TRAFICOM
Transport- och kommunikationsverket
Cybersäkerhetscentret

SÄKER UTVECKLING: MED SIKTE PÅ GODKÄNNANDE

TURVALLINEN TUOTEKEHITYS – KOHTI HYVÄKSYNTÄÄ

Traficom 003/2018 J

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

Transport- och kommunikationsverket
Cybersäkerhetscentret
Telefon: +358 29 534 5000 (växel)
PB 313 (Erik Palméns plats 1)
00561 Helsingfors

www.ncsc.fi
www.kyberturvallisuuskeskus.fi/sv
www.traficom.fi/sv

INNEHÅLL

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.

FÖRORD

Cybersäkerhet spelar en viktig roll vad gäller att skydda samhället, dess sårbara installationer och medborgarna. En effektiv metod för att förbättra cybersäkerheten är att ta itu med potentiella problem redan under framtagningen av mjukvarubaserade produkter och tjänster. Säker utveckling ger bättre systemstabilitet och driftsäkerhet samt förebygger dataintrång och dataläckor.

Till Kommunikationsverkets cybersäkerhetscenters uppgifter hör att godkänna kryptografiprodukter avsedda för att skydda internationellt säkerhetsklassificerad information i Finland. För att uppfylla detta åtagande har den nationella myndigheten för informationssäkerhet (NCSA-FI), som utgör en del av Cybersäkerhetscentret, genomfört bedömningar av kryptografiprodukter inklusive deras utvecklingsprocesser. Produkternas säkerhet har blivit bättre tack vare utvärderingarna. De har dessutom utgjort ett effektivt och förebyggande bidrag till den nationella cybersäkerheten samtidigt som de har främjat försäljningen och exporten av finländska krypteringsprodukter.

Våren 2018 gjorde NCSA-FI en utredning av hur leverantörerna kunde få bättre stöd med säker utveckling och utformning av sina produkter inför bedömning och ackreditering. Föreliggande handbok och en plan för kommunikation om förebyggande åtgärder arbetades fram som en del av utredningen. Utredningen anlitade både branschexperter och NCSA-FI:s egna sakkunniga vid framtagningen av handboken.

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.

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.

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.

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.

SAMMANFATTNING

Syftet med föreliggande handbok är att hjälpa leverantörerna ta fram högvärdiga och säkra produkter. Handboken riktar sig till organisationer som ansöker om godkännande för sina kryptografiprodukter från NCSA-FI och till andra finländska leverantörer som eftersträvar konkurrensfördelar genom informationssäkerhet. Den hjälper läsarna att bättre förbereda sig för bedömning, ger råd för säker produktutveckling och stödjer framtagningen av produkter för myndigheter och för export.

granskning av produkten. Utöver de av produktens egenskaper som omfattar säker utveckling ingår även såväl utvecklingsprocessen som systemförvaltning. De lokaler och den utrustning som används vid utvecklingsarbetet måste vara säkra och personalen ska vara utbildad. Självutvärdering med verktyget för kvalitetsrevisioner Katakri utgör ett bra underlag för att förbereda utvecklingsmiljön och organisationen inför godkännandeprocessen.

Denna handbok är en sammanfattning av de olika momenten i säker utveckling.

Informationssäkerhet består av olika egenskaper, som kryptering, samtidigt som den är en av systemets kvalitetsfaktorer. Därför är det viktigt att inte enbart fokusera på kryptering utan beakta alla egenskaper i produkten.
En säkerhetsbedömning är en övergripande

Ett av de viktigaste verktygen vid utformningen och förvaltningen av en produkt är hotmodellering. Hotmodellen inbegriper fallbeskrivningar av produkten i drift och av hotmiljön samt den värdefulla information och de tjänster som systemet genererar.

Hotmodellerna hjälper dig utvärdera hur väl produkten svarar mot hoten. Det finns fler en ett sätt att ta fram hotmodeller, men det viktigaste är att man gör det och tar till vara resultaten av modelleringen för säker produktutveckling. Med hjälp av hotmodellen kan man också bättre avgränsa och inrikta bedömningarna.

Bättre produktsäkerhet kan uppnås genom sådana etablerade principer för systemets uppbyggnad och utformning som att minimera sårbarhetsytan, använda säkra standardinställningar, sanera indata, minimera rättigheterna, motarbeta på djupet, misslyckas på ett säkert sätt, inte lita på externa tjänster och undvika säkerhet genom otydlighet. Inarbetade och dokumenterade designprinciper underlättar både säker implementering och säkerhetsbedömningar.

De verktyg som används vid implementeringen av en produkt ska stödja säker utveckling. Likaså ska utvecklarna vara insatta i informationssäkerhet, och tredje parters komponenter och plattformar ska vara säkra. Många säkerhetsfrågor uppkommer under programmeringsskedet. Vilka programtekniker som är säkra och bristerna i dem beror på de plattformar, komponenter, programspråk och verktyg som används. Det är viktigt att man är insatt i dem alla. Det finns gott om tillgängligt material om säker programmering, bedömning av informationssäkerheten i beroenden och härdade system. Eftersom sårbarheter i tredje parters komponenter påträffas och publiceras regelbundet är det viktigt att uppdatera produkten för att upprätthålla informationssäkerheten. Uppdateringarna kan i sin tur kräva att produkten ska godkännas på nytt, vilket ur bedömningsperspektivet ökar betydelsen av utvecklings- och förvaltningsprocessernas mognad.

Heltäckande tester hör till kvalitetssäkringen av programvaran, inklusive bedömningen av dess informationssäkerhet. Testerna ska genomföras i en miljö som motsvarar den verkliga driftsmiljön innan produkten lämnas in för godkännande. Målet med testerna och kvalitetssäkringen ska vara att använda reproducerbara och automatiska metoder som möjliggör mer heltäckande tester där effekterna av förändringar i systemet kan mätas på ett effektivt och tillförlitligt sätt. God testbarhet hos produkterna och komponenterna underlättar dessutom godkännandeprocessen. Kodgranskning är en viktig del av kvalitetsbedömningen och den ska också innehålla granskning av informationssäkerheten. Tester, självutvärdering och eventuella tredjepartsbedömningar bidrar också till att höja kvaliteten på godkännandeprocessen.

Denna handbok är en sammanfattning av de olika momenten i säker utveckling. Därutöver är det viktigt att du är insatt i de specifika kriterierna, verktygen och plattformarna på ditt område. Då de verktyg, metoder, tester, lokaler och kompetens som anknyter till utvecklingsarbetet är utan anmärkning, höjer det kundernas förtroende för produkten – och underlättar godkännandeprocessen.

SÄKERHET ÄR VIKTIGT

Denna handbok är en vägledning som hjälper dig utforma högvärdiga och informationssäkra system. Den ger dig riktlinjer för att utforma, implementera och testa säkra system. Det är inte enbart programutvecklarna som behöver vara insatta i säker utveckling, utan det behövs också stöd från FoU-chefer, produktchefer och alla andra som medverkar vid produktutveckling. Utöver vägledning i säker utveckling ger vi också praktiska tips till dig som ansöker eller ämnar ansöka om NCSA-FI:s godkännande för kryptografiprodukter.

Denna handbok är en vägledning som hjälper dig utforma högvärdiga och informationssäkra system.

INFORMATIONSSÄKERHET GER FÖLJANDE KONKURRENSFÖRDELAR FÖR DITT FÖRETAG:

- Större medvetenhet om cybersäkerhet och integritetsfrågor har lett till att kunderna kräver att produkterna och tjänsterna ska vara säkra och högkvalitativa, oavsett om de kan verifiera detta eller inte.

- Genom att vara proaktiv står du bra till vad gäller säkerhetskriterier vid upphandling. Då blir du inte överrumplad när säljchefen meddelar att det behövs belägg för säker utveckling på ”kundmötet om en vecka”.

- När du har skött din del av att säkra systemet minskar sannolikheten för att dina kunder blir drabbade. Kunderna håller dig inte heller lika sannolikt ansvarig förutsatt att du har följt de rutiner som krävs.

- Det är obehagligt och dyrt att svara på it-säkerhetsincidenter.

Säker utveckling reducerar din riskexponering, är bra för verksamhetens kontinuitet och förbättrar arbetets kvalitet överlag. Säkerhet som affärspraxis har en övergripande positiv effekt.

Säkerheten i många mjukvarubaserade produkter var förr ofta undermålig. Vi har sett otaliga fall där leverantörerna sent omsider – till exempel under offertskedet – insett att de borde ha tagit hänsyn till informationssäkerheten. Att börja tänka på säkerhetsfaktorer i ett sent skede av utvecklingslivscykeln är inte smärtfritt för någon av de inblandade – leverantörerna, kontrollörerna och köparna. Det verkar som om företagsledningen inte är helt tydlig om att säkerhetsaspekten är ett krav och att informationssäkerhet därför inte beaktas på ett så framtidsinriktat sätt som vi tror.

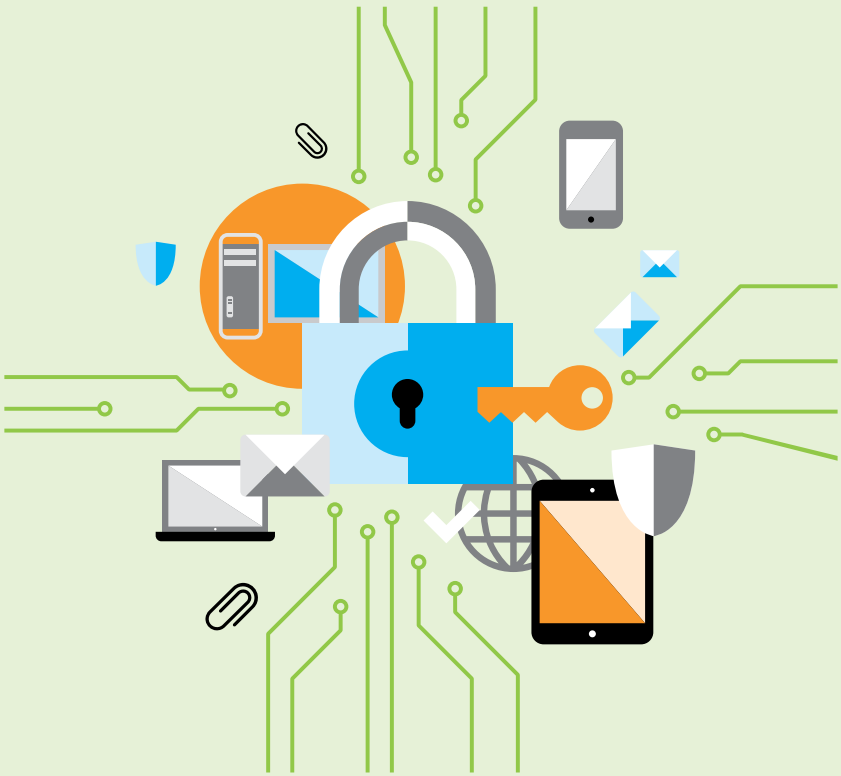
TIPS:

Förberedelserna för en säkerhetsrevision är bra för produkten:

- Om ledningen förväntar sig att du får godkänt betyg i säkerhetsrevisionen, bör den i gengäld vara beredd att satsa på säkerhet, kvalitet och kontinuitet samt att integrera dem i företagskulturen.
- Genom att vara proaktiv är det enklare att klara säkerhetsrevisionen.
- Granskningen i sig kan uppdaga fel som behöver åtgärdas och nya sätt att förbättra din produkt.

