jamk.fi

Tietoturvan ennakointi tuotekehityksessä

Case: Wimma Lab

Timo Kannus

Opinnäytetyö
Marraskuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Kannus, Timo	Opinnäytetyö, AMK	Marraskuu, 2020
	Sivumäärä 27	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Tietoturvan ennakointi tuotekehityksessä
Case: Wimma Lab

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Immonen Jani, Mieskolainen Matti

Toimeksiantaja(t)
Rintamäki Marko

Tiivistelmä

Toimeksiantona oli selvittää, kuinka tietoturvaa voidaan ennakoida paremmin tuotekehityksessä. Tavoitteena oli löytää Wimma Labille selkeä prosessi, kuinka sovelluskehityksessä käytettävien teknologioiden tietoturvallisuudesta voidaan varmistua. Ensimmäisenä selvitettiin mitä kanavia pitkin kehittäjät löytävät tietoa teknologioiden ja ohjelmointikielien mahdollisista haavoittuvuuksista. Todettiin, että internetistä löytyy tietoa, mutta tieto on pirstaleista ja lähteitä joudutaan toisinaan vertaamaan ristiin selkeyden saamiseksi.

Tutkimustyö tehtiin lähinnä julkisia lähteitä käyttämällä. Tutkimustyön ohella avattiin hieman tietoturvan hyvien käytäntöjen mukaista sovelluskehitystä ja asioita, joita tulee ottaa huomioon jo projektin alkuvaiheessa. Haasteena havaittiin tutkimustyön hajanaisuus ajallisesti. Toimeksianto saatiin 2018 kesällä, jolloin tietoturvallinen ohjelmistokehitys oli vielä lapsenkengissään. Sittemmin turvalliseen kehittämiseen on herätty ja tietoturva otettu oleelliseksi osaksi sovelluskehitystä jo suunnittelu vaiheessa.

Toimeksiannon toisena tavoitteena oli tutkia, onko ns. ”Tietoturva Oraakkeli” mahdollista toteuttaa millään tasolla. Asiaan liittyen tehtiin muutamia haastatteluja ja todettiin, että kyseinen Tietoturva Oraakkeli ei ole nykyteknologialla mahdollista toteuttaa, eikä sen toteuttaminen ole taloudellisesti, eikä ajallisesti mahdollista. Varsinaisen toteutuksen sijaan, tapailtiin hieman millainen Tietoturva Oraakkeli voisi mahdollisesti olla. Lopulta aikamääre oli täyttymässä, joten tulokset jäivät hieman vaillinaiseksi.

Avainsanat (asiasanat)
Tietoturva, sovelluskehitys

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Kannus, Timo	Bachelor's thesis	November, 2020
		Language of publication: Finnish
	Number of pages: 27	Permission for web publication: x

Title of publication

Security foresight in software development

Case: Wimma Lab

Degree programme

Information- and Communication Technology

Supervisor(s)

Immonen Jani, Mieskolainen Matti

Assigned by

Rintamäki Marko

Abstract

The task was to find out how information security can be better anticipated in product development. The aim was to find a clear process for WIMMA Lab on how to ensure the information security of the technologies used in application development. The first task was to find out through which channels developers can find information about potential vulnerabilities in technologies and programming languages. It was discovered that information can be found on the Internet; however, the information is fragmented, and the sources sometimes have to be cross-checked for clarity.

The research was conducted mainly using public sources. In addition to the research work, application development in accordance with good information security practices and issues that need to be considered at the beginning of the project were clarified. The fragmentation of the research work over time was identified as a challenge. The assignment was received in the summer of 2018, when secure software development was still in its infancy. Since then, secure development has been improved and information security has become an integral part of application development at the design and planning stages.

The second objective of the assignment was to examine whether it is possible to implement the so-called "Security Oracle" at any level. In this regard, a few interviews were conducted, and it was stated that the Information Security Oracle in question is not feasible with the current technology, and its implementation is not economically or temporally feasible. Instead of the actual implementation, the Oracle was theorized to some extent. Eventually, the time limit was running out, so the results remained incomplete.

Keywords/tags (subjects)

Information security, software development

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

HTML	Hypertext Markup Language
CVE	Common Vulnerabilities and Exposures
DPIA	Data Protection Impact Assesment
NMAP	Network Mapper -sovellus
OWASP	Open Web Application Security Project
PIA	Privacy Impact Assesment
XML	Extensible Markup Language
XSS	Cross Site Scripting
XXE	XML External Entity Injection

1 Johdanto

1.1 Toimeksiantaja

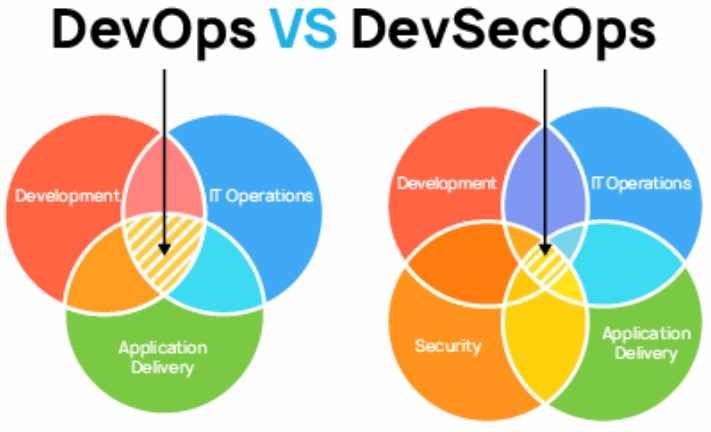
Vuotuisesti järjestettävä Wimma Lab on opiskelijoille räätälöity työelämäsimulaattori. Wimma Lab sai alkunsa vuonna 2011 nimellä Summer Factory. Vuonna 2014 nimeksi vaihdettiin Challenge Factory. Nimi Wimma Lab vakiintui vuonna 2017. Wimma Labissa opiskelijat rekrytoidaan virtuaalisiin yrityksiin ratkaisemaan oikeiden asiakkaiden oikeita ongelmia. Virtuaaliyrityksiä Wimma Labissa on neljä. MystiCons yrityksen tehtävänä on Wimma Labin sisäisten teknologisten ongelmien ratkaisu, sekä konsultaatio saatujen toimeksiantojen toteutustavoista. IoTitude ja Overflow yritykset ovat ohjelmistosuunnittelu painotteisia ja vastaavat tuotekehityksestä asiakkaan tarpeiden mukaan. Pengwin Media huolehtii Wimma Labin mediateknisistä tarpeista. Tähän kuuluvat muiden virtuaaliyritysten verkkosivut, käyntikortit sekä muu mainosmateriaali. Pengwin Median vastuulla on myös videoiden ja kuvien tuottaminen Wimma Labille.

Wimma Lab simuloi työelämää ohjelmistosuunnittelun parissa. Vuosittain useat suuret yritykset Jyväskylästä tarjoavat opiskelijoille mahdollisuuden ratkaista oikeita ongelmia joko tuotteen tai palvelun muodossa. Opiskelijat saavat kokemusta työstä ja asiakkaat tulevat tutuksi uusien kykyjen kanssa, täten saatu hyöty on molemminpuolista.

