TRAFICOM
Liikenne- ja viestintävirasto
Kyberturvallisuuskeskus

TURVALLINEN TUOTEKEHITYS: KOHTI HYVÄKSYNTÄÄ

SECURE DEVELOPMENT – TOWARDS APPROVAL

Traficom 003/2018 J

Liikenne- ja viestintävirasto
Kyberturvallisuuskeskus
Puhelin: 029 534 5000 (vaihde)
PL 313 (Erik Palménin aukio 1)
00561 Helsinki

Finnish Transport and Communications Agency
National Cyber Security Centre Finland (NCSC-FI)
Phone: +358 29 534 5000 (switchboard)
P.O. Box 313 (Erik Palménin aukio 1)
FI-00561 Helsinki

www.ncsc.fi
www.traficom.fi

SISÄLLYSLUETTELO

ESIPUHE

Kyberturvallisuudella on merkittävä rooli sekä yhteiskunnalle kriittisten järjestelmien ja toimintojen turvaamisessa että kansalaisten arjessa. Tehokas tapa parantaa kyberturvallisuutta on puuttua mahdollisiin ongelmiin jo ohjelmistotuotteiden ja -palveluiden kehitysvaiheen aikana. Turvallinen tuotekehitys edistää toimintavarmuutta ja ennaltaehkäisee tietomurtoja ja -vuotoja.

Viestintäviraston Kyberturvallisuuskeskuksen kansainvälisiin tietoturvavelvoitteisiin kuuluu salaustuotteiden hyväksyntä kansainvälisen turvallisuusluokitellun tiedon suojaamiseksi Suomessa. Velvoitteen täyttämiseksi Kyberturvallisuuskeskuksen National Communications Security Authority (NCSA-FI -toiminto) on arvioinut salaustuotteiden teknisiä toteutuksia ja niiden valmistajien tuotekehityskäytäntöjä. Arvioitujen tuotteiden tietoturva on parantunut, ja työ on osoittautunut tehokkaaksi osaksi ennaltaehkäisevää kansallista kyberturvallisuustyötä ja suomalaisten salaustuotteiden myynnin edistämistä.

Kevään 2018 aikana NCSA-FI toteutti selvitystyön turvallisen tuotekehityksen ja hyväksyntään valmistautumisen tukemisesta. Selvitystyössä valmisteltiin "Turvallinen tuotekehitys - kohti hyväksyntää" -opas ja suunnitelma ennaltaehkäisevästä tuoteturvallisuustyöstä viestimiseksi. Oppaan valmistelussa hyödynnettiin sekä NCSA-FI -toiminnon kokemusta että teollisuuden asiantuntijoita.

FOREWORD

Cyber security has a significant role in protecting society, its critical systems and its citizens. An effective way to improve cyber security is to address these potential problems early during the development stage of producing software-based products and services. Secure development improves robustness, ensures continuity, and helps prevent data breaches or leakages.

Duties of the FICORA's National Cyber Security Centre (NCSC-FI) include approving cryptographic products for protecting international classified information in Finland. In order to fulfil this obligation, the National Communications Security Authority (NCSA-FI), operating as part of the NCSC-FI, has carried out assessments of cryptographic products and their development processes. This has helped to improve the security of these products. Furthermore, this work has proven to be an effective and proactive contribution to national cyber security, while also promoting sales and exports of Finnish security products.

In the spring of 2018, NCSA-FI began exploring how better to support vendors in secure development and methods for preparing their products for assessment and accreditation. As part of this work, this guidebook called Secure Development: Towards Approval and plans for what information to include with this guidance were developed. Both industry experts and NCSA-FI's own experts participated in developing the guide.

YHTEENVETO

Tämän oppaan tarkoitus on auttaa valmistajia tekemään laadukkaita ja turvallisia tuotteita. Opas on suunnattu Kyberturvallisuuskeskuksen NCSA-FI salaustuotehyväksyntää hakeville ja muille tietoturvasta kilpailuetua tavoitteleville suomalaisille valmistajille. Opas tehostaa hyväksyntään valmistautumista, antaa neuvoja turvallisesta tuotekehityksestä ja tukee tietoturvatuotteiden kehitystä viranomaiskäyttöön ja vientiin.

Tämä opas on yhteenveto turvallisen tuotekehityksen vaiheissa huomioitavista asioista.

Tietoturva on tietoturvaominaisuuksia, esimerkiksi salaustoimintoja, mutta myös ohjelmiston laatutekijä. On siis tärkeää kiinnittää huomio myös salausta laajemmin tuotteen eri toimintoihin, ja ymmärtää, että myös ne kuuluvat hyväksynnän piiriin. Ominaisuuksien lisäksi tietoturvallinen tuotekehitys kattaa myös kehityksen ja ylläpidon aikaiset toimet, kuten tilaturvallisuuden, käytettyjen järjestelmien turvallisuuden sekä tuotekehityshenkilöstön koulutuksen. Itsearviointi Katakri-auditointityökalun ohjeistuksen avulla antaa hyvän pohjan kehitysympäristön ja -organisaation valmistelemiseksi hyväksyntään.

Uhkamallinnus on tuotteen suunnittelu- ja päivitysvaiheiden tärkeimpiä työkaluja. Uhkamallissa kuvataan tuotteen käyttötapaukset, ympäristö uhkien näkökulmasta, järjestelmän tuottamat arvokkaat tiedot ja palvelut, ja kuinka tuote vastaa näihin kohdistuviin uhkiin. Tärkeintä on, että uhkamalliin liittyvät asiat käydään läpi osana tuotekehitystä, eikä se millä metodilla uhka-arvio tehdään. Uhkamalli tukee myös tarkastustoimien tehokasta rajaamista ja kohdistamista.

Tuotteen arkkitehtuurissa ja suunnittelussa turvallisuutta lisäävät hyväksi todetut suunnitteluperiaatteet: hyökkäyspinta-alan minimointi, turvalliset oletusarvot, ulkopuolisten syötteiden tarkistus, oikeuksien minimointi, syvyyssuuntainen puolustus, turvalliset virhetilat, epäluottamus ulkopuolisiin palveluihin ja turvamekanismien yksityiskohtien salailuun pohjautuvien oletusten välttäminen.

Dokumentoidut ja omaksutut suunnitteluperiaatteet helpottavat sekä turvallista toteutusta että toteutuksen turvallisuuden arviointia.

Tuotteen toteutusvaiheessa on tärkeää varmistaa turvallista tuotekehitystä tukevat työkaluvalinnat, toteuttajien tietoturvaosaaminen sekä valittujen kolmansien osapuolten komponenttien ja alustaratkaisujen turvallisuus. Monet tietoturvaongelmat syntyvät ohjelmointivaiheessa. Turvalliset tekniikat sekä niiden puutteet riippuvat käytetyistä alustoista, komponenteista, ohjelmointikielistä ja työkaluista. Kaikkiin näihin tulee perehtyä. Materiaaleja tietoturvalliseen ohjelmointiin, riippuvuuksien turvallisuuden arviointiin ja alustojen koventamiseen on yleensä hyvin saatavilla. Kolmansien osapuolten komponenteista löytyy ja julkaistaan haavoittuvuuksia säännöllisesti, joten tietoturvan tason säilyttäminen vaatii tuotteen päivittämistä. Päivitykset puolestaan voivat vaatia uudelleenhyväksyntää, jolloin kehitys- ja päivitysprosessin kypsyyden merkitys korostuu arviointitoiminnassa.

Ohjelmiston laadunvarmistukseen, siis myös tietoturvaan, kuuluu kattava testaus. Testausta pitää suorittaa todellista käyttötilannetta vastaavassa ympäristössä ennen kuin tuote tulee hyväksyntään. Testauksessa ja laadunvarmennuksessa tulee pyrkiä kohti toistettavia ja automaattisia menetelmiä, koska niillä saavutetaan suurempi testikattavuus, ja järjestelmään tehtäviä muutoksia pysytään näin testaamaan tehokkaasti ja luotettavasti. Tuotteen ja sen osakokonaisuuksien helppo testattavuus nopeuttaa myös sen hyväksyntää. Myös katselmoinnit ovat tärkeä osa laadunvarmennusta, ja tuote pitäisi katselmoida myös tietoturvaperspektiivistä. Toteutetut testaukset, itsearvionnit ja mahdolliset kolmansien osapuolen suorittamat tarkastukset tukevat hyväksyntään valmistautumista.

Tämä opas on yhteenveto turvallisen tuotekehityksen vaiheissa huomioitavista asioista. Tämän lisäksi on välttämätöntä perehtyä oman erikoisalan ja valittujen työkalujen ja alustojen tietoturvan erityispiirteisiin. Jos voidaan todeta valmistajan tuotekehitystyökalujen ja -menetelmien, testauksen, tilojen ja kehittäjien osaamistason olevan kunnossa, asiakkaiden luottamus tuotteeseen parantuu ja hyväksyntä nopeutuu.

EXECUTIVE SUMMARY

The purpose of this guide is to help vendors to create high-quality, secure products. It is aimed at organisations applying for approval of cryptographic products from NCSA-FI and at other Finnish vendors seeking to gain a competitive advantage from information security. This guide helps readers to better prepare for the assessment, provides insight for anyone interested in secure development, and supports the development of products for govemental use and export.

This guide provides a summary of the phases of secure product development.

Security is made up of features such as encryption, but security is also a quality attribute of the system. Therefore, you should pay attention to all features, not just to the cryptographic functions. A security assessment will look at your product as a whole. Beyond features, secure product development encompasses the development process itself, including the maintenance phase. The development facilities and tools must be secured, and the staff must be trained. Doing self-assessment using the Katakri auditing tool gives a good foundation for preparing both the development environment and the organisation for passing the approval process.

Threat modelling is one of the most important tools for designing and maintaining your product. The threat model describes the use cases of the product, the threat environment, the valuable information protected by the system, and the services offered by the system. Threat models help you to assess how the product counters threats. There is no single correct way to do threat modelling; the important thing is to do it and to leverage the results to support better secure product development. The threat model also supports the effective specification and targeting of assessment activities.

Product security is enhanced by using established architecture and design principles: minimal attack surface, safe defaults, input sanitation, minimal privileges, defence in depth, failing safely, not trusting external services, and avoiding security by obscurity. Adopting and documenting design principles facilitates both secure implementation and security assessment.

When implementing a product, the tools you use should support secure development, developers should be security-aware, and third-party components and platforms should be secure. Many security issues arise during the programming phase. Secure programming techniques and security pitfalls are specific to the platforms, components, programming languages, and tools used. You must be familiar with all of these. Material for secure programming, dependency security assessment, and platform hardening are generally well known and easily available. Vulnerabilities in third-party components are found and announced regularly, so maintaining the security of the product requires constant updates. Updates may lead to re-approval, which places even more emphasis on maturity in the development and maintenance processes as an enabler for an efficient approval process.

Comprehensive testing is part of software quality assurance, including security assurance. Testing needs to be carried out before the product is submitted for approval, in an environment that matches the real-world situation. Testing and quality assurance should aim for reproducible, automated methods, as they provide greater test coverage and enable efficient and reliable testing of the changes to the system. Products and components that have good testability will speed up approval. Code reviews are an important part of quality assurance, and the product should also be reviewed from the security perspective. Carrying out various tests and self-assessments, and receiving third-party assessments improve the approval process.

This guide provides a summary of the phases of secure product development. On top of that, you need to familiarize yourself with the specifics of your domain and the tools and platforms you are using. After all, when a vendor's development tools and methods, testing, facilities and developer skills are in good shape, customers' confidence in the product will improve – and, again, the approval process will be faster.

TURVALLISUUDELLA ON MERKITYSTÄ

Tämän oppaan tarkoituksena on auttaa korkealaatuisten ja turvallisten järjestelmien luomisessa. Siinä kuvataan turvallisten järjestelmien suunnittelua, toteutusta ja testausta. Järjestelmäkehittäjien lisäksi tietoa turvallisesta kehittämisestä tarvitsevat myös tuotekehitys- ja tuotejohtajat sekä muut tuotekehitykseen osallistuvat henkilöt, sillä heidän tukensa on välttämätöntä turvallisuuden kannalta. Turvallista kehittämistä koskevien ohjeiden lisäksi oppaassa on myös käytännön vinkkejä, jotka koskevat Kyberturvallisuuskeskuksen NCSA-FI -salaustuotehyväksynnän hakemista tai sen suunnittelua.

Tämän oppaan tarkoituksena on auttaa korkealaatuisten ja turvallisten järjestelmien luomisessa.

TURVALLISUUS TUO KILPAILUETUA YRITYKSELLE:

- Yrityksissä ollaan aiempaa tietoisempia kyberturvallisuuteen ja tietosuojaan liittyvistä kysymyksistä. Tämän vuoksi asiakkaat vaativat niihin liittyviltä tuotteilta ja palveluilta turvallisuutta ja korkeaa laatua, vaikkeivät pystyisikään itse todentamaan näitä ominaisuuksia.

- Jos yrityksessä on osattu ennakoida tulevaa, sillä ei ole vaikeuksia täyttää tarjouskilpailujen turvallisuutta koskevia vaatimuksia. Yritys on valmis, kun myyntijohtaja toteaa, että ”ensi viikon asiakastapaamiseen” tarvitaan näyttöä turvallisista kehityskäytännöistä.

- Jos yritys on osaltaan varmistanut järjestelmänsä turvallisuuden, sen asiakkaat ovat paremmassa turvassa. Jos yritys osoittaa toimivansa riittävän huolellisesti (due diligence), on vähemmän todennäköistä, että sen asiakkaat vaativat sitä vastuuseen.