Mer läsning:

- *"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 mars 2017*



LOKALER OCH PERSONAL – OPERATIONSSEKRETESS

Säkra produkter tillverkas i säkra miljöer. Det är människor som utformar och implementerar programvaran och människorna behöver verktyg och lokaler för arbetet. Om anläggningar där källkoder, artefakter, verktyg eller arbetsdatorer förvaras blir utsatta för intrång kan detta äventyra de produkter som tillverkas där. Angriparen kan till exempel lägga in en bakdörr till källkoden. Säkerhetsfrågor ska med andra ord ses som någonting som gäller alla delar av systemet, inklusive personalen och lokalerna.

De verktyg som används för programutvecklingen ska vara anpassade till säkerhetskraven för själva produkten, och bland annat versionskontroll – som spårar alla redigeringar och vem gjort dem – är ett måste. Servrarna ska vara uppdaterade med de senaste säkerhetsfixarna och alla användare bör ha unika konton för loggning. En del av de populära molntjänsterna för programutvecklare kan vara bättre på att säkra och uppdatera sina plattformar än organisationer som saknar tillräckliga resurser för att säkra sin interna utvecklingsplattform.

Folk bör känna till de allmänna principerna för cyberhygien.

De populära modellerna för systemutvecklingslivscyklar kräver programutvecklarutbildning. Detta bör omfatta alla de medarbetare som deltar i utvecklingsarbetet. De bör känna till de allmänna principerna för cyberhygien (rutiner för bibehållande av hög it-säkerhet), som bland annat att inte klicka på alla inkommande e-postmeddelanden, vara försiktiga när de surfar på webben och helst inte använda arbetsdatorn för fritidssurfande över huvud taget, inte plocka upp och koppla in lösdrivande USB-minnen, uppdatera sina system och känna till grunderna av social manipulering. Utvecklare bör få ta del av ytterligare utbildning om säker design, hotmodellering och säker programmering.

Eftersom medarbetare byts ut måste utbildningen vara en integrerad del av introduktionen för nya anställda. Att förlita sig på en eller två allt-i-allo programmerare är ingen säker strategi vare sig för produkten eller för företaget.

Bärbara datorer som innehåller utvecklingsrelaterat material ska vara uppdaterade med de senaste säkerhetsfixarna och hårddisken ska vara krypterad.

Utöver lokalerna och personalen ska du också tänka på de processer som används i din organisation. Understöder de utvecklingen av högkvalitativa och säkra produkter? Enligt OWASP (www.owasp.org/index.php/Policy_Frameworks) *behövs åtminstone följande för en säker tillämpning:*

- *Organisationsförvaltningen som en förkämpe för säkerhet.*
- *En nedtecknad it-säkerhetspolicy som bygger på nationella standarder.*
- *En utvecklingsmetodologi med tillräckliga säkerhetskontrollpunkter och kontrollåtgärder.*
- *Säkra processer för leveranser och konfigurationshantering.*

Vidare enligt OWASP: "Ad-hoc utveckling sker alltför ostrukturerat för att kunna resultera i säkra applikationer. Därför ska organisationer som vill skapa säker programkod konsekvent tillämpa metoder som främjar detta mål. Välj noga: Små team bör aldrig överväga att tillämpa tungrodda metoder med många olika roller, medan stora team ska välja metoder som kan anpassas enligt deras behov."

TIPS:

Använd gärna verktyget för informationssäkerhetsauditering för myndigheter (Katakri) som underlag för självutvärdering av lokalerna, processerna och organisationsstrukturen. Verktyget kan användas av vem som helst som en checklista för översyn av organisationen och lokalerna. Extern bedömning följer sannolikt samma mönster, och om ditt företag är i linje med Katakri har den goda förutsättningar att klara bedömningen.

Mer läsning:

- *"Katakri 2015 – Verktyg för informationssäkerhetsauditering för myndigheter (på finska och engelska)"*

KRAV- OCH HOTMODELLERING

De funktioner som mjukvaran förväntas ha definieras i programkravspecifikationen i form av användningsfall eller kravförteckningar. I små projekt kan specifikationen vara mycket informell och ibland är den inte ens nedskriven.

Att fastställa de rätta kraven är svårt, men det är likafullt nödvändigt för att projektet ska lyckas. Oftast är det omöjligt att ha en komplett kravspecifikation redan i början av ett projekt – före utförande och implementering – eftersom både utvecklaren och kunden lär sig mer medan projektet framskrider. I dag är arbetsmomenten inordnade i en cykel bestående av kravspecifikation → design → implementering → verifiering osv. Säkerhetskrav kan vara ännu svårare att definiera än rent funktionella krav. ”Programmet får inte krascha” är ett bra mål för säker mjukvara, men det är alltför vagt formulerat för att kunna tas som ett kontrollerbart krav. Nedan diskuterar vi detaljerade säkerhetskravspecifikationer med hjälp av hotmodellering.

SÄKERHETSKRAV

Ibland finns det brister i kraven eller inga krav över huvud taget, vilket leder till att tid går förlorad både under design och implementering och att programvaran inte håller måttet. Detsamma gäller säkerhetskraven. De fastställer hur informationen och tjänsterna i systemet skyddas mot hotaktörer och andra missöden. Säkerhetskraven kan bland annat ange hur användarautentiseringen ska ske eller vilka data i systemet ska vara krypterade.

Precis som alla andra krav kan även säkerhetskraven vara såväl funktionella som icke-funktionella. Ett funktionellt krav kan vara att användarna ska logga in med användarnamn och lösenord.

Ett icke-funktionellt krav är till exempel att tillämpningen ska validera alla indata via ett nätverk och avvisa alla ogiltiga begäran.

För att kunna skapa robusta säkerhetskrav måste vi föreställa oss hur produkten kommer att användas i verkligheten och i hurdana miljöer. Kommer systemet att vara fysiskt isolerat i en bunker med väktare? Kommer det att finnas i en molntjänst? Har det komponenter som körs i en webbläsare? Vilka värdefulla tillgångar (oftast data) kommer systemet att hantera?

TIPS:

Skriv ner dina säkerhetskrav, även de implicita kraven. Nedtecknade säkerhetskrav ger dig en säkrare produkt och även säkerhetsbedömningarna är effektivare.

När vi först har beskrivit hur produkten är tänkt att användas kan vi sedan tänka på vad som kan gå fel. Kan produkten missbrukas? Och i så fall av vem, hur och när? Kan en angripare få obehörig åtkomst till någonting eller orsaka att du förlorar någonting värdefullt? Kan obehöriga komma åt servrarna och stjäla data eller till och med fysiska diskar? Tänk om webbläsarkomponenten utsätts för dekompilering och ersätts med ett skadligt klientprogram. På vilket sätt kan det påverka det drabbade systemet?

Det finns många olika sätt att äventyra ett system på och angreppen kommer i olika former. De flesta kända angrepp har ändå vissa gemensamma egenskaper och kan ordnas in allmänna kategorier. Exempelvis OWASP:s lista på de 10 största säkerhetsriskerna på webben omfattar följande sårbarhetskategorier vilka till stor del är relevanta även på andra områden än i webbapplikationer.

1.

Injektionsattacker:
Applikationen tar emot inmatade data utan att validera dem ordentligt. Det ger angriparen möjlighet att köra kommandon eller genomföra andra skadliga handlingar i den sårbara applikationen.

2.

Defekt autentisering:
Användarautentiseringen har inte implementerats korrekt. Lösenord verifieras inte, lösenord läcker ut eller lösenordsåterställning används för att angripa systemet. Det kan också finnas defekter i sessionshanteringen efter autentiseringen så att en angripare kan kapa sessioner.

3.

Otillräckligt skydd för känsliga data:
Känsliga data lagras eller transporteras i klartext eller saknar otillräckligt skydd.

4.

Externa referenser i XML-dokument (XXE):
Applikationen har osäker hantering av XML-filer, som felaktigt tillämpad SAML för samlad inloggning.

5.

Defekt åtkomstkontroll:
Applikationen tillåter åtgärder som inte hör till användarnas behörigheter.

6.

Felaktiga säkerhetskonfigurationer:
Systemet är inte härdat eller kör onödiga funktioner eller plattformen är gammal och sårbar eller innehåller osäkra gästkonton.

7.

Olämplig körning av skript (XSS):
Angripare kan köra oönskade HTML eller Javascript.

8.

Osäker läsning av lagrade data (osäker deserialisering):
Angripare kan manipulera data/objekt som applikationen deserialiserar och sedan använder för privilegier.

9.

Komponenter med kända sårbarheter:
Applikationen innehåller – avsiktligt eller oavsiktligt – komponenter med kända sårbarheter.

10.

Undermålig loggning och övervakning:
Applikationen har undermålig säkerhetsloggning med tanke på utredning eller den saknar kontroller eller aviseringar för angrepp.

Det finns många olika typer av ”missbrukare” – från skriptknattar till vinstjagande brottslingar och vidare till statliga aktörer – med sina respektive egenskaper. Vilka som är relevanta för dig beror på hur det planerade systemet kommer att användas.