1.2 Tavoitteet

Opinnäytetyön toimeksianto saatiin Jyväskylän Ammattikorkeakoululta, Wimma Labin perustajalta Marko Rintamäeltä. Wimma Labissa huomattiin ohjelmistosuunnittelun tietoturvan laiminlyönti jo suunnitteluvaiheessa. Kun tuotetta ja/tai palvelua aletaan suunnittelemaan asiakkaalle, huomattiin että tietoturva tulisi ottaa olennaiseksi osaksi jo projektin alkuvaiheessa. Toimeksiantona oli tutkia kuinka tietoturva tulisi hoitaa vaarantamatta tuotetta ja/tai palvelua kehityskaaren loppupuolella. Tutkitaan väylät, joista ohjelmistosuunnittelijat saavat tärkeää tietoa käyttämiensä työkalujen sekä käytänteidensä turvallisuudesta. Samalla tarkastellaan tietoturvaa ohjelmistosuunnittelussa yleisellä tasolla.

2 DevOps ja DevSecOps



Kuvio 1 DevOps ja DevSec Ops

2.1 DevOps

DevOps on käsite, jolla tarkoitetaan yhteistyötä kehittäjien ja DevOps insinöörien kesken. Kehittäjät ovat vastuussa ohjelmiston kehittämisestä ja DevOps insinöörit kehitysympäristöjen toimivuudesta. DevOps insinöörin keskeisiä osaamisalueita ovat

mm. versionhallita (esim. Git), jatkuva integraatio (CI eli Continuous Integration), konttiteknologiat (Docker tai Vagrant), kehitysympäristön automatisointi, pilvipalvelut (Google, Amazon), testaaminen sekä kommunikaatio. DevOps on voimakkaasti automatisoitua. Dynaamiset ja staattiset testit liitetään osaksi versionhallintaa. CI putkisto automatisoidaan käsittelemään kehitettävän sovelluksen versionhallintaa aina testattavista versioista tuotantoversioon saakka. (Nikiforova E. 2020.)

2.2 DevSecOps

Kuten DevOps, DevSecOps on käsite, jolla tarkoitetaan yhteistyötä kehittäjien ja DevOps insinöörien välillä. Käsitteen keskellä oleva Sec tarkoittaa, että yhteistyöhön liitetään mukaan tietoturva näkökulma. Tietoturva on kuitenkin asia, jota ei voida mitata mittareilla. Kuten DevOps, DevSecOps on voimakkaasti automatisoitua. CI putkeen liitetään turvallisuus tarkastuksia, joilla varmistetaan, että yleisimmät haavoittuvuudet huomataan ennen kuin ne pääsevät sovelluksen tuotantoversioon. (Nikiforova E. 2020.)

DevSecOps on osittain käytännön teknisiä toteutuksia, osittain se on tapa ajatella sovelluskehitystä. Tietoturvaa on vaikea mitata konkreettisesti. Tietoturva on sarja käytänteitä ja tapoja työskennellä. Uhkamallinnuksen avulla voidaan yrittää tunnistaa tapaa, jolla hyökkääjä voisi ajatella ja täten vahingoittaa tai väärinkäyttää järjestelmiä tai sovelluksia (Nikiforova E. 2020).

Tietoturvallinen sovelluskehittäminen, käytännön toteutus: Käyttämällä sovelluskehitykseen keskittyvää agnostista kehystä "Digital Security and Privacy Model" turvallisuuden, yksityisyyden ja luottamuksen varmistamiseksi digitaalisessa yhteiskunnassa organisaatiot voivat lähestyä DevOpsin turvallisuutta käytännöllisellä tavalla (Nikiforova E. 2020).

Yhdenmukaisuuden ja kehityksen välisen kuilun kurominen: Avain vaatimustenmukaisuuden ja kehityksen välisen kuilun poistamiseen on tunnistaa sovellettavat val-

vontatoimet, kääntää ne sopiviksi ohjelmistotoimenpiteiksi ja tunnistaa taivutuspisteet ohjelmiston elinkaaressa, jossa nämä kontrollit voidaan automatisoida ja mitata (Nikiforova E. 2020).

Mittaus, seuranta, raportointi ja toiminta: Pätevien ihmisten on jatkuvasti valvottava ohjelmistokehityksen ja toimituksen jälkeisiä tuloksia oikeaan aikaan, jotta DevSecOps onnistuu (Nikiforova E. 2020).

Yhteinen vastuu: Turvallisuus ei ole jotain konkreettista, jonka edistymistä ja panosta voitaisiin mitata. Jokaisella organisaation henkilöllä on oma turvallisuusvastuu, ja hänen on oltava tietoinen omasta panoksestaan organisaation turvallisuusasenteeseen (Nikiforova E. 2020).

3 Tietoturva ohjelmistosuunnittelussa

Tietoturva ohjelmistosuunnittelussa voidaan jakaa osa-alueisiin. Ohessa Traficomin suositukset asiakohtaisesti (kuvio 2).

Kuvio 2 Havainnekuva tietoturvan suunnittelusta

3.1 Henkilöstön sekä työtilojen turvallisuus

Jokaisella työpaikalla käytetään työkaluja. Oli kyse sitten ohjelmistosuunnittelusta tai vaikkapa talojen rakentamisesta. Työn sujuvuuden takaamiseksi ja vahinkojen välttämiseksi työkaluista ja henkilöstön kouluttamisesta huolehtiminen on tärkeää. Ohjelmistoja toteuttavat ihmiset. Tietokoneet ja niiden sisältämät työkalut eivät osaa itsenäisesti luoda mitään. Tietoturvahygieniasta huolehtiminen on kaikkien yrityksessä työskentelevien vastuulla. Jokaisella työntekijällä, joka työskentelee lähdekoodin tai, koontituotteiden parissa tulee olla oma tili ja salasana järjestelmiin. Tehdystä työstä tulee olla loki ja loki tulee säilyttää ympäristössä, jonka tietoturva on kunnossa. Viimeisimmät tietoturvapäivitykset tulee olla asennettuna ja levyjen tulee olla salattuja. (Traficom n.d.)

Tietoturva käytänteet olisi hyvä perustaa kansallisiin tietoturvastandardeihin. Näihin lukeutuvat myös julkaisun- ja konfiguraationhallinnan prosessit. Tietoturvaa ei pidä kohdentaa kehityksessä olevaan tuotteeseen, vaan yrityksen työtapoihin (Traficom n.d.).

3.2 Tietoturvavaatimukset ja uhkamallinnus

Kun aloitetaan uusi ohjelmistoprojekti, ohjelmiston odotettua toiminnallisuutta määritellään vaatimusmäärittelyssä. Vaatimukset ilmaistaan käyttötapauksina sekä toiminnallisuutta kuvaavissa listoissa. Vaatimusmäärittely on vaikea saada oikein projektin ensimmäisissä vaiheissa, sillä asiakkaan vaatimukset ja toiveet todennäköisesti muuttuvat projektin edetessä. Samaa periaatetta olisi hyvä noudattaa tietoturvan osalla. Turvallisuusvaatimukset on hyvä laatia samaan aikaan kun ohjelmistoa suunnitellaan. Turvallisuusvaatimusten määrittäminen on vielä vaikeampaa kuin toiminnallisen vaatimusmäärittelyn tekeminen. (Traficom n.d.)

