Pikaopas

SecOps

Eficode x Silverskin

Eficode x Silverskin

Eficode.fi

Silverskin.fi

SecOps

01

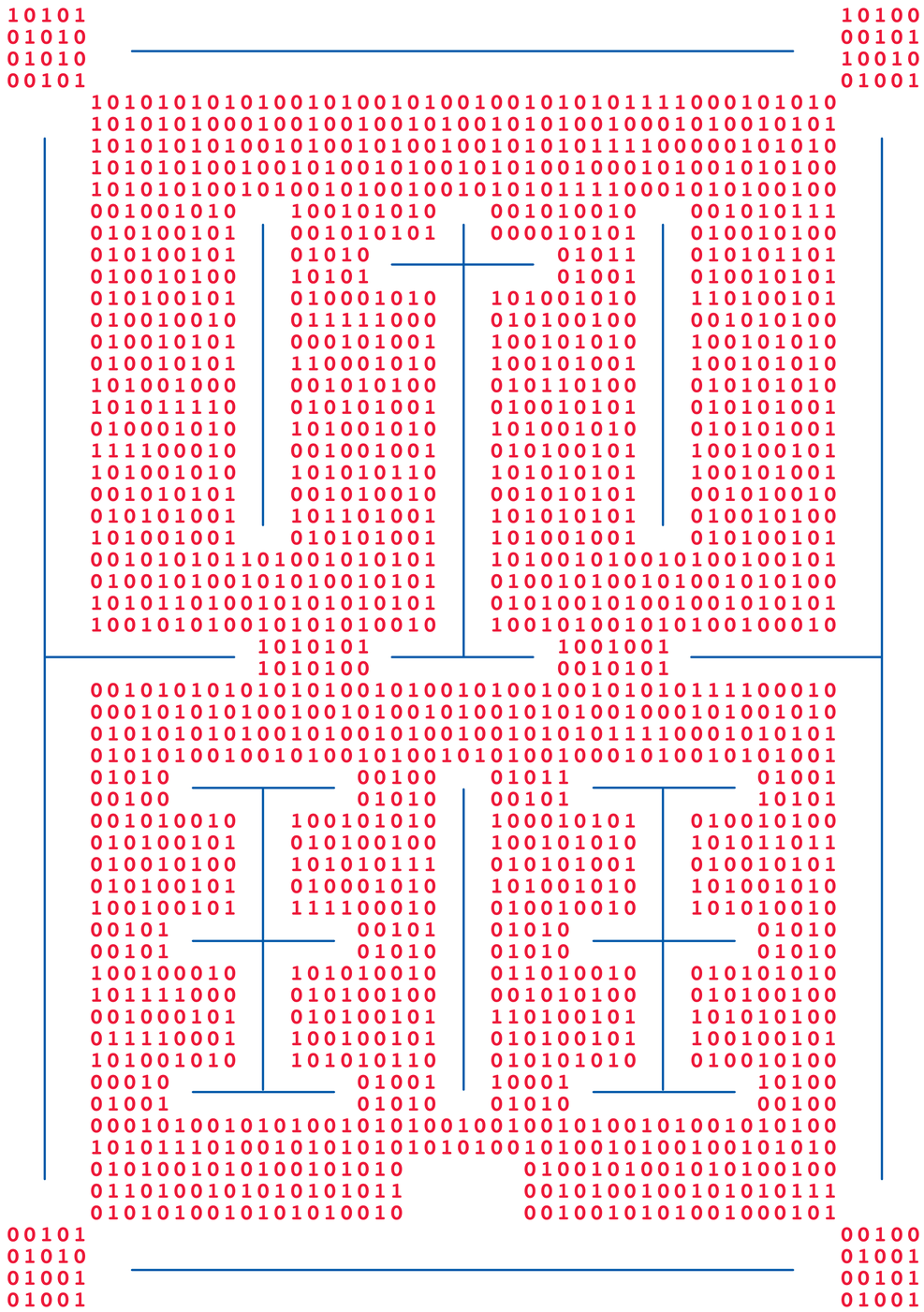
PÄHKINÄNKUORESSA

Tietoturvaan voidaan panostaa halutun riskitason saavuttamiseksi lähes kuinka paljon tahansa.

On tärkeää tietää, mihin ja missä vaiheessa panostukset tehdään, jotta vaikutus maksimoidaan.

Eficode.fi

Silverskin.fi



Miksi?

Kyberriskien
hallinta ohjelmistoprojekteissa.

Englannin kielen verbi ”to secure” tarkoittaa ennemminkin ”varmistamista” kuin ”turvallistamista”. Siksi sanat “tietoturva” ja “kyberturva” vievät ajatuksia usein väärään suuntaan. Kyberturvan varmistamisen toimenpiteet liittyvät ohjelmistokehitysprojektin aikana tuotteen turvallistamisen sijaan pääasiallisesti riskien tunnistamiseen ja niiden hallintaan.

Kyberturvaa pidetään usein teknisenä tai IT:n ongelmana, joka ratkaistaan jollain sopivalla teknisellä ratkaisulla. Kun ohjelmistotuotteen hallinnon piiriin lisätään muuttuvan lainsäädännön tai standardien mukanaan tuomat vaatimukset, esimerkiksi EU:n tietosuoja-asetus, ratkaisut eivät voi olla pelkästään teknologisia. ■

SECOPS PÄHKINÄNKUORESSA

“SecOps jakaa paljon asioita Devopsin kanssa. Secopsissa yhdistetään tietoturvasta, kehityksestä ja liiketoiminnasta vastaavat henkilöt sekä etsitään tapoja automatisoida toistuvia prosesseja uudenlaisten työkalujen avulla.”

0101 | 0101
1010 | 1010
1010 | 1010

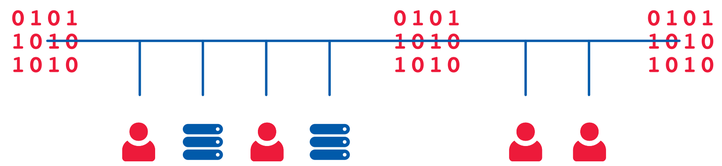
Jos tietoturvassa kerran ei ole kyse vain teknologiasta, miksi automatisoitu Secops on tarpeen?

TIETOTURVAN ERI TASOT

Tietoturvan hallintaa tehdään usealla eri tavalla ja tasoilla. Kaikilla tasoilla voidaan käyttää automaatiota vähintään tietoturvaan liittyvien prosessien tehostamiseksi sekä teknisillä tasoilla myös esimerkiksi tietoturvan testaamisessa.

Oppaan tällä puolella esitellään tietoturvan eri tasoja ja kääntöpuolella tapoja automatisoida asioita.

HALLINNOLLISET RISKIT



Varsinkin isojen kehityshankkeiden alkuvaiheessa työtavat ja projektiin liittyvät ihmiset hakevat oikeaa paikkaansa. Ihmiset eivät välttämättä tunne toisiaan, projektin tavoitteet eivät vielä ole kirkastuneet, eikä yhteistä visiota kehitettävästä tuotteesta vielä jaeta. Hallinnollisesta näkökulmasta hanketta kohtaa suuri määrä tunnistamattomia riskejä ja vaatimuksia.

Tunnistamattomat riskit liittyvät paitsi projektiin itseensä (mitä ollaan tekemässä, onko resursointi onnistunut, onko käytössä oikeat teknologiat, ovatko aikataulut ja budjetti realistiset), mutta myös kehitettävään ohjelmistoon:

- Ovatko kaikki ohjelmiston käsittelemää tietoa tai sen toimintaympäristöä koskevat vaatimukset tiedossa?
- Minkälaisia riippuvuuksia ohjelmistolla on olemassa oleviin järjestelmiin?

