyypistä on esitetty kuvausta virhetyypin toiminnasta, tietoa yleisimmistä virheen aiheuttamista uhista sekä yleisimpiä virheitä ehkäiseviä käytäntöjä ja apuvälineitä. Virheiden tarkastelussa käytetään apuna lyhyitä koodiesimerkkejä, jotka kuvaavat kunkin virheen kohdalla jotakin esimerkkitapausta haavoittuvasta ohjelmasta.

Avainsanat: Syöte, Tietoturva, SQL-injektio, XSS, Memory buffer

SISÄLLYSLUETTELO

1. JOHDANTO

Erimuotoisia syötteitä on käytetty jo vuosikymmenten ajan tietokoneohjelmien ja -järjestelmien ohjaamiseen. Monissa tapauksissa syötteiden käsittely on kuitenkin jäänyt puutteelliseksi. Syynä siihen on saattanut olla joko liiallinen luottamus käyttäjien tietämykseen tai se, että uhat ovat jääneet tiedostamatta. Tietoverkkojen yleistyessä puutteellisen syötteidenkäsittelyn aiheuttamat haavoittuvuudet ovat tulleet selkeämmin esille, kun lukuisat hyvää tai pahaa aikovat tahot ovat saaneet vaivattoman pääsyn haavoittuviin järjestelmiin internetin välityksellä.

Internetin ja sen välityksellä toimivien palveluiden suosion kasvun myötä on tunnistettu monia usein tavattavia syötteidenkäsittelyyn liittyviä virhetyyppejä. Monet tunnetuista virheistä koskevat myös verkostoitumattomia sovelluksia, mutta niiden vakavuus kasvaa käyttäjien määrän lisääntyessä. Palveluiden kasvun ja monipuolistumisen myötä myös näitä heikkouksia hyödyntävistä hyökkäyksistä saatava hyöty on kasvanut jatkuvasti. Yhä suurempi määrä tietoa liikkuu julkisten tietokanavien välityksellä ja yhä useampi palvelu on yhteydessä internettiin. Samalla useammat tahot kykenevät hyödyntämään palveluiden ja verkostoituneiden järjestelmien tunnettuja heikkouksia. Heikkouksien ja uhkien tunnistamisesta huolimatta samoja virheitä kuitenkin toistetaan ja havaitaan edelleen.

Tässä työssä tarkastellaan tavallisimpia syötteidenkäsittelyyn liittyviä tietoturvavirheitä. Tietoturvaan ja syötteisiin liittyvää taustatietoa käsitellään toisessa luvussa. Kolmannessa luvussa tarkastellaan tavallisimpia virhetyyppejä, niiden seurauksia sekä ehkäisykeinoja ja esitellään virhetyyppejä tarkemmin lyhyiden koodiesimerkkien avulla. Neljännessä ja viimeisessä luvussa kootaan yhteen työn keskeiset havainnot ja johtopäätökset.

2. TIETOTURVA JA SYÖTTEET

Nyky-yhteiskunnassa kasvavaan turvattavan tiedon määrään pyritään vastaamaan kehittämällä uusia turvamekanismeja ja valistamalla ihmisiä tietojen turvallisesta käsittelystä. Teknologian kehittymisestä ja teknologiatietoisuuden leviämisestä huolimatta monet vuosien takaiset ongelmat kuitenkin vaivaavat tietoyhteiskuntaa edelleen.

Tietoturvalle on määritelty useita eri malleja. Yksi tunnetuimmista tällaisista malleista on CIA-malli, jonka keskeisimmät tekijät ovat *luottamuksellisuus*, *eheys* ja *saatavuus (engl. confidentiality, integrity, availability). Luottamuksellisuus* tarkoittaa tietoon käsiksi pääsemisen rajoittamista vain asiaankuuluville tahoille. *Eheydessä* puolestaan vaaditaan, että tieto on aina syötettäessä asianmukaista ja oikeellista. Eheän tiedon on oltava myös saatavilla tietoa säilyttävästä lähteestä todennettavasti muuttumattomassa ja paikkansa pitävässä muodossa. *Saatavuudella* taataan tiedon olevan sitä tarvitsevan tahon saatavissa vaadittuina ajanhetkinä. Hyvän saatavuuden takaamiseksi järjestelmävirheiden vaikutus saatavuuteen on minimoitava ja menetetty tieto on voitava palauttaa riittävän lyhyen ajan kuluessa. [16]

Digitaalisissa tietojärjestelmissä käyttäjän tai toisen ohjelman järjestelmään kohdistama vuorovaikutus määräytyy pääsääntöisesti erilaisten syötteiden perusteella. Syötteet voivat olla mitä tahansa suurista tietojoukoista yksittäisiin nappien painalluksiin. Syötteiden käyttötarkoitus voi vaihdella käyttäjän henkilöllisyyden todentamisesta aina ohjelman toiminnan määrittämiseen. Näin ollen monet muutkin tietoturvaan liittyvistä mekanismeista liittyvät suoraan käyttäjän antamien syötteiden käsittelyyn. [1]

Lukuisat vanhat ja haavoittuvat järjestelmät, ohjelmistot ja ohjelmien lisäkkeet ovat edelleen laajasti käytössä [1]. Syvällisiä ongelmia sisältäviä ohjelmointikieliä, arkkitehtuureja ja ohjelmistokehikkoja käytetään jatkuvasti uuden luontiin. Entistä paremmista suunnittelumenetelmistä huolimatta uudet toteutukset saattavat periä edeltäjiensä heikkouksia ja jäädä alttiiksi vuodoille ja hyökkäyksille. Tilannetta pahentaa edelleen se, että teknologian kehittymisen myötä hyökkäyspinnat lisääntyvät ja hyökkäykset sekä niissä käytettävät työkalut kehittyvät jatkuvasti [1][12][13][17].

Hyvien tietoturvaratkaisujen tarve on suuri, ja sitä kasvattavat myös vuosia jatkuneet tietoverkkojen lisääntyminen ja yleistyminen. Lukuisat yritykset hyödyntävät sisäisessä tietoliikenteessään yksityisiä tietoverkkoja, ja arviolta noin puolet koko maailman kotitalouksista on yhteydessä internetiin [6]. Tietoverkkojen mahdollistama järjestelmien

ja ihmisten välinen viestintä tuo mukanaan myös omat haasteensa. Käsiteltävää tietoa ja pyyntöjä tiedon käsittelyyn voi tulla samanaikaisesti useilta eri tahoilta. Yksityinen järjestelmä voi myös viestiä julkisia kanavia pitkin, jolloin tietoliikenne on saatava pidettyä salassa asiaankuulumattomilta tahoilta samassa verkossa. Pelkkä toiminnan takaaminen ei välttämättä enää riitä, vaan suunnittelussa ja toteutuksessa tulee myös ottaa huomioon tietoturva.

Turvallisten käytäntöjen mukaisesti suunniteltu ja toteutettu ohjelma käsittelee syötteistä saatua tietoa jo ennen varsinaisen operaation suorittamista. Tietoturvan näkökulmasta syötteidenkäsittelyn yleinen tavoite on estää syötteitä vaikuttamasta ohjelman toimintaan odottamattomalla tai ei-toivotulla tavalla. [2] Tämä kattaa sekä tahattomasti tehdyt virheelliset tai puutteelliset syötteet että hyökkäyksissä käytettävät tahallisesti muotoillut haitalliset syötteet.

