Tampereen yliopisto

Juho Keto-Tokoi

TODENNUS OSANA WEB-SOVELLUSTEN TIETOTURVAA

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Matti Monnonen
Elokuu 2021

TIIVISTELMÄ

Juho Keto-Tokoi: Todennus osana web-sovellusten tietoturvaa
Kandidaatin tutkinto
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Elokuu 2021

Käyttäjien todennus on yksi web-sovellusten kriittisimmistä ominaisuuksista paitsi käyttäjien yksityisyyden, myös koko sovelluksen tietoturvallisuuden kannalta. Puutteellinen todennusmekanismi tai todennuksen kiertämisen mahdollistavat tietoturvavirheet voivat johtaa identiteettivarkauteen tai jopa koko sovelluksen haltuunottoon hyökkääjien toimesta. Tämän tutkielman tarkoituksena oli selvittää, minkälaisia hyökkäyksiä web-sovellusten todennusta vastaan kohdistetaan, ja mitä keinoja web-sovellusten kehittäjillä on käytettävissään näiden hyökkäysten estämiseksi.

Työ on kirjallisuuskatsaus, joka jakaantuu kolmeen osaan. Ensimmäinen luku käsittelee käyttäjän todennuskeinoja web-sovelluksissa. Toisessa luvussa esitellään yleisiä hyökkäyksiä, joita todennuskeinoja vastaan kohdistetaan. Kolmannessa luvussa tarkastellaan erilaisia metodeja, joita hyödyntämällä sovelluskehittäjä voi estää nämä hyökkäykset ja parantaa todennuksen tietoturvaa. Lähteenä käytetään alan kirjallisuutta, web-teknologioiden dokumentaatiota sekä alan tunnettujen organisaatioiden teettämiä tietoturvasuosituksia.

Työssä havaittiin, että tunnetuimmat todennusta vastaan kohdistetut hyökkäykset ovat automatisoidut salasanan ja istunnon tunnisteen murtamishyökkäykset, man-in-the-middle-hyökkäykset, sekä cross-site scripting (XSS) -hyökkäykset. Näissä hyökkäyksissä hyökkäysvektorina voi toimia sovelluksen palvelin, käyttäjän selain tai palvelimen ja selaimen välinen tietoliikenneväylä. Hyökkäysten motiivina on käyttäjän salasanan tai istunnon tunnisteen selvittäminen. Kyseiset hyökkäykset ovat yleisiä tänäkin päivänä, ja web-sovelluksissa on usein puutteita, jotka mahdollistavat hyökkäysten onnistumisen. Havaittiin, että web-sovellusten kehittäjillä on käytettävissään tehokkaita keinoja näiden hyökkäysten estämiseksi. XSS-hyökkäykset voidaan estää evästeiden tietoturva-asetusten ja käyttäjien syötteen puhdistuksen avulla. Man-in-the-middle-hyökkäykset saadaan estettyä salaamalla palvelimen ja selaimen välinen tietoliikenne HTTPS-protokollan avulla. Salasanojen ja istunnon tunnisteiden murtaminen voidaan estää vaatimalla niiltä tarpeeksi suurta entropian arvoa ja estämällä sanakirjasanojen käyttö. Salasanan murtaminen tiivisteestä voidaan estää käyttämällä suolaa ja vahvaa tiivistefunktiota salasanan tallennusta varten. Salasanan, istunnon tunnisteen sekä käytetyn tiivistefunktion vahvuuteen liittyviä vaatimuksia tulee arvioida jatkuvasti tietokoneiden laskentatehon noustessa. Salasanan paljastumisesta johtuvia riskejä voidaan pienentää käyttämällä monivaiheista todennusta.

Työ osoittaa, että web-sovellusten puutteellisten todennusmekanismien yleisyys ei johdu aiheeseen liittyvän tiedon saatavuuden puutteesta. Kirjallisuudessa käsitellään hyökkäysten ja haavoittuvuuksien toimintamekanismeja laajasti. Alan organisaatioiden julkaisemista suosituksista löytyy keinot todennukseen kohdistettujen hyökkäysten estämiseen. Nämä suositukset ovat kaikkien saatavilla ilmaiseksi organisaatioiden omilla sivuilla ja niitä päivitetään säännöllisesti. Puutteellisten todennusmekanismien yleisyys johtuu mahdollisesti siitä, että vanhoja sovelluksia ei päivitetä tarpeeksi usein tai osa ohjelmistokehittäjistä ei ole tietoisia todennukseen liittyvistä haavoittuvuuksista uusia sovelluksia kehittäessä.

Avainsanat: Todennus, tietoturva, web-sovellus, tietoturvahyökkäys

SISÄLLYSLUETTELO

1. JOHDANTO

Todennuksella tarkoitetaan palvelun käyttäjän identiteetin varmentamista – tietojärjestelmä varmistaa, että järjestelmässä esiintyvää identiteettiä käyttää vain sitä tosielämässä vastaava henkilö [1]. Web-sovellukset sisältävät usein käyttäjien yksityisiä tietoja ja eritasoisia käyttöoikeuksia sovelluksen resursseihin. Käyttäjien tiedot voivat sisältää esimerkiksi luottokorttitietoja tai käyttäjän henkilötunnuksen tai kotiosoitteen. Sovellusten ylläpitäjillä taas on pääsy ja käyttöoikeudet sovelluksen yhdelle tai useammalle palvelimelle, jotka sisältävät suuren osan koko sovelluksen datasta. Jotta vältyttäisiin yksityisten tietojen leviämiseltä ja käyttöoikeuksien väärinkäytöltä, on tärkeää, että käyttäjä pystyy esiintymään palvelussa vain omalla identiteetillään. Tätä varten web-sovelluksessa täytyy olla toimiva todennusmekanismi.

Toimivan todennuksen kriittisyydestä huolimatta todennusmekanismien viat ovat yleinen ongelma. Hyökkääjillä on usein käytettävissään monia keinoja näiden vikojen hyödyntämiseksi. The Open Web Application Security Project (OWASP) on määrittänyt virheellisen todentamisen web-sovellusten toiseksi kriittisimmäksi tietoturvauhaksi. Sovelluksen alasta riippuen virheellisen todennuksen seurauksena voi olla esimerkiksi luottokorttitietojen leviäminen ulkopuolisille, identiteettivarkaus tai jopa koko sovelluksen haltuunotto. [2]

Tämän tutkimuksen tarkoituksena on selvittää, mitä hyökkäyksiä web-sovellusten todennusmekanismeja vastaan käytetään ja millä keinoilla web-sovelluksen kehittäjä voi niitä estää. Luvussa 1 esitellään yleisimpiä metodeja, joita web-sovelluksissa käytetään käyttäjän todennukseen. Luvussa 2 käydään läpi erilaisia kyberhyökkäyksiä, joita hyökkääjät hyödyntävät web-sovelluksen todennuskeinoja vastaan. Luvussa 3 esitellään keinoja, joita web-sovelluksen kehittäjillä on käytössään näiden hyökkäysten estämiseksi.

