Turvallisen sovelluskehityksen käsikirja

[1 Johdanto 5](#_Toc40680185)

[1.1 Millaiseen ohjelmistotuotantoon ohje on suunnattu? 5](#_Toc40680186)

[1.2 Ohjeen kohdeyleisö 6](#_Toc40680187)

[2 Vaatimustenmukaisuus 7](#_Toc40680188)

[3 Tietoturva ja tietosuoja ohjelmistokehitysprosessissa 11](#_Toc40680189)

[3.1 Yleiskuvaus 11](#_Toc40680190)

[3.2 Vaatimukset ohjelmistokehitysprosessille 12](#_Toc40680191)

[3.3 Vastuunjako 13](#_Toc40680192)

[3.3.1 Palvelunomistajan vastuut 15](#_Toc40680193)

[3.3.2 Tuoteomistajan vastuut 15](#_Toc40680194)

[3.3.3 Arkkitehtien vastuut 17](#_Toc40680195)

[3.3.4 Ohjelmistokehittäjien vastuut 18](#_Toc40680196)

[3.3.5 Tietoturva-asiantuntijan vastuut 19](#_Toc40680197)

[3.3.6 Pilviasiantuntijan vastuut 19](#_Toc40680198)

[3.3.7 Käyttöpalvelun toimittajan vastuut 20](#_Toc40680199)

[3.4 Erityishuomioita tiettyihin ohjelmistokehitys- ja kulttuurisiin malleihin 20](#_Toc40680200)

[3.4.1 Scrum 20](#_Toc40680201)

[3.4.2 Kanban 21](#_Toc40680202)

[3.4.3 DevOps 21](#_Toc40680203)

[3.4.4 Automaatio, jatkuva integraatio ja toimitus 22](#_Toc40680204)

[4 Ohjelmistoturvallisuuden seurannan ja ulkoisen tuen malli 22](#_Toc40680205)

[4.1 Jatkuva tietoturvatyön näkyvyys ja työn jälkikäteinen auditoitavuus 22](#_Toc40680206)

[4.2 Tukipalveluiden työnohjausmalli 22](#_Toc40680207)

[5 Tuotantoon viennin vaatimukset (Liite 1) 23](#_Toc40680208)

[5.1 Vaatimusten pakottavuus ja poikkeusten dokumentointi 23](#_Toc40680209)

[5.2 Tuotantoon viennin vaatimusten kriteerit 23](#_Toc40680210)

[5.3 Tuotantoon viennin vaatimukset 24](#_Toc40680211)

[6 Tietoturvallisuuden yleisperiaatteet (Liite 2) 25](#_Toc40680212)

[6.1 Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet 25](#_Toc40680213)

[6.2 Käyttöpalveluympäristöt ja operationaaliset yleisperiaatteet 29](#_Toc40680214)

[6.3 Tietosuojan suunnittelun yleisperiaatteet 31](#_Toc40680215)

[6.4 Auditoitavuuden yleisperiaatteet 33](#_Toc40680216)

[6.5 Kehitysympäristöjen ja tuotantoon viennin yleisperiaatteet 35](#_Toc40680217)

[6.6 Pilviympäristöjen tietoturvaperiaatteet ja -vaatimukset 37](#_Toc40680218)

[6.7 Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet 39](#_Toc40680219)

[6.8 41](#_Toc40680220)

[6.9 Tietoturvatestauksen ja teknisen tarkastuksen yleisperiaatteet 42](#_Toc40680221)

[7 Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista (Liite 3) 44](#_Toc40680222)

[8 Tietoturva-arkkitehtuurin dokumentoinnin vähimmäisvaatimukset (Liite 4) 46](#_Toc40680223)

[9 Uhkamallinnuksen toteutusohje (Liite 5) 47](#_Toc40680224)

[9.1 Uhkamallinnus yleisesti 47](#_Toc40680225)

[9.2 Uhkamallinnuksen aikataulutus 48](#_Toc40680226)

[9.3 Uhkamallinnus käytännössä 48](#_Toc40680227)

[9.4 Uhkamallinnuksen laatu 51](#_Toc40680228)

[9.5 Uhkamallinnus, kun tekninen ratkaisu on vielä tuntematon 51](#_Toc40680229)

[10 Tietosuojan toteutumisen varmistaminen (Liite 6) 53](#_Toc40680230)

[10.1 Tietosuojavaikutusten arviointi yleisesti 53](#_Toc40680231)

[10.2 Tietosuoja-asetuksen vaatima tietosuojavaikutusten arviointi (DPIA) 54](#_Toc40680232)

[11 Tekninen liite (Liite 7, ohje) 55](#_Toc40680233)

[11.1 Palvelimen TLS-asetukset 55](#_Toc40680234)

[11.1.1 Yleistä TLS-asetuksien noudattamisesta 55](#_Toc40680235)

[11.1.2 Protokollaversio 55](#_Toc40680236)

[11.1.3 Varmenteet 56](#_Toc40680237)

[11.1.4 Varmenteiden myöntäjät 57](#_Toc40680238)

[11.1.5 Salausalgoritmit 57](#_Toc40680239)

[11.2 Muut kryptografiset toteutukset 59](#_Toc40680240)

[11.2.1 Satunnaisen tunnisteen luominen eri ohjelmointikielillä 59](#_Toc40680241)

[11.3 HTTP 60](#_Toc40680242)

[11.3.1 Evästeet 60](#_Toc40680243)

[11.3.2 HTTP-otsakkeet selaimille 60](#_Toc40680244)

[11.3.3 Subresource Integrity 65](#_Toc40680245)

[11.3.4 Istuntotunnisteet 66](#_Toc40680246)

[11.4 Lokitus 67](#_Toc40680247)

[11.4.1 Pyyntötunnisteet 68](#_Toc40680248)

[11.5 Docker, Kubernetes ja Terraform 69](#_Toc40680249)

[11.5.1 Isäntäkone 69](#_Toc40680250)

[11.5.2 Docker-kontin asetukset 69](#_Toc40680251)

[11.5.3 Kubernetes 71](#_Toc40680252)

[11.5.4 Terraform 73](#_Toc40680253)

[11.6 Salaisuuksienhallinta 73](#_Toc40680254)

[11.7 Ohjelmointirajapinnat (API) 74](#_Toc40680255)

[11.7.1 API-turvallisuus yleisesti 75](#_Toc40680256)

[11.7.2 Cross-Origin Resource Sharing (CORS) 76](#_Toc40680257)

[11.7.3 CSRF-tunniste 77](#_Toc40680258)

[11.7.4 Parametrit ja luottamuksellinen tieto 77](#_Toc40680259)

[12 Technical Appendix (Appendix 7, instructions) 78](#_Toc40680260)

[12.1 TLS settings of servers 78](#_Toc40680261)

[12.1.1 Recommendations on TLS in general 78](#_Toc40680262)

[12.1.2 Protocol version 79](#_Toc40680263)

[12.1.3 Certificates 79](#_Toc40680264)

[12.1.4 Certificate issuers 80](#_Toc40680265)

[12.1.5 Encryption algorithms 80](#_Toc40680266)

[12.2 Other cryptographic implementations 82](#_Toc40680267)

[12.2.1 Creating random identifiers in various programming languages 83](#_Toc40680268)

[12.3 HTTP 83](#_Toc40680269)

[12.3.1 Cookies 83](#_Toc40680270)

[12.3.2 HTTP headers towards browsers 84](#_Toc40680271)

[12.3.3 Subresource Integrity 89](#_Toc40680272)

[12.3.4 Session identifiers 89](#_Toc40680273)

[12.4 Logging 90](#_Toc40680274)

[12.4.1 The request identifier pattern 92](#_Toc40680275)

[12.5 Docker, Kubernetes and Terraform 92](#_Toc40680276)

[12.5.1 Docker hosts 92](#_Toc40680277)

[12.5.2 Docker containers 93](#_Toc40680278)

[12.5.3 Kubernetes 94](#_Toc40680279)

[12.5.4 Terraform 96](#_Toc40680280)

[12.6 Secrets management 96](#_Toc40680281)

[12.7 Application Programming Interfaces (APIs) 97](#_Toc40680282)

[12.7.1 API security in general 98](#_Toc40680283)

[12.7.2 Cross-Origin Resource Sharing (CORS) 99](#_Toc40680284)

[12.7.3 CSRF tokens 100](#_Toc40680285)

[12.7.4 Confidential data in parameters 100](#_Toc40680286)

# Johdanto

Tässä dokumentissa kuvataan <organisaatio> sovelluskehityksen keskeiset tietoturvallisuuden periaatteet, vaatimukset sekä kontrollit. Dokumenttia käytetään <organisaatio> sovelluskehityksessä.

## Millaiseen ohjelmistotuotantoon ohje on suunnattu?

Ohje on kirjoitettu olettaen, että ohjelmistotuotannossa seurataan nykyaikaiselle ohjelmistotuotannolle ominaisia periaatteita:

1. Kehitettävän kohteen toiminnallisuutta kehitetään iteratiivisesti ja kehityksen kohde voi muuttua nopeastikin (*agile* eli ketterä kehittäminen).
2. Tuotetiimi on suurelta osin autonominen ja yhdessä tuoteomistajan kanssa kantavat liiketoimintavastuun myös tietoturvasta ja joissakin tapauksissa myös tuotannonaikaisista toimenpiteistä (*DevOps* ja *DevSecOps*).
3. Tuotantoon viennin prosessi on automatisoitu, jos kyseessä on pilvipalvelu. Muissa tapauksissa integraation ja tuotantoon viennin automatisoinnille (*continuous integration / delivery / deployment*) ei saisi olla esteitä, vaikka sitä ei vielä olisikaan toteutettu.
4. Jos kyseessä on pilvipalvelu, sen infrastruktuuri kuvataan versiohallittuna koodina (*infrastructure as code* tai *declarative infrastructure*).

Tilanteessa, jossa näitä periaatteita ollaan ottamassa käyttöön, ohjeen voi ottaa käyttöön soveltuvin osin samalla kun periaatteet kehittyvät.

Käsikirjan kuvaamat työnkulut on tarkoitettu käytettäväksi <organisaatio> kokonaisketterän kehityksen viitekehyksessä. Se kuitenkin soveltuu käytettäväksi myös yksittäisissä projekteissa, jotka eivät ole kokonaisketterän kehityksen alaisuudessa.

Mikäli ohjelmistotuotanto on osin ulkoistettu - esimerkiksi ostavalla organisaatiolla on tuoteomistaja, mutta itse ohjelmistokehitys tehdään toimittajan toimesta toisaalla - ohjetta voi soveltaa niiden roolien osalta, jotka ovat ostavan organisaation hallussa. Muut ohjeen osat tulisi vaatia toimittajalta sopimuksin.

Useat ohjeen tietoturvaperiaatteet on kirjoitettu niistä lähtökohdista, että arkkitehtuuri voi olla nk. pilvinatiivi ja palvelu- tai mikropalveluorientoitunut, ja että tuotetiimillä voi olla valtaa ja vastuuta käytönaikaisista toimenpiteistä. Ohjetta voi kuitenkin soveltaa, vaikka näin ei olisikaan.

## Ohjeen kohdeyleisö

Ohjeen kohdeyleisönä ovat seuraavat **roolit**. Riippuen organisaatiosta, sama henkilö voi toimia useissa eri rooleissa. Tässä ohjeessa roolien nimiä käytetään siten kuin ne on kuvattu <organisaatio> roolikuvauksissa, eikä niillä ole välttämättä suoraa vastaavuutta virkanimikkeisiin. Tyypillisessä ohjelmistokehitystiimissä henkilöllä voi olla useita rooleja. Esimerkiksi pääkehittäjällä on vastuita, jotka ulottuvat helposti arkkitehdin vastuualueelle. Itse palvelunsa infrastruktuurista huolehtivan tuotetiimin kehittäjille kuuluu myös käyttöpalveluvastuita.

Tarkempi roolien vastuunjako on kuvattu kappaleessa **Vastuunjako**.

* **Palveluiden omistajat ("liiketoimintaomistajat").** Palveluiden omistajat vastaavat palvelun liiketoiminnan kehittämisestä, palvelun konseptista, strategiasta ja palvelun budjetoinnista. Viime kädessä vastaavat myös tietoturvan ja tietosuojan toteutumisesta resurssoinnin ja budjetoinnin kautta. Tietoturvasta huolehtimiseen kuuluvat palveluiden kriittisyysluokittelu, riskienhallinta sekä lakien ja hyvän tiedonhallintatavan, ohjeiden ja politiikkojen noudattaminen.
* **Tuoteomistajat (product owner).** Tuoteomistaja arvopakettien analyysistä ja pilkkomisesta ominaisuuksiksi ja käyttäjätarinoiksi, sekä tuotetiimin tukemisesta ja tuotantoon vientien koordinoinnista yhdessä tuotetiimin kanssa. Tämän vuoksi hänellä on tällä tasolla myös valtaa ajan ja muiden resurssien käyttöön ja jäännösriskien hyväksyntään. Tuoteomistaja vastaa viime kädessä siitä, että tietoturva- ja tietosuojavaatimukset ovat näkyvissä tehtävälistalla.
* **Arkkitehti** vastaa kehitettävän palvelun tai sovelluksen sopivuudesta sitä ympäröivään tietotekniseen ympäristöön ja tunnistaa läpileikkaavat asiat, joihin monet tietoturva- ja tietosuoja-asiatkin kuuluvat. Arkkitehti merkittävässä roolissa käytettyjen strategisten tekniikkavalintojen suhteen. Arkkitehdin suunnitteluhorisontti on yleensä koko palvelun elinkaaren mittainen, ja häneltä odotetaan näkemystä palvelun toteutukseen myös erityisnäkökulmista, joista tämän dokumentin näkökulmasta mainittakoon tietoturva-arkkitehtuuri ja jatkuvuussuunnittelu.
* **Ohjelmistokehittäjät (tuotetiimeissä).** Ohjelmistokehittäjät toimivat useimmiten pienehköissä tiimeissä ja ovat vastuussa toiminnallisuuden toteuttamisesta, testaamisesta, tuotantoon viennistä. Tietoturvamielessä ohjelmistokehittäjillä on päävastuu tietoturvan teknisestä toteutumisesta tuotehallinnan tarpeiden mukaisesti. Tuotetiimeissä on myös vastuuta ei-rutiininomaisista käytönaikaisista toimenpiteistä, joista tietoturvaan liittyvät esimerkiksi lokien seuraamisen ja monitorointikyvykkyyden järjestäminen, jatkuvuuden hallinta, toipumissuunnittelu ja tuen antaminen tietoturvapoikkeamatilanteissa.
* **Tietoturva-asiantuntijat.** Tuoteomistajat, arkkitehdit ja ohjelmistokehittäjät voivat tarvittaessa käyttää apuna erityisesti tietoturvaan keskittyneitä henkilöitä, joita ovat kehitysprojektiin varatut tietoturvatestaajat, -arkkitehdit tai -asiantuntijat. Tietoturva-asiantuntijat tukevat tyypillisesti tietoturvariskien löytämisessä ja arvioinnissa (esimerkiksi uhkamallinnuksessa), suunnitteluperiaatteiden ja ohjelmakoodin tarkistamisessa, tutkivassa tietoturvatestauksessa ja tietoturvatyökalujen integroimisessa tuotantoon viennin automaatioon. Myös ohjelmistokehitystiimeissä itsessään voi olla tietoturvaan erikoistuneita henkilöitä. Tietoturva-asiantuntijat ovat tässä ohjeessa konsultatiivisessa roolissa, ja lopullinen vastuu tietoturvan toteutumisesta on yllämainituilla rooleilla.
* **Pilvipalveluasiantuntijat ("pilvitiimi").** Pilvitiimin vastuulla on pilvipalvelujen läpileikkaavien tietoturvaratkaisujen kehittäminen. Tuotetiimien on pystyttävä luottamaan pilvitiimin suosittelemien pilviratkaisujen laadukkuuteen tietoturvanäkökulmista.
* **Käyttöpalvelujen toimittajat.** <organisaatio> on voitava luottaa siihen, että käyttöpalvelujen toimittaja takaa tietoturvan toteutumisen omalta osaltaan. Pilvipalveluilla on nk. jaetun vastuun malli, jonka vastuunjakolinja käyttöpalvelun tarjoajan ja tuotetiimin välillä kulkee vaihtelevassa kohdin käytettyä teknologiapinoa (technology stack). Jakolinja voi vaihdella eri käyttöpalvelujen tarjoajien välillä, vaikka käytettävä teknologiapino olisi samankaltainen. Tämän vuoksi pilvipalveluita käytettäessä käyttöpalvelujen tarjoajan ja <organisaatio>:n välisen vastuujaon on oltava selkeästi ymmärretty

# Vaatimustenmukaisuus

Jokaisesta **pakollisesta** vaatimuksesta sekä **valinnaisista toteutettavaksi päätetyistä** vaatimuksista on luotava joko vaatimus tuotteen tehtävälistalle tai vaatimuksen täyttymistä on ajettava jollakin tietoturva-aktiviteetilla, josta jää todiste (kuten esimerkiksi testitulos). Tällä tavoin vaatimuksista syntyy auditoitavissa oleva todiste myöhempää tarvetta varten.

Pakolliset (lakiin perustuvat) vaatimukset otetaan huomioon kokonaisketterän kehitysmallin portfoliokanbanissa. Tuotantoon viennin vaatimukset otetaan huomioon pääosin kokonaisketterän kehityksen suunnittelutasolla. Nämä johtavat todennäköisesti lopulta tehtävälistan tehtäviin, joilla vaatimustenmukaisuus syntyy.

Yleiset tietoturvaperiaatteet on koottu ryhmiksi, joista kussakin roolissa olevan henkilön on omaksuttava omaan rooliinsa liittyvät periaatteet ja huolehtia periaatteiden noudattamisesta oman työnsä viitekehyksessä. Esimerkiksi käyttöliittymäsuunnittelijoiden tulisi tutustua rajapintoja ja käyttöliittymiä koskeviin yleisiin tietoturvavaatimuksiin.

|  |  |  |  |
| --- | --- | --- | --- |
| **Vaatimustyyppi** | **Vaatimuslähde** | **Kuka voi päättää vaatimuksen tekemättä jättämisestä** | **Huomioita** |
| Pakolliset vaatimukset | Kansallinen lainsäädäntö, GDPR [[\*]](https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679), eIDAS [[\*]](https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG) ja tuleva ePrivacy-asetus |  | Huomioidaan portfoliokanbanissa.  Lain ja asetusten vaatimuksia ei voi jättää tekemättä. <organisaatio>:n tietosuojavastaava voi tarjota apuaan niiden tulkinnassa.  Tulkinnat on dokumentoitava Business Outcome -dokumenttiin. |
| Tuotantoon viennin vaatimukset | Liite 1: Tuotantoon viennin vaatimukset | Tuoteomistaja konsultoituaan <organisaatio>:n tietoturvahenkilöstöä | Huomioidaan suunnittelukanbanissa.  Tietoturvavaatimuksesta poikkeaminen on dokumentoitava Business Outcome -dokumenttiin. |
| Yleiset tietoturvaperiaatteet | Liite 2: Tietoturvallisuuden yleisperiaatteet | Tuotetiimi konsultoituaan <organisaatio>:n tietoturvahenkilöstöä | Kussakin roolissa toimiva henkilö tutustuu itselleen olennaisiin periaatteisiin ja huolehtii niiden täyttymisestä oman työnsä kontekstissa.  Tekemättä jättämispäätös on dokumentoitava |
| Lisätietolähteet | Ks. seuraava taulukko | Tuotetiimi | Näistä lähteistä otetaan vaatimuksia mukaan vain tarvittaessa, esimerkiksi kun tarvitaan tietyn aihealueen tarkempaa ohjeistusta. |

Yllä mainittujen vaatimustyyppien lisäksi tietyt ulkoiset lähteet on valittu lisätietolähteiksi. Lisätietolähteet on tarkoitettu ensisijaisiksi lähteiksi tietoturva-aktiviteetteja laajennettaessa esimerkiksi asiakasvaatimusten pohjalta. Tuoteomistajan tulee tunnistaa asiakasvaatimuksista tietoturvatarpeet ja käyttää näitä lähteitä – mahdollisesti tietoturva-asiantuntijan avustuksella – vaatimuksen täyttämiseen.

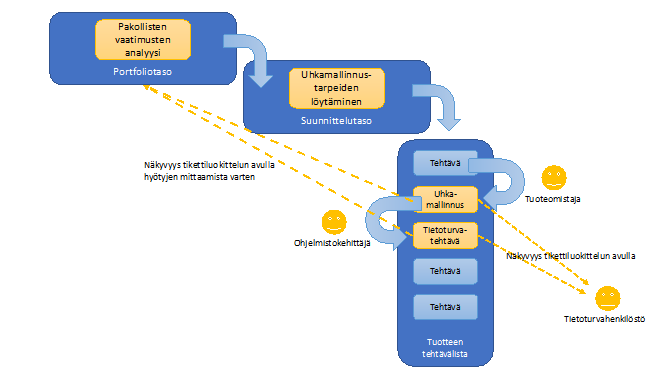
|  |  |  |
| --- | --- | --- |
| **Lisätietolähde** | **Soveltamisala** | **Linkki lähteeseen** |
| OWASP Application Security Verification Standard (ASVS) | Tietoturvatestauksen ja auditoinnin määrittely tehdään lähtökohtaisesti ASVS Level 2 -tason mukaisesti | [OWAP ASVS](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project) |
| MITRE ATT&CK Enterprise Matrix | Käyttöympäristöjen ja sovellusten uhkamallien muodostaminen. Pilvipalveluissa Cloud Matrixin mukaisesti | [MITRE ATT&CK Enterprise Matrix](https://attack.mitre.org/matrices/enterprise/)  [MITRE ATT&CK Cloud Matrix](https://attack.mitre.org/matrices/enterprise/cloud/) |
| CIS Benchmarks | Yksittäisten teknologioiden suojaamis- ja kovennusohjeistus | [CIS Benchmarks](https://www.cisecurity.org/cis-benchmarks/) |
| Pilvipalveluiden turvallisuuden arviointikriteeristö (PiTuKri) | Pilvipalveluiden valinta ja vastuunjaon määrittäminen, kun käsitellään salassa pidettävää, henkilö- tai turvaluokiteltua tietoa | [PiTuKri](https://www.kyberturvallisuuskeskus.fi/fi/toimintamme/ncsa) |
| Guidelines on Data Protection Impact Assessment (DPIA) wp248rev.01 | Tietosuojavaikutusten arviointi | [Wp248rev.01](https://ec.europa.eu/newsroom/article29/item-detail.cfm?item_id=611236) |
| VAHTI Sovelluskehityksen tietoturvaohje 1/2013 | Turvallinen sovelluskehitys niiltä osin, kun tarvitaan lisäohjeistusta tämän käsikirjan lisäksi | [VAHTI 1/2013](https://www.vahtiohje.fi/web/guest/vahti-1/2013-sovelluskehityksen-tietoturvaohje) |

# 

# Tietoturva ja tietosuoja ohjelmistokehitysprosessissa

## Yleiskuvaus

Tietoturva ja tietosuoja tuodaan tuotekehitysprosessiin tekemällä ne näkyväksi työnohjauksessa. Yleisellä tasolla tärkein perusperiaate on esitetty kuvassa.



Tietoturva- ja tietosuojatyö alkaa kokonaisketterän kehitysmallin portfolio- ja suunnittelutasoilla. Näillä tasoilla luodaan edellytykset suurimpien tietoturvavaatimusten noudattamiselle.

Tekstin sujuvuuden vuoksi dokumentissa puhutaan jatkossa pääosin tietoturvatyöstä, vaikka sillä tarkoitetaan myös tietosuojatyötä. Samaten tietoturvatehtävä voi olla minkä tasoinen tehtävä vain, esimerkiksi käyttäjätarina tai yksinkertainen bugikorjaus.

Valtaosa käytännön suunnittelun ja toteutuksen tietoturva- ja tietosuojatyöstä tuodaan näkyväksi tuotteen tehtävälistalla (engl. *product backlog*). Tärkein työtehtävä on **uhkamallinnus** (*threat modelling*), jossa löydetään suurin osa muista tarpeellisista tietoturvatehtävistä.

Uhkamallinnuksen tarve sidotaan yksittäisiin tuotteen tehtävälistan tehtäviin. Suositeltava tapa on valita tiimille sopiva tehtäväkoko, joihin uhkamallinnus liitetään erillisenä tikettinä tai hyväksyntäkriteerinä oletusarvoisesti aina. Tuotetiimi tai tuoteomistaja suorittaa tarvearvioinnin (*triage*) ja voi sen pohjalta tehdä päätöksen, että uhkamallinnusta ei suoriteta. Tämä päätös voidaan dokumentoida esimerkiksi kommenttina tikettiin.

Tuotetiimi suorittaa uhkamallinnuksen ja lisää sen perusteella löytyneet uudet tietoturvatehtävät tehtävälistalle.

Löytyneet tietoturvatehtävät luokitellaan (*label*) yhteisesti sovittuihin luokkiin, ja linkitetään siihen toteutustehtävään, johon ne liittyvät. Näin organisaation tietoturvahenkilöstö voi seurata tietoturvatyön kertymistä ja etenemistä ja mahdollisessa auditointitilanteessa on helpompi löytää todisteet tehdystä työstä.

Tärkeimmät dokumentissa mainitut tietoturvatehtävät ovat uhkamallinnus, tietosuojavaikutusten arviointi, tietoturva-arkkitehtuurin ylläpito ja tietoturvatestaus. Näiden välinen suhde on tiivis. Käytännössä kaikissa tehtävissä on yhteisiä osa-alueita, ja ne hyötyvät toisistaan näiltä osin - sama työ tehdään vain kertaalleen. Suhdetta voidaan kuvata seuraavasti:

Uhkamallinnuksen yhteyteen liittyvät muut tietoturvatehtävät ovat tietoturvatestaus (määrittelee testaustarpeet ja antaa palautetta uhkamallinnuksen laadusta), Tietosuojavaikutusten arviointi (antaa tietovirta- ja turvallisuusnäkökulmat) ja tietoturva-arkkitehtuuri (edellyttää dokumentointia, tuottaa perusteet tai vahvistaa oletukset). 

## Vaatimukset ohjelmistokehitysprosessille

Jotta ohjelmistokehitysprosessi mahdollistaa tietoturvallisen kehityksen ja tietosuojan tason tämän ohjeen mukaisesti, sen on täytettävä joitakin perusvaatimuksia. Ohjelmistokehitys, jossa seuraavat vaatimukset eivät täyty, ei ole tämän tietoturvaohjeen mukainen. Tämä koskee ohjelmistokehitystä myös alihankintatilanteessa, vaikka alihankkijan prosessit ja työkalut eivät olisikaan ostajalle näkyvissä.

Huoltovarmuuskriittisten järjestelmien toteutuksessa näiden vaatimusten toteutumisesta, esimerkiksi kehitys- ja tuotantoon vientityökalujen saatavuudesta, on huolehdittava myös jatkuvuussuunnittelumielessä.

Ohjelmistotuotantoprosessissa on oltava käytössä:

* **Tuotteen tehtävälista** (*product backlog*), joka on priorisoitu luettelo työtehtävistä, joita tuotteen (palvelun tai sovelluksen) toteuttamiseksi on tehtävä. Tehtävillä voi olla eri tasoja, kuten *epic*, kertomus eli *story* ja tehtävä eli *task*. Tässä dokumentissa "tehtävä" voi olla minkä tahansa tasoinen.
* **Tuoteomistajan** (*product owner*) **rooli**, joka on kuvattu aiemmin tämän ohjeen kohderyhmänä.
  + **Tietoturva- ja tietosuojariskien hyväksyminen** on pääosin vastuutettu tälle roolille kuten aiemmassa vaatimusluokittelussa on kuvattu.
  + **Tietoturvatyön priorisointi** suhteessa muuhun työhön on oltava vastuutettu tälle roolille, eikä esimerkiksi erilliselle ohjausryhmälle.
* **Tiketöintiin** **pohjautuva työnohjausjärjestelmä, joka tukee luokituksia (*label* tai *tag*)**. Tämä tarkoittaa sitä, että tehtävälistalla olevien työtehtävien edistymistä voidaan erikseen ja yksittäin seurata, työlistalle ilmestyvät tietoturva- ja tietosuojatehtävät voidaan helposti huomata, ja työtehtävien teosta jää auditoitava jälki. Useimmiten tuotteen tehtävälista ja työnohjausjärjestelmä toimivat samassa työkalussa, mutta välttämättä näin ei ole.
* **Koodin versionhallintajärjestelmä, joka mahdollistaa koodin vertaiskatselmoinnin koordinoidun suorittamisen osana toteutustyötä**. Tällainen voi olla esimerkiksi nk. *pull request* -menettely.
* Pilvipalveluissa **tuotantoon viennin automaatio**(nk. CI/CD-putki), joka mahdollistaa pilvipalvelun infrastruktuurin määrittämisen koodina, tuotantoon viennin mahdollisimman vähäisellä ihmistyöllä sekä jossain määrin mahdollisuuksia tietoturvatestauksen automatisointiin.

Tehtävälistan ja työnohjausjärjestelmän on käytännössä tarpeen elää työkalussa, joka on standardoitu useamman projektin välille, koska esimerkiksi linkkien luominen eri projektien työtehtävien välillä on tärkeää. Tällainen työkalu on <organisaatio>:n käyttämä Jira-järjestelmä.

## Vastuunjako

Ohjeen kohdeyleisö -kappaleessa luetelluilla rooleilla on tarkasti määritellyt vastuualueet. Vastuu seuraa rooleja eikä henkilöitä; henkilöllä voi olla useita rooleja ja toisaalta rooli voi olla jaettu useammalle henkilölle.

Mikäli rooli on jaettu useammalle henkilölle, näiden henkilöiden osalta on sovittava, kuka kantaa päävastuut seuraavassa luetelluista tietoturvavastuista. Mikäli päävastuullista henkilöä ei ole jaetun roolin tilanteessa määritelty, ohjelmistokehitysprosessi ei ole tämän tietoturvaohjeen mukainen.