Tietoturvaa vaarantavista ohjelmointivirheistä on saatavilla runsaasti tietoa tieteellisen kirjallisuuden ja dokumentaation muodossa. Jotkin tahot ovat jopa koonneet tilastotietoja tunnetuista virhetapauksista ja koonneet listoja yleisimmistä tai vaarallisimmista ohjelmointivirheistä. Eräs tällainen taho on Common Weakness Enumeration eli CWE. CWEn tarkoitus on ylläpitää yhtenäisesti muotoiltua kirjastoa tunnetuista tavanomaisista ohjelmointivirheistä ja heikkouksista. Se on myös koonnut virheiden vakavuuden ja yleisyyden perusteella listan vaarallisimmista, usein tavattavista ohjelmointivirheistä. Tämä listan vaarallisimpien virheiden joukosta löytyy myös merkittävä määrä syötteidenkäsittelyyn liittyviä virheitä. [1]

3. SYÖTTEIDENKÄSITTELYN VIRHEET

Syötteidenkäsittelyyn liittyy lukuisia uhkia. Jotkin näistä uhista liittyvät yleisiin ja vaarallisiin virheisiin. Ohjelmointivirheiden kannalta puutteelliset toteutukset tarjoavat puolestaan hyökkääjille heikkouksia, joita nämä voivat käyttää hyväkseen. Tällaiset heikkoudet ovat uhkia myös järjestelmien tietoturvalle.

Uhkien lähtökohtana on syötteiden luotettavuus. Turvallisesti suunniteltu ja toteutettu järjestelmä ei luota käyttäjiltä tai ulkoisilta järjestelmiltä saamiinsa syötteisiin sellaisenaan. Kuka tai mikä tahansa syötteitä lähettävä taho voi kohdistaa hyökkäyksen järjestelmää kohtaan. Järjestelmän on osattava käsitellä hyökkäyksissä käytettäviä syötteitä oikein.

3.1 Syötteen tarkastus

Yksinkertaisin heikkouksiin johtava syötteidenkäsittelyn virhe on puutteellinen syötteiden tarkastus (engl. input validation), johon voidaan viitata myös tiedon tarkastuksena (engl. data validation). Käytännössä puutteellinen syötteen tarkastus tarkoittaa sitä, että ohjelma voi hyväksyä sellaisia syötteitä, jotka vaikuttavat sen toimintaan odottamattomilla tai ei-toivotuilla tavoilla. [2]

ESSnet project ValiDat foundation määrittelee tiedon tarkastuksen toimintana, jolla varmistetaan tietyn arvojen yhdistelmän kuuluvuus hyväksyttävien arvojen joukkoon [14]. Ohjelmoinnissa ja syötteidenkäsittelyssä tämä voidaan tarkentaa tarkoittamaan toimintaa, jolla varmistetaan sitä, että saatu syöte vastaa ohjelman odotuksia ja toiminnallisia vaatimuksia. Käsitteen laajuuden vuoksi sen voidaan katsoa kattavan myös monia täsmällisempiä heikkouksia, joista osaa käsitellään tässä työssä myöhemmin [2].

Puutteellinen syötteen tarkastus on yksinkertaisuudestaan huolimatta yleinen virhe. Sen esiintyminen ei riipu käytettävästä ohjelmointikielestä. Tämän lisäksi virhe voi tapahtua aina, kun ohjelman tarvitsee ottaa vastaan tietoa itsensä ulkopuolelta. Mahdollisuuksia syötteen tai tiedon tarkastuksen puutteille on siis lukuisia. Erityisen helposti tällainen virhe voi tapahtua silloin, kun syötteen tarkastuksen oletetaan tapahtuvan ohjelman ulkopuolella tai syötteen lähdettä pidetään luotettavana. [2] Web-ohjelmointi tarjoaa lukuisia esimerkkejä syötteiden tarkastuksesta varsinaisen ohjelman toteutuksen ulkopuolella, sillä sen yhteydessä ohjelman varsinainen toteutus ja käyttäjälle näkyvä verkkosivu ovat usein erillisiä toteutuksia.

Kuten aiemmassa kappaleessa mainittiin, puutteellinen tietojen käsittely on laaja käsite. Tämän takia sen mahdollisten haitallisten seurauksien määrittäminen voi olla hankalaa. Heikkouden hyväksikäytön aiheuttama vahinko riippuu vahvasti kyseessä olevan syötteen käyttötarkoituksesta, ja seuraukset voivat koskea jopa kaikkia kolmea tietoturvallisuuden CIA-mallin osa-aluetta [2].

Saatavuuteen on mahdollista vaikuttaa syöttämällä ohjelmalle käyttökelvotonta tai hankalasti prosessoitavaa tietoa haavoittuvissa suorituksen vaiheissa. Käyttökelvoton syöte voi tällöin kaataa ohjelman tai muuten häiritä ohjelman ajoa. Hankalasti prosessoitavalla syötteellä voidaan sen sijaan saada ohjelma varaamaan suuria määriä resursseja tiedon säilyttämiseen ja prosessointiin. Tällaisissa tapauksissa voidaan puhua myös palvelunestosta (engl. denial of service, DoS). [2]

Luottamuksellisuus voi vaarantua silloin, kun syötteillä ohjataan tietojen lukemista ohjelman ajon aikana. Haavoittuva tilanne voisi olla esimerkiksi sellainen, jossa ohjelma määrittää luettavan tiedon tai tiedoston sijainnin käyttäjän syötteestä. Hyökkääjän olisi tällöin mahdollista saada ohjelma lukemaan tietoja myös sellaisista sijainneista, joihin ohjelman ja sitä käyttävän tahon ei ole tarkoitus päästä käsiksi. Jos hyökkääjä tai käyttäjä kykenee vastaavassa tilanteessa muokkaamaan tai poistamaan tietoja, kaikki kolme CIA-mallin osa-aluetta vaarantuvat. [2]

Ohjelmassa 1 on esitetty hyvin yksinkertainen ja pelkistetty puutteellisen syötteen tarkastuksen esimerkkitapaus C++-ohjelmointikielellä. Kyseessä on kuvitteellisen kauppasovelluksen funktio. Funktiossa käyttäjää pyydetään syöttämään lukumäärä ostoskoria esittävään tietosäiliöön lisättävälle tuotteelle. Käyttäjän syötteen oletetaan olevan kokonaisluku, joten muuttujan tyyppi on alustettu sen mukaisesti. Lukumäärälle on lisäksi asetettu maksimiarvo, jonka mahdollinen ylittyminen on huomioitu esimerkkikoodissa.

```
void tuotemäärä (str tuotenimi, Ostoskori kori) {
   // Ostoskori on tyyppiä std::map oleva säiliö.
   // Koriin lisätään käyttäjän tieto valitusta tuotteesta sekä
   // käyttäjän syöttämästä tuotteiden lukumäärästä.
   // lkm maksimiarvo on 99.
   int lkm;
   cout << ”Syötä tuotteiden lukumäärä (max. 99)”;
   cin >> lkm;
   if (lkm <= 99) {
      lisää_koriin (tuotenimi, lkm, kori);
   }
   return;
}
```

Ohjelma 1. *Yksinkertainen C++-esimerkki puutteellisesta syötteen tarkastuksesta.*

Esimerkissä kuvattu funktio ei rajaa muuttujan negatiivisia arvoja, mikä johtaa moniin eri heikkouksiin ohjelman toiminnassa. Funktion kontekstista voidaan päätellä, että lukumäärän tulee aina olla määritelty ja positiivinen kokonaisluku. Tätä voidaan perustella vaikkapa ostoskorin sisältöjen kokonaisarvon laskennan vaatimuksilla; kokonaisarvoa laskettaessa negatiiviset lukumärät johtaisivat kokonaisarvon laskuun, jolloin myös negatiivisen kokonaisarvon saavuttaminen olisi mahdollista. Tuotteen lukumäärän asettaminen nollaksi puolestaan ilmaisee sen puuttumista ostoskorista, mikä tuottaa ostoskoria käsiteltäessä tarpeetonta työtä. C++-kielisessä ohjelmassa alustamattoman lukumäärämuuttujan jättäminen määrittelemättömäksi voi puolestaan johtaa ohjelman odottamattomaan toimintaan. Alustamattoman kokonaisluvun lukeminen tuottaa ohjelmalle kokonaislukuarvon, joka vastaa muuttujalle varatun muistialueen sisältämää satunnaista tietoa. [10]