Aivan kuten muutkin vaatimukset, turvallisuusvaatimukset voivat olla toiminnallisia tai ei-toiminnallisia. Esimerkkinä toiminnallisesta vaatimuksesta voisi olla, että käyttäjän on ensin kirjauduttava käyttäjätunnuksella ja salasanalla järjestelmään. Käyttä-

jänimen ja salasanan kohdalla voidaan vielä käyttää erillistä vaatimusta näiden monimutkaisuudesta. Enemmän erikoismerkkejä sekä tietty pituus lisäävät käyttäjänimen ja salasanan turvallisuutta ja pienentävät mahdollisuutta saada nämä murretuksi. Ei-toiminnallisesta vaatimuksesta esimerkkinä voisi olla, että sovellus tarkistaa kaikki verkosta saapuvat syötteet ja hylkää kaikki virheelliset pyynnöt. (Traficom n.d.)

Tietoturvavaatimuksia pohdittaessa on tiedostettava, millaisessa ympäristössä tuotetta käytetään ja kuinka sovellusta käytännössä käytetään. Tietoturvavaatimuksiin vaikuttavat monet seikat. Jos järjestelmä toteutetaan pilvipalveluna, ovat vaatimukset hieman erilaiset kuin esimerkiksi fyysisillä palvelimilla toimivilla järjestelmillä. Jos sovelluksessa on verkkoselainta käyttäviä komponentteja, ovat vaatimukset jälleen hieman toisenlaiset kuin puhtaasti omalla käyttöliittymällä toimivilla sovelluksilla. Kun järjestelmän eri osat on tunnistettu ja käyttötavat määritelty voidaan miettiä mikä voisi mennä vikaan. Erilaisia skenaarioita voisi olla esimerkiksi, voiko järjestelmää käyttää väärin? Onko järjestelmästä mahdollista saada ulos tietoa, jota ei ole tarkoitettu ulkopuolisille? Voidaanko osia järjestelmästä korvata haitallisilla sovelluksilla takaisinmallinnusta käyttäen? Kaikki voidaan summata kysymykseen mitä seurauksia olisi, jos järjestelmän turvallisuus vaarantuisi? (Traficom n.d.)

Esimerkkinä verkkosovellusten turvallisuusvaatimuksista. Open Web Application Security Project (OWASP) määrittelee kymmenen tärkeintä uhkaa verkkosovelluksille. Kyseiset haavoittuvuudet pätevät myös muualla kuin verkkosovelluksissa.

Injektio: Ulkoiset syötteet jäävät tarkastamatta ja hyökkääjä voi suorittaa komentoja tai urkintaa järjestelmässä.
Rikkinäinen todennus: Käyttäjien todennus on jäänyt varmentamatta. Salasanat saattavat vuotaa. Istunnonhallinta on rikki, jolloin istuntoja voidaan kaapata.
Arkaluontoisten tietojen paljastuminen: Tietoja säilytetään selkokielisenä tai salaus on heikko.
Ulkoiset XML-entiteetit (XXE): Sovelluksen XML-käsittely ei ole turvallista.
Rikkinäinen käytönvalvonta: Käyttäjillä on oikeuksia suorittaa toimintoja järjestelmässä, joihin heillä ei pitäisi olla valtuuksia.
Turvallisuusasetukset väärin: Järjestelmän kovennusta ei ole tehty. Alusta voi olla

vanhentunut ja se saattaa sisältää tarpeettomia palveluja. Järjestelmässä saattaa olla myös paikallisia tilejä, jotka ovat turvattomia.

XSS- (Cross Site Scripting) haavoittuvuus: Hyökkääjän on mahdollista suorittaa haitallista HTML- (Hypertext Markup Language) ja/tai JavaScript koodia.

Sarjallistettujen tietojen turvaton lukeminen (insecure deserialisation): Sarjallistetut tiedot ovat hyökkääjien käsiteltävissä ja täten valtuudet vaativat tiedot tai oliot ovat käytettävissä.

Komponenttien käyttö järjestelmässä joissa on tunnettuja haavoittuvuuksia: Tahallinen tai tahaton haavoittuvuuksia sisältävien komponenttien käyttö järjestelmässä.

Lokiinkirjaus ja seuranta riittämätöntä: Sovellus ei seuraa mahdollisia hyökkäyksiä, eikä varoita niistä. Lokiinkirjauksen puuttuminen estää vikojen diagnosoinnin. (OWASP Top ten n.d.)

Turvallisuusvaatimuksia käsiteltäessä mainittiin jo joitain uhkamallinnuksen keskeisiä periaatteita. Uhkamallinnus tarkoittaa järjestelmän turvallisuuden arviointia ja sitä voidaan toteuttaa monilla eri menetelmillä. Uhkamallinnuksessa yritetään ajatella kuten hyökkääjä. Sovelluksella voi olla monenlaisia uhkamallinnuksia ja usein näkemyseroja tiimien välillä onkin. Uhkamallissa otetaan huomioon tuotteen todelliset käyttötilanteet ja ominaisuudet. Millaisia seurauksia hyökkäyksellä olisi esim. asiakkaalle? Millaisia huijauksia vastaan asiakkaan tulisi suojautua? Jotta uhkamallinnus voidaan toteuttaa, tulee järjestelmässä olla korkean tason arkkitehtuuri eri rooleineen ja turvallisuusominaisuuksineen. Turvatasoja eri tietojen käsittelyyn tulisi olla useita. (Traficom n.d.)

Kun tietoturva sisällytetään kehitysprojektiin alusta saakka, saadaan kehitettyä tuote, jonka turvallisuus on kiinteä ominaisuus. Tällä tarkoitetaan sisäänrakennettua turvallisuutta. Tietoturva voidaan lisätä liitännäisenä, jos sovelluksessa havaitaan haavoittuvuuksia tai vikoja jälkikäteen. Liitännäisiä turvallisuustekijöitä voivat olla esimerkiksi suojauskomponentit tai -ominaisuudet. Sisäänrakennettu turvallisuus on aina parempi vaihtoehto. Liitännäisten turvallisuustoimien riskinä ovat integraatio-ongelmat, haavoittuvuuksien suurempi riski sekä lisäkustannukset. Liitännäisiä lisäämällä myös tuotteen käytettävyys ja suorituskyky saattaa heikentyä. Testattavuus vaikeutuu ja yleisesti ottaen tuotteen laatu heikkenee. (Traficom n.d.)

Kun tietoturva on integroitu järjestelmään jo suunnitteluvaiheessa, turvallisuusarviointien sekä turvallisuuspäivitysten suorittaminen helpottuu. Liitännäiset turvallisuus toimet kuten palomuurit, monitorointi ja sovellusten hiekkalaatikointi (sandboxing) toki lisäävät tuotteen turvaa, mutta näitä tulisi käyttää vain hätävarana siinä tapauksessa, jos turvallisuus ei ole ollut oleellinen osa projektin suunnittelua. (Traficom n.d.)

Uhkamallinnus tulisi suorittaa työpajatyyppisesti, työpajaan osallistuvat kaikki järjestelmän kehitykseen osallistuvat osapuolet. Mikäli järjestelmässä on integraatioita pilvipalveluihin, myös tältä osa-alueelta on hyvä saada edustaja paikalle jakamaan näkemyksiään. Järjestelmästä voidaan piirtää kaavio, jossa on näkyvillä kaikki järjestelmän komponentit ja liitännäisyydet. Kuvan laatimisen jälkeen mietitään järjestelmää hyökkääjän näkökulmasta. Microsoftin kehittämä STRIDE-menetelmä on järkevä tapa kuvata uhkia. STRIDE-menetelmä alla. (Turvallisen sovelluskehityksen käsikirja. 2020.)