Tänk också på vilken motivation någon kan ha för att hacka in i systemet och de värdefulla tillgångarna i det samt på vilken inverkan ett lyckat angrepp kan ha på dina kunder och dig.

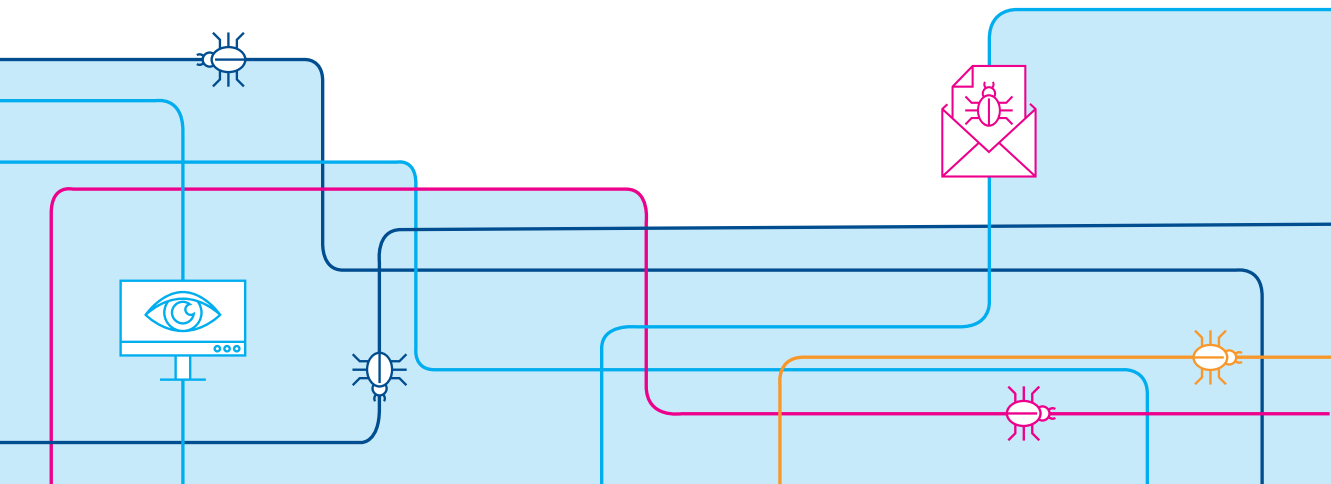
HOTMODELLERING

Hotmodellering avser utvärdering av systemets säkerhet. Vissa allmänna principer för hotmodellering ingick redan i diskussionen om säkerhetskrav ovan. Det finns flera olika metoder för att genomföra hotmodellering, men de kräver inga särskilda färdigheter eller kunskaper.

”Det finns flera olika tillvägagångssätt för hotmodellering... Vilken metod som används för riskbedömningen är inte alls lika viktigt som att man verkligen genomför en strukturerad hotmodellering. Microsoft konstaterar att den viktigaste faktorn i bolagets program för att höja säkerheten var att införa hotmodellering på bolagsnivå.” - Threat Risk Modeling - OWASP

Det är mindre en fråga om hur du genomför modelleringen, bara du gör det. Tänk som en angripare och bygg in säkerhetskrav i din åtgärdsplan. Systemet ska ha högvärdig arkitektur för hotmodellering eftersom olika komponenter har olika roller och säkerhetsegenskaper och innehåller information med olika säkerhetsnivåer. Eftersom utformningen av arkitekturen normalt betraktas som en del av designen bör du tillämpa iterativ systemutveckling, med iterationer mellan systemuppbyggnad och hotmodellering.

Om du har svårigheter att få det gjort bör du gärna anlita utomstående konsulter för att underlätta processen.



TIPS:

Om din produkt ska genomgå säkerhetsrevision kommer revisionen att omfatta hotmodellering. Revisorerna kan fokusera på en viss uppsättning egenskaper i produkten och de utgår sannolikt från specifika scenarier och miljöer för användningen. Om din produkt innehåller till exempel tjugo egenskaper är det möjligt att endast fem av dem granskas. De övriga femton egenskaperna ska eventuellt kopplas bort såvida de inte har någon inverkan på den allmänna säkerheten.

Det är möjligt att revisorernas hotmodell inte matchar din modell. De fokuserar på de hot som är relevanta för det aktuella uppdraget. I egenskap av en teknikleverantör ska du vara insatt i vad dina kunder kräver och samarbeta med dem för att ta fram en relevant hotmodell. Vilka typer av data och på vilka säkerhetsnivåer kommer de att hantera med ditt system? Hur påverkas kunden av ett eventuellt säkerhetsbrott? Vem är bovarna eller fienderna som kunden måste se upp för? Vem kommer att använda systemet i framtiden?

Revisorerna kommer sannolikt att använda systemet ur sitt perspektiv med fokus på användarfall och scenarier som är relevanta för deras uppdrag. De observerar hur systemet fungerar och återkopplar till den hotmodell som håller på att utforma. Täcker hotmodellen de aktuella användningsscenarierna för och egenskaperna hos produkten?

Det här är nyttigt för dig om du redan har ett system men är inte helt övertygad av dess säkerhetsegenskaper. Genom att sätta dig in i prioriteterna hos den kund som kommer att bedöma din produkt skapar du en att-göra-lista på vad du ska prioritera när du uppdaterar säkerheten. Frågor som saknar omedelbar betydelse för kunden kan skjutas fram till senare projekt.

INBYGGDA VS. PÅBYGGDA SÄKERHETSLÖSNINGAR

Med inbyggda säkerhetslösningar avses ofta att säkerhetskraven har beaktats redan från början av utvecklingsprojektet och att säkerhetsegenskaperna på så sätt är integrerade i produkten. Motsatsen är påbyggda säkerhetslösningar där säkerhetskraven tas i betraktande först efter implementering av produkten och genomförs genom att lägga till nya komponenter och egenskaper.

Inbyggda säkerhetslösningar anses vara bättre på flera grunder. Att lägga till nya komponenter medför alltid problem med integration och tilläggskostnader samt ökar sannolikheten för sårbarheter. Allt detta kan undvikas med inbyggda säkerhetslösningar där säkerhetsaspekten är integrerad i komponenterna. Säkerhet är också en av kvalitetsfaktorerna vid sidan av underhållsmässighet, testbarhet, dataflöde och användbarhet. Det är svårt att ingjuta säkerhet i en produkt genom att lägga till nya komponenter.

Genom att ta hänsyn till säkerhet från början och utbilda utvecklingspersonalen förankrar du säkerhetstänket i teamen och främjar de senare skedena av projektet. Säkerhetsbedömningar, säker implementering och säkerhetsfixar utgör grunden för en säker produkt.

Påbyggda säkerhetslösningar – som brandväggar, sandboxing av programvara och intrångsdetekteringssystem – kan ge ytterligare säkerhet. De är ändå bara nödfallslösningar om produkten inte är framtagen med solid fokus på säkerhet. Om detta är fallet, rekommenderar vi att du börjar bygga på med säkerhetslösningar så fort som möjligt i de följande programleveranserna.

TIPS:

Om den inbyggda säkerheten i din produkt inte håller måttet eller produkten ska användas i synnerligen krävande förhållanden är det möjligt att du efter säkerhetsbedömningen är tvungen att stärka skyddet. Detta inbegriper också fysiska hinder och brandväggar.

INTEGRITET

Informationssäkerhet anknyter till integritet och skydd av personuppgifter. Trots att informationssäkerhet är en förutsättning för integritetsskydd tas frågor som lagringstider för personuppgifter och skyldighet att informera kunderna om personuppgifter lagras inte upp i denna handbok. Vi nämner detta bara för att du inte ska vaggas in i tron att du inte behöver tänka på dessa frågor.

Mer läsning:

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

DESIGN

När du är klar med kravspecifikationerna är det dags att utforma systemet. Förhoppningsvis tillämpar du iterativ systemutveckling och återkopplar till kraven och utformningen flera gånger. Det underlättar arbetet eftersom allting inte behöver vara perfekt från start.

Utformningen avgör hurdan systemarkitektur som behövs för att möjliggöra de specificerade funktionerna och egenskaperna. Arkitekturen har också en stor inverkan på hur enkelt (eller svårt) det är att bygga upp ett säkert system. I samband med hotmodelleringen för den planerade systemarkitekturen kommer du sannolikt att uppdaga flera ändringar i den som gör det enklare att skapa en säker produkt.

PRINCIPER FÖR SÄKER DESIGN

Det finns vissa tumregler för säker design. De som presenteras här följer principerna för säkerhet genom design (OWASP Security by Design Principles).

Minimera sårbarhetsytan

Med sårbarhetsyta (eller attackyta) avses den del av systemet som är exponerad mot omvärlden, antingen fysiskt eller via nätverk eller filer. Alla angrepp förväntas komma via denna yta, förutsatt att du inte har missat någonting.

Försök hitta sätt att minimera sårbarhetsytan genom att eliminera sådana gränssnitt som inte är nödvändiga. Till exempel: Behövs det en USB-port i enheten eller kan porten täckas över för att göra det svårare att komma åt den? Kan vissa förvalda funktioner i de operativsystem som en plattform körs på kopplas bort för att härda plattform? Vi återkommer till plattformhärdning senare i denna handbok. Ska administratörsåtkomst tillåtas bara från en lokal värddator? Kan webbtjänsten i systemet eventuellt byggas med endast statiska sidor och inte med dynamiska webbsidor på servern?

Att reducera sårbarhetsytan har många fördelar. Du behöver ta hänsyn till färre hotscenarier och utrymmet för misstag vid implementeringen minskar, vilket gör det enklare att testa systemet.

Fastställ säkra standardinställningar

Dina kunder ska inte behöva vara experter för att kunna använda systemet på ett säkert sätt. Dessvärre är systemen oftast osäkra från början och det blir systemadministratörens uppgift att säkra dem. Eftersom en administratör inte är lika insatt i systemet som systemutvecklare är det viktigt att systemet levereras med säkra standardinställningar och minimalt behov av konfigurering. Alla åtgärder som försvagar säkerheten ska följa av användarens medvetna beslut. Tänk på ditt ansvar. Vill du ta ansvaret för att ett system som levereras är osäkert från början, eller vill du hellre låta administratören fatta beslut om att sänka säkerheten efter en bedömning av riskerna?