Vastuutus on viestittävä kohdeyleisötahoille.

|  |  |
| --- | --- |
| **Rooli** | **Vastuut (tarkemmin alla)** |
| Palvelunomistaja | * Tietoturva- ja tietosuojatyön resurssointi ja budjetointi * Palvelun kriittisyysluokittelu ja riskienhallinta * Lakien, hyvän tiedonhallintotavan, politiikkojen ja ohjeiden noudattaminen |
| Tuoteomistaja | * Vaatimuksista poikkeamisesta päättäminen (jäännösriskin hyväksyntä) * Tietoturvavaatimusten näkyvyyden varmistaminen tuotteen tehtävälistalla * Uuden ja muuttuvan toiminnallisuuden tietosuojavaikutusten arviointi * Tietoturva- ja tietosuojatyön priorisointi suhteessa muuhun työhön |
| Arkkitehti | * Yleisten ja läpileikkaavien tietoturvaperiaatteiden noudattaminen suunnittelussa * Yleisen tietoturva-arkkitehtuurin rakentaminen ja ylläpito |
| Ohjelmistokehittäjä | * Tietoturvatyön tiketöinti ja siirto tuotteen tehtävälistalle * Tietoturvatyön suorittaminen, mukaan lukien käytönaikaiset ei-rutiininomaiset tietoturvatehtävät * Rajatuissa tapauksissa vaatimuksista poikkeamisesta päättäminen ja täten jäännösriskin hyväksyminen (yleensä koko tuotetiimi yhdessä) |
| Tietoturva-asiantuntija | * Sovitun tietoturvatyön suorittaminen ja dokumentointi (siltä osin, kun sovittu ohjelmistokehittäjien kanssa) |
| Pilvipalveluasiantuntija (esim. osana pilvi-infrastruktuurin tukitiimiä) | * Tietoturvatyön koordinointi tuotetiimien välillä jaettujen palvelujen osalta * Pilvi-infrastruktuurin parhaiden käytänteiden levittäminen kaikille tiimeille * Jaettujen tietoturvaan liittyvien palveluiden tarjoaminen |
| Käyttöpalvelun toimittaja | * Sovitun tietoturvatyön suorittaminen ja dokumentointi |

### Palvelunomistajan vastuut

#### Tietoturva- ja tietosuojatyön resurssointi ja budjetointi

Palvelunomistajien on otettava huomioon tietoturvatyön tarvitsemat resurssit sekä ajallisessa että rahallisessa budjetoinnissa. Ajallisesti tietoturvatyö aiheuttaa jonkin verran työtä esimerkiksi uhkamallinnuksen muodossa. Budjetointimielessä ulkoisilta tahoilta ostetut tietoturvatarkastukset ovat suurimpia yksittäisiä menoeriä. Resurssointimielessä palvelunomistaja voi vaikuttaa myös ohjeistamalla tietoturvatyön painopistealueita esimerkiksi tuomalla tietoturvatyötä aikaisempaan kehitystyön vaiheeseen ja vaatimalla tietoturvatyön näkyvyyttä tehtävälistoilla, jolloin kustannusten seuranta on mahdollista.

#### Palvelun kriittisyysluokittelu ja riskienhallinta

Palvelunomistajien tulee määritellä palvelun kriittisyys ja pitää yllä kuvaa siihen liittyvistä tietoturvariskeistä. Kriittisyysluokan ja riskien tulee olla sidoksissa palvelun toteutuksen (toiminnallisiin tai ei-toiminnallisiin) vaatimuksiin. Näin voidaan parhaassa tapauksessa johtaa suora vaatimuslinja kriittisyysluokasta ja riskeistä toteutustason tehtäviin.

#### Lakien, hyvän tiedonhallintotavan, politiikkojen ja ohjeiden noudattaminen

Palvelunomistajien tulee olla tietoisia palveluun kohdistuvasta lainsäädännöstä ja muusta regulaatiosta. Kuten myös kriittisyysluokan ja riskien osalta (yllä), lainsäädännön ja politiikojen vaatimusten tulisi olla sidoksissa palvelun (toiminnallisiin tai ei-toiminnallisiin) vaatimuksiin. Näin voidaan parhaassa tapauksessa johtaa suora vaatimuslinja toteutustason tehtäviin.

### Tuoteomistajan vastuut

#### Vaatimuksista poikkeamisesta päättäminen (jäännösriskin hyväksyntä)

Mikäli ohjelmistotuotantoprosessissa on tarve poiketa tietoturva- tai tietosuojavaatimuksista, tuoteomistajan tai joissakin rajatuissa tapauksissa tuotetiimin on tehtävä päätös tästä aiheutuvan riskin hyväksynnästä. Päätös on dokumentoitava (esimerkiksi huomioimalla riskin mahdollisuus tietoturva-arkkitehtuuridokumentissa tai yksittäisten teknisten vaatimusten osalta tiketin kommenteissa). Päätäntävalta on kuvattu tarkemmin kappaleessa Vaatimustenmukaisuus.

#### Tietoturvavaatimusten näkyvyyden varmistaminen tuotteen tehtävälistalla

Tuoteomistajan on varmistettava, että tietoturvavaatimukset on tunnistettu. Vaatimuslähteet on lueteltu kappaleessa **Vaatimustenmukaisuus**. Tuoteomistaja varmistaa, että nämä vaatimukset on muutettu tuotteen tehtävälistalle tehtäviksi. Tehtävät voivat olla eri tasoisia (esimerkiksi kehitysaihio- (*epic-*) tai kertomustasoisia).

Laeista ja asetuksista kumpuavat vaatimukset voivat olla haastavia, kun vaatimuksia muutetaan tehtäviksi. Tämän vuoksi tuotantoon vientivaatimukset (Liite 1: Tuotantoon viennin vaatimukset) ja yleiset tietoturvavaatimukset (Liite 2: Tietoturvallisuuden yleisperiaatteet) on tehty kattamaan useimmat lakitekniset vaatimukset. On kuitenkin erityistilanteita, joissa ne eivät välttämättä riitä, jolloin tilanteen analysointi yhdessä lain tulkitsijan – esimerkiksi organisaation juristin – kanssa on tarpeen.

Joissakin tapauksissa pakollinen vaatimus ei välttämättä ole yksittäinen toiminnallisuus tai tehtävä, vaan esimerkiksi vaatimus tehdä ohjelmistokehitystä tietyllä tavalla, kuten esimerkiksi vaatimus tehdä jatkuvaa koodikatselmointia. Jatkuvia tai yhä uudelleen toistuvia tehtäviä ei voi järkevästi viedä tuotteen tehtävälistalle. Tällöin se voidaan muuttaa prosessikehitystehtäväksi esimerkiksi "kehittäjät määrittelevät koodikatselmointiperiaatteet ja kouluttavat sen kaikille tiimiläisille" tai portfoliotyössä mukaan otetaan tarvittavia sidosryhmiä, jotka muutoin eivät olisi olleet mukana.

Käytännössä monien toiminnallisten vaatimusten vienti tehtävälistalle voidaan tehdä kertaluonteisesti merkittävän uuden kehityshankkeen alussa. Ylläpitovaiheessa tilanne voidaan katselmoida esimerkiksi vuosittain.

Tuoteomistajan varmistettava, että tietoturvaan ja tietosuojaan liittyvät tehtävät on merkitty tikettipohjaisessa työnohjausjärjestelmässä erityisin luokituksin (*label*). Näin organisaation tietoturva- ja tietosuojaorganisaatiot voivat seurata näiden osa-alueiden etenemistä.

Luokituksissa käytettävät avainsanat on kuvattu dokumentissa Liite 3: Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista.

Luokitukset on tarkoitettu ainoastaan auditoitavuus- ja seurantatarkoituksiin. Luokituksilla ei ole vaikutusta priorisointiin eikä työtehtävien elinkaaren vaiheisiin. Niitä ei myöskään käytetä ryhmittelemään työnohjaustikettejä.

#### Uuden ja muuttuvan toiminnallisuuden tietoturvavaikutusten arviointi

Tuoteomistajan on vaadittava toteutuksen hyväksyntäkriteereinä tietoturvariskien selvitys ("uhkamallinnus") sekä mahdollisesti soveltuvan tasoinen tietoturvatarkistus. Uhkamallinnuksen yleinen ohje on Liite 5: Uhkamallinnuksen toteutusohje. Uhkamallinnustehtävä on näkyvissä tuotteen tehtävälistalla.

Mikäli on selvää, että toteutuksessa ei ole tietoturvariskejä, uhkamallinnus voidaan jättää tekemättä. Käytännössä tuoteomistajalla on käytössään lyhyt ja korkealla tasolla kirjoitettu tarkastuslista (Liite 3: Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista), jonka perusteella riskialttiit toiminnot voidaan tunnistaa.

Yleensä tuoteomistajan on helpointa tehdä tämä uhkamallinnustarpeen arviointi osana tuotteen tehtävälistan parannustyötä (*backlog maintenance, backlog grooming*).

#### Uuden ja muuttuvan toiminnallisuuden tietosuojavaikutusten arviointi

Vastaavasti kuin edellä tietoturvan osalta, myös tietosuojaan vaikuttava uusi tai muuttunut toiminnallisuus on tunnistettava. Tämä tehdään samalla tavalla kuin tietoturvan osalta, mutta tietoturvauhkamallinnuksen sijaan tai lisäksi tuoteomistaja luo tuotteen tehtävälistalle vaatimuksen tietosuojavaikutusten arvioinnista (DPIA, *data protection impact assessment*). Käsitteellisesti kyse on saman tyyppisestä työstä. Ohje on dokumentissa Liite 6: Tietosuoja.

#### Tietoturva- ja tietosuojatyön priorisointi suhteessa muuhun työhön

Tuoteomistaja määrittelee tuotteen tehtävälistalla olevien tehtävien keskinäisen prioriteetin. Käytännössä tämä tarkoittaa, että tuoteomistaja määrittelee tehtäviin käytettävissä olevan ajan.

Koska tietoturva- ja tietosuojatyö on näkyvissä tuotteen tehtävälistalla, tämä tarkoittaa sitä, että tuoteomistaja päättää myös tämän työn prioriteeteista ja sitä kautta hyväksyy jäännösriskin, mikäli tietoturvatehtävät väistyvät muiden (toteutus)tehtävien tieltä.

### Arkkitehtien vastuut

#### Yleisten tietoturvaperiaatteiden noudattamisen suunnittelussa

Arkkitehti on vastuussa siitä, että yleisiä arkkitehtuuriperiaatteita noudatetaan ohjelmiston suunnittelutoiminnassa. Arkkitehtuuriperiaatteet on kuvattu kappaleessa Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet.

#### Yleisen tietoturva-arkkitehtuurin määrittely ja ylläpito

Arkkitehtiroolin vastuulla on myös yleisen tietoturva-arkkitehtuurin ylläpito yli kaikkien toteutustiimien. Tähän liittyvät muun muassa tiimien väliset tietoturvaolettamat ja viestiminen käyttöpalveluiden tuottajien kanssa siitä, miten käyttöpalvelut vaikuttavat sovellustason tietoturva-arkkitehtuuriin.

### Ohjelmistokehittäjien vastuut

#### Tietoturvatyön tiketöinti ja siirto tuotteen tehtävälistalle

Jotta tietoturvatyö tulisi näkyväksi ja sille varattaisiin riittävästi aikaa, ohjelmistokehittäjien tärkeimpiä vastuita on varmistua siitä, että tietoturvatyö näkyy työnhallintajärjestelmässä. Kun tietoturvatyö on kirjattu tehtäväksi tikettiin, esimerkiksi työjaksojen sisältöä määriteltäessä on selkeämpää, paljonko aikaa tietoturvatyö vie.

Mikäli ohjelmistokehittäjä havaitsee uutta tietoturvatyötä, jota pitäisi tehdä, heidän vastuullaan on lisätä se tuotteen tehtävälistalle. Tyypillisimmin tämä tapahtuu uhkamallinnuksen seurauksena, jolloin löydettyjen riskien hallitsemiseksi luodaan uusia tehtäviä.

*Näkyvyyden varmistamiseksi ohjelmistokehittäjien tulee myös luokitella tiketit kuten Liite 3: Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista kuvaa.*

#### Tietoturvatyön suorittaminen ja dokumentointi

Vastuu tietoturvatyön suorittamisesta on ohjelmistokehittäjillä.

On selvää, että kaikki ohjelmistokehitystiimit eivät aina pysty suorittamaan kaikkia näitä tietoturvatehtäviä aika- tai kompetenssirajoitteista johtuen. *Vastuu ei kuitenkaan tällöin siirry, vaan rajoite on ratkaistava joko koulutuksella, priorisoinnilla tai tiimin ulkopuolisiin resursseihin tukeutumalla.*

Joissakin tapauksissa ympäröivä organisaatio voi luoda palvelun, jonka käyttö on kustannustehokkaampaa kuin tiimin sisäisten resurssien käyttö. Esimerkiksi tutkivan tietoturvatestauksen asiantuntijaresurssit voi olla tehokkaampaa hankkia keskitetysti koko organisaation käyttöön.

Toteutusaikainen tietoturvatyö koostuu yleisimmin seuraavista tehtävistä:

* + Kehitysympäristöjen ja -työkalujen tietoturvasta vastaaminen (ks. kappale Kehitysympäristöjen ja tuotantoon viennin yleisperiaatteet)
  + Uhkamallinnus (ks. Liite 5: Uhkamallinnuksen toteutusohje ja kappale Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet)
  + Tietosuojavaikutusten arviointi (ks. Liite 6: Tietosuoja)
  + Koodikatselmointi tai staattinen analyysi (eli koneellinen koodikatselmointi)
  + Automaattisten testitapausten toteutus (ks. kappale Tietoturvatestauksen ja teknisen tarkastuksen yleisperiaatteet)
* Tutkiva tietoturvatestaus (ks. kappale Tietoturvatestauksen ja teknisen tarkastuksen yleisperiaatteet)
* Ohjelmistoriippuvuuksien haavoittuvuusseuranta ja paikkaus (ks. kappale Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet)
* Tuotantoon viennin tietoturvadokumentaation ylläpito (ks. kappale Kehitysympäristöjen ja tuotantoon viennin yleisperiaatteet)
* Palvelun (mukaan lukien tuotetiimin ylläpitämän infrastruktuurin) monitoroinnin ja tietoturvapoikkeamiin reagoinnin järjestäminen

Tehdyn tietoturvatyön dokumentointi tehdään ensisijaisesti tiketöintijärjestelmään ja versiohallittuihin dokumentteihin (esimerkiksi wiki).

Mikäli ohjelmistokehittäjät tukeutuvat joiltakin osin käyttöpalvelutoimittajan komponentteihin (esimerkiksi käyttöjärjestelmään tai pilvipalvelun orkestrointirajapintaan), ohjelmistokehittäjien vastuulla on määritellä vastuunjaon raja käyttöpalvelutoimittajaan nähden. Käyttöpalvelutoimittajan vastuualueista on sovittava sopimuksellisesti.

### Tietoturva-asiantuntijan vastuut

#### Sovitun tietoturvatyön suorittaminen ja dokumentointi

Tietoturva-asiantuntijan vastuulla on häneltä pyydetyn tietoturvatyön toteuttaminen riskilähtöisesti ja parhaan asiantuntemuksensa mukaisesti. Hänen on myös dokumentoitava tietoturvatyön tulokset. Dokumentointi tehdään ensisijaisesti tiketöintijärjestelmään ja versiohallittuihin dokumentteihin (esimerkiksi wiki). Tietoturvatestaukseen ja -tarkastukseen liittyvät vaatimukset on lueteltu kappaleessa Tietoturvatestauksen ja teknisen tarkastuksen yleisperiaatteet.

### Pilviasiantuntijan vastuut

#### Tietoturvatyön koordinointi tuotetiimien välillä jaettujen palveluiden osalta

Useamman tuotetiimin (kehitystiimin) käyttäessä jaettua tai keskitetysti suositeltua pilvipalvelua pilviasiantuntijan (pilvitiimin) vastuulla on koordinoida tietoturvatyö tuotetiimien välillä.

Yksinkertaisimmillaan tämä voi olla esimerkiksi tietoturvallinen käyttö- ja kovennusohje tai tietoturvatarkastuksen tarjoaminen tietylle pilvipalvelulle.

Mikäli useampi tiimi käyttää jaettua palvelua, pilvitiimin vastuulla on koordinoida, etteivät tiimit vahingossa riko toistensa tietoturvaolettamia.

#### Pilvi-infrastruktuurin parhaiden käytänteiden levittäminen kaikille tiimeille

Pilvi-infrastruktuurin parhaat käytänteet kehittyvät varsin nopeasti ja pilvipalveluntarjoajilla on paljon uusia (tietoturva)palveluita ja materiaalia. Näiden seuraaminen erikseen jokaisen tuotetiimin (kehitystiimin) osalta ei ole välttämättä järkevää.

Pilviasiantuntijan (pilvitiimin) vastuulla on identifioida olennaisimmat ja parhaat käytänteet, esimerkiksi pilvipalveluntarjoajan uudet tietoturvaan liittyvät asetukset ja referenssiarkkitehtuurit ja levittää näistä helposti ymmärrettävää tietoa tuotetiimeille.

#### Jaettujen tietoturvaan liittyvien palveluiden tarjoaminen

Kaikkia tietoturvaan liittyviä palveluita ei ole tarkoituksenmukaista tuottaa jokaisessa tuotetiimissä (kehitystiimissä), vaikka ne olisivatkin DevOps-tiimejä. Tällaisia ovat esimerkiksi lokien keräyksen tekninen järjestäminen ja palvelunestohyökkäyssuojaus kaikille organisaation palveluille.

Pilviasiantuntijan (pilvitiimin) vastuulla on identifioida ne tietoturvapalvelut, jotka kannattaa tuottaa keskitetysti joko kustannus-, tehokkuus- tai tietoturvasyistä ja tarjota niitä tuotetiimien käyttöön.

Jaetuissa palveluissa on kuitenkin huomioitava, että tuotetiimi ymmärtää oman vastuunsa rajat. Esimerkiksi lokien keräyksessä niiden tekninen keräys voidaan määritellä yhteisesti, mutta lokien sisältö ja niiden tulkintaohje esimerkiksi <organisaatio>:n SOCille ovat edelleen tuotetiimien vastuulla, koska ne ovat palvelukohtaisia.

### Käyttöpalvelun toimittajan vastuut

#### Sovitun tietoturvatyön suorittaminen ja dokumentointi

Käyttöpalvelun toimittajilta odotetaan, että se suorittaa käyttöpalveluja hankkivan organisaation kanssa sovitun tietoturvatyön. Viitteellisesti nämä kattavat kappaleissa Auditoitavuuden yleisperiaatteet ja Käyttöpalveluympäristöt ja operationaaliset yleisperiaatteet luetellut asiat.

Muut käyttöpalvelutoimittajan osalle siirretyt vastuut on määriteltävä sopimuksellisesti.

## Erityishuomioita tiettyihin ohjelmistokehitys- ja kulttuurisiin malleihin

Ketterää ja *lean*-ohjelmistokehitystä voidaan tehdä useilla eri malleilla. Tässä kappaleessa on lyhyesti kuvattu, miten yllä mainitut vastuut voidaan ottaa mukaan kussakin mallissa.

### Scrum

Uhkamallinnuksen tarpeen arviointi voidaan sisällyttää tehtävien *Definition of Ready* -kriteereihin, jos niitä käytetään esiehtona työn aloittamiselle.

Tietoturvatyötehtävät sisällytetään Scrum-työjaksoihin (*sprint*) aivan kuin ne olisivat toiminnallisten vaatimusten toteutusta. Esimerkiksi uhkamallinnuksen ja tutkivan tietoturvatestauksen tarve tuodaan esille tehtävänä tai tehtävään sidottuna hyväksyntäkriteerinä.

Yleisiä kaikkiin kehitystehtäviin liittyviä *Definition of Done* -kriteereitä ei käytetä tietoturvatyön ohjaamiseen, koska tällöin niille ei erikseen varata aikaa. Ohjelmistokehittäjät saavat toki omatoimisesti tehdä tietoturvaan liittyviä Definition of Done -kriteerejä oman laadunvalvontansa tueksi.

Jos tietoturvatyö luodaan tehtävälistalle erillisenä tehtävänä, se merkitään yleensä riippuvuudeksi toiminnallisuuden toteutuksen valmistumiselle. Tämä tehdään tyypillisessä tiketöintijärjestelmässä linkittämällä tehtävät toisiinsa.

Koska Scrum-työjakson suunnitteluvaiheessa (*sprint planning*) on yleensä sitouduttava tiettyyn työmäärään, esimerkiksi uhkamallinnustyö kannattaa yleensä tehdä aiemmassa työjaksossa kuin missä varsinainen toiminnallisuus toteutetaan. Uhkamallinnus saattaa muuttaa työmääräarvioita voimakkaastikin.

### Kanban

Tietoturvatyötehtävien hallinta noudattelee Scrumin mallia (yllä). Koska kanban-menetelmässä ei ole työjakson pituuteen perustuvaa työmäärän (*Work in Progress*) rajoitusta, esimerkiksi uhkamallinnus on luonnollisemmin toteutettavissa toiminnallisuuden hyväksyntäkriteerinä sen sijaan, että se olisi erillinen työtehtävä.

### DevOps

Kulttuurisena käsitteenä *DevOps* eli autonomisten kehitys- ja tuotantoaikaisista toimenpiteistä vastaavien tiimien luominen on tietoturvan kannalta lähes aina positiivinen asia. DevOps-kypsyyden parantaminen on tämän vuoksi suositeltavaa.

Tietoturvan yhteydessä käytetään enenevässä määrin termiä *DevSecOps*, jolla viitataan siihen, että tuotetiimillä on vastuuta käytönaikaisten toimenpiteiden (*Ops*) lisäksi myös tietoturvasta (*Sec*).

Pilvipalveluiden osalta tuotetiimin vastuuttaminen tietoturvan käytönaikaiseen havainnointiin ja reagointiin voi olla kuormittavaa ja vaatia kehittyneitä työkaluja. *Vastuuta ei voi siirtää tuotetiimille ilman riittävää tukea ja resursseja, joten organisaation on löydettävä kehitystiimeille sopiva malli, jossa tiimit voivat tukeutua kaikille yhteisiin malleihin.* Esimerkiksi pilvipalveluiden standardiratkaisujen käyttöönottoa ja tietoturvan monitorointi- ja reagointimallien luomista voidaan helpottaa keskitetyn pilvipalvelutiimin toimesta.

### Automaatio, jatkuva integraatio ja toimitus

Ne tietoturvatehtävät, jotka ovat luonteeltaan nopeasti toistuvia tai jatkuvia, kuten staattinen analyysi, automaattiset haavoittuvuusskannaukset, riippuvuuksien haavoittuvuusseuranta ja käytönaikainen tietoturvaan liittyvä monitorointi, pitäisi mahdollisuuksien mukaan automatisoida.

Kun automatiikka on riittävän kypsää, jatkuvan integraation (*continuous integration* tai *CI*) järjestelmään voidaan rakentaa logiikkaa, joka estää tuotantoon viennin, jos automatiikka huomaa mahdollisen tietoturvariskin.

Tietoturvatehtävien automatisointi on itsessään työtä, ja sikäli kun ohjelmistokehittäjät toteuttavat automatisoinnin, tämäkin kehitystyö ohjataan tuotteen tehtävälistan kautta. Tällöin sen kustannukset ja priorisointi tulevat käsitellyksi.

# Ohjelmistoturvallisuuden seurannan ja ulkoisen tuen malli

## Jatkuva tietoturvatyön näkyvyys ja työn jälkikäteinen auditoitavuus

Jotta tietoturvatyön edistymistä voidaan seurata reaaliaikaisesti, tietoturvatyötä ohjaavat tehtävät luokitellaan ja ne linkitetään siihen toiminnallisuuteen, mihin ne liittyvät. Organisaation tietoturva- ja tietosuoja-toiminnot voivat tällöin identifioida ne projektit, joissa luokittelua ei tapahdu ja reagoida asiaan aikaisessa vaiheessa. Tietoturva-aktiviteettien toteutumisen näkyvyys kokonaisketterän mallin portfoliotasolle voidaan tehdä samalla tavalla.

Luokittelu on kuvattu kappaleessa Liite 3: Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista. Linkitys tehdään käytetyn tiketöintijärjestelmän linkkejä käyttäen.

Jos ohjelmistoja teetetään ulkoisena hankintana, jossa kehitystä ei tehdä ostajan työnohjauksen alaisuudessa, ostaja voi halutessaan pyytää näkymää työnohjausjärjestelmään ja vaatia sopimuksellisesti vastaavan luokittelun. Näin ostaja voi varmistua siitä, että tietoturva- ja tietosuojatyötä tehdään jatkuvasti koko kehitystyön ajan.

## Tukipalveluiden työnohjausmalli

Ohjelmistokehittäjille, arkkitehdeille ja tuoteomistajille voidaan tarjota erilaisia tietoturvan ja tietosuojan tukipalveluita. Tyypillisin näistä on tutkiva tietoturvatestaus, joka tehdään usein ohjelmistotiimin ulkoisia resursseja käyttäen.

Organisaatio luo tukipalveluiden toimittamista varten oman erillisen tehtävälistansa (*backlog*), johon kuka tahansa organisaation kehitystehtävissä toimiva henkilö voi lisätä tarvitsemansa tukipalvelutarpeen kuvauksen.

Mikäli tehtävälistaa ylläpidetään samassa tiketöintijärjestelmässä kuin tuotteen tehtävälistaa, tukipalvelupyyntö voidaan linkittää ristiin näiden tehtävälistojen välillä. Näin tukipalveluiden tarve tehdään näkyväksi.

Tuotetiimiä ympäröivä organisaatio tuottaa tukipalveluita keskitetysti tämän tehtävälistan pohjalta ja määrittelee prioriteettijärjestyksen toimitettaville tukipalveluille.

# Tuotantoon viennin vaatimukset (Liite 1)

## Vaatimusten pakottavuus ja poikkeusten dokumentointi

Seuraavat hyväksyntäkriteerit on täytettävä ennen kuin kehitettävän toiminnallisuuden saa viedä tuotantoon. *Näistä kriteereistä poikkeaminen on sallittua vain tuoteomistajan erillisellä päätöksellä.*

*Päätös on dokumentoitava.* Dokumentointi tulisi tehdä tavalla, joka mahdollistaa päätöksen perusteiden löytämisen myöhemmin (esimerkiksi tietoturvapoikkeaman jälkihoidossa) ja ennen kaikkea niin, että mahdollinen jäännösriski otetaan huomioon myöhemmässä toiminnassa. Mahdollisuuksia ovat esimerkiksi

* Kirjaus yleiseen riskirekisteriin, jolloin jäännösriskiä voidaan seurata riskienhallintaprosessissa
* Kirjaus (tietoturva-)arkkitehtuuridokumenttiin, jolloin puute pysyy näkyvissä ja voidaan ottaa huomioon jatkosuunnittelussa
* Yksittäisissä tehtävissä kirjaus tikettiin esimerkiksi avaamalla asiasta tiketti ja viemällä se työjonoon tulevaisuudessa toteutettavaksi

## Tuotantoon viennin vaatimusten kriteerit

Tuotantoon viennin vaatimukset otetaan kokonaisketterässä kehityksessä mukaan suunnittelukanbanissa.

Jotta jatkuva tuotantoon vienti on mahdollista, näiden vaatimusten on oltava sellaisia, jotka voidaan joko

1. täyttää suunnittelu- tai kehitysaikaisesti ennen kuin koodi menee automaattiseen integraatioon tai tuotantoon vientiin; tai
2. voidaan toteuttaa kertaluontoisesti vain yhden kerran koko järjestelmän elinaikana, tai
3. voidaan toteuttaa automaattisin työkaluin osana tuotantoon vientiä.

Dokumentointi- ja auditoitavuusvaatimukset tehdään ensisijaisesti joko työnohjauksessa käytetyn tiketin sisällössä tai tilassa, versiohallitussa wikisivussa tai automaattisen työkalun tuottamassa tiedossa, joka tallennetaan. Erillisiä vaatimustenmukaisuusdokumentteja kirjoitetaan vain, jos niiden tuottamiselle on erillinen tehtävä tehtävälistalla.

## Tuotantoon viennin vaatimukset

* Ohjelmiston yleisarkkitehtuurille ja pääasiallisille toiminnallisuuksille on toteutettu uhkamallinnus noudattaen liitteen 5 ohjeita soveltuvin osin ja mahdollinen jäännösriski on dokumentoidusti hyväksytty. Uhkamallinnuksen toteutus ja sen tulokset on tiketöity. Tämä vaatimus voi täyttyä yhdistelemällä kertaluontoisia uhkamallinnuksia ja tämän prosessin kuvaamia toiminnallisuuskohtaisia pienemmistä uhkamallinnuksista.
* Ohjelmistolle on suoritettu tietosuojavaikutusten arviointi (tietosuoja-asetuksen tarkoittama *Data Protection Impact* Assessment, DPIA), mikäli se on lakisääteisesti tarpeellista. Suoritettu tietosuojavaikutusten arviointi ja sen tulokset ovat auditoitavissa.
* Mikäli ohjelmisto käsittelee henkilötietoja, ohjelmistosta on olemassa ajantasainen kuvaus henkilötietovirroista ja niiden tallennuspaikoista, henkilötietotyyppikohtaisesti eriteltyinä. Tämä vaatimus on erillinen tietosuojavaikutusten arvioinnista ja sen tulee toteutua kaikille henkilötietoja käsitteleville ohjelmistoille.
* Ohjelmiston tietoturvatestaustarpeet on määritelty ja ne on dokumentoitu. Ensisijaisesti testaustarpeet dokumentoidaan tiketteinä tuotteen tehtävälistalla.
* Ohjelmistolle on tehty säännöllinen *game day*. *Game day*ssä tunnistetaan ja havaitaan (uusia) riskejä, selvitetään ohjelmiston palautumiskyvykkyys ja koulutetaan tiimien jäseniä pilvioperointiin ja toimintaohjeiden (*playbook* tai *runbook*) noudattamiseen. *Game day*n konseptin omistaa kirjoitushetkellä pilvitiimi.
* Ohjelmistosta on olemassa ajantasainen tietoturva-arkkitehtuurikuvaus, joka täyttää minimivaatimukseltaan kappaleen Liite 4: Tietoturva-arkkitehtuurin dokumentoinnin vähimmäisvaatimukset vaatimukset.
* Muutosoikeudet käyttöpalvelujen ylläpitoon on rajattu vain henkilöille, joiden on tarpeen tehdä tuotannonaikaisia toimenpiteitä. Tämä sisältää ajoympäristöjen virtuaalikoneet, käyttöjärjestelmät, orkestrointijärjestelmät, koodivarastot (*repositories*), automaattisen tuotantoon viennin järjestelmät ja verkon konfiguraatiot. Jos infrastruktuuri on kuvattu koodina, myös versionhallinnan katsotaan kuuluvaksi käyttöpalvelujen ylläpitoon.
* Käyttöpalvelujen ylläpitoon liittyvien ympäristöjen (yllä) omistajuus ja päivitysperiaatteet on vastuutettu ja kuvattu selkeästi.
* Käyttöpalvelut täyttävät niille erikseen asetetut kovennusvaatimukset noudatellen palvelutoimittajan ja palvelun käyttäjän vastuurajoja. Ellei kovennusvaatimuksia ole erikseen määritelty, lähteinä käytetään ensisijaisesti CIS Benchmark -dokumentteja.
* Ohjelmiston tuottamat lokit siirretään keskitettyyn lokijärjestelmään.
* Ohjelmiston ajallisesti rajoitetut komponentit (kuten varmenteet) sekä salaisuuksien hallinta on dokumentoitu, ja niiden hallinta on siirretty mahdollisen tuotantoa hoitavan tahon haltuun.