Muuttujan rajojen puutteellista tarkastusta voidaan lisäksi hyödyntää hyökkäyksissä syöttämällä ohjelmalle mahdollisimman pitkiä negatiivisia kokonaislukuja. Tällöin ohjelma joutuu käyttämään ostoskorin sisältöjen ja niiden perusteella laskettujen arvojen tallettamiseen ja käsittelyyn suuremman määrän muistia ja prosessointitehoa, tai muuttujille varatut muistialueet voivat ylittyä. [2]

Syötteiden tarkastukselle ei ole olemassa yksityiskohtaisia ehkäisymenetelmiä. Sen sijaan virheiden ehkäisyssä voidaan käyttää apuna yleiskäyttöisempiä menetelmiä. Helpoin ja yleisin keino ehkäistä ja välttää puutteellisesta syötteiden tarkastuksesta aiheutuvia heikkouksia on hyvien ohjelmointikäytäntöjen noudattaminen. Turvallisten ja hyväksi todettujen ohjelmointikäytäntöjen lisäksi syötteiden tarkastuksen tukena on erilaisia ohjelmoinnin apuvälineitä. Turvallisuutta edistäviä ohjelmointikehyksiä (engl. framework) ja -kirjastoja. Kehys tai kirjasto voi tarjota valmiita toimintoja syötteiden tarkistamisen takaamiseksi tai helpottamiseksi. [2] Yksi esimerkki syötteiden tarkastusta helpottavasta ohjelmointikehyksestä on avoimen lähdekoodin OWASP Enterprise Security API [7].

3.2 Syötteen puhdistus

Syötteen puhdistus (engl. input sanitization) eroaa syötteen tarkastuksesta siten, että puhdistuksessa syötettä muokataan ohjelmalle sopivaksi pelkän tarkastuksen sijaan. Syötteidenkäsittelyn yhteydessä ohjelma tarkistaa syötteitä esimerkiksi puuttuvien tai ei-toivottujen erikoismerkkien ja haitallisten elementtien varalta ja muokkaa saamiaan syötteitä tarvittaessa. [11]

Seuraavissa alaluvussa keskitytään kolmeen tavallisimpaan syötteiden puhdistukseen liittyvään virheeseen: XSS eli cross-site scripting , SQL-injektio (engl. SQL injection) ja ohjelman toiminnan rajaus muistialueelle. Virheet eroavat toisistaan merkittävästi, mutta kutakin niistä ehkäistään syötteen puhdistuksen keinoin. [1]

3.2.1 Cross-site scripting

Cross-site scripting on erittäin yleinen web-ohjelmoinnin virhe [1]. Haavoittuva web-sovellus toistaa saamansa syötteen puhdistamattomassa muodossa uudelleen luomallaan verkkosivulla, jolloin myös haitalliset syötteet toistetaan. Myös sivun sisältöä ja ulkoasua muokkaavia, haavoittuvia skriptejä voidaan käyttää hyökkäyksen toteuttamisessa. Hyökkääjä voi hyödyntää heikkoutta esimerkiksi haitallisten HTML tai JavaScript-elementtien toimittamiseen ja suorittamiseen muiden käyttäjien selaimissa. [3][17]

Cross-site scripting käsitteenä voidaan karkeasti jaotella kolmeen alaluokkaan: ei-pysyvät, pysyvät ja DOM-pohjaiset. Ei-pysyvä hyökkäys heijastaa haitallisen sisällön suoraan palvelimelta käyttäjälle. Pysyvä hyökkäys hyödyntää palvelimelle talletettua tietoa. DOM-pohjaiset hyökkäykset toimivat puolestaan paikallisesti kohdehenkilön selaimessa. Myös muita heikkouden tyyppejä on olemassa, mutta edellä mainitut kolme ovat keskeisimmät. [3][17]

Edellä mainitusta alaluokista yleisin on ei-pysyvä eli heijastettu XSS. Siinä haavoittuva web-sovellus toistaa saamansa haitallisen syötteen suoraan takaisin luomalleen verkkosivulle. Pyynnön lähettänyt selain lukee automaattisesti myös haitallisen sisällön, kun palvelimelta saatu sivu avataan. Syötteet lähetetään palvelimelle pyynnön mukana muuttujissa. Hyökkäyksen toteuttaminen vaatii kuitenkin sitä, että uhri itse lähettää pyynnön palvelimelle. Tämä voidaan toteuttaa esimerkiksi jakamalla valmiiksi muotoiltuja ja luotettavalta vaikuttavia linkkejä mahdollisille uhreille. [3][17]

Pysyvän XSSn tapauksessa hyökkääjä voi sen sijaan tallettaa haitallisia syötteitä tietokantaan tai muuhun tietovarastoon. Tällöin ne voidaan lukea ja toistaa myöhempänä ajankohtana. Hyökkäyksen uhreiksi päätyvät tällöin kaikki ne, jotka katselevat hyökkääjän tallettamaa syötettä tahallisesti tai tahattomasti. Hyökkäys voi tapahtua myös silloin, kun palvelimen ylläpidosta vastaava henkilö tarkastelee tietovaraston sisältöjä varsinaisen sivuston ulkopuolella. [3][17]

Document Object Model eli DOM on ohjelmointikielestä ja alustasta riippumaton rajapinta, joka mahdollistaa HTML- ja XML-asiakirjojen sisältöjen paikallisen lukemisen

ja muokkaamisen. DOM kuvaa asiakirjan sisällön olioista koostuvana loogisena puuna. [15]

DOM-pohjainen XSS-hyökkäys poikkeaa muista XSSn tyypeistä merkittävästi. Se ei vaikuta suoraan palvelimen luoman verkkosivun sisältöön. Tämän sijaan se hyödyntää heikkouksia palvelimen tarjoamissa skripteissä, jotka muokkaavat sivun sisältöä ja ulkoasua paikallisesti käyttäjän selaimessa. Hyökkäykseen käytettävää syötettä ei tämän vuoksi tarvitse lähettää palvelimelle, jolloin palvelin ei voi havaita hyökkäystä. [3][17]

Tietoturvan näkökulmasta XSS on vakava heikkous. Tyypillinen hyökkäys pyrkii varastamaan käyttäjien henkilökohtaisia tietoja tai lähettämään pyyntöjä palvelimille käyttäjän nimissä. Hyökkäysten vakavuutta kasvattaa ennestään se, että kohdehenkilöt voivat kokea hyökkäyksissä käytetyt sivustot luotettaviksi. Aiheutuva vahinko ei kuitenkaan ole suoraan yhteydessä haitallisen syötteen toimitustapaan. [3]

Käyttäjien tietoja varastavan XSS-hyökkäyksen vakavuus voi olla tapauksen mukaan erittäin suuri. Hyökkäyksestä koituvan vahingon suuruus riippuu ensisijaisesti selaimen evästeiden sisällöstä. Vaarassa ovat kaikki ne evästeet, joihin hyökkäykseen käytetyllä sivustolla on pääsyoikeus. Mikäli evästeet eivät sisällä mitään tärkeää tietoa, aiheutuneen vahingon voidaan katsoa olevan vähäistä tai mitätöntä. Toisaalta evästeet saattavat sisältää käyttäjätunnuksia ja salasanoja tai muita tunnistavia henkilötietoja. Tällöin seuraukset ovat huomattavasti suuremmat. Tietojen luottamuksellisuus vaarantuu kuitenkin kaikissa tapauksissa. [3]