Sanera indata[1]

Utebliven validering av indata kan göra det möjligt för angripare att manipulera eller sabotera någon sårbar del av systemet. Dessutom kan de ofta också köra kommandon i systemet eller på annat sätt kontrollera det. Om möjligt, tänk på att validera alla indata som matas in i systemet utifrån. Med validering avses att indata (t.ex. meddelanden) syntaktiskt följer reglerna och är semantiskt korrekt.

Kryptering och integritetskontroller används ofta för att skydda data i transit men om du inte har fullkomlig kontroll över indatakällan måste också krypterade indata genomgå noggrann validering.

[1] Finns inte på OWASP:s lista.

Glöm inte att gränssnitt kan vara indirekt exponerade. Meddelanden från angriparen kan förmedlas djupt in i systemet. Indata kan spridas i systemet till exempel så här:

1. En webbserver tar emot ett användarnamn och ett lösenord (inloggningsuppgifter).

2. Servern använder en automatisk autentiseringsfunktion för att vidarebefordra inloggningsuppgifterna till autentiseringsservern.

3. Autentiseringsservern kontrollerar uppgifterna mot en databas.

4. Användarnamnet registreras i loggfilen.

5. Administratören granskar loggen i sin webbläsare.

Beakta alla gränssnitt. Försäkra dig om att det finns åtminstone någon person som förstår och dokumenterar dataflödena i systemet. Lär ditt team att utgå ifrån att fientliga indata kan nå deras komponenter oavsett var de är.

Fuzzning är en utmärkt metod för att testa indatasanering. Vi diskuterar det närmare i avsnittet om tester.

Försäkra dig om att det finns åtminstone någon person som förstår och dokumenterar dataflödena i systemet.

Separera uppgifter

Separering av uppgifter avser till exempel en situation där en person lämnar in en ansökan om reseersättning och en annan person ska granska och godkänna ansökan. Samma logik gäller också mjukvarukomponenter. Ett klassiskt exempel är att överföra ansvaret för ändringsloggar till ett annat system som inte kan påverkas samtidigt som systemet som genererade loggen. Aktivitetsloggar till en databas ska sparas en separat databas. Om databasen blir utsatt för intrång kan du inte längre lita på loggen för att spåra händelserna.

Genom att separera uppgifterna mellan olika komponenter kan du också finfördela de tillgängliga resurserna och rättigheterna.

Tilldela minsta möjliga rättigheter

När du först har fördelat uppgifterna mellan komponenterna bör du sedan minimera åtkomsträttigheterna till de olika processerna eller undersystemen. Varje komponent ska tilldelas det minimiantal åtkomsträttigheter som behövs för att genomföra komponentens uppgift. Detsamma gäller användarna: Alla behöver inte administratörsbehörigheter och administratörerna bör inte heller vara allsmäktiga. Då rättigheterna tilldelas enligt minimiprincipen kan man med större sannolikhet begränsa intrången och angriparens åtkomst till systemet. Du kan också se på frågan från andra hållet: Om en komponent kan fungera med minimibehörigheter är säkerheten i komponenten mindre kritisk för systemet än om komponenten har administratörsbehörighet.

TIPS:

Du vill inte råka ut för att behöva erkänna under säkerhetsbedömningen att din kod har tilldelats de största möjliga rättigheterna. Du bör kunna redogöra för vilka rättigheter som används och var de används.

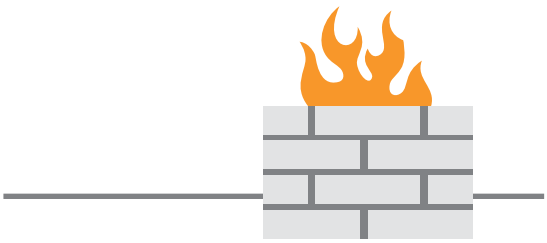
Motarbeta på djupet

Försvar på djupet innebär att du lägger flera försvarsnivåer i systemet. Alla säkerhetskontroller kan innehålla fel som gör systemet utsatt för intrång. Dessutom hittar en beslutsam angripare på sätt att kringgå enskilda kontroller. Genom djupförsvar stärker du systemets motståndskraft då intrång genom ett skikt inte utsätter hela systemet.

Du kan till exempel inte utgå ifrån att enbart brandväggen räcker till som skydd. Brandväggar har oftast regler för att släppa igenom trafik. Ett sabotageprogram kan ta sig igenom skalskyddet med hjälp av e-post. En bärbar dator som normalt är uppkopplad mot ditt nät kan infekteras under en arbetsresa. Användare surfar på webben och kan bli utsatta för intrång via skadliga webbplatser som ger angriparen fotfäste i ditt nätverk. Försvar på djupet betyder att du även om du har en brandvägg ska härda det interna nätet, kryptera all intern trafik och kräva autentisering för åtkomst till interna tjänster. På så sätt stärker du motståndskraften även om brandväggen utsätts för intrång.

Inget systemkonto ska ges obegränsad åtkomst till systemet, utan det bör i stället finnas flera olika användarroller som tilldelas enligt principen om minsta rättigheter. Användarnas åtgärder ska kunna spåras i en logg som inte går att manipulera. Åtkomst till de roller som har mer behörigheter ska kräva tvåfaktorautentisering (tvåstegsverifiering).

Ett annat begrepp för detta är *säkerhet på djupet*.



Säkra dig för fel

Var beredd på att någonting kommer att gå fel. Hårdvara går sönder, nätuppkopplingar bryts, batterier töms, program kraschar och så vidare. Du bör utforma systemet så att fel inte äventyrar systemets säkerhet, vilket ibland kan vara svårt. Du kan välja mellan ändläge ”öppen” (fail open) eller ändläge ”stängd” (fail closed). Ska systemet tillåta eller vägra åtkomst i felläge? Om den komponent som ger användarna åtkomst till systemet är onåbar kan det vara bra att alltid förbjuda åtkomst. Å andra sidan kan vägran att ge åtkomst få drastiska följder – till exempel leda till förorening av dricksvattnet för många människor – och då bör du noga överväga vilken strategi du väljer. Kärnan i vårt budskap är: Du bör inte strunta i vad som händer om en komponent fallerar.

Misstro externa tjänster

Ditt system använder troligen tjänster från externa system, och du har inte kontroll över deras mjukvara, lokaler och personal. Tredjepartstjänster kan också utsättas för intrång, och du ska inte ge angripare en chans att ta sig in den vägen.

I enlighet med principerna om minsta rättigheter och djupförsvar ska du inte lita blint på externa tjänster. Du ska förhålla dig till dem som till externa aktörer, validera alla data från dem och ha säkra rutiner för lägen då tjänsten är utslagen.

Du ska inte begränsa din syn på vad som kan betraktas som externa tjänster utanför din kontroll. Tänk till exempel på webbaserade system där användarens webbläsare är en extern tjänst som kör den del av din kod. Du kan inte lita på de säkerhetskontroller som finns i användarens webbläsare utan ska ha obligatoriska säkerhetskontroller på din server. Detsamma gäller alla system där komponenter körs i klientsystem.

Undvik säkerhet genom otydlighet eller sekretess

Med säkerhet genom otydlighet avses att informationssäkerheten bygger på att utformningen eller implementeringen av produkten hålls hemlig. Dessvärre läcker hemligheter ut och system kan också kopieras genom rekonstruktion för att få fram vad de innehåller. Du ska inte förlita dig på säkerhet genom otydlighet. Medan de data och åtkomstuppgifter som systemet hanterar kan vara konfidentiella, bör systemet i sig tåla granskning. Begränsa de hemligheter som systemet behöver för att utföra sin uppgift. Tänk också på hur dessa hemligheter kan justeras när de läcker ut. Går det enkelt att byta ut de privata nycklarna i systemet? Hur enkelt kan systemägaren begära att användarna ändrar sina lösenord om dessa äventyras?

Trots att du genom att hemlighålla systemets uppbyggnad och källkod kan lägga till en extra säkerhetsbarriär ska du inte enbart lita på den.

Föredra det enkla

All mjukvara och komponenter kan innehålla fel och sårbarheter. Det gäller även säker mjukvara och säkerhetsegenskaper. Mindre kod ger färre fel och omvänt: fler konfigurationsmöjligheter innebär fler konfigurationsfel. Sikta på att göra systemet så enkelt som möjligt och ifrågasätt alltid det skenbara behovet av mer komplexitet.

Det är lättare att se över och säkra ett enkelt system än ett komplext. Att undvika onödig komplexitet är också ett smart ekonomiskt val. Enkla system är mer lättbegripliga och därmed mer kostnadseffektiva att utveckla och underhålla.

TIPS:

Ju mindre omfattande säkerhetsrevision det behövs, desto mindre kostar revisionen. Du bör alltså antingen se till att du har en enkel produkt eller vara beredd att bevisa att endast en del av produkten är kritisk för de användarfall för vilka den ska godkännas.

Var beredd att åtgärda säkerhetsfrågor på rätt sätt

När du får information om en sårbarhet i ditt system bör du antagligen fixa eller åtgärda den. Då känner du av nyttan med ändamålsenliga utvecklings- och testprocesser. Du bör känna dig bekväm med att fixa buggar och ge ut nya systemleveranser. Vid ad-hoc utveckling med otränad personal och bristande testautomatik innebär varje ändring en risk och då kan det kännas lockande att ignorera problemet.

Vi har sett många situationer där en säkerhetsbugg har fixats bara på ett ställe i systemet då det egentligen behövs flera fixar i kodbasen. Det kan bero på att man har varit rädd för biverkningar och därför har valt att fixa så lite som möjligt. Att fixa ett av flera likartade problem bidrar inte nämnvärt till säkerheten. Vi har också sett fall där leverantören fixar buggen i underhållsversionen men inte i följande leveransversion, så att problemet dyker upp igen i följande större uppdatering.

Det händer ibland att leverantörer väljer att göra minsta möjliga modifieringar i underhållsversionen av produkten och skjuter upp större fixomgångar eller refaktorisering till följande större leverans.

Tidskänsliga buggar kan också upptäckas då dina nyckelutvecklare är på semester eller sjuka. Det är ytterligare en orsak varför du bör ha lämpliga processer och utbildning så att det går att delegera sådana problemfixar till de medarbetare som du har att tillgå.

Du bör känna dig bekväm med att fixa buggar och ge ut nya systemleveranser.