- Turvallisuushäiriöiden hoitaminen vie valtavasti aikaa ja rahaa.

Turvallinen kehittäminen pienentää riskejä, on hyväksi liiketoiminnan jatkuvuudelle ja yleisesti parantaa työn laatua. Kun turvallisuus otetaan osaksi liiketoimintakäytäntöjä, se vaikuttaa myönteisesti koko yritykseen.

Varsinkin ohjelmistopohjaisten tuotteiden turvallisuus on aiemmin usein ollut huonolla tolalla. Monissa tilanteissa toimittaja herää turvallisuuskysymyksiin vasta myöhään, esimerkiksi tarjousvaiheessa. Jos tuotteiden turvallisuusominaisuuksiin kiinnitetään huomiota vasta myöhäisessä kehitysvaiheessa, se vaikeuttaa kaikkien toimintaa: toimittajien, tarkastajien ja ostajien. Vaikuttaa siltä, että johto ei aina korosta sitä, kuinka välttämätöntä turvallisuus on, joten siihen ei aina suhtauduta niin ennakoivasti kuin voisi kuvitella.

VINKKI:

Turvallisuustarkastuksiin valmistautuminen hyödyttää tuotekehitystä:

- Jos yrityksen johto odottaa, että tarkastus läpäistään, sen on myös investoitava turvallisuuteen, laatuun ja jatkuvuuteen ja omaksuttava ne osaksi yrityskulttuuria.

- Ennakoiva toiminta helpottaa tarkastusten läpäisemistä.

- Itse arvioinnissa voidaan löytää korjattavia puutteita ja uusia tapoja tuotteen parantamiseksi.

Lisälukemista

- *“Computer security is broken from top to bottom”, The Economist, 8 April 2017*
- *“Why Cybersecurity Should Be a No. 1 Business Priority For 2017”, Forbes.com, 20 March 2017*

TILAT JA HENKILÖSTÖ – TOIMINNAN TURVALLISUUS

Turvallisia tuotteita tehdään turvallisissa ympäristöissä. Ohjelmistot ovat ihmisten suunnittelemia ja toteuttamia, ja ihmiset tarvitsevat työkaluja ja työtiloja. Jos tilat, joissa lähdekoodia, koontituotteita, työkaluja tai työkoneita säilytetään, eivät ole turvallisia, hyökkääjä voi vaarantaa niissä valmistettavien tuotteiden turvallisuuden. Lähdekoodiin voi esimerkiksi lisätä takaportin. Turvallisuutta ajateltaessa on ajateltava kaikkia järjestelmän osia, myös henkilöstöä ja tiloja.

Kehityksessä käytettävien työkalujen on vastattava kehitettävien tuotteiden turvallisuusvaatimuksia. Esimerkiksi versionhallinta, jossa kirjataan kaikki muokkaukset ja niiden tekijät, on välttämätöntä. Palvelimiin on aina tehtävä viimeisimmät turvallisuuspäivitykset, ja kaikilla käyttäjillä on oltava oma tili lokitietojen tallennusta varten. Sovelluskehittäjille suunnatut suositut pilvipalvelut saattavat pystyä varmistamaan tarjoamansa alustan turvallisuuden ja ajantasaisuuden paremmin kuin organisaatiot, joilla ei ole tarpeeksi resursseja sisäisen kehitysalustansa turvallisuuden varmistamiseen.

Ihmisten pitäisi tuntea tietoturvahygienian yleiset periaatteet.

Kehittäjät tarvitsevat koulutusta suosituista turvallisen ohjelmistokehityksen elinkaarimalleista. Koulutusta on tarjottava kaikille niiden parissa työskenteleville työntekijöille. Ihmisten pitäisi tuntea tietoturvahygienian yleiset periaatteet: älä avaa jokaista sähköpostiviestiä, ole varovainen selaillessasi ja vältä vapaa-ajan selailua työkoneella, älä käytä sattumalta löytämiäsi USB-muistitikkuja koneellasi, pidä järjestelmäsi ajan tasalla ja ole tietoinen käyttäjien manipuloinnin peruspiirteistä. Kehittäjille on annettava lisäkoulutusta turvallisesta suunnittelusta, uhkamallinnuksesta ja turvallisesta ohjelmoinnista.

Ihmiset vaihtuvat, joten koulutusta on tarjottava osana uusien työntekijöiden perehdytystä. Yhteen tai kahteen huippuohjelmoijaan ei kannata tukeutua, koska se ei ole turvallinen strategia tuotekehityksessä eikä yrityksen kannalta.

Kehitysmateriaalia sisältäviin kannettaviin tietokoneisiin on myös asennettava viimeisimmät turvallisuuspäivitykset, ja ne on varustettava levyn salauksella.

Tilojen ja henkilöstön lisäksi on huomioitava myös organisaatiossa käytössä olevat prosessit. Tukevatko ne korkealaatuisten ja turvallisten tuotteiden kehittämistä? Lainaus OWASP-järjestön sivustolta (https://www.owasp.org/index.php/Policy_Frameworks): *"... turvallisia sovelluksia koskevat seuraavat vähimmäisvaatimukset:*

- *turvallisuutta vaaliva organisaation johto*
- *kansallisiin standardeihin perustuva kirjallinen tietoturvakäytäntö*
- *riittävät turvallisuustarkistukset ja -toiminnot sisältävät kehittämismenetelmät*
- *turvalliset julkaisun- ja konfiguraationhallinnan prosessit."*

OWASP-järjestö toteaa myös seuraavaa: "Tuotekohtaisella kehittämisellä ei saada aikaan turvallisia sovelluksia, sillä se ei ole riittävän järjestelmällistä. Turvallisen koodin luontiin pyrkivien organisaatioiden onkin johdonmukaisesti käytettävä menetelmiä, jotka tukevat kyseistä tavoitetta. On tehtävä huolellisia valintoja – pienten tiimien ei kannata koskaan käyttää raskaita, monia eri rooleja sisältäviä menetelmiä, ja suurten tiimien on valittava menetelmiä, jotka voidaan skaalata niiden tarpeisiin sopiviksi."

VINKKI:

Katakri-kriteeristön (tietoturvallisuuden auditointityökalu viranomaisille) avulla yritykset voivat arvioida itse tuotekehityksessä käyttämiään tiloja ja prosesseja sekä organisaatiorakennettaan. Sitä voi käyttää tarkistuslistana organisaation ja sen tilojen turvallisuuden varmistamisessa. Ulkoisissa arvioinneissa noudetaan todennäköisesti samoja periaatteita, joten jos organisaation toiminta on Katakri-kriteeristön mukaista, tällaiseen arviointiin valmistautuminen on helppoa.

Lisälukemista

- *"Katakri 2015 - Tietoturvallisuuden auditointityökalu viranomaisille (saatavilla suomeksi ja englanniksi)"*

VAATIMUKSET JA UHKAMALLINNUS

Toiminnallisuus, jota ohjelmistolta odotetaan, määritetään ohjelmistovaatimuksissa. Ohjelmistovaatimukset ilmaistaan käyttötapauksina tai vaatimusluetteloina. Pienissä projekteissa vaatimukset voivat olla hyvinkin epämuodollisia, eikä niitä välttämättä edes kirjata muistiin.

Oikeanlaisten vaatimusten määrittäminen on vaikeaa, mutta se on välttämätöntä projektin onnistumisen kannalta. Vaatimusten määrittäminen aivan projektin alussa, ennen suunnittelua ja toteutusta, on yleensä mahdotonta, koska sekä kehittäjäorganisaation että asiakkaiden tietämys lisääntyvät projektin edetessä. Nykyään suositaan yhä enemmän toistuvissa sykleissä tehtävää työtä: vaatimusten kokoaminen → suunnittelu → toteutus → todentaminen. Turvallisuusvaatimusten määrittäminen voi olla vielä vaikeampaa kuin puhtaasti toiminnallisten vaatimusten määrittäminen. ”Ohjelmiston ei pitäisi kaatua” on hyvä tavoite turvalliselle ohjelmistolle, mutta todennettavaksi vaatimukseksi se on yleensä liian yleispätevä. Turvallisuusvaatimusten määrittämistä tarkastellaan tarkemmin uhkamallinnuksen yhteydessä.

TURVALLISUUSVAATIMUKSET

Toisinaan vaatimukset ovat puutteellisia tai niitä ei ole määritetty, minkä seurauksena suunnittelussa ja toteutuksessa hukataan aikaa ja valmiissa ohjelmistossa on puutteita. Sama pätee turvallisuusvaatimuksiin. Niissä määritetään, millä tavoin järjestelmän tietoja ja palveluja suojataan haitallisilta toimijoilta ja muilta vastoinkäymisiltä. Turvallisuusvaatimuksissa voidaan esimerkiksi määrittää, miten käyttäjät tunnistetaan tai mitkä järjestelmän tiedot on salattava.

Samoin kuin muutkin vaatimukset, turvallisuusvaatimukset voivat olla toiminnallisia tai ei-toiminnallisia. Toiminnallisessa vaatimuksessa voidaan esimerkiksi todeta, että käyttäjien on ensin kirjauduttava sisään käyttäjätunnuksella ja salasanalla. Ei-toiminnallinen turvallisuusvaatimus voi olla se, että sovelluksen on tarkistettava kaikki verkon kautta saapuvat syötteet ja hylättävä kaikki virheelliset pyynnöt.

Toimivia turvallisuusvaatimuksia laadittaessa on mietittävä, kuinka tuotetta käytännössä käytetään ja millaisissa ympäristöissä. Käytetäänkö järjestelmää eristetyssä ja vartioidussa bunkkerissa? Toteutetaanko järjestelmä pilvipalveluna? Onko siinä verkkoselaimessa toimivia komponentteja? Millaisia tärkeitä resursseja (yleensä tietoja) järjestelmässä käsitellään?

VINKKI:

Turvallisuusvaatimukset, myös epäsuorat vaatimukset, kannattaa kirjata muistiin. Kirjalliset turvallisuusvaatimukset parantavat tuotteen turvallisuutta ja turvallisuusarviointien tehokkuutta.

Kun tuotteen käyttötarkoitus on määritetty, voidaan miettiä, mikä voisi mennä vikaan. Onko tuotteen väärinkäyttö mahdollista? Jos on, kuka sitä voi käyttää väärin, miten ja milloin? Voiko hyökkääjä saada haltuunsa jotakin hänelle kuulumatonta, tai voidaanko hyökkäyksen seurauksena menettää jotakin arvokasta? Voivatko muut kuin valtuutetut henkilöt päästä palvelimille ja varastaa tietoja tai päästä jopa fyysisille levyille? Entä jos selainkomponentti takaisinmallinnetaan ja korvataan haitallisella sovelluksella? Mitä seurauksia olisi järjestelmän turvallisuuden vaarantumisella?

Järjestelmän turvallisuus voidaan vaarantaa monella eri tavalla, ja myös hyökkäyksiä on monenlaisia. Useimmissa tunnetuissa hyökkäyksissä on kuitenkin yhteisiä piirteitä, ja ne voidaan luokitella muutamaan yleiseen tyyppiin. Esimerkiksi OWASP on julkaissut listan kymmenestä tärkeimmästä verkkosovelluksia koskevasta turvallisuusriskistä. Siinä mainitaan seuraavat haavoittuvuudet, jotka pätevät myös muualla kuin verkkosovelluksissa.

1. **Injektio:** Sovellus hyväksyy ulkoiset syötteet, mutta ei tarkista niitä kunnolla. Tällöin hyökkääjä voi suorittaa komentoja tai tehdä muuta vahinkoa haavoittuvassa sovelluksessa.

2. **Rikkinäinen todennus:** Käyttäjien todennusta ei ole toteutettu oikein. Salasanoja ei varmenneta, salasanat vuotavat tai järjestelmään voi hyökätä salasanan palautuksen avulla. Todennuksen jälkeen tapahtuva istunnonhallinta voi myös olla rikkinäinen niin, että istuntoja voidaan kaapata.

3. **Arkaluonteisten tietojen paljastuminen:** Arkaluonteisia tietoja säilytetään tai siirretään selkokielisinä tai käyttäen heikkoa salausta.

4. **Ulkoiset XML-entiteetit (XXE) (XML external enti-ties (XXE)):** Sovelluksen XML-käsittely ei ole turvallista; esimerkiksi SAML-kertakirjautuminen on toteutettu väärin.

5. **Rikkinäinen käytönvalvonta:** Sovellus sallii käyttäjien suorittaa toimia, joihin heillä ei ole käyttöoikeuksia.

6. **Vääränlaiset turvallisuusasetukset:** Järjestelmän turvallisuutta ei ole kovennettu tai siinä on tarpeettomia palveluja käynnissä, tai alusta on vanha ja haavoittuva ja sisältää turvattomia paikallisia tilejä.

7. **XSS-haavoittuvuus:** Hyökkääjät voivat suorittaa haitallista HTML- tai JavaScript-koodia.

8. **Turvaton sarjallistetujen tietojen lukeminen (insecure deserialisation):** Hyökkääjät pääsevät käsiksi sarjallistettuihin tietoihin tai olioihin, jotka sovellus lukee ja käyttää sitten toimiin, joihin vaaditaan valtuudet.

9. **Tunnettuja haavoittuvuuksia sisältävien komponenttien käyttäminen:** Sovelluksessa käytetään tahallisesti tai tahattomasti komponentteja, joiden tiedetään sisältävän haavoittuvuuksia.