**Lähteet:**

* Tietosuojavaikutusten arviointitarpeen määrittelee yleisen tietosuoja-asetuksen 35. artikla sekä Article 29 Data Protection Working Partyn ohje WP248 ‘Guidelines on Data Protection Impact Assessment (DPIA) and determining whether processing is “likely to result in a high risk” for the purposes of Regulation 2016/679’.
* Center for Internet Security Benchmarks: https://www.cisecurity.org/cis-benchmarks/

# Tietoturvallisuuden yleisperiaatteet (Liite 2)

## Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Järjestelmän ei-julkinen tieto on tunnistettu ja sen käsittelyperiaatteet on määritelty | Tunnistetaan, mitä ei-julkista tietoa järjestelmässä käsitellään, ja sille määritellään käsittelyn tietoturva- ja tietosuojavaatimukset. Näiden vaatimusten tulisi päätyä tuotteen tehtävälistalle, jotta vaatimukset voidaan ottaa huomioon uhkamallinnuksessa. |
| Palvelun keskeiset käyttötilanteet on tunnistettu | Tunnistetaan palvelun käyttötapaukset (*use case*) ja erityisesti tietosuojamielessä käyttökokemus (*user experience*). Erityisen kiinnostavia ovat poikkihallinnolliset käyttötilanteet, joissa tietoa siirretään organisaatioiden välillä, sekä ylläpito- ja hallintakäyttötilanteet. Tiedot välitetään ohjelmistokehittäjille käyttötapauskuvauksissa, jotta vaatimukset voidaan ottaa huomioon uhkamallinnuksessa. |
| Arkkitehtuurin tietoturva ja tietosuoja on dokumentoitu | Ks. Liite 4: Tietoturva-arkkitehtuurin dokumentoinnin vähimmäisvaatimukset |
| Arkkitehtuuri on modulaarinen | Arkkitehtuuri rakentuu määritellyin rajapinnoin eriytetyistä komponenteista tai palveluista, jotka toteuttavat itsenäisesti tietyn toiminnallisuuden tai tarjoavat pääsyn palvelun tarvitsemaan tietoon.  **Esimerkki:** Mikropalvelut (*microservices*) ovat yksi tapa tuottaa tämänkaltainen arkkitehtuuri. Niitä kutsutaan tarkasti määritellyn rajapinnan kautta, eikä esimerkiksi niiden hallitsemiin tietoihin ole pääsyä kiertoteitse. |
| Arkkitehtuuri eriyttää palveluntarjoajista riippuvat osat selkeillä rajapinnoilla | Jatkuvuussuunnittelun helpottamiseksi erityisesti pilvipalveluiden tietystä tarjoajasta riippuva toiminnallisuus on eriytettävä selkeillä rajapinnoilla.  **Esimerkki:** Palvelussa päätetään käyttää pilvialustan omaa *serverless*-toteutusta tai vaikkapa valmista pilvitietokantapalvelua. Arkkitehtuuri on suunniteltava niin, että näiden korvaaminen eri toteutuksella ei pakota muuttamaan koko ympäröivää arkkitehtuuria. |
| Tietojen käsittely ja rajapintojen syntaktinen monimutkaisuus on minimoitava | Palvelun osan tulee toteuttaa vain halutun toiminnallisuuden kannalta välttämättömät toiminnot. Tämä pätee sekä tiedon käsittelyyn että rajapintojen toiminnallisuuteen ja ilmaisuvoimaan.  Henkilötietoja saa käsitellä vain niin siinä laajuudessa, kuin palvelun tekniseen toteuttamiseen vaaditaan.  **Esimerkki:** REST-rajapinnat määritellään OpenAPI-dokumentteina ja niiden kautta vastaanotetun tiedon syntaktinen oikeellisuus varmistetaan tiukasti. Esimerkiksi DDD (*Domain-Driven Design*) saattaa auttaa tässä ajattelussa, ks. [Johnsson et al., *Secure b*y Design (Manning Publications, 2019)](https://www.manning.com/books/secure-by-design). |
| Arkkitehtuurissa on selvää, mikä osa on vastuussa tietojen pysyvästä tallennuksesta | Se palvelun osa, joka tallentaa tietoja pysyvästi (*persistence*), on myös vastuussa tietojensa pääsynhallinnasta ja niiden poistamisesta elinkaaren aikana. Näiden osien määrä kannattaa minimoida.  **Esimerkki:** Arkkitehtuurisuunnittelu ja teknologiavalinnat mahdollistavat sen, että palveluiden osia voidaan ajaa *read-only* -tiedostojärjestelmistä niin laajalti kuin mahdollista. Tällöin tietoja käsitellään vain muistinvaraisesti. |
| Arkkitehtuuriin on luotava selvät luottamusalueet, jotka on dokumentoitu tietoturva-arkkitehtuurissa | Jokaisen komponentin osalta on määriteltävä, mihin muihin komponentteihin se luottaa, ja tämä määrittelee luottamusalueet. Eri luottamusalueiden välille on suunniteltava verkko- ja ajoympäristöjen riittävä erottaminen ja tietovirtojen sisällön oikeellisuuden tarkistus ja todennus.  Komponentit, joilla on rajapinta loppukäyttäjään tai kolmannen osapuolen palveluun eriytetään omaan luottamusalueeseensa.  Jos luottamusalueilla on keskinäinen hierarkia (esimerkiksi suojaustaso), suojaustasojen väliset eheys- ja luottamuksellisuusvaatimukset on määriteltävä.  **Esimerkki:** Kubernetes-klusterissa ajettavat mikropalvelut luokitellaan eri nimiavaruuksiin esimerkiksi sen mukaan, palvelevatko ne suoraan Internetissä olevaa asiakasta vai muita taustapalveluita ja onko mikropalveluilla hallussaan pääsyavaimia taustapalveluihin (esim. tietokantoihin). Niiden välinen kommunikointi rajataan esimerkiksi NetworkPolicy-määrittelyllä siten, että vain toiminnan kannalta oleelliset yhteydet ovat mahdollisia. |
| Tietovarannot on eriytettävä luottamusalueiden välillä | Kaikki tietovirrat luottamusalueiden välillä on toteutettava rajapinnoilla. Suorat tietovarantopääsyt luottamusalueelta toisille (esimerkiksi vapaamuotoiset tietokantakyselyt tai tiedostojärjestelmän luku) on arkkitehtuurin tasolla estettävä.  Luottamusalueella, jolla on suora yhteys loppukäyttäjään tai kolmannen osapuolen palveluun ei saa sijaita tietovarantoja, vaan tiedot on noudettava tarvittaessa rajapinnan kautta.  **Esimerkki:** Useat erilaiset mikropalvelut tarvitsevat tietoa samasta tietokannasta. Sen sijaan, että kaikille eri mikropalveluille annetaan suora tietokantapääsy, ne kutsuvat erillistä rajapintaa, joka tarjoaa niille tarpeellisen pääsyn (mutta ei mitään muuta). Itse tietokantaan annetaan pääsy vain tämän rajapinnan toteutukselle. |
| Tietovirtojen tietoturva- ja tietosuojavastuut on selkeästi määritelty | Vastuu tietovirtojen sisällön oikeellisuuden tarkistamisesta ja niihin luottamisesta on aina vastaanottajalla.  Tietosuojan osalta vastuu siitä, että tietoja ei luovuteta toiselle komponentille lain vastaisesti, on aina lähettäjällä.  **Esimerkki:** Jos komponentti hakee tietoa rajapinnasta esitettäväksi esimerkiksi käyttöliittymässä, on rajapinnan kutsujan vastuulla varmistaa, että tiedon sisällyttäminen esimerkiksi web-sivun kontekstissa on turvallista. Tietokannalla tai sinne tietoa tallentavilla tahoilla ei voi olla tietoa kaikista niistä konteksteista, joissa tietoa joskus tullaan esittämään ja mitä erityistarpeita esimerkiksi syntaksille tuolloin on.  **Esimerkki:** Jos henkilötietoja lähetetään kolmannen osapuolen rajapintaan, lähettäjän on poistettava siitä ne tiedot, joita ei teknisesti tarvita. Jos vastuu tästä siirrettäisiin vastaanottajalle, on se lakiteknisesti jo liian myöhäistä. |
| Järjestelmän saatavuustavoitteiden on oltava selkeitä vaatimuksia | Saatavuustavoitteet on määriteltävä ja niille on löydettävä tekninen ratkaisu esimerkiksi hajauttamalla ja automaattisella skaalautumisella. Erityishuomiota on kiinnitettävä vanhojen järjestelmien integraatiopisteisiin ja komponentteihin, joita voi käsitteellisesti olla vain yksi kappale. |
| Valmiita toteutuksia ja ohjelmistoriippuvuuksia käytettäessä tietoturvavastuu on määriteltävä | Mikäli käytetään jonkin muun tahon kirjoittamia ohjelmistoja, tietoturvavastuu kuuluu sille komponentille, joka valmista toteutusta käyttää. Vastuun voi siirtää sopimuksellisesti, mikäli käyttö perustuu sopimukseen. Tyypillisessä avoimen lähdekoodin tapauksessa käyttö perustuu vain rajoitettuun lisenssiin, jolloin tietoturvavastuuta ei voida siirtää.  **Esimerkki:** Halutaan ottaa käyttöön kirjasto, joka säästää kehitysaikaa arviolta henkilötyökuukauden. Arvion mukaan kirjaston tietoturvatestaus ja haavoittuvuusseuranta kuitenkin aiheuttaisi elinkaaren aikana tätä enemmän lisätyötä, joten kirjastoa ei oteta vain tämän projektin vuoksi käyttöön. |
| Kolmannen osapuolen komponenttien laatu on selvitettävä ja seurattava | Ohjelmistokehityksessä käytettävien, ulkopuolelta hankittavien (esim. avoin lähdekoodi tai valmis ostettu komponentti) haavoittuvuustilannetta on seurattava jatkuvasti. Tämä tulisi tehdä automatisoidulla järjestelyllä. Käyttöön otettaessa ulkopuolisen komponentin tausta ja ylläpitotilanne on selvitettävä ja siihen liittyvät riskit on hallittava.  **Esimerkki:** Halutaan ottaa käyttöön kirjasto, mutta sen taustatarkistuksessa ilmenee, että sillä on vain yksi aktiivinen kehittäjä ja tietoturvaongelmia on korjattu ilman, että niille on haettu CVE-numeroa tai mainittu versiodokumentaatiossa. Tutkitaan, olisiko muita vaihtoehtoja olemassa. |

## 

## Käyttöpalveluympäristöt ja operationaaliset yleisperiaatteet

|  |  |  |
| --- | --- | --- |
| **Periaate** | **Kuvaus** | **Pilviympäristöjen erityishuomiot** |
| Käyttöpalveluympäristön on toteutettava turvallisuusarkkitehtuurin luottamusalueet | Tietoturva-arkkitehtuurissa kuvattujen luottamusalueiden (ks. Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet) on oltava näkyvissä ja teknisesti toteutettu verkko- ja ajoympäristöjen toteutuksessa, sikäli kun niiden toteutus on tällä tasolla mahdollista. | Luottamusalueiden toteutuksesta Kubernetes-klusterissa ks. tekninen liite. |
| Kehitystä tukeva pääsy tuotantoaikaisiin käyttöpalveluympäristöihin on toteutettava turvallisesti | Mikäli kehitystä varten on avattava pääsy tuotantoaikaisiin käyttöpalveluympäristöihin, pääsyn on oltava avoin vain erikseen nimetyille henkilöille, tehdyistä toimenpiteistä on jäätävä auditoitava merkintä, ja pääsy on rajattava verkkoteknisesti pienimpään mahdolliseen. | Forensiikka- ja viantutkintatapauksissa pitää ensisijaisesti luoda pilviresurssista kopio, jota voidaan tutkia. |
| Auditointilokien seuraaminen on järjestettävä | Auditointilokeja (ks. Auditoitavuuden yleisperiaatteet) on seurattava automaattisin hälytyksin. Hälytykset määritellään perustuen uhkamallinnukseen ja tärkeimpiin käyttötapauksiin (ks. Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet ja Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet). |  |
| Palvelun saatavuuden monitorointi on järjestettävä | Palvelun saatavuutta on monitoroitava sekä palvelun käyttäjille tarkoitetusta rajapinnasta että monitorointirajapintojen kautta (ks. Auditoitavuuden yleisperiaatteet). Saatavuusongelmiin on määriteltävä soveltuvat hälytykset. |  |
| Ajoympäristöt on kovennettava | Palvelun sovellusten ajamiseen käytettävät ajoympäristöt on kovennettava. Suositeltava kovennustapa on CIS-kovennusohjeiden noudattaminen soveltuvin osin, sekä sovellusten ajaminen rajoitetuissa ajoympäristöissä (esim. kontit) mahdollisuuksien mukaan. | Docker- ja Kubernetes-ympäristöistä ks. tekninen liite. |
| Internetiin avoinna olevien sovelluspalveluiden edustalla tulee olla sisällönjakeluverkko (*content delivery network*), välityspalvelin tai kuormantasain | Sovelluksen itsensä tulee olla päätepiste mahdollisimman ohuelle protokollapinolle, jotta mahdolliset tuntemattomat haavoittuvuudet näissä protokollissa eivät vaikuta suoraan sovelluksen ajoympäristöön.  **Esimerkki:** Sovelluspalvelimen eteen laitetaan kuormantasain, joka terminoi TLS-kerroksen ja sen alapuolella olevat protokollakerrokset. Kuormantasain välittää selaimen ulkoisen IP-osoitteen sovellukselle lokitusta varten. | Kuormantasaukseen ja jakeluverkkona käytetään mieluiten pilvipalvelun tarjoamia palveluita, joilla on tunnetusti hyvä suorituskyky.  Ks. tekninen ohje TLS-protokollasta ja lokien luonnista. |
| Liikenteen salaus | Kaikki julkisen Internetin yli siirrettävä tieto on salattava kulloinkin vahvana pidettävällä tavalla. Jos muuta ei ole määritelty, soveltuva tapa on uusin TLS-protokollan versio Kyberturvallisuuskeskuksen konfiguraatio-ohjetta noudattaen. | Mikäli käytetty pilvipalvelukomponentti (esimerkiksi kuormantasain) ei tue määriteltyjä TLS-vaatimuksia, tehdään päätös siitä, onko riskialttiimpaa toteuttaa komponentti itse vai hyväksyä poikkeava salauskonfiguraatio.  Ks. tekninen ohje TLS-protokollasta. |
| Tallennettavan tiedon salaus | Kaikilla henkilötietoja tai turvallisuusluokiteltua materiaalia sisältävillä tallennusmedioilla on oltava käytössä tallennusmediatason salaus. Muilta osin salauksen tarve voidaan vaatia riskiperusteisesti erikseen. |  |

## 

## Tietosuojan suunnittelun yleisperiaatteet

Nämä periaatteet pätevät ainoastaan järjestelmiin, joissa käsitellään henkilötietoja. Osa vaatimuksista on mainittu muissa yleisperiaatelistoissa, jolloin tässä on annettu niihin vain viite.

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Tietosuoja otetaan huomioon uhkamallinnuksessa | Kun suoritetaan uhkamallinnusta (ks. Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet), henkilötietovirtojen osalta keskustelussa on käytävä läpi myös tietosuoja-asiat.  **Esimerkki:** Jos uhkamallinnusta tehdään esimerkiksi Microsoft STRIDE -menetelmällä, laajennetaan keskustelua esimerkiksi TRIM-menetelmällä (ks. uhkamallinnuksen ohje) tai muulla kevyellä tietosuojavaikutusten arvioinnin menetelmällä. |
| Henkilötietoja käsitellään vain, jos se on teknisesti tarpeen | Henkilötietojen käsittelyn tulee olla teknisesti välttämätöntä halutun toiminnallisuuden toteuttamiseksi. Jos toiminnallisuus voidaan toteuttaa ilman jotakin tiettyä henkilötietoa, sitä ei pidä käsitellä lainkaan. |
| Henkilötietojen siirto järjestelmien välillä on tehtävä rajapintojen avulla | Henkilötietoja ei saa siirtää tiedostovientien tai suoran tietokanta- tai tiedostojärjestelmäpääsyn kautta, vaan niiden järjestelmien välillä siirtämistä varten on luotava kontrolloitu rajapinta.  **Esimerkki:** SQL-tietokanta sisältää henkilötietoja, joita useampi sovellus tarvitsee. Toteutetaan HTTP-rajapinta, josta henkilötiedot ovat saatavilla ja ainoastaan HTTP-rajapinnan toteutus pääsee SQL-kantaan. HTTP-rajapinta luo auditointilokitapahtumia rajapinnan käytöstä. |
| Tietosuoja-asetuksen näin vaatiessa järjestelmän on toteutettava väliaikainen käsittelyn keskeytys | Jos tietosuoja-asetus käsittelyn perusteet huomioon ottaen tätä vaatii (artiklat 18 ja 21), järjestelmän on toteutettava kullekin järjestelmässä rekisteröidylle henkilölle tila, jossa järjestelmä lopettaa kyseisen henkilön tietojen käsittelyn (ml. lukemisen) muihin kuin erikseen määriteltyihin ylläpidollisiin tai viranomaistarkoituksiin. Tila on oltava kytkettävissä päälle ja pois. |
| Henkilötiedot on pystyttävä tuomaan järjestelmästä | Järjestelmän on toteutettava riittävät rajapinnat sille, että tietyn rekisteröidyn henkilön henkilötiedot ja häntä koskevat järjestelmän käytöstä syntyneet tiedot voidaan tuoda ulos koneluettavassa muodossa. |
| Henkilötiedot on pystyttävä poistamaan | Järjestelmän on toteutettava rajapinta henkilön poistamiselle järjestelmästä, mikäli järjestelmän käyttötarkoitus mahdollistaa poistopyynnöt lain nojalla tai jos henkilötietoja käsitellään suostumukseen perustuen. Tämä asettaa vaatimuksia käytetylle tietovarantorakenteelle, joka ei saa joutua epäkonsistenttiin tilaan henkilöiden poistamisen vuoksi. |
| Henkilötietojen viennistä toiselle rekisterinpitäjälle on jäätävä merkintä | Mikäli henkilötietoja viedään toiselle rekisterinpitäjälle, viennistä on jäätävä merkintä. Merkinnän on oltava saatavissa, jos rekisteröity käyttää lakisääteistä tietojenpoisto-oikeuttaan, ja nämä luovutustiedot on pystyttävä tuomaan järjestelmästä. Jos vientejä säännönmukaisesti tehdään vain ja ainoastaan tietyille muille rekisterinpitäjille, tätä ominaisuutta ei välttämättä tarvita. |
| Henkilötietojen käsittelyyn pyydettävästä suostumuksesta ja sen perumisesta on jäätävä merkintä | Ks. Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet |
| Henkilötietojen käyttöön liittyvät tiedot ja valinnat tuodaan käyttäjille oikea-aikaisesti ja oikeassa viitekehyksessä | Ks. Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet |
| Henkilötietojen siirto on tehtävä aina salattua yhteyttä käyttäen | Ks. Käyttöpalveluympäristöt ja operationaaliset yleisperiaatteet |
| Henkilötiedot on salattava silloin, kun ne on tallennettu | Ks. Käyttöpalveluympäristöt ja operationaaliset yleisperiaatteet. Tapauskohtaisesti on harkittava, tarvitaanko myös tietovarantotasoista (esimerkiksi tietokantataulu- tai rivikohtaista) salausta. |
| Henkilötietoihin kohdistuvista luku- ja muutostapahtumista on tehtävä merkintä | Ks. Auditoitavuuden yleisperiaatteet |
| Henkilötietoja paljastavan rajapinnan on aina tarkistettava tunnistus ja valtuutus | Ks. Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet |
| Viitteenä henkilöön tulisi käyttää satunnaista, käyttötapauskohtaista tunnistetta | Ellei ole erityistä syytä muuhun, mikäli henkilöön pitää antaa viite esimerkiksi toiselle palvelulle tai sidosryhmälle tai lokitiedostoon, viitteeksi pitäisi luoda kryptografisesti satunnainen ja käyttötapauskohtainen tunniste. Tunnisteen korrelointi henkilöön tulisi järjestelmän ulkopuolella olla vaikeaa tai mahdotonta. |
| Henkilötiedoille on määritelty enimmäissäilytysaika | Tallennettaville henkilötiedoille on määritelty maksimisäilytysaika, joka perustuu lakiin tai operatiiviseen tarpeeseen. On myös määritelty, miten tiedot teknisesti poistetaan maksimisäilytysajan umpeuduttua. |

## Auditoitavuuden yleisperiaatteet

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Järjestelmien tulee tuottaa riittävästi lokitietoa oleellisista tapahtumista | Osana uhkamallinnusta kartoitetaan ne toiminnat, mistä lokitietoa on kerättävä, sekä mitä tietoa on kerättävä. Useimmiten lokitiedon keräys tehdään rajapintakutsun yhteydessä. Lokimerkintöjen sisällöstä on seurattava periaatteita, jotka on kuvattu teknisessä liitteessä.  Erityisesti henkilötietojen lukemisesta ja muuttamisesta on aina tehtävä lokimerkintä, jonka perusteella voidaan selvittää henkilö, joka on suorittanut lukemisen tai tehnyt muutoksen.  Sisään- ja uloskirjautumisista sekä sovelluksen että alustajärjestelmien tasolla, käyttöoikeustason muutoksista, asetustiedostojen muutoksista ja luvusta sekä ulkoisesta syystä aiheutuneista virhetilanteista on aina tehtävä lokimerkintä. |
| Lokitapahtumien tietoturvamerkityksen on oltava lokien seurannan tiedossa | Lokeissa ja monitoroinnissa saattaa olla tapahtumia, joilla on merkitystä tietoturvahavaintoina. Erillisen operationaalisen tietoturvaseurannan (SOC, *Security Operations Centre*) on kuitenkin lähes mahdotonta pystyä havaitsemaan esimerkiksi liiketoimintalogiikkakohtaisia tietoturvahavaintoja, ellei heille ole annettu näistä tietoa ja tulkintaohjetta etukäteen.  Tietoturvan kannalta merkittävät sovelluskohtaiset loki- ja monitorointitapahtumat tulee tuoda SOCin tietoon, jotta niiden tapahtuminen voidaan havaita.  **Esimerkki:** Tuotetiimi toteuttaa rajapinnan, johon pitäisi tulla vain hyvin muodostuneita kutsuja luotetusta toisesta järjestelmästä. Tiimi toteaa uhkamallinnuksessaan, että mahdollinen väärin muotoiltu kutsu on selvä merkki mahdollisesta tietoturvapoikkeamasta ja ne kirjataan sisältöineen lokiin. SOCille kerrotaan, että tällainen lokitapahtuma on todennäköisesti joko osoitus tietomurrosta (IoC, *Indication of Compromise*) tai ainakin vikaantuneesta palvelusta; kummassakin tapauksessa välitön hälytys on aiheellinen. SOC lisää tämän erityistapauksen seurantaan. |
| Henkilötietojen tallennusta lokijärjestelmään on vältettävä | Lokijärjestelmään ei saa tallentaa henkilötietoja, ellei sitä voida teknisesti muutoin toteuttaa.  **Esimerkki:** IP-osoitteen, käyttäjätunnuksen tai vastaavan tunnisteen tallennus lokiin voidaan tehdä, jos ilman sitä lokimerkintää ei voida käytännössä käyttää tarkoitukseensa. |
| Eri järjestelmien lokien on oltava korreloitavissa keskenään | Kaikki lokilähteet tulee kerätä lopulta samaan paikkaan, jotta niiden analyysi on mahdollista yhdessä ja samassa järjestelmässä. Tämä mahdollisuus tulee tuottaa tiimeille yhteisenä palveluna.  Kaikki lokitallennus synkronoidaan koordinoituun yleisaikaan (UTC) luotetun aikapalvelimen avulla ja jokainen lokimerkintä varustetaan aikaleimalla, jonka tarkkuus on mahdollisimman suuri mutta vähintään sekuntitasolla.  On lisäksi suositeltavaa, että järjestelmän ulkopuolelta tuleviin pyyntöihin sidotaan satunnainen pyyntötunniste (*request ID*), joka välitetään taustajärjestelmältä toiselle ja kirjataan lokiin. Näin lokeista voidaan nähdä, mitkä toimenpiteet liittyivät mihinkin ulkopuolelta tulleeseen pyyntöön ja miten pyynnöt ketjuuntuvat (mikro)palveluarkkitehtuurissa. Tämän voi toteuttaa myös hajautetulla pyyntöseurantajärjestelmällä (*distributed tracing*), jos sellainen on käytössä esimerkiksi virheiden tutkimista varten. |
| Rajapintoihin tulevien pyyntöjen lähdeosoite on kirjattava lokiin | Rajapintoihin tulevien pyyntöjen lähdeosoite (IP-osoite) on kirjattava lokiin, myös mikäli rajapinta sijaitsee ei-julkisessa, luotetussa verkossa. Luotettujen verkkojen IP-osoitteiden varaamisesta käyttöön on pidettävä loki, josta käy ilmi IP-osoitteiden ja sitä käyttäneiden laitteiden laitteisto-osoitteiden vastaavuus. Julkisesta Internetistä tulevien pyyntöjen lähde-IP-osoite on otettava ylös verkon ulkolaidalla ja tallennettava lokiin tai taustapalvelimelle lokitusta varten. |
| Auditointilokimerkinnät on siirrettävä säännöllisesti pois niitä luovilta palvelimilta | Riippumatta lokienhallinnan ja -analyysin ympäristöstä, lokit tulee siirtää säännöllisesti ja mielellään reaaliaikaisesti pois kohteesta, jota lokitetaan. Tällä varmistetaan lokien luotettavuus, mikäli kohteen eheys vaarantuu. Siirrosta riippumatta lokeja säilytetään myös lähdepalvelimella 14 päivän ajan lokieheyden varmistamiseksi. |
| Lokimerkinnät on luotava standardoidulla tavalla | Organisaation omissa sovelluksissa lokimerkinnät on tehtävä rakenteisessa muodossa, mieluiten JSON-muotoisina, jotta niiden analyysi ja jäsennys on helposti tehtävissä. Lokimerkintöjen teknisestä toteutuksesta on seurattava periaatteita, jotka on kuvattu teknisessä liitteessä.  Valmisohjelmistojen osalta on pyrittävä yleisesti käytössä olevien konventioiden seuraamiseen. |
| Monitorointirajapinnan toteutus saatavuusongelmien havaitsemiseen | Järjestelmän osien on toteutettava sisäinen monitorointirajapinta, tai monitorointi voidaan suorittaa lokeja seuraamalla.  Uusissa monitorointirajapinnoissa tulisi pyrkiä julkaisemaan onnistuneiden pyyntöjen määrä aikayksikköä kohti, epäonnistuneiden pyyntöjen määrä aikayksikössä ja pyynnön käsittelyaika sekunneissa (pienin, isoin, keskimääräinen, mediaani) aikayksikössä (eli nk. *RED*-tiedot: *rate, errors, duration*). |