VAL AV PLATTFORM

Din produkt körs på en eller flera plattformar. Vanliga plattformar är bland annat Linuxsystem för servrar, Android för mobila enheter, olika molnplattformar för grundsystem, Docker för undersystem, Microsoft Windows och .NET och så vidare. De olika plattformarna har specifika säkerhetsegenskaper och du måste vara insatt i egenskaperna i den plattform som du bygger din produkt på.

Plattformar uppdateras tidvis med nya egenskaper som svar på uppdagade säkerhetsproblem. Du bör ta det i betraktande när du lägger upp leveranscykeln och livscykeln för din produkt. Du behöver titta på leveranshistoriken för den plattform eller de plattformar som du överväger att använda för att få en känsla för säkerheten och uppdateringarna.

När du har satt dig in i plattformen ska du bestämma när du behöver uppdatera din produkt efter uppdateringar av plattformen. Ska du beakta alla uppdateringar, de största uppdateringarna, bara säkerhetsuppdateringarna eller inga alls?

Att inte beakta säkerhetsuppdateringar är självfallet problematiskt. På plattformar med öppen källkod kan du plocka ut de relevanta uppdateringarna och på så sätt upprätthålla en egen implementation av plattformen. I så fall ska du vara medveten om att antivirusprogram kan ge falskt positivt utslag och att du ska kunna bevisa att du har åtgärdat sårbarheten.

TIPS:

Säkerhetsrevisioner omfattar också de underliggande plattformarna. Revisorn kontrollerar säkerhetsegenskaperna i den plattform som används i produkten mot de vanligaste fallgroparna för plattformen i fråga. Ett vanligt krav i säkerhetsbedömningar är att alla egenskaper i plattformen som inte är nödvändiga för produktens funktion ska kopplas bort eller, om möjligt, tas bort. Du behöver också redogöra för din uppdateringspolicy och hur du håller dig underrättad om plattformuppdateringar för att se till säkerheten i din produkt.

MJUKVARUKOMPONENTER

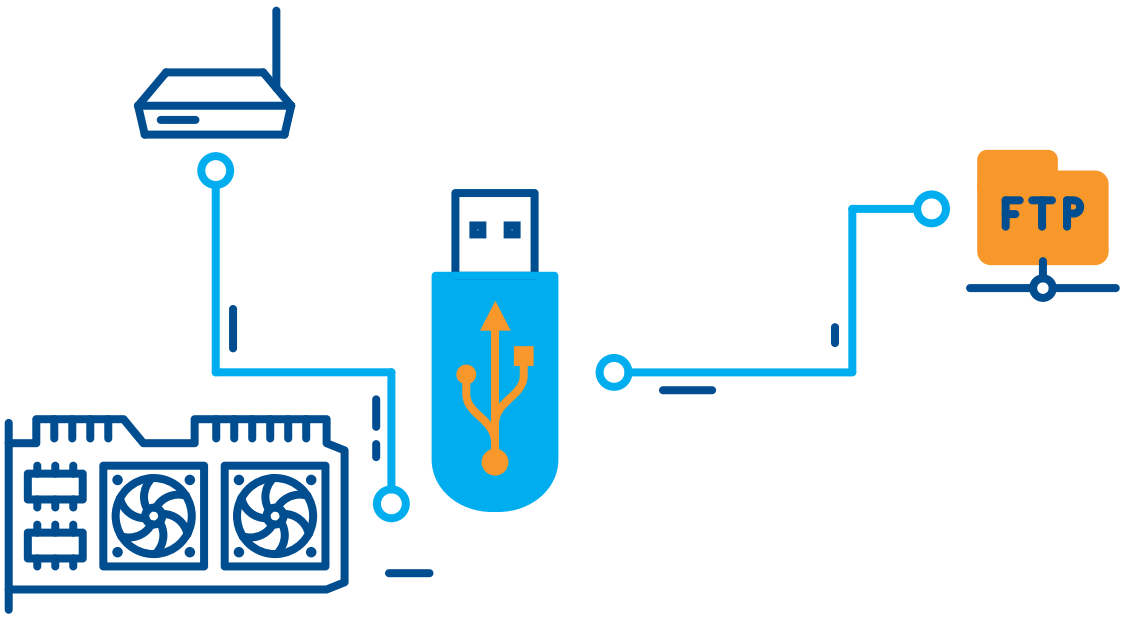
Din produkt innehåller olika komponenter. Du kan tänka på dem inordnade enligt olika abstraktionsnivåer, som mjukvaruklient, grundsystem, VPN-terminator, slumptalsgenerator, mjukvarubibliotek och så vidare. Vissa komponenter kan du implementera in-house, andra går att köpa eller utkontraktera, vissa är gratisvara och en del kanske till och med har öppen källkod. Du måste absolut känna till vilka komponenter din produkt verkligen använder! Förteckningen över de delar som ingår i en produkt kallas stycklista (ibland även bom-lista, från ”bill of materials”).

En del komponenter är mer säkerhetskritiska än andra. Normalt ges komponenter som anknyter till den centrala affärslogiken eller säkerheten mer hänsyn. Dessvärre kan även ett ordinärt bildhanteringsbibliotek i ett säkerhetsintensivt meddelandeprogram ge upphov till en fatal sårbarhet när den återger en indexbild av den person som försöker kontakta dig. Vi uppmanar dig att behandla alla komponenter som kritiska och att rensa bort alla mindre nödvändiga delar och på så sätt minska komplexiteten i systemet.

Du kanske inte heller anser att installationskoden är kritisk med tanke på säkerheten. Men intrång i installationsrutinen kan leda till att den missbrukas för att installera en manipulerad version av programvaran.

När du har lagt upp arkitekturen och komponenterna kan du tänka på säkerhetsegenskaperna i varje enskild komponent:

- Komponenterna bygger på olika tekniska lösningar och plattformar. Är du insatt i hur de påverkar säkerheten i din produkt?
- Uppdateringsfrekvensen för olika komponenter varierar. Hur tänker du synkronisera dessa uppdateringar med uppdateringarna för din produkt?
- Olika komponenter kräver olika åtkomstbehörigheter. Hur ska du tillämpa principen om minsta nödvändiga rättigheter på dem?
- Komponenternas funktioner kan avvika en hel del från din kärnprodukt. Behövs det olika testmetoder och kvalitetssäkringsmetoder för dem?



Försörjningskedjor

Tredje parters komponenter utgör försörjningskedjan för din mjukvara. Eftersom leverantörerna av dina komponenter sannolikt anlitar underleverantörer, kan försörjningskedjan vara lång. Uppdateringar i de olika komponenterna förs vidare genom försörjningskedjan och i slutändan är du tvungen att besluta vilka uppdateringar som ska införas.

Kryptobiblioteket Open SSL är ett bra exempel på en populär men besvärlig komponent i försörjningskedjan. Den har buntats ihop med otaliga inbyggda system, appar, tjänster och andra komponenter, programspråk och plattformar.

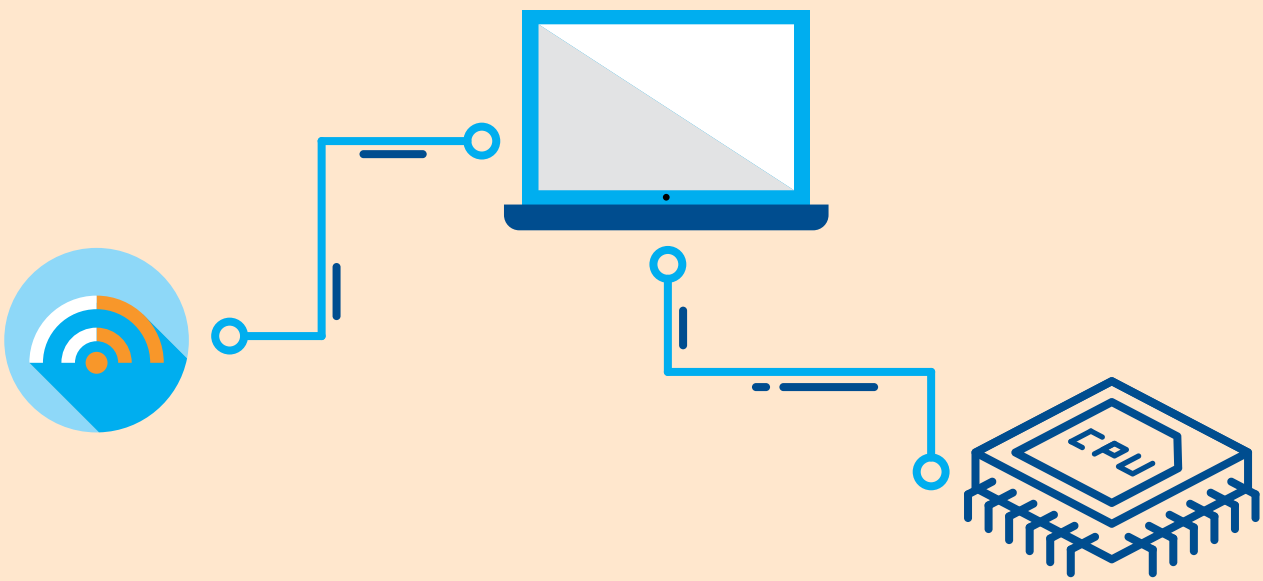
Författarna har ofta sett produkter innehållande flera olika versioner av Open SSL buntade ihop inuti dem. Eftersom Open SSL förvaltas aktivt och sårbarheter uppdagas ofta i den, är också uppdateringsfrekvensen relativt hög. Om din produkt använder TLS-protokoll eller X.509-certifikat är det högst sannolikt att du använder Open SSL – antingen direkt eller indirekt.

TIPS:

I samband med en säkerhetsrevision vill revisorerna få en uppfattning av din försörjningskedja för komponenter. Du bör vara beredd att visa upp din version av stycklistan, inklusive relevanta hårdvarukomponenter. Revisorerna kan använda verktyg för att dubbelkontrollera din komponentlista mot de komponenter som verkligen ingår i produkten. Komponenter som är gamla, överflödiga, har ett tvivelaktigt rykte eller är annars främmande ger sannolikt upphov till frågor som du ska vara beredd att svara på.