2. TODENNUS WEB-SOVELLUKSISSA

Web-sovelluksissa käyttäjien todennus koostuu usein kolmesta osasta – käyttäjän rekisteröitymisestä, kirjautumisesta sekä istunnosta. Rekisteröityminen on vain kerran suoritettava toimenpide, jonka aikana käyttäjälle luodaan identiteetti palveluun. Rekisteröitymisen aikana palveluun tallennetaan tietoa käyttäjästä, jonka avulla hänet voidaan tunnistaa myöhemmin. Kirjautuessaan palveluun käyttäjä syöttää kyseisen tiedon uudelleen palveluun, jotta palvelu voi todentaa hänen identiteettinsä. Istunto on kirjautumisen jälkeistä kommunikaatiota palvelimen ja selaimen välillä. Sen aikana käyttäjä voidaan todentaa jatkuvasti ilman käyttäjän aktiivista osallistumista.

Käyttäjän kirjautuessa palveluun todennusta varten voidaan hyödyntää erilaisia käyttäjän yksilöiviä ominaisuuksia. Nämä ominaisuudet voidaan jakaa seuraavasti [1]:

1. Jotain, mitä käyttäjä tietää, kuten salasana tai PIN-koodi.
2. Jotain, mitä käyttäjä omistaa, kuten pankkikortti tai matkapuhelin.
3. Jotain, mitä käyttäjä on, kuten sormenjälki. Tunnetaan myös biometriikkana.

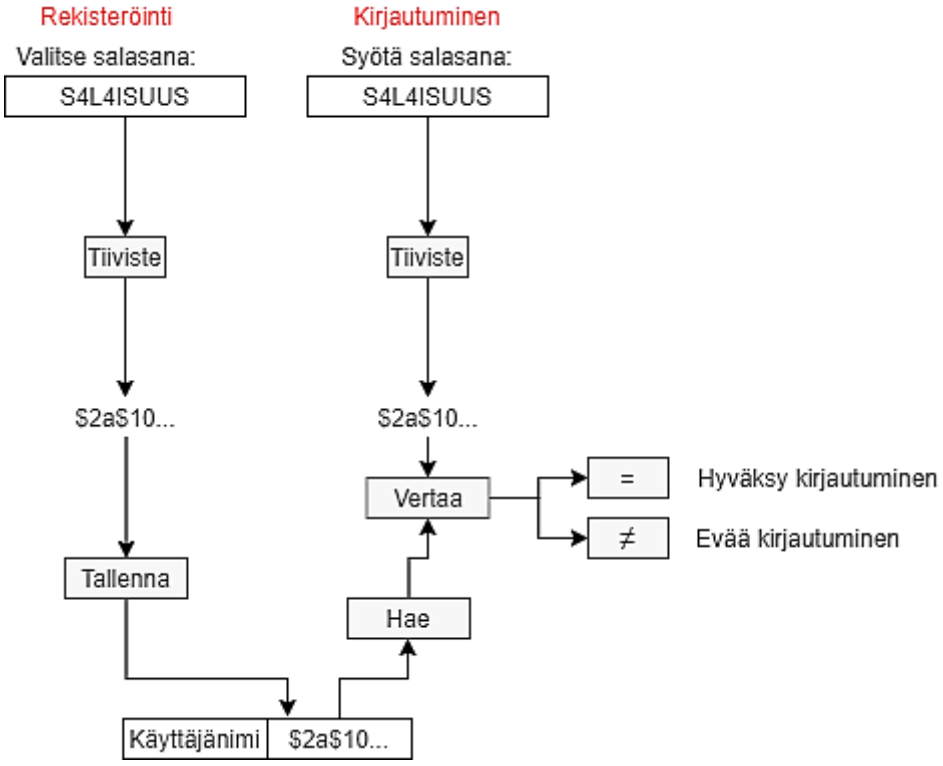
Lisäksi tunnistuksen tukena voidaan käyttää ei-yksilöiviä ominaisuuksia, kuten käyttäjän sijaintia. Tällöin voidaan esimerkiksi sallia vain tietyt alueet, joista palveluun pääsee kirjautumaan, tai rajata tiettyjä alueita pois palvelun piiristä. [3] Jos todennus perustuu useampaan kuin yhteen ominaisuuteen, puhutaan monivaiheisesta todennuksesta. Jos todennus perustuu vain yhteen ominaisuuteen, puhutaan yksivaiheisesta todennuksesta. Sekä monivaiheista että yksivaiheista tunnistautumista käytetään laajasti web-sovelluksissa. Se, mitä ominaisuuksia todentamiseen käytetään ja millä tavalla tieto välitetään, riippuu paljon käytettävän järjestelmän vaatimuksista sekä organisaation tietoturvakäytännöistä [3]. Web-sovelluksissa käytetään harvoin biometriikkaa todennuksen keinona. Sen sijaan keinot 1 ja 2 ovat web-sovelluksissa yleisiä.

2.1 Salasanaan perustuva todennus

Verkossa yleisin käyttäjän todentamiseen käytetty keino on salasana [1][4]. Salasana on merkkijonomuotoinen salaisuus, joka on vain käyttäjän itsensä tiedossa. Rekisteröityessä käyttäjän salasana tallennetaan palvelimelle yhdessä jonkin muun yksilöivän tiedon, kuten käyttäjätunnuksen tai sähköpostin kanssa. Kirjautumista varten

käyttäjä syöttää salasanan uudelleen palveluun, jonka jälkeen palvelin tarkistaa, vastaako syötetty salasana palvelimelle tallennettua salasanaa. Jos salasanat täsmäävät, käyttäjän todennus onnistuu. Yleensä web-sovelluksen käyttäjien salasanat tallennetaan sovelluskehittäjän määrittämään tietokantaan, mutta palvelimen pääkäyttäjien salasanat sijaitsevat palvelimen käyttöjärjestelmän määrittämässä tiedostossa [4].

Salasanoja ei tulisi koskaan tallentaa selkotekstinä eli luettavassa muodossa [5] [4]. Jos salasanat vuotavat palvelimelta luettavassa muodossa, niiden lukija voi käyttää niitä suoraan palveluun kirjautumiseen [5]. Tältä voidaan välttyä käyttämällä selkotekstisille salasanoille tiivistefunktiota. Tiivistefunktiot muuttavat salasanan satunnaiselta näyttäväksi merkkijonoksi, jonka jälkeen tiivistettä ei voida enää palauttaa alkuperäiseen muotoon. [4] Toisin sanoen ne ovat yksisuuntaisia funktioita. Kuvassa 1 on kuvattu salasana-perusteisen todennuksen vaiheet, kun salasana salataan tiivistefunktiolla. Salasana yhdistetään käyttäjän käyttäjätunnukseen, joka on määritetty rekisteröitymisen yhteydessä.



Kuva 1: Salanaan perustuva kirjautumistoiminto [4]

Monesti sovelluskehittäjät toteuttavat tällaisen kirjautumistoiminnon itse web-sovellukseen. Tällöin sovelluskehittäjä toteuttaa palveluun sekä kirjautumissivun, että kuvan 1 mukaisen toiminnon palvelimelle valitsemillaan teknologioilla.