- Miten tunnistautuminen toteutetaan? Liittyykö siihen riskejä?
- Miten tietoliikenne suojataan?
- Onko projektissa riittävä osaaminen tietoturvasta ja kryptografiasta?
- Käsitteleekö ohjelmisto henkilötietoja? Miten ne tulee käytännössä suojata?
- Liittyykö ohjelmisto julkishallinnon järjestelmiin?
- Miten VAHTI-vaatimukset huomioidaan?

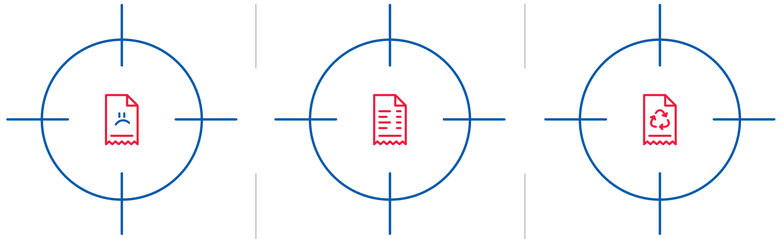
Kun näihin kysymyksiin on löydetty tyydyttävä vastaus, päästään syvemmälle tietoturvaan liittyviin kysymyksiin, kuten:

- **Voidaanko kehitettävää ohjelmistoa jotenkin käyttää vastoin tarkoitustaan?**
- **Mitä tapahtuu, jos se ”hakkeroidaan”?**
- **Voidaanko murtautumista hyödyntämällä päästä käsiksi arkaluonteiseen tietoon tai muihin järjestelmiin?**
- **Onko mahdollista että henkilötiedot vuotavat?**

Näihin kysymyksiin saadaan kyllä vastaus, mutta ei itsestään. Riskien tunnistaminen ja hallinta vaatii hankkeen johdolta osaamista. Sen lisäksi tekemiseen pitää löytyä riittävästi aikaa. Riskeihin liittyvä ajankäyttö on halvempaa projektin alussa kuin lopussa. Vasta viime metreillä tunnistettu riski, oli se sitten väärin suunniteltu tekninen ratkaisu tai huomiotta jäänyt kolmannen osapuolen vaatimus, tarkoittaa tyypillisesti sitä, etteivät aikataulu tai kustannukset pidä. Riski voidaan aina toki hyväksyä, mutta miten käytännössä tehdään päätös puutteelliseksi tiedetyn järjestelmän viemisestä tuotantoon? ■

KONTROLLI

Missä vaiheessa ja miten tietoturva varmistetaan? Tietoturvan uhkiin vastataan projektin edetessä toistuvasti tehtävillä kolmella eri tyyppisellä kontrollilla: torjuvalla, tunnistavalla ja korjaavalla.



Torjuva

Torjuvat kontrollit pyrkivät ennakoivasti estämään riskiä tai uhkaa realisoitumasta. Nämä kontrollit on suunniteltu estämään oikeudetonta pääsyä tietoon tai toiminnallisuuteen ja suojaamaan niitä tahalliselta tai tahattomalta vahingolta, muutokselta tai paljastumiselta.

0101
1010
1010

Tunnistava

Tunnistavat kontrollit pyrkivät havaitsemaan, mikäli tietoturvahaavoittuvuutta hyödynnetään tai järjestelmää väärinkäytetään.

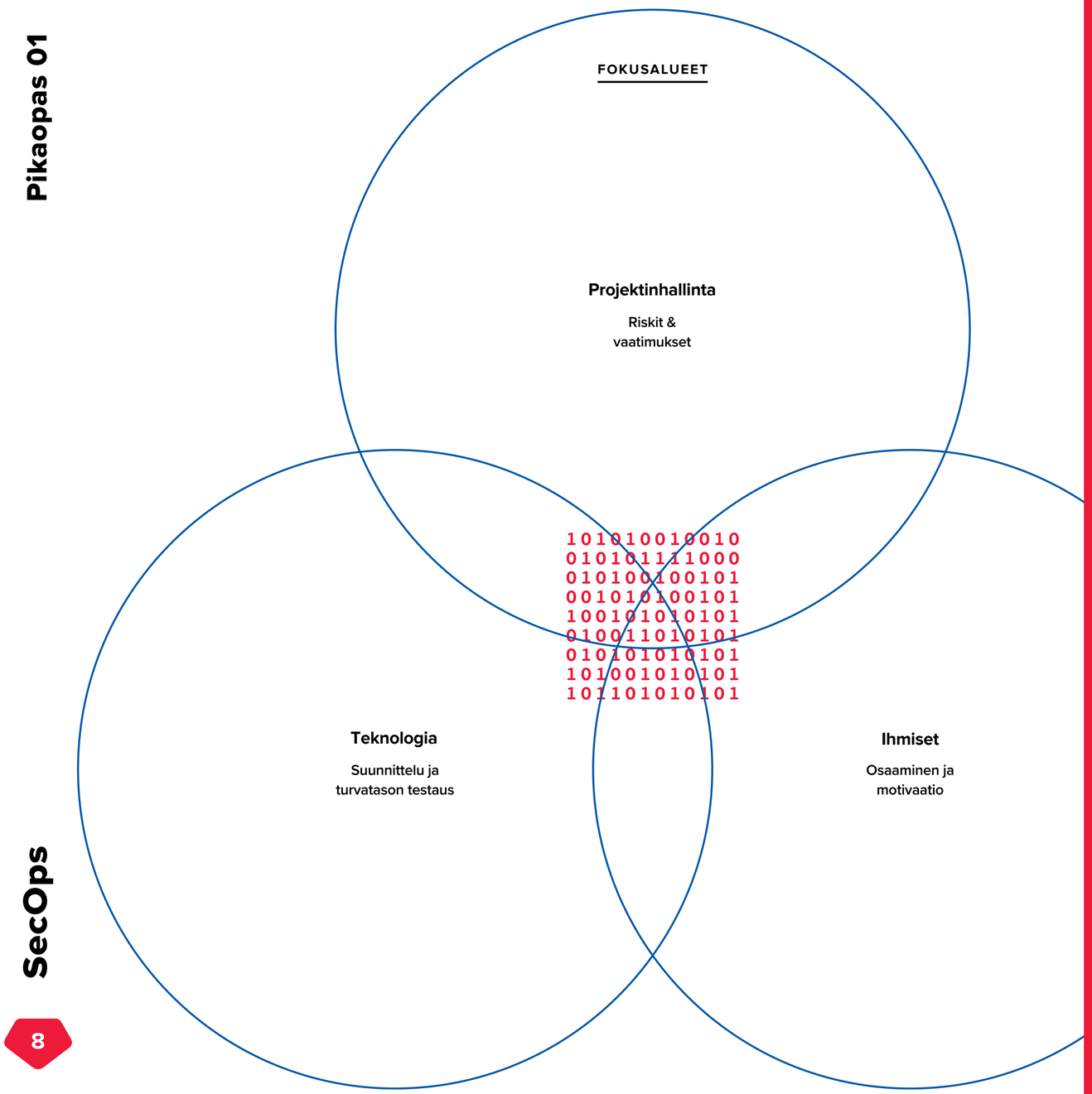
Kontrollit monitoroivat järjestelmän, laitteiden tai tietoverkon tilaa havaitakseen epänormaalia aktiviteettia tai merkkejä hyökkäyksestä. Kun hyökkäys tai tietovuoto havaitaan, tunnistavat kontrollit tuottavat hälytyksen, johon korjaavat kontrollit reagoivat.