10. **Riittämätön lokiinkirjaus ja seuranta:** Sovelluksen lokit eivät ole riittävän turvallisia myöhempää tarkistamista varten, tai sovellus ei seuraa mahdollisia hyökkäyksiä tai varoita niistä.

”Väärinkäyttäjiä” on monenlaisia – harrastelijahakkereista hyötyä tavoitteleviin rikollisiin ja valtiollisiin toimijoihin – ja myös heidän valmiutensa poikkeavat toisistaan. Se, millaisiin hyökkäyksiin järjestelmän suunnittelussa on varauduttava, riippuu järjestelmän käyttötarkoituksesta.

On mietittävä myös mahdollisten hyökkääjien motiiveja, järjestelmän sisältämiä tärkeitä resursseja sekä onnistuneen hyökkäyksen vaikutuksia omaan yritykseen ja asiakkaisiin.

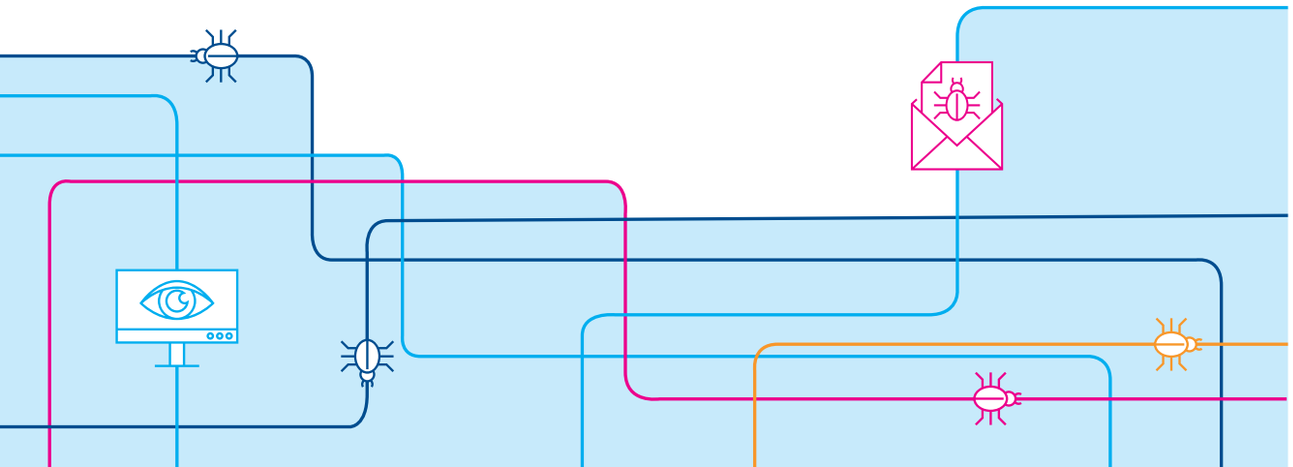
UHKAMALLINNUS

Uhkamallinnuksella tarkoitetaan järjestelmän turvallisuuden arviointia. Edellä turvallisuusvaatimusten käsittelyn yhteydessä kuvattiin jo joitakin uhkamallinnuksen yleisiä periaatteita. Uhkamallinnuksessa voidaan käyttää monia eri menetelmiä, mutta siihen ei vaadita erityisosaamista tai monimutkaisten menetelmien opettelemista.

”Uhkamallinnukseen on monia erilaisia lähestymistapoja... Paljon tärkeämpi kuin tietty riskien arviointiin käytettävä menetelmä on järjestelmällisen uhkamallinnuksen toteuttaminen käytännössä. Microsoft on todennut, että tärkeintä sen turvallisuuden parannusohjelmassa oli se, että uhkamallinnus otettiin käyttöön koko yrityksessä.” - Threat Risk Modeling - OWASP

Väliä ei siis ole niinkään sillä, miten uhkamallinnus toteutetaan, vaan sillä, että se toteutetaan. On osattava ajatella mahdollisen hyökkääjän tavoin, ja sisällytettävä turvallisuusvaatimukset toimintasuunnitelmaan. Uhkamallinnusta varten järjestelmässä on oltava korkean tason arkkitehtuuri, sillä eri komponenteilla on erilaiset roolit ja turvallisuusominaisuudet, ja ne sisältävät eri turvallisuustason tietoja. Arkkitehtuurin suunnittelu kuuluu yleensä osaksi suunnittelua, joten kehityksessä on toistettava suunnittelu- ja uhkamallinnusvaiheita. Tämäkin puoltaa toistuvissa sykleissä tapahtuvaa kehitystä.

Jos uhkamallinnus vaikuttaa haastavalta, siihen voi hakea apua konsultilta.



VINKKI:

Jos tuotteelle tehdään turvallisuustarkastus, siihen on sisällyttävä uhkamallinnus. Tarkastajat voivat keskittyä tuotteen tiettyihin ominaisuuksiin, ja todennäköisesti heillä on oma näkemyksensä tuotteen käyttötilanteista ja ympäristöstä. Tuotteella voi esimerkiksi olla 20 ominaisuutta, mutta tarkastus koskee niistä vain viittä. Muut 15 ominaisuutta saatetaan pyytää poistamaan käytöstä tarkastuksen ajaksi, jos niillä ei ole vaikutusta tuotteen turvallisuuteen.

Tarkastuksessa käytetty uhkamalli ei välttämättä vastaa kehittäjien omaa uhkamallia. Tarkastuksessa keskitytään tarkastuksen kannalta olennaisiin uhkiin. Teknologian toimittajan on ymmärrettävä asiakkaansa tarpeita ja luotava toimiva uhkamalli yhteistyössä tämän kanssa. Millaisia ja minkä turvallisuustason tietoja toimitetulla järjestelmällä käsitellään? Millaisia seurauksia tietoturvaloukkauksella olisi asiakkaalle? Millaisilta huijareilta tai vihollisilta asiakkaan on suojauduttava? Ketkä ovat järjestelmän tulevia käyttäjiä?

On todennäköistä, että tarkastajat käyttävät järjestelmää omasta näkökulmastaan käsin ja korostavat oman tehtävänsä kannalta olennaisia käyttötapauksia ja -tilanteita. He havainnoivat järjestelmän toimintaa ja tukeutuvat omaan uhkamalliinsa. Otetaanko siinä uhkamallissa huomioon tuotteen todelliset käyttötilanteet ja ominaisuudet?

Tämä on hyvä tilanne, jos toimittajalla on jo olemassa järjestelmä, jonka turvallisuudesta sillä ei ole täyttä varmuutta. Kun toimittaja ymmärtää, mitä asioita tuotteen lopulta arvioiva asiakas pitää tärkeinä, se voi laatia niiden pohjalta ensisijaisen tehtävälistan turvallisuuspäivityksiä varten. Kysymykset, joilla ei juuri nyt ole asiakkaalle suurta merkitystä, voidaan jättää tuleviin projekteihin.

SISÄÄNRAKENNETTU TURVALLISUUS TAI TURVALLISUUS LIITÄNNÄISENÄ

Sisäänrakennetulla turvallisuudella viitataan yleensä siihen, että turvallisuusvaatimukset huomioidaan kehitysprojektin alusta alkaen, jolloin tuloksena on tuote, jossa turvallisuus on kiinteä ominaisuus. Toinen vaihtoehto on turvallisuus liitännäisenä, jolloin turvallisuuteen kiinnitetään huomiota vasta tuotteen toteutuksen jälkeen ja sen turvallisuus saadaan aikaan lisäämällä uusia suojauskomponentteja ja -ominaisuuksia.

Sisäänrakennettu turvallisuus on parempi vaihtoehto useasta syystä. Uusien komponenttien lisäämiseen liittyy aina integrointiongelmia, suurempi haavoittuvuuksien riski ja ne aiheuttavat lisäkustannuksia. Ne voidaan välttää sisäänrakennetulla turvallisuudella, jossa turvallisuus on komponenttien kiinteä ominaisuus. Turvallisuus on myös laatua määrittävä tekijä ylläpidettävyyden, testattavuuden, suorituskyvyn ja käytettävyyden ohella. On vaikeaa varmistaa laatua uusia komponentteja lisäämällä.

Kun turvallisuus huomioidaan alusta alkaen ja kehityshenkilöstö koulutetaan turvallisuutta vaalivaan ajattelutapaan, siitä on hyötyä projektin myöhemmissä vaiheissa. Turvallisuusarviointien tekeminen, turvallinen käyttöönotto ja turvallisuuspäivitykset ovat välttämättömiä tuotteen turvallisuuden kannalta.

Turvallisuusliitännäiset, kuten palomuurit, sovellusten hiekkalaatikointi (sandboxing) ja hyökkäyksen havaitsemisjärjestelmät, voivat tuoda lisäsuojaa. Ne toimivat kuitenkin vain hätävarana, jos turvallisuus ei ole ollut keskeinen lähtökohta tuotteen kehityksessä. Tällöin on suositeltavaa alkaa viipymättä ”rakentaa turvallisuutta sisään” tuotteen seuraaviin julkaisuihin.

VINKKI:

Jos tuotteen turvallisuudessa on puutteita ja jos suunnitellut käyttötapaukset ovat erityisen haastavia, arviointi voi edellyttää sitä, että tuotteeseen lisätään suojaominaisuuksia. Niitä voivat olla esimerkiksi fyysiset esteet tai palomuurit.

TIETOSUOJA

Tietosuoja ja henkilötietojen suojaaminen liittyvät turvallisuuteen. Tietosuoja edellyttää turvallisuutta, mutta esimerkiksi sitä, kuinka kauan henkilötietoja voidaan säilyttää tai pitääkö asiakkaille ilmoittaa näiden tietojen säilyttämisestä, ei käsitellä tässä oppaassa. On siis muistettava, että näitä kysymyksiä ei useinkaan voi jättää huomiotta.

Lisälukemista

- *Synopsys, kesäkuu 2016: Are You Making Software Security a Requirement?*
- *OWASP: Threat Risk Modeling*

SUUNNITTELU

Kun vaatimukset on määritetty, siirrytään järjestelmän suunnitteluun. On toivottavaa, että kehittämisessä sovelletaan toistuvia syklejä ja että vaatimuksiin ja suunnitteluvaiheisiin palataan useita kertoja. Se helpottaa työtä, koska ensimmäisellä yrittämällä tuskin syntyy täydellistä järjestelmää.

Suunnittelussa määritetään järjestelmän arkkitehtuuri, jota tarvitaan vaatimuksissa määritetyn, suunnitellun toiminnallisuuden aikaansaamiseen. Arkkitehtuurilla on kuitenkin valtava vaikutus myös järjestelmän turvallisuuden varmistamisen kannalta. Kun arkkitehtuurisuunnitelmalle suoritetaan uhkamallinnus, hyvin luultavasti huomataan, millaisilla arkkitehtuuriin muutoksilla voidaan helpottaa tuotteen turvallisuuden varmistamista.

TURVALLISEN SUUNNITTELUN PERIAATTEET

Turvallisessa suunnittelussa on olemassa joitakin yleisiä nyrkkisääntöjä. Seuraavassa kuvatut säännöt on laadittu OWASPin turvallisen suunnittelun periaatteiden (Security by Design Principles) pohjalta.

Hyökkäyspinnan minimoiminen

Hyökkäyspinnat ovat niitä järjestelmän osia, jotka ovat kosketuksissa ulkomaailmaan joko fyysisesti tai verkon tai tiedostojen välityksellä. Kaikkien hyökkäysten odotetaan tapahtuvan tämän pinnan kautta, jollei järjestelmään ole jäänyt jotakin perustavaa vikaa.

Hyökkäyspintaa voi minimoida poistamalla rajapintoja, jotka eivät ole välttämättömiä. Voidaan miettiä esimerkiksi, tarvitseeko laitteessa olla USB-porttia vai voisiko sen poistaa käytöstä, jotta laitteeseen pääsy vaikeutuu. Voiko alustaa koventaa poistamalla käytöstä sen käyttöjärjestelmän oletuspalveluja? Alustan koventamiseen palataan jäljempänä tässä oppaassa. Voisivatko pääkäyttäjät käyttää järjestelmää vain paikallisesta koneesta käsin? Voisiko järjestelmän verkkopalvelun toteuttaa staattisilla sivuilla ilman dynaamista käsittelyä verkkopalvelimella?

Hyökkäyspinnan pienentäminen hyödyttää monin tavoin. Huomioon otettavia uhkatilanteita on vähemmän, toteutuksessa tapahtuvien virheiden mahdollisuus pienenee ja järjestelmän testaus on helpompaa.

Turvalliset oletusasetukset

Asiakkaiden ei tarvitse olla tietoturvan asiantuntijoita, jotta he voivat käyttää järjestelmää turvallisesti. Valitettavasti järjestelmät ovat usein turvattomia oletusarvoisesti, ja järjestelmän turvallisuuden varmistaminen jää pääkäyttäjän tehtäväksi. Jos järjestelmän pääkäyttäjä ei tunne järjestelmää yhtä hyvin kuin sen kehittäjä, järjestelmä on toimitettava siten, että sen asetukset ovat turvalliset ja vaativat mahdollisimman vähän määrittämistä. Jos järjestelmän turvallisuutta aiotaan huonontaa asetuksia muuttamalla, sen pitäisi olla valinta, jonka käyttäjä tekee tietoisesti. Järjestelmän kehittäjän on harkittava omaa vastuutaan. Onko parempi olla vastuussa oletusarvoisesti turvattoman järjestelmän toimittamisesta vai antaa järjestelmän pääkäyttäjän tehdä päätös turvallisuuden huonontamisesta riskien arvioinnin jälkeen?