On myös mahdollista käyttää HTTP-protokollan (Hypertext Transfer Protocol) määrittämiä Basic- ja Digest -todennustapoja, jotka antavat web-sovelluksille HTTP-natiivit työkalut salasanaan perustuvan kirjautumisen toteuttamiseen [5]. Tällöin selain vastaa kirjautumisikkunan näyttämisestä käyttäjälle HTTP-protokollan mukaisesti, sekä Digest-todennuksen tapauksessa myös tekee salasanasta tiivisteen [9].

2.2 Todennus istunnon aikana

Web-sovellusten väliseen kommunikaatioon käytetty HTTP on tilaton protokolla, jossa jokainen pyyntö-vastaus-pari on itsenäinen eikä tiedä aiemmasta kommunikaatiosta [9][4]. Usein on kuitenkin toivottavaa, että web-sovellus pitäisi yllä jonkinlaista tilaa käyttäjälle, jotta käyttäjän ei tarvitse kirjautua uudelleen jokaisen pyynnön yhteydessä. Palvelimen ja selaimen välistä jatkuvaa kommunikaatiota, jossa välitetään käyttäjän identifioivaa informaatiota, kutsutaan istunnoksi. Istunnon aikana voidaan välittää lisäksi tietoa käyttäjän toiminnasta palvelussa. Esimerkiksi verkkokaupan istunnossa voidaan välittää käyttäjän ostoskorin sisältö ja sosiaalisessa mediassa viimeksi vieraillut profiilit. Istunnot voidaan jakaa kahteen osaan: palvelinpohjaisiin istuntoihin ja selainpohjaisiin istuntoihin [5].

2.2.1 Palvelinpohjainen istunto

Palvelinpohjaisessa Istunnossa istuntoon liittyvät tiedot tallennetaan palvelimelle. Kirjautumisen yhteydessä palvelin generoi istunnolle merkkijonomuotoisen tunnisteen ja liittää sen kirjautuneeseen käyttäjään tietokannassa. Istunnon tunniste lähetetään käyttäjän selaimeen HTTP-vastauksen yhteydessä. Tämän jälkeisissä HTTP-pyynnöissä käyttäjä todennetaan kyseisen tunnisteen avulla. [5]

Usein tunniste tallennetaan selaimeen evästeenä. Evästeet ovat HTTP-protokollan päälle rakennettu standardi, jota selaimet osaavat hyödyntää datan tallennusta varten. Evästettä käytettäessä palvelimen tulee välittää generoimansa istunnon tunniste HTTP-vastauksen Set-Cookie-otsikossa. Selain taas välittää tunnisteen palvelimelle HTTP-pyynnön Cookie-otsikossa. Palvelin tarkistaa HTTP-pyynnön mukana saamansa tunnisteen ja varmistaa, että tunniste löytyy tallennettuna palvelimelta. Jos tunniste löytyy palvelimelta ja on yhdistettynä käyttäjään, palvelin käsittelee pyynnön kyseisen käyttäjän nimissä. [5] Selaimessa on myös muita tallennusmuotoja evästeiden lisäksi.

Evästeisiin, toisin kuin muihin selaimen tallennusmuotoihin, on määritelty istuntojen kannalta tärkeitä tietoturvaominaisuuksia, joten se on suositeltu tapa tallettaa istunnon tunniste [5].

Palvelimelle tallennettu istunto voidaan katkaista käyttäjän uloskirjautumisen yhteydessä poistamalla voimassa oleva tunniste palvelimelta. Seuraavan kerran kun käyttäjä avaa palvelun, palvelimelle ja selaimeen tallennetut istunnon tunnisteet eivät vastaa enää toisiaan, ja käyttäjää pyydetään kirjautumaan uudelleen sisään. Kirjautumisen yhteydessä palvelin generoi taas uuden tunnisteen ja istunto alkaa alusta.

Palvelinpohjaisten istuntojen etu on se, että palvelimelle voidaan tarvittaessa tallentaa suuria määriä käyttäjän istuntoon liittyvää dataa. Koska istunnon tunniste on vain yksi merkkijono, se ei vie paljoa tilaa käyttäjän selaimesta eikä se lisää HTTP-viestin kokoa merkittävästi.

Istuntojen ongelmana on niiden heikko skaalautuvuus suuriin käyttäjämääriin. Koska istunnot on tallennettu keskitetysti yhdelle palvelimelle, kyseinen palvelin voi muodostua sovelluksen pullonkaulaksi. [5]

2.2.2 Selainpohjainen istunto

Selainpohjaisessa istunnossa istunnon tiedot tallennetaan kokonaisuudessaan käyttäjän selaimeen. Selainpohjainen istunto tuo muutamia etuja palvelinpohjaiseen istuntoon verrattuna etenkin sovelluksissa, joissa tietoa pitää jakaa monen eri palvelun välillä. Sovelluksen suorituskyky paranee, sillä palvelun eri osien ei tarvitse tehdä istunnon muutoksia keskitetylle palvelimelle. Eri palvelimet voivat suoraan muokata istunnon sisältöä HTTP-pyynnön yhteydessä ja lähettää päivitetyn istunnon takaisin selaimeen. Lisäksi palvelimelta säästyy tallennustilaa, sillä istunto on tallennettu käyttäjän selaimeen palvelimella sijaitsevan tietokannan sijasta. [5]

Koska palvelin ei sisällä tallennettua tietoa käynnissä olevasta istunnosta, istunnon tunnisteena on käytettävä jotain muuta palvelimelta löytyvää käyttäjän tietoa, kuten käyttäjätunnusta. Käyttäjätunnus on kuitenkin usein julkista tietoa, joten istunnon väärentämiseltä tulee suojautua. Istunnon sisältö voidaan digitaalisesti allekirjoittaa, jolloin palvelin voi varmistua istunnon alkuperästä ja siitä, ettei istunnon sisältöä olla muokattu. Lisäksi istunnon sisältö voidaan salata salausalgoritmilla, jolloin istunnon sisältö ei ole luettavassa muodossa. [5] Toisin kuin tiivisteen tapauksessa, salausalgoritmilla salattu teksti voidaan palauttaa palvelimella luettavaan muotoon salausalgoritmin käyttämän avaimen avulla.

Haasteena selainpohjaisissa istunnoissa on istunnon päättäminen. Istuntoa ei voida poistaa palvelimelta kuten palvelinpohjaisten istuntojen tapauksessa, sillä istuntoa ei ole tallennettu palvelimelle. Istunnon poistaminen käyttäjän selaimesta ei taas varmista sitä, että istuntoa ei voisi käyttää kyseisen selaimen ulkopuolelta. Vaikka istunnon poistaisi käyttäjän selaimesta, palvelin hyväksyisi istunnon muualta lähetettynä. Selainpohjaisten istuntojen tapauksessa istunnon päättämiseen täytyy siis käyttää muita keinoja.