0101
1010
1010

Korjaava

Korjaava eli palauttava kontrolli on vaste havaittuun tietoturvatapahtumaan, tyypillisesti tunnistavan kontrollin raportoimaan haitalliseen tai epätoivottuun tapahtumaan. Korjaavat kontrollit pyrkivät palauttamaan järjestelmän takaisin turvalliseen tilaan.

0101
1010
1010



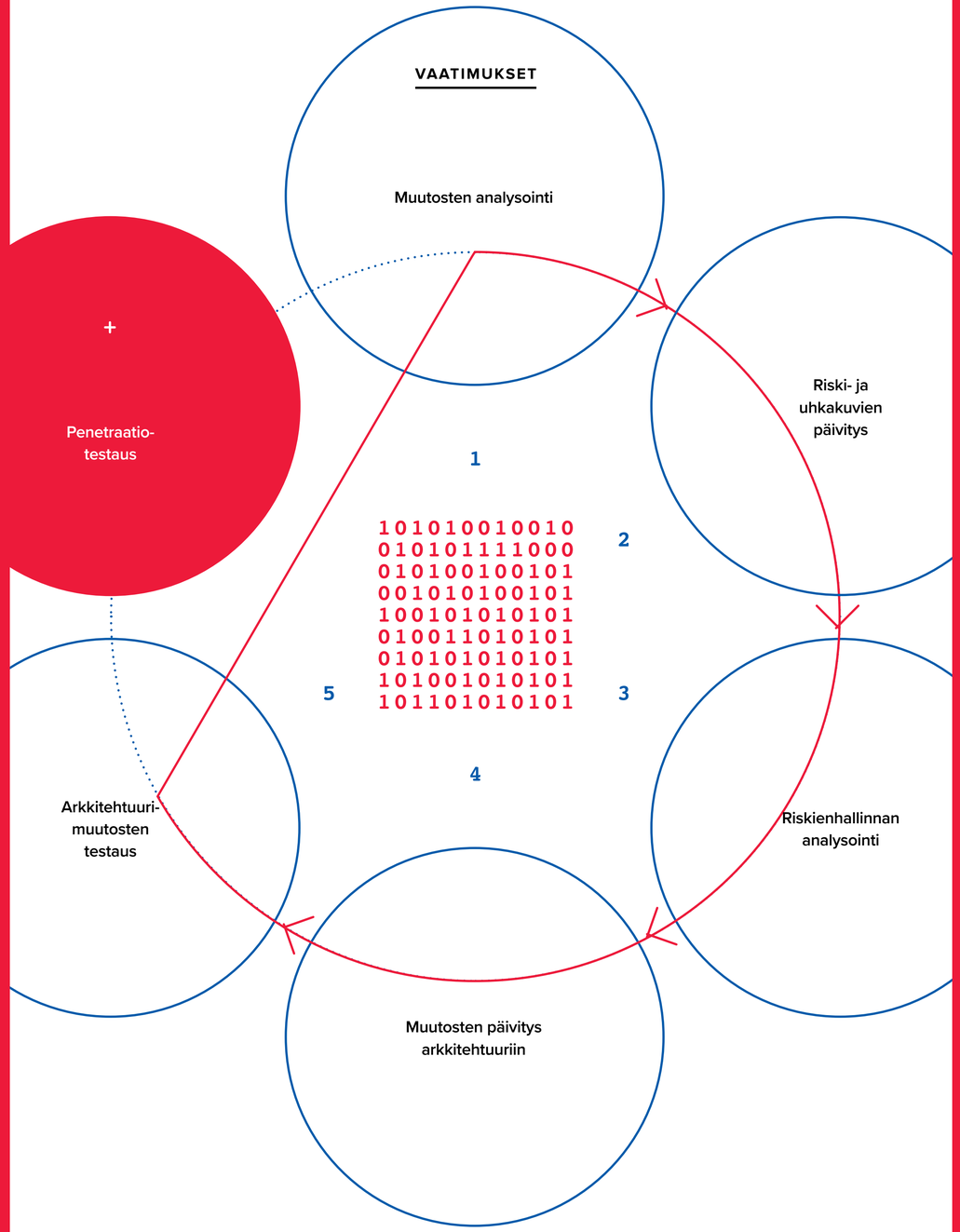
Kyberturvaongelmat näyttäytyvät usein teknologisella tasolla, mutta niiden pääsyy on usein muualla: ohjelmistokehityksen aikana ei olla täysin ymmärretty, mitä ollaan tekemässä tai resursseja kyberturvan varmistamiselle ei ole varattu riittävästi. Teknologisten ratkaisujen lisäksi on välttämätöntä parantaa sekä projektinhallinnan että ihmisten toimintaa osana tietoturvan varmistamista.

Vasta kun kaikki kolme fokusaluetta ovat päällekkäin, voidaan alkaa puhua kyberturvallisesta tai onnistuneesta hankkeesta. Järkevä projektinhallinta mahdollistaa pätevien ihmisten fiksun ajankäytön turvallisen teknologian tuottamiseen. ■

Devopsin tärkeimpiä tavoitteita ovat kyky jatkuvaan mittaamiseen sekä kyky nopeaan reagointiin.

Kyberturvan osalta hankkeissa on usein jääty vesiputoustyyliseen ajatteluun, jossa oletetaan, että kaikki järjestelmää koskevat riskit ja ulkoiset vaatimukset pystytään tunnistamaan ennen kuin järjestelmän arkkitehtuuria aletaan miettiä.

Sen sijaan, järjestelmään kohdistuvat uhkakuvat ovat muuttuvia, täsmälleen samalla tavalla kuin järjestelmän vaatimukset, koodi tai testitapaukset.



DOKUMENTAATIO

Et voi suojata sellaista, mitä et tiedä olevan olemassa.

Osana toteutustyötä on tärkeää kyetä tunnistamaan tietoturvan kannalta relevantit asiat ja arvioimaan niiden mukanaan tuomat riskit. Jokaisen uuden toiminnallisuuden osalta on syytä tunnistaa kohdistuuko siihen esimerkiksi ulkoisia vaatimuksia tai onko se muulla tavoin arkaluonteista? Liittyykö uuteen toiminnallisuuteen käyttäjän tai järjestelmien tunnistamiseen liittyviä asioita? Pääsyoikeuksiin tai valtuuttamiseen liittyviä asioita? Seurantaan, raportointiin tai lokittamiseen liittyviä asioita?

Tyypillisesti tietoturvariskit liittyvät arkaluonteiseen tietoon, kriittiseen toiminnallisuuteen, käyttäjän tunnistamiseen ja pääsyoikeuksiin, sekä järjestelmien väliseen kommunikaatioon ja interaktioon.

Monimutkaisten järjestelmien tietoturvadokumentaatio kuvaa tietosisältöön, toiminnallisuuteen sekä käyttöön liittyviä turvallisuusasioita. Tietosisällön osalta kuvataan, mitä järjestelmän käsittelemä arkaluonteinen tieto on, miksi se on arkaluonteista, sekä miten se on suojattu. Tärkeimmät käyttötapaukset kuvataan yhdessä väärinkäyttötapausten kanssa. Integraatioiden osalta kuvataan, mihin suuntaan mitäkin tietoa järjestelmien välillä liikkuu, sekä minkälaiset mekanismit liittyvät liikutettavan tiedon suojaamiseen.