## Kehitysympäristöjen ja tuotantoon viennin yleisperiaatteet

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Ohjelmistokehitykseen käytettävät tietokoneet ja muut päätelaitteet on suojattava | Päätelaitteet, joilla otetaan yhteyksiä ohjelmistokehityksen palvelimiin (kuten versionhallintaan tai tuotantoonvientiautomaatioon) tai joilla tehdään ylläpidollisia toimia, on tuotava organisaation tietotekniikkaylläpidon määrittelemien tietoturvavaatimusten piiriin (vakiointi, päivitykset, haittaohjelmilta suojautuminen, jne.).  Tämä vaatimus ei päde, mikäli päätelaitteella tehdään vain testausta julkisiksi tarkoitettuja käyttöliittymiä tai rajapintoja vasten. |
| Ohjelmistokehitykseen käytettävien palvelinten on oltava lokienhallinnan piirissä | Ohjelmistokehityksessä käytettävät palvelimet, kuten versionhallinta-, integrointi- ja muut palvelimet, on tuotava lokienhallinnan piiriin. Lokienhallinnan vaatimukset sovelletaan kappaleen Auditoitavuuden yleisperiaatteet perusteella. |
| Kehitysympäristöt ja työkalut on suojattava ja niiden saatavuus on taattava | Kehitysympäristöjen ja -työkalujen tietoturvavastuu kuuluu oletusarvoisesti ohjelmistokehittäjille. Vastuu voidaan siirtää erikseen sopimalla tai käyttämällä organisaation toimittamia standardoituja ympäristöjä ja työkaluja. Kehitysympäristöjen ja -työkalujen haavoittuvuuksien seuranta on järjestettävä kuten kappaleessa Tietoturvallisen arkkitehtuurin ja suunnittelun yleisperiaatteet on kuvattu. |
| Kehitys- ja tuotantoinfrastruktuurin käyttöoikeudet on rajoitettava mahdollisuuksien mukaan | Kehitys- ja tuotantoinfrastruktuurissa on vältettävä jaettuja tunnuksia ihmisten käytössä, paitsi jos sitä ei voi teknisesti välttää. Automaation käyttämien tunnusten oikeudet on rajoitettava niihin toimintoihin, jotka automaatio tarvitsee. |
| Ohjelmistokehittäjien käyttöoikeudet on sidottava projektissa työskentelyyn | Ohjelmistokehittäjien oikeudet on poistettava, mikäli he eivät enää työskentele projektissa. Mikäli käytetään jaettua tunnusta, jaetun tunnuksen tunnistustiedot on uusittava aina, kun jokin tunnusta jakanut henkilö ei enää työskentele projektissa. |
| Ihmisten kehitysjärjestelmien käyttäjätunnuksissa on käytettävä kaksivaiheista tunnistusta | Ihmisten käytössä olevilla käyttäjätunnuksilla on oltava käytössä kaksiosainen tunnistus.  **Esimerkki:** Kaksiosainen tunnistus ei välttämättä ole interaktiivinen. Kaksi vaihetta voivat olla esimerkiksi versionhallintaa kehittäjän koneelta käytettäessä tietokoneen kiintolevyn salausavain ja tällä kiintolevyllä säilytettävä ssh-avain. Interaktiivisen kaksivaiheisen tunnistuksen tulisi mieluiten olla erikseen laskettava aikapohjainen tunniste (TOTP), eikä esimerkiksi tekstiviestitodennus. |
| Salaisuuksien hallinta on järjestettävä | Salaisuudet, esimerkiksi salaus- ja rajapinta-avaimet, on hallittava niin, että ne voidaan tarvittaessa helposti vaihtaa niin testaus- kuin tuotantoympäristöissäkin, ja että pääsy salaisuuksiin on vain nimetyillä henkilöillä. Salaisuudet on vietävä tuotantoon konfiguraationa. Salaisuuksien säilytys osana lähdekoodia on kielletty. |
| Ohjelmistoriippuvuuksien ja koodin versionhallinnan saatavuus on turvattava | Kaikki ohjelmistoriippuvuudet on tunnettava ja tämä koskee myös riippuvuuksien riippuvuuksia. Mikäli järjestelmä on huoltovarmuuskriittinen, ohjelmistoriippuvuuksia (ja näiden riippuvuuksia) on käytettävä Suomessa sijaitsevan kopion välityksellä. Versionhallinnan sekä tuotantoon vientiautomaation on sijaittava kokonaisuudessaan Suomessa. |
| Ohjelmistoriippuvuuksien ja koodin eheys on taattava | Mikäli ohjelmistoriippuvuus tukee käytettävän komponentin eheyden tarkastamista kryptografisin menetelmin (esimerkiksi digitaalinen allekirjoitus), se on tehtävä.  Mikäli koodin versionhallinta tukee digitaalisin allekirjoituksin toteutettua eheystarkistusta, sen käyttöönottoa tulisi harkita. |
| Ohjelmakoodi katselmoidaan | Ohjelmakoodille tehdään koodikatselmointia (*code review* tai automaattisena staattista analyysiä, *static analysis*). Ohjelmistotuotetiimi valitsee heille parhaiten sopivan tavan suorittaa tätä. Suosituksena on nk. *pull request* -menettelyn käyttö koodikatselmointiin. Erityistä tietoturvakatselmointia tarvittaessa tämä on syytä pyytää ja tiketöidä erikseen. |
| Tuotantoon viedyistä komponenteista tallennetaan kopio forensiikkatarkoituksiin | Luodaan kyky selvittää jälkikäteen, mitä koodia tuotannossa on ollut ajossa. Esimerkiksi kontteja käytettäessä tuotantoon vietävästä kontin kuvasta voidaan tehdä arkistokopio. |

## Pilviympäristöjen tietoturvaperiaatteet ja -vaatimukset

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Verkkoliikenteen oletetaan aina kulkevan ei-luotetussa verkossa (*zero-trust networking*) | Kun komponentit liikennöivät keskenään, mitään verkkoyhteyttä ei arkkitehtuurin tasolla oleteta oletusarvoisesti turvalliseksi. Verkon osa voidaan olettaa turvalliseksi vain, jos sen konfiguraatio ja siihen pääsy on riittävän hyvin suojattu ja suojausmekanismi on teknisesti arvioitavissa, eli turvallisuus ei perustu pelkään tietoturvapolitiikan määrittelyyn.  **Esimerkki:** Pilvipalvelusta halutaan yhteys itse hallinnoituun (*on-premises*) tietokantaan, johon on AWS Direct Connect -yhteys. Tästä huolimatta tietokantayhteydessä on käytettävä salausta ja kummankin pään todennusta.  **Esimerkki:** Kubernetes-klusterissa on monimutkainen mikropalvelu. Arkkitehtuurisuunnittelussa kartoitetaan mahdollisuus turvata palveluiden väliset yhteydet käyttäen *service meshiä*, joka tuottaa palveluiden välille salatut ja tunnistetut yhteydet.  **Esimerkki:** Kubernetes-klusteri sijaitsee yhdessä AWS:n VPC:ssä. Klusterin työkuormien luonne huomioon ottaen riskihyväksyntäpäätöksenä todetaan, että VPC:n sisäiseen liikenteeseen voidaan suhtautua niin, että klusterin sisällä ei tässä tapauksessa tarvita salattuja yhteyksiä. |
| Työkuormat viedään aina tuotantoon konteissa | Työkuormat viedään aina tuotantoon kontitettuna, jotta niiden riippuvuudet ovat selkeitä ja on selvää, mikä koodi kulloinkin on ajossa.  **Esimerkki:** Arkkitehtuuri hyötyisi tilanteessa nopeasti skaalautuvasta rinnakkaisprosessoinnista, joka olisi helppo toteuttaa *serverless*- tai FaaS (*Function as a Service*) palvelulla kuten AWS:n Lambdalla ilman konttia. Koska koodilla ei ole ulkoisia riippuvuuksia eikä se ole jatkuvuussuunnittelun kannalta kriittinen osa palvelua, päätetään, että kontittoman Lambdan käytön riski on hyväksyttävissä. |
| Pilvipalveluntarjoajan tilejä käytetään erottelemaan sovellukset ja niiden eri ympäristöt toisistaan | Eri sovelluksia tai sovellusten eri versioita (testaus/tuotantoversio) ei saa ajaa samalla tilillä.  **Esimerkki:** Tiimi ylläpitää kahta eri sovellusta, joilla ei ole suoria ajoaikaisia riippuvuuksia. Kummallekin on luotu tuotantoa varten oma AWS-tili (*account*), jotka molemmat on alistettu saman AWS-organisaation (*organization*) alaisuuteen. |
| Identiteetin- ja pääsynhallinnassa käytetään minimioikeuksin varustettuja rooleja | Oikeudet pilvipalvelun rajapintoihin jaetaan roolien kautta. Käyttäjät saavat käyttöönsä roolin, jolla on minimimäärä tarvittavia oikeuksia.  **Esimerkki:** Kehittäjällä on verraten harvoin ilmenevä tarve kutsua ”käsin” jotakin pilvipalvelun rajapintaa. Tätä varten luodaan rooli, jolla tämä oikeus on ja kehittäjä omaksuu tämän roolin (AssumeRole) vain tarvittaessa. Käsin tapahtuvassa käytössä roolin omaksuminen voi vielä vaatia kaksivaiheisen tunnistautumisen.  **Esimerkki:** Pilvipalvelun tuotantokonfiguraatiolle tilataan tutkivaa tietoturvatarkastusta. Tätä varten erilliseen tietoturvatestausta varten luotuun AWS-tiliin luodaan uusi käyttäjä ulkoiselle testaajalle. Tuotantotilille luodaan rooli, joka mahdollistaa konfiguraatioiden lukemisen ja tarkastelun. Testaajan käyttäjätunnukselle annetaan mahdollisuus omaksua tämä lukurooli. Testauksen loputtua tämä oikeus ja testaajan tunnus poistetaan. |
| Pilvessä ajettavien työkuormien verkkoliikenteen kohteet minimoidaan | Kun pilvessä ajetaan työkuormia, näiden pitäisi voida ottaa yhteyttä ainoastaan paikkoihin, joihin niiden on tarkoituskin olla yhteydessä.  **Esimerkki:** Pilvipalvelussa ajetaan kontteja Kubernetes-orkestraattorilla. Kubernetes-klusteriin määritellään politiikka (NetworkPolicy) tai *service mesh*, joka estää yhteydenotot muihin kuin erikseen määriteltyihin kohteisiin. |
| Pilvessä ajettavien työkuormien eheys varmistetaan | Pilvessä ajettavien työkuormien (*workload*) eheydestä on varmistuttava. Eheyden tulisi kattaa koko ajettavan koodin ja sen riippuvuuksien käsittelyketju.  **Esimerkki:** Riippuvuudet, kuten Docker-pohjakuvat ja kirjastoriippuvuudet kiinnitetään tiettyihin versioihin ja mieluiten varmistetaan kryptografisesti. Tarvittavia artefakteja käytetään Suomessa sijaitsevasta luotetusta kopiosta.  **Esimerkki:** Erittäin suurta tietoturvan tasoa vaativissa sovelluksissa harkitaan, pitäisikö myös versionhallinnassa käyttää kryptografisia eheyttä tukevia toimintoja kuten Gitin allekirjoitettuja muutoksia. |
| Pilvessä ajettavien työkuormien elinkaari on auditoitavissa | Pilvessä ajettavien työkuormien elinkaaresta jää auditoitavia todisteita, joiden avulla voidaan jälkikäteen päätellä, mitä koodia kulloinkin oli ajossa.  **Esimerkki:** Kaikista tuotantoon viedyistä Docker-konteista tehdään erillinen arkistokopio. Kubernetes-podien poistamisesta ja luomisesta jää auditoitava lokimerkintä. |
| Pilvipalveluntarjoajan palvelut on rajattu vain tarpeellisiin | Pilvipalveluiden tarjoajien palvelut rajataan niin, että vain tarpeellisia palveluita ja maantieteellisiä alueita voidaan käyttää.  **Esimerkki:** AWS-tilit luodaan AWS-organisaation alle, jolla on asianmukaiset palvelupolitiikat (*service control policies*). Näillä estetään esimerkiksi muiden alueiden (*region*) käyttö. |
| Pilvipalveluntarjoajan tallennetun tiedon salausratkaisut ovat käytössä | Pilvipalvelut tarjoavat yleensä tavan salata niihin tallennetun tiedon niin, että salaus on palvelun käyttäjälle läpinäkyvää. Vaikka tämä ei suojaakaan tietoa sovelluksen haavoittuvuuksilta, näitä tulee kuitenkin käyttää siksi, että tieto tallennetaan pilvipalveluntarjoajan muistivälineille salattuna.  **Esimerkki:** AWS:n S3-ämpäreiden oletusarvoinen salaus otetaan käyttöön. |
| Tietoturvamonitorointi on vastuutettu | Tuotetiimi määrittelee tuotteen uhkamalliin perustuen, mitä tietoturvaan liittyviä tapahtumia on seurattava. Tapahtumien seurannan vastuutus on määritelty.  **Esimerkki:** Mikäli käytetään erillistä SOCia, tuotetiimi kertoo SOCille ne tietoturvatapahtumat, joiden seuranta tuotteen osalta on tärkeää. Tuotetiimi huolehtii siitä, että nämä tapahtumat luodaan lokitapahtumavirtaan. |
| Pilvipalveluiden resurssit hallitaan versionhallinnassa säilytettävän konfiguraation kautta | Pilvipalvelun konfiguraatiota hallitaan versionhallinnassa. Käsityönä tehtävää pilvipalveluiden hallintaa ei tuotannossa saa tehdä salaisuuksien hallintaa lukuun ottamatta, eikä sitä suositella myöskään testiympäristöille.  Mikäli pilvipalvelun tarjoaja antaa käyttöön inventaariorajapinnan, sitä käytetään apuna ylimääräisten pilviresurssien havaitsemisessa. |

## Tietoturvallisten palvelurajapintojen ja käyttöliittymien yleisperiaatteet

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Käyttöliittymät ja rajapinnat eriytetään käyttäjätyyppien tarpeiden perusteella | Käyttäjätyypit jaotellaan käyttötapaustensa mukaan, esimerkiksi viranomaiset, kuluttajat/kansalaiset/asiakkaat, ylläpito, yritykset ja avoimen datan rajapinnat. Käyttötapausten salliessa rajapinnat eriytetään siten, että samasta rajapinnasta ei saa erityyppistä palvelua pelkästään esimerkiksi todennetun käyttäjätyypin perusteella, vaan esimerkiksi rajapinnan osoite tai polku on erilainen ylläpito- ja loppukäyttäjätoiminnallisuudessa. Käyttöliittymät rajataan toiminnallisesti sisältämään vain käyttäjäryhmän tarvitsema minimitoiminnallisuus.  **Esimerkki:** Sama palvelu tarjoaa rajapintoja kansalaisille ja ylläpidolle. Koska palvelu on monoliittinen ja sitä ei olla jakamassa esimerkiksi mikropalveluihin, päätetään tarjota ylläpitotoiminnallisuudet ulkoisen rajapinnan tasolla "eri" rajapinnasta (esim. juuresta alkaen eri URI-polku), vaikka toteutus onkin samassa sovelluksessa. Tämä mahdollistaa rajapintojen helpomman valvonnan esimerkiksi lokijärjestelmissä tai web-sovelluspalomuureissa. |
| Käyttövaltuudet hallitaan keskitetysti ja ne sovitetaan kulloiseenkin tarpeeseen | Käyttövaltuudet hallitaan keskitetystä paikasta, jotta vältetään hajautetun valtuuksien hallinnan mahdolliset ristiriitaisuudet ja katvealueet. Käyttäjillä on käyttökokemukseen ja -tarpeeseen sovitetut käyttövaltuustasot ja tarvittaessa mahdollisuus vaihtaa niitä, jotta kaikkia toimintoja ei tarvitse tehdä aina korkeimmalla tasolla. |
| Henkilötietojen käyttöön liittyvät tiedot ja valinnat tuodaan käyttäjille oikea-aikaisesti ja oikeassa viitekehyksessä | Henkilötietojen käsittelyyn (erityisesti keräämiseen ja siirtoon) liittyvät tiedotteet ja luvat näytetään käyttäjille käyttökokemuksen perusteella parhaiten soveltuvassa tilanteessa ja ilmaisu on tehty käyttäjän ymmärrystason huomioon ottaen. Henkilötietojen keräyksestä informoidaan asteittain tarkentuvalla tiedolla sen mukaan, kuinka tarkkoja tietoja käyttäjä tietojen käytöstä haluaa (nk. *layered notice*). |
| Suostumuksen kysymisen käyttöliittymässä on oltava auditoitavissa | Suostumusta käyttäjältä kysyttäessä sen perumisen on oltava yhtä helppoa kuin sen antaminen. Suostumuksen antamisesta ja perumisesta on jäätävä aikaleimattu merkintä. |
| Käyttäjä saa tehdä tietoturvaan vaikuttavan päätöksen vain, jos se on tarpeellista | Käyttäjä saa tehdä tietoturvaan vaikuttavan päätöksen (esimerkiksi poikkeustilanteen hyväksynnän) vain, jos käyttäjä voi todellisuudessa tuoda päätökseen lisätietoa, jota järjestelmällä itsellään ei ole. Käyttöliittymän on uskottavasti pystyttävä takaamaan, että käyttäjä ymmärtää päätöksensä seuraukset. Muussa tapauksessa järjestelmän tulee automaattisesti tehdä turvallinen päätös käyttäjän puolesta, eikä käyttäjä saa pystyä ohittamaan päätöstä. |
| Rajapinnat toteutetaan kieliopiltaan mahdollisimman yksinkertaisiksi | Rajapintojen hyväksymien syötteiden kieliopin on oltava mahdollisimman yksinkertainen, jotta syötteen oikeellisuuden tarkastaminen on helppoa tai edes mahdollista. |
| Käyttöliittymät ja rajapinnat hylkäävät väärät syötteet kokonaisuudessaan | Mikäli rajapintaan tulee väärä syöte, koko syöte hylätään mahdollisimman varhaisessa vaiheessa. Syötettä ei yritetä tulkita, mikäli se on osittainkin virheellinen. |
| Käyttöliittymät ja rajapinnat piilottavat tietovarantojen sisäisen toiminnallisuuden | Tietovarantojen sisäiseen toiminnallisuuteen liittyvät asiat, kuten tietokantarivien järjestysnumerot tai palvelimen tiedostojärjestelmän nimet eivät ole näkyvissä rajapinnan tai käyttöliittymän läpi. |
| Jokainen rajapinta vaatii todennuksen ja valtuutuksen riippumatta verkkoympäristöstä | Jokaisen tarjottavan rajapinnan on todennettava kutsujan ja tarkistettava sen valtuutus. Rajapintoja ei saa avata edes luotettuun verkkoon niin, että kuka tahansa verkon alueelta pystyy kutsumaan rajapintaa tunnistautumatta. Tästä ainoana poikkeuksena ovat tilanteet, joissa rajapintojen välillä on pisteestä toiseen toteutettu VPN-yhteys tai jos rajapinta on julkisen tiedon haku- tai monitorointirajapinta. |
| Rajapinnoissa suositaan standardeja | Rajapintojen toteutuksissa suositaan protokollia ja tietojen esitys- ja siirtomuotoja, jotka ovat laajalti hyväksyttyjä ja käytössä.  Organisaation eri osissa standardoidaan ja yhtenäistetään samoihin tarkoituksiin tarkoitetut rajapinnat ja näiden toteutukset. |
| Rajapintojen todennuksessa ja valtuutuksessa käytetään standardoituja protokollia ja algoritmeja | Todennus ja valtuutus perustuvat kryptografiaan. Kryptografisten protokollien ja algoritmien suunnittelu itse ei ole sallittua. Yleisesti hyväksyttyjen todennus- ja valtuutusprotokollien toteutus itse on sallittua, mutta niille on tehtävä tarkka suunnittelu- ja toteutuskatselmointi sekä testaus. |
| Järjestelmien lähtevän sähköpostin asetukset ehkäisevät roskapostia ja tietojen kalastelua | Lähtevää sähköpostia varten on määritelty domain-pohjainen lähtevän sähköpostin todennus (DMARC), joka pyytää vastaanottajaa hylkäämään vialliset viestit.  **Esimerkki:** Organisaation domain-ylläpito määrittelee DKIM-, SPF- ja DMARC-tiedot ja tarjoaa sovelluskehittäjille tavan lähettää sähköpostia näiden alaisuudessa. |

## 

## Tietoturvatestauksen ja teknisen tarkastuksen yleisperiaatteet

|  |  |
| --- | --- |
| **Periaate** | **Kuvaus** |
| Tietoturvatestauksen ja -tarkastuksen kohteet määritellään hyökkäyspinnan ja mahdollisen vaikutuksen mukaan | Tietoturvatarkastukset ja niiden tyyppi kohdennetaan arvioimalla eri toiminnallisuudelle suhteellinen riski hyökkäyspinnan (riskin todennäköisyys) ja toiminnallisuuden tärkeyden tai käsiteltyjen tietojen (riskin vaikutus) mukaan. Suositeltavaa on tehdä tämä uhkamallinnuksen yhteydessä. Tietoturvatestaus jakaantuu pääasiassa automaattisiin testitapauksiin ja tutkivaan testaukseen; tutkiva testaus kohdistetaan ensisijaisesti niihin kohteisiin, jotka ovat suhteessa riskialttiimpia. |
| Tietoturvatarkastuksen kattavuus ja tyyppi määritellään hyökkäyspinnan ja mahdollisen vaikutuksen mukaan | Tietoturvatarkastuksen tarvittava taso on lähtökohtaisesti OWASP ASVS Level 2, mikäli se on kohteeseen sovellettavissa. OWASP ASVS -dokumenttia voi käyttää myös testauksen tarkastuslistana ostettaessa web-sovelluksen tietoturvatarkastusta. Joissakin tapauksissa tarkastuksen tyypin on mahdollisesti oltava esimerkiksi Liikenne- ja viestintäviraston akkredointi. Pilvipalveluiden tarkastuksessa tarkastuksen on katettava myös tässä listassa luetellut pilvipalveluiden erityisalueet. |
| Tietoturvatestaustarpeet ja tekniset tarkastukset merkitään tehtävinä tuotteen tehtävälistalle | Automaattisen tietoturvatestauksen kehitystehtävät merkitään tuotteen tehtävälistalle kehittäjille tarkoitettuina työtehtävinä. Tekniset tarkastukset, vaikka ne tekisikin tuotetiimin ulkopuolinen taho, merkitään myös omina tehtävinään tuotteen tehtävälistalle, jotta niiden tilasta jää jälki. Ulkoisen testauksen työlistatehtävien omistaja on oletusarvoisesti tuoteomistaja, ellei niitä erikseen delegoida. |
| Tietoturvatestauksen ja teknisen tarkastuksen löydökset merkitään tuotteen tehtävälistalle | Kaikki tietoturvatestauksesta ja tarkastuksesta kumpuavat löydökset lisätään tuotteen tehtävälistalle niin, että tehtävänä on joko korjata ongelma, tuottaa kompensoiva kontrolli tai hyväksyä riski. |
| Tutkivasta tietoturvatestauksesta ylläpidetään vallitsevan tilanteen kuvausta | Tutkiva tietoturvatestaus tuottaa tuloksenaan löydösten lisäksi tietoturvan vallitsevan tilanteen kuvauksen, jota ylläpidetään kohdejärjestelmäkohtaisesti versioituna dokumenttina. Tämän dokumentin päivitys vastuutetaan tutkivan tietoturvatestauksen suorittajalle. |
| HTTP-rajapintojen tekniset kovennukset pidetään ajan tasalla selainten ja standardien kehittyessä | Rajapintoja vasten tehdään joko automaattista tai tutkivaa testausta, jolla huomataan puuttuvat rajapintojen tekniset kovennukset (esimerkiksi HTTP-tietoturvaotsakkeet, jotka on kehitetty sovelluksen alkuperäisen käyttöönoton jälkeen).  **Esimerkki:** Hankittavilta tietoturvaskannereilta ja tietoturvatestaajilta vaaditaan hankinnan ja toimeksiannon yhteydessä, että he huomioivat rajapintojen teknisten kovennusten ajantasaisuuden suhteessa alati kehittyviin standardeihin. |
| Koodina hallittu infrastruktuuri on osa tietoturvatarkastusta | Tietoturvatarkastuksen kattaessa pilvipalvelun myös kyseisen palvelun infrastruktuurikuvaukset katselmoidaan tarkastuksen yhteydessä.  **Esimerkki:** Tilataan tietoturvatarkastus sovellukselle, jota ajetaan Kubernetes-klusterissa AWS:ssä. Tietoturvatarkastukseen sisällytetään AWS:n Terraform-konfiguraatiot, Kubernetes-klusterin konfiguraatiot sekä konttien Dockerfilet. |
| Pilvi-infrastruktuurin identiteetin- ja pääsynhallinta on osa tietoturvatarkastusta | Tietoturvatarkastuksen kattaessa pilvipalvelun myös kyseisen palvelun IAM-konfiguraatio katselmoidaan tarkastuksen yhteydessä. |
| Tuotantoonviennin automaatio on tietoturvatarkastuksen piirissä | Tuotetiimin käyttämä tuotantoonviennin automaatio (CI/CD) kuuluu tarkastettaviin kokonaisuuksiin kuten tuotetiimin ylläpitämät sovelluksetkin. Tarkastettavat osa-alueet ovat vähintään pääsynhallinta itse CI/CD-automaatioon, salaisuuksien hallinta sekä CI/CD-automaation pääsynhallinnan järjestäminen pilvi-infrastruktuuriin. |
| Tunnusten luontia ja vanhentumista on seurattava | Pilvipalvelun tunnusten luontia, roolien omaksuntaa ja niiden käyttöä on valvottava.  **Esimerkki:** Uusien IAM-käyttäjien, pääsyavainten (*access keys*) ja roolien luomisesta on tultava heräte, jonka ihminen (esimerkiksi SOC) tarkastaa ilman suurta viivettä. Roolin omaksumisesta (AssumeRole) on tallennettava auditoitava lokitapahtuma. Käyttäjätunnukset ja pääsyavaimet, joita ei ole käytetty tietyn ajan jälkeen, on säännöllisesti poistettava. |

# 

# Uuden tai muuttuneen toiminnallisuuden tietoturva- ja tietosuojatyön tarkastuslista (Liite 3)

Tietoturva- ja tietosuojatyö tulee kohdentaa oikein, jotta resursseja ei tuhlata kohteisiin, joissa ei ole merkittäviä tietoturvariskejä. Tätä tarkastuslistaa voidaan käyttää sen määrittämiseen, mitä tietoturvatyötä toiminnallisuudelle on tarpeen tehdä.

Tarkastuslista ei kata kaikkia tietoturva-aktiviteetteja vaan ainoastaan ne tehtävät, joiden avulla loput aktiviteetit voidaan löytää. Tyypillisimmin olennaisin tehtävä on uhkamallinnus. Uhkamallinnuksen suorittaminen paljastaa muut tietoturvatarpeet.

Tarkastuslistaa voidaan käyttää eri tasoisille työlistan tehtäville. Yleisimmin kohteena on käyttäjäkertomustasoinen (*user story*) tehtävä, mutta erityisesti tietosuojakysymyksiin voidaan usein vastata korkeammalla tasolla (*epic*). Joskus taas tarkastuslistan läpikäyminen voi olla tarpeen myös hyvin matalan tason tehtävillekin - esimerkiksi jos todennusmekanismissa suoritetaan bugikorjaus.

Tarkastuslistaa käytetään vastaamalla vasemman sarakkeen kysymyksiin. Jos vastaus on myöntävä, varmistetaan, että tehtävälistalla on oikeassa sarakkeessa merkitty tehtävä. Syvällisempää analyysiä ei välttämättä tässä vaiheessa tarvita; listan lopussa on esitetty mallitekstejä, jotka voi yleisimmin leikata ja liimata.

Tehtävät luokitellaan käyttäen työnohjausjärjestelmän leimoja (*label* tai *tag*), jotta tietoturva- ja tietosuojatyön seuraaminen on mahdollista.

|  |  |
| --- | --- |
| **Liittyykö uusi tai muuttunut toiminnallisuus...** | |
| **...uuden tietovarannon kuten tietokannan käyttöönottoon?**  **...todennuksen, valtuutuksen tai istuntojen hallinnan toteutukseen?**  **...järjestelmän tuotantoonviennin periaatteen muutoksiin?**  **...uuden kolmannen osapuolen riippuvuuden käyttöönottoon?**  **...uuden rajapintaintegraation käyttöönottoon?**  **...olemassa olevan oman rajapinnan toiminnallisuuden laajentamiseen?** | * Tehtävälistalla on tehtävä (1) joka linkitetään toiminnallisuuden toteuttavaan tehtävään * Tehtävälistan tehtävällä on (1) luokitus "uhkamallinnus" |
|  |
| **...muutokseen henkilötietojen käsittelyssä (mitä tai miten käsitellään)?**  **...muutokseen analytiikassa, mainostuksessa tai profiloinnissa?** | * Tuotteen tehtävälistalla on tehtävä (2) joka linkitetään toiminnallisuuden toteuttavaan tehtävään * Toiminnallisuuden toteuttavalla tehtävällä on luokitus "tietosuoja" ja lisätyllä tehtävällä (2) luokitus "pia" * Tuotteen tehtävälistalla on tehtävä (1) joka linkitetään toiminnallisuuden toteuttavaan tehtävään * Tehtävällä (1) on luokitus "uhkamallinnus" |
| **...henkilötietojen laajamittaisen käsittelyn, kattavan profiloinnin, järjestelmällisen valvonnan tai tietojen EU:n ulkopuolelle viennin aloittamiseen?**  **...muutokseen henkilöiden arvioinnissa, automaattiseen oikeusvaikutusten luomisessa?**  **...muutokseen arkaluontoisten tai haavoittuvassa asemassa olevien henkilöiden tietojen käsittelyssä?**  **...uuden tai innovatiivisen henkilötietojen hankinnan tai käsittelytekniikan käyttöönottoon?** | * Tuotteen tehtävälistalla on tehtävä (3) joka linkitetään toiminnallisuuden toteuttavaan tehtävään * Toiminnallisuuden toteuttavalla tehtävällä on luokitus "tietosuoja" ja lisätyllä tehtävällä (3) luokitus "pia" * Tuotteen tehtävälistalla on tehtävä (1) joka linkitetään toiminnallisuuden toteuttavaan tehtävään * Lisätyllä tehtävällä (1) on luokitus "uhkamallinnus" |

Tehtävä (1): "Suoritetaan uhkamallinnus toiminnallisuudelle käyttäen STRIDE-menetelmää (turvallisen ohjelmistokehityksen käsikirjan Liite 5: Uhkamallinnuksen toteutusohje). Uhkamallinnuksessa löytyvien riskien ehkäisemiseen tarkoitetut vaatimukset kirjataan tuotteen tehtävälistalle."

Tehtävä (2): "Suoritetaan tietosuojavaikutusten arviointi (turvallisen ohjelmistokehityksen käsikirjan Liite 6: Tietosuoja). Tietosuojavaikutusten arvioinnissa löytyvien riskien ehkäisemiseen ja lain noudattamiseen tarkoitetut vaatimukset kirjataan tuotteen tehtävälistalle."

Tehtävä (3): "Suoritetaan tietosuojavaikutusten arviointi tietosuoja-asetuksen määräämällä tavalla (DPIA) (turvallisen ohjelmistokehityksen käsikirjan Liite 6: Tietosuoja). Tietosuojavaikutusten arvioinnissa löytyvien riskien ehkäisemiseen ja lain noudattamiseen tarkoitetut vaatimukset kirjataan tuotteen tehtävälistalle."

# Tietoturva-arkkitehtuurin dokumentoinnin vähimmäisvaatimukset (Liite 4)

Tietoturva-arkkitehtuurikuvaus tulisi mieluiten tehdä versiohallittuun kirjoitusalustaan (esimerkiksi Confluenceen), jotta erillisten, vanhentuvien dokumenttien leviämistä ehkäistään.