Kun palvelin on generoinut istunnon, sille voidaan määrittää voimassaoloaika. Tällöin palvelin sisällyttää istuntoon ajankohdan, mihin asti istunto on voimassa. Voimassaoloajan pituudella on merkitystä tietoturvallisuuden kannalta: mikäli hyökkääjä saa istunnon haltuunsa, voimassaoloaika määrittää sen, kauanko hyökkääjä pystyy käyttämään istuntoa tunnistautumiseen. Ainoa tapa mitätöidä selainpohjainen istunto ennen voimassaoloajan päättymistä on, että palvelin pitää yllä listaa mitätöidyistä istunnoista ja lisää käyttäjän istunnon listalle uloskirjautumisen yhteydessä. Tässä metodissa tulee taas esiin kuitenkin sama ongelma, kuin palvelinpohjaisissa istunnoissa, eli palvelimen täytyy tarkistaa istunnon tila keskitetystä paikasta.

3. HYÖKKÄYKSET TODENNUSTA VASTAAN

Todennukseen liittyvissä hyökkäyksissä päämääränä on jonkun käyttäjän kirjautumistietojen tai istunnon haltuun saaminen, jotta hyökkääjä voisi esiintyä palvelussa kyseisenä käyttäjänä. Hyökkääjien tavoitteena voi olla esimerkiksi käyttäjän profiilin yksityisten tietojen levittäminen, huijausviestien lähettäminen käyttäjäprofiilin kautta tai käyttäjän maksutietojen väärinkäyttö. Jos kaapatuilla käyttäjätunnuksilla on laajat käyttöoikeudet web-sovelluksessa, hyökkääjä voi vaarantaa koko sovelluksen toiminnan. Hyökkäyksen kohteena voi olla palvelin- tai selainpuolen sovellus tai reitti, jota pitkin tietoliikenne kulkee. Tässä luvussa käsitellään erilaisia hyökkäystyyppejä, joita hyökkääjät voivat käyttää käyttäjän salasanan tai istunnon tunnisteen selvittämiseen.

3.1 Salasanan ja istunnon tunnisteen murtaminen

Yleinen hyökkääjien käyttämä tapa salasanojen selvittämiseen on hyödyntää algoritmeja, jotka muodostavat erilaisia merkkijonoja, kunnes ennen pitkää joku vastaa oikeaa salasanaa. Tehokkaat tietokoneet pystyvät suorittamaan näitä algoritmeja erittäin tehokkaasti, joten heikot salasanat pystytään ratkaisemaan usein nopeasti. Tällaisia automatisoituja hyökkäyksiä voidaan kohdistaa palveluiden kirjautumissivuille, kunnes kirjautuminen onnistuu, tai niitä voidaan hyödyntää palvelun ulkopuolella, mikäli hyökkääjällä on hallussaan palvelimelta vuotaneita salasanojen tiivisteitä. Salasana voidaan selvittää tiivisteestä vertaamalla algoritmin muodostamaa, tiivistettyä salasanaa oikeaan salasanan tiivisteeseen. Jos tiivisteet ovat samat, algoritmin muodostama merkkijono on sama kuin oikea salasana. [5]

Brute-force-hyökkäyksellä tarkoitetaan menetelmää, jossa hyökkääjän käyttämä algoritmi kokeilee kaikkia mahdollisia merkkijonoja, kunnes merkkijono vastaa käyttäjän salasanaa. Teoriassa brute-force-hyökkäys onnistuu aina, mutta algoritmin suoritusaika kasvaa eksponentiaalisesti salasanan pituuteen nähden ja polynomiaalisesti käytettävän merkkijoukon kokoon nähden. Myös istunnon tunnisteet ovat alttiita brute-force-hyökkäyksille, sillä ne ovat käytännössä vain satunnaisia merkkijonoja. [6]

Sanakirjahyökkäys on brute-force-hyökkäyksestä kehittyneempi hyökkäysmuoto, jossa algoritmin suoritusaikaa pyritään lyhentämään kokeilemalla vain todennäköisimpiä merkkiyhdistelmiä. Helpoiten muistettavat salasanat ovat usein sanakirjoista löytyviä sanoja, tai jonkinlaisia muunnoksia niistä. [6][7] Muista palveluista vuotaneet salasanat ovat hyökkääjille houkuttelevia, sillä useat käyttäjät käyttävät samaa tai lähes samaa

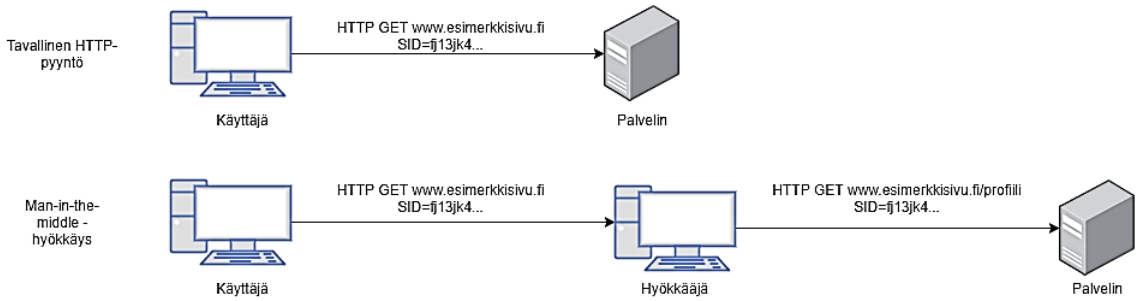
salasanaa useassa eri palvelussa [3]. Sanakirjahyökkäystä varten kootaan sanakirja, joka sisältää joitain näistä todennäköisimmistä merkkiyhdistelmistä ja niitä käytetään suoraan tai niitä koitetaan muokata, jotta päästäisiin oikeaan salasanaan. Sanakirjahyökkäyksen aikakompleksisuus on lineaarinen käytettävän sanakirjan pituuteen ja sanaan tehtävien muokkausten määrään nähden. [7]

Salasanojen selvittämiseen tiivisteistä voidaan käyttää sanakirjaa, joka koostuu suuresta määrästä erilaisia tiiviste-selkoteksti -pareja. Jokaisen salasanatiivisteen kohdalla voidaan tarkistaa, löytyykö tiiviste sanakirjasta. Jos löytyy, saadaan selville tiivisteen selkotekstinen muoto. [5] Tiivisteen haku sanakirjasta on paljon nopeampi operaatio, kuin tiivistefunktion laskeminen jokaiselle merkkijonolle. Toisaalta sanakirjan tallentaminen vie paljon tallennustilaa. Tallennustilan ja suoritusajan suhteen optimoitu versio tällaisesta sanakirjasta on nimeltään Rainbow Table -sanakirja. [7]

3.2 Man-in-the-middle-hyökkäys

Man-in-the-middle-hyökkäyksessä hyökkääjä onnistuu seuraamaan tietoliikennettä käyttäjän ja palvelimen välissä [5][4]. Hyökkääjän on mahdollista kuunnella palvelimen ja selaimen välisiä HTTP-viestejä, sillä HTTP-viestit ovat salaamattomia. Tämän seurauksena hyökkääjä voi lukea tai muokata pyynnön sisältöä ja sen myötä saada selville viestin sisältämän istunnon tunnisteen tai käyttäjän salasanan. [5]

Langattomat lähiverkot ovat yleinen kohde man-in-the-middle-hyökkäyksille. Usein reitittimet sisältävät hyvin minimaalisen Linux-käyttöjärjestelmän version, johon ei usein olla asennettu uusimpia tietoturvapäivityksiä. Tällöin reititin todennäköisesti sisältää haavoittuvuuksia, joita hyödyntämällä hyökkääjä voi saada pääsyn reitittimeen ja lukea sen läpi kulkevaa tietoliikennettä. [5] Kuvassa 3 on esimerkki man-in-the-middle-hyökkäyksestä, jossa hyökkääjä lukee istunnon tunnisteen (SID) käyttäjän HTTP-pyynnöstä ja käyttää sitä todentaakseen itsensä käyttäjän profiilisivulle.