Syötteenkäsittely[1]

Jos syötteitä ei tarkasteta, hyökkääjä saattaa pystyä vioittamaan järjestelmän haavoittuvia osia tai kaatamaan ne. Lisäksi on usein mahdollista suorittaa järjestelmässä komentoja tai muutoin saada se hallintaan. Järjestelmään ulkopuolelta tulevat syötteet on tarkastettava aina, kun mahdollista. Tarkastaminen koskee sitä, että syötteen (esim. viestien) syntaksi vastaa odotettuja sääntöjä ja että syöte on semanttisesti kelvollista.

Siirrettävät tiedot suojataan usein salauksella ja eheystarkistuksilla, mutta jos syötetietojen alkuperä ei ole täysin omassa hallinnassa, myös salatut syötteet on tarkistettava huolellisesti.

[1] Ei sisälly OWASPin listaamiin periaatteisiin.

On otettava huomioon myös rajapinnat, joihin syötteet vaikuttavat epäsuorasti. Hyökkääjän viestit saattavat kulkeutua syvälle järjestelmän ydinosiin asti. Seuraavassa on esimerkki syötteen etenemisestä.

1. Verkkopalvelin vastaanottaa käyttäjätunnuksen ja salasanan (tunnistetiedot).

2. Se lähettää tunnistetiedot todennuspalvelimelle todennuksen avustajan avulla.

3. Todennuspalvelin tekee tietokantahaun.

4. Käyttäjätunnus kirjataan tarkastuslokiin.

5. Järjestelmän pääkäyttäjä tarkistaa tarkastuslokin omassa selaimessaan.

Kaikkiin rajapintoihin on kiinnitettävä huomiota. On varmistettava, että vähintään yksi henkilö ymmärtää ja dokumentoi järjestelmän tietovirtoja. Kehitystiimin on lähdettävä siitä oletuksesta, että komponentteihin voi tulla haitallisia syötteitä, olipa niiden sijainti mikä tahansa.

Fuzz-testaus sopii erittäin hyvin syötteenkäsittelyn testaukseen. Sitä käsitellään tarkemmin testausosiossa.

On varmistettava, että vähintään yksi henkilö ymmärtää ja dokumentoi järjestelmän tietovirtoja.

Erilliset tehtävät

Tehtävien erottaminen toisistaan tarkoittaa esimerkiksi sitä, että yksi henkilö tekee matkakulujen korvaushakemuksen, mutta eri henkilö tarkistaa ja hyväksyy sen. Sama periaate pätee myös ohjelmistokomponentteihin. Klassinen esimerkki on tarkastuslokien käsittelyn siirtäminen eri järjestelmään lokit tuottavasta järjestelmästä, jolloin niiden turvallisuutta ei voi vaarantaa samanaikaisesti. Tietokannan toimintaa koskevia tarkastuslokeja ei pitäisi tallentaa samaan tietokantaan. Jos tietokantaan tunkeudutaan, myöskään siihen liittyviin jäljitysketjuihin ei voi enää luottaa.

Kun tehtävät jaetaan eri komponenteille, voidaan myös komponentteihin liittyviä resursseja ja valtuuksia määrittää ja eriyttää tarkemmin.

Mahdollisimman suppeat valtuudet

Kun tehtävät on jaettu eri komponenttien kesken, kyseisiin prosesseihin ja alijärjestelmiin on määritettävä mahdollisimman suppeat käyttöoikeudet. Jokaisen komponentin pitäisi suoriutua tehtävästään mahdollisimman rajallisilla käyttöoikeuksilla. Sama pätee käyttäjiin: kaikilla ei pidä olla pääkäyttäjän oikeuksia, eikä pääkäyttäjilläkään pidä olla kaikkia mahdollisia oikeuksia. Kun valtuudet määritetään mahdollisimman suppeiksi oikealla tavalla, järjestelmään kohdistuvat hyökkäykset ja hyökkääjän pääsy järjestelmän eri osiin pystytään rajaamaan tehokkaammin. Saman asian voi ajatella myös toisin päin: jos komponentti pystyy toimimaan mahdollisimman suppeilla käyttöoikeuksilla, sen turvallisuus ei ole niin kriittinen kuin tilanteessa, jossa komponentilla on pääkäyttäjän oikeudet.

VINKKI:

Turvallisuusarvioinnissa läpäisemistä ei helpota, jos joudutaan kertomaan, että kaikki koodi ajetaan laajimmilla mahdollisilla valtuuksilla. On pystyttävä kuvaamaan, millaisia valtuuksia on käytössä ja missä.

Syväsuojaus

Syväsuojaus tarkoittaa sitä, että järjestelmään suunnitellaan useita suojaustasoja. Kaikkiin turvatoimiin voi sisältyä turvallisuuden vaarantavia virheitä. Päättäväinen hyökkääjä myös keksii keinot tiettyjen turvatoimien kiertämiseen. Syväsuojaus parantaa järjestelmän turvallisuutta, sillä yhden tason vaarantuminen ei vaaranna koko järjestelmää.

Ei voida esimerkiksi olettaa, että palomuuri riittää yksin suojaamaan järjestelmää. Palomuuriin liittyy yleensä sääntöjä, joiden perusteella sen läpäiseminen sallitaan. Haittaohjelma voi pystyä ylittämään järjestelmän suojatun ulkorajan sähköpostin kautta. Sisäiseen verkkoon tavallisesti yhteydessä oleva kannettava tietokone voi saastua liikematkan aikana. Käyttäjät selailevat verkossa ja voivat joutua haittasivuston hyökkäyksen kohteeksi, jolloin hyökkääjä pääsee tunkeutumaan sisäiseen verkkoon. Käytännössä syväsuojaus tarkoittaa sitä, että palomuurin lisäksi sisäinen verkko kovennetaan, sisäinen liikenne salataan ja sisäisten palvelujen käyttöön vaaditaan käyttäjän todennus. Tällöin järjestelmä ei vaarannu, vaikka palomuurin läpi pystyttäisiinkin tunkeutumaan.

Järjestelmässä ei pidä olla tiliä, jolla päästään rajoittamattomasti järjestelmään. Sen sijaan siinä pitäisi olla erilaisia käyttäjärooleja, joilla on mahdollisimman suppeat käyttöoikeudet. Käyttäjien toimista on jäätävä järjestelmään jäljitysketju, jota ei ole mahdollista muuttaa. Ainakin sellaisten roolien käyttöön, joilla on laajemmat käyttöoikeudet, on vaadittava kaksivaiheinen todennus.

Syväsuojauksesta puhuttaessa voidaan toisinaan käyttää myös ilmaisua monitasoinen turvallisuus.

Turvallisuus vikatilanteissa

Häiriöihin ja vikoihin kannattaa varautua. Tietokonelaitteistot rikkoutuvat, verkkoyhteydet katkeavat, akut tyhjentyvät, ohjelmistot kaatuvat jne. Järjestelmä on suunniteltava siten, etteivät nämä häiriöt ja viat vaaranna järjestelmän turvallisuutta. Se voi joskus olla hankalaa. Valittavissa on kaksi eri strategiaa: ”avautuminen” (fail-open) ja ”sulkeutuminen” (fail-closed) vikatilanteessa. Pitäisikö järjestelmän sallia vai kieltää käyttö vikatilanteessa? Jos käyttöoikeudet myöntävään komponenttiin ei saada yhteyttä, voi olla viisainta kieltää käyttö. Mutta jos käytön kieltämisellä voi olla kohtalokkaita seurauksia – kuten esimerkiksi monien ihmisten vesivarojen saastuminen – valittavaa strategiaa on harkittava tarkasti. Olennaisinta on, että komponentin vikaantumisen seurauksiin kiinnitetään huomiota.

Ei liikaa luottoa ulkoisiin palveluihin

On todennäköistä, että järjestelmässä käytetään ulkopuolisten järjestelmien palveluja. Niiden taustalla oleva ohjelmisto, tilat ja henkilöstö eivät ole omassa hallinnassa. Kolmannen osapuolen palveluihin voi myös kohdistua hyökkäyksiä, joten hyökkääjälle ei kannata tarjota etenemismahdollisuuksia.

Mahdollisimman suppeiden valtuuksien ja syväsuojauksen hengessä ulkoisiin palveluihin ei kannata luottaa sokeasti. Ulkoisia palveluja on kohdeltava ulkopuolisina toimijoina, niistä tulevat tiedot on aina tarkastettava ja turvallisuuden on säilyttävä vikatilanteessa, jossa palvelu ei ole käytettävissä.

Kartoitettaessa ulkopuolisia palveluja, jotka eivät ole omassa hallinnassa, kannattaa ajatella laajasti. Yksi esimerkki ovat selainpohjaiset järjestelmät: käyttäjän selain on ulkopuolinen palvelu, joka ajaa kehittäjän omaa koodia. Käyttäjän selaimessa ajettavaan koodiin perustuviin turvatoimiin ei voi luottaa. Turvatoimet on aina toteutettava palvelimella. Sama pätee mihin tahansa järjestelmään, jonka komponentti toimii asiakasjärjestelmässä.

Salassapitoon perustuvan turvallisuuden välttäminen

Salassapitoon tai piilotteluun perustuva turvallisuus (security by obscurity) tarkoittaa sitä, että tuotteen turvallisuuden perustana on se, että sen rakenne tai toteutustapa pysyvät salassa. Valitettavasti salaisuuksia vuodetaan ja järjestelmien salaisuuksia voidaan paljastaa takaisinmallinnuksella. Salassapitoon perustuvaan turvallisuuteen ei kannata luottaa. Vaikka järjestelmässä käsiteltävät tiedot ja käyttöoikeustietueet ovat salassa pidettäviä, itse järjestelmän on kestettävä tarkastelua. Järjestelmässä ei pitäisi olla suurta määrää salaisuuksia, jotta se voi täyttää tehtävänsä. On myös otettava huomioon, miten salaisuuksia voidaan muuttaa, jos ne vuodetaan. Onko järjestelmän yksityisten avainten vaihtaminen helppoa? Kuinka helposti järjestelmän käyttäjiä voidaan pyytää vaihtamaan salasanansa, jos ne vaarantuvat?

Vaikka järjestelmän rakenteen tai lähdekoodin salassapidolla voidaan lisätä järjestelmään yksi suojaustaso, se ei yksinään riitä varmistamaan järjestelmän turvallisuutta.

Yksinkertainen järjestelmä

Kaikki ohjelmistot ja niiden komponentit voivat sisältää virheitä ja haavoittuvuuksia. Myös turvaohjelmistoissa ja -ominaisuuksissa on haavoittuvuuksia. Kun koodia on vähemmän, myös virheitä on vähemmän. Ja mitä enemmän on määritettäviä asetuksia, sitä enemmän on myös niissä tapahtuvia virheitä. Onkin pyrittävä luomaan mahdollisimman yksinkertainen järjestelmä ja suhtauduttava aina kriittisesti näennäiseen tarpeeseen saada monimutkaisempia ominaisuuksia.

Yksinkertaisen järjestelmän arvioiminen ja sen turvallisuuden varmistaminen on helpompaa kuin monimutkaisen järjestelmän. Tarpeettomien monimutkaisuuksien välttäminen vaikuttaa myös taloudelliseen tulokseen. Yksinkertaista järjestelmää on helpompi ymmärtää, joten sen kehittäminen ja ylläpitäminen on kustannustehokkaampaa.

VINKKI:

Mitä vähemmän tarkastettavaa on, sitä vähemmän turvallisuustarkastus maksaa. Kannattaa kehittää yksinkertainen tuote tai valmistautua osoittamaan, että vain osa järjestelmästä vaikuttaa kriittisesti hyväksyttäviin käyttötapauksiin.

Turvallisuusongelmien korjaaminen oikein

Kun omassa järjestelmässä havaitaan haavoittuvuus, se pitäisi korjata tai siihen pitäisi puuttua. Tällöin toimivista kehitys- ja testausprosesseista on hyötyä. Ohjelmointivirheiden korjaamisen ja uusien ohjelmistoversioiden julkaisemisen ei pitäisi aiheuttaa epävarmuutta. Jos prosessit ovat tuotekohtaisia, henkilöstö kouluttamatonta eikä testausautomaatiota ole, kaikki muutokset tuovat mukanaan riskejä ja houkutuksena voi olla ongelman jättäminen huomiotta.

Usein turvallisuusvirhe korjataan vain tietyssä järjestelmän osassa, vaikka useita vastaavia korjauksia pitäisi tehdä myös muualla koodikannassa. Syynä tähän voi olla mahdollisten sivuvaikutusten pelko, ja sen vuoksi korjataan vain vähin välttämätön. Se, että korjataan vain yksi ongelma, vaikka korjattavana olisi monta muuta vastaavaa, ei paranna turvallisuutta merkittävästi. Toinen tyypillinen tilanne on, että toimittaja korjaa ohjelmiston ylläpitoversion, mutta jättää ongelmat huomiotta seuraavassa

julkaisussa, jolloin ne ovat edelleen mukana seuraavassa suuremmassa päivityksessä. Joskus toimittajat tekevät tuotteen ylläpitoversioon pienimmät välttämättömät muutokset ja jättävät laajemman ”korjauskierroksen” tai refaktoroinnin seuraavaan isompaan julkaisuun.

