A27495 - Åpen

Rapport

Modenhetsmodell for innebygd sikkerhet (BSIMM)

Måling av programvaresikkerhetsaktiviteter i utviklingsorganisasjoner

Forfatter(e)
Martin Gilje Jaatun
Basert på engelsk originaltekst av Gary McGraw, Jacob West og Sammy Migues

SINTEF Digital
Information Security
2018-10-19

SINTEF Digital
Postadresse:
Postboks 4760 Torgarden
7465 Trondheim
Sentralbord:
Telefaks: Telefaks: 73594302

Foretaksregister:
NO 948 007 029 MVA

Rapport

Modenhetsmodell for innebygd sikkerhet (BSIMM)

Måling av programvaresikkerhetsaktiviteter i utviklingsorganisasjoner

EMNEORD:
Måling, programvaresikkerhet, BSIMM

VERSJON
1.3

DATO
2018-10-19

FORFATTER(E)
Martin Gilje Jaatun
Basert på engelsk originaltekst av Gary McGraw, Jacob West og Sammy Migues

OPPDRAGSGIVER(E)

OPPDRAGSGIVERS REF.

PROSJEKTNR
102009398

ANTALL SIDER OG VEDLEGG:
32

SAMMENDRAG
Dette dokumentet inneholder norske oversettelser av aktivitetene beskrevet i Building Security In Maturity Model (BSIMM), lett omarbeidet fra bloggen http://infosec.sintef.no. Teksten er basert på BSIMM-V, men aktivitetsnummereringen er oppdatert for å stemme med BSIMM9, samt at en ny aktivitet er lagt til i versjon 1.1, tre nye er lagt til i versjon 1.3.

BSIMM er et rammeverk med 116 programvaresikkerhetsaktiviteter fordelt på 4 domener som hver er delt i 3 praksiser. Utover aktivitetsbeskrivelsene som er gjengitt på norsk i denne rapporten, inneholder det offisielle BSIMM-dokumentet også resultater fra evaluering av 120 (hovedsakelig amerikanske) virksomheter, og er derfor et godt grunnlag for å vurdere hva som er "god praksis" innen programvaresikkerhet.

UTARBEIDET AV
Martin Gilje Jaatun
SIGNATUR

KONTROLLERT AV
Inger Anne Tøndel
SIGNATUR

GODKJENT AV
Eldfrid Øvstedal
SIGNATUR

RAPPORTNR	ISBN	GRADERING	GRADERING DENNE SIDE
A27495	978-82-14-05923-6	Åpen	Åpen

Historikk

VERSJON	DATO	VERSJONSBESKRIVELSE
0.9	2015-11-06	Tekst kopiert fra http://infosec.sintef.no
0.91	2015-12-16	Oppdatert etter QA
1.0	2016-02-01	Første versjon
1.1	2016-10-13	Oppdatert til BSIMM 7
1.2	2017-10-02	Oppdatert til BSIMM 8
1.3	2018-10-19	Oppdatert til BSIMM 9

SINTEF

Innholdsfortegnelse

1 Innledning

Programvaresikkerhet er viktig for all programvare, ikke bare programvare som har spesielle sikkerhets-funksjoner. Programvaresikkerhet er en iboende egenskap (på samme måte som "kvalitet"[2]), og er ikke det samme som sikkerhetsmekanismer. Imidlertid er programvaresikkerhet noe som er svært vanskelig å måle [3]; det er bortimot umulig å ta to forskjellige programmer og avgjøre hvilket av dem som er "mer sikkert". En tilnærming til dette problemet har vært å i stedet måle andre-ordens egenskaper, dvs. måle hvilke *programvaresikkerhetsaktiviteter* utviklerne utfører som en del av utviklingsprosessen.