Tietoturva-arkkitehtuurin kuvauksen täsmällinen muoto ja sisältö riippuu toteutettavasta palvelusta, mutta minimissään se sisältää kolme tietovuokuvausta (*Data Flow Diagram*):

1. Järjestelmän kontekstitason kuva, joka näyttää tietovirrat ja eriteltyinä henkilötietovirrat järjestelmän ja sen ulkopuolisten liityntöjen välillä, mukaan lukien käyttäjät. Kontekstitason yhteydessä kirjoitetaan myös auki keskeisimmät käyttötapaukset lyhyesti; käyttötapauksissa mainittujen järjestelmien tulisi olla näkyvissä kuvassa.
2. Järjestelmän komponenttitason kuva, joka näyttää
   1. järjestelmän rakenteen yksittäisten käyttöjärjestelmäprosessien tasolle; sekä
   2. näiden prosessien väliset tietovirrat, henkilötietoja sisältävät tietovirrat erikseen merkittyinä; sekä
   3. järjestelmän luottamusalueet eli mitkä komponentit luottavat absoluuttisesti toisiinsa eli eivät tarvitse keskinäistä todennusta eivätkä valtuutusta; sekä
   4. monimutkaisemmista protokollista tai interaktioista viestisekvenssikaaviot (*Message Sequence Chart*). Tuotantoon viennin tietovirtakaavio, joka näyttää tuotantoon viennin järjestelmät ja tietovirrat. Näitä järjestelmiä ovat muun muassa kehittäjien koneet, versionhallinta, integraatiopalvelimet ja salaisuuksien hallinnassa käytettävät palvelut.

Kuvat voidaan tuottaa osana uhkamallinnusta ja henkilötietoihin liittyvä osuus tietosuojavaikutusten arvioinnin yhteydessä.

Mikäli arkkitehti hallitsee töitään tiketöintipohjaisen työnohjauksen kautta, arkkitehtuuridokumentaation päivittäminen kannattaa myös tiketöidä, jotta sille tulee varattua riittävästi aikaa.

Mikäli organisaatio ei ole osoittanut tietosuojavaikutusten arviointien tuloksille erillistä paikkaa, myös tietosuojavaikutusten arviot kirjataan tietoturva-arkkitehtuurikuvauksen jatkoksi. Tämän dokumentaation sisältö on kuvattu kappaleessa Tietosuoja.

# Uhkamallinnuksen toteutusohje (Liite 5)

## Uhkamallinnus yleisesti

Uhkamallinnus (*threat modeling* tai *architectural risk analysis*) on menetelmä, jossa kartoitetaan arkkitehtuurin ja tietovuokaavion avulla mahdollisia sovellusturvallisuuden ja tietosuojan riskejä sekä riskien hallintamekanismeja.

Tässä ohjeessa suositellaan käytettäväksi Microsoftin STRIDE-menetelmää. Projektin tietoturva-asiantuntijaa, mikäli tällainen on nimetty, voi käyttää uhkamallinnuksen fasilitaattorina, jolloin mukana olijat oppivat menetelmän työn ohessa. Lyhyt selitys menetelmästä on myös alempana.

Uhkamallinnuksen tulisi olla tuotetiimin yhteistyön tulos. Uhkamallinnuksen tavoite on olla luova riskienkartoitusharjoitus ja usean eri henkilön näkökulmat yhdessä antavat yleensä paremman näkyvyyden mahdollisiin ongelmiin.

Uhkamallinnuksen tuloksena syntyy lista mahdollisista tietoturvaheikkouksista tai haavoittuvuuksista ja yleensä myös suoraan ehdotuksia asioista, mitä näille voisi tehdä. Tyypillisiä tuloksia ovat esimerkiksi testaustarpeet, tietyt yksittäiset testitapaukset, tarve muuttaa arkkitehtuuria tai matalamman tason suunnittelua, tai jonkin tietoturvaominaisuuden lisääminen.

Nämä löydökset siirretään tuotteen tehtävälistalle uusina tehtävinä tai olemassa olevien tehtävien hyväksyntäkriteereinä. Uudet tehtävät luokitellaan avainsanalla "uhkamallinnus".

**Lähteet:**

* STRIDE-metodi: Adam Shostack: *Threat Modeling: Designing for Security*. Wiley, 2014.

## Uhkamallinnuksen aikataulutus

Paras tulos uhkamallinnukselle saadaan, jos sitä vaaditaan osana tuotteen tehtävälistaa tärkeimpien toiminnallisten vaatimusten hyväksyntäkriteerinä tai erillisenä tehtävänä, joka on linkitetty toiminnalliseen vaatimukseen. Näin mallinnustehtävät säilyvät järjellisen kokoisina ja asioista keskustellaan ajoissa, jolloin muutosten teko on helpointa.

Mikäli käytetään Scrum-mallia, uhkamallinnusta ei yleensä kannata tehdä osana Sprint Planningia vaan pikemminkin työjakson (sprintin) kestäessä. Yksi vaihtoehto on poimia työjakson sisältöön esimerkiksi tulevien työjaksojen toiminnallisuuksien uhkamallinnusta, jolloin uhkamallinnus kulkee 1-2 työjaksoa toteutusta edellä. Uhkamallinnuksen tekeminen edeltävällä työjaksolla ehkäisee myös sitä, että työjakso epäonnistuisi ilmenevän lisätyön vuoksi. Uhkamallinnuksen käytännön aikataulutus on myös vapaampaa, jos se ei kohdistu juuri sillä hetkellä toteutuksen alaisena olevaan toiminnallisuuteen. Ks. myös kappale Erityishuomioita tiettyihin ohjelmistokehitys- ja kulttuurisiin malleihin.

Uhkamallinnuksen voi myös suorittaa toteutuksen jälkeen. Vaikka se ei tuolloin pystykään ennustamaan tietoturvaongelmia, toimii se silti erinomaisena retrospektiivinä. Jälkikäteisen uhkamallinnuksen tulokset ovat usein konkreettisempia ja se toimiikin hyvänä tietoturvatarkastusta edeltävänä aktiviteettina.

## Uhkamallinnus käytännössä

Tyypillinen uhkamallinnus suoritetaan työpajatyyppisesti. Paikalle pyydetään kaikki ne henkilöt, jotka tietävät tekniset yksityiskohdat arkkitehtuurista ja toteutuksesta. Organisaation pilvitiimin edustajan tulisi olla mukana uhkamallinnuksessa ainakin silloin, kun toiminnallisuudella on yhtymäkohtia pilvi-infrastruktuuriin tai läpileikkaaviin, kaikille tuotetiimille yhteisiin teemoihin. Käytännössä noin viisi henkilöä on sopiva määrä. Paikalle kannattaa pyytää myös tuoteomistaja ja käyttökokemuksesta vastaava henkilö, mikäli mallinnettava toiminnallisuus on täysin uutta. Työpajalle on hyvä varata 1-2 tuntia aikaa. Pienet toiminnallisuudet hoituvat nopeammin, jopa kahvipöytäkeskusteluna, mutta kaksi tuntia ylittää yleensä ihmisten sietokyvyn. Jos kyseessä on laaja analyysi, esimerkiksi jo rakennetun toiminnan retrospektiivinen mallinnus, sessioita on todennäköisesti järjestettävä useita.

Työpajassa piirretään tussitaululle tietovuokaavio (*data flow diagram*, DFD) tai viestisekvenssi-kaavio (*message sequence chart*, MSC). Viestisekvenssikaavio kannattaa mahdollisesti tuottaa jo ennakkoon niiden monimutkaisuuden vuoksi.

Tietovuokaaviota käytetään, jos mallinnetaan ajossa olevaa järjestelmää (sen prosesseja, tie­tovarastoja ja eri komponenttien välistä kommunikaatiota) tai tuotantoon viennin arkkitehtuuria, kuten yhteyksiä versionhallintajärjestelmiin tai CI/CD-putkea.

Viestisekvenssikaaviota käytetään, jos mallinnetaan useita viestejä edestakaisin lähettäviä toiminnallisuuksia - esimerkiksi todennusta tai toiminnallisuutta, jossa selainta ohjataan eri rajapintoihin tai palveluihin toiminnon aikana.

*Kaavion piirtäminen tussitaululle on olennainen osa mallinnusta.* Valmiita kuvia voi käyttää referenssinä, mutta niitä ei pitäisi (monimutkaista MSC:tä lukuun ottamatta) ottaa suoraan pohjaksi. Usein käy niin, että valmis kuva ei vastaakaan todellisuutta tai kaikkien osallistujien käsitys arkkitehtuurista ei ole yhteneväi­nen. Nämä olettamusten erot ovat kokemusperäisesti usein tietoturvaongelmien lähde.

Kaavion voi piirtää esimerkiksi yksi käyttäjätarina kerrallaan, jolloin mukaan otetaan ne komponentit, jotka kulloinkin tarvitaan. Kaaviosta ei pidä unohtaa ihmisiä, joita ovat sekä käyttäjät, kehittäjät että ylläpito.

Kun kuva on valmis, jokainen piirretty tietovuo ja tietovarasto käsitellään keskustelemalla kuutta eri aihealuetta (STRIDE):

1. **Spoofing** eli todennukseen liittyvät asiat (esim. komponenttien välisen liikenteen päätepisteiden todennus)
2. **Tampering** eli eheyteen liittyvät asiat (esim. hyökkääjän mahdollisuus muuttaa tietoa)
3. **Repudiation** eli kiistettävyyteen liittyvät asiat (esim. auditointilokin luonti ja mitä sinne tallennetaan)
4. **Information disclosure** eli luottamuksellisuuteen liittyvät asiat (esim. pääseekö hyökkääjä käsiksi salaamattomaan tietoon, tai vuotaako tietoa sivukanavia pitkin)
5. **Denial of Service** eli saatavuus ja palvelunestohyökkäykset (esim. jos tietty komponentti ei vastaa, mitä se aiheuttaa, tai onko jonkin komponentin kautta mahdollisuus generoida liikaa kuormaa jollekin toiselle komponentille)
6. **Elevation of Privilege** eli käyttövaltuuksien ylittäminen (esim. koodi-injektiohyökkäykset)

Keskustelun voi ohjata esimerkiksi niin, että kehittäjät ottavat yhden tietovuon kerrallaan (joko DFD- tai MSC-kuvasta) ja lausuvat ääneen teknisen perusteen sille, miksi esimerkiksi todennus on turvallisesti tehty.

**Esimerkki:** "Palvelin A tietää, että sitä kutsuu palvelin B, koska niiden välillä käytetään TLS:ää ja palvelin A laittaa palvelinkohtaisen API-avaimen jokaisen HTTP-kutsun otsakkeeseen."

Muiden tehtävä on kyseenalaistaa tätä argumenttia ja tehdä tarkentavia kysymyksiä.

**Esimerkki:** "Tuo sama API-avain on käytössä myös A:n testiversiolla, joten tarkasti ottaen B ei tiedä, kutsuuko sitä tuotanto- vai testiversio." "Muuten hyvä, mutta emme ole vaihtaneet API-avainta vuosiin." "Ja API-avain on tallennettu versionhallintaan koodin kanssa."

Vastalauseet saavat olla hypoteettisia, kunhan ne ovat jollakin tavoin perusteltuja. Vastalauseista kirjataan tiketti, joka korjaa kunkin havaitun ongelman tai heikkouden tai vähentää hypoteettisen ongelman riskiä.

Kun kukaan ei enää pysty keksimään heikkoja kohtia argumentaatiosta (tai argumentti on osoitettu vaillinaiseksi ja tarvittavat toimenpiteet on kirjattu), voidaan siirtyä eteenpäin. Seuraava keskustelun kohde voidaan valita mekanistisesti STRIDE-mallin mukaan tai vapaammin mielenkiinnon ja prioriteettien mukaan.

Mikäli uhkamallinnusta tehdään henkilötietoja käsittelevälle toiminnallisuudelle, keskustelua voidaan jatkaa nk. TRIM-lisäyksellä, jota käytetään aivan samoin kuin STRIDEä:

1. **Transfer** of personal data: Siirretäänkö tietovuossa henkilötietoja maan, organisaation tai sopimusteknisen rajan ylitse, ja onko tähän oikeutus ja lupa? Onko siirron kohde oma rekisterinpitäjänsä vai ainoastaan käsittelijä ja jos jälkimmäinen, onko se sopimuksellisesti sovittu?
2. **Retention / Removal**: Jos komponentti tallentaa henkilötietoja, onko määritelty aika- tai muu kriteeri, jonka perusteella tallennetut tiedot tuhotaan; miten teknisesti toteutetaan yksittäisen henkilön tietojen poisto tarvittaessa ja miten yksittäisen henkilön tietojen käsittely voidaan pyynnöstä keskeyttää.
3. **Inference**: Yhdisteleekö komponentti useista eri lähteistä tulevia henkilötietoja muodostaen niistä uusia henkilötietotyyppejä, joita tallennetaan tai välitetään edelleen? Muuttaako se tietoa, joka aiemmin ei ollut henkilötietoa, henkilötiedoksi yhdistämällä sitä muuhun henkilötietoon? Onko nämä uudet henkilötietotyypit otettu huomioon tietosuojavaikutusten arvioinnissa?
4. **Minimisation**: Kun komponentti lähettää henkilötietoja eteenpäin (esimerkiksi johonkin rajapintaan), onko tämä henkilötietojen joukko teknisesti ottaen pienin mahdollinen, vai voisiko sitä vielä entisestään pienentää teknisen toiminnallisuuden kärsimättä?

**TRIM ei korvaa tietosuojavaikutusten arviointia** (PIA, *privacy impact assessment* tai DPIA, *data protection impact assessment*), mutta se toimii alimman tason turvaverkkona sen suhteen, että jos henkilötietojen käsittelystä ei ole aiemmin tehty vaatimuksia, asia nousee viimeistään tässä vaiheessa keskusteluun.

Mikäli TRIM-analyysissä ilmenee, että jokin tietosuojanäkökanta on huonosti selvitetty, lisätään tehtävälistalle uusi tehtävä tietosuojavaikutusten arvioinnista (ks. Liite 6: Tietosuoja), joka luokitellaan avainsanalla "pia".

## Uhkamallinnuksen laatu

Uhkamallinnuksen tulosten laatuun vaikuttaa usein melko paljon sitä tekevien henkilöiden kokemus. Monet heikkoudet nimittäin ilmenevät eri järjestelmissä uudestaan ja uudestaan, ja kokemus mahdollistaa tehokkaan uhkamallinnuksen paremmin kuin vain listojen läpikäynti.

Kuitenkin on käynyt kokemuksen pohjalta selväksi, että nimenomaan STRIDE-tyyppinen, suhteellisen vapaamuotoinen mutta kuitenkin ohjattu analyysimenetelmä tuottaa varsin hyviä tuloksia ammattimaisten ohjelmistokehittäjien käsissä jo yhden tai kahden fasilitoidun esimerkkityöpajan jälkeen.

Yksi tärkeimmistä laadun takeista on olla hyväksymättä oletuksiin perustuvia tietoturva-argumentteja. Jos keskustelussa käy ilmi, että jostakin tietovirrasta tai sen sisällöstä ei ole täyttä varmuutta, se kannattaa varmistaa esimerkiksi koodia lukemalla, rajapintakuvauksesta tai protokolla-analysaattorilla. Samaten kuvan piirtämisessä kannattaa olla huolellinen; notaatiolla ei juuri ole merkitystä, kunhan on selvää, mikä komponentti keskustelee minkäkin kanssa.

## Uhkamallinnus, kun tekninen ratkaisu on vielä tuntematon

Jos uhkamallinnusta tehdään korkean tason tehtävälle (*epic*), esimerkiksi portfoliokanbanin analyysivaiheessa tietosuojatarkastelun ohessa, on hyvin mahdollista, että toteutusta ei vielä tunneta riittävällä tarkkuudella DFD- ja MSC -tyyppisten kaavioiden piirtämiseksi ja STRIDE-keskustelun pohjaksi.

Tällöin voidaan käyttää muita riskienlöytömenetelmiä. Yksi sellaisista on nk. hyökkääjätarinoiden (*attacker story* tai *abuse case*) luominen, jossa pyritään löytämään ei-toivottuja käyttäjätarinoita, jotka ovat yleensä muotoa

"Hyökkääjä (jokin järjestelmän kanssa yhteydessä oleva ihminen) käyttää hyväkseen (hypoteettista) heikkoutta tai haavoittuvuutta (oletetussa) kohdejärjestelmässä ja saa aikaan teknisen vaikutuksen, joka johtaa liiketoiminnalliseen vaikutukseen (ei-toivottuun lopputulokseen)."

Hyökkääjätarinat voidaan kirjata käyttäjätarinoiksi, mutta niiden toteutus tehdään toteuttamalla tietoturvatoiminnallisuutta (kontrolleja) tai tietoturva-aktiviteetteja. Hyökkääjätarinan olemassaolo tehtävälistalla kuitenkin varmistaa sen, että identifioitu tietoturvatyö tulee näkyväksi.

Hyökkääjätarinoita voi luoda yksinkertaisesti luomalla esimerkiksi valkotaululle neljä saraketta, joihin saadaan sisältö seuraavasti:

* Hyökkääjät: Kaikki ihmiset, jotka pystyvät toimimaan järjestelmän kanssa kuten käyttäjät, kehittäjät, ylläpitäjät, asiakaspalvelu, satunnaiset Internet-käyttäjät, jne.
* Hyökkäysskenaariot: Voivat olla hypoteettisia, eli näiden heikkouksien olemassaoloa ei tarvitse todistaa - kunhan ne ovat realistisia.
* Vaikutukset järjestelmille: Tiedossa olevat tekniset järjestelmät ja millaisen vaikutuksen niille voisi aiheuttaa.
* Ei-toivottu lopputulos: Jotakin, jota ei haluta tapahtuvan.

Käytännössä sarakkeiden täyttäminen kannattaa aloittaa reunoilta (hyökkääjät ja ei-toivotut lopputulokset). Ei-toivotut lopputulokset voidaan saada selville ottamalla toivottu lopputulos ja listaamalla tapoja, joilla ne voivat mennä pieleen. Esimerkiksi "asiakas saa sähköpostilla kuittauksen" voisi muuttua muotoihin "sähköpostikuittausten saaminen estyy" tai "sähköpostikuittaus lähetetään hyökkääjän osoitteeseen". Vaikutuksen ei tarvitse olla liiketoiminnallinen tai organisatorinen - tietosuojamielessä myös vaikutus johonkuhun rekisteröityyn henkilöön voi olla ei-toivottu lopputulos.

Vaikutukset järjestelmille on usein helpohko johtaa ei-toivotuista lopputuloksista kysymällä, millainen järjestelmäongelma sitten aiheuttaisi asianomaisen ei-toivotun lopputuloksen. Näitä saattaa olla useampia. Esimerkiksi edellä mainitun sähköpostikuittauksen saamisen estymisen voisi aiheuttaa sähköpostipalvelimen palvelunestotila tai sähköpostien päätyminen roskapostiin, koska palvelin on päätynyt mustalle listalle.

Hyökkäysskenaarioiden luetteleminen on näistä ehkä hankalin vaihe, mutta tähän voi käyttää esimerkiksi tietoturva-asiantuntijan tai MITREn ATT&CK- ja CAPEC-resurssien apua. [MITRE ATT&CK](https://attack.mitre.org/) on tässä käsikirjassa mainittu aiemmin yhtenä ohjeellisena vaatimuslähteenä. [MITRE CAPEC](https://capec.mitre.org/) toimii sovellustasolla sen vastinparina.

Kun kaikissa sarakkeissa on sisältöä, näitä yhdistelemällä voidaan luoda hyökkääjätarinoita, jotka voidaan sitten kirjata tehtävälistalle "negatiivisina" käyttötapauksina. Hyökkääjätarinoiden keskinäinen tärkeysjärjestys voidaan saada selville valitsemalla ikävimmät ei-toivotut vaikutukset ja seuraamalla näihin loppuvia tarinan kaaria.

# Tietosuojan toteutumisen varmistaminen (Liite 6)

## Tietosuojavaikutusten arviointi yleisesti

Tietosuojavaikutusten arvioinnin tarvittavat taustatiedot ja monet toimenpiteetkin ovat käytännössä samat kuin uhkamallinnuksen ja tietoturva-arkkitehtuurin, joten samat henkilöt, jotka ovat tekemässä uhkamallinnusta ja dokumentoimassa arkkitehtuuria ovat todennäköisesti parhaita tekemään myös tietosuojavaikutusten arvioinnin. Tarvittaessa apua voidaan pyytää organisaation tietosuojafunktiolta.

Tietosuojavaikutusten arvioinnin vaiheet ovat:

1. Määritellään kohde. Yleensä tämä on tietyn tuotteen tehtävälistan tehtävän kuvaama toiminnallisuus tai laajempi kokonaisuus. Tietosuojavaikutusten arvioinnista avataan dokumentointia varten oma tehtävä, joka linkitetään kohteena olevaan tehtävään ja luokitellaan avainsanalla "pia".
2. Selvitetään kohteen henkilötietovirrat ja tallennuspaikat konteksti- ja komponenttitasolla. Tämä on sama vaatimus kuin mikä tarvitaan tietoturva-arkkitehtuurin dokumentaatioon (ks. Liite 4: Tietoturva-arkkitehtuurin dokumentoinnin vähimmäisvaatimukset). Henkilötietovirroista on selvitettävä niissä siirrettävät henkilötietotyypit ja se, millä tavalla siirron tai tallennuksen suojaus on toteutettu.
3. Varmistetaan, että kohteen uhkamallinnus (ks. Liite 5: Uhkamallinnuksen toteutusohje TRIM-lisäyksin) on tehty.
4. Kohdassa 2 selvitetyille henkilötietovirroille ja -varannoille kirjataan:
   1. Käsittelyn laillinen peruste
   2. Käsiteltävien henkilötietojen määrä ja laatu. Mikäli käsitellään erityisiä henkilötietoryhmiä eli "arkaluonteisia" henkilötietoja tai erillislainsäädännön alaisia henkilötietoja, tämä on huomioitava erikseen
   3. Henkilötietojen maantieteellinen sijainti ja peruste niiden viemiselle pois EU:n alueelta, jos näin tapahtuu
   4. Luettelo ulkopuolisista tietojen käsittelijöistä, jotka saavat henkilötietoja, mukaan lukien mahdolliset käyttöpalvelun toimittajat; jos näitä on, varmistetaan, että sopimuksissa on huomioitu tietosuoja-asetuksen vaatimukset
   5. Käydään läpi kappaleen Tietosuojan suunnittelun yleisperiaatteet vaatimukset ja varmistetaan, että ne täyttyvät
   6. Mikäli kohdan 4 keskustelussa ilmenee puutteita tavoitetilaan (esimerkiksi vastausta ei osata antaa), puutteiden korjaamisesta luodaan tehtävä tuotteen tehtävälistalle. Tämä tehtävä linkitetään kohteena olevaan tehtävään ja luokitellaan avainsanalla "pia".
   7. Muut kohdan 4 tiedot dokumentoidaan mieluiten versioituun kirjoitusalustaan kuten wikiin, ja luodut korjaustehtävät linkitetään tähän dokumentaatioon. Organisaation tietosuojatoiminto voi osoittaa tälle dokumentaatiolle keskitetyn paikan. Muussa tapauksessa dokumentaatio kirjataan tietoturva-arkkitehtuurikuvauksen yhteyteen.
   8. Mikäli yllä kuvattujen analyysien ja uhkamallinuksen pohjalta päädytään siihen, että henkilötietoihin kohdistuu todennäköisesti korkea riski, otetaan yhteyttä <organisaatio>:n tietosuojavastaavaan. Tietosuojavastaava käynnistää tarvittaessa tietosuoja-asetuksen vaatiman tietosuojavaikutusten arvioinnin (DPIA). Todennäköisesti korkea riski muodostuu aina kun käsitellään suuria määriä henkilötietoja tai laajamittaisesti erityisiä henkilötietoluokkia ("arkaluontoisia" tietoja).

## Tietosuoja-asetuksen vaatima tietosuojavaikutusten arviointi (DPIA)

<organisaation> tietosuojavastaavan käynnistämässä ja ohjaamassa tietosuoja-asetuksen tarkoittamassa tietosuojavaikutusten arvioinnissa (*Data Protection Impact* Assessment, DPIA) kehitysprojekti toimittaa ainakin seuraavat tiedot:

1. järjestelmällinen kuvaus suunnitelluista käsittelytoimista ja käsittelyn tarkoituksista; ja
2. arvio käsittelytoimien tarpeellisuudesta ja oikeasuhteisuudesta tarkoituksiin nähden; ja
3. arvio rekisteröityjen oikeuksia ja vapauksia koskevista riskeistä; ja
4. suunnitellut toimenpiteet riskeihin puuttumiseksi, mukaan lukien suoja- ja turvallisuustoimet ja mekanismit.

Kohdan 4 osalta tuotetiimin tulisi pystyä identifioimaan ne tuotteen tehtävälistan tehtävät, joissa suunnitellut toimenpiteet on kuvattu.

Tietosuojavaikutusten arvioinnissa yllämainittujen kohtien teemat keskustellaan tietosuojavastaavan, tuoteomistajan ja tarvittavien tuotetiimin teknisten asiantuntijoiden kesken. Keskustelu dokumentoidaan. Mikäli keskustelun tuloksena vaaditaan toiminnallisia muutoksia, ne avataan uusina tehtävinä tuotteen tehtävälistalle samoin kuin uhkamallinnuksesta seuraaville uusille vaatimuksille tehdään. Nämä uudet tehtävät leimataan avainsanalla ”pia”.

# Tekninen liite (Liite 7, ohje)

## Palvelimen TLS-asetukset

### Yleistä TLS-asetuksien noudattamisesta

Käytettäessä pilvipalveluiden tarjoamia edustapalveluita, kuten CDN:ää (*content delivery network)* tai kuormantasaimia (*load balancer*), täyttä kontrollia TLS:n konfiguraatioon ei ehkä ole. Tällöin on käytettävä parhaiten vaatimuksia vastaavaa pilvipalveluntarjoajan TLS-konfiguraatiota. Riippuen arkkitehtuurista saattaa olla myös mahdollista käyttää kuormantasaukseen TCP/UDP-tason kuormantasainta (esimerkiksi AWS:n Network Load Balancer) ja purkaa TLS vasta sovelluksen omassa edustapalvelimessa (esimerkiksi Kubernetes-klusterin Ingress-resurssi). Tämä tietenkin siirtää osan protokollapinosta tuotetiimin vastuulle, joka ei välttämättä muista syistä ole toivottava tilanne.

Riskit, jotka syntyvät vaatimusten vastaisesta TLS-konfiguraatiosta verrattuna TLS:n purkuun lähellä työkuormia ovat erilaisia. Vaatimusten vastainen TLS-konfiguraatio liittyy lähinnä aktiivisten välityshyökkäysten ja laajamittaisen tiedustelun riskeihin. TLS:n purku (esimerkiksi) Kubernetes-klusterin sisällä taas tuo monimutkaisuutta ja lisää hyökkäyspintaa ja voi altistaa tätä kautta tietomurrolle.

Jos TLS-toteutusta aiotaan hallita itse, lähitulevaisuuden kannalta on tarpeellista ottaa huomioon HTTP/3 ja TLS 1.3, jotka tulevat yhtenäistämään TLS-vaatimuksia ja toisaalta tuomaan tarpeita uusille protokollatoteutuksille (QUIC) ja UDP:n tukemiselle siirtoprotokollana.

CDN:n osalta riskit ovat TLS:n osalta yleensä pienemmät, koska staattinen sisältö ei yleensä ole luottamuksellista. CDN:n sisällön suhteen suurempi huolenaihe on sisällön eheys. Tätä voidaan esimerkiksi ajettavan JavaScript-koodin osalta auttaa *subresource integrityllä* (ks. myöhemmin).

### Protokollaversio

Palveluiden tulee tukea TLS 1.2 -versiota ja tukea TLS 1.3:ta heti kun mahdollista. Nykyaikaiset selaimet tukevat lisääntyvässä määrin TLS 1.3 -versiota jo nyt.

TLS:n versiota 1.1 tulee käyttää ainoastaan, jos toinen osapuoli ei tue 1.2-versiota. On kuitenkin hyvin harvinaista, että osapuoli tukisi 1.1:tä mutta ei 1.2:ta. Vanhempia TLS- tai SSL-versioita (TLS 1.0, SSL) ei tule tukea.

Tuen poistaminen TLS 1.0 -protokollalta aiheuttaa yhteensopivuusongelmia lähinnä vanhempien mobiiliselainten kuten Android 4.4:n WebView’n, sekä Java 8:aa vanhempien versioiden kanssa, koska ne tukevat ainoastaan TLS 1.0:aa.

**Lähteet:**

1. Yhteensopivuustaulukot
   1. TLS 1.3-tuki <https://caniuse.com/#feat=tls1-3>
   2. TLS 1.2 -tuki <https://caniuse.com/#feat=tls1-2>
2. Viestintäviraston määräys 72A/2018 sähköisistä tunnistus- ja luottamuspalveluista, joka vaatii TLS 1.2:n käyttöä:<https://www.viestintavirasto.fi/attachments/maaraykset/M72A_2018_M.pdf>
3. HTTP/3: https://datatracker.ietf.org/doc/draft-ietf-quic-http/

### Varmenteet

Järjestelmissä, joilla ei ole kansallista suojaustasoa, varmenteiden tulee olla joko RSA-avaimia, jolloin tulee käyttää vähintään 2048-bittisiä avaimia, tai ECDSA-avaimia, jolloin tulee käyttää vähintään 224-bittisiä avaimia. Varmenteiden allekirjoitus tulee perustua SHA-256-tiivisteeseen. Nämä avaimenpituus- ja algoritmisuositukset tulee säännöllisesti tarkistaa, jotta edistysaskeleet kryptoanalyysissä otetaan huomioon. Pitkäikäisiksi tarkoitetuissa järjestelmissä pitää huomioida se, että käytettyjä algoritmeja ja avainpituuksia voidaan vaihtaa tarvittaessa.

Jos järjestelmälle on määritelty kansallinen suojaustaso, avainpituuksissa tulisi noudattaa Viestintäviraston kryptografisen vahvuuden ohjetta (ks. Lähteet).

Varmenteen vanhetessa tulisi luoda uusi avain ja tälle avaimelle hakea uusi varmenne. Varmenteet eivät saa olla wildcard-varmenteita, jotka ovat kelpoja kaikille alidomaineille.

Uusista myönnettävistä varmenteista tallennetaan tieto Certificate Transparency -lokeihin. Tämä mahdollistaa seuraamisen organisaatiolle myönnetyistä varmenteista. Tämä myös saattaa paljastaa sisäisesti käytettyjä palveluosoitteita, jos näille haetaan varmenne julkisesta lähteestä.