XSS-hyökkäys voi myös pyrkiä lähettämään pyyntöjä haavoittuvalle palvelimelle tai sen ulkopuolelle. Hyökkäyksessä toimitettu haitallinen koodi voi automaattisesti lähettää pyyntöjä käyttäjän selaimelta. Hyökkääjä saattaa esimerkiksi yrittää aiheuttaa vahinkoa ohjaamalla kohteen automaattisesti toiselle haavoittuvalle sivulle. Hyökkääjä voi yrittää suoritta haavoittuvan palvelun tapahtumia kohteen nimissä, mikäli kohdehenkilö on kirjautunut palveluun omilla tunnuksillaan. [3] Yksinkertaisena esimerkkinä tästä voisi olla viestin julkaiseminen keskustelupalstalle käyttäjän tunnuksilla tai käyttäjän tietojen muokkaaminen. Tällaisen hyökkäyksen vaikutukset riippuvat vahvasti siitä, millaisia pyyntöjä hyökkääjä voi suorittaa haavoittuvalta sivulta käsin. Mahdolliset seuraukset voivat vaikuttaa kohdehenkilön tietojen eheyteen, saatavuuteen ja luottamuksellisuuteen. [3][17]

Vahinko voi kohdistua käyttäjien tietojen ja käyttäjätilien lisäksi päätelaitteisiin. Hyökkääjä voi pyrkiä asentamaan haittaohjelmia laitteille tai suorittamaan

järjestelmäkomentoja huomaamattomasti. Tällaisen vahingon aiheuttaminen ei kuitenkaan aina ole mahdollista. [3]

Ohjelma 2 esittelee pysyvää ja ei-pysyvää cross-site scripting -heikkoutta. Esimerkkiohjelma on kirjoitettu PHP-ohjelmointikielellä. Esimerkissä ohjelma yhdistää yksinkertaiseen PostgreSQL-tietokantaan ja yrittää lisätä siihen uutta tietoa palvelimelle lähetetyn pyynnön tietojen perusteella. Lisäksi ohjelma tulostaa käyttäjälle näkyvälle sivulle tiedon tietokantatapahtuman tilasta.

```
<?php
/* Ohjelma lisää syöteparametrien mukaisen monikon
valtiotietokantaan ja tulostaa käyttäjälle ilmoituksen lisäyksen
tilasta. */
$dbconn = pg_connect(”dbname=valtiodb”);
$valtio = $_GET[’valtio’];
$array = array[
   ’vid’ => $_GET[’id’],
   ’nimi’ => $valtio,
   ’kieli’ => $_GET[’kieli’],
   ’asukasluku’ => $_GET[’asklkm’],
];
$result = pg_insert($dbconn, ‘valtio’, $array);
if ($result) {
   echo ‘<p>Valtion ‘ .$valtio. ’ lisääminen onnistui.</p>’;
} else {
   echo ‘<p>VIRHE: Valtion ‘ .$valtio. ’ lisääminen tietokantaan
epäonnistui.</p>’;
}
pg_close($dbconn);
?>
```

Ohjelma 2. *Yksinkertainen PHP-esimerkki ei-pysyvästä ja pysyvästä XSS-haavoittuvuudesta.*

Ei-pysyvän eli heijastetun XSSn heikkous ilmenee esimerkissä käyttäjälle tulostettavien ilmoitusten yhteydessä. Ilmoitusten on tarkoitus sisältää lisättävän valtion nimi. Valtion nimi tulee suoraan palvelimelle lähetetyn pyynnön sisältämästä muuttujasta. Koska muuttujan sisältöä ei tarkisteta tai puhdisteta, siihen voidaan sisällyttää haitallista koodia. Syötteen sisältö toistetaan suoraan tulostettavalle sivulle.

Pysyvän XSSn heikkous löytyy puolestaan esimerkin tietokantatapahtumasta. Tietokantaan lisättävän monikon tiedot haetaan sellaisenaan palvelimen saamasta pyynnöstä. Ohjelma ei itsessään tarkista tai puhdista tietokantaan lisättävää tietoa. Näin ollen jokainen monikon alkio on haavoittuva. Pysyvä XSS-hyökkäys on mahdollinen, jos tietokanta hyväksyy haitallisen syötteen. Talletettu syöte toistetaan silloin, kun kyseistä tietoa tarkastellaan selaimessa.

Pysyvän cross-site scriptingin haavoittuvuudessa on lisäksi kyse SQL-injektiosta. Syynä tähän on tiedon talletukseen käytettävän tietovaraston tyyppi. Esimerkissä käytettävä tietovarasto on SQL-tietokanta. Haavoittuvuus mahdollistaa pysyvän XSS-hyökkäyksen, sillä se mahdollistaa haitallisen koodin injektoinnin tietokantaan. [4] SQL-injektiota käsitellään tarkemmin seuraavassa alaluvussa.

DOM-pohjaisen XSSn esimerkki ohjelmassa 3 pohjautuu kirjaan Mastering Moder Web Penetration Testing [17]. Esimerkkiohjelma sisältää yksinkertaisen HTML-tiedoston. Tiedosto sisältää lyhyen JavaScript-skriptin, joka muokkaa tiedoston sisältöä paikallisesti DOM-rajapinnan välityksellä.

```
    <!-- Ohjelma tulostaa käyttäjälle tervehdyksen paikallisen
2 syötteen perusteella ja DOM:n avulla. -->
    <html>
4     <head>
        <title>DOM-based XSS</title>
6     </head>
      <body>
8       <script>
          name = location.hash.substring(1);
10        document.write("<b>Hey "+unescaped(name)+"! Nice to meet
        you</b>");
12      </script>
      </body
    </html>
```

Ohjelma 3. *Yksinkertainen esimerkki DOM-pohjaisesta XSS-haavoittuvuudesta. Esimerkki pohjautuu lähteeseen. [17]*

DOM-pohjainen haavoittuvuus toimii lähes ei-pysyvän XSS-haavoittuvuuden tavoin. Molemmissa tapauksissa syöte voidaan toimittaa uhrille osana linkkiä, josta se toistetaan takaisin käyttäjän selaimeen. Merkittävimpänä erona on syötteen toiston tapahtumapaikka. Ei-paikallisessa hyökkäyksessä haavoittuva palvelin toistaa syötteen, kun taas esimerkin kaltaisessa tapauksessa toisto tapahtuu käyttäjän päätelaitteessa. Esimerkkiohjelmassa syötteen luku ja toisto tapahtuu rivien 8 ja 12 välillä sijaitsevassa skriptissä. Skriptin lukema syöte sijaitsee verkkosivun osoitekentässä palvelimelle sivun url:in jälkeen. Paikallinen syöte on erotettu palvelimelle lähetettävästä osasta #-merkillä (engl. hash). Hyökkäys on mahdollinen, sillä skripti sijoittaa saamansa tiedon muuttumattomana HTML-elementtiin ja lisää sen sivun HTML-asiakirjaan. [17]

Cross-site scriptingin yleisyyden ja vakavuuden vuoksi sen ehkäisemiseen on olemassa monia käytäntöjä ja apuvälineitä [1][3][17]. Eräs tällainen käytäntö on saman alkuperän käytäntö SOP (engl. same-origin policy). SOP rajoittaa tiettyyn verkkoalueeseen (engl. domain) kuuluvan sivun suorittamat pyynnöt vain kyseiselle verkkoalueelle. Käytäntö

rajoittaa merkittävästi XSS-haavoittuvuuksista aiheutuvaa vahinkoa, sillä se estää haitallisia skriptejä lähettämästä luvattomia pyyntöjä muille sivustoille. [17]