Kuva 2: Esimerkki man-in-the-middle-hyökkäyksestä

Hyökkääjä voi myös ylläpitää omaa langatonta tukiasemaansa julkisella paikalla. Pahaa aavistamaton käyttäjä yhdistää laitteensa hyökkääjän tukiasemaan, jonka seurauksena hyökkääjä saa selville salaamattoman tietoliikenteen, jota laite vastaanottaa tai lähettää. [5]

3.3 Cross-site scripting -hyökkäys

Cross-site scripting (XSS) -hyökkäykset ovat tavallisimpia hyökkäyksiä, joita web-sovelluksiin kohdistetaan [2]. XSS-hyökkäyksen tavoitteena on injektoida JavaScript-koodia osaksi käyttäjälle lähetettävää HTML-sivua, jonka seurauksena koodi suoritetaan käyttäjän selaimessa. Tällaista hyökkäystä voidaan käyttää istuntotietojen varastamiseen tai käyttäjän selaimesta. [5] Hyökkäyksen seurauksena hyökkääjä voi lähettää HTTP-pyyntöjä palveluun kirjautuneen käyttäjän selaimesta, antaen hyökkääjälle täyden pääsyn käyttäjän tietoihin. Hyökkääjä voi myös lähettää hyökkäyksen seurauksena istunnon tunnisteen haluamaansa osoitteeseen, mikäli istunnon tunnistetta ei olla tallennettu turvallisin keinoin käyttäjän selaimeen. Tällöin hyökkääjä pystyy toimimaan palvelussa toisena käyttäjänä myös suoraan omalla selaimellaan. OWASPin raportin mukaan noin kaksi kolmasosaa web-sovelluksista on haavoittuvaisia XSS-hyökkäyksille [2]. XSS-hyökkäykset voidaan jakaa kolmeen eri luokkaan; tallennettuun, heijastettuun ja DOM-pohjaiseen. Lopputulos on sama kaikissa näistä varianteista, eli hyökkääjä onnistuu suorittamaan JavaScript-koodia käyttäjän selaimessa, mutta ne eroavat hieman toteutustavaltaan. [5][8]

Tallennetussa XSS-hyökkäyksessä hyökkääjä onnistuu tallentamaan JavaScript-koodinsa sellaiseen sovelluksen tietokantaan, josta palvelin hakee dataa HTML-sivun muodostamista varten. Tämän seurauksena palvelin sisällyttää tietokantaan tallennetun hyökkääjän koodin osaksi selaimen pyytämää sivua ja lähettää sivun selaimelle. Sivun vastaanotettuaan selain suorittaa sivulle upotetun JavaScript-koodin, jolloin myös hyökkääjän koodi suoritetaan. [5][8] Tällainen hyökkäys on mahdollinen sellaisissa sovelluksissa, joissa käyttäjän syötettä tallennetaan tietokantaan ilman kunnollista

syötteen tarkistusta. Kohteita tälle hyökkäykselle voivat olla esimerkiksi sovelluksen kommenttipalstat, profiilisivut tai muut sovelluksen osa-alueet, joissa käyttäjä voi itse syöttää pitkäaikaista sisältöä palveluun.

Heijastetussa XSS-hyökkäyksessä palvelin sisällyttää HTTP-pyynnössä lähetettyjä parametreja osaksi generoimaansa HTML-sivua. Parametreja ei tallenneta pysyvästi palvelimen käyttämään tietokantaan, vaan ne sisällytetään sivuun vain kyseisen pyynnön yhteydessä. Sivulle sisällytettävät parametrit voidaan ottaa esimerkiksi pyynnön URL:n (Uniform Resource Locator) kysely-osasta. Tässä tapauksessa hyökkääjä voisi sisällyttää JavaScript-koodia URL:n kyselyosaan. Jos palvelin ei tarkista ja korjaa syötettä, JavaScript koodi lisätään sellaisenaan osaksi HTML-sivua ja se suoritetaan käyttäjän selaimessa. [5][8] Esimerkiksi web-sovelluksen hakutoiminto, jossa käyttäjän syöttämät hakusanat sisällytetään sivun URL-osoitteeseen, voisi olla haavoittuvainen heijastettua XSS-hyökkäystä vastaan.

Kuten heijastetussa XSS-hyökkäyksessä, myös DOM-pohjaisessa hyökkäyksessä hyökkääjä voi välittää JavaScript-koodia osana URL:ia tai muiden HTTP-pyynnön parametrien mukana. Erona on se, että DOM-pohjaisessa XSS-hyökkäyksessä selain muodostaa HTML-sivun sisällön, eikä palvelin. [5][8]

4. HYÖKKÄYSTEN ENNALTAEHKÄISY

Aiemmassa luvussa esitellyt hyökkäykset voivat johtaa vakaviin seurauksiin käyttäjien yksityisyyden ja koko sovelluksen turvallisuuden kannalta. Siksi on tärkeää, että sovelluskehittäjät pyrkivät ennaltaehkäisemään hyökkäysten mahdollisuuksia web-sovellusta kehittäessään. Tässä osiossa käydään läpi keinoja, joita hyödyntämällä sovelluskehittäjä voi minimoida kyseisiä uhkia.

4.1 Monivaiheinen todennus

Pelkkää salasanaa käyttävän todennuksen toimivuus perustuu siihen, että salasanan tietää vain käyttäjä itse. Salasanan vuotaessa vääriin käsiin, kuka tahansa voi kirjautua palveluun käyttäjän tietämättä. Tilannetta pahentaa se, että käyttäjät saattavat käyttävää samaa salasanaa useassa eri palvelussa, jotta salasanan muistaminen olisi helpompaa. Tällöin jos salasana paljastuu yhdestäkin palvelusta ulkopuolisille, käyttäjän identiteetti on vaarassa myös muissa palveluissa. Salasanan paljastumisen seurauksia voidaan lieventää käyttämällä monivaiheista todennusta. Monivaiheinen todennus on todennustapa, jossa käyttäjän todentamiseen käytetään enemmän kuin yhtä käyttäjään liittyvää ominaisuutta [3]. Näitä eri ominaisuuksia käsiteltiin luvussa 1.

Web-sovelluksissa usein käytetään salasanapohjaisen tunnistautumisen lisäksi jotakin käyttäjän omistamaa laitetta monivaiheisen todentamisen toteuttamiseksi. Laitteeseen voidaan lähettää vahvistusviesti, jota painamalla todennus hyväksytään tai kertakäyttökoodi, jonka syöttämällä palveluun todennus onnistuu [3]. Tällöin todennukseen hyödynnetään kahta erillistä käyttäjän ominaisuutta: salasanaa, eli jotain mitä käyttäjä tietää, sekä matkapuhelinta, eli käyttäjän omaisuutta.