Varmenteiden voimassaolo voidaan tarkistaa OCSP-protokollalla (Online Certificate Status Protocol). OCSP:n käyttö aiheuttaa sen, että selain lähettää varmenteen myöntäjälle pyyntöjä ja nämä pyynnöt saattavat vuotaa tietoa selailusta varmenteen myöntäjälle. Tämän vuoksi palvelinten tulisi käyttää proaktiivista OCSP-vasteiden jakelua TLS-kättelyssä, joka tunnetaan nimellä OCSP Stapling. Tämä myös nopeuttaa yhteyksien luomista.

**Lähteet:**

* Kyberturvallisuuskeskuksen ohjeen 190/651/2015 ajantasainen versio: Kryptografiset vahvuusvaatimukset luottamuksellisuuden suojaamiseen – kansalliset suojaustasot https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/regulation/ohje-kryptografiset-vahvuusvaatimukset-kansalliset-suojaustasot.pdf
* <https://www.certificate-transparency.org/>
* Varmenneasetusten, mm. OCSP Staplingin tarkastaminen**:** <https://testssl.sh/>

### Varmenteiden myöntäjät

Käytettävien juurivarmenteiden tulee olla kaikkien palvelua käyttävien tahojen laitteissa oletuksena luotettuna. Tämän tulisi sisältää ainakin käytetyimmät selaimet, käyttöjärjestelmät ja mobiililaitteet. Käyttäjiä tulee tukea ymmärtämään varmennevirheiden tietoturvavaikutukset, kuten ohjeistaa olemaan hyväksymättä selaimen antamaa varoitusviestiä epäkelvosta varmenteesta.

### Salausalgoritmit

TLS 1.3:n algoritmit määritellään TLS 1.3-protokollan määrittelyissä ja toteutusten tulee seurata tätä, ellei toimivaltainen viranomainen ole antanut eri ohjetta tiettyä käyttötarkoitusta varten.

Jos käytössä on TLS 1.2, noudatetaan alla olevia ohjeita.

#### Avaintenvaihto (TLS 1.2)

Jos järjestelmälle ei ole määritetty kansallista suojaustasoa, avaintenvaihdossa on käytettävä ECDHE- tai DHE-menetelmiä. Käytettäessä ECDHE-menetelmää tulee avaimen pituuden olla vähintään 256 bittiä, DHE-menetelmää käytettäessä 2048 bittiä. Nämä avaimenpituus- ja algoritmisuositukset tulee säännöllisesti tarkistaa, jotta edistysaskeleet kryptoanalyysissä otetaan huomioon. Pitkäikäisiksi tarkoitetuissa järjestelmissä pitää huomioida se, että käytettyjä algoritmeja ja avainpituuksia voidaan vaihtaa tarvittaessa.

Jos järjestelmälle on määritelty kansallinen suojaustaso, avainpituuksissa tulisi noudattaa Viestintäviraston kryptografisen vahvuuden ohjetta (ks. Lähteet).

Käytettäessä elliptisten käyrien menetelmiä (ECC) tulee kaikki käytettävät käyrät luetella konfiguraatiossa.

Suositeltuja käytettäviä käyriä ovat BrainpoolP256r1, BrainpoolP384r1, BrainpoolP512r1, NIST Curve P-224, NIST Curve P-256, NIST Curve P-384 ja NIST Curve P-521.

#### Allekirjoitus (TLS 1.2)

Jos järjestelmälle ei ole määritetty kansallista suojaustasoa, allekirjoitusalgoritmina voidaan käyttää joko ECDSA- tai RSA-algoritmia. ECDSA-avaimen pituus tulee olla vähintään 256 bittiä ja RSA-avaimen 2048 bittiä. Nämä avaimenpituus- ja algoritmisuositukset tulee säännöllisesti tarkistaa, jotta edistysaskeleet kryptoanalyysissä otetaan huomioon. Pitkäikäisiksi tarkoitetuissa järjestelmissä pitää huomioida se, että käytettyjä algoritmeja ja avainpituuksia voidaan vaihtaa tarvittaessa.

Jos järjestelmälle on määritelty kansallinen suojaustaso, avainpituuksissa tulisi noudattaa Viestintäviraston kryptografisen vahvuuden ohjetta (ks. Lähteet).

#### Symmetrinen salaus (TLS 1.2)

Salausalgoritmin tulee olla AES käyttäen joko 128-bittistä tai 256-bittistä avainta. Salausmoodin on oltava CBC tai GCM. Nämä avaimenpituus- ja algoritmisuositukset tulee säännöllisesti tarkistaa, jotta edistysaskeleet kryptoanalyysissä otetaan huomioon. Pitkäikäisiksi tarkoitetuissa järjestelmissä pitää huomioida se, että käytettyjä algoritmeja ja avainpituuksia voidaan vaihtaa tarvittaessa.

Jos järjestelmälle on määritelty kansallinen suojaustaso, avainpituuksissa tulisi noudattaa Viestintäviraston kryptografisen vahvuuden ohjetta (ks. Lähteet).

#### Tiivistefunktiot (TLS 1.2)

Tiivistefunktion on oltava joko SHA-256, SHA-384, SHA-512 tai SHA-3. Nämä algoritmisuositukset tulee säännöllisesti tarkistaa, jotta edistysaskeleet kryptoanalyysissä otetaan huomioon. Pitkäikäisiksi tarkoitetuissa järjestelmissä pitää huomioida se, että käytettyjä algoritmeja ja avainpituuksia voidaan vaihtaa tarvittaessa.

Jos järjestelmälle on määritelty kansallinen suojaustaso, tiivistefunktion valinnassa tulisi noudattaa Viestintäviraston kryptografisen vahvuuden ohjetta (ks. Lähteet).

#### TLS-salausalgoritmimääritykset (TLS 1.2)

Järjestelmille, joille ei ole määritelty kansallista suojaustasoa, alla on listattu nämä ehdot täyttävät TLS 1.2-algoritmit:

DHE-RSA-AES256-GCM-SHA384  
ECDHE-ECDSA-AES256-GCM-SHA384  
ECDHE-RSA-AES256-GCM-SHA384  
DHE-RSA-AES128-GCM-SHA256  
ECDHE-ECDSA-AES128-GCM-SHA256  
ECDHE-RSA-AES128-GCM-SHA256  
DHE-RSA-AES256-SHA256  
ECDHE-ECDSA-AES256-SHA384  
ECDHE-RSA-AES256-SHA384  
DHE-RSA-AES128-SHA256  
ECDHE-ECDSA-AES128-SHA256  
ECDHE-RSA-AES128-SHA256

Lähteet:

* Viestintäviraston ohjeen 190/651/2015 ajantasainen versio: Kryptografiset vahvuusvaatimukset luottamuksellisuuden suojaamiseen – kansalliset suojaustasot <https://www.viestintavirasto.fi/attachments/tietoturva/Kryptografiset_vahvuusvaatimukset_-_kansalliset_suojaustasot.pdf>
* TLS 1.3, RFC 8446 https://tools.ietf.org/html/rfc8446
* eIDAS - Cryptographic requirements for the Interoperability Framework
* Määräys sähköisistä tunnistus- ja luottamuspalveluista, Viestintävirasto 72A/2018 M
* <https://wiki.mozilla.org/Security/Server_Side_TLS>
* <https://www.owasp.org/index.php/TLS_Cipher_String_Cheat_Sheet>
* Asetusgeneraattori: <https://mozilla.github.io/server-side-tls/ssl-config-generator/>
* Asetustentarkastaminen: <https://testssl.sh/>

## Muut kryptografiset toteutukset

Omien kryptografisten toteutuksien tekemistä tulee välttää. Käytettäessä kryptografisia kirjastoja on suositeltavaa käyttää tunnettuja ja aktiivisesti ylläpidettyjä kirjastoja, jotka tarjoavat korkean tason rajapinnat kryptografisiin operaatioihin. Matalan tason kryptografisten primitiivien käyttöä tulee välttää.

Algoritmi- ja avainpituusvalinnoissa seurataan TLS 1.2:n valintojen periaatteita (yllä).

### Satunnaisen tunnisteen luominen eri ohjelmointikielillä

#### Java

import java.util.SecureRandom;  
SecureRandom random = new SecureRandom();  
byte bytes[] = new byte[32];  
random.nextBytes(bytes);

#### Python 3.6 ja uudemmat

import secrets  
secrets.token\_urlsafe()

#### Python ennen versiota 3.6

import random  
generator = random.SystemRandom()  
generator.getrandbits(256)

#### JavaScript

var array = new Uint32Array(8);  
window.crypto.getRandomValues(array).join("");

## HTTP

HTTP-palvelut tulee kehittää ilman riippuvuuksia selaimeen asennettaviin lisäosiin (plugin) kuten esimerkiksi Javaan tai Flashiin.

### Evästeet

Evästeet (cookies) tulisi rajata mahdollisimman tiukasti ainoastaan niitä tarvitsevien palveluiden käyttöön. Mahdollisia kaikilla selaimilla toimivia rajauskeinoja ovat Path-parametri sekä Secure ja HttpOnly -attribuutit.

Evästeiden nimeämisessä kannattaa lisäksi käyttää \_\_Secure- -etuliitettä evästeille, jotka on asetettu HTTPS-lähteestä. Jos eväste voidaan rajata vain tiettyyn palvelinnimeen (alidomainit eivät tarvitse evästettä), nimi kannattaa aloittaa \_\_Host-, joka sisältää myös \_\_Secure- -toiminnallisuuden.

Kaikkien käytettyjen evästeiden tulisi poistua käyttäjän selaimesta kun selain suljetaan, ellei vastakkaiselle ole erikseen määriteltyä vaatimusta, jonka tietoturvariski on arvioitu. Tämä saavutetaan jättämällä evästeestä pois Expires-arvo.

### HTTP-otsakkeet selaimille

Otsakkeet on asetettava vähintään kaikille niille HTTP-vastauksille, joita kutsutaan web-selaimista.

Yleisesti on myös suositeltavaa asettaa nämä otsakkeet rajapinnoille, joita selaimien ei ole tarkoitus kutsua, erityisesti jos nämä rajapinnat ovat avoinna Internetiin. Lisää tietoa API-turvallisuudesta on tämän ohjeen API-turvallisuuskappaleessa.

#### Content Security Policy

Content Security Policy (CSP) -otsakkeesta on määritelty tasot 1 ja 2. Kaikki modernit selaimet tukevat CSP taso 2:ta. Huomionarvoisesti Internet Explorer 11 ei tue standardia CSP-otsaketta.

CSP-otsakkeella voidaan rajata, mistä lähteistä selain saa ladata sivulle esimerkiksi kuvia, skriptejä tai tyylitiedostoja.

On suositeltavaa asettaa niin tiukka CSP-otsake kuin suinkin mahdollista, sallien ainoastaan nimetyt lähteet. Mikäli tuki Internet Explorer 11 -selaimelle on tarpeen, myös vanhemmat tietoturvaotsakkeet (alempana) pitäisi asettaa.

**Esimerkkejä CSP-otsakkeista:**

Seuraava otsake sallii sisällön latauksen ainoastaan samasta lähteestä ja rajoittaa kehysten käyttöä:

Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; child-src 'self'; frame-ancestors 'none'

Seuraava otsake on sama kuin edellinen, mutta lisäksi raportoi poikkeukset raportointirajapintaan, josta voi olla hyötyä konfiguraatio-ongelmien löytämisessä:

Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; child-src 'self'; frame-ancestors 'none'; report-uri/some-report-uri

Seuraava otsake ainoastaan raportoi sääntöjä rikkovat pyynnöt, muttei rajoita niiden lataamista. Tätä vaihtoehtoa kannattaa käyttää vain palvelun kehitysvaiheessa, ei tuotannossa.

Content-Security-Policy-Report-Only: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; child-src 'self'; frame-ancestors 'none'; report-uri/some-report-uri

**Lähteet:**

* CSP-otsakkeen selaintukitaulukko <https://caniuse.com/#search=csp>
* <https://content-security-policy.com/>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
* Kaupallinen työkalu raporttien seurantaan <https://report-uri.com/>

#### HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) -otsake pakottaa selaimen käyttämään kaikissa tulevissa yhteyksissä sivustolle HTTPS-protokollaa. HSTS-otsake tulisi asettaa kaikille HTTPS-yhteyden kautta annettaville vastauksille, ja koska kaikki palvelut tulisi tarjota TLS:llä, HSTS tulisi asettaa kaikille vastauksille.

Säännölle annetaan yläraja sekunteina, jonka jälkeen voidaan taas muodostaa salaamattomia yhteyksiä. Sopiva arvo HSTS-otsakkeelle on 6 kuukautta eli 15 552 000 sekuntia. Palveluille, joita käytetään samalta selaimelta harvemmin (esimerkiksi kerran vuodessa), pidempi ajanjakso voi olla tarpeen. Sääntö voidaan myös ulottaa käsittämään kaikki aliosoitteet includeSubDomains -parametrilla.

HSTS-otsake saattaa estää palvelun käytön, jos palvelimen varmenne on unohdettu uusia tai jos kuormantasaus tai CDN-palvelu käyttää samaa osoitetta ilman toimivaa varmennetta.

HTTPS:n oletusarvoisen käytön varmistamiseksi sivusto voidaan myös ilmoittaa etukäteen selainvalmistajien ’preload’ -listalle (ks. Lähteet). Tällöin otsakkeeseen on lisättävä preload -direktiivi.

**Esimerkki:**

Strict-Transport-Security: max-age=15552000

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>
* Palvelun ilmoittaminen preload-listalle: <https://hstspreload.org/>

#### Vanhat otsakkeet

Seuraavat otsakkeet ovat poistuvia otsakkeita, jotka ovat korvautuneet selainten uusilla ominaisuuksilla tai Content Security Policy -otsakkeella. Koska TLS-vaatimukset estävät monien vanhojen selainten käytön, vanhat otsakkeet ovat tarpeen lähinnä Internet Explorer 11:n vuoksi. Kirjoitushetkellä näiden otsakkeiden tarve poistunee viimeistään Windows 10:n myötä 2025, elleivät uudet TLS-vaatimukset sulje IE 11:tä pois ennen sitä.

##### X-Content-Type-Options

Estää selaimia arvaamasta sisällön tyyppiä. Yleisesti ottaen kaikkien palvelimen vastausten tulisi aina määritellä sisällön tyyppi Content-Type -otsakkeella, jossa pitäisi myös määrittää käytetty merkistö, jos sisältötyyppi on tekstiä.

**Esimerkki:**

X-Content-Type-Options: nosniff

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>

##### X-XSS-Protection

Kertoo selaimelle ottaa käyttöön suojaukset XSS-hyökkäyksiä vastaan. Tämä on vanha otsake ja sisällytetty ainoastaan Internet Explorer 11:n vuoksi, joka ei tue Content Security Policyä. Moderni vaihtoehto on Content Security Policy, joka ei salli unsafe-inlineä.

**Esimerkki:**

X-XSS-Protection: 1; mode=block

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>

##### X-Frame-Options

X-Frame-Options -otsakkeella voidaan estää selaimia upottamasta sivua osaksi toista sivustoa. Tätä käytetään suojaamaan käyttäjiä clickjacking-hyökkäyksiltä, jossa käyttäjä houkutellaan painamaan selaimessa linkkiä, jonka "eteen" on asetettu näkymätön linkki, jota käyttäjä todellisuudessa tulee painaneeksi.

Nykyaikaisilla selaimilla Content Security Policy tason 2 frame-ancestors -direktiivillä saadaan aikaan sama efekti. Content Security Policyssä on lisäksi frame-src -direktiivi, jolla kontrolloidaan sivun itse lataamia kehyksiä, ja tätä X-Frame-Options ei kata.

**Esimerkkejä:**

X-Frame-Options: DENY  
X-Frame-Options: SAMEORIGIN  
X-Frame-Options: ALLOW-FROM <https://example.com/>

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors>

#### Cache-Control ja Expires

Cache-Control-otsakkeella ohjataan selaimen välimuistin ja mahdollisten välityspalvelimien toimintaa. Otsakkeella voidaan määrittää sisällölle vanhenemisaika, rajoittaa sisällön tallentamista välimuisteihin ja ohjeistaa välimuistia sisällön päivittämisestä.

Turvallisuusnäkökulmasta voi olla tarpeen estää luottamuksellisen tiedon tallennus välimuistiin. Seuraava otsake-esimerkki kieltää sisällön tallentamisen välimuistiin, kieltää kaiken HTTP-pyyntöön liittyvän tiedon tallentamisen ja pakottaa selaimen aina tekemään pyynnön uudestaan. Tätä otsaketta ei tulisi asettaa staattisten resurssien osalta suorituskykyvaikutusten vuoksi.

**Esimerkki:**

Cache-Control: no-cache, no-store, must-revalidate

Expires-otsake määrittää aikaleiman, jolloin kyseinen sisältö tulisi merkitä vanhentuneeksi. Nämä esimerkit määrittävät tiedon olevan jo valmiiksi vanhentunutta. Tämänkaltaista vanhenemisotsaketta tulisi käyttää yhdessä Cache-Control -otsakkeen kanssa.

**Esimerkkejä:**

Expires: Thu, 01 Jan 1970 00:00:00 GMT  
Expires: 0

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expires>

#### Public key pinning

Julkisen avaimen sitominen (public key pinning) tarkoittaa, että palvelimelta hyväksytään vain ennakkoon määritelty julkinen avain tai varmenne. Tämän tietoturvaominaisuuden tarkoitus oli pienentää riskiä siitä, että hyökkääjän avaimelle myönnetään varmenne, joka mahdollistaa palvelimeksi tekeytymisen.

Tuki julkisen avaimen sitomiselle on kuitenkin poistumassa selaimista, koska siitä ei tullut kovin suosittua ja koska sitä käytettäessä on suuri riski itseaiheutetusta palvelunestotilasta. Jotkin selaimet tukevat sitä edelleen, mutta huomionarvioisesti Chrome poisti sen käytöstä ja Safari ei koskaan tukenutkaan sitä.

Julkisen avaimen sitomista tulisi edelleen käyttää mobiilikehityksessä, jossa mobiilisovellukselle on helpompaa määritellä luotettujen varmenteiden joukko, ja luotetut varmenteet voidaan hätätilassa päivittää ohjelmistopäivityksellä sovelluskauppojen kautta.

Julkisen avaimen sitominen selaimissa tehdään HPKP-otsakkeella, Public-Key-Pins.

HPKP:n sijaan organisaation tulisi aktiivisesti seurata Certificate Transparency -lokeja ja reagoida, jos heidän hallussaan oleville domaineille myönnetään varmenteita ilman lupaa.

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning>
* <https://www.certificate-transparency.org/>

#### Referrer-policy ja noopener

Osana HTTP-otsakkeiden oikein määrittelyä kannattaa myös varmistaa, että uuteen välilehteen avautuvilla ulkoisilla (ja ei välttämättä luotetuilla) linkeillä on asetettu attribuutti rel=noopener. noopener-attribuutti estää sen, ettei uudessa välilehdessä avattu sivusto voi ohjata alkuperäistä välilehteä uudelleen esimerkiksi tietojenkalastelusivustolle. Mikäli tuki Internet Explorer 11 -selaimelle on tärkeää, tämän lisäksi asetetaan rel=noreferrer, jolla on sama (sivu)vaikutus.

Ulkoisiin kohteisiin linkitettyjen pyyntöjen mukana kulkee myös Referrer-otsake. Tämä saattaa vuotaa tietoja siitä, mitä henkilö oli tekemässä ennen kuin hän seurasi sivulla olevaa linkkiä. Väärin toteutetuissa sovelluksissa Referrer-tieto saattaa vuotaa myös esimerkiksi tunnisteita. Tämän vuoksi on suositeltavaa asettaa Referrer-Policy -otsake, esimerkiksi

Referrer-Policy: same-origin

Esimerkin otsake lähettää Referrer-tiedot vain samalle sivustolle kuin missä linkkikin oli. Näin sivuston omat tilastoinnit eivät mene rikki. Sivustoilla, joilla tieto siellä käymisestäkin saattaa olla arkaluontoista, otsakkeelle voi antaa arvon no-referrer.

**Lähteet:**

* noopener: https://html.spec.whatwg.org/multipage/links.html#link-type-noopener
* Referrer-Policy: https://www.w3.org/TR/referrer-policy/#referrer-policy-same-origin

### Subresource Integrity

Subresource integrity mahdollistaa vähemmän luotetusta lähteestä kuten esimerkiksi CDN-palveluista ladattavan JavaScript-kirjastojen eheyden tarkistamisen selaimessa. Subresource integrityä voidaan käyttää myös joidenkin toimitusketjuhyökkäysten estämiseen: subresource integrity hylkää hyökkääjän muuttamat JavaScript-riippuvuudet, ellei hyökkääjä muuta myös eheyden tarkistukseen käytettäviä tiivisteitä.

Subresource integrityä tulisi käyttää kaikelle JavaScriptille, joka ladataan kolmannen osapuolen hallussa olevasta sijainnista.

Käytettäessä subresource integrity -tarkisteita resurssit on haettava lähteestä, joka osoittaa resurssin tunnettuun ja muuttumattomaan versioon. Subresource integrityä käyttäessä ei ole mahdollista osoittaa ajoittain muuttuvaan, viimeisimpään (’latest’) versioon.

Esimerkkinä jQuery-kirjaston version 3.3.1 sisällyttäminen sivulle ja sen eheyden tarkistus SHA-256-tiivisteen avulla:

<script src="<https://code.jquery.com/jquery-3.3.1.min.js>"  
integrity="sha256-47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU="  
crossorigin="anonymous"></script>

Kirjaston SHA-256-tiiviste Base64-muodossa voidaan laskea seuraavalla komennolla, jos lähdeosoitteeseen voidaan luottaa komennon ajohetkellä:

curl -s <https://code.jquery.com/jquery-3.3.1.min.js> | openssl dgst -sha256 -binary | openssl base64 -A

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity>

### Istuntotunnisteet

HTTP-protokolla on tilaton, jolloin käyttäjän käyttämä selain on yksilöitävä jollain sovellustason parametrilla, ts. istuntotunnisteella. Turvallisia tapoja istuntotunnisteen välittämiseksi ovat evästeet, lomakkeiden kohdalla POST-pyynnön parametrit, ja JavaScript-kutsuissa HTTP-otsakkeet.

Mikäli käytössä on valmis ja luotettavana pidetty istuntotunnisteet hoitava sovelluskehikko, sitä tulee käyttää.

Istuntotunnisteena käytettävän tunnisteen tulee olla yksilöllinen, riittävän pitkä, ja kryptografisesti satunnainen. Suositeltu ratkaisu on käyttää vähintään 256 bittiä pitkää satunnaista tunnistetta.

Monissa tilanteissa on saatavissa UUID-tunniste. Jos käytetään UUID-tunnisteita, tulee varmistua, että käytetään UUIDv4-muotoisia tunnisteita, jotka generoidaan satunnaisesti. Tällöinkin tulee varmistua, että UUIDv4-toteutuksessa käytetään kryptografisesti vahvaa satunnaislukugeneraattoria. Tyypillisesti tämä sanotaan selvästi UUID-kirjaston dokumentaatiossa.

Istuntotunniste on luotava uudelleen aina kun käyttäjän tunnistautumisen tila tai valtuutustaso muuttuu (esimerkiksi sisäänkirjautumisessa, pääkäyttäjäksi siirtyessä tai kirjautuessa ulos). Edellisen istuntotunnisteen on tämänkaltaisissa tilanteissa vanhennuttava.

Jos sessiossa käytetään JWT:tä (JSON Web Token), sen kryptografiset vaatimukset noudattelevat TLS 1.2:n vaatimuksia soveltuvin osin (yllä).

**Lähteet:**

* UUIDv4: <https://en.wikipedia.org/wiki/Universally_unique_identifier#Variants>
* <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>

## Lokitus

Sovellusten tulisi tuottaa auditointilokia, joka mahdollistaa forensisen analyysin. Lokitiedot tulisi lähettää reaaliaikaisesti keskitetylle lokien keräily- ja analyysijärjestelmälle, jollaisen organisaatio yleensä tuottaa yhteisenä palveluna.

Jotta lokitiedot olisivat käyttökelpoisia forensiikkaan, on tärkeää, että lokitiedoissa on tarkat UTC-aikaan sidotut aikaleimat. Kaikkien palvelinten on oltava synkronoituja yhteiseen aikastandardiin, yleensä NTP:llä. Aikaleimojen tallennukseen tulisi käyttää yhteneväistä kielioppia, mieluiten RFC 3339:n mukaan.

Lisäksi erityisesti mikropalveluarkkitehtuureissa lokitietoihin tulisi tallentaa pyyntötunniste, jota voidaan käyttää korreloimaan lokitapahtumia eri mikropalveluiden välillä. Pyyntötunnisteesta on lisäkuvaus alempana.

Palveluiden tulisi lokittaa kaikki sisään tulevat rajapintapyynnöt. Lokiin tulisi merkitä myös pyynnön lähettänyt IP-osoite, vaikka palvelu olisikin auki vain sisäverkkoon. Mikäli palvelu on kuormantasaimen tai välityspalvelimen takana, välityspalvelin on konfiguroitava välittämään ulkoinen IP-osoite edelleen proxy-protokollalla tai X-Forwarded-For -otsakkeessa.

Lokitapahtumien tulisi yleensä olla JSON-objekteja. Tämä mahdollistaa uusien ja tapauskohtaisten tietoelementtien lisäämisen sekä helpon lokien jäsentämisen millä tahansa nykyaikaisella lokianalyysityökalulla.

Erityisesti ympäristössä, joka skaalautuu elastisesti kuorman mukaan, palvelujen tulisi tallentaa lokiin myös yksikäsitteinen instanssitunnisteensa, joka koostuu sovelluksen ajossa olevan komponentin nimestä ja esimerkiksi prosessin, virtuaalikoneen tai Kubernetes-podin tunnisteesta.

Kehittäjän tulee pystyä liittämään vapaamuotoista kontekstuaalista tietoa lokitapahtumiin. Tämän tiedon tulisi antaa tietoa siitä, mitä sovellus yritti kyseisessä tilanteessa tehdä. Jos sovellus saa lokitettavaan tapahtumaan liittyen toiselta sovellukselta ihmisluettavan virheviestin, se on hyvä liittää mukaan.

Tapahtumat, jotka tulisi kirjoittaa lokimerkintä, ovat:

* Henkilötietojen lukeminen, muuttaminen ja poisto. Lokikirjauksen tulisi sisältää sovelluksen sisäinen tunniste siitä henkilöstä, kenen tietoihin kajottiin ja kuka niihin kajosi, mutta yleensä ei henkilötietoja itsessään
* Henkilötietojen käytön suostumus ja suostumuksen peruminen, käyttöehtojen hyväksyminen, viestintäasetusten muuttuminen
* Muutokset istuntojen todennuksen tai valtuutuksen tilassa (sisäänkirjautuminen, käyttäjätason vaihtuminen, uloskirjautuminen)
* Istuntotunnisteiden tarkistuksen virheet
* Epäonnistuneet todennus- tai valtuutusyritykset

Seuraavat tapahtumat voivat generoida liikaa lokidataa, mutta niitä voidaan myös harkita auditointitarkoituksiin:

* Epäonnistuneet kutsut taustapalvelimelle tai muihin mikropalveluihin, koska tämä saattaa olla merkki vääristä kutsuparametreista. Lokiin tulisi tällöin kirjoittaa palautettu virheviesti vapaamuotoisena tietona, jos sellainen on.
* Epäonnistunut syötteen tarkastus (esimerkiksi rajapintakutsu, johon tulee vääränlainen syöte). Internetiin auki olevista avoimista rajapinnoista voi tulla liikaa tämänkaltaisia lokitapahtumia, mutta suljettujen rajapintojen osalta vialliset kutsut tulisi todennäköisesti aina kirjata lokiin.

**Lähteet:**

* RFC 3339: https://tools.ietf.org/html/rfc3339
* Proxy-protokolla: http://www.haproxy.org/download/2.2/doc/proxy-protocol.txt

### Pyyntötunnisteet

Pyyntötunniste on erityisesti mikropalveluarkkitehtuureille hyödyllinen tapa korreloida lokitietoja. Niiden avulla voidaan jälkikäteen rakentaa kuva siitä, mitä sisään tulleen pyynnön seurauksena tapahtui.

Sovelluksella saattaa olla käytössä hajautettu pyyntöjen seuranta (*distributed tracing*). Tällaista järjestelyä voidaan käyttää pyyntöjen seurantaan, mikäli se tuottaa riittävän pysyvän lokitiedon. Mikäli tällaista järjestelyä ei ole olemassa, pyyntötunnisteet voidaan toteuttaa myös seuraavasti.

Pyyntötunnisteen tulisi olla satunnainen tunniste, joka luodaan mahdollisimman aikaisessa vaiheessa sisään tulevia pyyntöjä käsitellessä (esimerkiksi kuormantasaimessa). Se lisätään HTTP-pyyntöön sopivassa otsakkeessa, kuten X-Request-Id. On huomattava, että ulkopuolelta syötetyt pyyntötunnisteet tulee poistaa, koska ne ovat hyökkääjän manipuloitavissa.

Jokainen palvelu (mikropalvelu), joka vastaanottaa X-Request-Id -arvon, kopioi vastaavan arvon kaikkiin muihin palveluihin lähettämiinsä pyyntöihin. Lisäksi palvelu tallentaa tämän arvon jokaiseen auditointilokitapahtumaan, jotka pyyntö aiheutti.

Yhdistettynä aikaleimoihin pyyntötunniste antaa mahdollisuuden muodostaa täydellisen kuvan mikropalveluiden kutsuhierarkiasta ja korreloida kaikki pyynnöt siihen ulkoiseen IP-osoitteeseen, josta alkuperäinen pyyntö tuli.

## Docker, Kubernetes ja Terraform

Dockerin ja konttien orkestrointiin käytettävien ohjelmistojen (mm. Kubernetes) kovennus on laaja kokonaisuus, jota varten esimerkiksi Center for Internet Security (CIS) on luonut omat ohjeensa. Tässä ohjeessa keskitytään lähinnä arkkitehtuuri- ja ohjelmistosuunnittelutason aspekteihin. Docker- tai orkestrointityökalun varsinaisen asennuksen kovennus tulee toki myös varmistaa.

### Isäntäkone

Docker-kontteja ajavien isäntäkoneiden tulee olla vain tässä käytössä. Palvelimella ei saa ajaa muita palveluja.

