Tampereen ammattikorkeakoulu

Tietoturvan näkökulma vaatimusmäärittelyssä ja järjestelmäsuunnittelussa

Mark Eric Stenbäck

OPINNÄYTETYÖ
Marraskuu 2020

Tietojärjestelmäosaamisen koulutus YAMK

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaamisen koulutus YAMK

STENBÄCK, MARK E.:
Tietoturvan näkökulma vaatimusmäärittelyssä ja järjestelmäsuunnittelussa

Opinnäytetyö 68 sivua, joista liitteitä 4 sivua
Marraskuu 2020

Tietoturvan merkitystä ohjelmisto- ja järjestelmäsuunnittelulle erityisesti sekä tietoyhteiskunnalle yleisesti ei voi korostaa liiaksi. Yksityisen ja luottamuksellisen tiedon suojaaminen on paitsi toiminnan edellytys yrityksille ja muille organisaatioille, myös kansallisten lakien ja kansainvälisten sopimusten velvoittama käytännön vaatimus.

Tämän opinnäytetyön keskeisenä tutkimuskysymyksenä oli, miten tietoturvan näkökulma tulisi huomioida jo hankkeen vaatimusmäärittely- ja suunnitteluvaiheissa, jotta vältyttäisiin ongelmilta järjestelmän elinkaaren myöhemmissä vaiheissa. Ensisijaisena tutkimusmenetelmänä oli aineistotutkimus, jossa pääasiallisesti tarkastelin Valtionhallinnon tietoturvallisuuden johtoryhmän julkaiseman Sovelluskehityksen tietoturvaohjeen lukua Sovelluskehityksen vaiheet ja tietoturvatasojen vaatimukset. Täydentävinä aineistoina tarkastelin soveltuvin osin mm. Microsoftin Tietoturvakehityksen elinkaaren mallia, OWASP-tietoturvahankkeita sekä muita asiayhteyteen sopivia menetelmiä.

Loppupäätelmä on, että jokaisessa organisaatiossa ja kehityshankkeessa on ymmärrettävä, että tietoturvassa on kyse ohjelmisto- ja järjestelmäsuunnittelun kannalta keskeisestä, elintärkeästä osa-alueesta eikä suinkaan valinnaisesta lisäominaisuudesta. Puutteet järjestelmien tietoturvassa asettavat paitsi yksittäisten ihmisten yksityisyydensuojan uhanalaiseksi, niin ne voivat johtaa myös huomattaviin liiketoiminnallisiin tappioihin sekä vaarantaa jopa kansallisen turvallisuuden.

Asiasanat: tietoturva, vaatimusmäärittelyt, ohjelmistosuunnittelu, järjestelmäsuunnittelu, IT-arkkitehtuuri

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master's Degree Programme in Information System Competence

STENBÄCK, MARK E.:
Information Security Perspective in Requirements Specification and Systems Design

Master's thesis 68 pages, appendices 4 pages
November 2020

The significance of information security for software and systems design, as well as for the information society in general, cannot be overstated. The protection of private and confidential information is not only vital for corporations and other organisations, but it is also a mandatory requirement based on national laws and international treaties.

The research question of this thesis is how the information security perspective should be acknowledged during requirements specification and design phases, in order to avoid problems in the later stages of a system's life cycle. This thesis strives to provide an answer by means of material research. The primary source is the information security guideline, published by the Finland's Ministry of Finance. Additional sources include Microsoft's Secure Development Lifecycle, OWASP Application Security Verification Standard, OWASP Web Security Testing Guide and other contextually relevant literature.

The conclusion of the thesis is that all organisations and development projects must understand that information security is a vital, integral part of software and systems design. It is not an optional, additional feature. Failures in information security compromise not only individual privacy, but may also result in significant financial liabilities, and even jeopardize national security.

Key words: information security, requirements specifications, software design, systems design, IT architecture

SISÄLLYS

1 JOHDANTO 7
1.1 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013) 9
1.2 OWASP Application Security Verification Standard (ASVS) 11
1.3 OWASP Web Security Testing Guide (WSTG) 12
1.4 Tietoturvakehityksen elinkaari (SDL) 15
2 ESITUTKIMUS 16
2.1 Ohjelmiston tietoturvaohje (VAHTI 1/2013) 17
2.1.1 Perustaso: Tarkoitus ja kriittisyys (ESI-001) 17
2.1.2 Perustaso: Liiketoiminnan vaikutusanalyysi (ESI-002) 18
2.2 Tietoturvakehityksen elinkaari (Microsoft SDL) 19
2.2.1 Käytäntö #1 – Tarjoa koulutusta 19
2.2.2 Käytäntö #2 – Määrittele tietoturvavaatimukset 20
2.2.3 Käytäntö #3 – Määrittele mittarit ja noudattamisraportointi 21
3 VAATIMUSMÄÄRITTELY 22
3.1 RFC 2119 – Vaatimusmäärittelyn avainsanat 23
3.2 Toiminnallinen ja ei-toiminnallinen määrittely 23
3.2.1 Toiminnallinen määrittely 24
3.2.2 Ei-toiminnallinen määrittely 25
3.3 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013) 26
3.3.1 Perustaso: Tietoturvaratkaisuiden dokumentointi (VTM-001) 27
3.3.2 Perustaso: Lainsäädännölliset vaatimukset (VTM-002) 27
3.3.3 Perustaso: Tietoturva-analyysi (VTM-003) 28
3.3.4 Perustaso: Sovelluksen tietoturvataso (VTM-004) 29
3.3.5 Perustaso: Sovelluksen riskianalyysi (VTM-005) 30
3.3.6 Perustaso: Sovelluksen tietoturvavaatimukset (VTM-006) 32
3.3.7 Perustaso: Lait, määräykset ja ohjeistukset (VTM-007) 32
3.3.8 Korotettu taso: Uhkamallinnus (VTM-008) 33
3.3.9 Korotettu taso: Arkkitehtuurilinjaus (VTM-009) 33
3.3.10 Korotettu taso: Uhkamallinnuksen tarkennus (VTM-010) 34
3.3.11 Korkea taso: Komponenttien uhka-arvio (VTM-011) 34
3.4 Tietoturvakehityksen elinkaari (Microsoft SD) 36
3.4.1 Käytäntö #4 – Suorita uhkamallinnus 36
3.4.2 Käytäntö #5 – Määrittele suunnitteluvaatimukset 36
3.4.3 Käytäntö #6 – Määrittele kryptografiset standardit 37
3.5 OWASP Application Security Verification Standard 38

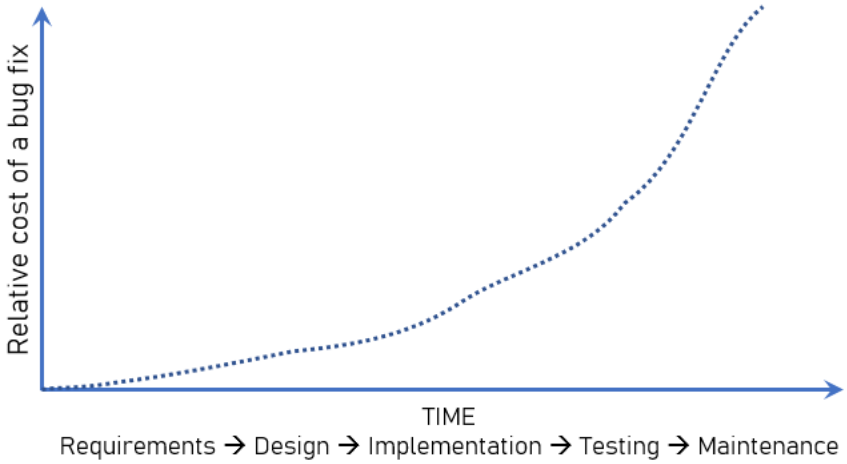
3.6 OWASP Web Security Testing Guide 39
3.6.1 Ohjelmiston elinkaaren määrittely..... 39
3.6.2 Käytäntöjen ja standardien katselmointi 40
3.6.3 Mittaustapojen kehitys ja tulosten jäljitettävyys 40
4 SUUNNITTELU..... 42
4.1 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013) 42
4.1.1 Perustaso: Yleiset standardit (SNT-001) 43
4.1.2 Perustaso: Tietoturvapäivitysten suunnittelu (SNT-002)..... 44
4.1.3 Perustaso: Suositellut ohjelmistot (SNT-003) 45
4.1.4 Perustaso: Ulkoiset rajapinnat (SNT-004) 47
4.1.5 Perustaso: Tietoturvalliset suunnittelumallit (SNT-005)..... 48
4.1.6 Perustaso: Hyökkäyspinta-ala (SNT-006) 49
4.1.7 Perustaso: Arkkitehtuurin tietoturvavaatimukset (SNT-007)50
4.1.8 Perustaso: Tietoturvamekanismien kattavuus (SNT-008) .. 51
4.1.9 Perustaso: Tunnistautumismenetelmä (SNT-009)..... 51
4.1.10 Perustaso: Salasanavaatimusten konfigurointi (SNT-010). 52
4.1.11 Perustaso: Käyttöoikeustasot (SNT-011)..... 55
4.1.12 Perustaso: Luottamusrajat (SNT-012) 56
4.1.13 Perustaso: Salausratkaisut (SNT-013)..... 56
4.1.14 Perustaso: Tuki- ja ylläpitoyhteydet (SNT-014)..... 57
4.1.15 Korotettu taso: Uhkamallinnuksen syventäminen (SNT-015)..... 58
4.1.16 Korotettu taso: Monitasoarkkitehtuurit (SNT-016)..... 58
4.1.17 Korkea taso: Vahva tunnistautuminen (SNT-017)..... 59
5 POHDINTA 60
LÄHTEET..... 63
LIITTEET 65
Liite 1. Business Model Canvas..... 65
Liite 2. Tyypillisen käyttötapauksen rakenne..... 66
Liite 3. RFC 2119 – RFC Key Words 67

LYHENTEET JA TERMIT

ASVS	Application Security Verification Standard
CC BY-SA 4.0	Creative Commons Attribution-ShareAlike 4.0 -lisenssi
CVSS	Common Vulnerability Scoring System
DREAD	Damage, Reproducibility, Exploitability, Affected users, Discoverability
FFIEC	Federal Financial Institutions Examination Council
FMEA	Failure Modes and Effects Analysis
ICT	Information Communications Technology
OWASP	The Open Web Application Security Project
RFC	Request for Comments
SDL	Secure Development Lifecycle
SDLC	Software Development Life Cycle
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
TAMK	Tampereen ammattikorkeakoulu
VAHTI	Valtionhallinnon tietoturvallisuuden johtoryhmä
WSTG	Web Security Testing Guide
UML	Unified Modelling Language

1 JOHDANTO

On olemassa väite[1], jonka mukaan ohjelmistohankkeen ohjelmointi- ja suunnitteluvirheiden suhteelliset kustannukset kasvavat sitä suuremmaksi, mitä myöhäisemmässä vaiheessa ne havaitaan ja korjataan. Kustannusten nousua kuvataan usein summittaisella graafilla (kuvio 1):



KUVIO 1. ”Pressman Ratio” havainnekuva

Väitteen taustalla on ns. ”Pressman Ratio”, mutta koska ensikäden tietoa kyseisestä tutkimuksesta ei ole saatavilla, asiaan viitataan tässä yhteydessä ainoastaan anekdootillisesti. Väite kustannusten noususta järjestelmän elinkaaren edetessä vaikuttaa uskottavalta periaatteellisella tasolla: järjestelmän tulevaan toimintaan on halvempaa vaikuttaa määrittely- ja suunnitteluvaiheessa kuin muuttaa tuotantokäytössä olevan järjestelmän toimintaa.

Nykypäivänä on valitettavan yleistä, että tuotantokäytössä olevissa järjestelmistä paljastuu kyseenalaisia tietoturvaratkaisuja, sikäli kuin tietoturvan näkökulmaa ylipäätään huomioitu asianmukaisella tavalla. Niinpä tämän opinnäytetyön keskeisenä tutkimuskysymyksenä on, miten tietoturvan näkökulma tulisi

[1] Ns. ”Pressman Ratio” esittää, että jos ohjelmistovirheen korjauksen kustannus on yksi yksikköä suunnitteluvaiheessa, niin vastaava kustannus on toteutusvaiheessa 6,5 kertainen, testausvaiheessa 15 kertainen ja ylläpitovaiheessa jo satakertainen. Lähteeksi mainitaan usein IBM Systems Science Institute. Näihin lukuihin tulisi suhtautua kyseenalaistaen sillä alkuperäistä tutkimusta ei voi löytää eikä IBM:llä ole koskaan ollut tämän nimistä tutkimuskeskusta. On mahdollista, että lähteenä on ollut IBM:n vuosina 1967-1981 toiminut sisäinen koulutusohjelma, jonka materiaalit pohjautuvat aikaisempaan tuntemattomaan lähteeseen. ”Pressman Ratio” vaikuttaa periaatteellisesti pätevältä, mutta se ei perustu tunnettuun tutkimukseen. Vaikka kertoimet perustuisivatkin todelliseen dataan, se ei kuvaisi moderneja tietotekniikan alan menetelmiä ja kustannusrakenteita.
https://gist.github.com/Morendil/ebfa32d10528af04e2ccb8995e3cb4a7

huomioida jo hankkeen vaatimusmäärittely- ja suunnitteluvaiheissa, jotta vältyttäisiin ongelmilta järjestelmän elinkaaren myöhemmissä vaiheissa.

Ensisijaisena tutkimusmenetelmänä on aineistotutkimus.

> ”A second precondition for solving our problems is a realization that all of them are interlocked, with the result that they cannot be solved piecemeal. They must be approached as a set of problems, each of which interacts with the others. Sometimes it is easier to solve ten interlocking problems simultaneously than to solve one by itself.” (Brown, H. 1972)

Tämän tutkintotyön pääasiallisena aineistona on Valtionhallinnon tietoturvallisuuden johtoryhmän sekä Valtionvarainministeriön julkaisema Sovelluskehityksen tietoturvaohje (VAHTI 1/2013).

> Sovelluskehityksen tietoturvaohje on tarkoitettu esimiehille, hankkeiden vetäjille, sovelluskehittäjille sekä tietoturvallisuudesta vastaaville. (VAHTI 1/2013, 5)

Erityisesti tässä työssä tarkastellaan Sovelluskehityksen tietoturvaohjeen lukua 3.3 eli *Sovelluskehityksen vaiheet ja tietoturvatasojen vaatimukset*. Kyseessä on vaatimusmäärittely, joka huomioi tietoturvakysymykset valtionhallinnon ohjelmistohankkeiden elinkaaren kaikissa vaiheissa. Yleisellä tasolla tarkasteltuna ohjeistuksen voi nähdä ns. *best practices* -kokoelmana, joka on soveltuvin osin hyvä hyödyntää myös muiden organisaatioiden ohjelmistohankkeissa.

Sovelluskehityksen tietoturvaohjeen toimiessa taustana, tässä tutkintotyössä tarkastellaan lisäksi soveltuvin osin seuraavia tietoturvaohjeistuksia:

- OWASP Application Security Verification Standard (ASVS) 4.0.2
- OWASP Web Security Testing Guide (WSTG) 4.1

Soveltuvin osin tarkastellaan myös menetelmiä, kuten esimerkiksi *Failure Modes and Effects Analysis* (FMEA) ja *Common Vulnerability Scoring System* (CVSS), joiden käytöstä on hyötyä vaatimusmäärittelyn ja suunnittelun eri vaiheissa.

Lisäksi tarkastellaan Microsoftin kehittämää Tietoturvakehityksen elinkaaren[2] mallia (Microsoft SDL). Yleisesti ottaen kyse Microsoftin suosittelemasta tavasta kehittää organisaation tietoturvakulttuuria omaksumalla kaksitoista tietoturvaa kehittävää käytäntöä.

1.1 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013)

Sovelluskehityksen tietoturvaohjeen tavoitteena on ”auttaa valtionhallinnon organisaatioita toteuttamaan sovelluskehityksessä vaaditut tietoturvatehtävät heti kehityksen alkuvaiheesta lähtien, ja auttaa myös sovellushankintojen vaatimusmäärittelyjen tekemisessä.” (VAHTI 1/2013, 11) Vaikka ohje on tarkoitettu valtionhallinnon organisaatioiden hankkeissa noudatettavaksi, on sitä hyödyllistä soveltaa myös muissa hankkeissa.

> Nyt julkaistava Sovelluskehityksen tietoturvaohje on tarkoitettu esimiehille, hankkeiden vetäjille, sovelluskehittäjille sekä tietoturvallisuudesta vastaaville. Ohjeen tavoitteena on opastaa sovelluskehittäjiä ottamaan tietoturvavaatimukset huomioon kaikissa vaiheissa. Tietoturvallisuuden ottaminen huomioon sovelluskehityksen alusta lähtien on olennaista niin turvallisuuden toteuttamisen kuin kustannustehokkuudenkin näkökulmasta. (VAHTI 1/2013, 5)

Tietoturvaa ei voi rakentaa jälkikäteen vaan se on rakennettava sovelluksen sisään sen elinkaaren eri vaiheissa (VAHTI 1/2013, 44). Todennäköisesti jokainen ohjelmistokehityshankkeissa mukana ollut tietää, että käytännössä näin ei useinkaan ole vaan kireisiin aikatauluihin, pieniin budjetteihin ja rajalliseen henkilöstöön vedoten tehdään usein ratkaisuja, jotka tietoturvan näkökulmasta ovat vähintäänkin kyseenalaisia. Ratkaisu tähän ongelmaan ei ole aikataulujen väljentäminen eikä budjettien kasvattaminen vaan ymmärryksen laajentaminen tietoturvakysymysten suhteen, jotta organisaatiossa osattaisiin tehdä ratkaisuja, jotka kohdentavat rajalliset resurssit tehokkaimmalla mahdollisella tavalla.

Microsoft SDL:n ensimmäinen asia on tarjota tietoturvakoulutusta organisaation kaikille jäsenille. Ei siksi, että kaikista pitäisi tulla tietoturvan asiantuntijoita, mutta jokaisella on kuitenkin oltava ymmärrys tietoturvan perusteista sekä tiedettävä,

[2] Security Development Lifecycle

miten ohjelmistoja ja palveluita voidaan toteuttaa tietoturvallisesti huomioiden samalla myös liiketoiminnan tarpeet.[3] (Microsoft SDL, n.d.)

Sovelluskehityksen tietoturvaohjeen vaatimukset jakautuvat kolmelle tasolle ICT-varautumisen vaatimukset (VAHTI 2/2012) ohjeistuksen mukaisesti (kuvio 2).

KUVIO 2. Vaatimusten jakautuminen tasoittain (VAHTI 2/2012, 21)

Perustaso mahdollistaa turvallisesti organisaation normaalin, voimakkaasti verkostoituneen toiminnan. (VAHTI 2/2012, 21) Perustaso on rinnastettavissa OWASP ASVS 1. ja 2. tasoon ja on suositeltavaa, että kaikki järjestelmät, jotka

[3] Practice #1 – Provide Training
"Security is everyone's job. Developers, service engineers, and program and product managers must understand security basics and know how to build security into software and services to make products more secure while still addressing business needs and delivering user value.