- Spoofing eli todennukseen liittyvät uhat.
- Tampering eli eheyteen liittyvät uhat.
- Repudiation eli kiistettävyyteen liittyvät uhat.
- Information disclosure eli luottamuksellisuuteen liittyvät uhat.
- Denial of service eli palvelunesto ja saatavuus uhat.
- Elevation of privilege eli käyttövaltuuksien ylittämiseen liittyvät uhat.

Mikäli uhkamallinnusta tehdään myös tietosuojan osalta, voidaan STRIDE-menetelmään tehdä TRIM-lisäys. TRIM selitys alla. (Turvallisen sovelluskehityksen käsikirja. 2020.)

- Transfer of personal data, eli siirrelläänkö henkilötietoja maan, organisaation tai sopimusteknisen rajan ylitse ja onko tähän oikeutus ja lupa? Onko siirron kohde oma rekisterinpitäjänsä vai ainoastaan käsittelijä ja jos jälkimmäinen, onko se sopimuksellisesti sovittu?
- Retention / Removal, eli mikäli komponentti tallentaa henkilötietoja, onko määritelty aika- tai muu kriteeri, jonka perusteella tallennetut tiedot tuhotaan; miten teknisesti toteutetaan yksittäisen henkilön tietojen poisto tarvittaessa ja miten yksittäisen henkilön tietojen käsittely voidaan pyynnöstä keskeyttää.

- Inference, eli yhdisteleekö komponentti useista eri lähteistä tulevia henkilötietoja muodostaen niistä uusia henkilötietotyyppejä, joita tallennetaan tai välitetään edelleen? Muuttaako se tietoa, joka aiemmin ei ollut henkilötietoa, henkilötiedoksi yhdistämällä sitä muuhun henkilötietoon? Onko nämä uudet henkilötietotyypit otettu huomioon tietosuojavaikutusten arvioinnissa?

- Minimisation, eli Kun komponentti lähettää henkilötietoja eteenpäin, onko tämä henkilötietojen joukko teknisesti ottaen pienin mahdollinen, vai voisiko sitä vielä entisestään pienentää teknisen toiminnallisuuden kärsimättä?

TRIM ei korvaa PIA (privacy impact assesment) tai DPIA (data protection impact assesment) tietosuojavaikutusten arviointia, mutta sitä voidaan käyttää alimman tason turvaverkkona, jos edellä mainittuja arviointeja ei ole vielä tehty. (Turvallisen sovelluskehityksen käsikirja. 2020.)

3.3 Suunnittelu

Kun kehitysprojektin vaatimukset on saatu määriteltyä, siirrytään suunnitteluvaiheeseen. Suunnittelun tarkoitus on tuottaa vaatimuksissa määritelty toiminnallisuus järjestelmään. Suotavaa olisi, että suunnitteluvaiheessa toistetaan syklejä ja että vaatimuksiin, että suunnitteluvaiheisiin palataan useita kertoja. Ensimmäinen suunnittelu hyvin epätodennäköisesti tuottaa täydellisen järjestelmän. (Traficom n.d.)

Turvallinen suunnittelu sisältää joitain yleisiä sääntöjä, joita olisi suotavaa noudattaa. Tällaisia ovat esimerkiksi hyökkäyspinnan minimoiminen, oletusasetusten turvallisuus sekä syötteenkäsittely (Traficom n.d.).

Hyökkäyspinnoilla tarkoitetaan niitä järjestelmän osia, jotka ovat kosketuksissa ulkomaailmaan. Järjestelmään kohdistuvat hyökkäykset odotettavasti tulevat tätä kautta. Hyökkäyspinnat voivat olla fyysisiä tai verkon tai tiedostojen välityksellä tapahtuvaa liikennettä. Järjestelmän ei olennaisten osien ja liitännäisten poisto pienentää hyökkäyspinta-alaa. Käyttöjärjestelmistä voidaan ottaa pois käytöstä oletuspalveluja ja järjestelmän kriittisten osien etäkäyttöä tulisi arvioida. Pääkäyttäjät voidaan mahdollisesti sitoa käyttämään järjestelmää vain paikallisesti, kun etäkäyttö poistetaan, yksi hyökkäysvektori voidaan poistaa uhkamallista. Mitä pienemmäksi hyökkäyspinta-ala

saadaan, sitä pienempi virheiden mahdollisuus on ja järjestelmää on helpompi testata. (Traficom n.d.)

Oletusarvoisesti järjestelmät ovat asetustensa puolesta melko turvattomia. Järjestelmän käyttäjien ei tarvitse olla tietoturvan asiantuntijoita voidakseen käyttää järjestelmää turvallisesti, mutta usein turvallisuuden varmistaminen jää käyttäjille. Ensimmäinen hyvä käytänne onkin määritellä asetukset valmiiksi toimitusversioon ja varmistaa että asetuksia ei tarvitse muutella. Jos asetuksia voidaan määritellä uudestaan, on todennäköistä, että järjestelmän turvallisuus heikkenee. (Traficom n.d.)

Syötteenkäsittely on järjestelmän turvallisuuden kannalta olennainen osa. Jos syötteitä ei tarkasteta, voivat hyökkääjät suorittaa syötekenttien välityksellä haitallista koodia, jolla pahimmassa tapauksessa kaadetaan koko järjestelmä tai päästään käsiksi arkaluontoiseen tietoon. Erikoismerkkien karsiminen syötekentästä ja syötteen maksimipituus tulisi varmistaa. Syötetiedoilla siirrettävä tieto pitää suojata salauksella sekä eheystarkistuksilla. Huomioitavaa on myös rajapinnat, joihin syötteillä on epäsuora vaikutus. Hyökkääjän syöttämä syöte saattaa kulkeutua syvälle järjestelmään aiheuttaen vahinkoa. (Traficom n.d.)

3.4 Syväsuojaus

Järjestelmän turvallisuus ei saa nojata yhteen suojaustoimeen. Turvallisuus järjestelmässä tulee rakentaa suunnittelemalla useita suojaustasoja. Tällä varmistetaan järjestelmän kokonaisvaltainen turvallisuus siitäkin huolimatta, että yksi suojaustaso murretaan. Palomuuri yksistään ei riitä järjestelmän kokonaisvaltaiseen suojaamiseen. Palomuurit sisältävät sääntöjä, jotka taitava hyökkääjä pystyy kiertämään. Järjestelmän sisäisen liikenteen salauksella ja sisäisten palvelujen käyttöön vaadittavalla käyttäjän todennuksella saavutetaan kovennettu turvallisuus. Palomuurin ohittamisen onnistuminen ei takaa hyökkääjälle onnistumista edellä mainittujen toimien ollessa käytössä. (Traficom n.d.)

3.5 Vikatilanteiden turvallisuus

On olemassa monenlaisia vikoja, joita vastaan tulee varautua. Ohjelmistojen kaatumiset, verkko ongelmat, sähkökatkot yms. Vikatilanteen sattuessa järjestelmän turvallisuus voi vaarantua, ellei näihin vikoihin varauduta jo suunnitteluvaiheessa. Valittavana on kaksi eri strategiaa: avautuminen (fail-open) ja sulkeutuminen (fail-close). Avautumisessa järjestelmä sallii järjestelmän käytön, kun taas sulkeutumisessa järjestelmä lukittuu ja estää käytön. Strategian valintaan ei ole yksiselitteistä, helppoa tapaa. Vikatilanteessa avautuva järjestelmä vaarantuu, jos esimerkiksi autentikaatiosta vastaava komponentti ei toimi halutulla tavalla. Tässä riskinä on tietojen päätyminen ulkopuolisille ja osia järjestelmästä voidaan korvata haitallisilla komponenteilla. Sulkeutuminen ei ole välttämättä parempi vaihtoehto, jos käytön estyessä vika eskaloituu useampaan järjestelmään. Strategia tulee olla olemassa vikatilanteiden varalle joka tapauksessa. (Traficom n.d.)