Mer läsning:

- ***OWASP: Security by Design Principles***



SÄKER PROGRAMMERING

Programmering innebär att designen omvandlas till en applikation som förhoppningsvis uppfyller de ursprungliga kraven. Dessvärre är det lätt hänt att sårbarheter slinker in i produkten under programmeringsskedet. Lyckligtvis är detta ett välkänt problem och det finns olika slags resurser, handböcker och kurser i säker programmering. Du ska känna till säkerhetsställningen hos den plattform och de externa komponenter som du använder. Du bör också vara medveten om vilken inverkan programmeringsspråket och de övriga verktygen har. Att skriva säker kod är möjligt – och lika svårt som att skriva felfri kod. Därför är det nyttigt att inkludera ett säkerhetsperspektiv i kodgranskningarna.

Din kod ska vara väldokumenterad, modulär, läslig, testbar och testad. Orsaken till det är att du ska kunna underhålla koden under en längre tid och lägga in fixar utan att äventyra säkerheten i koden.

Din kod ska vara väldokumenterad, modulär, läslig, testbar och testad.

Statiska analysverktyg är inriktade på att automatiskt hitta fel – inklusive säkerhetsproblem – genom att analysera källkoden. Dessa verktyg kan vara till stor hjälp och det finns både gratislösningar och kommersiella alternativ för de flesta programspråk. De dåliga nyheterna är att många av dem ofta ger falskt positivt utslag, liksom varnar för kodkonstruktioner som inte egentligen utgör något problem. Du bör alltså lägga både tid och kraft på att lägga in statisk kodanalys i din produktutveckling, i synnerhet om kodbasen är stor och inte tidigare har analyserats.

KRYPTOGRAFI

Praktiskt taget alla system och produkter behöver ha kryptografiska funktioner eftersom de ska överföra eller lagra konfidentiell information, autentisera användare och tjänster och så vidare.

Kryptografi är en av hörnstenarna för säkerhet. Det är ytterst viktigt att du lägger den stenen rätt. Den första regeln är att du inte ska uppfinna hjulet på nytt. Använd kända, högvärdiga kryptografibibliotek i din produkt och undvik hemgjorda hopkok. Du ska alltid följa etablerade standarder och bästa praxis istället för att hitta på egna.

En vanlig fallgrop är att man använder en svag slumptalsgenerator. Många kryptografiska funktioner kräver verkligen starka slumptal och att generera sådana är inte lätt. Du bör sätta dig väl in i tillförlitliga sätt att generera slumptal på den plattform som du planerar att använda

Det är lätt att göra misstag i användningen av kryptografiska funktioner. Många tillämpningar har bristfällig validering av certifikat innan de litar på dem (se Common x509 certificate validation/creation pitfalls for examples). Använd alltså välkända och etablerade tekniker och lär dig att använda dem rätt.

TIPS:

I en säkerhetsrevision ligger fokus speciellt på de kryptografiska funktionerna i din produkt. Utöver en ingående granskning av utformningen och koden, kommer koden dessutom att utsättas för granskning och felsökning då den körs. Slumptalsgeneratorn kommer helt säkert att granskas. Revisorerna kommer att fråga vilka standarder eller kända implementeringar din kryptering baserar sig på.

Tillförlitlighetskraven för kryptering finns på finska på:
https://www.viestintavirasto.fi/attachments/tietoturva/Kryptografiset_vahvuusvaatimukset_-_kansalliset_suojaustasot.pdf

En bra allmän handledning finns på:
http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf
Specifika råd för ditt användarfall fås direkt från NCSA-FI.

HANTERA BEROENDEN

Modern mjukvaruutveckling handlar ofta mer om att lägga ihop komponenter än att skriva sluten kod. Programmeringsmiljöerna har utvecklats i en riktning där programmerarna enkelt kan importera tredje parters komponenter i sina produkter.

Det finns massor av gratis komponenter med öppen källkod för alla vanliga plattformar och programspråk. Kryptografifunktioner, dataparsning och systemintegrationer hanteras i allmänhet bäst genom att importera komponenter. Det kan finnas kommersiell support även för gratiskomponenter.

Du bör ha en policy eller fastställd process för hur komponenter godkänns för användning. Det är ett beslut som inte kan hänskjutas till de enskilda utvecklarna. Varje komponent kan medföra nya sårbarheter. Du ska också vara medveten om licenserna i de tredjepartskomponenter som du använder.

Externa beroenden handlar ibland helt enkelt om kopierad och inklistrad kod. Programmerare behöver lösa invecklade problem och ofta går lösningen att hitta på internet som en kodsträng. ”Den fanns på webben” är ingen garanti för säkerhet och kan dessutom leda till överraskningar med licenser. Om du inte förstår någonting, ska du inte kopiera det.

GENOMFÖR KODGRANSKNING

Kodgranskningar är ett utmärkt sätt att kontrollera att riktlinjerna för programmeringen har följts. När du är noga med andra kvalitetsfaktorer – som kommentarer i källkoden, bra namngivningsrutiner och så vidare – kräver fixar mycket mindre arbete. Dessutom säkerställer bra kvalitetsrutiner kontinuiteten i arbetet genom att information förmedlas inom teamet.

TIPS:

I en ingående säkerhetsrevision ingår också kodgranskning. En källkod som inte är läslig, dokumenterad och väl versionskontrollerad är i sig en varningssignal. Revisorerna kommer inte att läsa och förstå hela källkoden, men utifrån sin erfarenhet, användningsscenarierna för produkten, hotmodellen och andra faktorer kommer de vilja hitta och granska sådana delar av koden som de bedömer som kritiska. Om de koddelarna är svåra att hitta på grund av att koden är slarvigt eller illa skriven kan det försvåra din väg mot att få godkännande för produkten.

Ofta räcker det inte med kodgranskning för att visa om programmet fungerar korrekt eller fungerar över huvud taget. Revisorn kan be dig underlätta körning och felsökning av koden. Eftersom många kryptografiska funktioner kan användas fel eller inte alls, är det möjligt att revisorn vill gå igenom kodens funktion genom att köra den steg för steg. Då behövs din hjälp, speciellt om du har använt mindre vanliga komponenter i hårdvaran och mjukvaran. Revisorn behöver också kunna förstå hur du har framställt och byggt koden för att kunna bedöma säkerheten i byggprocessen.

KONTINUERLIG INTEGRATION

Kontinuerlig integration avser regelbunden framställning av byggen av ett program. I automatiska processer genereras ett nytt bygge av produkten efter varje slutligt tillägg (commit) till källdatakatalogen. Du kan också lägga in automatiska test som ger utvecklarna omedelbar återkoppling om potentiella fel som de har gjort. Med kontinuerlig integration och automatiska test åtgärdas fel genast då de uppstår och då kan du känna dig säkrare på varje enskilt bygge.

Att skapa en miljö för kontinuerlig integration är en investering som det lönar sig att överväga. Även om du inte har helautomatiska processer för kontinuerlig integration uppmanar vi dig att gå in för korta intervall mellan byggen och införa så många automatiska test som möjligt.

Mer läsning:

- *Handböcker i säker kodning*
  - ***OWASP***
  - ***SEI CERT Coding Standards***

- *Referensmaterial för säkrare konstruktionsalternativ i vissa system*
  - ***C-Based Toolchain Hardening (Microsoft och GCC)***
  - ***Debian Hardening***
  - ***Microsoft - Security Best Practices for C++***

TESTNING OCH VERIFIERING

Med tester kontrollerar du att ett implementerat system uppfyller kraven. Med krav avses här både skriftligt fastställda, explicita krav och implicita krav. Ett program ska till exempel inte krascha, trots att det inte uttryckligen har nämnts i kravspecifikationen.

Du ska också tänka på att köra negativa test och pröva sådant som inte ska gå att göra.

Programmets säkerhetsegenskaper ska testas, liksom också andra viktiga systemfunktioner. Du ska också tänka på att köra negativa test och pröva sådant som inte ska gå att göra. Meningen är att skapa ett högkvalitativt system och då krävs det automatiska tester eftersom manuella test är helt enkelt alltför tunga för att det ska gå att erhålla tillräckligt god täckning för stora system med normala byggen. Vi ska först sammanfatta de olika tester som krävs.

Enhetstester är automatiserade tester som programmeraren själv kör på delar av koden i en komponent. Eftersom programmerarna förväntas köra testerna själv kan eventuella buggar åtgärdas snabbt och billigt. Kodgranskningar är ett utmärkt tillfälle för kontroll av att det finns enhetstester för största delen av koden.

Komponenttester innebär att komponenter körs isolerade, eventuellt med simulerade komponenter som står för resten av systemet. Komponenttesterna kan utformas av ett testteam. I idealfall är komponenttesterna automatiserade och körs dagligen eller under natten.

Systemtester omfattar ett bygge av hela systemet. Genom att systemtester kan kräva manuella teståtgärder kan de ta tid och vara dyra jämfört med enhetstester och komponenttester. Det är också möjligt att automatisera systemtest, men det kan kräva betydande investeringar i testinfrastruktur.

Acceptanstest utförs av ett oberoende testteam, kunden eller tredje part. Betydande problem som uppdagas vid acceptanstest kan kräva omfattande ändringar i programmet och ett nytt acceptanstest, vilket kan vara mycket dyrt och tidskrävande.

Statiska test genomförs utan körning av den egentliga produkten. Istället inspekteras olika artefakter, som källkod och binärfiler.

- Kodgranskningar och inspektioner kan ses som en form av statiska test.
- Automatiserad källkodsanalys är statisk testning.
- En relativt ny metod är analys av mjukvaruuppbyggnaden, som riktas på en färdig binärfil och analyserar vilka komponenter som ingår i den. Granskningen avslöjar de externa beroendena i produkten, vilka kan vara svåra att få fram på grund av det hela tiden ökande antalet utvecklare och moduler i en produkt.

Dynamiska test går ut på att köra produkten och studera hur det beter sig.

- Traditionellt innebär det att man kör manuella eller automatiska test för att verifiera att produkten uppfyller kraven.
- Fuzzning är en testmetod som fokuserar på säkerheten.
- Nuförtiden bör du också testa produkten mot säkerhetskraven och försök till intrång i och missbruk av produkten bör ingå i de dynamiska testerna.
- Belastningstest är dynamisk testning med fokus på programmets prestanda.