Hyökkäyksen tehokkuutta voi rajoittaa lisäksi evästeiden kautta. Evästeen luonnin yhteydessä sille voidaan asettaa HttpOnly-ominaisuus. Tällä ominaisuudella varustettua evästettä ei voi tavallisesti lukea käyttäjälaitteessa. Sen sijaan lukemisen on tapahduttava palvelimella, jolle eväste lähetetään. Käytettävän selaimen on erikseen tuettava kyseistä ominaisuutta, jotta sitä voidaan käyttää. HttpOnly-evästeet voivat tehokkaasti pienentää hyökkäyspinta-alaa. Se estää haitallisia skriptejä lukemasta käyttäjän evästetietoja. [3]

XSSn mahdollisuus on syytä huomioida ohjelman suunnittelun ja toteutuksen aikana. Syötteiden tarkastuksen tavoin myös syötteiden puhdistuksen avuksi on saatavilla valmiita ratkaisuja ja toteutuksia. Monet ohjelmointikielten kirjastot sekä ohjelmointikehykset tarjoavat syötteiden puhdistusta edesauttavia menetelmiä. Myös yleiset syötteiden tarkastuksen ja puhdistuksen käytännöt pätevät. Kaikki ohjelman ulkopuolelta tulevat syötteet ovat mahdollisesti epäluotettavia, joten niiden turvallisuus tulee tarkistaa. Samalla on tunnistettava eri syötteiden kontekstit. Cross-site scriptingin tapauksessa silmällä pidettävät kontekstit koskevat toistettavaa, käyttäjiltä saatua tietoa. Syötteen puhtaus tulee varmistaa, jos palvelin tai selain toistaa sen dynaamisesti osana tuotettua verkkosivua. [3]

3.2.2 SQL-injektio

SQL-injektio on tavanomainen SQL-tietokantoja käyttävien sovellusten haavoittuvuus [1]. Siinä haavoittuva sovellus muodostaa tietokantakyselyn puhdistamattoman, ohjelman ulkopuolelta saadun syötteen perusteella. Sen seurauksena hyökkääjä voi vaikuttaa kyselyn sisältöön tarkoitettua enemmän. Ohjelman alkuperäistä tarkoitusta laajemmat kyselyt ovat mahdollisia. Jopa täysin uuden kyselyn suorittaminen onnistuu tapauksen mukaan. [4][12]

SQL-injektiossa hyödynnetään Structured Query Language -tyyppisten tietokantakyselyiden syntaksia eli rakennetta. Kyselyillä on selkeä ja johdonmukainen rakenne. Kyselyn muodostuksessa käytetään myös tiettyjä erikoismerkkejä. Erikoismerkkejä käytetään muun muassa arvojen vertailussa, funktioparametrien rajauksessa ja vakioiden määrittämisessä. [9]

Tietokantasovellus odottaa useimmiten käyttäjältä yksinkertaisia syötteitä, kuten tietokannan taulukoiden ja sarakkeiden nimiä tai haettavan tiedon parametrejä. Kyselyn syntaksissa käytettävät erikoismerkit ovat sen sijaan useimmiten ei-toivottuja. SQL-

injektion tapauksessa syötteiden puhdistukseen kuuluu muun muassa näiden erikoismerkkien poistaminen syötteistä. Puutteellinen syötteiden puhdistus mahdollistaa suoritettavien kyselyiden rakenteen muuttamisen. Hyökkääjä voi ohittaa tietokannan varmennusmekanismeja, laajentaa kyselyn kattavuutta tai suorittaa jonkin muun kyselyn. [4][12]

Varmennusmekanismien ohittaminen on mahdollista, jos käyttäjä voi lisätä varmennettavan tiedon rinnalle jonkin muun ehdon. Esimerkiksi aina paikkaansa pitävä ehto ohittaa varmennuksen täysin. Kyselyn laajentaminen on mahdollista vastaavalla tavalla. [4]

Toisen kyselyn suorittaminen onnistuu, mikäli tietokanta hyväksyy usean peräkkäisen kyselyn lähettämisen. Tällöin käyttäjä voi sijoittaa haluamansa kyselyn osaksi syötettä. Sen voi tehdä esimerkiksi kirjoittamalla alkuperäisen kyselyn loppuun asti syötteessä ja lisäämällä uuden kyselyn tämän perään. Alkuperäisen kyselyn loppuosan voi mitätöidä merkitsemällä sen kommentiksi. [4][12]

Haavoittuvuudesta koituva tietoturvan uhka on ilmeinen. Hyökkäys vaarantaa välittömästi tietokantaan talletettujen tietojen luottamuksellisuuden ja eheyden. Aiheutuva vahinko voi yltää myös tietokannan sisältämien tietojen ulkopuolelle. Uhan laajuus kasvaa, jos tietokantaa voidaan käyttää muissa hyökkäyksissä. [4][12]

SQL-injektion ilmeisin uhka koskee tietokannan sisältämien tietojen turvallisuutta. Hyökkäyksen avulla asiaankuulumaton taho voi saada pääsyn luottamuksellisiin ja salassa pidettäviin tietoihin tietokannassa. Luvatonta pääsyä voidaan yrittää ehkäistä tietokannan sisäisen pääsynvalvonnan avulla. [4][12] Pääsynvalvonta ei kuitenkaan ole täysin luotettava ehkäisykeino [4]. Kuten tässä luvussa mainittiin aiemmin, pääsynvalvonta on mahdollista ohittaa SQL-injektion avulla. Aiheutuva vahinko on helposti eriteltävissä CIA-mallin osa-alueiden avulla [16].

Tietokantaan talletetun tiedon luottamuksellisuus pettää, jos haavoittuvuus sallii luottamuksellisten tietojen haun tietokannasta. Ulkoinen taho voi tällöin hakea ja lukea tietoja oikeudetta. [4]

Tiedon eheys vaarantuu puolestaan silloin, kun haavoittuvuus ilmenee tiedon muutoksen, poiston tai lisäyksen yhteydessä. Lisättävä tieto voi olla muodoltaan ja sisällöltään odotusten vastaista. Kun tietoa poistetaan tai muutetaan, hyökkääjä voi vaikuttaa kohdetiedon sisältöön ja laajuuteen. Esimerkiksi kokonaiseen taulukkoon kohdistuva muutos tai poisto on erittäin vahingollinen. Eheys voidaan kuitenkin osittain taata pitämällä kirjaa tietokantatapahtumista ja varmuuskopioimalla tietoja. Mikäli

haavoittuvuus mahdollistaa peräkkäisten kyselyiden suorittamisen, hyökkääjä kykenee suorittamaan mitä tahansa edellä mainituista toimista. [4]

Monimutkaisempiakin hyökkäyksiä on olemassa. Hyökkääjä voi esimerkiksi kerätä ensin yksityiskohtaista tietoa tietokannasta ja sen rakenteesta. SQL-tietokannat sisältävät keinoja juuri tällaisten tietojen hakemiseen. Hyökkääjä voi hyödyntää näitä keinoja vapaasti, jos haavoittuva sovellus ja tietokanta sallivat useiden peräkkäisten kyselyiden suorittamisen. [12]

Monet SQL-tietokannat voivat lisäksi lukea muita järjestelmän tiedostoja. Esimerkiksi MySQL ja PostgreSQL kykenevät lukemaan tai kopioimaan tiedostojen sisältöä tietokantaan. Seurauksena on hyökkäyspinta-alan laajeneminen. Myös tiedostojen kirjoittaminen tietokannan välityksellä on mahdollista. Monissa tapauksissa tiedostojen kirjoittaminen vaatii kattavammat käyttöoikeudet tietokantaan. [12]

Injektiolla voidaan myös toimittaa muissa hyökkäyksissä käytettäviä haitallisia syötteitä [4][12]. Edellisessä alaluvussa mainittiin jo SQL-injektion hyödyntäminen XSS-hyökkäyksessä. Hyökkäyksessä käytettävä koodi voidaan injektoida tietokantaa, josta se toistetaan myöhempänä ajankohtana [3].