Ohjelmointivirheitä, jotka on korjattava viipymättä, voi tulla ilmi myös silloin, kun kehitystiimin tärkeimmät jäsenet ovat lomalla tai sairaana. Tämänkin vuoksi on tärkeää, että käytetyt prosessit ja koulutus ovat toimivia ja riittäviä, jotta paikalla oleva henkilöstö pystyy huolehtimaan virheiden korjaamisesta heti, kun se on tarpeen.

Ohjelmointivirheiden korjaamisen ja uusien ohjelmistoversioiden julkaisemisen ei pitäisi aiheuttaa epävarmuutta.

ALUSTAN VALINTA

Kehitettävä tuote toimii yhdellä tai useammalla eri alustalla. Tunnetuimpia ovat muun muassa Linuxin palvelinjärjestelmät, Android-mobiilialusta, taustajärjestelmien pilvialustat, alijärjestelmille tarkoitettu Docker, Microsoft Windows ja .NET. Kaikilla alustoilla on omat turvallisuusominaisuutensa, ja on tärkeää tuntea oman tuotteen taustalla olevan alustan ominaisuudet.

Alustoille julkaistaan aika ajoin uusia ominaisuuspäivityksiä, kun niissä huomataan turvallisuusongelmia. Tämä on otettava huomioon tuotteen julkaisusykliä ja elinkaarta suunniteltaessa. Alustaa valittaessa kannattaa tutustua eri alustojen julkaisuhistoriaan, jotta saa kuvan niiden turvallisuus- ja päivityshistoriasta. Kun alusta on valittu, on päätettävä oman tuotteen päivittämisestä alustan päivitysten mukaan: tehdäänkö kaikki päivitykset, tärkeimmät päivitykset, vain turvallisuuspäivitykset vai ei mitään niistä?

On selvää, että turvallisuuspäivitysten huomiotta jättäminen tuo mukanaan ongelmia. Avoimen lähdekoodin alustassa on mahdollista tehdä vain oman järjestelmän kannalta hyödylliset päivitykset ja siis ylläpitää omassa käytössä olevaa alustan osa-aluetta. Tällöin on muistettava, että turvaohjelmat saattavat varoittaa ongelmista virheellisesti, ja tarkastuksessa on pystyttävä osoittamaan, että haavoittuvuudet on korjattu.

VINKKI:

Turvallisuustarkastuksessa kohteena ovat myös taustalla olevat alustat. Tarkastuksessa varmistetaan, että alustan turvallisuusominaisuudet ovat käytössä, ja tarkastetaan alusta siihen liittyvien tyypillisten ongelmien varalta. Yksi tavallinen turvallisuusarviointiin sisältyvä vaatimus on, että tuotteen toiminnallisuuden kannalta epäolennaiset alustan ominaisuudet on poistettu käytöstä ja poistettu, jos mahdollista. Tarkastuksessa on myös kuvattava tuotetta koskeva päivityskäytäntö ja osoitettava, että alustan päivityksiä seurataan tuotteen turvallisuuden varmistamiseksi.

OHJELMISTOKOMPONENTIT

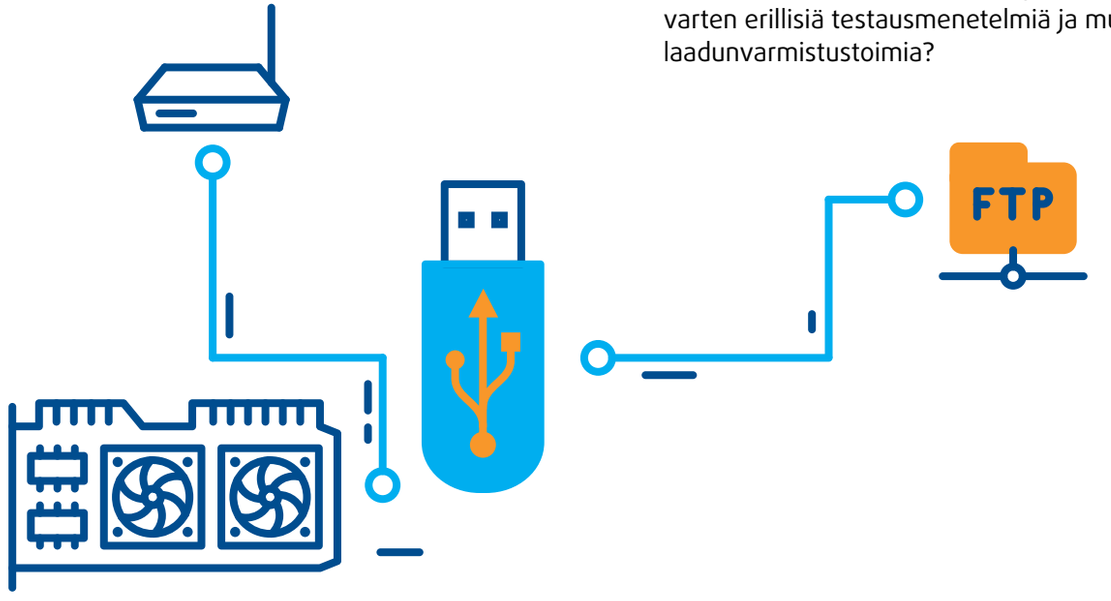
Tuotteeseen sisältyy erilaisia komponentteja. Komponentit vaihtelevat monimutkaisuusasteeltaan: on asiakasohjelmia, taustajärjestelmiä, tietokantoja, VPN-päätepisteitä, satunnaislukugeneraattoreita, ohjelmistokirjastoja jne. Jotkin komponenteista toteutetaan omana työnä tai alihankintana, jotkin ostetaan ja toiset ovat vapaasti käytettäviä tai perustuvat avoimeen lähdekoodiin. On erittäin tärkeää tuntea omassa tuotteessa käytössä olevat komponentit. Sovelluksen komponenttiluetteloa voidaan kutsua myös tuoterakenteeksi.

Jotkin komponentit ovat toisia kriittisempiä turvallisuuden kannalta. Tavallisesti liiketoiminnan ydinlogiikkaan tai itse turvallisuuteen liittyviin komponentteihin kiinnitetään erityistä huomiota. Valitettavasti esimerkiksi erittäin turvalliseen viestiohjelmaan sisältyvä tavallinen kuvankäsittelykirjasto voi aiheuttaa kohtalokkaan haavoittuvuuden luodessaan pikkukuvan yhteyttä ottavasta henkilöstä. Onkin suositeltavaa käsitellä kaikkia komponentteja turvallisuuden kannalta kriittisinä ja yksinkertaistaa tuotetta poistamalla sen kannalta toisarvoiset komponentit kokonaan.

Toinen esimerkki komponentista, jolla ei vaikuta olevan suurta merkitystä turvallisuuden kannalta, on asennuskoodi. Kuitenkin jos asennusohjelma joutuu hyökkäyksen kohteeksi, sitä voidaan käyttää väärin ja asentaa sen avulla virheellinen ohjelmistoversio.

Kun järjestelmän arkkitehtuuri ja komponentit on määritetty, voidaan arvioida kunkin komponentin turvallisuusominaisuuksia:

- Komponentit hyödyntävät eri teknologioita ja alustoja. Ollaanko tietoisia niiden vaikutuksista tuotteen turvallisuuteen?
- Eri komponenteilla on eripituiset päivitysvälit. Kuinka komponenttien päivitykset synkronoidaan tuotteen päivitysten kanssa?
- Eri komponenteille on määritettävä erilaiset käyttöoikeudet. Kuinka komponentteihin voidaan soveltaa mahdollisimman suppeiden valtuuksien periaatetta?
- Komponenttien toiminnallisuudet voivat poiketa paljonkin tuotteen ydintoiminnallisuudesta. Tarvitaanko komponentteja varten erillisiä testausmenetelmiä ja muita laadunvarmistustoimia?



Toimitusketjut

Kolmannen osapuolen komponentit muodostavat ohjelmiston toimitusketjun. Komponenttien toimittajilla on myös todennäköisesti omia toimittajiaan, joten toimitusketju voi olla pitkäkin. Kun komponentteja päivitetään, päivitykset siirtyvät vähitellen eteenpäin toimitusketjussa omaan tuotteeseen asti, jolloin on tehtävä päätös päivitysten tekemisestä.

OpenSSL-salauskirjasto on hyvä esimerkki suositusta, mutta haastavasta toimitusketjuun sisältyvästä komponentista. Sitä toimitetaan monien eri sulautettujen järjestelmien, sovellusten, palvelujen sekä muiden komponenttien, ohjelmointikielten ja alustojen kylkiäisenä.

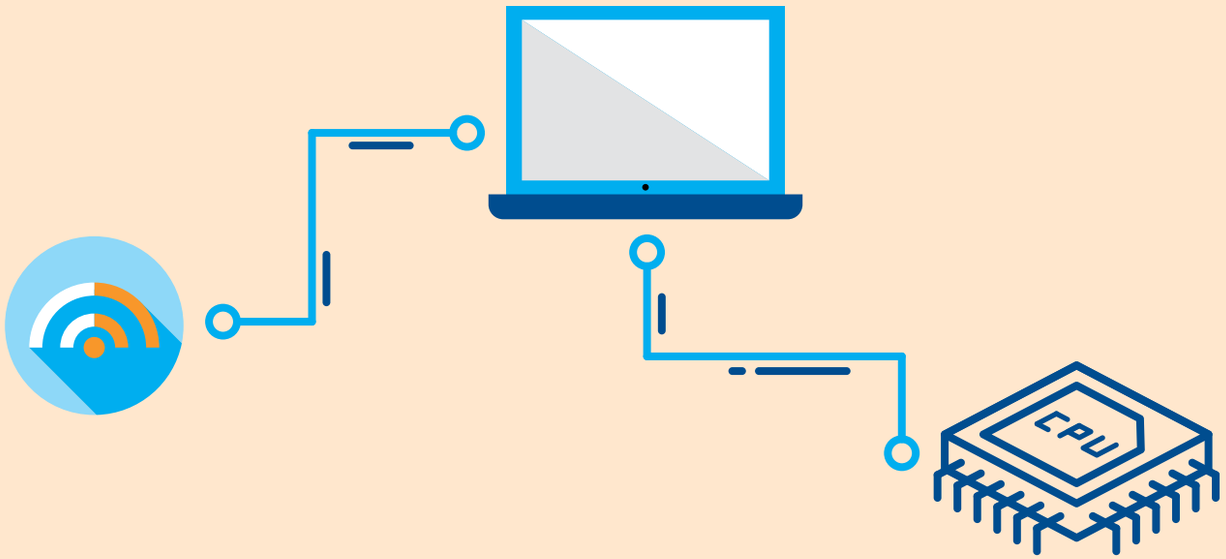
Tämän oppaan kirjoittajat ovat toistuvasti nähneet tuotteita, joihin sisältyy useita eri versioita OpenSSL-kirjastosta. OpenSSL-kirjastoa ylläpidetään aktiivisesti, ja siitä on löydetty usein haavoittuvuuksia, joten sitä myös päivitetään usein. Jos tuotteessa on käytössä TLS-protokolla tai X.509-varmenteet, on hyvin todennäköistä, että myös OpenSSL on käytössä suoraan tai epäsuorasti.

VINKKI:

Turvallisuustarkastuksessa kiinnitetään huomiota myös komponenttien toimitusketjuun. Kannattaakin valmistautua esittelemään järjestelmän tuoterakenne, myös mahdolliset laitteistokomponentit. Tarkastajilla on työkaluja, joilla he voivat tarkistaa, vastaako esitetty komponenttiluettelo tuotteen todellisia komponentteja. Vanhoista, tarpeettomista, maineeltaan kyseenalaisista tai muutoin epätavallisista komponenteista pyydetään luultavasti lisäselvityksiä, joihin on pystyttävä vastaamaan.

Lisälukemista

- *OWASP: Security by Design Principles*



TURVALLINEN OHJELMOINTI

Ohjelmoinnin avulla suunnitelman pohjalta luodaan sovellus, joka toivottavasti täyttää määritetyt vaatimukset. Valitettavasti myös ohjelmointivaiheessa järjestelmään voi melko helposti tulla haavoittuvuuksia. Hyvä puoli on se, että ongelma tunnetaan hyvin, joten turvallisen ohjelmoinnin tueksi on saatavilla monenlaista apua, oppaita ja koulutusta. On tärkeää olla selvillä käytettävän alustan ja ulkoisten komponenttien turvallisuuden tilasta. On myös tärkeää olla tietoinen ohjelmointikielen ja muiden valittujen työkalujen vaikutuksista. Turvallisen koodin kirjoittaminen on mahdollista, mutta se on yhtä vaikeaa kuin virheettömän koodin kirjoittaminen. Siksi onkin hyödyllistä tarkastella koodeja turvallisuuden kannalta koodikatselmuksissa.

Koodin on oltava hyvin dokumentoitua, modulaarista, luettavaa, testattavaa ja testattua. Ajan kuluessa koodia on nimittäin pystyttävä ylläpitämään ja korjaamaan sen turvallisuutta vaarantamatta.

Koodin on oltava hyvin dokumentoitua, modulaarista, luettavaa, testattavaa ja testattua

Staattisten analysointityökalujen toimintaperiaate on se, että turvallisuusongelmat ja muut virheet löydetään automaattisesti lähdekoodia analysoimalla. Niistä on paljon apua, ja niistä on saatavilla sekä ilmaisia että kaupallisia versioita useimmille ohjelmointikielille. Näiden työkalujen huono puoli on se, että monet niistä varoittavat usein aiheettomasti virheistä, esimerkiksi koodikonstruktioista, jotka eivät oikeasti ole virheitä. Tuotekehityksessä on siis varattava aikaa ja vaivaa staattisen koodianalyysin tekemiseen etenkin silloin, kun koodikanta on suuri, eikä sitä ole analysoitu aiemmin.