Monivaiheisen todennuksen etu on siinä, että vaikka yhden tunnistautumiseen käytetyn vaiheen tiedot paljastuisivatkin hyökkääjälle, hyökkääjä ei pysty tunnistautumaan käyttäjänä läpäisemättä kaikkia tunnistautumisen vaiheita. Pelkkään salasanaan pohjautuvassa tunnistautumisessa hyökkääjä voi tunnistautua käyttäjänä palvelussa pelkän salasanan selvittämällä. [3]

OWASPin mukaan pelkän salasanan käyttö on usein syy sille, että todennukseen kohdistuvat hyökkäykset onnistuvat. OWASP kehottaa käyttämään monivaiheista todennusta pelkän salasanapohjaisen todennuksen sijaan. [2]

4.2 Vahvat salasanat

Jotta salasanojen murtaminen olisi vaikeampaa, on tärkeää käyttää vahvoja salasanoja. Eräs yleisesti käytetty salasanan vahvuutta mittaava suure on entropia. Entropia kuvaa salasanan epävarmuutta, eli kuinka todennäköisesti arvattu salasana on oikea salasana mahdollisten salasanojen joukosta. Salasanan entropia on siis sitä suurempi, mitä pidempi salasana on ja mitä suurempi on sallittujen merkkien joukko. Salasanan entropia voidaan kuvata funktiolla

$$\mathrm{H(w)} = log_2(N^m), \tag{1}$$

jossa valittu sanasana on *w*, salasanan pituus on *m* ja sallitun merkkijoukon koko *N*. Salasanan pituutta ja merkkijoukon kokoa ilmaistaan merkkeinä. Entropian yksikkö on bitti. [7]

Entropian lisäksi salasanan vahvuutta mitattaessa voidaan ottaa huomioon ihmisten yleiset taipumukset sanasanojen muodostuksessa. Salasanoja usein luodaan käyttämällä sanakirjoista löytyviä sanoja tai sanoja hieman muokkaamalla, jotta ne olisi helpompi muistaa. Useat nykyisistä sanakirjahyökkäyksistä ottavat tämän huomioon ja niihin käytetyt algoritmit osaavat tuottaa yksinkertaisia sanojen muunnoksia. Tästä syystä salasanan vahvuuden mittarina voidaan käyttää sitä, kuinka monta kirjainten lisäystä, muokkausta tai poistoa täytyisi tehdä, jotta sanakirjasanasta päädyttäisiin oikeaan salasanaan. [7]

National Institute of Standards and Technology (NIST) on aiemmin julkaissut ohjeet salasanan vahvuudelle H, joka ottaa huomioon salasanan pituuden, käytetyn merkkijonon pituuden sekä vahvuuden sanakirjakirjahyökkäystä vastaan: [7]

$$H = \begin{cases} +4\ ensimmäisestä\ merkistä \\ +2\ joka\ toisesta\ ja\ 8{:}sta\ merkistä \\ +1.5\ joka\ 9{:}stä\ ja\ 20{:}stä\ merkistä \\ +6\ isojen\ merkkien\ sekä\ numeroiden\ tai\ erikoismerkkien\ käytöstä \\ +6\ salasana\ ei\ löydy\ sanakirjoista \end{cases}$$

Uusimmassa julkaisussaan NIST ei kuitenkaan suosittele vaatimuksia sille, että salasanasta löytyy tiettyjä merkkejä. Sen sijaan NIST suosittelee salasanan vahvuuden mittarina käytettävän vain salasanan pituutta sekä sitä, että salasana ei löydy sanakirjasta. Merkkivaatimusten jättäminen pois suosituksista johtuu siitä, että käyttäjät usein pyrkivät täyttämään vaatimukset ennalta-arvattavalla tavalla, eli tyypillisesti vaihtamalla ensimmäisen alkukirjaimen isoksi sekä lisäämällä vaaditut erikoismerkit salasanan loppuun. [10] Toisin sanoen käyttäjät kyllä täyttävät palvelun vaatimukset, mutta käyttäjien ennalta-arvattavuuden takia suoja monia sanakirjahyökkäyksiä vastaan

jää usein pieneksi. Tästä syystä vaatimukset tietyistä merkeistä eivät usein paranna tietoturvaa, vaan lähinnä turhauttaa palvelun käyttäjiä. Uusimmassa julkaisussaan NIST painottaa, että salasanan pituus on tärkein salasanan vahvuuteen vaikuttava tekijä. [10] Tämä voidaan todeta myös entropian kaavasta. NIST suosittelee salasanan pituudeksi vähintään 8 merkkiä [10]. OWASP suosittelee pitkälti samoja kriteereitä salasanan vahvuuden mittaamiseksi, kuin NIST. OWASPin mukaan salasanan tulisi olla vähintään 8 merkkiä pitkä silloin kun monivaiheinen todennus on käytössä ja 10 merkkiä silloin, kun monivaiheinen todennus ei ole käytössä [2]. Lisäksi molemmat suosittelevat, että kaikkien merkkien käyttö sallitaan salasanaa muodostaessa [2][10]. Näillä suosituksilla salasanan entropiaa pyritään kasvattamaan mahdollisimman suureksi.

4.3 Vahvat salasanojen tiivistefunktiot

Useiden hyökkäysten tavoitteena on päästä käsiksi käyttäjien tietoja sisältäviin tietokantoihin. Mikäli hyökkääjä saa palvelimelta salasanat haltuunsa luettavassa muodossa, kaikki salasanan vahvuuteen käytettävät metodit muuttuvat turhiksi. Salasanojen luettavuus voidaan estää tiivistämällä palvelimelle tallennetut salasanat tiivistefunktiolla [11]. Koska tiivistettä ei voida muuttaa takaisin selkotekstiseen muotoon, ainoa tapa selvittää salasana tiivisteestä on muodostaa erilaisista merkkijonoista tiivisteitä ja verrata niitä salasanan tiivisteeseen. Valittu tiivistefunktio vaikuttaa siihen, kuinka helposti tämä prosessi voidaan toteuttaa [11].

Tietokoneiden laskentatehokkuuden kasvun myötä salasanoja pystytään arvaamaan automaattisesti entistä tehokkaammin. Tästä syystä monet aiemmin käytetyt tiivistefunktiot ovat muuttuneet ajan kuluessa heikoiksi salasanan murtamista vastaan [11]. Esimerkiksi HTTP-protokollan määrittämän Digest-todennustavan oletuksena käyttämää tiivistefunktiota MD5:tä pidetään nykyään heikkona salasanan tiivistefunktiona, sillä funktion suoritusaika on nopea nykyaikaisilla tietokoneilla [11]. Nopeuttaakseen salasanojen arvausta hyökkääjät voivat hyödyntää useita rinnakkaisia grafiikkasuorittimia, joilla on suuri laskentateho tai hyödyntää pilvipalveluita, joiden kautta he saavat ulkopuolista laskentatehoa käyttöönsä. Modernit salasanojen tiivistämiseen tarkoitetut funktiot ottavat huomioon laskentatehon kasvun ja tekevät salasanojen murtamisen tehokkaimmillakin laitteilla vaikeaksi. Useat näistä tiivistefunktioista mahdollistavat tiivisteen tuottamiseen kuluvan suoritusajan skaalaamisen lisäämällä funktion muistinkäyttöä tai tiivistämiskierroksia [11]. Ideaalisesti molempia. Näiden ominaisuuksien vuoksi salasanojen tiivistämiseen suositellaan juuri siihen tarkoitukseen suunniteltuja tiivistefunktioita. NIST suosittelee PBKDF2- ja Balloon

-tiivistefunktioita [10]. OWASP suosittelee Argon2id-, bcrypt-, scrypt- sekä PBKDF2-algoritmeja [11].