Yhdellä palvelimella tulee ajaa ainoastaan saman turvatason kontteja: esimerkiksi ulkoisille käyttäjille näkyvät Docker-kontissa ajettavat palvelut tulee ajaa eri palvelimella kuin missä taustapalvelinkontteja ajetaan.

Orkestrointijärjestelmät voivat antaa mahdollisuuksia konttien asetteluun virtuaalikoneille. Esimerkiksi Kubernetes tukee ominaisuutta nimeltä ’node restriction’.

**Lähteet:**

* CIS Docker Benchmark: https://www.cisecurity.org/benchmark/docker/
* <https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#node-isolation-restriction>

### Docker-kontin asetukset

Yksittäisen Docker-kontin tulee pohjautua luotettavaan pohjakuvaan (*base image*), johon on kontin rakennusvaiheessa asennettu ohjelmistoja vain luotetuista lähteistä. Lähteiden luotettavuus voi perustua kryptografisten allekirjoitusten tarkastamiseen tai siihen, että ne ovat organisaation hallussa.

Docker-konttien pohjakuvaan tulee viitata SHA-256-tunnisteella, eikä esimerkiksi viitata uusimpaan versioon latest-määreellä. Tätä viittaustapaa on käytettävä myös silloin, kun pohjakuva on sisäisesti luotu.

Alla esimerkki hyväksytystä tavasta viitata julkisesta lähteestä saatavaan pohjaan:

FROM alpine@sha256:3dcdb92d7432d56604d4545cbd324b14e647b313626d99b889d0626de158f73a

Kun kontteja rakennetaan Dockerfilellä, kaikkien asennettavien ohjelmien eheydestä on varmistuttava. Jos Dockerfile hakee ohjelmiston ei-luotetusta lähteestä, saman Dockerfilen ei pitäisi hakea eheystarkistukseen tarvittavia avaimia niin ikään ei-luotetusta lähteestä.

Riippuvuudet tulisi mieluiten asentaa paikallisesta varastosta. Tämä mahdollistaa sen, että organisaatio voi kontrolloida käytettyjä riippuvuuksien versioita, pitää yllä muutoshallintaprosessia, ja rakentaa Docker-kontit vaikka kolmannen osapuolen varasto ei olisi saatavilla.

Docker-kontit tulee aina ajaa read-only -tiedostojärjestelmällä, ellei ole erittäin painavaa syytä päättää toisin.

Docker-konttia ei tule koskaan ajaa --privileged-asetuksella tai antaa kontille pääsyä docker.sock-hallintaliittymään. Lisäksi Docker-kontille tulee antaa pääsy mahdollisimman pieneen osaan isäntäkoneen tiedostojärjestelmää ja verkkoa. Kontista ei saisi päästä kutsumaan esimerkiksi pilvipalveluntarjoajan virtuaalikoneille tarjoamaa metadata-rajapintaa.

Käytettäessä orkestrointijärjestelmää, joka tarjoaa nk. overlay-verkon, konttien verkkoliikenne tulisi rajata vain niihin kohteisiin, joiden kanssa niiden on pystyttävä kommunikoimaan. Esimerkiksi Kubernetes tuntee tämän nimellä NetworkPolicy.

Kun Dockeria tai orkestrointia käyttävälle järjestelmälle tehdään tietoturvatarkastus, Dockerfilen ja orkestrointijärjestelmän konfiguraation tarkastuksen on aina sisällyttävä tietoturvatarkastukseen.

**Lähteet:**

* CIS Docker Benchmark: https://www.cisecurity.org/benchmark/docker/
* <https://kubernetes.io/docs/concepts/services-networking/network-policies/>

### Kubernetes

Kubernetes-klustereilla on monia turvattomasta konfiguraatiosta johtuvia ongelmamahdollisuuksia. Mikäli tuotetiimi päättää käyttää itse hallittua Kubernetes-klusteria, heidän tulee huolehtia sen turvallisuudesta. Kuberneteksen kovennus on laaja aihealue ja erillisen kovennusohjeen ylläpito jää helposti ajastaan jälkeen. Tämän vuoksi on suositeltavaa käyttää CISin Kubernetes-kovennusohjetta.

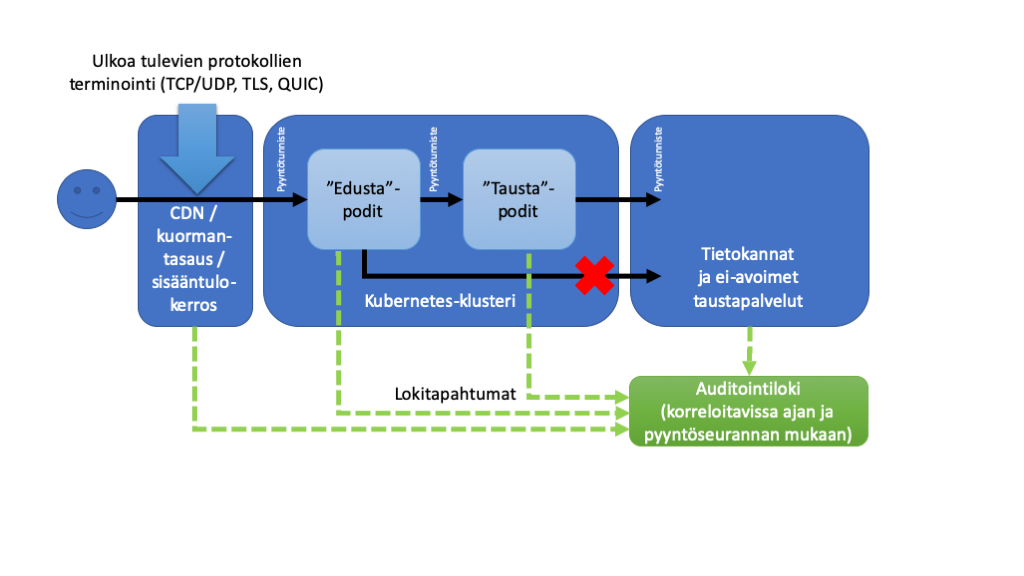
Ohjeen lisäksi on olemassa työkaluja, joilla klusterin asetuksien turvallisuutta voidaan arvioida. Tällaisia ovat esimerkiksi kube-bench CIS-kovennusohjeen tarkastukseen ja kube-hunter klusterin avoimien resurssien löytämiseen. Näiden työkalujen käyttö itse hallitun klusterin yhteydessä on suositeltavaa, joskaan ne eivät täydellisesti korvaa CIS-kovennusohjeen läpikäyntiä käsin. Mainituista työkaluista kube-hunteria voi ajaa myös klusterin sisäpuolelta, jolloin saadaan käsitys siitä, mitä hyökkääjä näkisi, jos se saisi Kubernetes-podin haltuunsa.

On suositeltavaa käyttää Kubernetesia valmiina palveluna (Platform-as-a-Service). Esimerkki tällaisesta on AWS:n Elastic Kubernetes Service (EKS). On kuitenkin huomattava, että jaetun vastuun periaate täytyy ymmärtää hyvin. Esimerkiksi jotkin Kubernetes-alustat vaativat, että alustan käyttäjän on huolehdittava klusteria pyörittävien koneiden päivityksestä itse. Vastuu ei ole välttämättä jakaantunut teknologiapinossa selkeää rajaa myöten. Jos pilvipalvelulla on kovennusopas Kubernetestä varten, tätä tulee noudattaa.

Kubernetes-palvelun tulee joka tapauksessa olla CNCF-sertifioitu (*CNCF Certified Kubernetes*).

Kubernetes perii useita Docker-turvallisuuden osa-alueita, jotka on lueteltu ylempänä. Näitä ovat esimerkiksi Kubernetes-podien ajaminen *read-only*-tiedostojärjestelmästä ja konttien ulkopuolisten tiedostojärjestelmien käytön välttäminen.

Kuberneteksessä ajettavissa sovelluksissa tulisi käyttää arkkitehtuuria, joka pakottaa hyökkääjän useampikerroksisen puolustuksen läpi. Esimerkki ajattelusta oheisessa kuvassa:



Perusajatuksena on se, että Kubernetes-podit luokitellaan erillisiin luottamusluokkiin sen mukaan, kuinka paljon hyökkäyspintaa ne avaavat mahdollisten hyökkääjien suuntaan. Yllä olevassa kuvassa edustapodit oletettavasti jäsentävät ja prosessoivat ulkopuolisen hyökkääjän sovelluskerroksen tietovirrat, joten niiden avaama hyökkäyspinta on taustapodeja isompi.

Tietokannat ja taustapalvelut tarvitsevat todennäköisesti salaisuuksia pääsynhallintaa varten, jolloin tässä esimerkissä salaisuudet annetaan vain taustapodeille.

Liikenne taustapodeille pakotetaan kulkemaan edustapodien kautta ja toisaalta liikenne tietokantoihin ja taustapalveluihin on sallittu vain taustapodien kautta. Tähän voi käyttää esimerkiksi NetworkPolicyä tai *service meshiä*.

Toisin kuin virtuaalikoneilla, podien välillä ei ole hypervisor-erottelua. Jos taustapalveluilla on erityisiä turvatarpeita, voi tämän vuoksi olla myös tarpeen harkita Kubernetesin *inter-pod anti-affinity* -asetusta, jonka perusteella Kubernetes laittaa määritellyt podit ajoon eri virtuaalikoneisiin. Hyvin korkeiden turvavaatimusten sovelluksia saattaa olla joskus tarpeen pilkkoa ajoon jopa eri pilvipalvelutileille, jolloin luonnollisesti niitä ajetaan kokonaan eri klusterissakin.

Yllä luottamustasoja on kaksi, mutta sovelluksissa niitä voi loogisesti olla useampiakin.

Tämänkaltainen tavoitearkkitehtuuri hankaloittaa hyökkääjän toimintaa, koska edustapodilta on vaikeampi vaikuttaa vapaavalintaisiin verkossa oleviin kohteisiin. Toisaalta edustapodin täydellinenkään hyökkääjän kontrolliin päätyminen ei vielä aiheuta esimerkiksi tietokantojen pääsyavaimien vuotamista, vaikka edustalta voidaankin tietysti tällöin tehdä pyyntöjä taustapodien läpi. Verkkoliikenteen rajoittaminen vain sallittuihin kohteisiin estää myös sen, ettei hyökkääjän haltuunsa saamilta podeilta voida liikennöidä esimerkiksi pilvipalvelun metadata- tai Kubernetes-klusterin hallinnollisiin rajapintoihin.

Huomionarvoista on myös se, että kuormantasauskerros tai sisääntulokerros (*ingress*) ei välttämättä puutu sovellustason dataan lainkaan, jolloin niiden vaikutus edustan kovennuksessa on pienempi.

Jos sovelluksen arkkitehtuuri on alun perin monoliittinen, sitä ei kannata suin päin muuttaa yllä kuvatun kaltaiseksi vaan tietoturvan tavoitearkkitehtuuri tulisi pitää mielessä silloin, kun sovelluksen arkkitehtuuria muutenkin muutetaan.

**Lähteet:**

* kube-bench: https://github.com/aquasecurity/kube-bench
* kube-hunter: https://github.com/aquasecurity/kube-hunter
* CNCF Certified Kubernetes: https://www.cncf.io/certification/software-conformance/
* Pod anti-affinity: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#inter-pod-affinity-and-anti-affinity
* NetworkPolicy: https://kubernetes.io/docs/concepts/services-networking/network-policies/#isolated-and-non-isolated-pods

### Terraform

Terraformin suurimmat tietoturvariskit liittyvät siihen, että jollakin osalla tuotantoonvientiautomaatiossa on oltava kyky muuttaa infrastruktuuria Terraform-konfiguraation mukaiseksi. Nämä oikeudet ovat varsin laajat. Infrastruktuuri koodina -ajattelua toteuttavalla tiimillä infrastruktuurin suojaus ei voi olla paremmalla tasolla kuin heidän tuotantoon vientiprosessiensa suojaus.

Terraform tallentaa infrastruktuurin vallitsevan tilan tilatiedostoon (*state file*). Terraform tukee tilan tallentamista jaettuun paikkaan, esimerkiksi AWS:n S3-ämpäriin. Tilatiedoston pääsynhallinnasta täytyy varmistua, jotta sen luvaton muokkaus ei ole mahdollista.

## Salaisuuksienhallinta

Salaisuuksia, kuten salasanoja, salaamattomia yksityisiä avaimia tai rajapintatunnuksia (*API token*), ei tule koskaan tallentaa selväkielisenä esimerkiksi versionhallintaan tai Dockerfile-tiedostoon. Pitkäaikaiseen säilöön salaisuudet kannattaa tallettaa salaisuuksienhallintajärjestelmällä. Näitä on erilaisia yksittäisen kehittäjän salasanamanagerista pilviasenteisiin holveihin (’vault’). Salaisuuksista tulisi myös olla varmuuskopio.

Ihmisten käsittelemien salaisuuksien hallintaan on käytettävä luotettavaa salasanamanageria.

Salaisuuksien toimittaminen ajettaville konteille on ympäristökohtainen ongelmakenttä, eikä yleispäteviä ratkaisuja ole helppo antaa. Yleisesti voidaan sanoa, että salaisuuksien viennissä tuotantoon tulisi mieluiten käyttää holvityyppisiä (’vault’) palveluita tai orkestrointityökalun tarjoamia mahdollisuuksia. Mikäli näitä ei ole saatavilla, salaisuudet voidaan laittaa salattuun tiedostoon, joka tallennetaan sopivaan paikkaan, johon kontti pääsee käsiksi. Tarvittava salauksenpurkuavain voidaan toimittaa kontille ympäristömuuttujassa. Käytettäessä ympäristömuuttujia salaisuuksien välittämiseen ne tulisi nollata niiden lukemisen jälkeen, koska monissa virhetilanteissa debug-tuloste sisältää ympäristön sisällön.

Salaisuuksien hallintaan on olemassa kolmansien osapuolien sovelluksia, kuten HashiCorp Vault. Kubernetesissa ja OpenShiftissä on ’secret’ -objekteja. Infrastruktuuripilvipalveluilla on kullakin oma avainten ja salaisuuksien hallintapalvelunsa, kuten AWS:n Secrets Manager. Orkestraattorin salaisuuksien hallinta on yleensä paras vaihtoehto. Jos käytetään pilvipalveluntarjoajan salaisuuksienhallintaratkaisua, jatkuvuudenhallintasyistä sen vaihtaminen (ja salaisuuksien uudelleen luonti) on suunniteltava valmiiksi.

**Lähteet:**

* <https://kubernetes.io/docs/concepts/configuration/secret/>
* https://docs.openshift.com/container-platform/4.2/nodes/pods/nodes-pods-secrets.html
* <https://www.vaultproject.io/>
* https://aws.amazon.com/secrets-manager/

## Ohjelmointirajapinnat (API)

Kaikki uudet web-ohjelmointirajapinnat tulisi toteuttaa käyttäen JSON-formaattia. Tätä puoltavat yksinkertainen esitystapa ja laaja ohjelmistotuki. Yleisesti käytetyissä JSONP- ja XML-formaateissa on molemmissa ongelmansa.

* JSONP-muotoisia rajapintoja ei tulisi käyttää, koska tällöin suoritetaan rajapinnan tarjoamaa dataa suoraan ohjelmakoodina. Tällöin erotus rajapinnan ja rajapintaa käyttävän palvelun välillä puuttuu täysin. Rajapinnan tulisi tarjota ainoastaan dataa, jota palvelu käsittelee.
* XML-jäsentimien (parser) on useasti osoitettu toimivan virheellisesti mahdollistaen datan tulkitsemisen usealla eri tavalla. Tämä muodostuu ongelmaksi, jos dataa tulkitaan useassa eri vaiheessa käyttäen eri jäsentimiä, jolloin dataa saatetaan tulkita useammalla eri tavalla. XML mahdollistaa myös tietosisällön esittämisen usealla eri esitystavalla, jonka jäsennin kuitenkin tulkitsee samalla tavalla.

### API-turvallisuus yleisesti

Ohjelmistorajapintojen tulisi varmistua jokaisesta sisään tulevasta kutsusta alla mainituista asioista. Arkkitehtuurista riippuen API-toteutus saattaa luottaa välissä olevaan API-välityspalvelimeen tai tehdä nämä tarkistukset itse.

Mikäli jokin tarkistus ei mene läpi, kutsu kannattaa suoraan hylätä ilman, että rajapinnan toteuttava osapuoli lähtisi arvailemaan, mitä kutsuja on tarkoittanut. Mikäli todennus-, valtuutus- tai eheystarkistukset epäonnistuvat, kutsun sisältöä ei pitäisi lähteä lainkaan jäsentämään pidemmälle vaan prosessointi tulisi päättää mahdollisimman tehokkaasti.

* Istuntotunniste: onko istunto voimassa? Mikä on istunnon valtuutuksen taso? (Ks. tämän ohjeen istuntotunnisteisiin liittyvä erityisohje.)
* Pyynnön valtuutus. Varmistutaan siitä, että kutsujalla on oikeus tehdä kysely tähän rajapintaan (esim. API-avain tai istunnon perusteella tehty valtuutus).
* Jos käytetään pyyntötunnisteita (*Request ID*, ks. yllä), onko kutsussa mukana pyyntötunniste? Jos on, ja teemme itse kutsuja eteenpäin muihin taustapalveluihin, liitämme pyyntötunnisteen näihin kutsuihin. Jos ei ole, generoimme pyyntötunnisteen itse (tai jos sen puuttuminen on epäilyttävää, jätämme pyynnön käsittelemättä).
* Kutsun ja tietojen eheys. Varmistutaan siitä, että kutsu on eheä. Jos käytetään API-avainta tai muuta kutsujan ja rajapinnan välistä salaisuutta, eheystarkistus voidaan sitoa siihen. Tyypillisiä toteutuksia ovat valtuutusten siirtoon JSON Web Tokens (JWT) ja kokonaisten rajapintakutsujen eheystarkistuksiin AWS Signature v4.
* Onko sisään tuleva kutsu oikein muotoiltu? Myös JSON-pohjaisten rajapintojen tapauksessa kutsu tulisi tarkistaa skeemaa vasten, jotta rajapintaan ei esimerkiksi pysty syöttämään ylimääräisiä tietokenttiä. Ellei skeematarkistusta tehdä, nämä ylimääräiset kentät päätyvät erityisesti NoSQL-tapauksissa usein tietokantaan asti ja joka tapauksessa ne indikoivat mahdollista hyökkäysyritystä. Muotoilun tarkastuksessa *Domain Driven Design* (DDD) saattaa olla hyödyllinen lähestymistapa, ks. *Secure by Design* alla.
* Valtuutus saada vastauksen tiedot. Joissakin tapauksissa rajapinnan kutsumisen valtuutus ja tietojen saanti rajapinnasta ovat erillisiä valtuutuspäätöksiä.
* Onko kutsu tarpeen lokittaa? (Ks. tämän ohjeen kappale auditointilokituksesta.)
* Asetetaanko vastauksessa kaikki tarpeelliset HTTP-otsakkeet, kuten HSTS ja Content-Type? (Ks. tämän ohjeen kappale HTTP-otsakkeista.)

Mikäli valtuutuksien siirtoon käytetään JWT:tä, sen käyttämien kryptografisten algoritmien ja avainpituuksien valintaan sovelletaan aiempaa TLS 1.2:n ohjetta soveltuvin osin.

**Lähteet:**

* JSON Web Tokens -teemasivusto <https://jwt.io/>
* AWS Signature v4, joka ei ole AWS-spesifinen <https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html>
* Secure by Design: https://www.manning.com/books/secure-by-design

### Cross-Origin Resource Sharing (CORS)

CORS on tapa, jolla palvelin ilmoittaa selaimelle, että se saa ohittaa normaalin nk. same-origin -rajoituksen ja kutsua toisessa domainissa sijaitsevaa APIa. Selaimet tekevät CORS-tarkistusta varten erillisen nk. preflight-kutsun, jonka perusteella saadaan tieto CORS-otsakkeista. On tärkeää huomata, että hyökkääjä voi tietenkin aina kutsua rajapintaa ilman selainta riippumatta CORSista. Palvelinpään pääsynhallinta tulee aina toteuttaa käyttäen oikeita pääsynhallintamekanismeja.

Kohdepalvelin ilmoittaa CORS-politiikkansa käyttäen Access-Control-Allow-Origin -otsaketta. Access-Control-Allow-Origin määrittelee ne JavaScript-koodin lähteet, jotka saavat kyseistä rajapintaa kutsua.

Jos esimerkiksi organisaatiolla on web-sovellus osoitteessa [app.company.com](http://app.company.com) ja API osoitteessa api.cloud.example, ja he haluavat app.company.comilta toimitetun JavaScriptin pystyä kutsumaan api.cloud.examplea, api.cloud.examplen rajapinnan on palautettava jotakin tämänkaltaista HTTP-otsakkeissaan:

Access-Control-Allow-Origin: [app.company.com](http://app.company.com)  
Access-Control-Allow-Methods: GET POST PUT DELETE

Mikäli rajapinta tarvitsee valtuutuksen (esimerkiksi evästeissä), sen tulee myös antaa selaimelle erityislupa näiden toimitukseen:

Access-Control-Allow-Credentials: true

Jos APIa pitää pystyä kutsumaan mistä tahansa JavaScript-koodista riippumatta sen alkuperästä, otsakkeen tulee olla

Access-Control-Allow-Origin: \*

Mikäli tämä otsake on näkyvissä rajapinnan otsakkeissa, mutta rajapinta ei ole tarkoitettu yleisesti kutsuttavaksi, CORS-politiikka on todennäköisesti liian lepsu.

**Lähteet:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
* <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy>

### CSRF-tunniste

Cross-Site Request Forgery (CSRF) -hyökkäyksessä hyökkäävä sivusto aiheuttaa selaimen kautta pyynnön jotakin toista sivustoa kohden. Jos selaimella on aktiivinen istunto kohdesivustolla, hyökkääjä voi tehdä käyttäjänä erilaisia operaatioita.

CSRF estetään antamalla jokaisen HTTP-pyynnön mukana kryptografisesti vahva ja riittävän pitkä salaisuus. Nämä salaisuudet (nk. CSRF-tokenit) tallennetaan selaimessa niin, että hyökkääjän sivusto tai siltä peräisin oleva JavaScript ei pääse niihin käsiksi, eikä siten pysty tekemään tekaistuja pyyntöjä.

CSRF on helpointa estää käyttämällä sovelluskehikkoa, joka toteuttaa CSRF-suojauksen automaattisesti. Useimmiten tämä on paras vaihtoehto.

Mikäli tällaista kehikkotukea ei ole, suositeltava tapa on käyttää nk. ”double submit cookie” -tapaa. Tämä metodi on kätevä, koska se on tilaton eikä vaadi palvelinpään tallennustilaa, joten se sopii myös tilanteisiin, jossa pyynnöt selaimelta tulevat satunnaisesti eri palvelimille kuormantasaajan läpi. Kuten monissa muissakin tietoturvaominaisuuksissa, tämän naiivi toteutus saattaa olla haavoittuva.

**Sources:**

* <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Double_Submit_Cookie>

### Parametrit ja luottamuksellinen tieto

HTTP-rajapintakyselyt tulee muotoilla niin, ettei henkilötietoja tai muita luottamuksellisia tietoja välitetä URL-osoitteen osana. URL-osoitteen osana olevat tiedot (ns. GET*-*parametrit) saattavat jäädä talteen lokeihin, selaimen välimuistiin ja selaushistoriaan tai välityspalvelimen tietoihin tai vuotaa Referrer-otsakkeissa.

Kaikki henkilötiedot tulee täten välittää HTTP POST -pyynnön parametreina.

Oikea tapa voisi näyttää esimerkiksi tältä:

POST /hae/osoite HTTP/1.1  
  
hetu=010170-123F

Väärä tapa voisi näyttää esimerkiksi tältä:

GET /hae/010170-123F/osoite HTTP/1.1

**Lähteet:**

<https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide> (erityisesti Mobile App Security Checklist)

<https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project> (ASVS 4.0)

<https://github.com/OWASP/CheatSheetSeries>

# Technical Appendix (Appendix 7, instructions)

## TLS settings of servers

### Recommendations on TLS in general

When using cloud service providers' fronting services such as CDNs (content delivery networks) or load balancers, it may not be possible to fully control the TLS configuration. In these cases, a configuration that most closely matches these requirements must be selected. Depending on the application architecture, it may also be possible to use a TCP/UDP layer load balancer (for example, a Network Load Balancer on AWS) and terminate TLS in the application's own front-end server (for example, an Ingress resource in a Kubernetes cluster). This approach would obviously shift an additional part of the maintenance responsibility to the development team, which may be undesirable for other reasons.

The risks posed by a non-compliant TLS configuration and allowing TLS termination close to workloads are different. Non-compliant TLS configuration risks are likely to mainly involve active man-in-the-middle attackers and enablement of wholesale surveillance. Terminating TLS (for example) within a Kubernetes cluster introduces complexity and additional attack surface, which may increase a risk of a server compromise.

If the intent is to manage a TLS termination point, a future-proof architecture should consider the imminent adoption of HTTP/3 and TLS 1.3, which is likely to harmonise TLS compliance requirements and introduce new protocol implementations (QUIC) and require the support of UDP for ingress traffic.

TLS-related risks are usually smaller as far as a CDN is concerned, as static content is not that often confidential. Integrity of CDN-supplied content would likely be a more pertinent question. For executable JavaScript content, integrity can be further assured through subresource integrity (see later in this document).

### Protocol version

Services must support TLS version 1.2, and support TLS 1.3 as soon as it is possible. Modern browsers increasingly support TLS 1.3.

TLS version 1.1 may only be used in situations where the other endpoint does not support TLS version 1.2. However, an endpoint that would support 1.1 but not 1.2 are very rare. Older TLS or SSL versions (TLS 1.0, SSL) must not be supported.

Removing TLS 1.0 support will cause compatibility problems mostly with older mobile browsers, such as Android 4.4 WebView that only supports TLS 1.0, and Java 7 and older.

**Sources:**

* Compatibility tables –
  + TLS 1.3 support <https://caniuse.com/#feat=tls1-3>
  + TLS 1.2 support <https://caniuse.com/#feat=tls1-2>
* Ficora (Traficom) regulation 72A/2018 on Electronic Identification and Trust Services requiring the use of TLS 1.2 https://www.finlex.fi/data/normit/42947/M72A-2018-EN-v2.pdf
* HTTP/3: https://datatracker.ietf.org/doc/draft-ietf-quic-http/

### Certificates

For services that do not have a national security level classification, certificates must use RSA keys, in which case the minimum key length is 2048 bits, or ECDSA keys, in which case the minimum key length is 224 bits. Certificates must use SHA-256 as their hashing algorithm. These key lengths and algorithm choices need to be revised regularly to keep current with cryptanalytic progress. Systems designed for a long lifespan should allow for changes in used algorithms and key lengths.

For services which have a national security level classification, key lengths should follow the cryptographic strength requirements from the National Cyber Security Centre (see Sources).

When certificates expire, new keys should be generated, and certificates should be created for those new keys. Certificates must not be wildcard certificates that would be accepted for all subdomains.

As new certificates are being issued, issuance is noted in Certificate Transparency logs. This allows external parties to follow which certificates have been issued to an organization. Additionally, if certificates are issued for internal service domains, this information may leak.

Certificate validity can be checked by the browser by using OCSP (Online Certificate Status Protocol). However, using OCSP causes the browser to send OCSP requests to the certificate issuer and this may leak information about the browsing activity to the certificate issuer. Therefore, TLS endpoints should enable OCSP Stapling, where the server retrieves OCSP responses and proactively provides them to browsers in the TLS handshake. This also speeds up the connection.

**Sources:**

* National Cyber Security Centre guideline 190/651/2015 (as current): Cryptographic strength requirements for confidentiality protection – national security levels (in Finnish) https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/regulation/ohje-kryptografiset-vahvuusvaatimukset-kansalliset-suojaustasot.pdf
* <https://www.certificate-transparency.org/>
* Checking certificate settings, including OCSP Stapling**:** <https://testssl.sh/>

### Certificate issuers

Certificate issuers must have valid trust roots in all of the endpoints that connect to the service, such as most common browsers, operating systems and mobile devices. Users must be educated to understand security implications of certificate error messages, for example, not to accept warning messages on untrusted certificates.

### Encryption algorithms

The required TLS 1.3 algorithms are defined in the TLS 1.3 specification and implementations need to be compliant with these, unless a competent authority defines otherwise for a specific application.

When using TLS 1.2, the guidance below should be followed.

#### Key agreement (TLS 1.2)

For services that do not have a national security level classification, key agreement must use either ECDHE or DHE algorithms. When using ECDHE, the minimum key length is 256 bits, and for DHE, 2048 bits. When using elliptic curve cryptography (ECC), all acceptable curves must be explicitly specified in the protocol configuration. These key lengths and algorithm choices need to be revised regularly to keep current with cryptanalytic progress. Systems designed for a long lifespan should allow for changes in used algorithms and key lengths.

For services which have a national security level classification, key lengths should follow the cryptographic strength guideline from the National Cyber Security Centre (see Sources).

Recommended elliptic curves are BrainpoolP256r1, BrainpoolP384r1, BrainpoolP512r1, NIST Curve P-224, NIST Curve P-256, NIST Curve P-384 and NIST Curve P-521.

#### Signatures (TLS 1.2)

For services that do not have a national security level classification, signatures must use either ECDSA or RSA algorithms. The minimum key length for ECDSA is 256 bits, and for RSA, 2048 bits. These key lengths and algorithm choices need to be revised regularly to keep current with cryptanalytic progress. Systems designed for a long lifespan should allow for changes in used algorithms and key lengths.

For services which have a national security level classification, key lengths should follow the cryptographic strength guideline from the National Cyber Security Centre (see Sources).

#### Symmetric encryption (TLS 1.2)

The encryption algorithm used must be AES in CBC or GCM mode, either with a 128 or 256 bit key. These key lengths and algorithm choices need to be revised regularly to keep current with cryptanalytic progress. Systems designed for a long lifespan should allow for changes in used algorithms and key lengths.

For services which have a national security level classification, key lengths should follow the cryptographic strength guideline from the National Cyber Security Centre (see Sources).