SALAUS

Lähes kaikkiin järjestelmiin ja tuotteisiin tarvitaan salaustoimintoja, sillä niiden avulla lähetetään tai tallennetaan luottamuksellisia tietoja, todennetaan käyttäjiä ja palveluja ja niin edelleen.

Salaus on yksi turvallisuuden kulmakivistä. On ratkaisevan tärkeää, että salaus tehdään oikein. Ensimmäinen sääntö on se, että pyörää ei pidä keksiä uudelleen. Tuotteessa kannattaa hyödyntää hyvin tunnettuja ja korkealaatuisia salauskirjastoja ja välttää omatekoisia kryptoprimitiivejä. On aina noudatettava standardeja ja parhaita käytäntöjä itse luotujen käytäntöjen sijaan.

Yksi tavallinen virhe on huonon satunnaislukugeneraattorin käyttäminen. Moniin salaustoimintoihin vaaditaan kryptografisesti vahvoja satunnaislukuja, eikä niiden tuottaminen ole helppoa. Kannattaa selvittää huolellisesti, millaisilla menetelmillä käytettävälle alustalle voidaan tuottaa hyviä satunnaislukuja luotettavasti.

Salaustoimintojen käytössä tapahtuu helposti virheitä. Esimerkiksi monissa sovelluksissa varmenteiden tarkistus ei toimi luotettavasti (katso esimerkkejä blogikirjoituksesta Common x509 certificate validation/creation pitfalls). On siis suositeltavaa hyödyntää hyvin tunnettuja tekniikoita ja selvittää, miten niitä käytetään oikein.

VINKKI:

Turvallisuustarkastuksessa keskitytään erityisesti tuotteen salaustoimintoihin. Tuotteen suunnittelu ja koodi arvioidaan perusteellisesti, ja myös salauskoodille tehdään katselmus ja virheenjäljitys, kun sitä suoritetaan. Kryptografisesti turvallisten satunnaislukujen tuottamistapa tarkistetaan varmasti. Tarkastuksessa on selvitettävä, mihin standardeihin tai tunnettuihin ratkaisuihin salaustoiminnot perustuvat.

Suomessa sovellettavat kryptografiset vahvuusvaatimukset löytyvät osoitteesta: https://www.viestintavirasto.fi/attachments/tietoturva/Kryptografiset_vahvuusvaatimukset_-_kansalliset_suojaustasot.pdf. Hyvät yleisohjeet ovat luettavissa osoitteessa: http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf. Kyberturvallisuuskeskuksen NCSA-FI-toiminnolta voi pyytää yksityiskohtaisempia, omia käyttötapauksia koskevia ohjeita.

RIIPPUVUUKSIEN HALLINTA

Nykyään ohjelmistokehityksessä on usein enemmän kyse komponenttien kokoamisesta kuin oman ohjelmiston kirjoittamisesta. Ohjelmointiympäristöjen kehityksessä ohjelmoijien on entistä helpompaa liittää kolmannen osapuolen komponentteja tuotteisiin.

Ilmaisia avoimen lähdekoodin komponentteja on saatavilla valtava määrä kaikille tärkeimmille alustoille ja ohjelmointikielille. Salaustoiminnot, erimuotoisten tietojen jäsennys ja järjestelmien välinen integrointi onnistuvat yleensä parhaiten komponentteja liittämällä. Myös ilmaisille komponenteille voi olla kaupallisia tukipalveluja saatavilla.

Sille, miten komponentit hyväksytään käyttöön, pitäisi olla olemassa käytäntö tai sovittu prosessi. Kyseisiä päätöksiä ei pidä jättää yksittäisten kehittäjien tehtäväksi. Jokainen komponentti voi tuoda mukanaan uusia haavoittuvuuksia. Lisäksi on oltava tietoa käytössä olevien kolmansien osapuolten komponenttien lisensseistä.

Joskus ulkoisissa riippuvuuksissa on kyse vain koodin kopioimisesta ja liittämisestä. Ohjelmoijien on ratkaistava monimutkaisia ongelmia, ja usein ratkaisu voi löytyä verkosta koodinpalasena. Se, että jokin on ”verkosta löydetty”, ei kuitenkaan takaa sen turvallisuutta ja voi tuoda mukanaan yllätyksiä lisenssien kannalta. Sellaista, mitä ei ymmärrä, ei kannata kopioda.

KOODIKATSELMUKSET

Katselmusten avulla voidaan varmistaa, että turvallista ohjelmointia koskevaa ohjeistusta on noudatettu. Kun huolehditaan myös muista laatua määrittävistä tekijöistä, kuten esimerkiksi lähdekoodin kommenteista ja hyvistä nimeämiskäytännöistä, päivitysten toimittaminen ei ole niin työlästä. Niiden avulla tieto siirtyy myös kehitystiimin jäsenten välillä, mikä varmistaa jatkuvuuden.

VINKKI:

Perusteelliseen turvallisuustarkastukseen kuuluu lähdekoodikatselmus. Se, että lähdekoodi ei ole luettavaa ja dokumentoitua eikä sille tehdä versionhallintaa, toimii jo itsessään varoitussignaalina. Tarkastajat eivät lue ja ymmärrä lähdekoodia kokonaisuudessaan, mutta he etsivät ja tarkastavat koodista kriittisinä pitämiään osia oman kokemuksensa, tuotteen suunniteltujen käyttötilanteiden, uhkamallin ja muiden tekijöiden perusteella. Jos kyseisten osien löytäminen on vaikeaa koodin sekavuuden tai jopa huonon laadun takia, tarkastuksen läpäiseminen voi hankaloitua.

Pelkästään koodikatselmuksen perusteella ei voida yleensä varmistaa, että ohjelmisto toimii oikein tai että se ylipäänsä toimii. Tarkastuksessa saatetaan pyytää tukea koodin ajamiseen ja virheenjäljitykseen. Esimerkiksi salaustoimintoja voidaan usein käyttää väärin tai jättää käyttämättä, joten tarkastuksessa saatetaan käydä läpi toiminnon suorittaminen vaihe kerrallaan. Tähän voidaan tarvita tuotteen kehittäjän apua, varsinkin jos käytössä on tuntemattomampia laitteisto- tai ohjelmistokomponentteja. Tarkastuksessa on myös selvitettävä koodin rakentamis- ja kääntämistapa, jotta itse rakentamisprosessin turvallisuutta voidaan arvioida.

JATKUVA INTEGROINTI

Jatkuva integrointi tarkoittaa, että ohjelmistosta tehdään uusia koontiversioita säännöllisesti. Jos jatkuva integrointi on automatisoitu, tuotteesta tehdään uusi koontiversio aina, kun lähdekoodiin tehdään pysyvä muutos. Prosessiin voi sisällyttää myös automatisoidut testit, joiden avulla kehittäjät saavat välitöntä palautetta mahdollisesti tekemistään virheistä. Jatkuvan integroinnin ja automatisoitujen testien avulla virheet voidaan korjata heti, kun niitä syntyy, ja koontiversioiden luominen on varmempaa.

Jatkuvaa integrointia noudattavan ympäristön luominen on investointi, jota kannattaa harkita. Vaikka jatkuva integrointi ei olisikaan täysin automatisoitua, on suositeltavaa pyrkiä siihen, että tuotteesta voidaan luoda koontiversioita usein ja että automatisoituja testejä tehdään mahdollisimman paljon.

Lisälukemista:

- *Turvallisen koodauksen oppaita*
  - *OWASP*
  - *SEI CERT Coding Standards*
- *Tietoa turvallisemmista kääntämisvaihtoehdoista tietyissä järjestelmissä*
  - *C-Based Toolchain Hardening (Microsoft and GCC)*
  - *Debian Hardening*
  - *Microsoft - Security Best Practices for C++*

TESTAUS JA TODENTAMINEN

Testauksen avulla tarkistetaan, että toteutettu järjestelmä vastaa määritettyjä vaatimuksia. Tämä koskee sekä muistiin kirjattuja, julki lausuttuja vaatimuksia, että epäsuorasti ilmaistuja vaatimuksia. Ohjelmiston ei esimerkiksi pitäisi kaatua, vaikka tätä ei olisikaan kirjattu erikseen vaatimuksiin.

Myös negatiivisiin testeihin kannattaa kiinnittää huomiota eli testata asioita, jotka ei pitäisi onnistua.

Järjestelmän turvallisuusominaisuudet sekä muut tärkeät ominaisuudet on testattava. Myös negatiivisiin testeihin kannattaa kiinnittää huomiota eli testata asioita, joiden ei pitäisi onnistua. Koska tavoitteena on korkealaatuinen järjestelmä, testausautomaatiota tarvitaan, sillä manuaalisella testaamisella ei voida testata tehokkaasti suurempia järjestelmiä, joista tehdään säännöllisesti uusia koontiversioita. Seuraavassa esitellään lyhyesti vaaditut testauksen tasot ja osa-alueet.

Yksikkötestauksella tarkoitetaan automatisoituja, kehittäjän suorittamia testejä, joiden kohteena on yksittäiseen komponenttiin sisältyvä koodin osa. Nämä testit ovat kehittäjien itsensä vastuulla, joten havaittujen virheiden korjaamisen pitäisi tapahtua nopeasti ja edullisesti. Koodikatselmuksessa voidaan varmistaa, että kehitetyt yksikkötestit kattavat pääosan koodista.

Komponenttitestauksessa komponentti suoritetaan eristyksissä ja mahdollisesti niin, että muut simuloidut komponentit edustavat järjestelmää kokonaisuudessaan. Komponenttitestit voivat olla testaustiimin suunnittelemia. Jos mahdollista, komponenttitestien pitäisi olla automatisoituja, ja ne pitäisi suorittaa kerran vuorokaudessa.

Järjestelmätestauksessa testataan koko järjestelmän koontiversio käyttämällä sitä. Järjestelmätesteihin voi kuulua manuaalisia testaustoimia, joten ne voivat olla aikaa vieviä ja kalliita yksikkö- ja komponenttitesteihin verrattuna. Järjestelmätestejä voidaan myös automatisoida, mutta automatisointi edellyttää sitä, että testausinfrastruktuuriin on tehtävä mittavia investointeja.

Hyväksymistestauksen suorittaa riippumaton testaustiimi, asiakas tai kolmas osapuoli. Jos hyväksymistestauksessa havaitaan merkittäviä ongelmia, tuotteeseen voidaan joutua tekemään suuria muutoksia ja hyväksymistestit voidaan joutua tekemään uudelleen, mikä voi olla hyvin kallista ja aikaa vievää.

Staattinen testaus suoritetaan itse tuotetta käyttämättä ja tarkastamalla eri komponentteja, kuten lähdekoodia ja binääritiedostoja.

- Koodikatselmuksia ja -tarkastuksia voidaan pitää staattisina testeinä.
- Automatisoitu lähdekoodianalyysi on staattista testausta.
- Yksi verrattain uusi menetelmä on ohjelmistokoostumusanalyysi (software composition analysis), jossa tarkastelun kohteeksi otetaan käännetty binääritiedosto sekä sen kokoamiseen käytetyt komponentit. Prosessi tuo esiin tuotteen ulkoiset riippuvuudet, joita voi olla muutoin hankala selvittää varmasti, jos tuotetta on ollut kehittämässä monia henkilöitä ja siinä on paljon moduuleja.

Dynaamisessa testauksessa tuotetta testataan sitä käyttämällä ja tarkastetaan sen toiminta.

- Perinteisesti sillä on tarkoitettu manuaalisia tai automaattisia testejä, joilla todennetaan, että tuote on asetettujen vaatimusten mukainen.
- Fuzz-testaus on turvallisuusnäkökulmaa painottava dynaaminen testausmenetelmä.
- Nykyään tuote on testattava myös turvallisuusvaatimusten osalta, ja dynaamisiin testeihin pitäisi sisällyttää tuotteeseen kohdistuvat hyökkäys- ja väärinkäyttöyritykset.
- Kuormitustestaus on dynaamista testausta, jossa keskitytään tuotteen suorituskykyyn.

VINKKI:

Turvallisuustarkastuksessa tuote voidaan testata esimerkiksi seuraavasti:

1. Käyttöoppaan lukeminen tai pyyntö saada tuotetta koskevaa koulutusta.
2. Tuotteen asetusten määrittäminen suunniteltua käyttöä varten.
3. Tuotteen käyttäminen suunniteltujen käyttötilanteiden mukaisesti.
4. Kaikkien käytettävissä olevien valikkojen ja valintaikkunoiden valitseminen.
5. Pääkäyttäjän toimien kokeileminen tuotteen suunnitellun käytön mukaisesti.
6. Tuotetta koskevien poikkeus- ja kuormitustilanteiden luominen (näitä käsitellään lisää jäljempänä).

Tarkastajat myös vertaavat näkemäänsä uhkamalliin ja päivittävät sitä tarvittaessa. Tarkastuksen tavoitteena on tärkeimpien uhkien kartoittaminen riippumatta siitä, vastaavatko ne alkuperäistä vaatimusmäärittelyä.

FUZZ-TESTAUS

Fuzz-testaus on turvallisuusnäkökulmaa painottava testausmenetelmä, jossa virheitä etsitään altistamalla tuote odottamattomille ja virheellisille syötteille.

Fuzz-testausta voi tehdä ilman pääsyä tuotteen lähdekoodiin. Fuzz-testaus tuo ongelmat esille melko varmasti, ja ne kannattaakin havaita ennen muita. Fuzz-testaus on myös automatisoitavissa.