3.6 Järjestelmän monimutkaisuus

Monimutkaiset järjestelmät ovat hankalia ylläpidettäviä. Koodissa havaittu virhe tai haavoittuvuus voi olla yllättävän vaikea korjata, jos havaittu vika tai haavoittuvuus on komponentissa, joka vaikuttaa suorasti tai epäsuorasti järjestelmän toisiin osiin. Tästä syystä suositeltavaa onkin pysytellä mahdollisimman yksinkertaisissa järjestelmissä ja suunnitella järjestelmä alusta asti sellaiseksi. Yksinkertainen järjestelmä on helpompi testata ja uudelleenfaktorointi huomattavasti kevyempi toteuttaa. Tietoturvan varmistaminen yksinkertaisessa järjestelmässä on helpompaa ja kustannustehokkaampaa. Monimutkaisten arkkitehtuurien suunnittelua tulisi aina tarkastella kriittisesti edellä mainituista syistä (Traficom n.d.)

Järjestelmän rakenteen tai lähdekoodin salassapidolla voidaan lisätä yksi turvallisuustaso lisää, mutta riskinä on, että järjestelmästä tulee tarpeettoman monimutkainen hallittava. Salaisuuksia vuodetaan ja takaisinmallintamalla järjestelmän ominaisuuksia salaisuudet paljastuvat. Vaikka järjestelmässä käsiteltävät tiedot ja tietueet ovatkin salaisia, tulisi järjestelmän itsessään kestää tarkastelua. Tietomurron sattuessa

suuri määrä salaisuuksia aiheuttaa ongelmia. Esimerkiksi kuinka helposti järjestelmän käyttäjiä saadaan vaihtamaan salasanansa, jos ne ovat vaarantuneet. (Traficom n.d.)

3.7 Turvallisuusongelmien korjaaminen

Havaittu turvallisuusongelma tulisi käsitellä pikaisesti. Haavoittuvuudet ja viat, jotka havaitaan tuskin aiheuttavat epävarmuutta, jos noudatetaan syklistä kehitys- ja testausprosessia. Riskinä on, että havaittuja virheitä jätetään korjaamatta ja keskitytään huolehtimaan siitä vain järjestelmän tietyssä osassa. Kyseinen virhe saattaa olla muuallakin järjestelmässä, tai se saattaa päätyä sinne useiden tuotekehitys syklien jälkeen. Syitä korjaamatta jättämiselle voi olla esimerkiksi pelko sivuvaikutuksista, tai järjestelmän toimivuudesta korjauksen jälkeen. Vikojen huomiotta jättäminen onkin yksi suurimmista ongelmista tuotekehityksessä. Kehitystiimi saattaa tyytyä ajattelemaan, että kaikki järjestelmät murretaan jossain vaiheessa, joten mittavia korjaustoimia ei kannata lähteä tekemään. Alustan valinnalla on myös roolinsa, johon paneudutaan enemmän tutkimusosiossa. (Traficom n.d.)

3.8 Järjestelmän testaus

Kaikki järjestelmään rakennetut turvallisuusominaisuudet sekä muut tärkeät ominaisuudet on testattava. Testauksella varmistetaan, että järjestelmä käyttäytyy suunnitteluvaiheessa määritetyllä tavalla. Manuaalisella testaamisella ei kyetä testaamaan kattavasti suurempia järjestelmiä, joten automaatiotestausta tarvitaan. Ohessa lyhyesti testauksen osa-alueet sekä vaaditut testauksen tasot. (Traficom n.d.)

Yksittäisen komponentin tai komponentin osan testausta kutsutaan yksikkötestaukseksi. Yksikkötestauksen tarkoitus on varmistaa, että komponentin osa toimii kuten sen kuuluukin. Puhuttaessa proseduraalisesta ohjelmoinnista, testattava osa voi olla yksittäinen ohjelma, funktio tai proseduuri. Olio-ohjelmoinnissa pienin testattava osanen voi olla esimerkiksi yksi metodi, joka kuuluu tietynlaiseen luokkaan. (Unit testing n.d.)

Komponenttitestaus tarkoittaa yksittäisen komponentin testausta. Komponentti testataan niin että komponentti ei ole kanssakäymisessä muiden komponenttien kanssa, tai osana järjestelmää. Tarvittaessa muut komponentit tai järjestelmän osa voidaan simuloida testin tuloksen saamiseksi. (Component testing n.d.)

Järjestelmätestaus on sarja testejä, jotka suoritetaan kokonaisessa sovelluksessa. Testauksen tarkoituksena on arvioida päästä päähän (end-to-end) määritelmät. Testauksella selvitetään kuinka sovelluksen komponentit toimivat keskenään, sekä kokonaisen järjestelmän sisällä. (System testing n.d.)

Hyväksymistestaus on kuin järjestelmätestausta, mutta testeille annetaan oletustulokset. Testien suorittamisen jälkeen arvioidaan erot oletettujen tulosten ja saatujen tulosten välillä. Testitulosten arvioinnin suorittaa aina ihminen, automaatioon ei voida luottaa. Kun testaus on suoritettu, arvioidaan, hyväksytäänkö poikkeama, jätetäänkö poikkeama huomiotta, vai hylätäänkö testi. (Approval testing 2018.)

Staattinen testaus tarkoittaa sovelluksessa olevien virheiden tarkistamista, ilman että sovelluksen koodia suoritetaan. Esimerkiksi koodikatselmoinnit ovat staattista testaamista. Staattinen testaus on ensiarvoisen tärkeää ohjelmistoprojektin alkuvaiheessa, jotta virheet saadaan korjattua ajoissa. (Static testing n.d.)

Dynaaminen testaus tarkoittaa sovelluksessa olevien virheiden tarkistamista suorittamalla sovelluksen koodia. Dynaamisella testauksella varmistetaan, että syötteet ja riippuvuudet toimivat kuten niiden on määritelty toimivan. Olennainen ero staattiseen testaamiseen on, että dynaamisella testauksella varmistetaan sovelluksen oikeanlainen toiminta ja käyttäytyminen, kun taas staattisella testauksella pyritään havaitsemaan virheet koodissa. (Dynamic testing n.d.)

Fuzz-testaus tarkoittaa sovelluksen mahdollisten tietoturvaongelmien havainnoimista. Fuzzer on ohjelma, joka syöttää näennäisen satunnaista dataa sovellukseen tai järjestelmään ja havainnoimaan tästä aiheutuvia virheitä. Syötettävä data voi olla tunnettuja vektoreita pitkin syötettävää, tunnettua vaarallista dataa. Data voi olla myös täysin satunnaista. (Fuzzing n.d.)