Building Security In Maturity Model (BSIMM) tar utgangspunkt i et programvaresikkerhetsrammeverk (*Software Security Framework*) som består av 4 domener som hver er delt inn i 3 praksiser (se Tabell 1). Hver av praksisene inneholder en rekke mål og aktiviteter som er benyttet blant programvareutviklende bedrifter, blant annet Adobe, F-Secure, Box, Visa, Cisco og General Electric. Alt etter hvilke aktiviteter som benyttes innenfor en bedrift, er det mulig å si noe om modenhetsnivået på programvaresikkerhetsarbeidet. Det er hittil gitt ut 9 rapporter med resultater av målinger fra slike virksomheter (i den siste utgaven, BSIMM9 [1], er det med resultater fra 120 forskjellige virksomheter).

Teksten i aktivitetsbeskrivelsene er formulert som om man i utgangspunktet ønsker å oppfylle alle aktivitets-kravene, men det er viktig å huske på at BSIMM primært er et måleverktøy, og alle aktivitetene er ikke nødvendigvis relevante for alle virksomheter. Når det står i en aktivitetsbeskrivelse f.eks. at "Virksomheten må ...", skal dette leses som: "*For at vi skal telle at virksomheten utfører denne aktiviteten,* må virksomheten ...". BSIMM aktivitetene er rangert i tre nivåer, der nivå 1 regnes som det laveste modenhetsnivået. For virksomheter med et lavt modenhetsnivå med hensyn til programvaresikkerhet kan det være formålstjenlig å bruke BSIMM som en sjekkliste for å *vurdere* om aktivitetene er relevante for dem – og da kan det være lurt å starte med aktivitetene på nivå 1.

Tabell 1: BSIMM programvaresikkerhetsrammeverk

Programvaresikkerhetsrammeverk (*Software Security Framework*)			
Ledelse og styring (*Governance*)	**Etterretning (*Intelligence*)**	**SSDL Trykkpunkter (*SSDL Touchpoints*)**	**Utrulling (*Deployment*)**
Strategi og måling (*Strategy and Metrics*)	Angrepsmodeller (*Attack Models*)	Arkitekturanalyse (*Architecture Analysis*)	Penetreringstesting (*Penetration Testing*)
Etterlevelse av lover, regler og retningslinjer (*Compliance and Policy*)	Sikkerhetsfunksjonalitet og design (*Security Features and Design*)	Kodegjennomgang (*Code Review*)	Programvaremiljø (*Software Environment*)
Opplæring og øvelser (*Training*)	Standarder og krav (*Standards and Requirements*)	Sikkerhetstesting (*Security Testing*)	Konfigurasjonsstyring og sårbarhetsstyring (*Configuration Management and Vulnerability Management*)

Selv om BSIMM hevdes å ikke være preskriptiv, gis det noen generelle råd for hvordan programvare-sikkerhetsarbeidet kan organiseres, og før vi går videre med beskrivelsen av de enkelte aktivitetene skal vi i det følgende se litt nærmere på to anbefalinger: Etablering av en programvaresikkerhetsgruppe (*software security group* – SSG) og etablering av en "satellitt".

1.1 Programvaresikkerhetsgruppen (SSG)

I enhver organisasjon som utvikler programvare må det finnes noen som har ansvaret for programvaresikkerheten. I BSIMM-studien [1] fant de at ALLE virksomhetene som ble undersøkt hadde en programvaresikkerhetsgruppe (*software security group* – SSG [4]) – den kan være så liten som en person, men den må være der.

Hvis noe skal bli gjort, må noen ha ansvaret. Hvis alle har ansvaret, betyr det i praksis at ingen har ansvaret, og sannsynligheten for at noe blir gjort er minimal. Man trenger derfor en SSG, og hvis den utgjør mer enn en person, trenger den også en leder. Hvis man tildeler noen et ansvar, er det viktig å følge opp med tilhørende myndighet, og det kan derfor være fornuftig å plassere lederen av SSG tilstrekkelig høyt oppe i virksomhetens ledelseshierarki – dette vil også kunne sikre forankring hos toppledelsen, noe som er en forutsetning for at SSG skal lykkes med sin oppgave.