"Effective training will complement and re-enforce security policies, SDL practices, standards, and requirements of software security, and be guided by insights derived through data or newly available technical capabilities."

tallentavat ja käsittelevät käyttäjätietoja sekä muuta luottamuksellista informaatiota, määrittelee vaatimukset vähintään perustason mukaisella tasolla. Perustasolle sijoitetaan palvelut ja järjestelmät, joiden hetkellinen lamautuminen ei keskeytä organisaation ydintoimintoja. (VAHTI 2/2012, 21)

Sekä korotettu taso että korkein taso ovat rinnastettavissa OWASP ASVS 3. tasoon. Korotettu taso on tarkoitettu organisaation kriittisille toiminnoille. Vain osa organisaation toiminnasta, palveluista ja järjestelmistä on tarkoituksenmukaista toteuttaa tällä tasolla. (VAHTI 2/2012, 22). Korkea taso täyttää yhteiskunnan turvallisuusstrategian uhkamallien mukaisiin laajoihin häiriötilanteisiin ja poikkeusoloihin varautumisen tarpeet erityisturvallisuutta vaativissa toiminnoissa. (VAHTI 2/2012, 22)

1.2 OWASP Application Security Verification Standard (ASVS)

OWASP[4] on avoin ohjelmistojen tietoturvaprojekti, joka pyrkii parantamaan erityisesti verkkopalveluiden tietoturvan laatua. Kyseessä on avoimen lähdekoodin hanke, johon osallistuminen on avointa ja joka on vapaasti kaikkien hyödynnettävissä *Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0)* lisenssin mukaisesti. OWASP ylläpitää useita aliprojekteja, joista ASVS on yksi päähankkeista.

OWASP ASVS tavoitteena on määritellä tietoturvavaatimusten viitekehys sekä hallintamekanismit toiminnallisille ja ei-toiminnallisille tietoturvakäytänteille moderneja verkkopalveluita ja -sovelluksia suunniteltaessa ja toteutettaessa (OWASP ASVS, 2020, 8)[5] Hankkeella on kaksi päätavoitetta: auttaa organisaatiota kehittämään ja ylläpitämään tietoturvallisia sovelluksia, sekä edistää tietoturvapalveluita ja tietoturvatyökaluja myyvien tahojen sekä kuluttajien vaatimusten ja tarjontojen yhdenmukaistamista. (OWASP ASVS, 2020, 8)

OWASP ASVS määrittelee kolme tasoa, joiden puitteissa ohjelmistojen tietoturvaa voidaan tarkastella.

---

[4] The Open Web Application Security Project

[5] “The ASVS is a community-driven effort to establish a framework of security requirements and controls that focus on defining the functional and non-functional security controls required when designing, developing and testing modern web applications and web services.”

Taso 1 määrittelee tietoturvan minimivaatimukset, jotka kaikkien ohjelmistojen tulisi vähintäänkin täyttää. Sovellus saavuttaa tason 1, jos se pystyy tyydyttävästi puolustautumaan helposti löydettäviä tietoturvauhkia vastaan, mukaan lukien OWASP Top 10 ja muut vastaavat tarkistuslistat (OWASP ASVS, 2020, 10)[6]

Taso 2 määrittelee tietoturvan perusvaatimukset, jotka useimpien ohjelmistojen tulisi lähtökohtaisesti täyttää. Sovellus saavuttaa tason 2, jos se pystyy tyydyttävästi puolustautumaan useimpia ajantasaisia tietoturvauhkia vastaan (OWASP ASVS, 2020, 10)[7] Sovelluksen tulisi saavuttaa tämä taso, mikäli se sisältää ja käsittelee luottamuksellista tietoa ja/tai tarjoaa liiketoiminnallisesti merkittäviä palveluita.

Taso 3 määrittelee tietoturvan vaatimukset ohjelmistoille, joilta vaaditaan ehdotonta luotettavuutta järjestelmän toiminnan ja tiedon käsittelyn osalta. Nämä vaatimukset ovat tyypillisiä sotilas-, väestönsuojelu- ja kriittisen infrastruktuurin järjestelmille. (OWASP ASVS, 2020, 11)[8]

1.3 OWASP Web Security Testing Guide (WSTG)

Projektin tavoitteena on auttaa ihmisiä ymmärtämään *mitä, miksi, milloin, missä* ja *miten* verkko-ohjelmistoja testataan. (OWASP WSTG, 2020, 9)[9]. Lopputuloksena syntynyt testauksen puitekehys[10] on kokonaisvaltainen ja kattaa ohjelmiston elinkaaren lähes kokonaisuudessaan. Kyseessä on avoimen lähdekoodin hanke, johon osallistuminen on avointa ja joka on vapaasti kaikkien hyödynnettävissä *Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0)* lisenssin mukaisesti. OWASP ylläpitää useita aliprojekteja, joista WSTG on yksi päähankkeista.

---

[6] ”An application achieves ASVS Level 1 if it adequately defends against application security vulnerabilities that are easy to discover and included in the OWASP Top 10 and other similar checklists.”

[7] ”An application achieves ASVS Level 2 (or Standard) if it adequately defends against most of the risks associated with software today.”

[8] ”This level is typically reserved for application that require significant levels of security verification, such as those that may be found within areas of military, health and safety, critical infrastructure, etc.”

[9] ”The aim of the project is to help people understand the *what, why, when, where,* and *how* of testing web applications.”

[10] The Web Security Testing Framework

OWASP WSTG perustana on ohjelmistokehityksen elinkaari eli SDLC[11] (kuvio 3), jota ei pidä sekoittaa Microsoft SDL:ään, mutta joka on melko suoraan rinnastettavissa VAHTI 1/2013 kuvaamaan ohjelmiston elinkaareen.

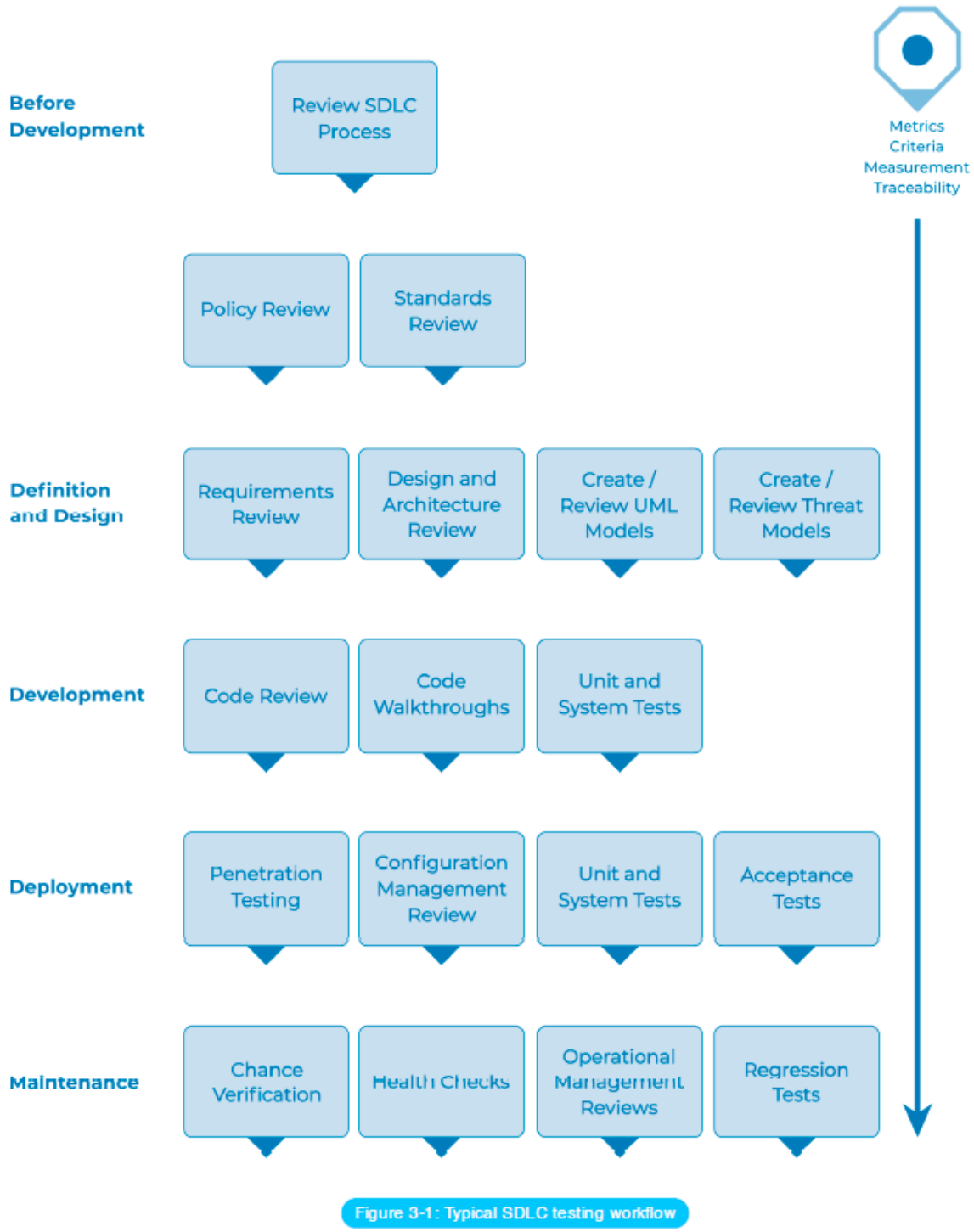
KUVIO 3. Ohjelmistokehityksen elinkaari SDLC mukaisesti (SDLC, owasp.org)

OWASP WSTG:n määrittelemä puitekehys jakaa kehitettävän ohjelmiston elinkaaren viiteen osa-alueeseen sekä niissä huomioitaviin tehtäviin ja käytäntöihin, joilla on joko suora tai välillinen vaikutus kehitettävän ohjelmiston tietoturvaan.

- Vaihe 1: Ennen kehityksen aloittamista
- Vaihe 2: Määrittelyn ja suunnittelun aikana
- Vaihe 3: Kehityksen aikana
- Vaihe 4: Tuotantoon viennin aikana
- Vaihe 5: Ylläpidon aikana

Tässä tutkintotyössä keskitytään näistä kahteen ensimmäiseen vaiheeseen (kuvio 4).

[11] Software/Systems Development LifeCycle, SDLC



KUVIO 4. Ohjelmiston elinkaari OWASP WSTG mukaisesti (OWASP WSTG, 2020, 38)

1.4 Tietoturvakehityksen elinkaari (SDL)

Tietoturvakehityksen elinkaari määrittelee kaksitoista käytäntöjä, jotka tukevat tietoturvan varmistamista sekä säädösten noudattamista (Microsoft SDL, n.d.)[12]

- Käytäntö #1 – Tarjoa koulutusta[13]
- Käytäntö #2 – Määrittele tietoturvavaatimukset[14]
- Käytäntö #3 – Määrittele mittarit sekä noudattamisraportointi[15]
- Käytäntö #4 – Suorita uhkamallinnus[16]
- Käytäntö #5 – Määrittele suunnittelun vaatimukset[17]
- Käytäntö #6 – Määrittele ja käytä kryptografisia standardeja[18]
- Käytäntö #7 – Hallitse kolmannen osapuolen komponenttien käyttöön liittyviä riskejä[19]
- Käytäntö #8 – Käytä hyväksyttyjä työkaluja[20]
- Käytäntö #9 – Suorita staattisen analyysin tietoturvatestausta (SAST)[21]
- Käytäntö #10 – Suorita dynaamisen analyysin tietoturvatestausta (DAST)[22]
- Käytäntö #11 – Suorita tunkeutuvaa testausta[23]
- Käytäntö #12 – Ota käyttöön yleinen käytäntö tietoturvapoikkeamien käsittelyyn[24]

SDL on tarkoitettu sovitettavaksi osaksi organisaation olemassa olevia ohjelmistokehityksen ja tietoturvan käytäntöjä. Kunkin käytännön omaksumiseen on erilaisia lähestymistapoja, joten organisaatiokohtaisen ratkaisun räätälöintiin on hyvät edellytykset.

---

[12] ”The Security Development Lifecycle (SDL) consists of a set of practices that support security assurance and compliance requirements.”
[13] ”Practice #1 – Provide training”
[14] ”Practice #2 – Define security requirements”
[15] ”Practice #3 – Define metrics and compliance reporting”
[16] ”Practice #4 – Perform threat modeling”
[17] ”Practice #5 – Establish design requirements”
[18] ”Practice #6 – Define and use cryptography standards”
[19] ”Practice #7 – Manage the security risk of using third-party components”
[20] ”Practice #8 – Use approved tools”
[21] ”Practice #9 – Perform Static Analysis Security Testing (SAST)”
[22] ”Practice #10 – Perform Dynamic Analysis Security Testing (DAST)”
[23] ”Practice #11 – Perform penetration testing”
[24] ”Practice #12 – Establish a standard incident response process”

2 ESITUTKIMUS

Tyypillisesti ohjelmistokehityshankkeiden käynnistävänä tekijänä on organisaation tunnistama tarve tai mahdollisuus. Tarpeista puhuttaessa kyse voi olla uuden kyvykkyyden kehittämisestä, olemassa olevan toiminnan tehostamisesta tai vaikkapa sopeutumisesta muuttuneeseen lainsäädäntöön ja asetuksiin. Mahdollisuudet puolestaan voivat liittyä monin eri tavoin liiketoiminnan kehittämiseen.

Esitutkimuksen tekemiseen on useita menetelmiä alkaen markkina- ja asiakastutkimuksesta aina kokonaisarkkitehtuurityön puuteanalyysin (engl. gap analysis) paljastamiin kehitystarpeisiin. Esimerkiksi Alexander Osterwalderin kehittämä *Business Model Canvas* (BMC) menetelmä tarkastelee korkealla tasolla liiketoiminnan kannalta yhdeksää keskeisintä osa-aluetta.

1. Avainkumppanit
2. Avaintoiminnot
3. Asiakassuhteet
4. Asiakasryhmä
5. Avainresurssit
6. Kanavat
7. Kustannusrakenne
8. Tulovirrat

Katso liite 1: Business Model Canvas

2.1 Ohjelmiston tietoturvaohje (VAHTI 1/2013)

> Esitutkimusvaihe keskittyy sovelluksen analysointiin liiketoiminnan näkökulmasta. Tässä vaiheessa otetaan huomioon laeista ja asetuksista tulevat vaatimukset, sopimukset ja sidosryhmien vaatimukset sekä organisaation periaatteet ja ohjeet. Näistä kaikista liiketoimintaa ohjaavista seikoista johdetaan sovelluksen tietoturvavaatimuksia, kuten esimerkiksi käsiteltävän tiedon vaatima suojaustaso. (VAHTI 1/2013, 45)

Jo esitutkimusvaiheessa on muodostettava ymmärrys siitä millaista tietoa järjestelmän puitteissa tullaan käsittelemään. Tietoturvanäkökulmasta olennaista on arvioida käsiteltävän tiedon luottamuksellisuus sekä lainsäädännön vaikutus mm. kyseisen tiedon varastointiin, käsittelyyn ja jakeluun. Tämä arvio vaikuttaa suoraan myöhemmin tehtävään vaatimusmäärittelyyn, järjestelmän suunnitteluun, toteutukseen ja testaukseen sekä tuotantoympäristön infrastruktuuriin ja ylläpidon käytäntöihin. Käsiteltävän tiedon luottamuksellisuus ja sen turvaamiseen tarvittavat ratkaisut heijastuvat hankkeen kehityskustannuksiin, minkä lisäksi tulisi huomioida myös järjestelmän tuotantokäyttöön liittyvät riskit ja millaiset taloudelliset vastuut liittyvät riskien realisoitumiseen. Kaikki tämä on olennaisessa roolissa, kun arvioidaan josko hankkeessa on perusteltua siirtyä eteenpäin.

2.1.1 Perustaso: Tarkoitus ja kriittisyys (ESI-001)

> Toteutettavan sovelluksen liiketoimintatarkoitus sekä sovelluksen kriittisyys sitä käyttävän organisaation liiketoiminnalle tulee määritellä. Määrittely on dokumentoitava ja sovellus luokiteltava sen avulla. Lisäksi sovelluksen käsittelemien tietojen luottamuksellisuus tulee määritellä. […] Määritykset toimivat koko projektin ajan sovelluksen tietoturvamekanismien valintaa, suunnittelua ja toteutusta ohjaavina tekijöinä. (VAHTI 1/2013, 45)

Ohjelmistokehityshanke ei synny tyhjiössä vaan saa usein alkunsa muun prosessin tuotoksena. Esimerkiksi kokonaisarkkitehtuurityön (esim. JHS 179 tai TOGAF) tuotoksena organisaatio on tunnistanut toimintansa kannalta keskeisiä liiketoiminnan vaatimuksia, näiden perusteella määritellyt sekä nyky- että tavoitetilan ja puuteanalyysin (engl. gap analysis) kautta tunnistanut kehitystarpeita, jotka voivat johtaa yhteen tai useampaankin kehityshankkeeseen. Tämän työn pohjalla olevat liiketoiminnalliset vaatimukset ovat merkittävässä roolissa, kun esitutkimuksesta siirrytään vaatimusmäärittelyyn.

Riippumatta siitä miten kehityshankkeen tarve on tunnistettu ja esitutkimustyö on päätetty aloittaa, on ensiarvoisen tärkeää ymmärtää mistä hankkeessa on kyse: mitä sovelluksen tai järjestelmän on tarkoitus tehdä, mitä informaatiota sen toiminta edellyttää sekä miten tärkeää tuon informaation suojaaminen tulee olemaan.

2.1.2 Perustaso: Liiketoiminnan vaikutusanalyysi (ESI-002)

> Toteutettavalle sovellukselle tulee tehdä liiketoiminnan vaikutusanalyysi, jossa selvitetään mitä vaikutusta erilaisten uhkakuvien toteutumisella on organisaation toiminnalle. (VAHTI 1/2013, 45)

Liiketoiminnan vaikutusanalyysi (engl. Business Impact Analysis, BIA) on osa organisaation riskienhallintaa. Kyseessä on prosessi, jossa pyritään tunnistamaan miten kriisit / häiriötilanteet vaikuttavat organisaation toimintakykyyn. (FFIEC Business Continuity Management, 9) Liiketoiminnan vaikutusanalyysin (kuvio 5) suorittaminen ei sisälly tähän tutkintotyöhön, mutta yleisellä tasolla siihen sisältyy kolme pääaluetta:

1. Liiketoiminnan kannalta kriittisten toimintojen tunnistaminen
2. Organisaation toimintojen vuorovaikutussuhteiden analyysi (miten häiriö yhdessä toiminnossa heijastuisi muihin toimintoihin)
3. Toimintoon kohdistuvan häiriön konkreettisten vaikutusten arviointi sekä suunnitelma häiriöiden vaikutusten rajaamiseksi ja niistä toipumiseksi.

KUVIO 5. Havainnekuva liiketoiminnan vaikutusanalyysin rakenteesta (FFIEC Business Continuity Management, 11)

2.2 Tietoturvakehityksen elinkaari (Microsoft SDL)

Suotavaa olisi, että organisaation tietoturvakulttuuria olisi ryhdytty kehittämään jo ennen kehityshankkeen alkua, mutta esitutkimusvaiheessa viimeistään olisi hyvä tähän ryhtyä. Ohjelmistojen ja järjestelmien tietoturvassa kyse ei ole pelkästään teknisistä ratkaisuista vaan suuremmassa roolissa ovat organisaation omaksumat asenteet, arvot ja päivittäisen työn toimenpiteet.