Penetraatiotestaus on koko järjestelmän tietoturvakypsyyden arvioimista. Penetraatiotestauksen suorittavat tietoturvallisuuden ammattilaiset. Testattaviin osa-alueisiin kuuluvat verkot, sovellukset sekä fyysiset laitteet. Penetraatiotestauksessa testaajat pyrkivät murtautumaan järjestelmään ja antavat palautetta, kuinka korjata löytyneet puutteet, sekä antavat palautetta kuinka erilaisiin hyökkäyksiin voi varautua ja kuinka ennalta ehkäistä hyökkäyksiä. (Talamantes J. n.d.)

Stressitestaus on lähes samaa testausta kuin kuormitustestaus, mutta tavoitteena on löytää suorituskyvyn rajat ja ylittää ne. Stressitestauksella varmistetaan vikatilanteiden turvallisuusvaatimukset ja syväsuojauksen periaatteet. Laitteen käynnistyminen, verkkoyhteyden katkeaminen ja vaikka sähkökatkon aiheuttama uudelleen käynnistyminen voivat synnyttää hyökkäysvektoreita, joita on vaikea ennakoida uhkamallinnusta tehdessä. Tästä syystä vikatilanteita on hyvä simuloida ja testata. (Traficom n.d.)

Takaisinmallinnus voi koskea ohjelmatiedostojen, laiteohjelmistojen ja verkkoliikenteen takaisinmallinnusta. Kohdassa järjestelmän monimutkaisuus, sivuttiin järjestelmän osien salassapitoa. Takaisinmallintamalla voidaan tämä salaus tehdä tyhjäksi. Takaisinmallinnukseen on tarjolla monia perustyökaluja kuten Wireshark (verkkoliikenteen nuuskimiseen), Nmap (isäntäkoneiden, avointen porttien ja palvelujen etsintään) sekä Strings (merkkijonojen tulostusta mistä tahansa tiedostosta). (Traficom n.d.)

4 Ongelman määrittely

4.1 Yleiset tietoturvahaasteet

Uutta sovellusprojektia aloittaessa, on suotavaa ajatella tietoturvakysymyksiä jo ennen kuin valitaan sovelluksen kehityksessä käytettävät teknologiat. Uhkamallinnus tulisi tehdä samaan aikaan kun sovelluksen käyttäjiä ja tarkoitusta suunnitellaan. Käyttäjätarinoita miettiessä on hyvä samalla miettiä kuinka sovellusta voisi käyttää

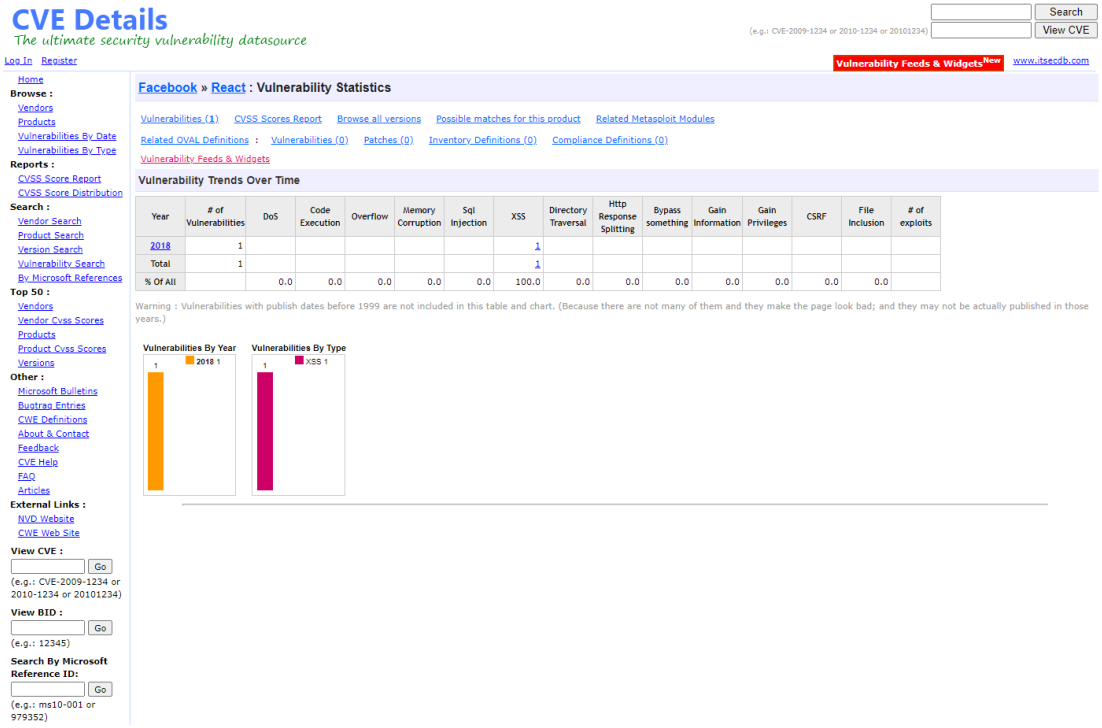
väärin, tahallaan tai vahingossa. Potentiaalisten uhkien tunnistaminen jo suunnitteluvaiheessa on ensiarvoisen tärkeää.

Ensimmäinen huomioitava seikka projektia aloittaessa onkin varmistua siitä, että kehittäjät, sekä kaikki projektin parissa työskentelevät ovat perillä projektin tietoturvallisesta toteutuksesta. Puhtaan pöydän periaate; työasemat lukittuina ja työpiste tyhjänä arkaluontoisesta materiaalista. Projektista ei puhuta sivullisille ja varmistutaan että projektista puhuessa, kukaan sivullinen ei vahingossa kuule keskustelua.

4.2 Tiedonkeruu tietoturvauhkista

Kehitykseen käytettävien teknologioiden valinnassa tulisi ottaa huomioon eri frameworkien, ohjelmointikielien sekä liitännäisten potentiaaliset haavoittuvuudet. Tämän tiedon kerääminen on työlästä. On olemassa kanavia, joista kehittäjä voi tarkastaa käyttämiensä teknologioiden haavoittuvuudet. Yksi tällainen kanava on cvedetails.com. CVE details listaa tunnettuja haavoittuvuuksia aina kun uusia löydetään. Käyttäjä voi hakea sivulta tietoa valmistajan/kehittäjän nimellä, tai teknologian/sovelluksen nimellä. Esimerkiksi jos halutaan tietää, onko React frameworkin käyttäminen turvallista, hakukenttään syötetään *React*, sivusto ehdottaa tuloksia ja yks tulok-

sista on *Facebook > React*. Kuviossa 3 nähdään että vuonna 2018 on löydetty haavoittuvuus, joka koskee XSS hyökkäystä.



Kuvio 3 CVE Details -sivuston löytämä haavoittuvuus.

Kuviossa 4 löydettyä haavoittuvuutta tarkastellaan lähemmin ja huomataan että CVE Details -sivusto on antanut uhkapisteet 4.3. Vakavuudeltaan löydetty haavoittuvuus

ei ole merkittävä, mutta silti huomionarvoinen.

Kuvio 4 CVE Details -sivuston tarkempi kuvaus tunnetusta haavoittuvuudesta.

Huomattavaa kuitenkin on, että haavoittuvuus on korjattu pienemmillä päivityksillä ja todennäköisesti kyseessä oleva haavoittuvuus ei ole enää uhka.

CVE Details on tällä hetkellä kattavin tietopankki tietoturvauhkista. Teknologioiden toimittajat ja tuotteiden kehittäjät pyrkivät pitämään tuotteensa päivitettyinä ja turvallisina, mutta eivät yleensä itse mainosta puutteita tuotteissaan. Tästä syystä kehittäjän tuleekin kyseenalaistaa toimittajan tai teknologian kehittäjän lupaukset tuotteidensa täydellisestä turvallisuudesta.