#### Hash functions (TLS 1.2)

Hash functions must be either SHA-256, SHA-384, SHA-512 or SHA-3. These algorithm choices need to be revised regularly to keep current with cryptanalytic progress. Systems designed for a long lifespan should allow for changes in used algorithms and key lengths.

For services which have a national security level classification, hash function choice should follow the cryptographic strength guideline from the National Cyber Security Centre (see Sources).

#### TLS cipher suites (TLS 1.2)

For services that do not have a specified national security level, the following TLS 1.2 cipher suites fulfil the requirements above.

DHE-RSA-AES256-GCM-SHA384  
ECDHE-ECDSA-AES256-GCM-SHA384  
ECDHE-RSA-AES256-GCM-SHA384  
DHE-RSA-AES128-GCM-SHA256  
ECDHE-ECDSA-AES128-GCM-SHA256  
ECDHE-RSA-AES128-GCM-SHA256  
DHE-RSA-AES256-SHA256  
ECDHE-ECDSA-AES256-SHA384  
ECDHE-RSA-AES256-SHA384  
DHE-RSA-AES128-SHA256  
ECDHE-ECDSA-AES128-SHA256  
ECDHE-RSA-AES128-SHA256

**Sources:**

* National Cyber Security Centre guideline 190/651/2015 (as current): Cryptographic strength requirements for confidentiality protection – national security levels (in Finnish) https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/regulation/ohje-kryptografiset-vahvuusvaatimukset-kansalliset-suojaustasot.pdf
* TLS 1.3, RFC 8446 https://tools.ietf.org/html/rfc8446
* eIDAS - Cryptographic requirements for the Interoperability Framework
* Ficora (Traficom) regulation 72A/2018 on Electronic Identification and Trust Services requiring the use of TLS 1.2 https://www.finlex.fi/data/normit/42947/M72A-2018-EN-v2.pdf
* <https://wiki.mozilla.org/Security/Server_Side_TLS>
* <https://www.owasp.org/index.php/TLS_Cipher_String_Cheat_Sheet>
* TLS settings generator: <https://mozilla.github.io/server-side-tls/ssl-config-generator/>
* Verifying TLS settings: <https://testssl.sh/>

## Other cryptographic implementations

Using self-made cryptographic implementations is to be avoided. The recommendation is to only use well-known and actively maintained libraries which offer high-level access to cryptographic operations. The use of low-level cryptographic primitives should be avoided.

In algorithm and key level choices, the principles laid out for TLS 1.2 (above) should be applied.

### Creating random identifiers in various programming languages

#### Java

import java.util.SecureRandom;  
SecureRandom random = new SecureRandom();  
byte bytes[] = new byte[32];  
random.nextBytes(bytes);

#### Python 3.6 and newer

import secrets  
secrets.token\_urlsafe()

#### Python before version 3.6

import random  
generator = random.SystemRandom()  
generator.getrandbits(256)

#### JavaScript

var array = new Uint32Array(8);  
window.crypto.getRandomValues(array).join("");

## HTTP

HTTP based services must not depend on browser extensions (plugins), such as Java applets or Adobe Flash.

### Cookies

Cookies should be scoped as strictly as possible only for the services that rely on them. This can be done for all browsers using the Path parameter and Secure and HttpOnly attributes.

When naming cookies, the cookie name should additionally start with \_\_Secure- if it is set from a HTTPS source. If the cookie can be scoped to a specific host (subdomains do not require the cookie), it should start with \_\_Host- which also implies \_\_Secure-.

All cookies should be purged from the users’ browsers when the browser is closed, unless there is an explicit conflicting requirement that has been analysed for its security impact. This can be done by leaving out the Expires parameter of the cookie.

### HTTP headers towards browsers

These headers must be set for any content provided to web browsers.

In general, it is also recommended to set these headers for APIs that are not supposed to be called by browsers, especially if those APIs are visible to the Internet. For further API security considerations, please refer to this guideline’s API security section.

#### Content Security Policy

There are two levels (1 and 2) of the Content Security Policy header. All modern browsers support CSP Level 2. Notably, Internet Explorer 11 does not support the standard CSP header.

A CSP header can be used to restrict the sources from where the browser may load images, scripts or style information.

It is recommended to set a CSP level 2 header that is as strict as possible, using a whitelisting approach. If support for Internet Explorer 11 is required, also the legacy headers (below) should be set.

**Examples of CSP headers:**

The following header only allows content from the same source, and also restricts framing:

Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; frame-src 'self'; worker-src ‘self’; frame-ancestors 'none'

The following header is the same as above, but also reports problems towards a reporting endpoint that may be useful for detecting configuration problems:

Content-Security-Policy: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; frame-src 'self'; worker-src ‘self’; frame-ancestors 'none'; report-uri/some-report-uri

The following header only reports non-compliant requests but does not actually restrict them. This is only meaningful for testing, and not for production.

Content-Security-Policy-Report-Only: default-src 'none'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src 'self'; frame-src 'self'; worker-src ‘self’; frame-ancestors 'none'; report-uri/some-report-uri

**Sources:**

* Browser compatibility tables for CSP header <https://caniuse.com/#search=csp>
* <https://content-security-policy.com/>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
* A commercial tool for following up on non-compliance reports <https://report-uri.com/>

#### HTTP Strict Transport Security

The HTTP Strict Transport Security (HSTS) header forces the browser to use HTTPS for all subsequent requests towards the server. The HSTS header should be set for every response over HTTPS, and because everything should be served over TLS, this means HSTS should be set for every request.

The header has an expiration time, specified in seconds, after which unencrypted connections become possible again. A suitable starting point for a HSTS expiration is 6 months (15 552 000 seconds). For services that are visited less often (e.g., once a year), a longer expiration time may be useful. The header may also be specified to include all subdomains using the includeSubDomains parameter.

Using a HSTS header may cause a self-inflicted Denial of Service condition in the case that the service’s certificate expires, or a load balancer or a content delivery network is using the same address without a proper certificate.

For additional assurance of browsers using HTTPS by default, your site may be submitted to the browser vendors’ ‘preload’ list (see Sources). This requires the header to include the preload directive.

**An example:**

Strict-Transport-Security: max-age=15552000

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>
* Submitting services on the preload list: <https://hstspreload.org/>

#### Legacy headers

The following headers have been deprecated and mostly replaced with new browser features and the Content Security Policy header. However, for some browsers (given the TLS 1.2 requirement that also weeds out old browsers, most notably Internet Explorer 11), these would still be useful for some time. At the time of writing, the need for these headers should cease latest in 2025 with the phase-out of Windows 10 support, unless TLS requirements block IE 11 earlier.

##### X-Content-Type-Options

Forbids browsers from guessing the content type. Generally, all responses from the server should always define the content type using the Content-Type header, including the character set, if applicable.

**An example:**

X-Content-Type-Options: nosniff

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>

##### X-XSS-Protection

Instructs the browser to use its internal heuristics against Cross-Site Scripting (XSS) attacks. This is an old header, included only for Internet Explorer 11 that does not support Content Security Policy. A modern option is a Content Security Policy that disallows unsafe-inline.

**An example:**

X-XSS-Protection: 1; mode=block

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>

##### X-Frame-Options

The X-Frame-Options header prohibits browsers from embedding the page within another page using frames. This protects against a clickjacking attack, where the user would be clicking on a link that is behind an invisible link that actually gets activated.

For modern browsers, Content Security Policy level 2 frame-ancestors directive can be used for the same effect. The Content Security Policy frame-src directive controls the loading of frames within the current page, which is not covered by X-Frame-Options.

**Examples:**

X-Frame-Options: DENY  
X-Frame-Options: SAMEORIGIN  
X-Frame-Options: ALLOW-FROM <https://example.com/>

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors>

#### Cache-Control and Expires

The Cache-Control header controls the functionality of the browser cache and caching proxies. The header can be used to specify an expiration time for content, restrict caching on intervening proxies, and provide information for cache refreshes.

From the security perspective, it may be useful to restrict caching of confidential data. The following example prohibits storing any information about the HTTP request or response and forces a new request to be made. These headers should likely not be set for static resources due to performance reasons.

**An example:**

Cache-Control: no-cache, no-store, must-revalidate

The Expires header specifies a time after which content will be marked stale in a cache. These examples prohibit caching by immediate expiry of content. This type of expiry should be used in conjunction with a Cache-Control header.

**Examples:**

Expires: Thu, 01 Jan 1970 00:00:00 GMT  
Expires: 0

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Expires>

#### Public key pinning

Public key pinning means only accepting a specific public key (or certificate) for a server. This security feature was intended to reduce the risk of certificates being issued for attacker’s keys and then used in a spoofing attack.

However, support for the public key pinning headers is being phased out in browsers, as it failed to gather momentum and because it introduces a high risk for self-inflicted denial of service. It is still supported in some browsers, but notably was deprecated by Chrome, and never supported by Safari.

Public key pinning should still be used in mobile application development, where it is easier to specify the acceptable trust roots, and those can be updated in an emergency through an application update through the app store.

Public key pinning in browsers is implemented using a HPKP header, Public-Key-Pins.

Instead of HPKP, the organization should actively monitor Certificate Transparency logs and react if they notice unauthorized certificates being issued for their domains.

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning>
* <https://www.certificate-transparency.org/>

#### Referrer-Policy and noopener

As a part of hardening the HTTP headers it is useful to ensure that links that open external, potentially untrusted pages into new tabs have set the attribute rel=noopener. The noopener attribute prevents the newly opened tab from redirecting the original tab, for example on a phishing page. If Internet Explorer 11 needs to be supported, an additional rel=noreferrer attribute will have the same (side) effect.

External links will also cause a Referrer header to be sent with the requests. These headers may leak information about what the person was doing before following a link on a page. In incorrectly implemented applications, Referrer headers have also been known to leak various identifiers. For these reasons, it is recommended to set a Referrer-Policy header, for example

Referrer-Policy: same-origin

This example will only provide Referrer information if the link points towards the original website. This allows using the Referrer header locally for statistical purposes. On web sites where even the fact that a person is visiting the site may be sensitive, a no-referrer value can be useful.

**Sources:**

* noopener: https://html.spec.whatwg.org/multipage/links.html#link-type-noopener
* Referrer-Policy: https://www.w3.org/TR/referrer-policy/#referrer-policy-same-origin

### Subresource Integrity

Subresource integrity is a method to ensure the integrity of resources (for example, JavaScript libraries) that are loaded from a potentially untrusted source, such as a content delivery network. Subresource integrity can also be used to deter supply chain attacks in some cases; unauthorized changes to JavaScript dependencies would be rejected by subresource integrity unless the integrity checksums would also be changed.

Subresource integrity should be used for any JavaScript that is loaded from a location that is under a third-party control.

When using subresource integrity, the resources need to be fetched from a source which points to a known and immutable version of the resource. It is not possible to point to a changing ‘latest’ version of a resource when using subresource integrity.

As an example, embedding the version of jQuery 3.3.1 while ensuring its integrity using subresource integrity would look like this:

<script src="<https://code.jquery.com/jquery-3.3.1.min.js>"  
integrity="sha256-47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU="  
crossorigin="anonymous"></script>

The following command can be used to create a base64 encoded SHA-256 hash (assuming the source is trusted at the moment of running the command):

curl -s <https://code.jquery.com/jquery-3.3.1.min.js> | openssl dgst -sha256 -binary | openssl base64 -A

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity>

### Session identifiers

The HTTP protocol is stateless. The browser must therefore be uniquely identified using an application layer parameter, namely a session identifier. Secure ways of implementing session identifiers include cookies, POST-type form submission parameters, and custom HTTP headers added with JavaScript.

If a third-party, trustworthy application framework is available providing session identifiers, that must be used.

Session identifiers must be unique and cryptographically random. A recommended length for a random identifier is at least 256 bits.

In many contexts, UUIDs are available. In this case, the only acceptable version of UUIDs is UUIDv4, which are random. Even in this case, it needs to be ensured that the UUIDv4 is generated using a cryptographically strong random number generator. Usually this would be explicitly stated in the UUID function documentation.

Session identifiers must be recreated every time there is a change in the authentication status or authorization level of the session (for example, an anonymous user logging in, or a user elevating themselves to an admin user, or a user logging out). The previous session identifier must expire when such a change occurs.

When JWT (JSON Web Tokens) are used for session management, its cryptographic requirements follow the TLS 1.2 requirements as applicable (see above).

**Sources:**

* UUIDv4: <https://en.wikipedia.org/wiki/Universally_unique_identifier#Variants>
* <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>

## Logging

Applications should produce audit log records that allow forensic analysis. Log data should be sent in real time to a central log aggregation and analysis system, which would usually be provided by the organization as a common service.

In order for log data to be useful for forensics, it is important that log items contain accurate timestamps in UTC. All servers must be synchronized to a common time standard, typically using NTP. Timestamps should be stored using a unified syntax. RFC 3339 syntax is recommended.

Additionally, especially in a microservice architecture, log items should have a ‘request ID’ that can be used to correlate log items of different microservices together. For details of this pattern, see below.

Services should log all incoming API requests. They should always log the IP address from where a request originated, even when the service would only be visible to an internal network. If the service is behind a load balancer or a proxy, the proxy must be configured to pass the external IP address towards the service using the proxy protocol or in an X-Forwarded-For header.

Logs should usually consist of JSON objects. This allows the addition of new and custom data elements, and easy parsing of log items in any modern log analysis software.

Especially in a deployment that scales elastically under load, the service should put their unique instance identifier into the log items. The instance identifier should include the name of the running software component augmented with an identifier of the process, virtual machine instance, or a Kubernetes pod.

The developer must be able to add free-form contextual information in the log events. This information is intended to provide information on what the application tried to perform at the time. If an application receives a human-readable error message from another service, that message should be included.

The actions that should be audit logged include:

* Reading, changing or deleting personal data. This log event should contain an application-internal identifier of the person whose personal data was targeted, but usually not the personal data itself
* Consent for using personal data, withdrawal of consent, approval of terms of service, changing communication preferences
* Changes in session authentication or authorization state (log in, change of user privilege level, log out)
* Failure to verify session identifiers
* Failure to authenticate or authorize an action

The following may generate too much log data, but they may be useful for auditing purposes:

* A failed request towards a back-end server (or another microservice), as this may be an indication of invalid input provided. Returned error messages should be written into log events as freeform information.
* Failure to validate input (e.g., an API receives a malformed request). Open, Internet-facing APIs may generate too many events of this type. Closed APIs should always log these failures.

**Sources:**

* RFC 3339: https://tools.ietf.org/html/rfc3339
* The proxy protocol: http://www.haproxy.org/download/2.2/doc/proxy-protocol.txt

### The request identifier pattern

For microservice architectures, a request identifier pattern may help correlating various log items together to reconstruct what happened as a result of an incoming request.

The application may be using a distributed request tracing solution. If so, this can be used for tracking request identifiers if it provides a sufficiently permanent log. Unless distributed tracing is in use, this functionality can be built as follows.

The request identifier should be a random identifier that is created at first opportunity when there is an incoming request (e.g., in a load balancer). It would be added to the HTTP request headers in a suitable custom header such as X-Request-Id. Note that externally supplied request identifier headers would need to be removed as these are under attacker control.

Every service (microservice) that receives an X-Request-Id would copy that value into all outgoing requests towards other services. The service would also store this identifier with all the audit log events that are generated as a result of this request.

Paired with timestamps, a request identifier allows forensic reconstruction of the complete microservice call tree, and its correlation with the external IP address from where the original request originated.

## Docker, Kubernetes and Terraform

Hardening Docker and orchestration tools (such as Kubernetes) is a more complicated topic. The Center for Internet Security (CIS) has published its own guidelines for hardening these systems. This guideline concentrates on architectural and design level aspects. The hardening of Docker and orchestration tool installation would also have to be done.

### Docker hosts

Servers who are running Docker containers (hosts) must be dedicated to hosting containers. These hosts must not be used for other purposes.

One server may only run containers that have an identical trust level. As an example, Docker containers supplying externally exposed services should be run on a different host than those running back-end services.

When using an orchestration system, the placement of containers on different VMs can sometimes be controlled. For example, Kubernetes has the possibility of node restrictions.

**Sources:**

* CIS Docker Benchmark: https://www.cisecurity.org/benchmark/docker/
* <https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#node-isolation-restriction>

### Docker containers

Any single Docker container must be built on a trusted base image, and any software built in have to originate from a trusted source. This trustworthiness can be based on verification of cryptographic signatures or the fact that the source is under the organization’s control.

The base images must be pinned to a specific version using a SHA-256 hash value, even if the base image would be internal. This means that using a latest base image is not possible.

An example of how a base image can be pinned to a specific version:

FROM alpine@sha256:3dcdb92d7432d56604d4545cbd324b14e647b313626d99b889d0626de158f73a

Building containers with Dockerfiles must ensure that any installed software is integrity checked. If the Dockerfile downloads software from an untrusted source, the same Dockerfile should not also obtain the integrity verification keys from a similarly untrusted source.

Dependencies should preferably be installed from a local repository. This allows the organization to control which version of the dependency is being used, to exert change control, and to build the Docker image even if the third-party repository would be unavailable.

Docker containers should always be run with read-only root filesystems unless there is a serious technical rationale of not to.

Docker containers must never be run using the --privileged option or provide access to the docker.sock management interface. Docker containers should be provided with as little host file system and network access as possible. For example, applications within container should not be able to call the cloud service providers' metadata endpoints provided to cloud virtual machines.

When using an orchestration system that provides an overlay network, the containers’ network traffic should be restricted to only those destinations that are necessary for the container to work. For example, in Kubernetes, this is known as NetworkPolicy.

When performing a security assessment for a system that uses Docker or an orchestration system, a review of Dockerfile and the orchestration system’s configuration files must always be included in the scope of the security assessment.

**Sources:**

* CIS Docker Benchmark: https://www.cisecurity.org/benchmark/docker/
* <https://kubernetes.io/docs/concepts/services-networking/network-policies/>

### Kubernetes

Kubernetes clusters have multiple failure modes that stem from insecure cluster configuration. If the development team decides to use a self-managed Kubernetes cluster, they would need to address these potential weaknesses. Hardening Kubernetes is a large topic and maintaining a hardening guide separately would easily fall behind the times. The CIS Benchmark for Kubernetes is a good source for this.

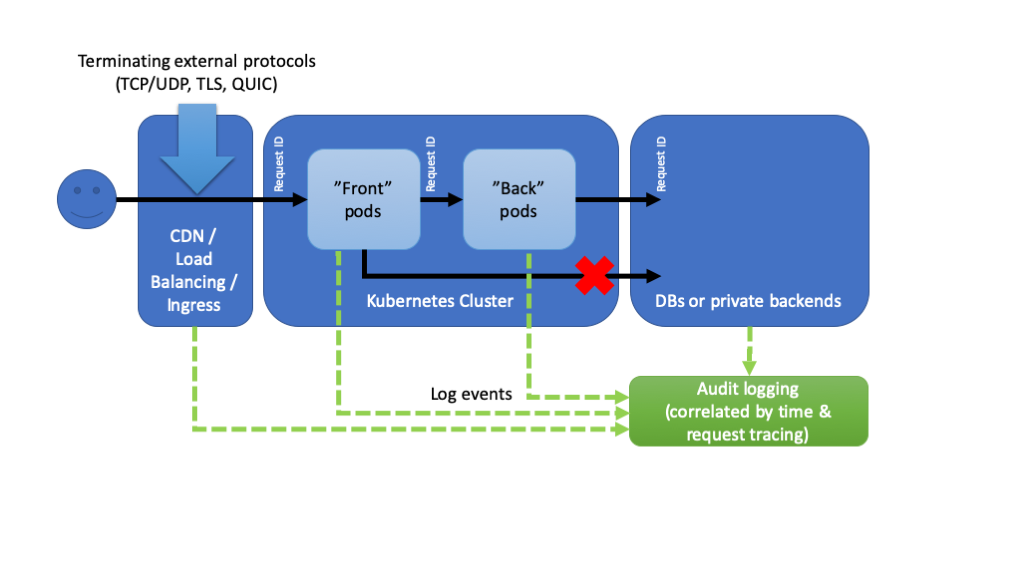
In addition, there exist tools which may help in determining whether the cluster has been securely configured. These include kube-bench for checking compliance against the CIS benchmark, and kube-hunter for discovering available resources in the cluster. The use of these tools for self-managed clusters is recommended, however, they are not a full replacement for manual review based on the CIS Benchmark. Of these, kube-hunter can also be run from within the cluster to give a view of what an attacker would see, having compromised a Kubernetes pod.

The recommended way to approach Kubernetes clusters is to use a Platform-as-a-Service type Kubernetes deployment. An example of these is AWS Elastic Kubernetes Service (EKS). However, the shared responsibility model on the selected Kubernetes PaaS would have to be understood. For example, some Kubernetes platforms require the users of the platform to upgrade the underlying nodes when necessary. The responsibility split is not necessarily a clean line on the technology stack. If the selected cloud service has a Kubernetes hardening guide, that should be followed.

Regardless of which Kubernetes solution is used, the Kubernetes deployment (or its deployment tools) must be CNCF Certified Kubernetes.

Kubernetes security inherits many of the Docker security aspects, see above. Things such as running Kubernetes pods from read-only filesystems and not mounting host directories apply.

An architectural pattern that should be applied to Kubernetes deployments is forcing a potential attacker to meet a multi-tier defensive architecture. An example of this is pictured below.



The underlying idea is to divide Kubernetes pods into separate trust domains depending on their exposed attack surface. In the picture above, the front-end pods are likely to parse and process application layer data flows from the attacker, so they have more exposed attack surface.

Databases and backend services are likely to require credentials. In this model, these credentials would only be available to the backend pods.

Network traffic towards the backend pods would be forced to traverse the front-end pods, and similarly traffic to databases or private backends would only be accepted from the back-end pods. This can be performed using a NetworkPolicy or a service mesh.

Unlike virtual machines, there is no hypervisor-governed separation between pods. If the back-end services have specific security needs, it may be useful to consider Kubernetes inter-pod anti-affinity settings. Based on these settings, Kubernetes may be instructed to schedule pods in different trust domains on different virtual machine nodes. Applications with high security requirements might be having to be split into different cloud provider accounts, which naturally splits them into different clusters.

In the example above, there are two trust domains, but applications might have several logical trust domains.

This type of target architecture is intended to frustrate an attacker, as it is more difficult to pivot towards other networked resources from the front-end pod. On the other hand, even a complete compromise of the front-end pod would not leak credentials to the back-end servers - although, obviously, making requests towards them through back-end pods would become possible. Restricting the network traffic also prevents the attacker from calling cloud service providers' metadata or Kubernetes cluster's control plane endpoints.

It should be noted that a load balancing or ingress layer does not necessarily process application layer data, so they may not have that much of an effect hardening the front pods.

If the application's original architecture is a monolith, it should not necessarily immediately be refactored into this type of target architecture. Instead, these security targets should be kept in mind when there are other changes done to the application's architecture.

**Sources:**

* kube-bench: https://github.com/aquasecurity/kube-bench
* kube-hunter: https://github.com/aquasecurity/kube-hunter
* CNCF Certified Kubernetes: https://www.cncf.io/certification/software-conformance/'
* Pod anti-affinity: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#inter-pod-affinity-and-anti-affinity
* NetworkPolicy: https://kubernetes.io/docs/concepts/services-networking/network-policies/#isolated-and-non-isolated-pods

### Terraform

The largest security risks of Terraform are related to the highly privileged parts of a deployment pipeline that need to be able to perform the infrastructure changes. When a development team manages their infrastructure as code, the infrastructure cannot be more secure than their deployment process.

Terraform saves its state in a state file. Terraform supports a shared state file, for example, using AWS S3 buckets. The access controls of the state files must be verified to protect the integrity of the state files.

## Secrets management

Secrets, such as passwords, unencrypted private keys or API keys must never be stored in plaintext in version control systems or Dockerfiles. Long-term secrets storage should be done using secrets management software. These range from password managers for individual developers to various ‘vault’ services that can be deployed as cloud services. Secrets should be also backed up.

Secrets handled manually by humans should be managed using a trustworthy password manager.

Delivering secrets to running containers is a development and delivery pipeline specific problem, and it is not easy to give a solution that would work in all cases. Secrets provisioning should preferably be done using a ‘vault’ type service, or a system provided by the orchestration tools. If these are not available, secrets can be provided through encrypted files made available on a storage location, and the decryption key can be provided to the container in an environmental variable. If using environmental variables, these should be zeroed after the value has been obtained, as in many failure modes, debugging output dumps the environment contents.

Third-party software exists for secrets management, such as HashiCorp Vault. Kubernetes and OpenShift have ‘secret’ objects. Infrastructure clouds have their own key and secrets storage systems, such as the AWS Secrets Manager. Using an orchestrator-provided secrets provisioning scheme is usually the best choice. When using a secrets management system provided by the cloud vendor, continuity planning must prepare for eventual change and secrets re-provisioning.

**Sources:**

* <https://kubernetes.io/docs/concepts/configuration/secret/>
* <https://docs.openshift.com/container-platform/3.11/dev_guide/secrets.html>
* <https://www.vaultproject.io/>
* https://aws.amazon.com/secrets-manager/

## Application Programming Interfaces (APIs)

All new web interfaces should be implemented as JSON. The JSON presentation is simple and readable, and support from software is good. The other common implementations, JSONP and XML, have distinct challenges:

* JSONP should not be used as the API responses are executed as program code. This removes the abstraction between an interface and the application using the interface. Interfaces should only provide data which is processed by the calling application.
* XML parsers have often had errors that have led to data being interpreted incorrectly. This may be a problem if different XML parsers interpret their input differently. XML also allows information to have multiple different structures that in the end are parsed in a similar way.

### API security in general

For each incoming request, the API should perform multiple checks, described below. Depending on the architecture, the API implementation may trust an intervening API gateway or proxy or perform these checks itself.

Should one of these checks fail, the request should be immediately rejected. The API should not try to handle malformed requests. If there is a problem with authentication, authorization, or integrity of data, the content of the request should not be parsed. Instead, processing should end with minimal overhead.

* Session identifiers: Is the session valid? What is the level of authorization? (See the session identifier section in this guideline.)
* Request authorisation. Ensure that the caller has the authority to call the API (for example, an API key or an authorisation based on the session).
* When using the request ID pattern (see above), does the request include it? If it does, and we perform subsequent calls to other systems, we need to include this identifier. If it doesn't, we need to create it (or if it should be present, we should drop the request).
* Integrity of the request and the data. Determine the integrity of the request. When using an API key or another shared secret between the caller and the API, integrity protection can build on that secret. Typically, authorization credentials are transferred using JSON Web Tokens (JWT) and complete API call integrity can be attained using AWS Signature v4 scheme.
* Is the incoming request well formed? Even when using JSON, the request should be validated against a schema. This protects the API against injection of arbitrary data fields. Without schema validation, there is a risk that arbitrary extra data fields may end up even in a database – a typical problem when using schema-less databases. Domain Driven Design (DDD) may be a useful approach, see *Secure by Design*, below.
* Authorisation of the returned data. In some cases, calling the API and actually getting specific data in the response are different authorisation decisions.
* Does the API need to create audit log events for this call? (Please see the section on audit logging in this guideline.)
* Does the API set all required HTTP headers in the response, such as HSTS and Content-Type? (Please see the section on HTTP headers in this guideline.)

When using JWT for authorisation, its choice of cryptographic algorithms and key lengths should follow the TLS 1.2 guideline as applicable (above).

**Sources:**

* JSON Web Tokens theme pages <https://jwt.io/>
* AWS Signature v4, which is not AWS specific <https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html>
* Secure by Design: https://www.manning.com/books/secure-by-design

### Cross-Origin Resource Sharing (CORS)

CORS is a framework where the server informs the browser that it is allowed to bypass the normal same-origin rules and call an API on another domain. The browser performs a separate initial request, called a preflight request, to determine whether this type of cross-origin access can be allowed. It is important to note that enforcement of CORS happens on the browser side. An attacker can still call an API without a browser irrespective of what its CORS set-up is. Server-side access control must always be implemented using appropriate access credentials.

The target server provides their CORS policy using the Access-Control-Allow-Origin header. Access-Control-Allow-Origin specifies the sources of JavaScript code that may call that API (that is, where the JavaScript originated that can call the API).

For example, if an organization has a web application at [app.company.com](http://app.company.com), and an API at api.cloud.example, and they want JavaScript from the [app.company.com](http://app.company.com) web page to be able to call api.cloud.example, the api.cloud.example API must be configured to return something like

Access-Control-Allow-Origin: [app.company.com](http://app.company.com)  
Access-Control-Allow-Methods: GET POST PUT DELETE

in their HTTP responses. If the API requires credentials (e.g., in cookies), it also has to explicitly grant the browser the possibility to deliver cross-origin credentials with

Access-Control-Allow-Credentials: true

If an API needs to be callable from whatever JavaScript code, irrespective from where it has been served, the header would read

Access-Control-Allow-Origin: \*

If the API is not supposed to be generally callable, existence of this header may indicate that the CORS policy is too lax.

**Sources:**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>
* <https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy>

### CSRF tokens

Cross Site Request Forgery (CSRF) is an attack where an attacker site performs a HTTP request towards another site using the user’s browser. If the browser has an active session on the target site, the attacker may perform actions on the user’s behalf.

CSRF can be mitigated by providing a cryptographically random and sufficiently long value, known as a CSRF token, with each HTTP request. The tokens are stored in the browser in such a way that the attacker site, or JavaScript that originates from the attacker site, cannot access them, and therefore cannot forge the requests.

CSRF is easily mitigated by using a proper framework that provides CSRF protection by default. In most cases, this would be preferable.

If this type of framework support is not available, a recommended way to mitigate is to use a “double submit cookie”. This method is useful because it is stateless, requiring no server-side storage, and as such it can be used when requests are routed to random servers through a load balancer. As with many other security features, naïve implementations of this may introduce vulnerabilities.

**Sources:**

* <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Double_Submit_Cookie>

### Confidential data in parameters

API calls must be formed in a way that personal data or other confidential data is not provided as a part of a URL. Parameters and path elements of a URL will easily get logged, stored in cache and browser history, be stored by proxies, or leak in Referrer headers.

All confidential data should be transferred in the body of a POST request.

The correct way could look for example like this:

POST /search/address HTTP/1.1  
  
SSN=010170-123F

The wrong way could look something like this:

GET /search/010170-123F/address HTTP/1.1

In these examples, 010170-123F is a Finnish personal identification number.