Tietoturvadokumentaatio kuvaa, mikä järjestelmässä on arvokasta ja miten se on suojattu. Sen lisäksi se toimii myös tietoturvallisuuden varmistavien toimenpiteiden toteutumisen seurannassa. Jokaisen järjestelmän osalta pidetään kirjaa siitä, milloin siihen on viimeksi toteutettu uhkamallinnus, arkkitehtuurikatselmointi, koodikatselmointi sekä tietoturvatestaus ja niin edelleen. Jos järjestelmään on tehty suuria muutoksia, ja edellisestä katselmointikerrasta on yli vuosi, on tärkeää uusia katselmointi ja päivittää dokumentaatio.

Käyttäjien, olivat ne sitten ihmisiä tai sovelluksia, tunnistamisen, valtuuttamisen sekä seurannan osalta kuvataan niihin liittyvät mekanismit, esimerkiksi ihmiskäyttäjien vahva tunnistaminen, rajapintojen tunnistusavaimet, roolipohjainen valtuutus ja keskitetty lokienhallinta.

UHKAMALLINNUS

Hyökkäys on paras puolustus

Kumpi oli ensin, kilpi vai keihäs? Voidaanko kestävää järjestelmää rakentaa ymmärtämättä, mikä kaikki voisi vaikuttaa siihen heikentävästi tai vaarantavasti? Väärinkäyttötapausten ja uhkamallien laatiminen on osa suunnitteluvaiheen kyberturvaa. Malleja laaditaan myös, kun lisätään uutta toiminnallisuutta tai jatkokehitetään vanhaa.

Jokainen toiminnallisuus sisältää mahdollisuuden sen väärinkäyttöön. Väärinkäyttötapauksissa ja uhkamallinnuksessa on tärkeää kuvata ne aktiivimuotoisilla virkkeillä – passiivin käyttö on kielletty. Kun hyökkäyksellä tai uhkalla on jokin subjekti, on helpompi kuvitella myös motiiveja.

Tunnistetut uhkat toimivat omana kanavanaan tietoturvavaatimuksille. Muut vaatimukset saadaan ulkoisista vaatimuskehikoista, kuten lainsäädännöstä, sopimuksista, toimintaympäristön riskienhallintaan liittyvistä vaatimuksista tai valitun teknologian parhaista käytännöistä.

RISKIEN HALLINTA

Ennaltaehkäise jos voit, hallitse silti aina riskit

Viime kädessä kyberturvan tuotantoa johtaa riskiajattelu, jossa arvioidaan riskin todennäköisyyttä ja vaikutusta. Riskien hallintamekanismeja ovat muun muassa riskin hyväksyminen, pienentäminen tai siirtäminen eli rahoittaminen. Henkilöriskien pienentäminen tapahtuu vaikkapa koulutuksella tai osaamisen siirtämisellä yksilöltä ryhmälle.

Teknisiin uhkiin vastataan aiemmin esitellyillä kontrolleilla, jotka pyrkivät torjumaan, tunnistamaan ja poistamaan haitallisia tilanteita. Esimerkiksi palomuuri on tunnettu verkkotason tietoturvakontrolli, joka pyrkii estämään luvattomien tietoliikenneyhteyksien muodostamisen tai kääntäen: mahdollistamaan ainoastaan sallittujen yhteyksien muodostamisen. ■

LOPUKSI

Tietoturvallisuus ohjelmistohankkeissa koostuu ulkoisten vaatimusten (compliance) hallinnasta sekä ohjelmiston laadun varmistamisesta.

Laadukas ohjelmisto kestää käytön ja estää väärinkäytön. Ohjelmiston dokumentaatio ja automaattiset testit taas mahdollistavat helpon auditoitavuuden sekä turvallisuuden ylläpidon. Ja vaikka ohjelmistoon itsessään ei kohdistuisi muutospaineita, niin maailma sen ympärillä muuttuu koko ajan; hakkerit löytävät uusia haavoittuvuuksia yleisesti käytössä olevista komponenteista, ja niiden päivittyminen saattaa vaikuttaa ohjelmiston toimintaan.

Kyberresilienssi eli vastustuskyky on järjestelmän tai organisaation kyvykkyys torjua ja toipua haitallisesta kybervaikutuksesta. Kyberresilienssiä voidaan kehittää, kun järjestelmä altistetaan tahallisesti sen uhkamallin mukaiselle hyökkäyssimulaatiolle, ja simulaatiossa löydetyt havainnot korjataan.

Ohjelmistojen jatkuva testaaminen on yksi tapa kehittää järjestelmien vastustuskykyä, mutta sitä voi testata ja kehittää myös yhdessä tietoturvaan perehtyneiden asiantuntijoiden kanssa.

Täydellistä tietoturvaa ei ole olemassa, riittävä tietoturva on.

SecOps

02

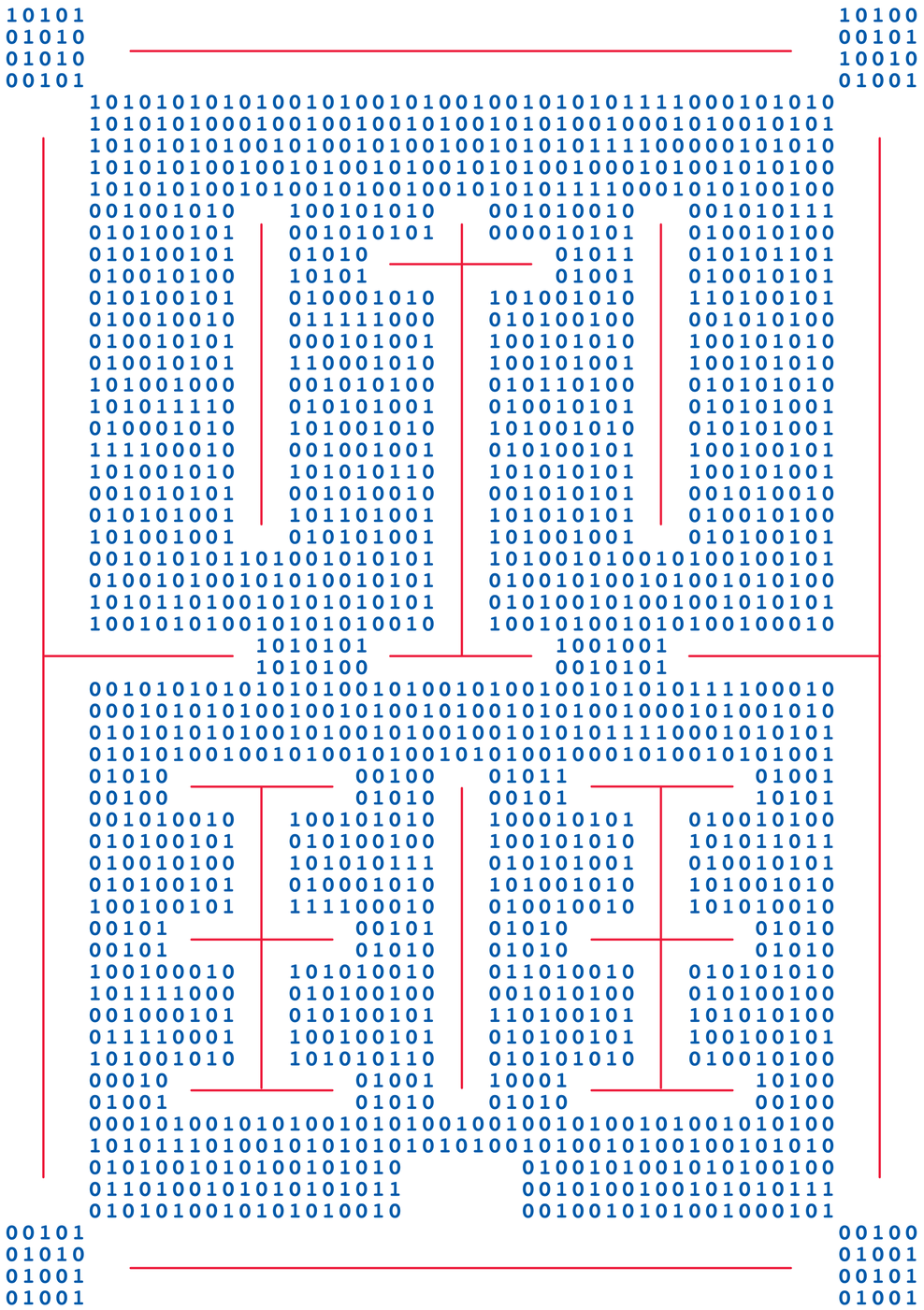
PÄHKINÄNKUORESSA

Monet tietoturvaan ja sen ylläpitoon liittyvät prosessit ovat toistuvia.