Ongelmaksi ohjelmistokehityksen tietoturvan ennakoinnissa on lähteiden hajanaisuus. Tietoa löytyy monesta paikasta, eikä aina ole selvää, onko tieto ajantasaista.

5 Tietoturvallinen kehitystyö projektissa

5.1 Huomioitavia seikkoja tietoturvallisesta sovelluskehityksestä

Projektin parissa työskentelevien työntekijöiden olisi hyvä pitää huolta työskentelytapojensa turvallisuudesta. Esimerkiksi koodari voi törmätä ongelmaan, johon löytyy ratkaisu valmiina koodina yksinkertaisella haulla internetissä. Muutama rivi koodia saattaa ratkaista välitöntä käsittelyä vaativan puutteen tai toimintahäiriön ohjelman toiminnassa, mutta pidemmällä tähtäimellä tämä saattaa aiheuttaa isompia ongelmia. Mitään sellaista, jota ei ymmärrä, ei pitäisi kopioida. Liitännäinen voi lisätä suurenkin osan sovellukseen halutusta toiminnallisuudesta, mutta jos liitännäistä ei tarkasteta etukäteen, voi se tulla kalliiksi projektin edetessä. Jos esimerkiksi kehitettävä sovellus saakin päivityksen, mutta liitännäinen ei, syntyy yhteensopivuusongelma joka pahimmillaan lisää sovelluksen hyökkäyspinta-alaa ja tällaisen tietoturvaongelman ratkaisu saattaa vaatia suuria refaktorointeja, joka on lähes poikkeuksetta kallista ja aikaa vievää.

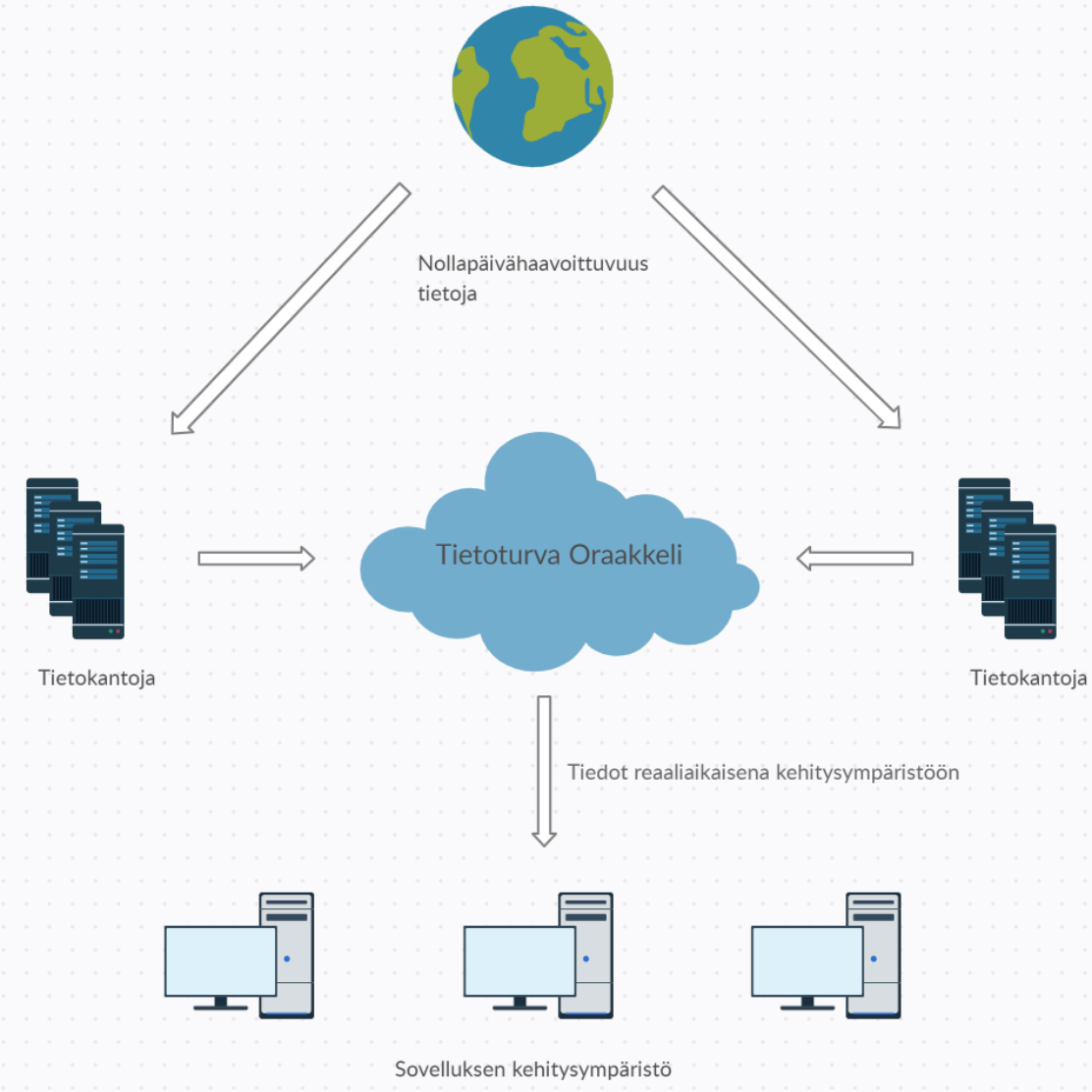
Kehitettävään sovellukseen tulisi tietoturva liittää hyvin varhaisessa vaiheessa. Käyttäjien autentikointi, sovelluksen eri toiminnallisuuksien roolittaminen ja sisään syötettävän tiedon tarkastamisella päästään jo pitkälle. Toiminnallisuuksien yhteensopivuus ja toimivuus voidaan tarkastaa koodikatselmoinneissa.

5.2 ”Tietoturva Oraakkeli”

Opinnäytetyön tilaajan toiveena oli kehittää eräänlainen oraakkeli, jolta voisi keskitetysti saada reaaliaikaista tietoa eri teknologioiden ja kehitysympäristöjen tietoturvallisuudesta. Tietoturva Oraakkelin mahdollisista toteutustavoista ja haasteista haastateltiin muutaman yrityksen sovelluskehittäjiä, sekä verkkoasiantuntijoita. Hyvin nopeasti selvisi, että tällaiselle palvelulle olisi käyttöä, mutta sen toteuttaminen olisi haasteellista. Mittasuhteiltaan tällainen palvelu olisi massiivinen.

Tietoturvallisten ohjelmointikielien ja kehitysympäristöjen reaaliaikainen seuranta nollapäivähaavoittuvuuksille olisi teknisesti lähes mahdotonta toteuttaa olemassa olevalla teknologialla.

Teoriassa tällainen Tietoturva Oraakkeli voisi toimia pilvipalveluna. Pilveen voitaisiin integroida lukuisia lähteitä, joiden kautta päivitetään tietokantoihin uudet tiedot tietoturvallisuudesta ohjelmointikielien ja verkkoteknologioiden osalta. Käyttäjä voisi esimerkiksi yksinkertaisella hakutoiminnolla joko etsiä tietoa suoraan, tai syöttää käytettävissä olevat teknologiansa ja täten saada tietoa välittömästi, tarvitseeko kyseessä olevaa stackia harkita uudelleen.