Tietoturva on prosessi, jonka avulla taloudellinen instituutio suojelee tiedon tuottamista, keräämistä, varastointia, käyttöä ja tiedonsiirtoa sekä arkaluonteisten tietojen hävittämistä, mukaan lukien tiedon tallentamiseen ja lähettämiseen käytettävien laitteistojen ja infrastruktuurin suojeleminen.[25] (FFIEC Information Security, 1)

Tietoturva-asioita saatetaan pitää teknisten asiantuntijoiden ”juttuna”, mutta jotta tietoturva voisi käytännössä toteutua niin organisaation kuin hankkeidenkin tasolla, on koko organisaation panostettava tietoturvakulttuurin kehittämiseen ja omaksumiseen. Organisaation johtoryhmän ja hallinnon on osaltaan ymmärrettävä ja tuettava tätä työtä järjestäen henkilöstölle riittävät edellytykset kehittää, omaksua ja ylläpitää tietoturvallisia käytäntöjä.[26] (FFIEC Information Security, 3) Juuri tähän asiaan liittyy Microsoft SDL:n kolme ensimmäistä käytäntöä:

2.2.1 Käytäntö #1 – Tarjoa koulutusta

Ensimmäinen ja tärkein käytäntö tietoturvakulttuurin kehittämiseksi ja omaksumiseksi on koulutuksen tarjoaminen koko henkilöstölle. Koulutuksen tarjoaminen itsessään ei kuitenkaan ole tarpeeksi, vaan sen tulee olla velvoittavaa: toteutuakseen tietoturva ei voi olla vapaaehtoista vaan organisaation jokaisen jäsenen aina hallituksen jäsenistä harjoittelijoin tulee ymmärtää mistä tietoturvassa on kyse ja mitä sen ylläpitäminen edellyttää. Tämä

---

[25] ”Information security is the process by which a financial institution protects the creation, collection, storage, use, transmission, and disposal of sensitive information, including the protection of hardware and infrastructure used to store and transmit such information.”

[26] ”The board and management should understand and support information security and provide appropriate resources for developing, implementing, and maintaining the information security program.”

puolestaan edellyttää systemaattista lähestymistä tietoturvan koulutukseen sekä sitä, että työntekijöiden osallistumista ja ymmärryksen omaksumista seurataan.

Vaikka tietoturva on osa jokaisen työtä, on tärkeää pitää mielessä, että jokaisen ei tarvitse tietoturvan asiantuntija, saati kouluttautua penetraatiotestaajaksi. Jokaisen tulisi kuitenkin ymmärtää hyökkääjän näkökulma ja tavoitteet.[27] (Microsoft SDL, n.d.)

2.2.2 Käytäntö #2 – Määrittele tietoturvavaatimukset

Tietoturvavaatimukset voivat olla paitsi hankekohtaisia myös organisaatiotason yleisiä vaatimuksia, jotka heijastuvat kaikkiin hankkeisiin. Optimaalinen ajankohta tietoturvavaatimusten tunnistamiselle, määrittelylle ja omaksumiselle on heti hankkeen alkuvaiheissa eli esitutkimuksen, vaatimusmäärittelyn sekä suunnittelun aikana. Tietoturvavaatimusten määrittelyssä tulisi huomioida mm. velvoittava lainsäädäntö, liiketoiminta-alan yleiset standardit, organisaation sisäiset määräykset, tietoturvallisen ohjelmistokehityksen käytännöt, tunnetut uhat sekä aikaisemmat tietoturvaan liittyvät kokemukset.[28] (Microsoft SDL, n.d.)

Vaatimustenhallinnan merkitystä paitsi yksittäisen hankkeen, mutta myös organisaation pitkän tähtäimen tavoitteiden onnistumisen kannalta, ei voi liiaksi korostaa. Vaatimustenhallintaa ei pitäisi yrittää tehdä Excelillä oman toimen ohessa, vaan se pitää nähdä projektin hallinnan ja arkkitehtuurityön keskeisenä, jopa elintärkeänä työkaluna, jonka vaikutus heijastuu sopimusneuvotteluista yksittäisiin teknisiin ratkaisuihin. Kyse on osin työtä tukevan ohjelmiston valinnasta, mutta laajemmin myös muutoshallinnasta ja kokonaisarkkitehtuurityöstä.

---

[27] ”Although security is everyone’s job, it’s important to remember that not everyone needs to be a security expert nor strive to become a proficient penetration tester. However, ensuring everyone understands the attacker’s perspective, their goals, and the art of the possible will help capture the attention of everyone and raise the collective knowledge bar.” (sic)

[28] ”Obviously, the optimal time to define the security requirements is during the initial design and planning stages as this allows development teams to integrate security in ways that minimize disruption. Factors that influence security requirements include (but are not limited to) the legal and industry requirements, internal standards and coding practices, review of previous incidents, and known threats.”

2.2.3 Käytäntö #3 – Määrittele mittarit ja noudattamisraportointi

Tietoturvavaatimusten toteutumisen seuranta edellyttää, että sitä voidaan kvantitatiivisesti testata ja mitata. Mitattavalle datalle voidaan määritellä raja-arvot ja näin ollen myös tietoturvan toteutumiselle voidaan määritellä minimi- ja tavoitetasot.

Yksi esimerkki on tuotannossa olevan ohjelmiston virheraportit. Paljonko tietokannassa on virheraportteja (engl. bug report) ja mitkä ovat niiden kriittisyyden tasot? Miten kauan keskimäärin kestää, että virhe korjataan sen jälkeen, kun se on ensimmäisen kerran raportoitu? Jos käytäntönä on, että uutta ohjelmistoversiota ei julkaista ennen kuin kaikki kriittiset ja merkittävät (engl. critical, major) virheet on korjattu, niin miten usein tästä säännöstä poiketaan esimerkiksi aikataulupaineisiin vedoten?

Niin tietoturvan kuin hankkeen kannalta muutkin keskeiset laadulliset mittarit (engl. key performance indicator, KPI) ja niihin liittyvät tiedon keräys- ja raportointikäytännöt tulee miettiä heti hankkeen alussa, jotta ne voidaan asianmukaisesti suunnitella osaksi arkkitehtuuria, huomioida toteutus- ja testausvaiheessa sekä hyödyntää tuotannossa että organisaation sisäisessä seurannassa yleisemminkin.

3 VAATIMUSMÄÄRITTELY

Vaatimusmäärittelyn tavoitteena on tunnistaa, analysoida, priorisoida sekä dokumentoida hankkeeseen vaikuttavat vaatimukset. Huolellisen vaatimusmäärittelyn merkitystä hankkeen onnistumisen kannalta ei voi korostaa liiaksi: kehityshankkeessa ei ole ainuttakaan osa-aluetta tai vaihetta, johon vaatimusmäärittely ei vaikuttaisi. Olennaisessa roolissa tässä työssä on *osakkaiden* (engl. stakeholders) tunnistaminen sekä näiden (usein osin ristiriitaistenkin) tarpeiden kerääminen ja yhteensovittaminen.

Osakas on mikä tahansa yksilö, ryhmä tai organisaatio, joka jollain tavalla vaikuttaa järjestelmään (esim. käyttäjät) tai joihin järjestelmä jollain tavalla tulee vaikuttamaan (esim. varastonhallinnan järjestelmä, joka tulee vaikuttamaan myyntijärjestelmienkin käyttäjiin). Näiden kahden pääryhmän välissä osakkaisiin lukeutuvat myös ne tahot, jotka mm. rahoittavat järjestelmän kehityksen edistääkseen omia liiketoiminnallisia tavoitteitaan.

Vaatimusmäärittely tarkastelee suunniteltavaa järjestelmää eri näkökulmista. Edellä mainittujen osakkaiden näkökulmien lisäksi tyypillisiä vaatimusmäärittelyn osa-alueita ovat **toiminnallinen määrittely** (*functional specification*), **ei-toiminnalliset vaatimukset** (*non-functional requirements*), **liiketoiminnalliset vaatimukset** (*business requirements*) ja **tietoturvavaatimukset** (*information security requirements*). Näistä liiketoiminnalliset vaatimukset määritellään tyypillisesti jo esitutkimusvaiheessa, kun mahdollisen hankkeen kannattavuutta selvitetään. Tietoturvavaatimukset puolestaan edellyttävät, että toiminnallinen ja ei-toiminnallinen määrittely on tehty vähintäänkin yleisellä tasolla ja onkin tyypillistä, että vaatimusmäärittely tarkentuu hankkeen edetessä.

Tilaajan / osakkaiden näkökulmasta vaatimusmäärittely ilmaisee tarpeen ja tahtotilan, jonka mukaisesti kehitettävää järjestelmää sekä sen lopullista toimitusta tullaan arvioimaan. Toteuttaako kehitetty järjestelmä kaikki sille asetetut vaatimukset? Jos ei, järjestelmä ei vastaa tarkoitustaan ja hanke on todennäköisesti epäonnistunut. Toimittajan näkökulmasta huolellinen vaatimusmäärittely hälventää käynnistyville hankkeille yleistä epätietoisuutta, ohjaa järjestelmäsuunnittelua keskittymään olennaiseen, tehostaa laadunvalvontaa sekä yleisesti ottaen toimii kuin vakuutus, jos tilaaja riitauttaa

toimituksen vedoten siihen, että se ei vastaa tarkoitustaan; jos järjestelmä toteuttaa sille asetetut vaatimukset, ongelma ei ole toteutuksessa vaan vaatimusmäärittelyssä.

Tekniseen toteutukseen ei oteta tässä vaiheessa kantaa, mutta tässä tehtävä määrittely rajaa ja ohjaa myöhemmin tehtävää teknistä suunnittelua.

3.1 RFC 2119 – Vaatimusmäärittelyn avainsanat

Yleisesti noudatettuna käytäntönä (engl. best practice) on kuvata vaatimusmäärittelyn vaatimustasot RFC 2119[29] [30] mukaisin avainsanoin:

> The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

RFC 2119 ohjeistus julkaistiin vuonna 1997, mutta 2017 sitä tarkennettiin RFC 8174[31] julkaisussa. Keskeinen tarkennus on, että ainoastaan nämä avainsanat tulisi tulkita RFC 2119 mukaisesti ainoastaan silloin, kun ne on kirjoitettu suuraakkosilla. Tarkennuksen tarkoituksena oli selventää tulkinnan epäselvyyksiä, jotka johtuivat avainsanojen kirjoitusasun vaihdellessa suur- ja pienaakkosten välillä.

RFC 2119 on luettavissa liitteenä 3.

3.2 Toiminnallinen ja ei-toiminnallinen määrittely

Toiminnallinen ja ei-toiminnallinen määrittely on järjestelmäsuunnittelun ja ohjelmistokehityksen kulmakiviä. Toiminnallisessa määrittelyssä kuvataan ketkä järjestelmää käyttävät eli tunnistetaan käyttäjätyypit / -roolit ja mitä toimintoja heidän on voitava suorittaa, kun taas ei-toiminnallinen määrittely kuvaa järjestelmän toiminnan puitteet.

---

[29] *Request for Comments* (RFC) on Internetin käytäntöjä ja standardeja kehittävien ja hallinnoivien organisaatioiden, kuten *Internet Society* (ISOC) ja *Internet Engineering Task Force* (IETF) julkaisu. Näissä julkaisuissa vertaisarvioitavaksi mm. käytäntöjä, käytäntöjä ja ratkaisuja, joista osa edelleen kypsyy mm. IETF standardeiksi.

[30] https://tools.ietf.org/rfc/rfc2119.txt

[31] https://tools.ietf.org/rfc/rfc8174.txt

3.2.1 Toiminnallinen määrittely

Toiminnalliseen määrittelyyn on useita lähestymistapoja. Yksi esimerkki tästä on ketterissä menetelmissä käytetty *käyttäjätarina* (engl. user story), joka on epämuodollinen, proosallinen kuvaus järjestelmän toiminnosta. Tyypillisesti nämä kirjoitetaan loppukäyttäjän näkökulmasta.

Tässä tutkintotyössä toiminnallisen määrittelyn menetelmänä on UML kuvantamiskielen mukainen *käyttäjätapaus* (engl. use case), joka kuvaa kattavasti *toimijan* (engl. actor) suorittaman toiminnon vaiheet ja niihin mahdollisesti sisältyvät poikkeukset ja virhetilat.

> Käyttötapaukset ovat menetelmä järjestelmän vaatimusten määrittelemiseksi eli ne kertovat, mitä järjestelmän tulisi tehdä. Keskeiset käsitteet ovat *toimijat, käyttötapaukset* ja *subjektit*. Jokaisen käyttötapauksen *subjekti* kuvaa käsillä olevaa järjestelmää. Käyttäjät ja muut vuorovaikutuksessa olevat järjestelmät ovat *toimijoita*. Käyttötapaus on toiminnallinen määritelmä.[32] (UML 2.5, 637)

Tarkkuus, jolla käyttötapaus määritellään, tyypillisesti tarkentuu määrittely- ja suunnittelutyön edetessä. Työ alkaa tunnistamalla järjestelmän ja sen käyttäjien kannalta keskeisimmät käyttötapaukset, joista kirjataan *nimi* ja *lyhyt kuvaus*. Työn edetessä käyttötapausta tarkennetaan kuvaamalla pääpiirteissään toiminnan vaiheet ja yksityiskohtaisen kuvauksen tulisi olla valmis, kun varsinainen kehitystyö alkaa. Hyvin kirjoitettu käyttötapaus toimii myös lukuisien testitapausten pohjana.

Käyttötapausten määrittelyn tulisi keskittyä keskeisten rajapintojen ja vuorovaikutusten määrittelyyn eikä niillä tule yrittää kuvata järjestelmää kokonaisuudessaan.

Käyttötapauksella on yksi alkupiste ja yksi loppupiste sekä täsmälleen yksi tavoite; yksi käyttötapaus kuvaa yhden toiminnallisuuden. Alku- ja loppupisteen

---

[32] ”UseCases are a means to capture the requirements of systems, i.e., what systems are supposed to do. The key concepts specified in this clause are *Actors, UseCases*, and *Subjects*. Each UseCase’s subject represents a system under consideration to which the UseCase applies. Users and any other systems that may interact with a subject are represented as Actors. A UseCase is a specification of behaviour.”

välissä voi olla kuitenkin useitakin vaihtoehtoisia polkuja ja määritellyillä ehdoilla toiminto voi myös keskeytyä virheeseen.

Käyttötapauksien rakenteissa on vaihtelua, mutta tyypillisesti niihin sisältyy:

- Käyttötapauksen nimi
- Käyttötapauksen yksilöllinen tunniste
- Toimija(t)
- Prioriteetti
- Ennakkotila / -ehdot
- Lopputila / -ehdot
- Peruspolku
- Poikkeavat polut
- Virhepolut

Katso *liite 1: Tyypillisen käyttötapauksen rakenne*.

Hyvänä käytäntönä on lisäksi viitata käyttötapauksen kuvauksessa siihen suoraan liittyviin liiketoimintasääntöihin sekä ei-toiminnallisiin vaatimuksiin.

3.2.2 Ei-toiminnallinen määrittely

> One set of requirements that impact the software architecture can be described as the ’ilities’: reliability, maintainability, testability, and usability. Some others include fault tolerance, error handling, security, portability, and performance. (Garland, Anthony, 2003, 49)

Ei-toiminnalliset vaatimukset pyrkivät määrittelemään *miten* järjestelmän tulisi toimia siinä missä toiminnalliset vaatimukset kuvaavat *mitä* järjestelmä tekee. Toiminnalliset vaatimukset määrittelevät järjestelmän toiminnot, kun taas ei-toiminnalliset vaatimukset määrittelevät järjestelmän arkkitehtuurin olennaiset ominaisuudet.

Jos kehiteltävä järjestelmä on organisaation toiminnan kannalta kriittisessä roolissa olennaisiin ominaisuuksiin todennäköisesti kuuluvat esimerkiksi *vikasietoisuus* (engl. fault tolerance), *vasteaika* (engl. responsiveness) ja *saatavuus* (engl. availability). Tiedon suojaamisen kannalta olennaista

ominaisuuksia voivat olla *salausmenetelmät* (engl. encryption methods), *hajautetut tietokannat* (engl. distributed databases) ja *salausavaimien hallinta* (engl. cryptographic key management). Jos organisaation infrastruktuuri perustuu Microsoftin teknologioihin tai AWS pilvipalveluihin, vaihtoehtoisten ratkaisujen pois rajaaminen voi olla perusteltua.

Lokien hallinta on yksi käytännön esimerkki ei-toiminnallisesta vaatimusmäärittelystä: mitä tietoja tulee kirjata lokeihin (esim. järjestelmän tilatietoja tai käyttäjien toimintaa seuraava auditointiloki, mitä tietoja lokeihin *ei saa* kirjata (esim. salasanat, henkilötunnukset, pankkikorttien tiedot), miten lokitietoja kerätään (esim. käytetäänkö paikallista hakemistoa vai erillistä lokipalvelinta), miten kauan lokeja säilytetään, miten niitä käsitellään ja millaisilla valtuutuksilla. Millaisilla säännöillä lokeja kirjoitetaan kehityksen, testauksen ja tuotantokäytön aikana? Mitkä lait ja säädökset tulee ottaa huomioon lokien suunnittelussa – varomattomasti muodostettu loki voikin paljastua laittomaksi henkilörekisteriksi.

3.3 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013)

> Vaatimusmäärittelyn suunnittelussa otetaan huomioon sovelluksen tietoturvataso ja kehitettävän sovelluksen kriittisyys liiketoiminnalle. Vaatimusmäärittelyssä on tärkeää kartoittaa sovellukselle asetetut ympäristöstä tulevat integrointivaatimukset ja ottaa huomioon organisaation kokonais- ja tietoturva-arkkitehtuuri. Uhkamallinnuksella ja riskianalyysilla haetaan rakennettavan sovelluksen potentiaalisia tietoturvallisuuden heikkouksia ja valitaan sovelluksessa käyttöön otettavat suojaukset ja kontrollit. (VAHTI 1/2013, 45)

Tässä kohden on syytä huomauttaa, että VAHTI 1/2013 vaatimusmäärittelyä käsittelevän luvun tavoitteena ei ole ohjeistaa, miten vaatimusmäärittelyä yleisellä tasolla tulisi tehdä vaan kyseessä on Valtionhallinnon tietoturvallisuuden johtoryhmän julkishallinnon hankkeille määrittelemät vaatimukset: asiat, jotka hankkeiden tulee huomioida ja määritellä osana suunnitteluprosessia.

Vaatimusmäärittelyä käsittelevä ohjeistus on kuitenkin soveltuvin tason hyödyllistä huomioida myös muiden organisaatioiden hankkeissa. Huomioitavaa on myös se, että alla käsiteltävät vaatimukset menevät osin limittäin eli eri vaatimuksissa käsitellään osittain samoja asioita, mutta hieman eri näkökulmista.

Näin ollen myös käytettävät menetelmät ja niistä johdetut tulokset voivat myös toistua eri vaatimusten mukaisessa dokumentaatiossa.

3.3.1 Perustaso: Tietoturvaratkaisuiden dokumentointi (VTM-001)

> Tietoturvavaatimukset toteuttavat ratkaisut tulee dokumentoida. jolloin dokumentit toimivat suunnittelun pohjana sovelluksen kehittäjille ja ratkaisujen riittävyys voidaan todentaa. Dokumenttien päivitys ja katselmointi pitää organisoida ja vastuuttaa. (VAHTI 1/2013, 45)

Toisin sanoen, jokainen menetelmä, kirjasto tai muu ratkaisu, joka toteuttaa tietoturvavaatimuksen, on oltava dokumentoitu ja sen ajantasaisuudesta vastaa nimetty henkilö tai ryhmä. Esimerkiksi tietojen salaamiseen valittu kryptografinen menetelmä tulee olla yksiselitteisesti määritelty ja dokumentoitu, jotta tieto olisi kehittäjien, testaajien ja katselmoijien käytettävissä.

3.3.2 Perustaso: Lainsäädännölliset vaatimukset (VTM-002)

> Sovelluksen pitää huomioida käsiteltyyn tietoon, sovelluksen toimintaympäristöön tai muihin tekijöihin vaikuttavat lainsäädännölliset vaikutukset. (VAHTI 1/2013, 45)

Kehitettävän sovelluksen tai järjestelmän luonteesta riippuu mitä velvoittavaa lainsäädäntöä siihen kohdistuu. Esimerkiksi käyttäjien henkilötietoja tallentavan järjestelmän yhteydessä tulee huomioida mm. Tietosuojalaki (5.12.2018/1050) ja EU:n GDPR[33] säädökset – olettaen, että tiedon fyysinen tallentaminen tapahtuu EU:n ja Suomen rajojen sisällä.