Niitä voidaan kuitenkin automatisoida käyttämällä nykyaikaisia työkaluja ja menetelmiä.

Eficode.fi

Silverskin.fi



Miten?

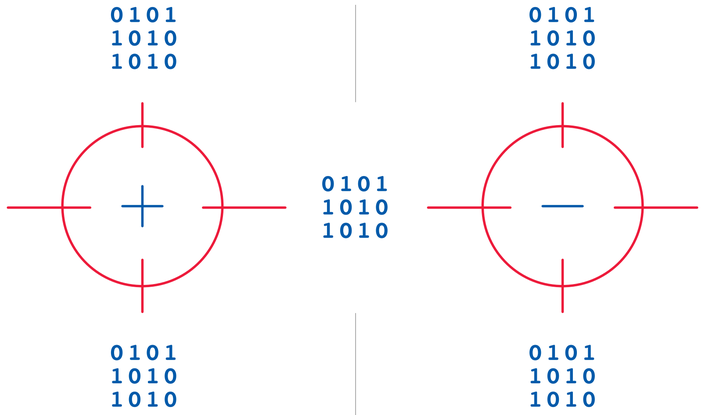
Tietoturvaa voidaan kehittää jatkuvasti.

Moderni devops-käytäntöjä hyödyntävä ohjelmistokehitys perustuu virtualisointiin ja korkeaan automaatioasteeseen. Automaation avulla poistetaan käsin tehtäviä toistuvia työtehtäviä ja eliminoidaan inhimillisiä virheitä. Vapautunut työaika suunnataan laadun ja asiakaskokemuksen parantamiseen, jolloin kehitystahti nopeutuu. Lisäksi automaatio mahdollistaa parhaimmillaan nopean ja jatkuvan julkaisun.

Tietoturvan hallintaan liittyy yhtä lailla manuaalisia ja toisteisia työtehtäviä. Manuaalinen työ vie aikaa ja on virhealtista. Ilman vastaavaa panostusta tietoturvatestauksen automaatioon testauksesta tulee pullonkaula tai sen laatu heikkenee. Manuaalisen työn tehokkuus ei riitä kattavaan testaukseen jokaisen julkaisun yhteydessä, jolloin toistuvia prosesseja on automatisoitava.

Oppaan tällä puolella kerrotaan esimerkkien avulla, mitä osia tietoturvan testauksesta voi ja kannattaa automatisoida. ■

AUTOMATISOINNIN PLUSSAT & MIINUKSET



Plussat

- Ulkoiset haavoittuvuuskirjastot päivittyvät automaattisesti ja paljastavat aikaisemmin tuntemattomia ongelmia ohjelmistossa. Kaikkea ei tarvitse löytää itse.
- Löydettyjen haavoittuvuuksien tutkiminen ja arvioiminen on mielenkiintoista mutta varsinaisten testien ajaminen ei. Tylsä työ kannattaa jättää koneille.
- Tieto puutteista saadaan nopeasti, ja kehittäjä voi reagoida niihin heti, ilman että hän ehtii siirtyä välissä tekemään jotain muuta.
- Alkuun pääsee melko vähälläkin, jos muu ohjelmistokehityksen infrastruktuuri on kunnossa.

Miinukset

- Kaikkea ei voi automatisoida – vielä. Tietoturvan arvioinnissa tarvitaan aina myös ihmisiä.
- Automaatiota on vaikea tehdä, jos muu infra ei ole kunnossa
- Automaation pystyttäminen vaatii erityisasiantuntemusta
- Tietoturvatestaukseen on paljon työkaluja, mutta kaikki niistä eivät integroidu kätevästi muihin työkaluihin.

AUTOMATISOINTI

Mitä osia ohjelmiston turvallisuudesta voidaan testata automaattisesti?

”Toimintaympäristön ja itse tehtyjen muutosten turvallisuusvaikutus tiedetään etukäteen ja odottamattomat muutokset havaitaan nopeasti.”

Automaattisen tietoturvatestauksen tavoite

1. JÄRJESTELMÄTASO

Jokainen ohjelmisto tarvitsee toimiakseen muita järjestelmiä ja infrastruktuuria. Tällaisia voivat olla esimerkiksi www-palvelimet, tietokannat sekä kuormantasaimet tai muut verkkoelementit. Infrastruktuurin turvallisuus on tärkeää, sillä sen murtuessa koko järjestelmä on haavoittuva.

Esimerkkikysymyksiä:

- Tiedetäänkö palvelimella käytössä oleva SSH:n versio turvattomaksi?
- Mitä portteja palvelimella on auki julkiseen verkkoon?
- Onko palvelimen SSL-sertifikaatti ajan tasalla?
- Onko palvelinohjelmisto konfiguroitu oikein?

Järjestelmätason turvallisuutta voidaan tutkia haavoittuvuusskannereilla, jotka pystyvät tunnistamaan eri infrastruktuurikomponentit, niiden versiot ja kriittisen konfiguraation. Haavoittuvuuskannerit tutkivat usein myös yksittäisissä versioissa ilmeneviä puutteita, jotka ilmenevät vain jos järjestelmä on konfiguroitu määrätyllä tavalla – haavoittuvuus voisi olla esimerkiksi Apachen SSL-sertifikaattien käsittelyssä mutta ilmenee vain, jos käytössä on 1024 bitin sertifikaatti.

Infrastruktuurin turvallisuutta voidaan tutkia myös verkkoskannereilla. Näiden skannereiden tarkoituksena on pitää huolta siitä, että ainoastaan oikeat palvelut näkyvät ulospäin. Tällä strategialla rajoitetaan hyökkääjän käytössä olevaa hyökkäyspinta-alaa. Ajatuksen ydin on, mitä vähemmän palveluita hyökkääjä näkee, sitä vähemmän mahdollisia haavoittuvuuksia on löydettävissä. Infrastruktuurin turvallisuus on siitä mielenkiintoinen osa ohjelmistojen turvallisuutta, että skannereiden ajaminen ei tyypillisesti vaaranna järjestelmän toimintaa. Tästä syystä infrastruktuurin turvallisuutta voidaan valvoa myös tuotannossa ajamalla skannerit esimerkiksi joka yö.

Infrastruktuurin turvallisuuden jatkuva seuranta on tärkeää erityisesti silloin, kun ohjelmistoon ei tehdä erityisiä muutoksia, ja muita turvallisuustarkastuksia ei välttämättä ajeta. Www-palvelimista, tietokannoista, käyttöjärjestelmistä ja muista infrastruktuurin osista löydetään kuitenkin jatkuvasti uusia haavoittuvuuksia, ja vakaviin löydöksiin on syytä reagoida välittömästi.