Tiivisteen muodostamiseen tulisi käyttää suolaa (engl. salt), eli jokaiselle käyttäjälle erilaista merkkijonoa, joka lisää satunnaisuutta tiivisteen lopputulokseen. Suola lisätään osaksi käyttäjän salasanaa ennen tiivisteen muodostamista, jolloin se tulee osaksi lopullista tiivistettä. Koska suolan arvo on eri kaikille käyttäjille, kaksi samaa salasanan tiivistettä ei esiinny samanlaisena merkkijonona tietokannassa. Toisin sanoen suolan käytön ansiosta hyökkääjä ei voi päätellä tiivisteistä, käyttääkö kaksi käyttäjää samaa salasanaa. Modernit tiivistefunktiot, kuten Argon2id, bcrypt-sekä PBKDF2 lisäävät suolan salasanoihin automaattisesti. [11] Suolaa käyttämällä voidaan tehokkaasti eliminoida salasanojen tiivisteisiin kohdistetut automatisoidut hyökkäykset, sillä hyökkääjän tulisi arvata pelkän salasanan lisäksi myös oikea suolan arvo. Arvausten määrä, ja sen myötä suoritusaika kasvaa niin suureksi, että hyökkäys tulee käytännössä kannattamattomaksi [11]. Rainbow tablea sekä muita sanakirjoja hyödyntävien hyökkäysten tapauksessa hyökkääjän sanakirja tulisi liian suureksi, sillä sanakirjan tulisi sisältää kaikki mahdolliset suolan arvojen ja merkkijonojen yhdistelmät [5][11].

4.4 Tietoliikenteen salaaminen

HTTP-viestit ovat oletuksena salaamattomia [5]. Kuten kappaleessa 3.2 todettiin, salaamattomat HTTP-viestit ovat alttiita Man-in-the-middle-hyökkäyksille. Hypertext Transfer Protocol Secure -protokolla (HTTPS) on HTTP-protokollan jatke, jonka avulla voidaan lähettää salattuja viestejä palvelimen ja selaimen välillä [5]. HTTPS-protokolla salaa selaimen ja palvelimen välisten viestien sisällön käyttäen Transport Layer Security (TLS)-protokollaa. HTTPS-yhteyttä muodostaessa suoritetaan TLS-protokollan mukainen kättelyvaihe, jonka aikana selain varmistaa palvelimen identiteetin sekä palvelin ja selain sopivat yhteyden salaamiseen käytettävistä parametreista. Palvelimen identiteetin varmistus tapahtuu varmenteen avulla, jonka palvelin on hankkinut luotetulta varmenteentarjoajalta. Palvelin lähettää varmenteen selaimelle TLS-kättelyn aikana ja selain varmistaa varmenteentarjoajalta, että palvelimen varmenne on voimassa. TLS-kättelyn suoritettuaan palvelin ja selain salaavat HTTP-viestit symmetrisellä salauksella. [5]

Salauksen ja palvelimen identiteetin varmistuksen ansiosta HTTPS-yhteyden käyttö estää tehokkaasti Man-in-the-middle-hyökkäyksien toteutuksen [5]. Koska HTTPS-protokolla salaa kaikkien HTTP-viestien sisällöt, hyökkääjä ei voi kuunnella viestejä pääsemättä suoraan käsiksi palvelimelle tai käyttäjän selaimeen. OWASP ja NIST suosittelevat, että HTTPS-yheyttä käytetään aina todennukseen käytettäviä tietoja

sisältävien HTTP-viestien yhteydessä [2][10]. Evästeitä käytettäessä tulisi varmistaa, että Secure -parametri on voimassa, koska silloin evästettä ei välitetä muun kuin HTTPS-yhteyden välityksellä.

4.5 Vahvat istuntojen tunnisteet ja niiden säilytys

Salasanojen tapaan myös palvelinpohjaisissa istunnoissa käytetyt istunnon tunnisteet ovat alttiita brute-force-hyökkäyksille [5]. Mikäli hyökkääjä onnistuu selvittämään toisen käyttäjän istunnon tunnisteen, hyökkääjä voi todentaa itsensä palveluun toisena käyttäjän identiteetillä. Koska istuntojen tunnisteet ovat merkkijonoja, niiden vahvuutta voidaan mitata entropialla [12]. Koska selain tallentaa tunnisteen eikä käyttäjän tarvitse muistaa sitä, tunniste tulisi pitää mahdollisimman satunnaisena ja pitkänä. OWASP suosittelee istunnon tunnisteiden pituudeksi vähintään 128 bittiä ja entropian arvoksi vähintään 64 bittiä [12]. Myös NIST suosittelee istunnon tunnisteen pituudeksi vähintään 64 bittiä [10]. OWASPin mukaan näillä arvoilla hyökkääjällä kestäisi 292 vuotta olemassa olevan istunnon tunnisteen arvaamiseksi, mikäli hyökkääjä pystyy tekemään 10 000 arvausta sekunnissa ja olemassa olevia tunnisteita on 100 000. [12]

Evästeiden säilytyksen ja lähetyksen turvallisuutta voidaan parantaa niihin sisäänrakennettujen tietoturva-asetusten avulla. HTTP-only -asetus varmistaa sen, että evästeen sisältöön ei päästä käsiksi JavaScript-koodin kautta. Secure -asetus varmistaa, että eväste lähetetään palvelimelle vain HTTPS-yhteyttä käyttäen. SameSite -asetuksella voidaan määrittää, että evästettä ei voida lähettää selaimesta kolmansien osapuolien palveluille. Nämä asetukset ovat käytettävissä vain evästeissä ja niitä tuetaan suurimmassa osassa selaimista. Vastaavia mekanismeja ei ole käytettävissä, mikäli istunto tai istunnon tunniste tallennetaan jonnekin muualle selaimen muistiin. Näiden asetusten yhdistelmällä voidaan varmistaa, että XSS-hyökkäyksen seurauksena hyökkääjä ei pysty lukemaan istunnon tietoja tai lähettämään niitä ulkopuoliseen palveluun.