Kuvio 5 Havainnekuva Tietoturva Oraakkelin mahdollisesta toteutuksesta

Kuviossa 5 hahmotellaan minkälainen Tietoturva Oraakkelin arkkitehtuuri hyvin karkealla tasolla.

5.3 Tietoturva Oraakkelin käyttötapaus

Esimerkkinä Tietoturva Oraakkelin käytöstä voisi olla tilanne, jossa projekti on suunnittelu vaiheessa. Front endin ja back endin osalta on tunnistettu tehtävään parhaiten sopivat ohjelmointikielet, tietokannat sekä integraatiot. Kehittäjillä on selkeä kuva käytettävästä stackista ja he haluavat tietää, onko kyseisen stackin teknologioissa haavoittuvuuksia. Tietoturva Oraakkelissa voisi olla yksinkertainen hakukenttä, tai mahdollisesti alasveto valikko, josta kehittäjä valitsee ohjelmointikielet kohtiin front end ja back end. Tietokantojen osalta valinta tehdään samalla tavalla. Oletetaan että front end teknologiaksi valikoitui React, back endiksi PHP ja tietokannaksi MariaDB. Kun tiedot on täytetty, käyttäjä painaa etsi-painiketta (tai kuinka se sitten olisikaan nimetty) ja saa viimeisimmät tiedot ruudulle teknologioiden tunnetuista haavoittuvuuksista. Esiin voisi tulla esimerkiksi CVE-details tyylinen näkymä, jossa haavoittuvuuksien vakavuus ilmoitettaisiin numeraalisesti, sekä värikoodattuna. Tämän tiedon turvin voidaan arvioida uudelleen, onko valikoitu stack tarpeeksi turvallinen.

6 Pohdinta

Tietoturvan ennakointi on yleistyvä trendi sovelluskehityksessä. Vielä jokunen vuosi sitten näin ei ollut. Oli yleistä työntää ulos nopeasti sovellusta sovelluksen perään ja tietoturvasta murehdittiin sitten kun siitä tuli ongelma. Parin vuoden aikana yritykset ja kehittäjät ovat heränneet todellisuuteen, että tietoturvallisuuden varmistaminen jokaisella sovelluskehityksen osa-alueella on tärkeää.

Toimeksianto saatiin vuonna 2018 ja sen hetkisen tilanteen mukaan, sovellusten tietoturvan ennakointi oli huonommalla tolalla kuin se on nykyään. Prosessit ja työskentelytavat ovat kehittyneet huomattavasti parempaan suuntaan.

Ajatuksena oli kehittää ns. Tietoturva Oraakkeli, jolta aloittelevat sovelluskehittäjät voisivat tarkastaa käytettyjen teknologioiden turvallisuuden. Asiaan liittyen tehtiin haastatteluita. Haastatteluissa kävi selväksi että ohjelmallisesti toteutettuna tällainen Oraakkeli olisi äärimmäisen työläs ja kallis kehittää, puhumattakaan tällaisen ylläpidosta. Tarvetta tällaiselle Oraakkelille varmasti olisi ja tällainen varmasti kehitetäänkin, heti kun se on teknisesti, taloudellisesti, sekä ajallisesti mahdollista.

Tietoturvan ennakointi tuotekehityksessä lepääkin projektissa työskentelevien harteilla täysin. Hyvin harvoin tietokone tekee virheen ilman ihmisen edesauttamista. Turvallisiin työskentelytapoihin keskitytään jatkuvasti enemmän ja maallikon tasollakin tietoturva on tunnetumpi aspekti jo hyvin monella työpaikalla.

Sovellukset, ohjelmointikielet ja kirjastot voidaan auditoida ja testata monin keinoin. Ihmistä emme vielä kykene skannaamaan ja toteamaan tämän olevan uhka yrityksen tietoturvalle. Tietoturva on tapa ajatella ja se on opetettavissa.

Lähteet

Approval testing. 2018. Approval Testing. Artikkeli softwaretestingmagazine.com verkkosivustolla. 15.11.2018. Viitattu 27.8.2020. https://www.softwaretestingmagazine.com/knowledge/approval-testing/

Component testing. n.d. What is Component Testing? Techniques, Example Test Cases. Artikkeli guru99.com verkkosivustolla. N.d. Viitattu 27.8.2020. https://www.guru99.com/component-testing.html

Dynamic testing. n.d. Dynamic Testing. Artikkeli professionalqa.com verkkosivustolla. 27.2.2020. Viitattu 27.8.2020. https://www.professionalqa.com/dynamic-testing

Fuzzing. n.d. Fuzzing. Artikkeli owasp.org verkkosivustolla. n.d. Viitattu 27.8.2020. https://owasp.org/www-community/Fuzzing

Nikiforova E. 2020. What Is the Difference between DevOps and DevSecOps? Artikkeli viva64 verkkosivustolla 4.2.2020. Viitattu 12.9.2020. https://www.viva64.com/en/b/0710/

OWASP. 2020. OWASP Top Ten. Artikkeli owasp.org verkkosivustolla. N.d. Viitattu 25.8.2020. https://owasp.org/www-project-top-ten/

Static testing. n.d. What is Static Testing? What is a Testing Review?. Artikkeli guru99.com verkkosivustolla. n.d. Viitattu 27.8.2020. https://www.guru99.com/testing-review.html

System testing. n.d. What is System Testing? Types & Definition with Example. Artikkeli guru99.com verkkosivustolla. n.d. Viitattu 27.8.2020. https://www.guru99.com/system-testing.html

Talamantes J. n.d. What is a Penetration Test and Why Do I Need It?. Artikkeli redteamsecure.com verkkosivustolla. n.d. Viitattu 27.8.2020. https://www.redteamsecure.com/blog/penetration-test-need/

Traficom n.d. Turvallinen Tuotekehitys: Kohti Hyväksyntää. Verkkojulkaisu kyberturvallisuuskeskuksen sivustolla. n.d. Viitattu 26.8.2020. https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/publication/Turvallinen_tuotekehitys_Suomi_J003_2018.pdf

Turvallisen sovelluskehityksen käsikirja. 2020. Verkkojulkaisu suomidigi.fi -sivustolla. 19.5.2020. Viitattu 29.11.2020. https://www.suomidigi.fi/sites/default/files/2020-05/Turvallisen%20sovelluskehityksen%20k%C3%A4sikirja.pdf

Unit testing. n.d. Unit Testing. Artikkeli softwaretestingfundamentals.com verkkosivustolla. n.d. Viitattu 27.8.2020. https://softwaretestingfundamentals.com/unit-testing/

Kuvalähteet

Kuvio 1 DevOps ja DevSec Ops. Muokattu 29.11.2020.
https://www.viva64.com/en/b/0710/

Kuvio 2 Havainnekuva tietoturvan suunnittelusta. Luotu 29.11.2020.
app.creately.com/diagram -sivustolla

Kuvio 3 CVE Details -sivuston löytämä haavoittuvuus. Muokattu 26.8.2020.
https://www.cvedetails.com/

Kuvio 4 CVE Details -sivuston tarkempi kuvaus tunnetusta haavoittuvuudesta.
Muokattu 26.8.2020. https://www.cvedetails.com/

Kuvio 5 Havainnekuva Tietoturva Oraakkelin mahdollisesta toteutuksesta. Luotu

29.11.2020. app.creately.com/diagram -sivustolla