2. OHJELMISTON LOGIIKKATASO

Ohjelmiston logiikkataso käsittelee ohjelmiston käyttötarkoitusta. Normaalit testit tutkivat, tekeekö ohjelmisto sen, mitä sen pitää. Turvallisuustestien tarkoituksena on tarkistaa, että ohjelmisto ei tee asioita, joita sen ei kuulu tehdä.

Esimerkkikysymyksiä:

- Voivatko loppukäyttäjät muuttaa pääkäyttäjien muutettavaksi tarkoitettuja tietoja, vaikkapa verkkokaupan hintoja?
- Voiko käyttäjä päästä näkemään toisen käyttäjän tietoja?
- Voiko käyttäjä huijata järjestelmän luulemaan, että maksu on tehty tekemättä maksua?

Ohjelmiston logiikassa on se hankaluus, että sen testaaminen edellyttää ohjelmiston tarkoituksen ymmärtämistä.

Tästä syystä yleismaailmallisten testien tekeminen ohjelmiston logiikalle on käytännössä mahdotonta. On siis erityisen tärkeää arvioida ohjelmiston riskitaso: onko hyökkääjä opportunistinen skannailija, vai onko hän erityisen kiinnostunut hyökkäämään nimenomaan tätä järjestelmää vastaan.

Tietoturvaan liittyviä testejä on syytä kirjoittaa kaikkiin ohjelmistoihin. Testien tulisi varmistaa ainakin, että väärillä salasanoilla ei pääse kirjautumaan sisään, ja että käyttäjät eivät pääse muokkaamaan tietoja, joiden muokkaaminen ei heille kuulu. Rajoituksena on kuitenkin, että testeillä voidaan testata vain sellaisia asioita, jotka testien kirjoittaja on ottanut huomioon.

Muista tasoista poiketen, muiden tekemiä havaintoja tietoturvapuutteista ei siis voida hyödyntää.

3. OHJELMISTON RAKENNETASO

Nykyiset ohjelmistot koostuvat usein useista komponenteista, joista monet tulevat ulkoisista lähteistä. Erilaiset ohjelmistokehykset ja kirjastot päivittyvät nopeasti, kun niiden kehittäjät korjaavat puutteita. Näiden komponenttien kehittymisen seuranta ja oikea käyttö ovat oleellinen osa ohjelmiston kokonaisturvallisuutta.

Esimerkkikysymyksiä:

- Onko käyttämässäni tietokantakirjastossa SQL-haavoittuvuuksia?
- Tallentaako käyttäjänhallintakirjasto salasanat turvallisesti?
- Onko käyttäjän istunto mahdollista kaapata?
- Käsitelläänkö käyttäjien syötteet turvallisesti?

Ohjelmiston rakenteen tutkimiseen tarvitaan käytettyihin teknologioihin erikoistuneita työkaluja. Yleisimpiä näistä ovat lähdekoodin staattiseen analysointiin tarkoitetut työkalut. Pohjimmiltaan ne lukevat koodia ja konfigurointitiedostoja etsien niistä yleisiä ohjelmointivirheitä tai komponenttien vanhentuneita versioita.

4. KÄYTTÖLIITTYMÄTASO

Voivatko käyttäjät vahingossa vaarantaa turvallisuuden ilman, että ohjelmisto varoittaa? Yleisemmin osa tietoturvapuutteesta on käyttöliittymässä, mutta tärkein puute on käyttöliittymää palvelevassa järjestelmässä, joka ei tarkista käyttäjältä tulevia tietojen tai toimenpiteiden luvallisuutta.

Tietoturvaongelmia voi esiintyä myös pelkässä käyttöliittymässä ilman, että taustajärjestelmän tietoturvamallissa on puutteita. Käyttöliittymän tietoturvapuutteet esiintyvät tilanteissa, jossa käyttäjällä on oikeus tehdä muutoksia järjestelmään, mutta käyttöliittymän puutteet saavat käyttäjän toimimaan väärin.

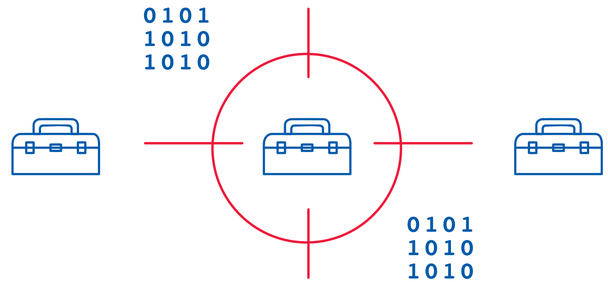
Konkreettinen esimerkki tällaisesta tietoturvapuutteesta voisi olla esimerkiksi verkkokauppajärjestelmän hintalista. Hintalistalle voidaan asettaa asiakaskohtaisia hintoja eri tuotteille. Järjestelmän pitäisi kuitenkin varoittaa, mikäli hintalistalla joidenkin tuotteiden hinnat on asetettu nollaksi, jotta käyttäjä ei epähuomiossa anna kalliita tuotteita ilmaiseksi. Ilmaiset tuotteetkin ovat kuitenkin ainakin teoriassa täysin tarpeellinen käyttötapaus, joten käyttäjällä tulee olla mahdollisuus niiden luomiseen.

Tällaisia käyttöliittymään liittyviä tietoturvapuutteita on lähes mahdoton löytää automaattisesti, sillä testin pitäisi pystyä tunnistamaan ihmisen kyky erehtymiseen. Käyttöliittymän puutteet tulee pyrkiä estämään hyvillä vaatimuksilla, jotka johdetaan järjestelmän uhkamallista.

Toinen käyttöliittymän tietoturvaan vaikuttava tekjiä liittyy pahojen ja erehtyväisten käyttäjien toiminnan huomioimisen käyttöliittymää suunniteltaessa. Erehtyväisille käyttäjille on syytä antaa mahdollisimman täsmällistä tietoa virhetilanteissa, mutta virheilmoituksia suunniteltaessa on syytä huomioida se mahdollisuus, että käyttäjä onkin vihamielinen.

Tästä syystä esimerkiksi kirjautumisvirheiden yhteydessä ei ole syytä antaa tietoa onko virheellinen tieto käyttäjätunnus vai salasana, sillä oikeaan käyttäjätunnukseen osuminen on hyökkääjälle arvokasta tietoa. ■

Ehdotuksia tietoturvan testaustyökaluiksi



1. Staattinen analyysi

Lähdekoodin staattista analyysiä on perinteisesti käytetty hyvän ohjelmointityylin ja käytäntöjen seurantaan, mutta se paljastaa myös useita eri tyyppisiä tietoturvapuutteita. Staattisella analyysillä voidaan havaita esimerkiksi puutteita syötteiden ja sessionkäsittelyssä, puutteellista kryptografista turvallisuutta tai turvatonta muistinkäsittelyä.

SonarQube on avoimeen lähdekoodiin perustuva staattinen analyysityökalu, johon on tarjolla useita eri kirjastoja tietoturvapuutteiden löytämiseksi. Hyvä alku ovat SANS 25- ja OWASP Top 10 -haavoittuvuuskirjastot.

Haavoittuvuuskirjastot toimivat vain niillä ohjelmointikielillä, joille ne on suunniteltu. SonarQube tarjoaa tuen useimmille ohjelmointikielille.

SonarQube ei tue läheskään kaikkia kieliä, mutta onneksi joillakin yleisillä frameworkeillä on omia turvallisuustestaustyökaluja. Ruby on Rails -sovelluksille on esimerkiksi tarjolla erinomainen Brakeman-tietoturvaskanneri, jonka vahvuutena on sen kehittäjien ymmärrys Rails-projekteissa käytettävistä yleisistä rakenteista ja virheistä.