Eräänlaisena nyrkkisääntönä on, että mitä lainsäädäntöä noudatetaan, riippuu pitkälti siitä, missä maassa tai alueella palvelinjärjestelmä fyysisesti sijaitsee. EU:n lainsäädäntö velvoittaa vain niitä palveluita, joiden palvelimet sijaitsevat EU:n alueella. Jos tieto siirretään esimerkiksi Yhdysvalloissa sijaitsevaan tietojärjestelmään, on lainsäädännön vaikutus toisenlainen.

[33] General Data Protection Regulation. (EU) 2016/679

3.3.3 Perustaso: Tietoturva-analyysi (VTM-003)

> Sovelluksen suunnittelu tulee aloittaa käymällä läpi esitutkimusvaiheessa määritelty sovellusprofiili ja määrittelemällä sen asettamat yleisen tason tietoturvavaatimukset. Analyysin tuotteena saadaan karkean tason kuva sovelluksen tietoturvavaatimuksista tietoturvallisuuden eri näkökulmista (luottamuksellisuus, eheys ja saatavuus). Analyysia käytetään sovelluksen suunnittelun pohjana, jolloin järjestelmän tietoturvaominaisuudet suunnitellaan suojattavan tiedon sekä järjestelmän kriittisyyden mukaan. (VAHTI 1/2013, 46)

Sovelluksen tietoturvaa tarkasteluun eri näkökulmista on useita menetelmiä, kuten esimerkiksi Microsoft STRIDE (taulukko 1).

TAULUKKO 1. STRIDE mallin sisältö

Uhka	Vaikutus
Spoofing	Hyökkääjä omaksuu jonkin toisen tahon identiteetin, esimerkiksi väärentämällä IP-osoitteensa tai sähköpostinsa.
Tampering	Luottamus järjestelmän datan eheyteen menetetään. Esimerkiksi lokeista voidaan poistaa tiedot hyökkääjän toimenpiteistä, väärää tietoa voidaan syöttää tai olemassa olevaa tietoa voidaan muuttaa.
Repudiation	Datan kiistämättömyyden kyseenalaistaminen. Esimerkiksi järjestelmän tehtävänä on taata tiedon luotettavuus ja alkulähde, mutta hyökkääjän toimien vaikutuksesta luottamus menetetään, jolloin sekä luotettavuus että alkulähde voidaan kiistää.
Information disclosure	Datan luottamuksellisuuden menetys. Ulkopuoliset tahot pääsevät käsiksi luottamukselliseen tietoon, esimerkiksi käyttäjätunnus-salasana -yhdistelmiin tai luottokorttitietoihin.
Denial of service	Tyypillisesti palvelunestohyökkäys estää käyttäjiä käyttämästä järjestelmää joko kokonaisuudessaan tai tiettyjen toiminnallisuuksien osalta, mutta yleisesti ottaen kyseessä voi olla mikä tahansa toimenpide, mikä estää sovelluksen normaalin käytön.
Elevation of privilege	Hyökkääjä onnistuu laajentamaan valtuuksiaan järjestelmässä. Tämä voi esimerkiksi tarkoittaa sitä, että olemassa oleva peruskäyttäjä onnistuu lisäämään tunnukselleen ylläpitäjän käyttöoikeuksia. Sen sijaan tällä ei lähtökohtaisesti tarkoiteta toisen käyttäjän tilin kaappaamista.

Tätä karkean tason analyysia tarkennetaan, jos kyseessä on korotetun tason (VTM-010) tai korkean tason (VTM-011) järjestelmä.

3.3.4 Perustaso: Sovelluksen tietoturvataso (VTM-004)

> Järjestelmän omistajan vastuulla on määritellä, mitä tietoturvatasoa toteuttavan järjestelmän tulee toteuttaa. Järjestelmän omistaja on lisäksi vastuussa järjestelmästä ja sen sisältämästä tietoaineistosta. Lisäksi omistajan tulee määritellä vaatimukset tietoaineiston arkistoinnille yhteistyössä organisaation asiakirjahallinnon kanssa. (VAHTI 1/2013, 46)

Valtionhallinnossa tietoturvatasoja on kolme: perustaso, korotettu taso sekä korkea taso. Salassa pidettävä tietoaineisto sijoitetaan tiedon merkityksen ja sen paljastumisen seurausten mukaan määräytyvään suojaustasoon (VAHTI 2/2010, 55).

VAHTI 2/2010 eli Ohje tietoturvallisuudesta valtionhallinnossa annetun asetuksen täytäntöönpanosta määrittelee, miten suojaustasot (sivu 57) ja tietoturvatasot rinnastetaan keskenään (sivu 16):

- **Suojaustaso IV** tarkoittaa tiedon *rajoitettua käyttöä,* jota voidaan käsitellä järjestelmissä, joiden tietoturvaratkaisut toteuttavat **perustason** vaatimukset.
- **Suojaustaso III** tarkoittaa *luottamuksellista* tietoa, jota voidaan käsitellä järjestelmissä, joiden tietoturvaratkaisut toteuttavat **korotetun tason** vaatimukset.
- **Suojaustaso II** tarkoittaa *salaista* tietoa, jota voidaan käsitellä järjestelmissä, joiden tietoturvaratkaisut toteuttavat **korkean tason** vaatimukset.
- **Suojaustaso I** tarkoittaa *erittäin salaista* tietoa, jota ei lähtökohtaisesti tallenneta sähköisiin järjestelmiin.

Nämä säädökset velvoittavat myös yrityksiä, jotka kehittävät ja käyttävät valtionhallinnon ja puolustusvoimien tietojärjestelmiä. Käytännössä kaikki yritykset ja muut organisaatiot luokittelevat informaation julkiseen tietoon (esim. kotisivut ja tiedotteet) ja luottamukselliseen tietoon (esim. asiakas- ja taloustiedot). Usein voi olla lisäksi tarve huolehtia siitä, että eri asiakkaiden tiedot pysyvät toisistaan erillään ja että niihin pääsevät käsiksi vain ne, joilla on siihen perusteltu syy. Toisin sanoen, jokaisessa hankkeessa, tavalla tai toisella, on määriteltävä käsiteltävän tiedon luottamuksellisuus.

3.3.5 Perustaso: Sovelluksen riskianalyysi (VTM-005)

> Sovelluksen tietoturvavaatimukset tulee ottaa huomioon jo suunnitteluvaiheessa kohdistamalla sovellukseen riskianalyysi. Vaatimukset määräytyvät järjestelmän omistajan tekemän tietoturvatasomäärityksen ja käytetyn autentikaatiomenetelmän valinnan kautta. Riskianalyysin ytimessä ovat sovelluksen tietoturvallisuuden pahimmat mahdolliset tapaukset (worst case scenario), jotka tulee listata käyttämällä lähtökohtana sovelluksen liiketoimintatarkoitusta, sovelluksen käsittelemää tietoa ja liiketoiminnan riskiprofiilia. (VAHTI 1/2013, 46)

Tämä liittyy läheisesti VTM-003 Tietoturva-analyysiin ja menetelmät voivat olla osittain samojakin. Riskianalyysin suorittamiseen on olemassa useita menetelmiä, joista tässä yhteydessä voisi mainita kaksi: **FMEA** ja **DREAD**.

FMEA eli *Failure Modes and Effects Analysis* menetelmän tavoitteena pohtia kaikkia tapoja, joilla tuote tai palvelu osoittautuu toimimattomaksi. Tuotteen toimimattomuus ilmenee, kun se ei toimi tarkoitetulla tavalla tai siinä ilmenee häiriöitä. Jopa yksinkertaisinkin tuote voi osoittautua toimimattomaksi monin eri tavoin.[34] (McDermott, Mikulak, Beauregard, 2009, 9).

Menetelmänä FMEA on melko suoraviivainen. Ensimmäiseksi tunnistetaan tarkasteltavan tuotteen komponentit ja toiminnallisuudet. Seuraavaksi tarkastellaan kutakin komponenttia tai toiminnallisuutta ja pyritään tunnistamaan mahdollisimman monta tapaa, joilla se voisi rikkoutua tai muulla tavoin toimia väärin; nämä ovat tuotteen *virhetiloja* (engl. failure modes).

Kukin virhetila arvioidaan kolmella kriteerillä, jotka pisteytetään yhdestä kymmeneen.

- Mikä on virhetilan mahdollinen vaikutus? Vaikutuksen *vakavuus* arvioidaan käytännössä merkityksettömästä (1) äärimmäisen vakavaan (10).
- Mikä on virhetilan mahdollinen syy? Syyn *ilmenemisen* arvioidaan erittäin harvinaisesta (1) erittäin yleiseen (10).

[34] ”The objective of FMEA is to look for all of the ways a process or product can fail. A product failure occurs when the product does not function as it should or when it malfunctions in some way. Even the simplest products have many opportunities for failure.”

- Miten helppoa tai vaikeaa virhetila on havaita? *Havaittavuus* arvioidaan erittäin helposta (1) lähes mahdottomaan (10).

Lisäksi pohditaan tällä hetkellä käytettävissä olevia keinoja estää virhetilaa tapahtumasta sekä millä keinoin toteutunut virhetila voidaan havaita. Virhetilalle lasketaan ns. riskiprioriteettiarvo (engl. Risk Priority Number, RPN) kaavalla: `vakavuus x ilmeneminen x havaittavuus`, jolloin kunkin virhetilan riski voidaan arvioida välillä 1 ja 1000.

Lopuksi laaditaan suositukset toimenpiteistä, joilla riski voidaan joko kokonaisuudessaan eliminoida tai jos se ei ole mahdollista, niin keinoista rajata toteutuneen riskin seurauksia, pienentää virhetilanteen ilmenemisen todennäköisyyttä tai parantaa havaittavuutta, jotta korjaaviin toimenpiteisiin voidaan ryhtyä nopeammin.

On suositeltavaa, että FMEA suoritetaan hankkeen aikana useammankin kerran, jotta voidaan arvioida suositeltujen toimenpiteiden toteutumista ja vaikutusta.

DREAD (taulukko 2) on viisikohtainen muistisääntö tietoturvallisuuden riskianalyysin suorittamiseksi, jota voi käyttää itsenäisenä menetelmä tai FMEA:n osana näkökulmien ominaisuudessa:

TAULUKKO 2. DREAD mallin sisältö

Damage	Mitä vahinkoa hyökkäys voi aiheuttaa?
Reproducibility	Mitä vaivatonta hyökkäys on toistaa?
Exploitability	Miten vaativaa hyökkäyksen suorittaminen on?
Affected users	Miten suureen ihmisjoukkoon hyökkäys vaikuttaa?
Discoverability	Miten helppoa hyökkäyksen uhan havaitseminen on?

Kukin arvo pisteytetään yhdestä kymmeneen ja riskit priorisoidaan pisteiden yhteissumman perusteella.

Riskianalyysin vähimmäistasona voi pitää Sovelluskehityksen tietoturvaohjeen mukaista ohjeistusta: jokainen riski tulee kuvata yhdellä lauseella ja määritellä se

hyökkääjän korkean tason tavoitteiksi. Tämän jälkeen jokaiselle riskille tulee määritellä esiehdot, joiden pitää päteä jokaiselle hyökkääjän tavoitteen onnistumiselle. Informaatio voidaan esittää ns. uhkapuuna tai rakenteellisena listana. (VAHTI 1/2013, 46)

3.3.6 Perustaso: Sovelluksen tietoturvavaatimukset (VTM-006)

> Kehitettävälle sovellukselle tulee määritellä tietoturvavaatimukset. Sovelluksen tietoturvavaatimukset tulee johtaa seuraavista lähtökohdista:
> - Sovelluksen kriittisyys liiketoiminnalle
> - Sovelluksen sisältämän ja käsittelemän tiedon asettamat vaatimukset
> - Sovelluksen toiminnallisista vaatimuksista johdetut vaatimukset
> - Uhka-analyysi
>
> (VAHTI 1/2013, 46)

Akronyymi CIA on tietoturvapiireissä yleisesti[35] käytetty muistisääntö: *Confidentiality, Integrity, Availability*. Toisin sanoen, määriteltäessä tietoturvavaatimuksia on tehtävä selväksi, kuinka turvataan järjestelmän sisältämän tiedon *luottamuksellisuus, eheys* ja *saatavuus*.

Mikä vaikutus liiketoiminnalle olisi, jos kyseisen järjestelmän sisältämät tiedot vuotaisivat ulkopuolisten tietoon? Entä millainen vaikutus sillä olisi, jos järjestelmän tiedot korruptoituisivat, esimerkiksi ulkopuolisten tahojen päästessä tekemään niihin muutoksia? Tai miten liiketoiminnalle kävisi, jos tietoja ei olisikaan käytettävissä?

3.3.7 Perustaso: Lait, määräykset ja ohjeistukset (VTM-007)

> Tietoturvavaatimusten määrittelyn pohjana tulee käyttää myös soveltuvia lakeja, määräyksiä, standardeja ja ohjeistuksia. Tällaisia ohjeistuksia voivat olla muun muassa toimialan best practice -suositukset, liiketoimintalaan tai valittuihin teknologioihin soveltuvat standardit, compliance -vaatimukset, ulkoiset tai sisäiset vaatimukset jne. Liiketoiminnan omistajat ja arkkitehdit määrittelevät käytettävät ohjeistukset. (VAHTI 1/2013, 47)

---

[35] Alkuperäistä lähdettä CIA akronyymille lienee lähes mahdotonta jäljittää. Siihen on viitattu jo 90-luvulla ilman lähdeviittauksia, ja käsitteet itsessään ovat olleet olemassa jo kauan ennen kuin tietoturvan käsitteen määrittelyä.

Suomessa luotettavia peruslähteitä ovat Suomen ajantasainen lainsäädäntö (https://www.finlex.fi/), Liikenne- ja viestintäviraston Kyberturvallisuuskeskus (https://www.kyberturvallisuuskeskus.fi/fi/ohjeet), mutta kaiken kaikkiaan kattavaa listausta ei voi tässä yhteydessä antaa: mahdollisia lähteitä on valtavasti ja sovellettavat tiedot vaihtelevat olennaisesti kehitettävän järjestelmän, toimialan ja organisaatiokohtaisten tarpeiden mukaan.

3.3.8 Korotettu taso: Uhkamallinnus (VTM-008)

> Sovellukselle on tehtävä uhkamallinnus, joka perustuu johonkin valittuun metodiin tai malliin. Esimerkkejä malleista ovat Microsoft STRIDE, DREAD ja väärinkäyttötapausten (abuse case) analyysi. Myös muita malleja voidaan soveltaa, esimerkiksi organisaation omaa mallia. (VAHTI 1/2013, 47)

Näihin metodeihin sekä uhkamallinnuksen suorittamiseen viitataan luvuissa 3.2.3, 3.2.4 ja 3.2.5. Uhkamallinnus on suositeltavaa tehdä myös perustason sovelluksille, mutta se on ehdottomasti tehtävä korotetun tason sovelluksille.

3.3.9 Korotettu taso: Arkkitehtuurilinjaus (VTM-009)

> Sovelluksen hankkivalla organisaatiolle tulee olla arkkitehtuurilinjaus, jonka mukaisia hankittavien järjestelmien tulee olla. […] Tällöin varmistutaan siitä, että kaikki hankittavat järjestelmät täyttävät tietyt tietoturvakriteerit ja sopivat hankkivan organisaation tietoturvainfrastruktuuriin. (VAHTI 1/2013, 48)

Arkkitehtuurilinjaus määrittelee yleiset suuntaviivat organisaation tarpeiden kannalta keskeisille tietoteknisille ratkaisuille. Esimerkiksi linjauksena voi olla, että hankittavien järjestelmien tulee toimia määrätyn pilvipalvelun teknisissä puitteissa tai että järjestelmien tulee perustua ennalta määrättyihin salausmenetelmiin, tunnistautumismenetelmiin tai lokien keräysmenetelmiin. (VAHTI 1/2013, 48)

Arkkitehtuurilinjaus voidaan nähdä eräänlaisena teknisenä sopimuksensa tai yhteisymmärryksenä järjestelmän omistajien ja kehittäjien välillä, joka arkkitehtuurisuunnitelman välityksellä määrittelee kehitettävät järjestelmät ja

osajärjestelmät sekä näiden liittymät rinnakkaisiin järjestelmiin.[36] (NAF4, 2018, 29)

3.3.10 Korotettu taso: Uhkamallinnuksen tarkennus (VTM-010)

> Rakennettua uhkamallia tulee tarkentaa ottamalla mukaan uhkia, jotka liittyvät sovelluksen toteutukseen tai arkkitehtuuriin. Tällaisia asioita ovat muun muassa:
> - Käyttäjäroolit
> - Tietoturvaoletukset
> - Käytetyt teknologiat
> - Tietoturvamekanismit
>
> (VAHTI 1/2013, 48)

Käytännössä käyttäjäroolit tulisi tunnistaa jo perustason sovellusten toiminnallisessa määrittelyssä, sillä käyttäjäroolit ovat olennainen tieto käyttötapauksia laadittaessa. Se, että VAHTI 1/2013 ohjeistuksessa viitataan toistuvasti uhkamallinnuksen suorittamiseen eri uhkatasoilla korostaa miten merkittävästä ja tärkeästä asiasta on kyse: ei ole niin yksinkertaista tuotannossa käytettävää sovellusta, että sen tietoturvaan voisi suhtautua naiivisti. Vähintäänkin tulisi aina ymmärtää sovelluksen käyttötarkoitus ja -ympäristö, sen käsittelemien tietojen luonne, julkiseen Internettiin avatut rajapinnat (erityisesti avoimet portit ja niiden tarjoamat oletuspalvelut) sekä se, miten huolehditaan käytettyjen palvelinohjelmistojen tietoturvapäivitysten ajantasaisuudesta.

3.3.11 Korkea taso: Komponenttien uhka-arvio (VTM-011)

> Sovelluksen käyttämille kolmansien osapuolten komponenteille tulee tehdä uhka-arvio. Näitä komponentteja ovat esimerkiksi avoimen lähdekoodin kirjastot, online-palvelut tai COTS-ohjelmistot (commercial off the shelf, valmisohjelmistot). Tulee selvittää, miten haavoittuvuudet tai suunnitteluvirheet komponenteissa vaikuttavat kehitettävän sovelluksen tietoturvallisuuteen. (VAHTI 1/2013, 48)

Uhka-arvion suorittamiseen on useita menetelmiä, mutta pääsääntöisesti uhka-arviossa pyritään vastaamaan viiteen keskeiseen kysymykseen:

---

[36] ”Architecture provides a technical contract to system owners and builders, through an architecture plan, by framing candidate systems and subsystems of interest and associated enabling systems. This includes the critical path from the earliest baseline to its numerous increments, which are handled by appropriate versions of the system engineering management plans.”

1. Mitä pitää suojella?
2. Kuka tai mikä muodostaa uhan ja millaisiin haavoittuvuuksiin se kohdistuu?
3. Mitä uhan toteutumisesta (esim. vahingoittumisen tai menetyksen myötä) seuraa?
4. Mikä on suojattavan kohteen arvo toimeksiantajalle?
5. Millä keinoin uhkaa ja sen vaikutuksia voidaan rajata?

Uhka-arvion tuotoksena tulisi syntyä joukko suosituksia toimenpiteistä, joilla suojattavan kohteen hyökkäyspintaa voidaan pienentää ja/tai vaikutuksia rajata, mikäli uhan toteutumista ei voida suoranaisesti estää. Uhka-arvio etenee vaiheittain, esim.