TIPS:

Det är möjligt att säkerhetsrevisorn gör följande tester:

1. Läser bruksanvisningen och/eller ber att få utbildning i att använda produkten.
2. Konfigurerar produkten för den tilltänkta användningen.
3. Använder produkten enligt det tilltänkta scenariot.
4. Klickar på tillgängliga menyer och dialoger i produkten.
5. Försöker administrera produkten enligt den tilltänkta användningen.
6. Skapar avvikande och stressade situationer för produkten (mer om dessa senare).

Kom dessutom ihåg att revisorerna dubbelkontrollerar sina observationer mot hotmodellen och uppdaterar den enligt behov. Syftet är ju att identifiera de väsentligaste hoten oavsett om de matchar de ursprungliga specifikationerna eller inte.

FUZZNING

Fuzzning, eller fuzz test, är en säkerhetsinriktad testmetod där produkten utsätts för oväntade och felaktiga indata i syfte att hitta buggar.

För fuzzning behövs inte tillgång till källkoden. Testerna kommer sannolikt att avslöja problem som du vill hitta innan någon annan gör det. Fuzzning kan också genomföras helt automatiserat.

Fuzzning avslöjar sårbarheter. Typiska tecken på en sårbarhet är programkrasch eller funktionsförlust, men problemet kan vara mer djupgående än så. Angripare kan använda särskilt utformade indata för att kapa systemet. Det kallas exploitation. Det finns såväl gratislösningar och köpprogram för fuzzning. (En mer detaljerad lista finns under ”Mer läsning”.) Använd någon av dem – angripare gör det säkert.

Fuzzning kommer sannolikt att avslöja problem som du vill hitta innan någon annan gör det.

TIPS:

Fuzzning ingår ofta i säkerhetsrevision eftersom det är enkelt att utföra utan att vara insatt i produktens inbyggda egenskaper och är ändå bra på att hitta brister. Eftersom många plattformar, om inte alla av de vanligaste, har genomgått omfattande fuzz tester, kommer säkerhetsrevisorn sannolikt inte att testa dem igen. Fuzz testerna riktas troligen på hemgjorda gränssnitt och ovanliga komponenter.

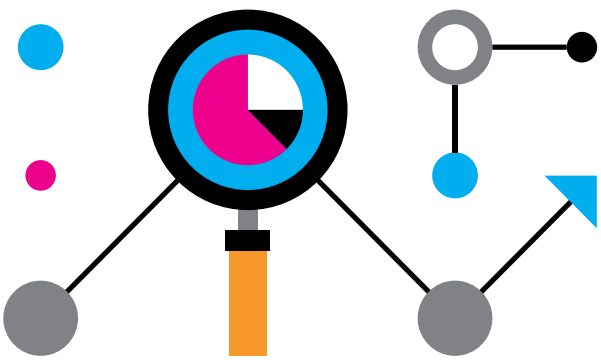
PENETRATIONSTEST

Penetrationstest är säkerhetstester där särskilda säkerhetsexperter försöker tränga in i en tjänst eller ett system för att uppdaga säkerhetsproblem. De används ofta och ger en opartisk syn på systemets säkerhetsnivå. Eftersom penetrationstest kräver manuellt arbete kommer de i allmänhet inte på fråga för varje version eller bygge av produkten. Penetrationstestens omfattning och djup kan variera och därför kan de snarare betraktas som acceptanstester.

STRESSTEST ELLER TORTYRTEST

Angripare kan försöka hitta sårbarheter genom att exponera produkten för avvikande belastning eller omständigheter. Eftersom det oftast inte går att förhindra detta bör ditt system vara utrustat för säkra fellägen och djupförsvar enligt ovan. Tänk bland annat på följande scenarier:

- Vad händer då enheten startas? Finns det okända attackbärare vid start (t.ex. vissa tangentkombinationer för tillgång till systemmenyn) eller accepterar enheten uppdateringar i fasta program från vem som helst under start? (Sådant har förekommit.)
- Vad händer då nätkontakten bryts? Kraschar systemet eller hamnar det i osäkert läge?
- Vad händer då strömmen bryts och enheten startar på nytt när strömmen återkommer?



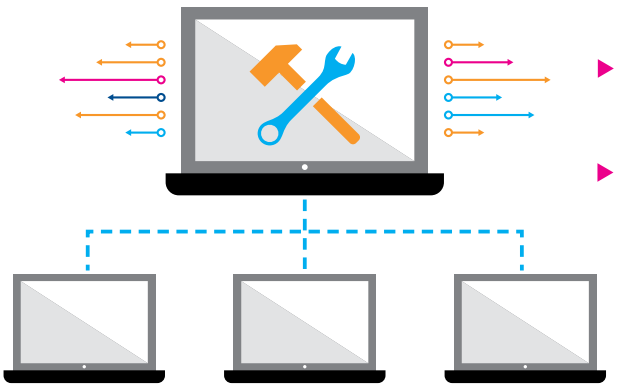
TIPS:

Att skapa stressituationer för produkten ingår i säkerhetsrevision. Sådana situationer är lätta att simulera (t.ex. dra ut kontakten) och kan vara mycket avslöjande.

REKONSTRUKTION

Som vi sagt ovan bör du inte tillämpa säkerhet genom otydlighet, det vill säga basera säkerheten i din produkt på att hemlighålla detaljer i programmet och algoritmerna. Det beror i hög grad på att det finns gott om verktyg för rekonstruktion av exekverbara filer, fasta program och nätverkstrafik. Det är orealistiskt att anta att detaljerna om systemets uppbyggnad hålls hemliga.

Överväg att rekonstruera dina egna produkter, bara för att få en känsla för vad det handlar om. Vi har hittat följande användbara verktyg:



- **Wireshark** övervakar och analyserar trafik i datornät.
- **Nmap** skannar datornät på värdar, öppna anslutningar och tjänster.
- **Strings** matar ut strängar från vilken som helst fil (t.ex. exekverbara filer, fasta program).

TIPS:

Säkerhetsrevisorn använder rekonstruktion för att dubbelkontrollera att dina påståenden om produkten stämmer. Vilka komponenter används de facto i produkten? Stämmer resultaten av nätskanningen överens med listan över de nödvändiga nättjänsterna? Stämmer trafiken i nätverket överens med hur du beskriver den? Stämmer de fysiska säkerhetskomponenterna överens med dina påståenden?

TESTSAMMANDRAG

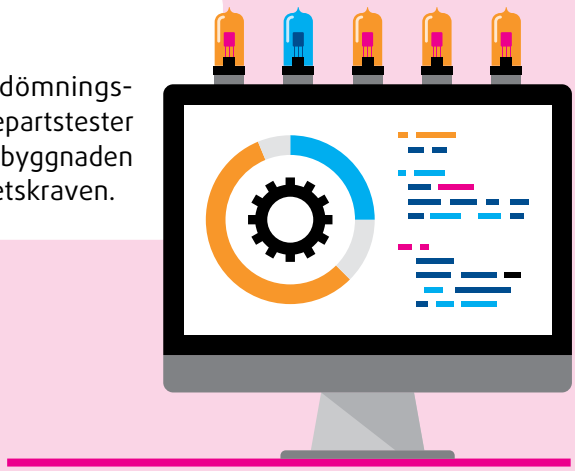
Vi vill inte bli underkända i acceptanstesten. Det bästa sättet att undvika det är att vara insatt i vad testerna innehåller och sedan köra ännu striktare tester själv före ansökan om godkännande. Du kan gärna överväga att anlita en tredje part för ytterligare tester innan du lämnar över systemet för acceptanstest.

Alla har råkat ut för att ett system fungerar perfekt på den egna datorn eller i labbet, men inte alls fungerar i andra miljöer.

Du ska absolut undvika en situation där acceptanstest är det första tillfället där ditt system körs utanför företaget. Att anlita tredje part som testare är ett sätt att undvika detta. Lämpliga tredjepartstestare är till exempel motiverade betakunder som kör produkten i verkliga användningssituationer och ger dig feedback.

TIPS:

Noggrann testdokumentation gör bedömningsprocessen snabbare. Resultat från tredjepartstester hjälper också. Kom ihåg att relatera uppbyggnaden och resultaten av testerna mot säkerhetskraven.



Mer läsning:

- ***OWASP: Web Application Security Testing Cheat Sheet***
- ***Wikipedia: Penetration Testing***
- ***Wikipedia: Stress testing***
- ***OWASP: Fuzzing (inkl. referenser till verktyg)***
- ***MICKAEL DORIGNY, Uppdaterad 19/10/2016: Qu'est ce que le durcissement?***

IMPLEMENTERING

När du har en färdig produkt och någon har köpt den gäller det att driftsätta den hos kunden. Förr innebar det att man gav kunden en installationsdisk eller skickade någon till kunden för att installera systemet. Nuförtiden sker installationen över internet eller åtminstone sköts uppdateringarna över webben. Molntjänster installeras inte över huvud taget, utan de används med webbläsare.

Nuförtiden kan applikationer också levereras paketerade i programbehållare, som till exempel Docker. Behållarna kan innehålla flera olika komponenter vars säkerhet och försörjningskedja du måste känna till. Vad gäller säkerhetsintensiva produkter som innehåller hårdvarukomponenter, bör du dessutom tänka på hur hårdvaran eventuellt kan utsättas för manipulation.

Du bör ge dina kunder anvisningar om säker livscykelhantering för produkterna.

När du tillhandahåller en nedladdningsportal för dina kunder ska du säkerställa att ingen kommer åt att infektera installationsavbildningarna i portalen. Du bör också säkerställa att dina kunder inte luras av falska nedladdningsportaler när de ska ladda ner din produkt. De förutsätter att du säkrar portalservern på ett ändamålsenligt sätt och alltid använder krypterade webbsidor med uppdaterade certifikat. Om din produkt självständigt laddar ner uppdateringar eller tillägg, ska den alltid kontrollera att den verkligen laddar ner från din server. Även vid signerade uppdateringar ska du vara medveten om nedgraderingsattacker där angriparen lurar kunderna att uppdatera till en äldre och sårbar version av din produkt.

Implementeringen kan ske på tillverkningsplatsen eller genom datatransit från tillverkaren till dina lokaler eller från dina lokaler till kunden. Med avseende på slutet av produktens livscykel bör du tänka på vad som händer när hårdvaran tas ur bruk. Exempelvis diskarna inuti produkten kan innehålla känsliga data som måste destrueras. Du bör ge dina kunder anvisningar om säker livscykelhantering för produkterna.