Fuzz-testauksella löydetään haavoittuvuudet. Tavallisesti ensimmäinen merkki haavoittuvuudesta on järjestelmän kaatuminen tai palvelunestotila, mutta sillä voi olla vakavampiakin seurauksia. Taidokkaan syötteen avulla hyökkääjä voi ottaa järjestelmän hallintaansa, jolloin kyseessä on haavoittuvuuden hyväksikäyttö. Fuzz-testaukseen on saatavilla erilaisia ilmaisia ja kaupallisia ratkaisuja. (Lisälukemista-kohdassa on niistä lista). Ratkaisuja kannattaa käyttää, sillä myös hyökkääjät käyttävät niitä varmasti.

Fuzz-testaus tuo ongelmat esille melko varmasti, ja ne kannattaakin havaita ennen muita.

VINKKI:

Turvallisuustarkastuksissa hyödynnetään usein fuzz-testausta, sillä se on helppo toteuttaa ilman syvällisempää tietoa tuotteen rakenteesta, ja silti sen avulla voidaan havaita ongelmia tehokkaasti. Useimmille, ellei kaikille tunnetuimmille alustoille on tehty kattavaa fuzz-testausta, joten tarkastuksessa ei luultavasti toisteta alustojen fuzz-testausta. Fuzz-testauksen kohteena ovat todennäköisimmin itse kehitetyt rajapinnat ja epätavallisemmat komponentit.

PENETRAATIOTESTAUS

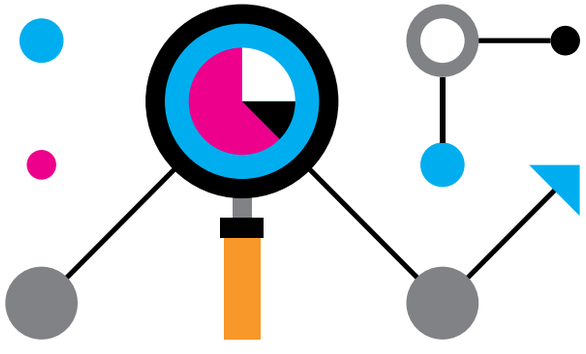
Penetraatiotestaus on tehtävään nimettyjen turvallisuusasiantuntijoiden suorittamaa turvallisuustestausta, jossa he pyrkivät tunkeutumaan palveluun tai järjestelmään ja havaitsemaan siihen sisältyviä turvallisuusongelmia. Penetraatiotestausta käytetään paljon, ja penetraatiotestaajat voivat antaa puolueettoman mielipiteen järjestelmän turvallisuuden tilasta.

Penetraatiotestaus tehdään kuitenkin manuaalisesti, joten yleensä sitä ei voi tehdä jokaiselle tuote- tai koontiversiolle. Se ei myöskään aina ole laajuudeltaan tai syvyydeltään johdonmukaista, minkä vuoksi se sopii paremmin hyväksymistestaukseen.

STRESSITESTAUS

Hyökkääjä voi pyrkiä löytämään tuotteesta haavoittuvuuksia kohdistamalla siihen poikkeuksellista kuormitusta tai poikkeustilanteen. Usein tällaista on mahdoton estää, joten tämänkin vuoksi järjestelmän olisi täytettävä edellä käsitellyt turvallisten vikatilanteiden ja syväsuojauksen periaatteet. Esimerkiksi seuraaviin kysymyksiin kannattaa miettiä vastauksia:

- Mitä tapahtuu laitteen käynnistyessä? Onko käynnistyksen aikana hyökkäysvektoreita, joita ei tunneta (esimerkiksi järjestelmävalikon avaaminen tietyllä näppäinyhdistelmällä), tai hyväksyykö laite laiteohjelmistopäivityksen keneltä tahansa käyttäjältä käynnistyksen aikana? (Tällaista on tapahtunut.)
- Mitä tapahtuu, kun verkkoyhteys katkeaa? Kaatuuko järjestelmä, tai meneekö se turvattomaan tilaan?
- Mitä tapahtuu, jos tulee sähkökatko ja laite käynnistyy uudelleen sen päätyttyä?



VINKKI:

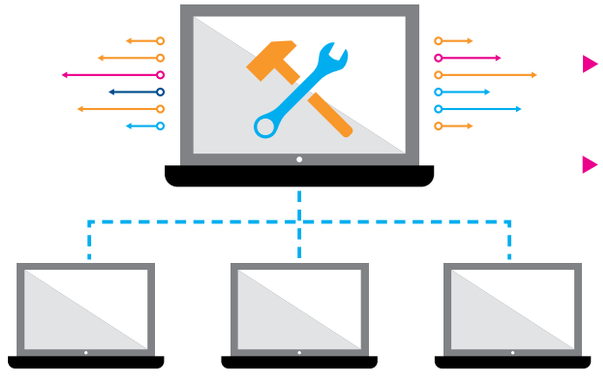
Turvallisuustarkastuksessa luodaan tuotetta koskevia kuormitustilanteita. Kyseisenlaisten tilanteiden simuloiminen on yleensä helppoa (esim. pistokkeen irrottaminen), mutta niiden avulla saadaan tehokkaasti tietoa tuotteen toiminnasta.

TAKAISINMALLINNUS

Tämän oppaan aiemmassa luvussa todettiin, että tuotteen turvallisuuden ei pitäisi perustua ohjelman tai sen algoritmien salassapitoon (security by obscurity). Yksi hyvä syy siihen on se, että ohjelmatiedostojen, laiteohjelmistojen ja verkkoliikenteen takaisinmallintamiseen on olemassa monia työkaluja. Ei ole realistista olettaa, että järjestelmän suunnittelun yksityiskohdat pysyvät salassa.

Takaisinmallinnukseen voi tutustua tarkemmin takaisinmallintamalla omia tuotteitaan. Esimerkiksi seuraavista perustyökaluista voi olla hyötyä:

- **Wireshark** verkkoliikenteen nuuskimiseen ja analysoimiseen
- **Nmap** isäntäkoneiden, avointen porttien ja palvelujen etsintään verkosta
- **Strings** merkkijonojen tulostukseen mistä tahansa tiedostosta (esim. ohjelmatiedostot ja laiteohjelmistot).



VINKKI:

Turvallisuustarkastuksessa takaisinmallinnusta hyödynnetään sen varmistamiseksi, että kehittäjän tuotetta koskevat väitteet pitävät paikkansa. Mitä komponentteja tuotteessa on oikeasti käytössä? Vastaavatko tuotteen verkkoskannauksen tulokset vaadittuja verkkopalveluja? Onko verkkoliikenne väitetyn mukaista? Ovatko fyysiset turvallisuuskomponentit väitetyn mukaisia?

YHTEENVETO TESTAUKSESTA

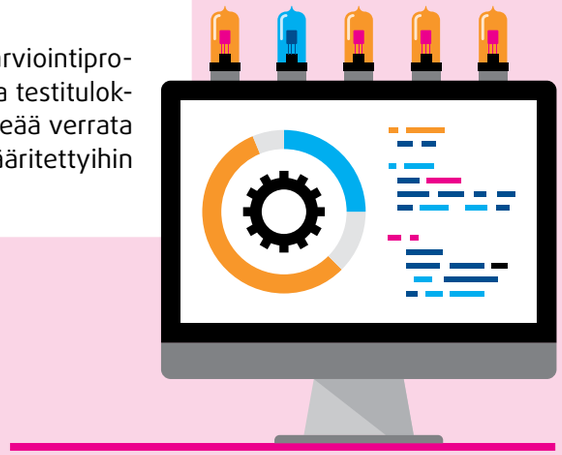
Kukaan ei tahdo tulla hylätyksi hyväksymistestauksessa. Tuotteen hylkääminen voidaan välttää parhaiten perehtymällä hyväksymistesteihin ja tekemällä perusteellisempaa testausta omissa tiloissa ennen hyväksynnän hakemista. Järjestelmä voidaan antaa myös kolmannen osapuolen testattavaksi ennen hyväksymistestausta.

Seuraava tilanne on kaikille tuttu: Järjestelmä toimii täydellisesti ”omalla koneella” tai omassa laboratoriossa, mutta lakkaa toimimasta, kun sitä kokeillaan muualla.

On erittäin suositeltavaa välttää tilannetta, että järjestelmä viedään kehitystilojen ulkopuolelle ensimmäistä kertaa silloin, kun sille tehdään hyväksymistestaus. Se voidaan välttää esimerkiksi käyttämällä kolmannen osapuolen testaajia. He voivat olla esimerkiksi testauksesta kiinnostuneita beeta-asiakkaita, jotka voivat käyttää tuotetta todellisessa käyttötilanteessa ja antaa siitä palautetta.

VINKKI:

Hyvin dokumentoitu testaus nopeuttaa arviointiprosessia. Kolmannelta osapuolelta saaduista testituloksista on myös apua arvioinnissa. On tärkeää verrata testaussuunnitelmaa ja -tuloksia määritettyihin turvallisuusvaatimuksiin.



Lisälukemista

- *OWASP: Web Application Security Testing Cheat Sheet*
- *Wikipedia: Penetration Testing*
- *Wikipedia: Stress testing*
- *OWASP: Fuzzing (with some tool references)*
- *MICKAEL DORIGNY, päivitetty 19/10/2016: What is hardening*

KÄYTTÖÖNOTTO

Kun tuote on valmis ja myyty asiakkaalle, se on otettava käyttöön. Aiemmin käyttöönotto tapahtui siten, että asiakkaalle toimitettiin asennuslevy tai asiakkaan tiloihin lähetettiin järjestelmän asentaja. Nykypäivänä järjestelmät asennetaan verkon välityksellä tai ainakin ensiasennuksen jälkeen tehtävät päivitykset tehdään verkon välityksellä. Pilvipalveluja ei asenneta lainkaan, vaan niitä käytetään verkkoselaimella.

Nykyään sovellus voidaan toteuttaa myös konttikuvana (container image), jollainen on esimerkiksi Docker-kuva. Kontit voivat sisältää suuren määrän komponentteja, joiden turvallisuudesta ja toimitusketjusta on oltava selvillä. Laitteistokomponentteja sisältävien erittäin turvallisten tuotteiden kohdalla on myös otettava huomioon laitteistoon mahdollisesti kohdistuvat peukalointiyritykset.

Asiakkaita on opastettava tuotteen turvalliseen elinkaarenhallintaan.

Kun asiakkaalle toimitetaan yhteys latausportaaliin, on varmistettava, ettei portaalin latauskuvakkeiden turvallisuus voi vaarantua. Lisäksi on varmistettava, ettei asiakas vahingossa yritä ladata tuotetta väärennetystä latausportaalista. Portaali-palvelimen turvallisuudesta on huolehdittava asianmukaisesti ja on aina käytettävä salattuja verkkosivuja, joiden varmenteet ovat ajan tasalla. Jos tuote lataa itse päivityksiä tai liitännäisiä, sen on aina tarkistettava, että lataukset ovat peräisin oikealta palvelimelta. Myös allekirjoitettuihin päivityksiin voi kohdistua hyökkäyksiä, joissa käyttäjiä houkutellaan lataamaan päivitys, joka on todellisuudessa tuotteen vanhempi ja haavoittuvampi versio.

Tuotteen käyttöönotto asiakkaalle voidaan toteuttaa esimerkiksi Linuxia tai muuta käyttöjärjestelmää käyttävän laitteen tai virtuaalikuvan muodossa. Oli käytettävä alusta mikä tahansa, sen kovennuksesta on huolehdittava asianmukaisesti. Käytännössä se tavallisesti tarkoittaa, että poistetaan käytöstä tai jopa poistetaan kaikki palvelut, jotka eivät ole välttämättömiä tuotteen toiminnan kannalta. Alustat sisältävät yleensä omia valinnaisia turvallisuusominaisuuksiaan, joita voi ottaa käyttöön tai poistaa käytöstä tarpeen mukaan.

Niitä voi tapahtua esimerkiksi valmistuspaikassa, tuotteen kuljetuksessa valmistajalta ohjelmistokehittäjälle tai kuljetuksessa ohjelmistokehittäjältä asiakkaalle. Tuotteen elinkaaren loppupäässä on mietittävä, mitä laitteistolle tapahtuu käytöstä poistamisen jälkeen. Tuotteen sisältämät levyt voivat esimerkiksi sisältää arkaluonteisia tietoja, jotka on tuhottava. Asiakkaita on opastettava tuotteen turvalliseen elinkaarenhallintaan.

YLLÄPITO JA KORJAUSPÄIVITYKSET

Kun tuote on hyväksytty, myyty ja asennettu, on tarpeen huolehtia sen ylläpidosta. Ennen pitkää tuotteesta löytyy luultavasti haavoittuvuus, joka todennäköisesti sisältyy tuotteen alustaan integroituun, kolmannen osapuolen toimittamaan komponenttiin. Ongelmaan on reagoitava nopeasti ja huolellisesti.

Jos tuotteen tietylle versiolle on hankittu hyväksyntä, version päivittäminen saattaa mitätöidä hyväksynnän. Toisaalta jos päivitystä ei tehdä, haavoittuvuus vaarantaa asiakkaiden turvallisuuden.

Tällaisiin tilanteisiin on varauduttava, ja niistä on keskusteltava hyväksyjän kanssa. Joitakin huomioon otettavia näkökohtia:

- Pienille muutoksille on helpompi saada hyväksyntä kuin suurempia kokonaisuuksia koskeville muutoksille, joiden lopullista laajuutta voi olla hankala arvioida.