1. Määrittele arvioitava kohde sekä arvioinnin laajuus ja/tai soveltamisala.
2. Kerää tietoa arvioitavasta kohteesta. Menetelmät vaihtelevat kohteen mukaan (esim. henkilö, järjestelmä, komponentti, kiinteistö), mutta ohjelmistoissa olennaisia tietoja ovat yleensä käsiteltävä tieto, rajapinnat ja liittymät muihin järjestelmiin, käyttäjäryhmät, käyttöoikeuksien hallinta, käyttö-/laiteympäristö, laatukriteerit ja standardit ja niin edelleen.
3. Arvioitavan kohteen toiminnan ja käytön analysointi (kun kyseessä on ohjelmisto tai tietojärjestelmä)
4. Uhkien tunnistaminen. Mahdollisia menetelminä on useita, kuten esimerkiksi hyökkäyspuu[37], Microsoftin STRIDE[38] sekä uhkalistaukset (e.g. OWASP Top 10). Tarvittaessa voidaan myös suorittaa haavoittuvuusanalyysi esimerkiksi CVSS[39] menetelmällä tai kun kohteena on tietojärjestelmä, skannaamalla sen rajapinnat esimerkiksi Nessus, SAINT tai Burp Suite -ohjelmistoilla.
5. Laadi suositukset uhkien hallinnalle: rajataan toteutuneen uhan vaikutuksia, mikäli uhkaa ei voi suoranaisesti poistaa.
6. Uhkien ja toimenpiteiden priorisointi.

[37] Attack Tree
[38] STRIDE: Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of Privilege
[39] Common Vulnerability Scoring System

Muita mainitsemisen arvoisia menetelmiä ovat riskianalyysissa usein käytetyt FMEA[40] ja DREAD[41].

3.4 Tietoturvakehityksen elinkaari (Microsoft SD)

Microsoftin tietoturvakehityksen elinkaaren 4. ja 5. käytäntö ovat olennaisia hankkeen vaatimusmäärittelyn kannalta, joskin niihin tulisi suhtautua myös yleisesti omaksuttavina, yksittäisestä hankkeesta riippumattomina toimintatapoina.

3.4.1 Käytäntö #4 – Suorita uhkamallinnus

Kuten myös Sovelluskehityksen tietoturvaohjeistuksessa toistuvasti tuodaan esille, uhkamallinnus on ehdoton edellytys sovelluksen, järjestelmän tai organisaation tietoturvan kehittämiseksi ja ylläpitämiseksi; uhat on tunnistettava, jotta niitä vastaan voisi valmistautua.

Uhkamallinnus tulisi tehdä ympäristöissä, joissa tietoturvariskit ovat merkittäviä. Uhkamallinnus voidaan kohdistaa komponenttiin, sovellukseen tai järjestelmään. Se on käytäntö, mikä edesauttaa kehitystiimejä pohtimaan, dokumentoimaan sekä keskustelemaan teknisen suunnitelman vaikutuksista tietoturvaan.[42] (Microsoft SDL, n.d.)

3.4.2 Käytäntö #5 – Määrittele suunnitteluvaatimukset

Suunnitteluvaatimukset ovat arkkitehtuurityötä ohjaavia, tarvittaessa myös rajaavia toiminnallisia määreitä; kaikkien arkkitehtuurityössä tehtävien ratkaisuiden on oltava linjassa ennalta määriteltyjen suunnitteluvaatimusten kanssa.

Tietoturvaan liittyvissä kysymyksissä tyypillinen suunnitteluvaatimus esimerkiksi kieltää kehittäjiä tuottamasta omaa, räätälöityä kryptografista ratkaisua vaan ohjeistaa aina käyttämään olemassa olevia, tunnettuja yleisesti käytettyjä

---

[40] Failure Modes and Effects Analysis
[41] DREAD: Damage, Reproducibility, Exploitability, Affected Users, Discoverability
[42] ”Threat modelling should be used in environments where there is meaningful security risk. Threat modelling can be applied at the component, application, or system level. It is a practice that allows development teams to consider, document, and (importantly) discuss the security implications of designs in the context of their planned operational environment and in the structured fashion.”

avoimen lähdekoodin ratkaisuja, koska näiden laatu ja luotettavuus on yksinkertaisesti, olennaisesti parempi kuin minkään kotikutoisen ratkaisun voi odottaa olevan ilman merkittävää investointia niin ajassa kuin rahassakin; valmiin, yleisesti luotettavana ja laadukkaana pidetyn ratkaisun käyttäminen on joitakin poikkeuksia lukuun ottamatta yksinkertaisesti se järkevin valinta.

Tämä on myös Microsoft SDL:n 5. käytännön sisältö. Tietoturvavaatimusten toteuttaminen edellyttää usein monenlaisia ratkaisuja, joiden epäonnistuneella toteutuksella voi olla monia vakavia, jopa liiketoiminnan kannalta kriittisiä seuraamuksia. Niinpä suunnitteluvaatimusten tulisi ennalta määritellä mitä valmiita ratkaisuja käytetään mm. tietoturvatoiminnallisuuksien toteutuksissa.[43] (Microsoft SDL, n.d.)

3.4.3 Käytäntö #6 – Määrittele kryptografiset standardit

Tietoturvakehityksen elinkaaren viides ja kuudes käytäntö liittyvät toisiinsa; kryptografisten standardien määrittelyn voi nähdä osana suunnitteluvaatimusten määrittelyä, mutta suunnitteluvaatimukset yleensä sisältävä kryptografian lisäksi myös muita osa-alueita. Kuten edellisessä luvussa mainittiin, kehittäjien tulisi lähtökohtaisesti pidättäytyä omien, kotikutoisten ratkaisujen kehittämisestä ja hyödyntää jo olemassa olevia, tunnettuja sekä laadukkaina ja luotettavina pidettyjä avoimen lähdekoodin kirjastoja osana kryptografisia ja muita tietoturvaan liittyviä ratkaisuja.[44] (Microsoft SDL, n.d.)

---

[43] ”The SDL is typically thought of as assurance activities that help engineers implement ”secure features”, in that the features are well engineered with respect to security. To achieve this, engineers will typically rely on security features, such as cryptography, authentication, logging, and others. In many cases, the selection or implementation of security features has proven to be so complicated that design or implementation choices are likely to result in vulnerabilities. Therefore, it is crucially important that these are applied consistently and with a consistent understanding of the protection they provide.”

[44] ”With the rise of mobile and cloud computing, it’s critically important to ensure all data, including security-sensitive information and management and control data, is protected from unintended disclosure or alteration when it’s being transmitted or stored. Encryption is typically used to achieve this. Making an incorrect choice in the use of any aspect of cryptography can be catastrophic, and it’s best to develop clear encryption standards that provide specifics on every element of the encryption implementation. This should be left to experts. A good general rule is to only use industry-vetted encryption libraries and ensure they’re implemented in a way that allows them to be easily replaced if needed.”

3.5 OWASP Application Security Verification Standard

OWASP ASVS on pohjimmiltaan kattava kokoelma vaatimuksia, jotka on jaettu neljääntoista kategoriaan ja edelleen luokiteltu tasoihin 1-3 vaatimuksen kriittisyyden mukaan: 1. tason vaatimukset ovat ehdottomia minimivaatimuksia, joiden täyttäminen riittää lähinnä tietoturvan kannalta vähäpätöisille sovelluksille, kun taas 2. tason vaatimukset ovat niitä, jotka kaikkien sovellusten tulisi lähtökohtaisesti toteuttaa. Viimeinen, 3. taso käsittelee kaikkein kriittisimpien järjestelmien vaatimuksia.

ASVS on erinomainen käsikirja hankkeen vaatimusmäärittelyn yhteydessä. Kun sovelluksen kriittisyys on määritelty ja rinnastettu ASVS tasoihin, vaatimusmäärittelijät voivat käydä ASVS:n vaatimuslistaukset soveltuvin osin läpi ja poimia sovelluksen kannalta merkitykselliset vaatimukset osaksi vaatimusmäärittelyä, jossa ne ohjaavat arkkitehtuurityötä, suunnittelua, toteutusta ja testausta.

ASVS:n toinen tärkeä käyttötapa on toimia tiekarttana sovelluksen tai järjestelmän tietoturvakartoitusta ja/tai -testausta tehtäessä: listaus käydään soveltuvin osin läpi tarkistaen, miten sovellus toteuttaa kyseiset vaatimukset. Vapaasti suomentaen ASVS:n vaatimuskategoriat ovat seuraavat:

- V1: Arkkitehtuuri, suunnittelu- ja uhkamallinnuksen vaatimukset[45]
- V2: Käyttäjätunnistuksen varmennuksen vaatimukset[46]
- V3: Sessionhallinnan varmennuksen vaatimukset[47]
- V4: Käytönhallinnan varmennuksen vaatimukset[48]
- V5: Vahvistamisen, siistimisen ja koodauksen varm. vaatimukset[49]
- V6: Tallennetun tiedon kryptografian varmennuksen vaatimukset[50]
- V7: Virheenkäsittelyn ja lokituksen varmennuksen vaatimukset[51]
- V8: Datan suojauksen varmennuksen vaatimukset[52]

---

[45] V1: Architecture, Design and Threat Modelling Requirements
[46] V2: Authentication Verification Requirements
[47] V3: Session Management Verification Requirements
[48] V4: Access Control Verification Requirements
[49] V5: Validation, Sanitization and Encoding Verification Requirements
[50] V6: Stored Cryptography Verification Requirements
[51] V7: Error Handling and Logging Verification Requirements
[52] V8: Data Protection Verification Requirements

- V9: Viestinnän varmennuksen vaatimukset[53]
- V10: Vihamielisen koodin varmennuksen vaatimukset[54]
- V11: Liiketoimintalogiikan varmennuksen vaatimukset[55]
- V12: Tiedostojen ja resurssien varmennuksen vaatimukset[56]
- V13: Rajapintojen ja verkkopalveluiden varmennuksen vaatimukset[57]
- V14: Konfiguraation varmennuksen vaatimukset[58]

3.6 OWASP Web Security Testing Guide

Jokaisen organisaation tulisi määritellä oma verkkoturvallisuuden testauskehys, joka kattaa kehityshankkeen kokonaisvaltaisesti alusta loppuun. Tämä on tärkeää ymmärtää: testaustyö alkaa jo määrittely- ja suunnitteluvaiheessa eikä vasta kehityksen aikana.

Ennen kehitysvaiheen alkua WSTG kehottaa määrittelemään

1. ohjelmistokehityksen elinkaaren
2. katselmoimaan käytännöt ja standardit
3. kehitä mittaustavat sekä varmistamaan tulosten jäljitettävyyden

3.6.1 Ohjelmiston elinkaaren määrittely

Ohjelmistohankkeen alussa, osana määrittelytyötä tulisi myös määritellä miten tietoturvan näkökulmat otetaan huomioon ohjelmiston elinkaaren eri vaiheissa. Tämä yksinkertaiselta kuulostava asia voisi olla oma tutkintotyön aiheensa, mutta pohjimmiltaan kyse on siitä, miten tietoturva rakennetaan vaihe vaiheelta osaksi paitsi ohjelmiston rakennetta myös osaksi ohjelmistosuunnittelun, -kehityksen, -testauksen ja tuotannon käytäntöjä. Jokainen vaihe rakentuu edellisen vaiheen luomalle perustalle alkaen siitä, että valitaan mitä kehitysmenetelmää noudatetaan (esim. Scrum, Kanban, eXtreme-ohjelmointi tai vesiputousmalli), mitä ohjelmistokehityksen ja tietoturvan käytäntöjä tulisi omaksua (tällä on suora liittymä tietoturvan kehityksen elinkaareen), miten varsinaista kehitystyötä tehdään, mitä testataan ja miten aina ylläpitovaiheen käytäntöihin.

---

[53] V9: Communications Verification Requirements
[54] V10: Malicious Code Verification Requirements
[55] V11: Business Logic Verification Requirements
[56] V12: File and Resources Verification Requirements
[57] V13: API and Web Service Verification Requirements
[58] Configuration Verification Requirements

Näihin kysymyksiin voi pyrkiä vastaamaan monista näkökulmista ja eri menetelmiä soveltaen, ja monet kehitettävän mallin erityispiirteet heijastelevat organisaation tarpeita, tapoja, arvoja ja käytäntöjä.

Lisätietoa asiasta löytyy mm. OWASP:n verkkosivuilta:
https://owasp.org/www-project-integration-standards/writeups/owasp_in_sdlc/

3.6.2 Käytäntöjen ja standardien katselmointi

Tässä on lähtökohtaisesti kyse samasta asiasta kuin Microsoft SDL:n käytännöissä #1 (tarjoa koulutusta) ja #2 (määrittely tietoturvavaatimukset). Heti hankkeen alusta lähtien tulisi olla selvää mitä käytäntöjä tulisi noudattaa sekä mistä asianmukainen ohjeistus on saatavilla. WSTG korostaa, että ihmiset voivat tehdä oikeita asioita vain, jos he tietävät mitä oikeat asiat ovat.[59] (OWASP WSTG 4.1, 34-35).

Mitä tämä käytännössä tarkoittaa, siihen valitettavasti ei ole yksinkertaista vastausta: riippuu hankkeesta ja organisaatiosta, muiden asioiden ohella. Esimerkiksi Scrum-tiimin tulisi määritellä hankkeen alussa *Definition of Done* eli mitä käytännössä tarkoitetaan sillä, kun sanotaan, että työ on valmis – ei vain valmis testattavaksi vaan valmis tuotantoon. Tämä ohjeistus määrittelee olennaisilta osin jokaisen käyttäjätarinan toteutuksen sisällön. Toisena esimerkkinä voisi olla valitun ohjelmointikielen kanssa noudatettava ohjelmointistandardi (joillakin kielillä näitä voi olla useitakin) tai lähdekoodin staattisessa analyysissa käytettävä säännöstö. Ihannetilanteessa vähintäänkin osa tässä yhteydessä tunnistettavista standardeista ja käytännöistä olisi jo määritelty osana organisaation SDL-työtä.

3.6.3 Mittaustapojen kehitys ja tulosten jäljitettävyys

Jotta voidaan testata, on tehtävä selväksi mitä testataan ja miten. Vaatimukset ja toiminnallisuudet sekä muiden tietoturvaominaisuuksien testaaminen edellyttää, että ne ovat kvantitatiivisesti määriteltävissä ja mitattavissa. Jos vaatimuksena on, että ”päivitetyn järjestelmän tulee olla alkuperäistä parempi”, niin mitä se

---

[59] ”Ensure that there are appropriate policies, standards, and documentation in place. Documentation is extremely important as it gives development teams guidelines and policies that they can follow. People can only do the right thing if they know what the right thing is.”

tarkoittaa? Miten ”parempi” konkreettisesti mitataan ja todistetaan ellei sille määritellä kvalitatiivisia arvoja?

Kun tiedetään mitä mitataan, niin on päätettävä myös, miten tietoa kerätään ja analysoidaan eli päätetään ovat hankkeessa noudatettavat testikäytännöt. Mitä ja miten testataan kehityksen aikana ja miten suoritetaan toimitusta edeltävä loppu- / hyväksyntätestaus? Huomioidaanko testauksessa yleisiä tietoturvalistauksia, kuten esimerkiksi OWASP Top 10 -julkaisut? Hyödynnetäänkö hankkeessa staattisia ja/tai dynaamisia tietoturvatestauksen menetelmiä?

Jokaiselle hankkeelle tulisi kirjoittaa testaussuunnitelma, ja nämä ovat osa siinä käsiteltäviä asioita, jotka määrittelevät miten tietoturvatestausta toteutetaan käsillä olevassa hankkeessa.

4 SUUNNITTELU

Esitutkimusvaiheessa hanketta tarkastellaan ensisijaisesti liiketoiminnan näkökulmasta, kun taas vaatimusmäärittelyvaiheessa tavoitteena on määritellä järjestelmän käytön ja toiminnan kannalta merkittävät piirteet. Suunnitteluvaiheessa keskitytään ratkaisuarkkitehtuurin suunnitteluun ja kuvaamiseen ottamatta kuitenkaan kantaa toteutuksessa käytettäviin teknologioihin, ellei vaatimusmäärittely tätä nimenomaisesti edellytä. Toisinaan rajan veto voi kuitenkin olla vaikeaa, mutta olennaista on kuvata arkkitehtuuri tavalla, joka on informatiivinen sekä kiteyttää olennainen.

> I mean architect as used in the words 'architect of foreign policy'. I mean architect as in the creating of systemic, structural, and orderly principles to make something work – the thoughtful making of either artefact, or idea, or policy that informs because it is clear. (Wurman, 1996, 16)

Suunnitteluvaiheessa tyypillisesti täsmennetään aikaisemmin tunnistettuja ja kuvattuja käyttötapauksia yksityiskohdiltaan toteutusvaiheen edellyttämälle tasolle. Ei ole mitenkään poikkeavaa, että suunnitteluvaiheessa tunnistetaan ja määritellään uusia paitsi uusia käyttötapauksia, niin myös uusia vaatimuksia: edeltävissä vaiheissa määrittely- ja suunnittelutyötä tehdään käytössä olevan tiedon ja ymmärryksen perusteella ja kun hanke etenee ja ymmärrys kasvaa, niin samalla myös järjestelmää koskevat määritykset ja suunnitelmat tarkentuvat. Tämä sama periaate pätee myös kehitys- ja testausvaiheeseen sekä tuotantoon.

4.1 Sovelluskehityksen tietoturvaohje (VAHTI 1/2013)

> Suunnittelussa toteutetaan määrittelyn vaatimukset organisaation arkkitehtuurien ja standardien mukaisesti. Tässä vaiheessa on suunniteltava myös monia tietoturvaominaisuuksia, kuten tunnistautumismenetelmät ja salausratkaisut. Suunnittelu on tehtävä siten, että sovelluksen hyökkäyspinta-ala on mahdollisimman pieni. (VAHTI 1/2013, 48)

Ratkaisuarkkitehtuurin kuvaamisen on monia menetelmiä. Esimerkiksi ArchiMate[60] erinomaisesti kokonaisarkkitehtuurityöhön ja tiettyyn rajaan asti sillä voi kuvailla myös ratkaisuarkkitehtuuriakin, mutta se soveltuu huonosti yksityiskohtiin menevään mallinnustyöhön. Kun on tarve tuottaa

[60] https://pubs.opengroup.org/architecture/archimate3-doc/

yksityiskohtainen prosessikuvaus, BPMN[61] eli Business *Process Model and Notation* on yleensä soveltuvin menetelmä.

> The ArchiMate language provides a broad view of the Enterprise Architecture, including business processes, whereas BPMN is well adapted to detailed business process modelling. However, the ArchiMate and BPMN languages are two separate standards, providing no guideline or support for a combined usage. (Open Group, n.d.)

Vastaavasti, kun arkkitehtuurityössä on tarve kuvailla tekninen ratkaisuarkkitehtuuri yksityiskohtaisesti, UML eli *Unified Modeling Language* tarjoaa tehtävään monipuoliset keinot. Edelleen, ERD eli *Entity Relationship Diagram* on laajalti käytetty rakenteellinen menetelmä tietomallien ja -kantojen kuvaamiseksi.

Arkkitehtuurin suunnitteluun ja kuvaamiseen on olemassa useita menetelmiä edellä mainittujen lisäksi. Käytetyistä menetelmistä riippumatta organisaatiolla tulisi olla systemaattinen lähestymistapa arkkitehtuurityöhön ja käytettävät menetelmät tulisi aina valita kuvaustarpeen mukaisesti sen sijaan, että yritettäisiin kuvata kaikkea mahdollista yhdellä ainoalla menetelmällä; sellaisen työn lopputulos harvoin vastaa työn tarkoitusta.