Tall fra BSIMM-studien indikerer at en fornuftig størrelse på SSG er rundt 1% av utviklerteamet, noe som tilsier 1 fulltids SSG-ansatt per 100 utviklere. BSIMMs "far", Gary McGraw, mener at det er enklere å lære en utvikler sikkerhet enn å lære en sikkerhetsperson om utvikling, men det er uansett viktig at SSG snakker "utviklingsavdelingens språk", og ikke oppfattes om en utenforstående klamp om foten. SSG skal ha det overordnede ansvaret for programvaresikkerheten, og det er derfor viktig at den har verktøyene og myndigheten til å gjøre jobben. SSG må yte bistand til utviklingsavdelingen i forbindelse med testing, kodegjennomgang og analyse, men må også bidra med veiledning og opplæring av utviklerne. Mange av oppgavene til SSG beskrives detaljert i de forskjellige BSIMM-aktivitetene.

1.2 Satellitten

I store virksomheter vil det være vanskelig å skalere SSG til å dekke behovene til alle utviklere, og derfor etablerer mange en mer eller mindre formalisert "utskutt" gruppe utviklere med spesiell interesse for sikkerhet; dette kalles i BSIMM for "Satellitten". Medlemmene i satellitten deltar aktivt i utviklingsarbeidet, og har ofte rollen som ansvarlig for sikkerhetsaktiviteter i utviklingsteamet.

Spesielt i virksomheter som driver med smidig utvikling kan det være avgjørende å ha en velfungerende satellitt som ivaretar den utøvende funksjonen i programvaresikkerhetsarbeidet. Slik kan SSG fylle rollen som støttefunksjon, og unngå å oppfattes som en kontrollinstans som stikker kjepper i hjulene for utviklerne. Satellitten bidrar til å unngå en "oss mot dem"-mentalitet, slik at sikkerhetseksperter bidrar til å løse problemer snarere enn å skape dem.

1.3 Videre organisering av dokumentet

I de neste avsnittene beskrives programvaresikkerhetsaktivitetene fortløpende slik de er organisert i Tabell 1. Avsnitt 2 presenterer domenet "Ledelse og styring" (*Governance*), avsnitt 3 tar for seg domenet "Etterretning" (*Intelligence*), avsnitt 4 dekker domenet "SSDL Trykkpunkter" (*SSDL Touchpoints*), mens avsnitt 5 omhandler domenet "Utrulling" (*Deployment*). Dokumentet avsluttes med en referanseliste i avsnitt 6.

2 Ledelse og styring (*Governance*)

Domenet **Ledelse og Styring** omfatter Strategi og måling; Etterlevelse av lover, regler og retningslinjer; og Opplæring og øvelser.

2.1 Strategi og måling (*Strategy and Metrics*)

Hovedmålet for praksisen Strategi og måling er transparens av forventninger og ansvarlighet for resultater. Toppledelsen må klarlegge organisasjonsmessige forventninger for SSDL (Secure Software Development Lifecycle) slik at alle forstår viktigheten av initiativet. I tillegg må toppledelsen angi spesifikke mål for alle SSDL interessenter, og sørge for at spesifikke individer kan stilles til ansvar for at disse målene nås.

2.1.1 SM Nivå 1: Få en felles forståelse av retning og strategi

Ledere[1] må sørge for at alle som er involvert i å lage, rulle ut, drifte og vedlikeholde programvare forstår de nedskrevne organisasjonsmessige programvaresikkerhetsmålene. Ledere må også forsikre seg om at organisasjonen som helhet forstår strategien for å oppnå disse målene. En felles strategisk forståelse er avgjørende for virkningsfull og effektiv gjennomføring av programmet.