Jotkin SQL-tietokannat tukevat myös järjestelmäkomentojen suorittamista. Microsoft SQL Server on hyvä esimerkki tällaisesta tietokantajärjestelmästä. MS SQL Server tukee järjestelmäkomentojen ja .NET-ohjelmistokehyksen komentojen suorittamista. Tietokantaan pääsevä käyttäjä voi suorittaa tuettuja komentoja etäyhteyden välityksellä. Komentojen tuki on molemmissa tapauksissa oletusarvoisesti pois päältä. Toisaalta komentotuki voidaan kytkeä päälle etäältä. Tuki voidaan kytkeä päälle erillisen SQL-injektion avulla. [12]

Ohjelmassa 4 on esitetty hyvin yksinkertainen SQL-injektiolle alttiin sovelluksen esimerkki. Ohjelma on kirjoitettu PHP-kielellä ja tietokannaksi on valittu PostgreSQL. Esimerkissä suoritettava haavoittuva kysely muodostetaan käyttäjältä saadun evästeen tietojen avulla. Lisäksi ohjelma ilmoittaa käyttäjälle kyselyn epäonnistumisesta ja suorittaa lisätoimintoja kyselyn onnistuessa.

```
<?php
/* Ohjelma hakee tietokannasta käyttäjää evästeistä löytyvän tunnisteluvun uid perusteella ja tulostaa viestin kyselyn tilasta.
*/
$dbconn = pg_connect("dbname=kayttajadb");
$query = "SELECT nimi, lkm, hinta FROM tuotehistoria WHERE ostajaid = '$_COOKIE['uid']'";
$result = pg_query($dbconn, $query);
if ($result) {
   if (pg_num_rows($result == 0)) {
      // Käyttäjää ei löydy
      echo '<p>VIRHE: Käyttäjää ei löydetty tietokannasta.</p>';
   } else {
      // Käyttäjä löytyy. Tulostetaan tuotehistoria.
      tulostus_tuotehistoria($result);
   }
} else {
   echo '<p>VIRHE: Kysely epäonnistui.</p>';
}
pg_close($dbconn);
?>
```

Ohjelma 4. *Yksinkertainen PHP-esimerkki SQL-injektiosta.*

Ohjelman haavoittuvuus perustuu tapaan, jolla kysely muodostetaan [4]. Kyselyn rakenne on helposti ymmärrettävä ja siihen sisällytetään puhdistamaton syöte. Muokattava kysely on talletettuna sellaisenaan merkkijonomuuttujaan. Evästeestä haettava syöte lisätään suoraan kyseiseen merkkijonoon muuttujan alustuksen yhteydessä. Ohjelma ei tarkista syötteen sisältöä, vaan olettaa sen olevan asiallista. Muodostettu kysely lähetetään tietokannan käsiteltäväksi.

Evästeestä haettava syöte voi vaikuttaa tavallista, käyttäjän itse tekemää syötettä turvallisemmalta. Tämä ei kuitenkaan pidä paikkaansa. Tavallisten evästeiden sisältöä voi muokata helposti selaimessa tai kolmannen osapuolen työkaluilla. HttpOnly-evästeiden muokkaus haastavampaa, sillä niiden käsittely vaatii siihen tarkoitettuja työkaluja. Evästeen suojaus ei kuitenkaan takaa sen luotettavuutta.

Esimerkkiohjelman haavoittuvuus mahdollistaa muiden käyttäjien tietojen lukemisen. Tämä onnistuu jo pelkällä hakuehdon muokkauksella. Kyselyn vaihtoehtoiseksi hakuehdoksi voidaan esimerkiksi lisätä aina paikkaansa pitävä väittämä. Se vaatii heittomerkin käyttöä syötteessä, sillä SQL-kyselyt määrittävät sen avulla vakioarvot [9]. Kysely palauttaisi tällöin kaikkien käyttäjien tiedot kohdetaulukosta.

Myös useamman kyselyn suorittaminen on mahdollista. PostgreSQL voi hyväksyä useita peräkkäin asetettuja kyselyitä yhdellä kertaa [9]. Esimerkin ohjelmaa voidaan näin ollen käyttää minkä tahansa kyselyn lähettämiseen. Usean kyselyn suorittaminen vaatii

hyökkääjältä vain oikein muotoillun syötteen. Syötteen tulee päättää alkuperäinen kysely lopetusmerkkiin asti. SQL-kyselyissä lopetusmerkkinä käytetään puolipilkkua [9]. Lopetusmerkin jälkeen hyökkääjä voi syöttää haluamansa kyselyt yksitellen. Alkuperäisestä kyselystä voi myös jäädä ylimääräinen, vajaa osa kyselyn loppuun. Se voidaan mitätöidä merkitsemällä se kommentiksi kahdella miinusmerkillä [9].

Esimerkissä käytetyn PHPn tapauksessa kyselyn turvallisuutta voitaisiin lisätä PHPn valmiiden funktioiden avulla. PHP tarjoaa PostgreSQL:lle valmiita funktioita syötettävän tiedon tarkastuksen, puhdistuksen ja muotoilun avuksi. Lisäksi kullekin tietokantatapahtumalle on erillinen funktio. Jo pelkän tapahtumakohtaisen funktion käyttö esimerkissä rajoittaisi hyökkäyspintaa merkittävästi. Silloin vain tietojen lukeminen onnistuisi. Hyökkäyspintaa voisi pienentää edelleen käyttämällä funktiota, joka muuttaa erikoismerkit turvallisempaan koodattuun muotoon. [8]

SQL-injektioiden ehkäiseminen voi olla hankalaa tekijöiden runsauden vuoksi. Jos kysely muodostetaan dynaamisesti, syötteiden sisältöä ja muotoilua on tarkkailtava. Käyttäjien henkilökohtaisia tietoja sisältävissä tietokannoissa pääsyä on valvottava. Myös virheilmoitusten sisältöä tulee valvoa, sillä ne saattavat paljastaa liiallista tietoa tietokannasta ja sen sisällöstä. [8]

Kuten esimerkin yhteydessä todettiin, valmiiden ja hyväksi todettujen apuvälineiden käyttö on hyödyllistä. Suoritettavia kyselytyyppejä rajoittavat ja syötteitä puhdistavat funktiot ja metodit pienentävät hyökkäyspinta-alaa merkittävästi. Tietokantaa käsiteltäessä kannattaa selvittää, mitä apuvälineitä käytettävälle ohjelmointikielelle tai ohjelmistokehykselle on saatavissa. Syötteen tarkastus ja puhdistus sekä tietokantatapahtumien rajaus on tietysti mahdollista myös ilman erillisiä työkaluja, mutta se voi vaatia merkittävästi enemmän työtä. [4]

Pääsynhallinnassa voidaan noudattaa matalimman tarvittavan oikeuden käytäntöä. Käyttäjille myönnetään vain ne oikeuden, joita he välttämättä tarvitsevat. Tarkoituksena on rajata käyttäjän toimintaa mahdollisimman paljon kunkin tapahtuman kohdalla. On myös tärkeää, että tarkastukset suoritetaan kaikilla merkityksellisillä tasoilla. Esimerkiksi käyttöliittymän puolella tapahtuvat tarkastukset voidaan mahdollisesti ohittaa, joten tarkastukset tulee tehdä myös tietokannassa. [4]

Tavanomaisesti tietokannat sisältävät valmiita virhe- ja järjestelmäilmoituksia. Nämä ilmoitukset voivat sisältää runsaasti hyökkääjille hyödyllistä tietoa. Tämän vuoksi myös tietokannan palauttamiin ilmoituksiin kannatta kiinnittää huomiota. Tavallisen käyttäjän tarvitsee todennäköisesti tietää virhetilanteissa vain virheen perustiedot.