4.1.1 Perustaso: Yleiset standardit (SNT-001)

> Arkkitehtuuria suunniteltaessa tulee suosia yleisesti käytössä olevia ja hyväksyttyjä standardeja, mikäli se on mahdollista. Näin varmistetaan se, että suunniteltava sovellus on helppo integroida muihin järjestelmiin. Lisäksi tunnetut ja yleisesti käytetyt ratkaisut ovat yleensä joutuneet tarkemman tietoturva-analyysin kohteeksi kuin itse toteutetut tai harvinaisemmat teknologiat. Lisäksi avointen standardien käyttämistä suositaan, koska niiden tietoturvaominaisuudet tunnetaan tyypillisesti suljettuja standardeja paremmin. (VAHTI 1/2013, 48-49)

Tätä asiaa on mahdollisesti sivuttu jo esitutkimuksen ja vaatimusmäärittelyn aikana, ehkä osana SDL prosessia eli tietoturvan kehityskaarta. Esimerkiksi tietoturvavaatimuksia määriteltäessä tai *Arkkitehtuurilinjauksen* (VTM-009)

---

[61] http://www.bpmn.org/

yhteydessä voidaan ottaa hyvinkin tarkasti kantaa siihen, millaiset tekniset ratkaisut ovat organisaation ja kehitettävän järjestelmän kannalta hyväksyttäviä.

Tässä yhteydessä on syytä korostaa, että erityisesti tietoturvan näkökulmasta sovellusarkkitehtuurissa tulisi aina lähtökohtaisesti hyödyntää avoimia standardeja sekä avoimen lähdekoodin ohjelmistokomponentteja suljettujen ja/tai kotikutoisten ratkaisujen sijaan. On toki mahdollista, että vaatimusten mukainen ratkaisuarkkitehtuuri edellyttää suljetun standardin tai kaupallisen tuotteen käyttämistä ja pääosa kehitettävän järjestelmän lähdekoodistakin tuotetaan tyypillisesti kehitystiimin toimesta; olennaista on käyttää asiantuntevaa harkintaa ja pyrkiä välttämään ainakin tietoturvan kannalta keskeisten toimintojen kohdalla kyseenalaisia ratkaisuja.

4.1.2 Perustaso: Tietoturvapäivitysten suunnittelu (SNT-002)

> Sovellus tulee suunnitella siten, että korjausten ja tietoturvapäivitysten asentaminen on mahdollisimman helppoa. (VAHTI 1/2013, 49)

Sovelluspäivityksiin on yksinkertaistaen kaksi lähestymistapaa: joko päivitetty sovellusversio korvaa vanhemman sovelluksen tai sovellus sisältää mekanismin ohjelmistopäivitysten asentamiseksi. Ensiksi mainittu tapa on melko yleinen, kun sovellus on kehitetty vain yhdelle organisaatiolle ja päivitykset tehdään osana järjestelmän ylläpitoa. Jälkimmäinen on suositeltavampi malli, kun kyseessä on laajalti käytössä oleva sovellus; jos sovellus on yhteydessä Internettiin, se voisi säännöllisesti tarkistaa tarjolla olevat ohjelmistopäivitykset ja automaattisesti pyytää käyttäjältä lupa päivityksen lataamiseen ja asentamiseen. Mitä helpompaa päivitysten asentaminen on käyttäjän tai ylläpidon kannalta, sitä todennäköisemmin se tulee myös asennettua.

Se, miten sovelluksen tai järjestelmän päivitettävyys käytännössä suunnitellaan, riippuu monista tapauskohtaisesti huomioon otettavista seikoista; yhtä yleispätevää ratkaisua tähän kysymykseen ei ole. Yleisesti voidaan kuitenkin sanoa, että modulaariset arkkitehtuurit ovat monoliittisia arkkitehtuureita helpommin päivitettävissä. Vastaavasti monitasoarkkitehtuurit ovat 1-tasoarkkitehtuureita joustavampia, toiminnallinen konfiguraatio pitäisi olla muokattavissa (esim. konfiguraatiotiedosto, komentoriviparametrit) ja kaikki

arkkitehtuuriin sisällytetyt kolmannen osapuolen kirjastoversiot tulisi myös olla konfiguroitavissa kovakoodauksen sijaan.

Kehitystiimin on hyvä muistaa, että vaikka järjestelmästä päivitettäisiin vain yhtä kirjastoa tai luokkaa, testauksessa on aina syytä huomioida mahdolliset heijastumat järjestelmän muihin toimintoihin. Toisin sanoen, etenkään tietoturvan näkökulmasta ei riitä, että testataan vain sitä mitä on päivitetty vaan kaikki testit tulisi aina suorittaa yksittäisenkin muutoksen jälkeen[62].

4.1.3 Perustaso: Suositellut ohjelmistot (SNT-003)

> Organisaation tulee perustaa ja ylläpitää listaa suositelluista ohjelmistokomponenteista, kirjastoista, sovelluskehyksistä jne., joita käytetään organisaation toteuttamissa sovelluksissa. Komponentit luokitellaan toiminnallisuuden mukaan. Komponenttien tietoturvatilannetta ja päivityksiä tulee seurata. (VAHTI 1/2013, 49)

Ohjelmistokehityksessä on tavallista, että kehittäjät käyttävät omaa harkintaansa ottaessaan käyttöön kolmannen osapuolen tuottamia kirjastoja ja sovelluskehyksiä (engl. framework), kun arvioivat niiden edistävän kehitystyötä. Lähtökohtaisesti voidaan sanoa, että kehittäjät yleensä tietävät mitä tekevät, mutta on organisaatioiden edun mukaista laatia velvoittava ohjeistus toiminnan tueksi.

Muutosten tekeminen kolmannen osapuolen avoimen lähdekoodin lisenssillä julkaisemaan komponenttiin on oma asiakokonaisuutensa, johon ei tässä yhteydessä puututa; tässä yhteydessä olettama on, että kolmannen osapuolen ohjelmistokomponentteja, kirjastoja ja sovelluskehityksiä käytetään as-is eli ilman muutoksia, mutta sisällyttäen ne kuitenkin osaksi kehittävän järjestelmän arkkitehtuuria.

Ensimmäinen asia, johon tulisi ohjeistuksessa kiinnittää huomiota, on kolmannen osapuolen määrittelemä lisenssi. Esimerkiksi, salliiko kyseinen lisenssi lähdekoodin linkityksen osaksi kaupallisesti lisensoitavaa tuotetta vai velvoittaako linkitys tekemään koko järjestelmän lähdekoodista julkista? Millä ehdoilla kolmannen osapuolen komponenttia on lupa levittää osana järjestelmän

[62] Testiautomaatio helpottaa, kun alkaa ahdistamaan eikä aikakaan enää oikein riittäisi.

asennuspakettia?[63] Jos kyseessä on kolmannen osapuolen kaupallisesti lisensoima komponentti, niin on erityisen tärkeää kiinnittää huomiota lisenssin ehtoihin, sillä ne voivat pahimmillaan tehdä oman tuotteen kaupallisen myynnin taloudellisesti kannattamattomaksi tai jopa suoraan estää sen.

Organisaation tulisi laatia kehitystiimille ohjeistukseksi lista lisensseistä, joiden mukaisten ohjelmistokomponenttien, kirjastojen ja sovelluskehysten käyttö on hankkeessa sallittua. Vastaavasti kehitystiimin on voitava pyydettäessä tuottaa ajantasainen lista kaikista kolmannen osapuolen tuotteista ja niiden lisensseistä. Tämä on erityisen tärkeää, sillä kolmannen osapuolen komponenttien riippuvuudet voivat olla yllättävänkin laajat. Hankkeen vaarantava lisenssi ei välttämättä ole kehitystiimin valitsemassa komponentissa vaan se voikin löytyä syvemmältä riippuvuushierarkiasta: järjestelmä on riippuvainen komponentista A, joka on riippuvainen komponentista B, joka puolestaan on riippuvainen komponenteista C, D ja E. Kehitystiimin on varmistettava, että näiden kaikkien lisenssit ovat hyväksyttyjen lisenssien listalla.

Tietoturvan näkökulmasta lisenssejäkin tärkeämpää on varmistua siitä, että kolmannen osapuolen kirjasto on turvallinen käyttää; ettei siinä ole tunnettuja tietoturva-aukkoja, bugeja tai muita järjestelmän tietoturvan kyseenalaistavia teknisiä ratkaisuja. Riippuvuuksien hallinnasta puhuttaessa on yleisesti ottaen suositeltavaa, että kehitystiimi käy läpi vähintäänkin kolmannen osapuolen kirjaston kehityshistorian (esim. onko projekti vielä aktiivinen eli tuotetaanko siihen säännöllisiä päivityksiä) sekä tarkistaa siinä olevat avoimet bugit – sikäli, kuin niistä on julkista tietoa tarjolla.

Suositeltavaa on, että organisaatio katselmoisi ohjelmistokehityksessä tarvittavat kolmannen osapuolen komponentit paitsi lisenssien ja kaupallisen käytettävyyden näkökulmasta, myös erityisesti tietoturvan näkökulmasta ja laatisi listan hyväksyttävistä komponenteista ja näiden versioista. Kyseistä listaa on ylläpidettävä säännöllisesti, jotta versiopäivitykset hyväksyttäisiin listalle mahdollisimman nopeasti. On myös erityisen tärkeää, että kehitystiimit voivat

[63] Tyypillisesti vähintäänkin komponentin lisenssin kuvaava tiedosto pitää sisällyttää osaksi levityspakettia.

esittää tarvitsemiaan kirjastoja katselmoitavaksi ilman tarpeettomia viivästyksiä, saati byrokratiaa.

4.1.4 Perustaso: Ulkoiset rajapinnat (SNT-004)

> Sovelluksen ulkoiset rajapinnat käydään läpi ja verrataan niitä organisaation arkkitehtuuri- ja sovelluskehitysohjeistukseen kirjattuihin periaatteisiin (TSK-001)[64]. (VAHTI 1/2013, 49)

Tarkastellaan vaikkapa käyttäjän sisäänkirjautumista suhteessa TSK-001 ohjeistuksen mukaiseen rajapintatoteutukseen.

- Verkkoliikenteen tulisi olla salattua, joten varmistetaan, että käytettynä protokollana on HTTPS eikä HTTP.
- Monikerroksinen suojaaminen, jos sisäänkirjautuminen tapahtuu client-sovelluksen kautta (esimerkki):
    - Client-sovellus varmistaa, että sisäänkirjautumistiedot ovat syntaktisesti oikein ennen kuin ne lähetetään palvelimelle. Syntaksivirheestä ilmoitetaan käyttäjälle.
    - Palvelin tarkistaa, että sisäänkirjautumiskutsu tulee hyväksytystä lähteestä. Jos ei, prosessointi päättyy välittömästi ilman paluuviestiä client-sovellukselle.
    - Palvelin tarkistaa, että sisäänkirjautumistiedot ovat syntaktisesti oikein. Palvelin antaa geneerisen ilmoituksen epäonnistuneesta sisäänkirjautumisesta paljastamatta yksityiskohtia.
    - Palvelin tekee tarvittavat toimenpiteet estääkseen mm. SQL-injektion ja muut vastaavat potentiaaliset uhat.
    - Palvelin tarkistaa, että käyttäjätunnus on olemassa ja salasanan tarkistussumma vastaa tietokantaan tallennettua tarkistussummaa

---

[64] Arkkitehtuuri- ja sovelluskehitysohjeistus (TSK-001): Organisaatiolla tulee olla arkkitehtuuri- ja sovelluskehitysohjeistus, jossa määritellään sovelluskehitystä ohjaavat periaatteet. Tyypillisiä periaatteita ovat:

- Turvalliset oletusarvot (secure default)
- Monikerroksinen suojaaminen (defence in depth)
- Heikoimman alueen suojaaminen (securing the weakest link)
- Virhetilanteiden käsittely tietoturvallisesti (secure failure)
- Matalimmat mahdolliset oikeudet (least privilege)
- Tehtävien eriyttäminen (separation of duties)
- Tietoturvamekanismien salassa pysymiseen ei saa luottaa (security by obscurity)

(VAHTI 1/2013, 38)

(salasanaa ei milloinkaan tallenneta selväkielisenä!). Jos vastaavuutta ei löydy, palvelin antaa geneerisen ilmoituksen epäonnistuneesta sisäänkirjautumisesta yksityiskohtia paljastamatta.

- Monitasoarkkitehtuurissa rajapinnan logiikka ei ole koskaan suoraan yhteydessä tietokantaan.

4.1.5 Perustaso: Tietoturvalliset suunnittelumallit (SNT-005)

> Sovelluksen arkkitehtuurin suunnittelussa tulee käyttää ns. turvallisia suunnittelumalleja (secure design pattern). Organisaation kehittämät sovellukset tulee luokitella yleisen arkkitehtuurin mukaan. (VAHTI 1/2013, 49)

Tietoturvalliset suunnittelumallit voidaan jakaa karkeasti arkkitehtuuri- ja suunnittelutasoille. Ensiksi mainittuun kuuluvat:

- **Distrustful Decomposition:** mallin perusideana on jakaa järjestelmän arkkitehtuurin useisiin toiminnallisuuksiin ja komponentteihin, jotka eivät luota toisiinsa. Toisin sanoen kaikki tieto ja käsiteltävä tieto tarkistetaan aina myös sisäisten rajapintojen läpi kulkiessa eikä vain ulkoisen rajapinnan yhteydessä.
- **Privilege Separation:** lähtökohtaisesti järjestelmän ajonaikaisilla prosesseilla tulisi olla minimaaliset käyttöoikeudet. Jokaiselle palvelinsovellukselle tulisi luoda oma, erillinen käyttäjä, jonka käyttö on estetty muilta palvelinsovelluksilta ja jolla on pääsy vain ja ainoastaan niihin järjestelmäresursseihin, jotka ovat sovelluksen käytön edellytyksenä.
- **Defer to Restricted Application or Area:** sikäli kuin sovelluksen käyttö edellyttää lisäoikeuksia, nämä toiminalliset osa-alueet tulisi eristää sovelluksen niistä osa-alueista, joiden käyttö on mahdollista alemmilla käyttöoikeuksilla.

Suunnittelutason tietoturvallisia suunnittelumalleja ovat:

- **Secure State Machine:** tietoturvasta vastaavat toiminnallisuudet tuli erottaa käyttäjätason toiminnallisuuksista esimerkiksi niin, että ne toteutetaan toisistaan erillisinä tilakoneina.
- **Secure Visitor:** tämä suunnittelumalli sopii tilanteisiin, jossa järjestelmän sisältämä data on jaettu hierarkkisesti tietoturvatasoihin ja noodeihin niin, että pääsy edellyttää tiettyä roolia tai käyttäjätunnista. Tässä mallissa tietojen käsittely edellyttää Visitor -luokan instantiointia käyttäjäroolin tai -tunnistuksen perusteella, ja luokan metodit huolehtivat siitä, että ainoastaan oikeuksia vastaavat noodit voidaan avata. Toisin sanoen, järjestelmän tietojen käsittely suoraan ilman sisäistä ”luotettua vierailijaa” on estetty.

Tietoturvallisten suunnittelumallien kuvausten lähde on *Architecture and Design Considerations for Secure Software*. Muita tyypillisiä suunnittelumalleja ovat mm. lokien keskitetty keräys ja käsittely, keskitetty virheidenhallinta ja Single Sign-On -ratkaisujen käyttö.

4.1.6 Perustaso: Hyökkäyspinta-ala (SNT-006)

> Sovelluksen hyökkäyspinta tulee tunnistaa. Hyökkäyspinnalla tarkoitetaan kaikkia järjestelmän toiminnallisuuksia, joissa osapuolet eivät voi täysin luottaa toisiinsa ja joita voidaan siten käyttää sovellusta vastaan hyökätessä. […] Tehty kuvaus muodostaa sovelluksen hyökkäyspinnan (attack surface). Sitä käytetään pohjana rajapintojen ja moduulien turvamekanismien suunnittelussa. (VAHTI 1/2013, 50)

Tiivistäen voidaan todeta, että järjestelmän jokainen rajapinta, jota voidaan kutsua järjestelmän ulkopuolelta, altistaa järjestelmän hyökkäyksille. Näitä ovat mm. käyttöliittymät, verkkorajapinnat sekä fyysiset rajapinnat eri muodoissaan. Uhka ei myöskään rajoitu pelkästään syötettä vastaanottaviin rajapintoihin vaan myös kyselyihin vastaaviin rajapintoihin.

Sovelluksen arkkitehtuurin on tunnistettava järjestelmän kaikki rajapinnat kaikilla tasoilla alkaen n-tasoarkkitehtuurin sisäisistä ja ulkoisista loogisista rajapinnoista, eri käyttäjäroolien käyttöliittymien operaatiot (esim. HTTP:n GET, PUT, POST, UPDATE, DELETE metodit) sekä fyysiset portit (esim. USB ja RJ-45). Kaikki, mikä mahdollistaa tiedonvaihdon järjestelmän ja ulkopuolisen tahon välillä. Tavoitteena estää tarpeettomien rajapintojen käyttö, minimoida pakollisten

rajapintojen määrä sekä huolehtia, että kaikki rajapintojen läpi tapahtuvat operaatiot suoritetaan alimmilla mahdollisilla käyttöoikeuksilla. Kaikki tunnistetut rajapinnat dokumentoidaan ja näiden turvamekanismit käydään läpi.

4.1.7 Perustaso: Arkkitehtuurin tietoturvavaatimukset (SNT-007)

> Sovelluksen arkkitehtuuri on analysoitava tunnettuja tietoturvavaatimuksia vasten. Vertaa tietoturvavaatimuksia sovelluksen hyökkäysrajapintaan ja sovelluksen arkkitehtuuriin sekä varmista, että kaikki vaatimukset toteutuvat arkkitehtuuritasolla. Paranna sovelluksen arkkitehtuuria ja suunniteltuja tietoturvamekanismeja, mikäli kaikki vaatimukset eivät täyty. (VAHTI 1/2013, 51)

Vaatimuksen sisältö melko yksitulkintainen. Sovelluksen tietoturvavaatimukset on määritelty vaatimusmäärittelyvaiheessa (kts. VTM-006, luku 3.3.6), joten suunnitteluvaiheessa on varmistettava, että arkkitehtuuri huomioi kyseiset vaatimukset. Myöhemmin seuraavissa testeissä on lopulta varmistettava, että käytännön kehitystyö on toteuttanut suunnitellun arkkitehtuurin sekä vaatimusmäärittelyn.

Mikäli arkkitehtuuri syystä tai toisesta ei huomioi vaatimusmäärittelyä, on arkkitehtuuri virheellinen ja se on korjattava ennen käytännön toteutustyön aloittamista. Muussa tapauksessa on lähes varmaa, että tuotantoon vietynä järjestelmä ei vastaa suunniteltua käyttötarkoitustaan. Tässä vaiheessa tehtävät muutostyöt voivat tulla hyvinkin kalliiksi tai pahimmillaan voidaan jopa todeta, että järjestelmän korjaaminen vaatimuksia vastaavaksi ei ole käytännössä mielekästä tai edes mahdollista.

4.1.8 Perustaso: Tietoturvamekanismien kattavuus (SNT-008)

> Suunnittelukatselmoinneissa on varmistettava, että yhteisesti hyväksytyt tietoturvaratkaisut (katso TSK-002[65]) ovat käytössä koko sovelluksessa. Arkkitehtuuria suunniteltaessa käydään läpi kaikki liittymät järjestelmien välillä ja tarkastetaan niiden tietoturvamekanismit. Analyysi tehdään sekä sisäisille että ulkoisille rajapinnoille. Analyysin tarkoituksena on varmistaa, että suunniteltuja tietoturvamekanismeja käytetään koko sovelluksen laajuisesti. (VAHTI 1/2013, 51)

Tietoturvamekanismien käyttö liittyy läheisesti järjestelmän hyökkäyspinta-alan (kts. SNT-006, 4.1.6) rajoittamiseen ja alttiina olevien rajapintojen suojaamiseen. Lisäksi on huomionarvoista, että esimerkiksi todentaminen, valtuuttaminen ja käyttäjän syötteen validointi olisi aiheellista kuvata toiminnalliseen määrittelyyn kuuluvissa käyttötapauksissa. Vastaavasti asiat kuten virheenkäsittely ja lokien keräys tulisi olla huomioituna osana tietoturva- ja/tai ei-toiminnallisia vaatimuksia.