- Joidenkin komponenttien muuttaminen voi olla arveluttavampaa hyväksynnän kannalta kuin toisten. Esimerkiksi käyttöliittymäkomponentteihin voi olla mahdollista tehdä päivityksiä ilman uutta hyväksyntää, kun taas salausmoduulin muokkaaminen ei välttämättä onnistu ilman sitä.

- Jos ohjelmistokehittäjällä on näyttöä siitä, että sen käytössä olevat kehitys-, todennus- ja julkaisuprosessit ovat luotettavia, hyväksyjä ei välttämättä ole niin huolissaan tuotteen päivityksistä mahdollisesti aiheutuvista haitallisista sivuvaikutuksista.

Joka tapauksessa ei kannata joutua tilanteeseen, jossa asiakas tiedustelee hiljattain paljastuneen haavoittuvuuden mahdollisista vaikutuksista omaan järjestelmään eikä hänelle osata antaa vastausta. Tällaiset tilanteet voidaan välttää, kun tunnetaan oman ohjelmiston ja sen toimitusketjun jokainen osa.

Vielä parempaa on, jos asiakkaille voidaan kertoa tuotteeseen vaikuttavista haavoittuvuuksista ennakoivasti. Ohjelmistokehittäjällä pitäisikin olla käytössä prosessi, jonka avulla seurataan käytetyissä komponenteissa ja alustoissa havaittuja haavoittuvuuksia.

Vuonna 2014 OpenSSL-salauskirjastosta paljastui Heartbleed -niminen haavoittuvuus (CVE-2014-0160). OpenSSL-kirjastot ovat hyvin suosittuja, ja niitä on käytössä monissa eri sovelluksissa, laitteissa, komponenteissa ja alustoissa suoraan tai epäsuorasti. Heartbleed oli hyvin haitallinen haavoittuvuus ja sai paljon huomiota, joten se haluttiin korjata OpenSSL-kirjastoja käyttävissä kohteissa. Asiakkaat ottivat yhteyttä toimittajiinsa ja halusivat tietää, oliko OpenSSL käytössä heidän tuotteissaan ja olivatko he altistuneet haavoittuvuudelle. Monetkaan toimittajat eivät osanneet sanoa, mikä versio heillä oli käytössä ja oliko asiakkaiden tietoturva vaarantunut. Jotkin toimittajat eivät edes tienneet, että OpenSSL oli käytössä heidän tuotteessaan.

Voi käydä myös niin, että henkilö löytää ohjelmistosta virheen ja tahtoo ilmoittaa siitä ohjelmiston kehittäjälle. Jos henkilö on ohjelmistokehittäjän asiakas, ilmoitukseen kiinnitetään varmasti huomioita. Kyseessä voi myös olla esimerkiksi tietoturvatutkija tai muu ulkopuolinen henkilö. Ohjelmistokehittäjällä olisikin hyvä olla käytössä kanava (esim. sähköpostiosoite tai verkkolomake) kyseisenlaisten ilmoitusten tekemiseen. Jos ohjelmistokehittäjään ei pysty saamaan yhteyttä, haavoittuvuuden löytäjä saattaa julkaista haavoittuvuuden tai myydä sen jollekin epäilyttävälle toimijalle.

Verkkosivustolla https://www.tietoturvailmoitus.fi/ on perustietoa tietoturvaan liittyvien ilmoitusten vastaanottamisesta.

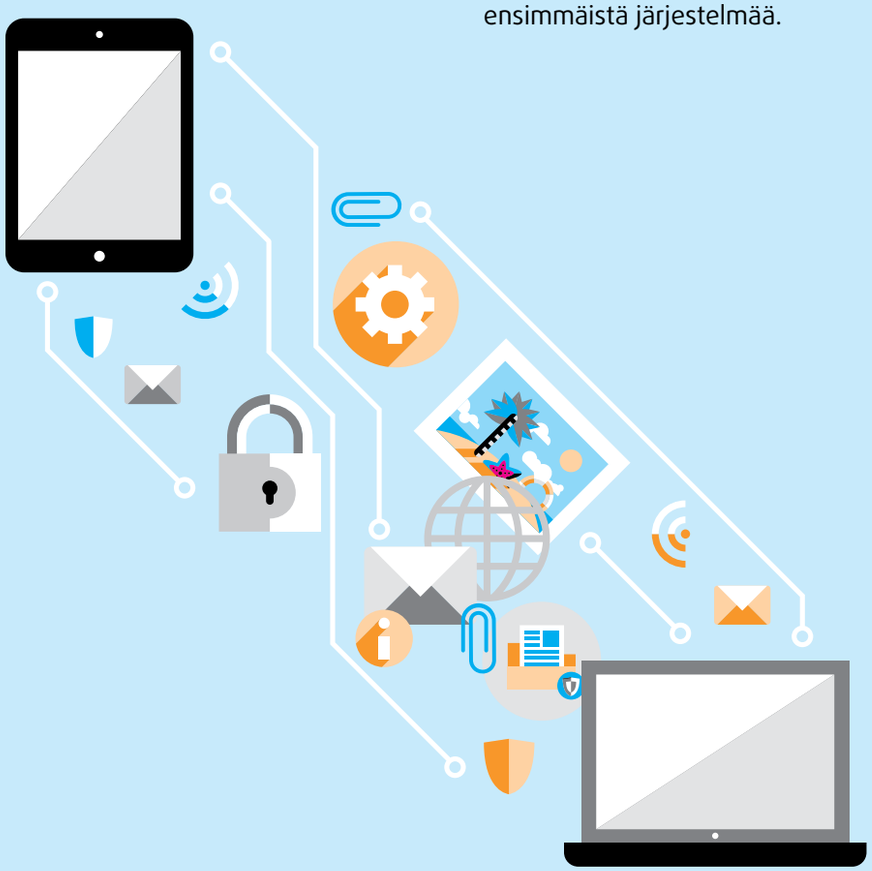
Haavoittuvuuksien löytämiseen kannustavien palkinto-ohjelmien (bug bounty programs) avulla on pystytty parantamaan tuotteiden turvallisuutta tehokkaasti. Niiden hyödyntäminen vaatii kuitenkin perehtymistä, ja tuotekehityksestä on oltava riittävästi kokemusta ja asiantuntemusta ennen kuin turvallisuustestaus jätetään ulkopuolisille toimijoille. Mutta turvallisuusongelmista huomauttavia henkilöitä ei ainakaan moittia tai tuomita. Sen sijaan he ansaitsevat kiitoksen – tai heidät voi jopa palkata työntekijöiksi!

Turvallisuuspäivityksiä tehtäessä seuraavat järjestelmät tulevat varmasti tutuiksi:

- CVE (Common Vulnerabilities and Exposures) on yleinen luettelointijärjestelmä, jonka avulla voidaan tunnistaa haavoittuvuuksia ja jossa jokaiselle löydetylle haavoittuvuudelle annetaan oma tunnuksensa (esim. CVE-2014-0160).

- CVSS (Common Vulnerability Scoring System) on luokitusjärjestelmä, jossa kullekin haavoittuvuudelle annetaan numeroarvo sen vakavuuden ja vaikutusten mukaan.

- CWE (Common Weakness Enumeration) on ohjelmistojen haavoittuvuuksia koskeva sanasto. Se ei ole yhtä tunnettu kuin kaksi ensimmäistä järjestelmää.



JOHTOPÄÄTÖKSET

Tässä oppaassa on käsitelty turvallisen ohjelmistokehityksen elinkaaren eri vaiheita. Sen onnistumista voi arvioida tuotetoimittajia koskevien Katakri-tarkastuskriteerien avulla. Ne sisältävät seuraavat vaatimukset:

1.
On varmistettu, että ohjelmistokehittäjät tietävät tietoturvasta riittävästi.

2.
Ohjelmistokehityksen aikana on suoritettu tietoturvauhka -analyysi ja havaitut riskit on joko kontrolloitu tai nimenomaisesti hyväksytty.

3.
Rajapinnat (ainakin ulkoiset) on testattu viallisilla syötteillä sekä suurilla syötemäärillä.

4.
Ohjelmointiympäristön mukaan helposti ongelmia aiheuttavia toimintoja ja rajapintoja varten on määritetty käytäntö ja sitä valvotaan (esim. Microsoftilla on listat kielletyistä toiminnoista).

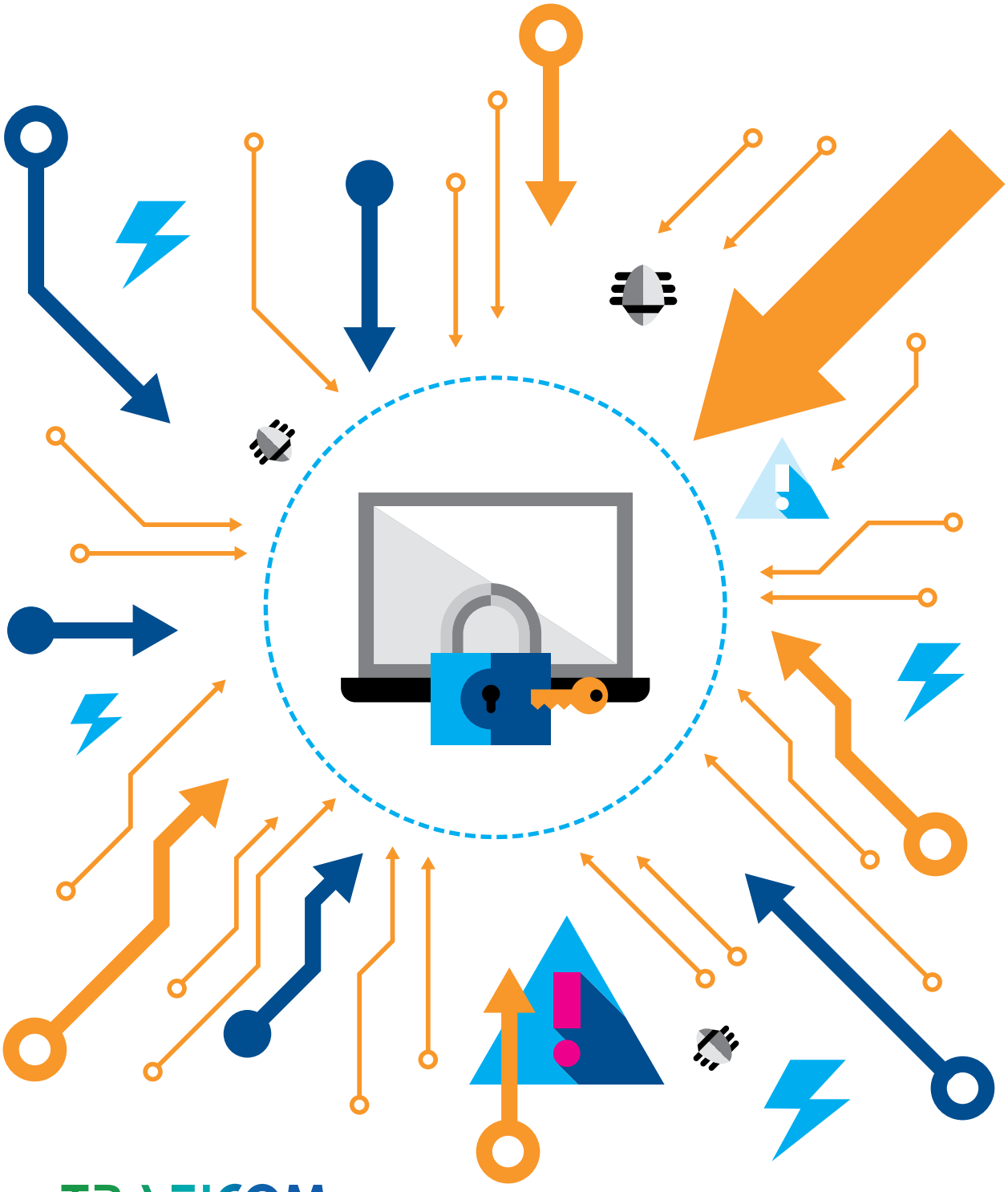
5.
Arkkitehtuuri ja lähdekoodi on katselmoitu.

6.
Tuotteen lähdekoodi on tarkastettu automatisoidulla staattisella analyysillä.

7.
Tuotteen lähdekoodin, sen versionhallinnan ja käytettyjen kehitystyökalujen eheys on varmistettu.

Lisälukemista

- *Viestintäviraston (NCSA-toiminto) asiakirjat*
- *Katakri 2015 - Tietoturvallisuuden auditointityökalu viranomaisille*
- *VAHTI 1/2013 Sovelluskehityksen tietoturvaohje*
- *OWASP, joka on oman kuvauksensa mukaan "... avoin yhteisö, jonka tavoitteena on auttaa organisaatioita suunnittelemaan, kehittämään, hankkimaan, käyttämään ja ylläpitämään luotettavia sovelluksia." OWASP on julkaissut huomattavan paljon materiaalia. Pari poimintaa:*
  - *OWASP / Kirjoittaja: Dharmesh M Mehta: Effective Software Security Management*
  - *Development OWASP Guide 3.0*
- *Microsoft SDLC oli yksi ensimmäisistä julkaistuista turvallisen ohjelmistokehityksen elinkaarista.*
- *NIST - Cryptographic Standards and Guidelines*



TRAFICOM
Liikenne- ja viestintävirasto
Kyberturvallisuuskeskus

www.ncsc.fi | www.kyberturvallisuuskeskus.fi