- **SM 1.1: Publiser prosess (roller, ansvar, plan), videreutvikle ved behov**
  Prosessen for å adressere programvaresikkerhet kringkastes til alle deltakere (utviklere, testere, ledere, etc.) slik at alle kjenner til planen. Mål, roller, ansvar og aktiviteter er eksplisitt definerte. De fleste virksomheter velger og vraker i en publisert metodologi som Microsoft SDL eller Cigital Touchpoints, og gjør deretter tilpasninger etter behov. En SSDL prosess utvikler seg mens virksomheten modnes og sikkerhetslandskapet endrer seg. I mange tilfeller er prosessen kun publisert internt, og er kontrollert av SSG. SSDL trenger ikke være offentliggjort utenfor firmaet for at denne aktiviteten skal vurderes som godkjent.
- **SM1.2: Opprett en evangelistrolle og foreta intern markedsføring**
  For å kunne bygge opp støtte for programvaresikkerhet gjennom hele virksomheten, må en i SSG spille rollen som evangelist. Denne interne markedsføringsfunksjonen hjelper til med å holde virksomheten oppdatert på størrelsen av programvaresikkerhetsproblemet og elementene som kan bidra til dets løsning. Evangelisten kan holde foredrag for interne grupper, invitere inn eksterne foredragsholdere, skrive notater for internt bruk, eller lage en samling av artikler, bøker og andre ressurser på en intern nettside, og oppfordre til at den blir brukt. Ad hoc samtaler mellom SSG og toppledelsen, eller en SSG hvor "alle er en evangelist" oppnår ikke de ønskede resultatene. Et klassisk eksempel på en slik evangelist er Michael Howards rolle hos Microsoft.
- **SM1.3: Opplæring av toppledelsen**
  Toppledelsen lærer om konsekvensene av utilstrekkelig programvaresikkerhet og de negative innvirkningene dårlig sikkerhet kan ha på forretningsdriften. De lærer også om hva andre virksomheter gjør for å oppnå programvaresikkerhet. Ved å forstå både problemet og hvordan det skal løses på en skikkelig måte, støtter ledelsen programvaresikkerhets-initiativet som en nødvendig del av risikostyring. I sin mest farlige form kommer slik opplæring som følge av angrep fra ondsinnede hackere eller at sensitive data kommer på avveie i all offentlighet. Det er å foretrekke at SSG kan demonstrere det verst tenkelige scenarioet i et kontrollert miljø med tillatelse fra alle involverte (f.eks. faktisk vise fungerende angrep og deres konsekvenser for virksomheten). I enkelte tilfeller kan presentasjon av slike angrep til styret hjelpe til med å skaffe ressurser til et pågående programvaresikkerhets-initiativ. Det kan ofte være nyttig å hente inn en ekstern guru når man prøver å få oppmerksomheten til toppledelsen.
- **SM1.4: Identifiser beslutningspunkter, samle nødvendige artefakter**
  Programvaresikkerhetsprosessen vil involvere beslutningspunkter/ sjekkpunkter/ milepæler på et eller flere steder i programvareutviklingslivssyklusen (*software development life cycle* – SDLC[2]). De to første stegene for å etablere beslutningspunkter er 1) identifiser beslutningspunkter som er kompatible med eksisterende utviklingspraksis, og 2) begynn å samle informasjon som er nødvendig for å ta en avgjørelse på å gå videre eller ei. Det er viktig å merke seg at på dette stadiet er ikke beslutningspunktene obligatoriske. F.eks. kan SSG samle inn resultater fra sikkerhetstesting for hvert prosjekt før det slippes, men uten å foreta noen vurdering av hva som utgjør tilstrekkelig testing eller akseptable testresultater. Poenget med å bare identifisere beslutningspunkter først, og gjøre dem obligatoriske senere, er at det bidrar til å dytte utvikling i retning av programvaresikkerhet uten store smerter. Sosialiser beslutningspunktene (dvs. innarbeid dem i organisasjonen), og aktiver dem kun

[1] Her og i det følgende mener vi primært "mellomledere" når vi sier "ledere" (fra *managers*), hvis vi snakker om topp-ledelsen (*executives*) angir vi det spesielt.
[2] eller SDLCene – det kan være flere i samme virksomhet. Dette kan fort bli litt komplisert, men i de fleste tilfellene har man én SSDL som dekker alle SDLCene innen en virksomhet [6].

når de fleste prosjekter allerede vet hvordan de skal lykkes. En slik gradvis tilnærming bidrar til å motivere god oppførsel uten å kreve det.