2. ZAP

Zed Attack Proxy on myös OWASP Top 10 -listaa ylläpitävän Open Web Application Safety Project -organisaation ylläpitämä projekti. ZAP on hyökkäysvälityspalvelin, joka asettuu käyttäjän ja testattavan palvelun väliin. Se tutkii välitettäviä viestejä ja syöttää niihin erilaisia hyökkäyksiä. ZAP osaa myös itsenäisesti skannata palvelun rajapintoja, mutta useimmissa sovelluksissa on parempi rakentaa esimerkkejä.

---

3. Tuotantopalveluiden turvallisuuden valvonta

Devops-turvallisuuden ops-puolta ja listan ainoaa tunnistavaa kontrollia edustaa OSSEC. OSSEC on palvelinten turvallisuuden valvontaan tarkoitettu työkalu, joka hoitaa (palvelinalustasta riippuen) erilaisia toimenpiteitä. OSSEC:n tärkein toiminnallisuus lienee lokien seurantaan perustuva poikkeustilanteiden tunnistaminen. Alustaan voi määritellä sääntöjä, joiden perusteella ylläpidolle tehdään hälytyksiä. Hälytyksiä voidaan nostaa esimerkiksi toistuvista kirjautumisyrityksistä tai porttien skannauksesta. OSSEC tarjoaa myös mahdollisuuden rekisterin valvontaan Windowsissa ja erilaisten konfiguraatiotiedostojen muutoksista hälyttämiseen.

---

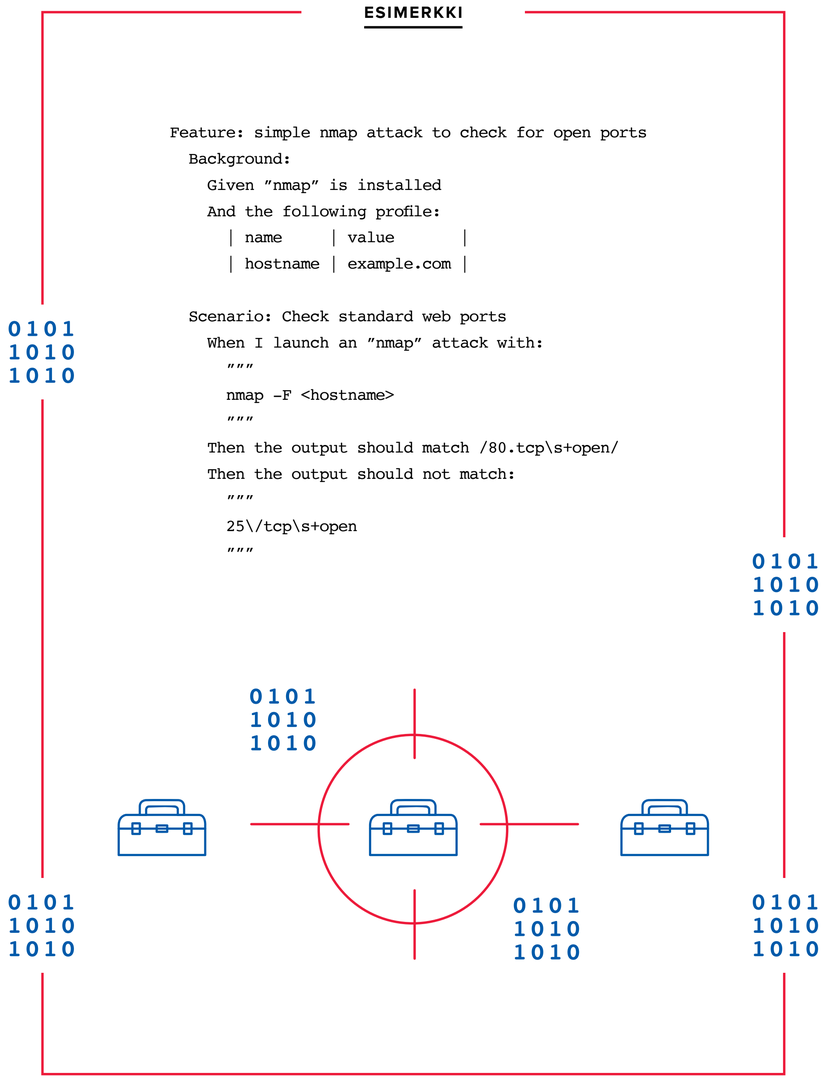
4. Turvallisuuden hyväksymistestaus

Ohjelmistojen turvallisuuteen liittyviä hyväksymistestejä voidaan kirjoittaa osittain myös normaaleilla testaustyökaluilla, kuten Robot Frameworkillä kuvailemalla testeissä ilkeämielisten käyttäjien käyttötapauksia ja tarkastamalla, että ne käsitellään oikein. Tietoturvatestaukseen erikoistuneita hyväksymistestityökaluja on kuitenkin olemassa. Yksi erinomainen työkalu tähän tarkoitukseen on Ruby-pohjainen Gauntlt, jolla voidaan kuvata esimerkiksi verkkoon näkyvien palveluiden profiilia, muun muassa SSL:n ja aukinaisten porttien osalta (esimerkki seuraavalla sivulla).

ESIMERKKI

```
Feature: simple nmap attack to check for open ports
  Background:
    Given "nmap" is installed
    And the following profile:
      | name     | value       |
      | hostname | example.com |

  Scenario: Check standard web ports
    When I launch an "nmap" attack with:
      """
      nmap -F <hostname>
      """
    Then the output should match /80.tcp\s+open/
    Then the output should not match:
      """
      25\/tcp\s+open
      """
```



Jatkuvan tietoturvatestauksen haasteet

Kaikessa testaamisessa on se hankaluus, että kiireessä erilaiset testit – jopa automatisoidut – jäävät helposti ajamatta. Tästä syystä testien ajaminen on syytä automatisoida siten, että testien ajaminen ei ole ihmisten muistamisesta kiinni.

Tietoturvatestaukseen on hurja määrä työkaluja, mutta hyvin harvat niistä integroituvat kätevästi osaksi jatkuvan integraation työkaluja. Integrointi usein edellyttääkin hiukan omaa työtä erilaisten liitännäisten muodossa. Oppaassa esiteltyihin työkaluihin on olemassa valmiit integraatiot, ellei toisin mainita.

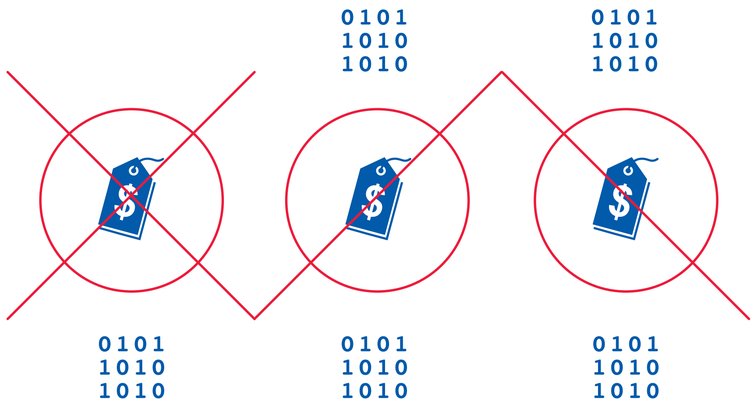
Tietoturvan testaus- ja valvontatyökaluille on tarjolla kaksi luonnollista integraatiopaikkaa, hiukan riippuen siitä, mitä ne tekevät. Penetraatiotestaukseen ja lähdekoodin skannaukseen perustuvat torjuvat työkalut kannattaa kiinnittää osaksi normaalia laadunvarmistusta jatkuvan integraation (CI) palvelimeen, esimerkiksi Jenkinsiin. Mikäli käytössä ei ole valmista CI-palvelinta, on olemassa myös valmiita tietoturvan testaukseen erikoistuneita automaatiopalvelimia, kuten Mozillan Minion.