4.1.9 Perustaso: Tunnistautumismenetelmä (SNT-009)

> Valitulla tunnistautumismenetelmällä on suuri vaikutus kokonaisuuden turvallisuuteen. Sovelluksen omistajan vastuulla on määritellä sovelluksen käsittelemän tiedon luokittelu, joten hänen vastuullaan on myös määritellä se, miten vahvaa tunnistautumista sovelluksessa käytetään. (VAHTI 1/2013, 51)

Yleisin menetelmä käyttäjän tunnistamiseen on käyttäjätunnuksesta ja salasanasta muodostuva yhdistelmä, mutta tämä ei kuitenkaan täytä vahvan tunnistautumisen ehtoja. Jotta tunnistautumismenetelmää voitaisiin pitää vahvana, on menetelmän huomioitava vähintään kaksi kolmesta kriteeristä:

1. Mitä käyttäjä tietää (esim. käyttäjätunnus ja salasana)
2. Mitä käyttäjällä on hallussaan (esim. avainlukulista, älypuhelin tai henkilökortti)

---

[65] Teknisten tietoturvaratkaisujen määritys (TSK-002): Organisaation tulee määritellä yhteisesti hyväksytyt tekniset tietoturvaratkaisut. Tietoturvaratkaisut liittyvät muun muassa seuraaviin osa-alueisiin:

- Todentaminen (authentication)
- Valtuuttaminen (authorization)
- Käyttäjän syötteen validointi (input validation)
- Sovelluksen tuotteen turvallinen enkoodaus (output encoding)
- Virheenkäsittely
- Lokien keräys

3. Mitä käyttäjä itsessään on eli biometrinen tunnistus (esim. sormenjälki, silmän iiris, kasvojen skannaus, DNA)

Tyypillisesti kaksivaiheisessa tunnistuksessa käytetään käyttäjätunnuksen ja salasanan lisäksi joko pankkitunnuksia, mobiilitunnistetta, tekstiviestillä välitettyjä kertakäyttöisiä tunnisteita tai erillistä sähköistä tunnistegeneraattoria (esim. Google Authenticator ja Microsoft Authenticator puhelinsovellukset tai RSA SecurID).

Käyttäjän vahva tunnistus on nykyään verrattain helppo toteuttaa, joten etenkin kaupallisissa ja julkishallinnon verkkopalveluissa se olisi hyvä ottaa käyttöön myös tavanomaisina miellettävissä järjestelmissä, sillä monilla ihmisillä on huono tapa käyttää samaa käyttäjätunnus ja salasana yhdistelmää useissa eri palveluissa.

4.1.10 Perustaso: Salasanavaatimusten konfigurointi (SNT-010)

> Mikäli sovelluksen omistaja määrittelee riittäväksi tunnistautumismenetelmäksi salasanan ja käyttäjätunnuksen, pitää sovellus toteuttaa siten, että ainakin seuraavat salasanan laatuvaatimukset ovat konfiguroitavissa: salasanan pituus, erikoismerkkien määrä, tunnuksen lukkiutuminen määräajaksi liian monen epäonnistuneen kirjautumisen jälkeen, salasanan vanhenemisaika sekä tunnuksen vanhenemisaika. (VAHTI 1/2013, 51-52)

Olettaen, että järjestelmä ei tallenna käyttäjien salasanoja selväkielisinä, näiden murtamiseen lukuisia keinoja, jotka ovat osaltaan muokanneet salasanojen muotoa ja pituutta koskevia ohjeistuksia. Pois lukien valistuneet arvaukset, olan yli urkkimiset ja sujuvasti puhuen huijaukset, kolme kenties tyypillisintä salasanoihin kohdistuvaa hyökkäystä variaatioineen ovat sanakirjahyökkäykset, ”brute force” -hyökkäykset sekä sateenkaarihyökkäykset.

Sanakirjahyökkäys perustuu aikaisempien hyökkäysten perusteella kerättyihin salasanoissa tyypillisesti käytettyihin sana- ja fraasilistoihin. Yksittäisten listojen pituudet voivat olla jopa miljoonia rivejä, mutta näiden automaattiseen läpikäynti

on varsin nopeaa. Erityisesti yksittäisiin, selväkielisiin sanoihin ja helppoihin merkkiyhdistelmiin perustuvat salasanat ovat haavoittuvaisia[66].

Sateenkaarihyökkäys on hyvin samankaltainen, mutta tässä lista koostuu selväkielisestä salasanasta sekä siitä tiivistefunktiolla (engl. hash function) laskettuun tarkistussummaan: tyypillinen tapa tallentaa salasana on tallentaa varsinaisen salasanan sijaan siitä laskettu tarkistussumma. Tietoturvan kannalta tarkistussummaa ei pitäisi koskaan laskea suoraan käyttäjän antamasta salasanasta, vaan järjestelmän tulisi ”suolata” salasana lisäämällä siihen lisämerkkejä ennen tarkistussumman laskemista: kun eri järjestelmät käyttävät ”suolauksessa” eri merkkijonoja, ei sateenkaarihyökkäykset ovat tehottomampia, koska kahden eri järjestelmän tarkistussummat eivät täsmää, vaikka molemmissa olisikin käytetty samaa salasanaa.

”Brute force” -hyökkäys on näistä kolmesta vaarallisin, sillä siinä hyökkääjä käy järjestelmällisesti läpi kaikki merkkiyhdistelmät, kunnes oikea salasana löytyy. Suojauduttaessa ”brute force” -hyökkäystä vastaan ensimmäinen tekijä on salasanan pituus: mitä pidempi salasana, sitä enemmän suuremmaksi kasvaa kombinaatioiden määrä.
Salasanan vahvuutta arvioitaessa huomiota kiinnitetään sekä salasanan pituuteen että kompleksisuuteen. NIST[67] suosittelee (800-63B Digital Identity Guidelines), että salasanan tulisi olla vähintään 8 merkkiä pitkä, mutta OWASP ASVS suosittelee vähintään 12 merkkiä. Mitä pidempi, sitä parempi.

---

[66] Iso-Britannian kansallisen kyberturvallisuuskeskuksen (NCSC) mukaan kymmenen yleisintä salasanaa ovat:

1. 123456
2. 123456789
3. qwerty
4. password
5. 111111
6. 12345678
7. abc123
8. 1234567
9. password1
10. 12345

[67] National Institute of Standards and Technology

Kun salasana suojataan laskemalla ja tallentamalla sen tiivistefunktio, on suositeltavaa käyttää joko SHA-256 tai SHA-512 algoritmeja[68]. Tiivistefunktioiden ongelma tietoturvan kannalta on kuitenkin siinä, että ne on optimoitu laskemaan tiiviste mahdollisimman nopeasti, mikä on ihanteellista ”brute force” -hyökkäyksen kannalta: mitä enemmän salasanoja voidaan käydä sekunnissa, sitä parempi.

Suositus käyttää salasanassa suuria ja pieniä kirjaimia, numeroita ja erikoismerkkejä perustuu kombinaatioiden määrän kasvattamiseen. Esimerkiksi, jos salasanan pituus on 4 merkkiä ja se sisältää pelkästään numeroita 0-9, mahdollisia kombinaatioita on 10^4 eli 10 000. Kun mukaan otetaan suomalaisen aakkosten 29 kirjainta, kombinaatioiden määrä on $(10+29)^4$ eli 2 313 441. Kun huomioidaan sekä isot ja pienet kirjaimet, kombinaatioita on jo $(10+29+29)^4$ eli 21 381 376. Kombinaatioiden määrä voi vaikuttaa suurelta, mutta keskiverto tietokoneellakin salasana löytyy parissa sekunnissa. Joten vaikka mahdollisten merkkien lisääminen osaltaan lisää kombinaatioiden määrää, tehokkainta on käyttää pidempää salasanaa.

Edellä olleen esimerkin mukaisesti kahdeksalla merkillä kombinaatioiden määrä on jo 68^8 eli 457 163 239 653 376. Tehokas pelikäyttöön tarkoitettu tietokone voi laskea suunnilleen 610 000 SHA256 tiivistettä sekunnissa, kun taas MD5 tiivisteitä lasketaan keskimäärin 52,1 miljoonaa sekunnissa. Näillä arvoilla kahdeksan merkkisen salasanan murtamiseen menisi SHA256 tiivisteenä enimmillään 23,8 vuotta[69]. Luku on kuitenkin sikäli harhaanjohtava, että läpikäytävien kombinaatioiden määrää voidaan karsia merkittävästi eri menetelmin. Käyttämällä vähintään 12 merkkisiä salasanoja suojaa salasanansa ”brute force” -murtoyrityksiltä pitkälle tulevaisuuteen.

Järjestelmän tietoturvan kannalta on kuitenkin suositeltavaa käyttää salasanojen suojaamiseen erityisesti tarkoitettuja algoritmeja, kuten BCRYPT ja SCRYPT. Nämä nostavat salasanan laskemisen ”hintaa” eli tekevät operaatiosta

---

[68] Aikaisemmin yleisesti käytetty tiivistefunktioita kuten MD5 ja SHA1 ei missään nimessä saa enää käyttää salasanojen tallentamiseen.

[69] Tai pari sekuntia, jos salasana sattuu löytymään heti kombinaatioiden alkupäästä eikä vasta loppupäästä.

hitaamman. Käyttäjä ei tätä viivettä havaitse kirjautuessaan järjestelmään, mutta viiveen kerrannaisvaikutus on huomattava, kun tavoitteena on käydä mahdollisimman suuri määrä kombinaatioita läpi mahdollisimman lyhyessä ajassa. Lisäksi nämä algoritmit skaalautuvat vastaamaan tietokoneiden suorituskyvyn kasvua.

4.1.11 Perustaso: Käyttöoikeustasot (SNT-011)

> Sovellukset tukevat yleensä useaa eri käyttöoikeustasoa (käyttäjä, ylläpitäjä jne.). Sovelluksen käyttäjille on myönnettävä vain sen tasoiset oikeudet, jotka ovat välttämättömiä heidän roolinsa kannalta (ns. least privilege). Myönnettyjen tunnusten on oltava henkilökohtaisia, ja yleiskäyttöisten tunnusten käyttö on estettävä tai kiellettävä. Sovelluksen tietoturvallisuuteen vaikuttaviin ominaisuuksiin tulee olla pääsy vain sovelluksen pääkäyttäjällä. Samoin sovellus tulee toteuttaa siten, että sen käynnistämät prosessit suoritetaan pienimmällä mahdollisella käyttöoikeustasolla. (VAHTI 1/2013, 52)

Järjestelmän edellyttäessä käyttäjien tunnistautumista, voidaan käyttäjätasoja tunnistaa vähimmilläänkin kolme: anonyymi käyttäjä, normaali käyttäjä sekä ylläpitäjä. Näistä anonyymi käyttäjä saa nähtäväkseen ainoastaan julkista tietoa, normaalille käyttäjälle näytetään sisäänkirjautumisen jälkeen ainoastaan hänelle kuuluvaa tietoa (esim. omat henkilötiedot), kun taas järjestelmän ylläpitäjällä on pääsy järjestelmän konfiguraatioon ja muihin hallinnan kannalta tarpeellisiin toimintoihin, mutta ei käyttäjätietoja, ellei sille ole perusteltuja syitä.

Kun käyttäjäroolien määrä kasvaa, tulisi käyttöoikeustasot perustua rooleihin[70] ja käyttäjäryhmiin sen sijaan, että käyttäjän rooli olisi jollain tavalla kovakoodattu käyttäjätilin tyyppiin. Kun valtuutus käsitellä kategorisoitua tietoa ja tähän liittyviä järjestelmän toiminnallisuuksia sidotaan rooliin käyttäjän sijaan, muuttuu käyttäjän oikeudet hänen rooliensa muuttuessa. Suuremmissa organisaatioissa käyttäjäoikeuksien hallinta helpottuu, kun roolit määritellään käyttäjäryhmittäin ja käyttäjät puolestaan liitetään ryhmiin sen mukaan mitä oikeuksia he tarvitsevat.

Yksi vaihtoehtoinen tapa hallita oikeuksia on sitoa oikeudet roolien sijaan vaikkapa käyttäjien, järjestelmän ja ympäristön attribuutteihin ja näiden

---

[70] Role-based access control (RBAC)

kombinaatioihin[71]. Esimerkiksi järjestelmä voi yhdistää tietoa useasta eri tietokannasta, mutta käyttäjällä on oikeus näistä vain yhden tietokannan sisältämiin tietoihin. Tässä tapauksessa järjestelmän suodatettava muihin tietokantoihin kuuluvat tiedot pois ennen kuin käyttäjä saa tiedot nähtäväkseen.

4.1.12 Perustaso: Luottamusrajat (SNT-012)

> Sovellusta suunniteltaessa tulee määritellä ns. luottamusrajat (trust boundaries). Kaikkea rajan toisella puolelta tulevaa dataa tulee käsitellä ei-luotettuna, ja luottorajan toisella puolella oleviin tarkastuksiin ei tule luottaa. Ei-luotettu data tulee validoida ja kanonisoida ennen käyttöä. Validointi tulee toteuttaa ns. white list -pohjaisesti, toisin sanoen vain erikseen sallittu syöte hyväksytään ja muut hylätään. Myös salausmenetelmiä ja sähköistä allekirjoitusta voidaan käyttää tiedon aitouden ja muuttumattomuuden varmistamiseksi. Samoin ulkoisille komponenteille luovutettava data tulee enkoodata ja kanonisoida ennen lähettämistä. Näin varmistutaan siitä, että luovutettava data on turvallista ulkoisen komponentin käytettäväksi. (VAHTI 1/2013, 52)

Yleistäen voidaan todeta, että luottamusrajat ovat myös hyökkäyspintoja (kts. SNT-006), mutta luottamusrajoissa tulee huomioida myös muita tekijöitä kuten käyttäjien tunnistus ja valtuutus, syötteen validointi, palautettujen tietojen enkoodaus sekä tiedon lähteet (kts. SNT-008 sekä siinä mainittu TSK-002 viittaus).

Käytännössä luottamusrajojen varmistaminen edellyttää monen osatekijän yhdistämistä: esimerkiksi, operaatio sallitaan, kun käyttäjä on tunnistettu ja tälle on myönnetty asianmukaiset käyttöoikeudet ja syöte on validointisääntöjen mukaista.

4.1.13 Perustaso: Salausratkaisut (SNT-013)

VAHTI 1/2013 määrittelee salausratkaisujen vaatimuksista tiivistäen seuraavaa:

- Järjestelmään tallennetaan vain se data, mikä on järjestelmän toiminnan kannalta tarvittavaa.
- Salausratkaisujen tulee olla tunnettuja, julkisia ja käyttötarkoitukseensa nähden vahvoiksi todettuja. Esimerkiksi SHA256 ja SHA512 tiivisteiden laskemiseen tai AES ja RSA julkisen avaimen kryptografiaan.

---

[71] Attribute-based access control (ABAC)

- Ainoastaan tunnettujen toteutusten (esim. OpenSSL) käyttö on sallittua. Luovat kotikutoiset ratkaisut ovat kategorisesti kielletty.
- Satunnaislukugeneraattoreiden on oltava kryptografisesti turvallisia ja niiden siemenarvojen tulee sisältää riittävästi entropiaa.
- Salasanat on aina ”suolattava” ja niistä saa tallentaa ainoastaan niistä lasketut adaptiivisella tiivistealgoritmilla (esim. bcrypt ja scrypt) lasketut tiivisteet.
- Kaikki julkisessa verkossa tapahtuva liikenne on salattava (mielellään myös sisäverkon liikenne).
- Käytetyt salausratkaisut on voitava vaihtaa toisiin.

VAHTI 1/2013 ohjeistuksen mukaan julkista tietoa ei tarvitse salata, mutta tietoturvan näkökulmasta on perusteltua kysyä, eikö hyökkääjän tehtävä helpotu, jos julkinen ja luottamuksellinen tieto on valmiiksi eroteltu toisistaan? Salaamalla myös julkinen tieto vaikeutetaan (vaikka vain marginaalisesti) hyökkääjän tavoitetta tunnistaa hyökkäyksen arvoinen tietosisältö.

4.1.14 Perustaso: Tuki- ja ylläpitoyhteydet (SNT-014)

> Mikäli sovelluksen tuki- tai ylläpitotoiminnallisuus vaatii toimittajan tai tukea toimittavan tahon pääsyn järjestelmään, on tämä otettava huomioon jo suunnitteluvaiheessa. Erityisesti kriittisiä ovat ulkopuolisten mahdollinen pääsy järjestelmän tietoihin ja mahdollisten etäyhteyksien suojaaminen. (VAHTI 1/2013, 53)

Esimerkki: järjestelmä sisältää turvaluokiteltua tietoa, joten vähintäänkin osa järjestelmästä on segmentoitu niin, että niihin ei ole suoraa yhteyttä ulkomaailmasta. Tuki- ja ylläpito julkiverkosta ei tässä tapauksessa ole mahdollista, joten jo suunnittelun aikana on otettava huomioon tukihenkilöstön tarvitsemat turvaluokitukset ja pääsyt suojattuihin tiloihin.

Joissakin tapauksissa tiedonkulku kahden eri turvaluokitellun verkkosegmentin voidaan katsoa tarpeelliseksi, mutta suoraa yhteyttä ei voida kuitenkaan sallia. Tällöin järjestelmä pitää mahdollisesti suunnitella niin, että kahden verkkosegmentin väliin rakennetaan DMZ (engl. demilitarized zone) jonka läpi kulkeva liikenne menee kahden datadiodin läpi (verkkoliikenne on mahdollista vain yhteen suuntaan). Tällaiset tavanomaisesta poikkeavat järjestelmät

edellyttävät tuki- ja ylläpitotoiminnallisuuksien kokonaisvaltaista, perusteellista ja huolellista suunnittelua.

4.1.15 Korotettu taso: Uhkamallinnuksen syventäminen (SNT-015)

> Vaatimusmäärittelyvaiheessa tehtyä uhkien mallintamista tulee syventää uudella tiedolla esimerkiksi valituista teknisistä ratkaisuista sekä sovelluksen toiminnallisuudesta. (VAHTI 1/2013, 53)

Vaatimusmäärittely- ja suunnitteluvaiheiden välissä voi suuremmissa hankkeissa kulua jopa kuukausia. Niinpä yleisesti ottaenkin aikaisemmin laadittujen uhkamallien ja riskikartoitusten uudelleen katselmointi sekä niihin mahdollisesti uuden ja tarkentuneen tiedon pohjalta tehtävät täsmennykset ja lisäykset on tietoturvan näkökulmasta suositeltavaa; ymmärrys järjestelmän teknisistä yksityiskohdista ja valinnoista muuttuu suunnitteluvaiheessa olennaisesti etenkin, kun sitä verrataan vaatimusmäärittelyvaiheessa muodostettuun ymmärrykseen. Tämä ymmärrys jalostuu ja tarkentuu edelleen myös kehitys- ja testausvaiheissa, joten katselmointeja, määrityksiä ja suunnitelmia on syytä tarkistaa aina, kun niiden käsittelemät asiat tulevat hankkeessa ajankohtaisiksi.