2.1.2 SM Nivå 2: Tilpass oppførsel med strategi, og sjekk etterlevelse

Toppledelsen må eksplisitt identifisere individer som skal stå ansvarlige for håndtering av risiko knyttet til programvaresikkerhet. Disse individene er igjen ansvarlige for vellykket gjennomføring av SSDL-aktiviteter. SSDL-ledere må sørge for rask identifikasjon og modifikasjon av enhver SSDL-oppførsel som resulterer i uakseptabel risiko. For å redusere uakseptabel risiko, må ledere identifisere og oppmuntre vekst av en programvaresikkerhets-satellitt.

- **SM2.1: Offentliggjør data om programvaresikkerhet internt**
  SSG offentliggjør data om tilstanden til programvaresikkerheten innad i virksomheten. Informasjonen kan komme i form av et dashboard med metrikker for toppledelsen og utviklingslederne. Noen ganger deles ikke informasjonen med alle i virksomheten, men kun med de relevante lederne. Et alternativ kan være at informasjonen dyttes opp til toppledelsen som deretter gjør noe med det og fungerer som en pådriver for endring i virksomheten. I andre tilfeller kan full åpenhet og publisering av data til alle interessenter sørge for at alle er klar over hva som skjer, etter filosofien at sollys er det beste desinfeksjonsmiddelet. Dersom bedriftskulturen oppfordrer til intern konkurranse mellom grupper, tilfører denne informasjonen en sikkerhetsdimensjon til spillet.
- **SM2.2: Følg opp beslutningspunkter og spor unntak**
  SSDL sikkerhetsbeslutningspunkter er nå obligatoriske; for å kunne passere et beslutningspunkt må prosjektet enten tilfredsstille det etablerte tiltaket eller få en dispensasjon. Til og med de mest motvillige utviklergruppene må nå følge de samme reglene. SSG sporer unntak. Et beslutningspunkt kan f.eks. kreve at et prosjekt foretar en kodegjennomgang og retter opp alle kritiske funn før utgivelse. I noen tilfeller er beslutningspunkter direkte assosiert med kontrollmekanismer som kreves av regelverk, kontraktsmessige avtaler, og andre typer forretningsmessig ansvar, og unntak spores i henhold til krav i lover og regler. I andre tilfeller gir tiltak i beslutningspunktene nøkkelindikatorer (KPI) som brukes til å styre prosessen.
- **SM2.3: Opprett eller utvid satellitten**
  Satellitten begynner som en samling mennesker spredt utover i virksomheten som er over gjennomsnittet interessert i sikkerhet og/eller har slike ferdigheter. Det å identifisere denne gruppen er et steg i retning av å lage et sosialt nettverk som framskynder integrering av sikkerhet i programvareutviklingen. En måte å gjøre dette på er å notere seg mennesker som utmerker seg under kurs og opplæring. En annen måte er å be om frivillige. I en mer ovenfra og ned tilnærming blir medlemmer av satellitten først oppnevnt for å sørge for at alle utviklings- og produktgrupper er dekket. Videre medlemskap bør baseres på faktisk ytelse. En sterk satellitt er et godt tegn på et modent programvaresikkerhets-initiativ.
- **SM2.6: Krev kvittering på sikkerhet**
  Virksomheten har en felles prosess for å akseptere sikkerhetsrisiko og dokumentere ansvar. Den som har ansvar for å akseptere risiko må kvittere på tilstanden til all programvare før den slippes videre. Kvitteringsretningslinjene kan f.eks. kreve at lederen for forretningsenheten må kvittere på kritiske sårbarheter som ikke er håndtert eller SSDL steg som har blitt hoppet over. Uformell risikoaksept alene teller ikke som sikkerhetskvittering, ettersom det å akseptere risiko oppleves mer konkret (og følgelig er mer effektivt i sikkerhetsarbeidet[3]) når det er formalisert (f.eks. ved en signatur, utfylling og innsending av et skjema, eller lignende) og tatt vare på for senere referanse.

[3] Det å tvinges til å formelt akseptere risiko kan ha en oppdragende effekt på ledelsen, og vil tvinge dem til å sette seg bedre inn i problemstillingen. I noen tilfeller vil da ledelsen oppdage at risikoen *ikke* kan aksepteres slik situasjonen er, og ytterligere tiltak må iverksettes.