Yksityiskohtaisempien virheilmoitusten ja tietokantajärjestelmän tietoja palauttavien komentojen käyttö tulee rajoitta vain asianomaisiin käyttäjiin. [4]

3.3 Toiminnan rajaus muistialueelle

Tässä alaluvussa keskitytään muistialueen (engl. memory buffer) ylitykseen liittyviin virheisiin. Tällaisiin virheisiin viitataan usein myös muistin ylivuotoina (engl. buffer overflow). Tyypillisesti ohjelmointikielet asettavat muuttujilleen ennalta määritellyt rajat. Nämä rajat määrittävät muuttujien sijainnin ja pituuden järjestelmän muistissa. Samalla ne vaikuttavat muuttujan sisältämän tiedon enimmäissuuruuteen. Muun muassa ohjelmointikielet C, C++ ja Assembly käyttävät tällaisia muistialueita. [5]

Syötteiden käsittelyssä tulee tarvittaessa huomioida muuttujia koskevat muistialueen rajat. Muuttujaan kohdistetun toimenpiteen tulee pysyä muuttujalle varatun muistin rajoissa. On helppoa kuvitella, että ohjelma varmistaa oletusarvoisesti määritellyllä muistialueella pysymisen. Monet ohjelmointikielet myös tekevät juuri näin. Edellisessä kappaleessa mainitut ohjelmointikielet ovat tässä mielessä merkityksellisiä. Ne sallivat tiedon lukemisen suoraan muistista, mutta ne eivät oletusarvoisesti estä muistialueen ylitystä. Tämän takia kyseiset kielet ovat alttiita muistialueen ylitykseen ja alitukseen liittyville virheille. Uudemmat ohjelmointikielet huomioivat muistialueen rajat pääsääntöisesti paremmin, mutta vanhempiakin kieliä käytetään edelleen. [5][13] Muistialueen rajoihin liittyvät virheet ovat erittäin yleisiä alttiiden ohjelmointikielten tapauksessa [1].

Muistialueen ylitys tapahtuu silloin, kun ohjelma tarkastaa muistialueen rajat puutteellisesti ennen luku- tai kirjoitustoimenpidettä. Tällöin ohjelma voi suorittaa toimenpiteen rajatun alueen ulkopuolelle. Toimenpide voi tällöin kohdistua toisen muuttujan sisältöön tai johonkin täysin muuhun muistiin talletettuun tietoon. Lukemisen tapauksessa tämä voi tarkoittaa asiaankuulumatonta tietojen tarkastelua. Tietoa kirjoitettaessa puolestaan voidaan muokata ohjelman tietoja luvattomasti. Tietojen muokkaus muistialueen ulkopuolella voi johtaa myös ohjelman tietojen korruptoitumiseen. [5][13]

Edellä avattiin alustavasti tietoturvalle koituvaa uhkaa. Jos toimintaa ei rajata asianmukaiselle muistialueelle, ohjelma voi lukea tai muokata muistialueen ulkopuoleista tietoa. Virheen seurauksena ulkoinen taho voi käsitellä luottamuksellista ja ohjelmalle kriittistä tietoa luvattomalla tavalla. Virheestä koituva uhka kohdistuu näin sekä tietoon että sitä käsittelevään ohjelmaan. [5][13]

Kun tietoja luetaan muistialueen rajojen ulkopuolelta, tavoitteena on todennäköisesti selvittää tärkeitä tietoja ohjelman toiminnasta. Tiedot muistialueen sijainnista muistissa sekä ohjelman toimintaa ohjaavasta rekisteristä ovat hyödyllisiä jatkohyökkäysten suunnittelussa. Tietoa muistista on mahdollista saada myös ohjelman virheraportin välityksellä. Jos ohjelma kaadetaan kirjoittamalla tietoa muistialueen ulkopuolelle, muistialueen ylityskohdan voi havaita virheraportin tiedoista. Haavoittuvuuden välityksellä vuotava tieto ei siis ole luottamuksellista samalla tavalla, kuin esimerkiksi henkilötieto. Kyseessä on ohjelman turvallisuuden kannalta tärkeiden tietojen luottamuksellisuus. [5][13]

Kuten edellisessä kappaleessa mainittiin, muistialueen yli kirjoittaminen voi kaataa ohjelman. Muistialueen ulkopuolelle kirjoittaminen voi vaatia sitä, että ohjelma käyttää tiedon kirjoittamiseen tiettyjä haavoittuvia funktioita. Pelkkä kaatuminen voi estää ohjelman tarjoaman palvelun saatavuuden. Muistialueen rajan ulkopuoleisten tietojen muuttaminen voi vaikuttaa ohjelman ajo-ohjeisiin, jolloin ajo todennäköisesti keskeytyy, ja ohjelma kaatuu. Tarkempi hyökkäys voi myös ohjata ohjelman ajon johonkin muuhun ajon vaiheeseen. Vakavin uhka koituu kuitenkin tätäkin tarkemmin suunnitelluista hyökkäyksistä. [5]

Vakavin uhka tietoturvalle koituu silloin, kun haavoittuvuutta käytetään mielivaltaisen koodin ajoon. Hyökkääjä voi toimittaa ohjelmalle haitallista, konekielistä koodia ja kirjoittaa ohjelman alkuperäisten toimintaohjeiden yli. Tämän seurauksena ohjelman ajoon voidaan vaikuttaa siten, että se suorittaa hyökkääjältä saadun haitallisen koodin. Hyökkäyksen seuraukset voivat olla kattavat. Hyökkääjä voi muun muassa muuttaa ohjelman tai järjestelmän käyttöoikeuksia tai suorittaa järjestelmän haltuunoton. Tämän vuoksi toiminnan rajaaminen tarkoitetulle muistialueelle on erittäin tärkeää. [5][13]

Erästä puutteellisen toiminnan rajauksen tyyppiä käsitellään ohjelman 5 esimerkissä. Esimerkki pohjautuu CWEn aineistoon [5]. Haavoittuva ohjelma hakee käyttäjältä saadun syötteen perusteella tietoa muuttujasta ja tulostaa haetun tiedon käyttäjän luettavaksi.

```
int getValueFromArray(int *array, int len, int index) {
   int value;
   // Tarkistetaan
    lukujonon järjestysluku index on
   // lukujonon pituutta len pienempi.
   if (index < len) {
      // Haetaan tiettyä järjestyslukua vastaava arvo.
      value = array[index];
   }
   // Järjestysluvun ollessa virheellinen tulostetaan
   // virheilmoitus ja palautetaan virhettä ilmaiseva arvo.
   else {
      printf("Value is: %d\n", array[index]);
      value = -1;
   }
   return value;
}
```

Ohjelma 5. *Yksinkertainen C-kielinen esimerkki puutteellisesta toiminnan rajoituksesta muistialueelle. Esimerkki perustuu lähteeseen. [5]*