4.6 Käyttäjän syötteen puhdistus

Kuten edellisessä luvussa todettiin, evästeiden tietoturva-asetuksien avulla voidaan suojautua siltä, että hyökkääjä ei pysty lähettämään evästettä itselleen XSS-hyökkäyksen seurauksena. XSS-hyökkäyksen avulla hyökkääjä voi kuitenkin lähettää pyyntöjä palveluun käyttäjän selaimesta ilman, että hän pääsee itse evästeeseen käsiksi. Tämä johtuu siitä, että evästeet lähetetään pyynnön mukana automaattisesti sille palvelimelle, josta selain on ne alun perin saanut [13]. XSS-hyökkäyksen aiheuttama istunnon kaappaus on siis evästeiden tietoturva-asetuksia käytettäessäkin edelleen

mahdollinen, vaikkakin evästeiden tietoturva-asetukset rajoittavat XSS-hyökkäysten toteutusta. XSS-hyökkäysten estämiseksi kokonaisuudessaan on varmistettava, että selain ei voi suorittaa hyökkääjän syötteen sisältämää JavaScriptiä. Tätä varten käyttäjien syöte on puhdistettava, ennen kuin se lisätään osaksi HTML-sivua. Syöte puhdistetaan muuttamalla JavaScriptin suorittamiseen vaadittavat merkit syötteestä joko poistamalla ne kokonaan tai muuttamalla ne johonkin toiseen muotoon.

Syötteen puhdistus on tarkkuutta vaativa prosessi, sillä syöte tulee puhdistaa eri tavoilla riippuen selaimen suorituskontekstista, jossa sitä käytetään. JavaScriptiä pystytään suorittamaan selaimessa eri tavoilla riippuen siitä, onko JavaScript-koodi esimerkiksi HTML-elementin sisällä, HTML-elementin attribuuttina, vai osana CSS-tyyliä. Erilaisia JavaScriptin suorittamisen mahdollistavia konteksteja on selaimissa monia erilaisia, joten syötteen puhdistuksen tekijällä tulee olla niistä jokaisesta hyvä käsitys. OWASP on julkaissut kattavat ohjeet erilaisista toimista, joiden avulla käyttäjän syöte voidaan puhdistaa kaikissa eri selaimen suorituskonteksteissa. Ensisijaisesti OWASP kuitenkin suosittelee sovelluskehittäjiä käyttämään puhdistusta varten siihen sopivaa kirjastoa. Tällaisia kirjastoja löytyy monille yleisesti käytetyille palvelinpuolen ohjelmointikielille. Lisäksi monet sovelluskehykset sisältävät sisäänrakennetun XSS-suojausmekanismin. [14]

Onnistuneen käyttäjän syötteen puhdistuksen seurauksena käyttäjä ei voi syöttää palveluun haitallista JavaScript-koodia missään suorituskontekstissa [14]. Tällöin XSS-hyökkäystä ei voida käyttää istunnon tunnisteen tai salasanan kaappaamiseen.

5. YHTEENVETO

Työn tarkoituksena oli perehtyä todennusta vastaan kohdistettuihin hyökkäyksiin web-sovelluksissa ja löytää keinoja niiden estämiseen. Tavoitteena oli löytää keinoja, joiden avulla web-sovelluskehittäjä voi toteuttaa käyttäjän todennuksen web-sovelluksessa mahdollisimman tietoturvallisesti. Hyökkäysten ja niiden ennaltaehkäisyyn käytettävien keinojen esittelyn tukena käytettiin lukua 2, jossa perehdyttiin erilaisiin käyttäjän todentamiskeinoihin.

Havaittiin, että tunnettuja todennukseen kohdistettuja hyökkäyksiä olivat automatisoidut salasanojen ja istunnon tunnisteiden murtamishyökkäykset, Cross-site scripting -hyökkäykset sekä man-in-the-middle-hyökkäykset. Kyseisten hyökkäysten avulla hyökkääjät voivat kaapata tai selvittää käyttäjän salasanan tai istunnon tunnisteen. Kirjallisuudesta löytyi toimivia keinoja jokaisen näistä hyökkäyksistä estämiseen. Kyseiset keinot löytyivät myös asiantuntijaorganisaatioiden julkaisemista, ohjelmistokehittäjille suunnatuista suosituksista. Koska todennus on kriittinen web-sovellusten osa-alue ja tietoa erilaisista uhkista ja niiden estämiskeinoista on laajasti saatavilla, jokaisen todennusmekanismeja toteuttavan web-sovelluskehittäjän tulisi olla niistä tietoisia. Sovelluskehittäjien tulisi seurata suosituksia säännöllisesti, sillä ne muuttuvat ajan kuluessa eivätkä aiemmat suositukset välttämättä ole tulevaisuudessa enää tietoturvallisia.

Tutkielmassa käytetyn kirjallisuuden perusteella web-sovellusten todennukseen kohdistetut hyökkäykset ja todennuksen haavoittuvuudet ovat jo pitkään tunnettuja. Asiantuntijaorganisaatioiden mukaan ne ovat silti edelleen yleisiä. Jatkotutkimuksena voisi selvittää, miksi todennuksen haavoittuvuudet ovat edelleen yleisiä, vaikka niiden ennaltaehkäisyyn käytetyistä keinoista on paljon tietoa saatavilla. Mahdollisesti tämä johtuu siitä, että vanhoja sovelluksia ei päivitetä tarpeeksi usein tai osa ohjelmistokehittäjistä ei ole tietoisia haavoittuvuuksista ja niihin kohdistuvista hyökkäyksistä uusia sovelluksia kehittäessä.

LÄHTEET

[1] Linden, M. (2017). Identiteetin- ja pääsynhallinta. (Tampereen teknillinen yliopisto. Tietotekniikan laboratorio. Raportti; Vuosikerta 7). Tampere University of Technology.

[2] OWASP Top 10 Web Application Security Risks, 2017. Viitattu 1.8.2021. Saatavilla: https://owasp.org/www-project-top-ten/

[3] Dasgupta, D. et al. (2017) Advances in User Authentication. Cham: Springer International Publishing AG.

[4] Nigel Chapman and Jenny Chapman. 2012. Authentication and Authorization on the Web. MacAvon Media.

[5] Malcolm McDonald (2020) Web Security for Developers. No Starch Press.

[6] Uchnar, M. & Hurtuk, J. (2017) 'Safe user authentication in a network environment', in 2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI). [Online]. 2017 IEEE. pp. 000451–000454.

[7] Hu, G. (2017) 'On Password Strength: A Survey and Analysis', in Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. [Online]. Cham: Springer International Publishing. pp. 165–186.

[8] Rodríguez, G. E. et al. (2020) Cross-site scripting (XSS) attacks and mitigation: A survey. Computer networks (Amsterdam, Netherlands : 1999). [Online] 166106960–.

[9] IETF (2014) Hypertext Transfer Protocol: Authentication. Viitattu 12.8.2021. Saatavilla: https://datatracker.ietf.org/doc/html/rfc7235

[10] NIST Special Publication 800-63B: Digital Identity Guidelines, Authentication and Lifecycle Management. Saatavilla: https://pages.nist.gov/800-63-3/sp800-63b.html

[11] OWASP Password Storage Cheat Sheet. Viitattu 11.8.2021. Saatavilla: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[12] OWASP Session Management Cheat Sheet. Viitattu 12.8.2021. Saatavilla: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[13] Mozilla developer documentation: Cookies. Viitattu 12.8.2021. Saatavilla: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[14] OWASP Cross Site Scripting Prevention Cheat Sheet. Viitattu 14.8.2021. Saatavilla: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html