4.1.16 Korotettu taso: Monitasoarkkitehtuurit (SNT-016)

> Mikäli sovellus toteuttaa ns. monitasoarkkitehtuuria (n-tier architecture), tulee eri komponenttien, kuten sovelluspalvelin ja tietokanta, välinen tietoliikenne suunnitella ja dokumentoida kattavasti. Sovelluksen tulee mahdollistaa ainakin:
>
> - eri komponentit voidaan sijoittaa eri verkkosegmentteihin
> - sovelluspalvelimen ja tietokannan välisiin yhteyksiin tulee mahdollisuuksien mukaan käyttää alustojen tukemia nimettyjä tietolähteitä (data source)
> - komponenttien välisen tietoliikenteen salaaminen, tunnisteiden käyttö sekä molemminpuolinen tunnistaminen (esimerkiksi mutual TLS)
> - hallinta- ja valvontaliikenteen erottaminen muusta liikenteestä
>
> (VAHTI 1/2013, 53)

Monitasoarkkitehtuurin käyttö on tyypillinen arkkitehtuurimalli mm. Client-Server ja Service Oriented Architecture (SOA) mallien mukaisissa palveluissa. Se soveltuu myös toisen tyyppisiin järjestelmiin, mutta sopivin ratkaisu pitää aina arvioida mm. järjestelmän vaatimusmäärittelyn pohjalta.

4.1.17 Korkea taso: Vahva tunnistautuminen (SNT-017)

> Korkean tietoturvatason järjestelmän arkkitehtuurin tulee toteuttaa seuraavat vaatimukset: järjestelmän kirjautumiseen on käytettävä vahvaa tunnistautumista, jolloin tunnistautumiseen tarvitaan tietoa kahdesta eri lähteestä (esimerkiksi käyttäjätunnus/salasana sekä RSA-tunniste). Sovelluksen on käytettävä monitasoarkkitehtuuria, jossa arkkitehtuurin eri komponentit on sijoitettava eri palvelimille. (VAHTI 1/2013, 54)

Vahvaa tunnistautumista on käsitelty tunnistusmenetelmien SNT-009 (4.1.9) yhteydessä ja monitasoarkkitehtuuria vastaavasti SNT-016 (4.1.16) yhteydessä. Asiaan liittyy läheisesti myös salasanavaatimusten konfiguroitavuus SNT-010 (4.1.10) sekä luottamusrajat SNT-012 (4.1.12).

5 POHDINTA

Ohjelmistojen ja tietojärjestelmien tietoturva on monitahoinen kysymys. Se alkaa organisaation kulttuurista ja käytännöistä, ulottuu moniin periaatteellisiin ja konkreettisiin menetelmiin päättyen lopulta yksittäisiin teknisiin valintoihin. Kun lisäksi huomioidaan tietoturvaan vaikuttavat inhimilliset tekijät, kuten kehitystiimin yksittäisten jäsenien osaaminen, huolellisuus sekä henkinen (ja fyysinen) hyvinvointi, niin on selvää, että jokaisen hankkeen tietoturva oma, ainutlaatuinen kokonaisuutensa.

Yleisellä tasolla kaikki ohjelmistokehityshankkeet noudattavat samaa elinkaarta. Kysymykset, joihin elinkaaren eri vaiheissa pyritään vastaan ovat myös pääpiirteissään aina samat[72]:

- **Esitutkimus** vastaa kysymyksiin hankkeen kaupallisesta kannattavuudesta ja sen taustalla olevista liiketoiminnallisista tarpeista.
- **Vaatimusmäärittely** vastaa kysymyksiin järjestelmän käyttäjistä ja toiminnallisuuksista (toiminnallinen määrittely), järjestelmän toiminnan puitteista ja suorituskyvyn kriteereistä (ei-toiminnallinen määrittely) sekä järjestelmän kannalta olennaisista tietoturvavaatimuksista.
- **Suunnittelu** vastaa kysymykseen siitä, millainen ratkaisuarkkitehtuuri toteuttaisi järjestelmän vaatimukset.

Jokaisen organisaation, joka käsittelee merkittävässä määrin luottamuksellista tietoa, tulisi kehittää toimintakulttuuriaan sisäistäen tietoturvaa tukevia periaatteita ja toimintatapoja. Microsoftin tietoturvakehityksen elinkaari on yksi laajalti tunnettu tämän toiminnan edistämiseen pyrkivä toimintamalli, mutta ei ole lajissaan ainoa. Perimmäinen viisaus asiassa on, että tietoturva ei ole pelkästään tekninen kysymys vaan loppujen lopuksi kysymys on aina ihmisistä: työyhteisön toimintatavoista, yleissivistyksestä sekä osaamisesta. Ilman asianmukaista tietoturvaa tukevaa toimintakulttuuria organisaation oma sekä sen kehittämien tuotteiden ja palveluiden tietoturva on parhaimmillaankin kyseenalainen.

---

[72] Ohjelmiston elinkaareen kuuluvat myös toteutus, testaus, käyttöönotto, ylläpito sekä käytöstä poisto.

Huolellisen vaatimusmäärittelyn merkitystä hankkeen onnistumisen kannalta ei voi liiaksi korostaa: on ensiarvoisen tärkeää ymmärtää ketkä ovat järjestelmän käyttäjät ja mitkä ovat heidän tarpeensa, mitkä ovat järjestelmän toiminnan kannalta olennaiset kriteerit, millaisiin uhkiin järjestelmän suunnittelussa tulisi varautua ja mitä tietosisältöä on suojeltava sekä tietenkin miten kehitettävä järjestelmä ylipäätään tukee organisaation liiketoiminnan tavoitteiden toteutumista. Mitä suuremmasta hankkeesta on kyse, sitä vakavammat seuraukset huolimattomalla ja välinpitämättömällä suhtautumisella vaatimusmäärittelyyn todennäköisesti tulee olemaan. Kun on kiire ja painetta minimoida kustannuksia, niin houkutus hypätä vaatimusmäärittelyn yli suoraan suunnitteluun, voi olla suuri.

Suunnitteluvaiheessa tavoitteena on laatia ratkaisuarkkitehtuuri, joka vastaa kaikkiin vaatimuksiin ottamatta kuitenkaan kantaa teknisiin ratkaisuihin, sikäli kuin niitä ei ole määritelty ei-toiminnallisissa vaatimuksissa. Tässä vaiheessa tulisi mm. syventää ymmärrystä järjestelmän toiminnallisuuksista, tunnistaa keskeiset tietomallit, laatia arkkitehtuurin periaatteet kehitys- ja testaustyön tueksi sekä yleisesti ottaen tuottaa korkean tason näkemys kehitettävästä kokonaisuudesta, joka kattaa järjestelmän elinkaaren kehityksestä ylläpitoon ja käytöstä poistoon.

On ihmisiä, jotka mieltävät järjestelmän elinkaaren jaon edellä kuvattuihin vaiheisiin vesiputousmallin mukaiseksi, vanhentuneeksi tavaksi ajatella ohjelmistokehitystä; erityisesti ketterissä, iteratiivisissa ja inkrementaalisissa menetelmissä pyritään keventämään ja suoraviivaistamaan kehitystyötä. Tämän tutkintotyön argumentti on kuitenkin se, että vaatimusmäärittely ja korkean tason suunnittelu tulisi eriyttää varsinaisesta kehitys- ja testaustyöstä: vertauskuvainnollisesti asiaa voisi ajatella, että jos kyseessä tutkimusretkikunta, niin vaatimusmäärittely asettaa tutkimusmatkan tavoitteet ja suunnittelu laatii kartan, kun taas kehitys- ja testausvaiheessa retkikunnan vastuulla on löytää reitti tavoitteisiin käytössä olevan kartan avulla.

Asiallinen, kiihkoton ja huolellinen suhtautuminen vaatimusmäärittelyyn ja korkean tason ratkaisuarkkitehtuurin suunnitteluun on liiketoiminnan kannalta järkevä ja tulosvastuullinen toimintamalli, mikä tukee myös organisaation pitkän

tähtäimen tavoitteiden saavuttamista (sikäli kuin organisaatiossa panostetaan kokonaisarkkitehtuurityöhön). Samalla tavoin, kun suunnitteluvaiheessa ei oteta kantaa teknisen toteutuksen yksityiskohtiin, niin teknisen toteutuksen vaiheessa ei pitäisi olla huolta ylimalkaisesta vaatimusmäärittelystä eikä huonosti laaditusta arkkitehtuurin visiosta.

Mitä puolestaan tulee tietoturvaan, niin jokaisessa organisaatiossa ja kehityshankkeessa on ymmärtävä, että kyse on ohjelmisto- ja järjestelmäsuunnittelun kannalta keskeisestä, elintärkeästä osa-alueesta eikä mistään valinnaisesta lisäominaisuudesta. Tietoturva on huomioitava heti ensimmäisistä askeleista alkaen ja rakentaa osaksi kokonaisuutta sen sijaan, että se olisi jotain mikä lisättäisiin muun kehitystyön päätteeksi aikataulun ja budjetin salliessa; se olisi kuin maalaisi ruosteen päälle.

LÄHTEET

Application Security Verification Standard 4.0.2. 2020. OWASP.
https://github.com/OWASP/ASVS/raw/v4.0.2/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.2-en.pdf

Architecture and Design Considerations for Secure Software. 2011.Software Assurance (SwA) Community. Yhdysvallat.
https://pdfs.semanticscholar.org/00dc/f49511729f595e05f4852a9cdb62e81d4d22.pdf

Brown, H. 1972. An American Renaissance? A Scientist's View of What America Can Be. Bulletin of the Atomic Scientists. Science and Public Affairs vol 28, no 6 (June 1972), 7-12.
https://books.google.fi/books?id=mwsAAAAAMBAJ

FFIEC Information Technology Examination Handbook – Business Continuity Management. 2019. Federal Financial Institutions Examination Council. Yhdysvallat.
https://ithandbook.ffiec.gov/media/296178/ffiec_itbooklet_businesscontinuitymanagement_v3.pdf

FFIEC Information Technology Examination Handbook – Information Security. 2016. Federal Financial Institutions Examination Council. Yhdysvallat.
https://ithandbook.ffiec.gov/media/274793/ffiec_itbooklet_informationsecurity.pdf

Garland, J., Anthony, R. 2003. Large-Scale Software Architecture – A Practical Guide Using UML. 1. painos. Yhdysvallat. John Wiley & Sons Ltd.

ICT-varautumisen vaatimukset (VAHTI 2/2012). 2012. Valtionhallinnon tietoturvallisuuden johtoryhmä. Helsinki. Valtionvarainministeriö
https://www.suomidigi.fi/sites/default/files/2020-07/2012_VAHTI_ohje_ICT_varautuminen_vaatimukset.pdf

McDermott, R.E., Mikulak, R.J., Beauregard, M.R. 2009. The Basics of FMEA. 2. painos. Yhdysvallat: Productivity Press.

Microsoft. N.d. What are the Microsoft SDL practices? Tulostettu 20.4.2020.
https://www.microsoft.com/en-us/securityengineering/sdl/practices

NATO Architecture Framework version 4. 2018. Architecture Capability Team. Consultation, Command & Control Board.

Ohje tietoturvallisuudesta valtionhallinnossa annetun asetuksen täytäntöönpanosta (VAHTI 2/2010). 2010. Valtionhallinnon tietoturvallisuuden johtoryhmä. Helsinki. Valtionvarainministeriö.
https://www.suomidigi.fi/sites/default/files/2020-06/pdf_2_2010.pdf

OMG Unified Modeling Language (OMG UML) version 2.5. 2015. Object Management Group. Yhdysvallat.
https://www.omg.org/spec/UML/2.5/PDF

Open Group. N.d. Combining BPMN and The ArchiMate standard for an enhanced BPM modeling support. Tulostettu 22.10.2020.
https://publications.opengroup.org/d236

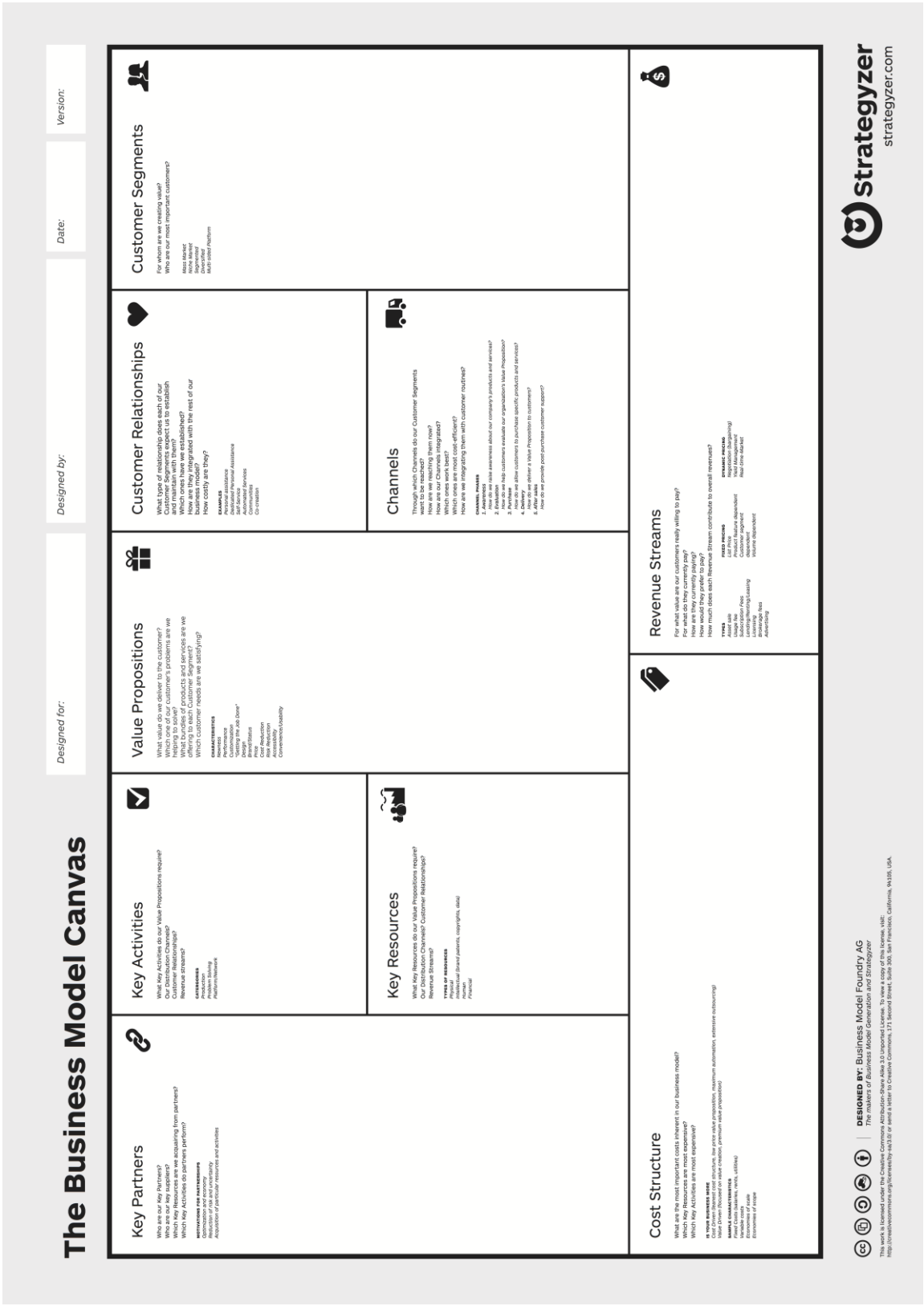
Sovelluskehityksen tietoturvaohje (VAHTI 1/2013). 2013. Valtionhallinnon tietoturvallisuuden johtoryhmä. Helsinki: Valtionvarainministeriö
https://www.suomidigi.fi/sites/default/files/2020-06/Vahti_ohje_1_2013_pdf_0.pdf

Web Security Testing Guide 4.1. 2020. OWASP.
https://github.com/OWASP/wstg/releases/download/v4.1/wstg-v4.1.pdf

Wurman, R.S. 1996. Information Architects. 1. painos. Yhdysvallat. Graphics, Inc.

LIITTEET

Liite 1. Business Model Canvas



Business Model Foundry AG

Creative Commons Attribution-Share Alike 3.0 Unported License.

Liite 2. Tyypillisen käyttötapauksen rakenne

Nimi	Toimintaa kuvaava nimi, esimerkiksi ”kirjaudu sisään”
Tunniste	Yksilöllinen tunniste, joka toimii muuttumattomana viitteenä. Tämä ei sinällään kuuluu käyttötapauksen kuvaukseen, mutta sen käyttö helpottaa elämää hankkeen edetessä.
Toimija (actor)	Yksi tai useampi käyttäjärooli / -tyyppi, joka voi suorittaa tämän toiminnallisuuden. Esimerkiksi ”anonyymi” (käyttäjää ei ole tunnistettu), ”käyttäjä” tai ”ylläpitäjä”.
Kuvaus	Lyhyt, vapaamuotoinen kuvaus toiminnallisuudesta.
Prioriteetti	Esimerkiksi ”matala – normaali – korkea”
Alkutila (pre-conditions)	Ennakkoehdot toiminnallisuuden suorittamiseksi. Esimerkiksi ”käyttäjää ei ole tunnistettu”.
Lopputila (post-conditions)	Järjestelmän tila toiminnallisuuden päätteeksi. Esimerkiksi ”Käyttäjäsessio on luotu”.
Peruspolku (basic path, happy day -scenario)	Vaihe vaiheelta etenevä kuvaus käyttäjän ja järjestelmän välisestä vuorovaikutuksesta, joka johtaa alkutilasta lopputilaan. Esimerkiksi: 1. Käyttäjä haluaa kirjautua järjestelmään 2. Käyttäjälle avautuu sisäänkirjautumislomake 3. Käyttäjä syöttää käyttäjätunnuksen ja salasanan 4. Käyttäjä painaa nappia ”Kirjaudu” 5. Järjestelmä tarkistaa käyttäjän kirjautumistiedot 6. Sisäänkirjautuminen hyväksytään
Vaihtoehtoiset polut (alternative paths)	Peruspolussa on usein kohtia, joissa käyttäjällä on mahdollisuus toimia vaihtoehtoisilla hyväksyttävillä tavoilla. Esimerkiksi: 2a. Käyttäjä sulkee lomakkeen 4a. Käyttäjä painaa nappia ”Peruuta”
Virhepolut (exception paths)	Käyttäjä tai järjestelmä toimii tavalla, joka on virheellinen. Esimerkiksi: e4.1 Käyttäjätunnus puuttuu e4.2 Salasana puuttuu e4.3 Käyttäjätunnusta ei ole järjestelmässä e4.4 Salasana on virheellinen

Liite 3. RFC 2119 – RFC Key Words

Network Working Group
Request for Comments: 2119
BCP: 14
Category: Best Current Practice

S. Bradner
Harvard University

March 1997

Key words for use in RFCs to Indicate Requirement Levels

Status of this Memo

This document specifies an Internet Best Current Practices for the Internet Community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Abstract

In many standards track documents several words are used to signify the requirements in the specification. These words are often capitalized. This document defines these words as they should be interpreted in IETF documents. Authors who follow these guidelines should incorporate this phrase near the beginning of their document:

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Note that the force of these words is modified by the requirement level of the document in which they are used.

1. MUST This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

2. MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

3. SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

6. Guidance in the use of these Imperatives

Imperatives of the type defined in this memo must be used with care and sparingly. In particular, they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm (e.g., limiting retransmisssions) For example, they must not be used to try to impose a particular method on implementors where the method is not required for interoperability.

7. Security Considerations

These terms are frequently used to specify behavior with security implications. The effects on security of not implementing a MUST or SHOULD, or doing something the specification says MUST NOT or SHOULD NOT be done may be very subtle. Document authors should take the time to elaborate the security implications of not following recommendations or requirements as most implementors will not have had the benefit of the experience and discussion that produced the specification.

8. Acknowledgments

The definitions of these terms are an amalgam of definitions taken from a number of RFCs. In addition, suggestions have been incorporated from a number of people including Robert Ullmann, Thomas Narten, Neal McBurnett, and Robert Elz.