Ohjelman haavoittuvuus on hyvin ilmeinen. Luettavan tiedon sijainti eli indeksi määräytyy käyttäjän syötteestä. ohjelma tarkastaa syötteen muistialueen ylittymisen varalta, mutta muistialueen alitusta ei huomioida. Muistialue voidaan alittaa syöttämällä tiedon sijainnille negatiivinen arvo, jolloin ohjelma lukee tietoa muistialueen ulkopuolelta. Hyökkääjä voi näin ollen lukea muistialuetta edeltävää tietoa. Virhe voitaisiin korjata helposti lisäämällä tarkastus myös muistialueen alarajalle. [5]

Yksinkertaisin tapa välttää muistialueen ylityksiä on valita ohjelmointikieli, joka ei ole altis kyseiselle virheelle. Vain tietyt vanhemmat ohjelmointikielet ovat alttiita muistialueen ylityksille. Ohjelma kannattaa toteuttaa uudempaa ja turvallisempaa ohjelmointikieltä käyttäen, mikäli se on mahdollista. Näin virheen syntymiseltä voidaan välttyä ilman ylimääräisiä apuvälineitä. [5]

Virheen syntymistä voidaan ehkäistä myös haavoittuvaa ohjelmointikieltä käytettäessä. Ohjelman toteutuksessa voidaan välttää tunnetusti haavoittuvia funktioita, jotka eivät takaa tai tue toiminnan rajoitusta kohdemuuttujan muistialueelle. Käsiteltävien muuttujien ja syötteiden rajat voidaan myös tarkastaa ja rajoittaa aina ennen käsittelyä. Kirjoittamisen kohteena oleville muuttujille voidaan myös asettaa tarvittavaa suuremmat rajat, jolloin pienet puutteet rajojen tarkastuksessa eivät aiheuta virhettä. Näiden ohjelmointikäytäntöjen tukena voidaan käyttää turvalliseksi todettuja kirjastoja ja ohjelmointikehyksiä. [5]

4. YHTEENVETO

Turvallisessa syötteidenkäsittelyssä tulee aina noudattaa yleisiä ja hyväksi todettuja tietoturvallisen ohjelmoinnin käytäntöjä. Usein on kuitenkin tarpeen olla tietoinen myös tapauskohtaisemmista virhetyypeistä ja niitä ehkäisevistä puolustuskeinoista.

Laajakäsitteisimmät syötteidenkäsittelyn virhetyypit ovat puutteellinen syötteiden tarkastus ja puhdistus. Yhdessä niiden voidaan katsoa kattavan tavalla tai toisella kaikki muut syötteidenkäsittelyn virheet. Vastaavasti kaikki syötteitä käsittelevät ohjelmat ovat niille alttiita. Virhetyyppien ehkäisemiseen ei niiden laajuuden vuoksi ole täsmällisiä keinoja, mutta niitä koskevien yleisten käytäntöjen noudattamista kannattaa soveltaa kaikkien virhetyyppien ehkäisyssä.

Järjestelmät ovat käyttötarkoitustensa mukaan alttiita myös tapauskohtaisemmille ja suppeammille virhetyypeille. Tällaisia virheitä ovat esimerkiksi SQL-injektio, XSS ja muistialueen rajojen ylitys. Nämä virhetyypit ovat hyvin tunnettuja ja erittäin yleisiä. Pienetkin virheet voivat vaarantaa järjestelmien takaisia tietoja. Pahimmillaan nämä virhetyypit voivat johtaa jopa järjestelmien hallinnan menetykseen. Niiden ehkäisemiseen on lukuisia keinoja. Tehokas ehkäisy vaatii kuitenkin uhkien riittävää tiedostamista ja jatkuvaa valppautta ohjelmoijan osalta.

Virheiden tuntemuksesta eivät hyödy vain ohjelmien toteuttajat. Hyökkääjät kykenevät paikantamaan pienetkin puutteet ohjelman toteutuksessa, ja mitättömältäkin vaikuttavat virheet voivat ajan myötä vaarantaa kokonaisia järjestelmiä. Vanhojen ja haavoittuvien toteutusten jatkuva käyttö takaa uusien virheiden syntymisen, vaikka turvallisuutta parantavia apuvälineitä ja uudempia toteutuksia on saatavilla. Syötteidenkäsittely on keskeinen osa ohjelmointia ja siihen liittyvät vaarat on osattava tunnistaa ja ehkäistä oikeissa tilanteissa.

LÄHTEET

[1] 2019 CWE Top 25 Most Dangerous Software Errors, CWE, 2019, Viittauspäivämäärä: 22.9.2019, Saatavissa: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

[2] CWE-20: Improper Input Validation, CWE, 2019, Viittauspäivämäärä: 10.11. 2019, Saatavissa: https://cwe.mitre.org/data/definitions/20.html

[3] CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), CWE, 2019, Viittauspäivämäärä: 11.10.2019, Saatavissa: https://cwe.mitre.org/data/definitions/79.html

[4] CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'), CWE, 2019, Viittauspäivämäärä: 11.10.2019, Saatavissa: https://cwe.mitre.org/data/definitions/89.html

[5] CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer, CWE, 2019, Viittauspäivämäärä: 20.11.2019, Saatavissa: https://cwe.mitre.org/data/definitions/119.html

[6] Measuring the Information Society Report Volume 1, ITU, 2018, Viittauspäivämäärä: 19.10.2019, Saatavissa: https://www.itu.int/en/mediacentre/Pages/2018-PR40.aspx

[7] OWASP Enterprise Security API, OWASP, Viittauspäivämäärä: 02.11.2019, Saatavissa: https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

[8] PHP: PostgreSQL - Manual, PHP, Viittauspäivämäärä: 21.11.2019, Saatavissa: https://www.php.net/manual/en/book.pgsql.php

[9] PostgreSQL 12.1 Documentation, PostgreSQL, 2019, Viittauspäivämäärä: 19.11.2019, Saatavissa: https://www.postgresql.org/docs/12/index.html

[10] Uninitialized variables, cppreference, Viittauspäivämäärä: 29.10.2019, Saatavissa: https://en.cppreference.com/book/uninitialized

[11] What Is Input Validation and Sanitization?, Oracle, Viittauspäivämäärä: 02.11.2019, Saatavissa: https://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_ovw3.htm

[12] J. Clarke, SQL Injection Attacks and Defense. 2nd ed, Waltham, Mass: Elsevier, 2012;2009.

[13] J. Deckard, V. Osipov, N. Bhalla, Buffer Overflow Attacks: Detect, Exploit, Prevent, Elsevier Science & Technology Books, Rockland, 2005, s. 317–358.

[14] M. Di Zio, N. Fursova, T. Gelsema, S. Gießing, U. Guarnera, J. Petrauskienė, L. Quensel-von Kalben, M. Scanu, K.O. ten Bosch, M. van der Loo, K. Walsdorfer, Methodology for data validation 1.0, EC, 2016;2013, Viittauspäivämäärä: 21.10.2019, Saatavissa:

https://ec.europa.eu/eurostat/cros/system/files/methodology_for_data_validation_v1.0_rev-2016-06_final.pdf

[15] P. Le Hégaret, The W3C Document Object Model (DOM), W3C, 2002, Viittauspäivämäärä: 10.11.2019, Saatavissa: https://www.w3.org/2002/07/26-dom-article.html

[16] Y. Le Roux, Information security - The CIA model, Director, 1993.

[17] P. Prasad, Mastering Modern Web Penetration Testing: Master the Art of Conducting Modern Pen Testing Attacks and Techniques on Your Web Application before the Hacker does! 1st ed. Birmingham: Packt; 2016.