Järkevä tapa integroida työkalut osaksi CI-putkea on jakaa ne kahteen osaan. Gauntltin tyyppiset työkalut, jolla kirjoitetaan täsmällisiä tietoturvavaatimuksia on syytä ajaa osana jokaisesta muutoksesta käynnistyvää koontiajoa. Laajemmat skannerit sen sijaan kannattaa ajaa omina erillisinä, esimerkiksi öisinä ajoinaan. Erittelyyn on kaksi syytä: ensinnäkin, laajemmat ajot usein vievät aikaa, eivätkä niiden tulokset muutu kovin usein; toiseksi laajempien skannereiden tuloksissa esiintyy paljon asioita, joihin ei ole tarpeen reagoida, eikä ohjelmistojen testejä ole syytä merkata epäonnistuneiksi niiden vuoksi.

Vinkki: Jos haluat tutustua Minion- ja Jenkins-pohjaiseen tietoturvatestaukseen tarkemmin, lue tekstit Eficoden blogista – siellä on esitelty esimerkki, jolla saat ohjelmiston testattua alusta loppuun.

Perinteisempiä palvelun saatavuutta testaavia järjestelmiä, kuten Shinkeniä, Nagiosta tai Zabbixia voidaan käyttää osittain myös turvallisuuteen liittyvien metriikoiden testaamiseen. Niihin voidaan kytkeä esimerkiksi NMAP-pohjainen porttiskannaus uusien palveluiden varalta; lisäksi OSSEC:n tyyppisten hyökkäystentunnistusjärjestelmien tuottamat tulokset voidaan liittää valvontapalvelimiin. ■

RAJOITUKSET



Mitä rajoituksia tietoturva-automaatiolla on?

Automaattisesti luodut tietoturvatestit eivät pääsääntöisesti ymmärrä ohjelmiston tarkoitusta. Ne eivät esimerkiksi erota helposti toisistaan tilannetta, jossa jonkin toiminnon suorittaminen ei ole sallittua kaikille käyttäjille. Verkkokaupassa hinnan muuttaminen on tietenkin sallittua, mutta ainoastaan järjestelmän ylläpitäjälle. Automaattisesti muodostettavat testit eivät välttämättä osaa erottaa tilanteita toisistaan, vaan tilanteille on kirjoitettava hyväksymistestit.

Myös hyväksymistestejä voi käyttää turvallisuuden testaamiseen, mutta niiden rajoituksena on, että jokainen testitapaus pitää erikseen määritellä. Tällöin niiden kehittäjän on pystyttävä ottamaan kaikki tapaukset huomioon – hyväksymistesti löytää siis vain regressioita.

Erilaisten tietoturvaskannereiden vahvuutena on, että ne löytävät regressioiden lisäksi ohjelmiston kehittäjälle uusia haavoittuvuuksia, jotka skannerin rakentaneen tahon tietoturva-asiantuntija on ottanut huomioon. Skanneritkaan eivät kuitenkaan löydä ohjelmistosta kokonaan uusia haavoittuvuustyyppejä, eivätkä osaa arvioida useiden löydösten yhteisvaikutusta. ■

MITEN PÄÄSTÄ ALKUUN?

Se, mistä kohdasta automaattisen tietoturvatestauksen kokeileminen kannattaa aloittaa, riippuu lähtötilanteesta.

”Ei mitään. Ei edes ohjelmistoa testattavaksi, mutta haluan kokeilla turvallisuustestausta.”

Katso blogistamme ohjeet kuinka teet Jenkins-pohjaisia turvallisuustestejä valmiiseen esimerkkiohjelmistoon. Tarvitset vain tietokoneesi.

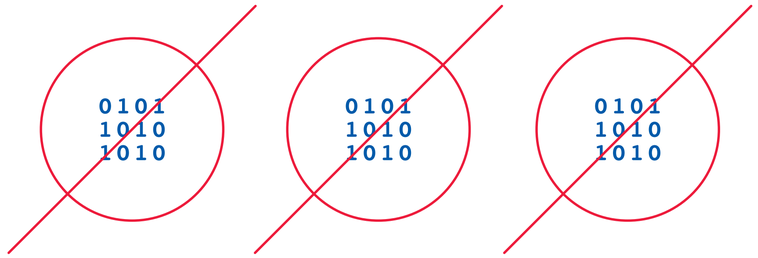
”Minulla on ohjelmisto ja haluan testata sitä pikaisesti.”

Kokeile tutoriaalimme avulla, miten teet ensimmäiset turvallisuustestit ohjelmistoosi Minionilla. Lue kuitenkin ensin oppaan Vinkkejä-osio ennen aloittamista.

”Ohjelmistoni on versionhallinnassa, ja CI-palvelin pyörittää koonteja jokaisesta muutoksesta. Haluan lisätä ajoon turvallisuuteen liittyviä vaatimuksia.”

Gauntltilla voit tehdä tietoruvaan liittyviä hyväksymistestejä, katso blogista ohjeet alusta loppuun.

VINKIT



- **Älä koskaan aja penetraatiotestaustyökaluja tuotantoympäristöjä vasten.**
- **Kräkkerin ja tietoturvatestaajan ero on lupa. Kokeile työkaluja vain ympäristöihin, joihin sinulla on lupa hyökätä.**
- **Aloita aina tarvearviosta: toisissa ohjelmistoissa on kovat tietoturvavaatimukset, mutta monissa vähempikin riittää.**
- **Ota työkalut käyttöön yksi kerrallaan ja arvioi niiden tuottamien tuloksien hyödyllisyys ennen integrointia devops-putkeen.**
- **Säädä työkalut tuottamaan tarpeellinen määrä varoituksia. Jos virheellisiä löydöksiä on liikaa, ihmiset lakkaavat luottamasta työkaluun. On parempi löytää vain kriittisimmät puutteet ja korjata ne kuin löytää kaikki ja jättää ne omaan arvoonsa.**
- **Kuten muussakin testauksessa, parhaita tuloksia saadaan, jos testiympäristö muistuttaa rakenteellisesti mahdollisimman paljon tuotantoympäristöä, erityisesti verkkoelementtien osalta.**
- **Ohjelmistokehityksen ja operoinnin käytäntöjä on syytä iteroida ohjelmiston elinkaaren edetessä myös tietoturvatestauksen ja monitoroinnin osalta. Projektin edetessä tarvitaan uusia käytäntöjä ja vanhoista voidaan luopua.**

KONTAKT

Eficode Oy

Marko Klemetti
CTO
marko.klemetti@eficode.com
044 522 5927

HEIKKI HÄMÄLÄINEN
Head of DevOps
heikki.hamalainen@eficode.com
+358 (0) 40 766 2610

KAJ JOKINIEMI
Vice President of DevOps
kai.jokiniemi@eficode.com
+358 (0) 40 592 6257

Eficode.fi

Pohjoinen Rautatiekatu 25
00100 Helsinki
+358 (0)207 40 11 22

www.eficode.fi

Silverskin Oy

Jani Kirmanen
Senior Security Advisor
+358 40 173 7440
jani@silverskin.fi

Tommi Lahtinen
Enterprise Defender
+358 40 753 7234
tommi@silverskin.fi

Silverskin Information Security
Meritullintori 3
00170 Helsinki

www.silverskin.fi

Silverskin.fi