Lämpliga implementeringsformat som du kan använda är till exempel en enhet med ett operativsystem (t.ex. Linux) eller en virtuell avbildning, som implementeras hos kunden. Oavsett vilken plattform du använder ska du försäkra dig om att den är tillräckligt säker. Detta innebär i regel att du ska koppla bort och eventuellt ta bort alla tjänster som inte är nödvändiga för produkten. Plattformar har ofta egna säkerhetsegenskaper och du bör överväga om de ska aktiveras eller inte.

SYSTEMFÖRVALTNING OCH PROGRAMFIX

När din produkt har godkänts, köpts och installerats behöver den underhåll. Förr eller senare uppdagas det sannolikt en sårbarhet, troligen i en tredje parts komponent som du har integrerat i produkten eller i en plattform som du använder. Då måste du reagera snabbt och målmedvetet.

Om du har fått godkännande för en viss version av produkten, är det möjligt att en uppdatering leder till att godkännandet inte längre gäller. Å andra sidan, om du inte uppdaterar produkten är dina kunder sårbara.

Du bör vara förberedd på detta och ta upp det med godkännaren. Tänk på problematiken ur följande perspektiv:

- Det är lättare att godkänna små och noga inriktade ändringar än övergripande ändringar vars slutliga omfattning är oklar.
- Ändringar i vissa systemkomponenter kan verka mindre oroväckande för godkännaren jämfört med andra. Det kan till exempel gå bra att uppdatera komponenter i användargränssnittet utan nytt godkännande, medan modifikationer i kryptomodulen kräver nytt godkännande.
- Om godkännaren vet att du har gedigna processer för utveckling, verifiering och leveranser, är det möjligt att de inte oroar sig över att uppdateringar ger upphov till oönskade biverkningar i produkten.

Du vill verkligen inte råka ut för att en kund frågar dig om en nyligen uppdagad sårbarhet påverkar ditt system och du inte kan svara på frågan. Du ska förstå uppbyggnaden av din produkt och försörjningskedjan bakom den.

Ännu bättre är det om du proaktivt kan informera dina kunder om sårbarheter som påverkar din produkt. Det förutsätter att du har en process för att följa med när sårbarheter hittas i relevanta komponenter och plattformar.

Buggen Heartbleed (CVE-2014-0160) upptäcktes 2014 i krypteringsbiblioteket i Open SSL. Open SSL är ett mycket populärt säkerhetsprogram som används direkt och indirekt i otaliga applikationer, enheter, komponenter och plattformar. Eftersom Heartbleed var ett allvarligt fel som väckte stor uppmärksamhet, var det mycket angeläget att fixa buggen överallt där Open SSL användes. Det ledde till att kunder frågade leverantörer om dessa hade använt Open SSL och om hur sårbara de var. Många leverantörer visste inte vilken version av programmet de använde och om de var sårbara. En del leverantörer visste inte över huvud taget om de använde Open SSL.

Det är också möjligt att någon annan hittar en sårbarhet och vill informera dig om det. Vi utgår i från att denna någon är din kund och att du är villig att lyssna och ta emot informationen. Det är också möjligt att denna person är någon annan, till exempel en säkerhetsforskare. För sådana fall är det bra om du har en lämplig rapporteringskanal, till exempel en särskild epostadress eller ett webbformulär. Om forskaren inte kan nå dig med sin feedback är det möjligt att hen publicerar sårbarheten eller säljer informationen till en opålitlig aktör.

På webbplatsen **www.tietoturvailmoitus.fi** (på finska) finns information om hur du ska gå till väga för att ta emot information om säkerhetsbrister.

Hittelön för buggar har visat sig vara ett effektivt sätt att höja säkerheten i produkter. Det förutsätter visserligen att du är insatt i saken och din produkt är tillräckligt mogen innan du övergår till att helt förlita dig på utomstående som säkerhetstestare. Du ska åtminstone inte åtala personer som pekar ut brister. Istället bör du tacka dem – eller till och med anställa dem!

Följande termer kommer sannolikt att dyka upp när du läser på säkerhetsuppdateringar:

- CVE (Common Vulnerabilities and Exposures) är ett system för samordnad namngivning av sårbarheter och exponeringar, så att alla vet vad som avses med till exempel **CVE-2014-0160**.
- CVSS (Common Vulnerability Scoring System) är ett system med mätetal för beräkning av ett riskindex för sårbarheterna eller exponeringarna.
- CWE (Common Weakness Enumeration) är en databas över sårbarheter. Den är mindre känd än de två förstnämnda.

SLUTLEDNINGAR

Denna handbok redogör för de olika faserna i systemutvecklingslivscykeln. Som ett självtest kan vi jämföra resonemangen mot Katakri-kriterierna på programleverantörer. Enligt Katakri ska

1. programleverantörernas kompetens ifråga om informationssäkerhet verifieras

2. en riskanalys ha genomförts under utvecklingsfasen och eventuella risker ha åtgärdats (antingen kontrollerats eller uttryckligen godkänts)

3. gränssnitten (åtminstone utåt) testas med falska indata och stora indatavolymer

4. leverantören, beroende på utvecklingsmiljön, ha en policy för användningen av problematiska funktioner och gränssnitt och denna policy ska övervakas (t.ex. Microsoft har en lista över förbjudna funktioner)

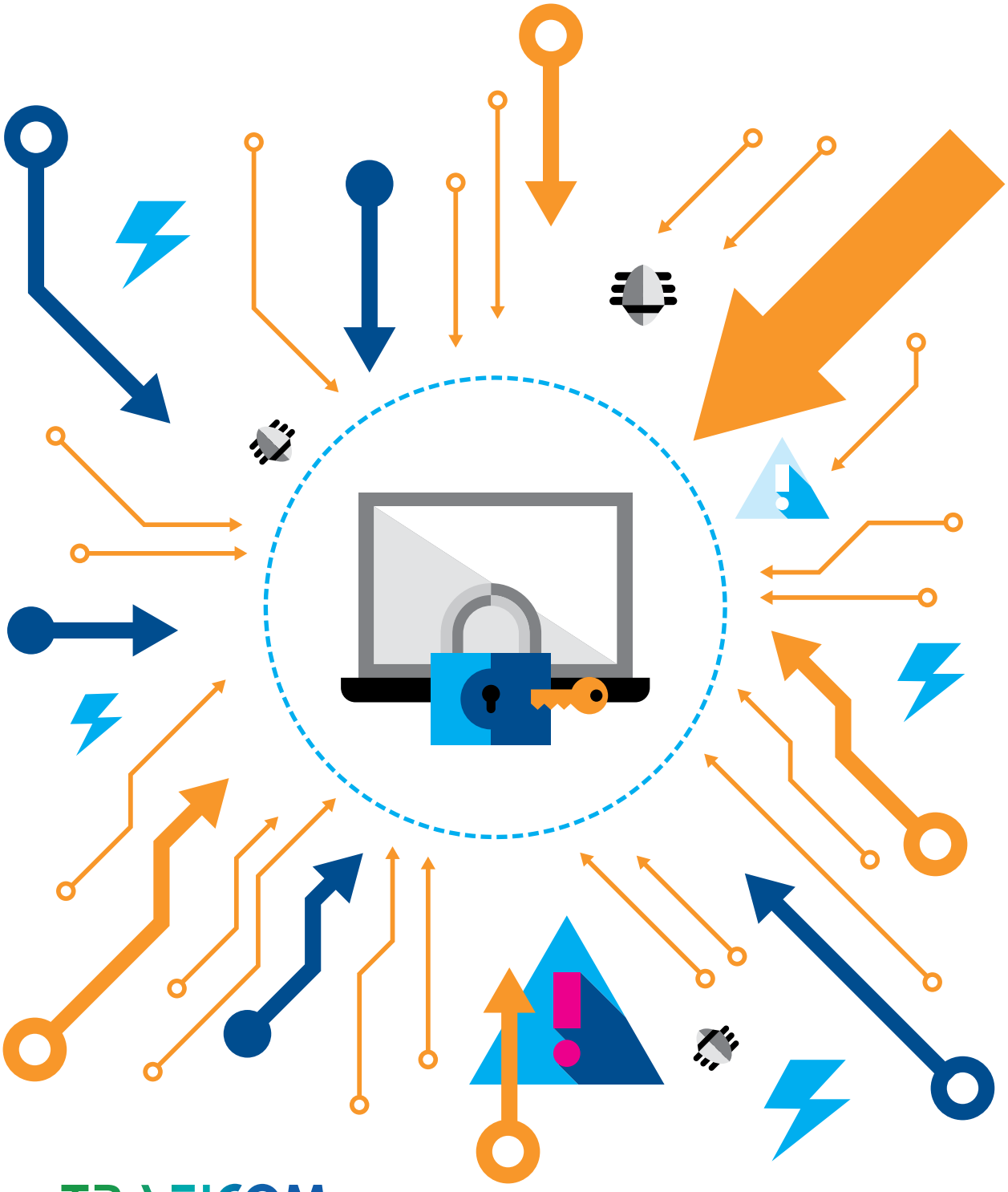
5. arkitekturen och källkoden genomgå säkerhetsrevision

6. källkoden ha granskats med hjälp av statisk analys

7. integriteten i produktens källkod, versionshantering och utvecklingsverktyg ha säkerställts.

Mer läsning:

- *NCSA-dokument (på finska)*
- *Katakri 2015 – Verktyg för informationssäkerhetsauditering för myndigheter (Engelsk översättning)*
- *VAHTI 1/2013 Sovelluskehityksen tietoturvaohje*
- *OWASP är en öppen sammanslutning som arbetar för att organisationer ska kunna ta fram, utveckla, förvärva, driva och förvalta tillförlitliga mjukvaruapplikationer. De har tagit fram mycket material och en del av det kan verka förvirrande. Här är några axplock:*
  – OWASP / Författare: Dharmesh M Mehta: Effective Software Security Management
  – Development OWASP Guide 3.0
- *Microsoft SDLC var en av de första publicerade systemutvecklingslivscyklarna.*
- *NIST - Cryptographic Standards and Guidelines*



TRAFICOM
Transport- och kommunikationsverket
Cybersäkerhetscentret

www.ncsc.fi | www.kyberturvallisuuskeskus.fi/sv | www.traficom.fi/sv