2.1.3 SM Nivå 3: Praktisér risikobasert porteføljeadministrasjon

Applikasjonseier og programvaresikkerhetsgruppen (SSG) må orientere ledelsen om risikoen assosiert med hver applikasjon i porteføljen. SSG må reklamere for sine aktiviteter eksternt for å bygge støtte for sin tilnærming, og muliggjøre sikkerhet i økosystemet som applikasjonene i porteføljen utgjør.

- **SM3.1: Bruk en intern sporingsapplikasjon med porteføljeoversikt**
  SSG bruker en sentralisert sporingsapplikasjon for å følge framdriften til hver bit av programvare som den har overoppsyn med. Applikasjonen registrerer sikkerhetsaktiviteter som er planlagt, underveis, og ferdige. Den tar opp i seg resultater fra aktiviteter som arkitekturanalyse, kodegjennomgang og sikkerhetstesting. SSG bruker sporingsapplikasjonen til å generere porteføljerapporter for mange av metrikkene de bruker. En kombinert "inventarliste" og risikoholdningsoversikt er fundamental. I mange tilfeller publiseres disse dataene (i det minste) til toppledelsen. Avhengig av kulturen kan dette føre til interessante resultater gjennom intern konkurranse. Etter hvert som initiativet modnes, og aktiviteter blir mer distribuerte, bruker SSG det sentraliserte rapporteringssystemet til å holde styr på alle bevegelige deler (dvs. alle programvarekomponenter som er under endring).
- **SM3.2: Driv et eksternt markedsføringsprogram**
  SSG markedsfører programvaresikkerhetsinitiativet utenfor virksomheten for å bygge ekstern støtte. Programvaresikkerhet vokser utover å være et risikoreduserende tiltak, og blir et forretningsfortrinn eller en måte å differensiere seg i markedet. SSG kan f.eks. forfatte artikler eller bøker, ha en offentlig blogg, delta på eksterne konferanser eller messer. I enkelte tilfeller kan en komplett SSDL-metodikk bli publisert og promotert eksternt. Ved å dele detaljer med omverdenen og invitere kommentarer kan man hente inn nye perspektiver til virksomheten.
- **SM3.3: Identifiser metrikker, og bruk dem til å påvirke budsjetter**
  SSG og ledelsen velger metrikker som definerer og måler framdriften til programvaresikkerhets-initiativet. Disse metrikkene vil være førende for initiativets budsjett og tildeling av ressurser, så enkle opptellinger og statistikk er ikke godt nok. Metrikker lar også SSG forklare sine mål og framdrift i kvantitative termer. En slik metrikk kunne være sikkerhetsfeil-tetthet. En reduksjon i sikkerhetsfeil-tetthet kunne brukes til å vise en redusert kostnad for utbedring over tid. Nøkkelen her er å binde tekniske resultater til forretningsmål på en klar og åpenbar måte for å underbygge og forsvare finansiering. Siden konseptet "sikkerhet" allerede framstår som ganske vagt for mange forretningsfolk, kan det være ganske nyttig å gjøre en slik eksplisitt sammenknytning.

2.2 Etterlevelse av lover, regler og retningslinjer (*Compliance and Policy*)

"Security Policy" er et av mange ord som ikke har en fullgod norsk oversettelse, noe som medfører at mange bruker bastard-begrepet "sikkerhets-policy". Muligens kan man oversette det med "sikkerhetsinstrukser og strategi", men jeg er usikker på om det treffer helt (noen foreslo nettopp "sikkerhetsretningslinjer" – det er kanskje bedre).

2.2.1 CP Nivå 1: Dokumentér og sy sammen juridiske og kontraktsmessige forutsetninger

SSG må samarbeide med relevante grupper for å identifisere hvilke krav fra lover og regler det er nødvendig å oppfylle, sy disse sammen til en helhet, og sørge for at denne kunnskapen blir tilgjengelig for SSDL interessenter (*stakeholders*).

- **CP1.1: Samle regulatoriske rammebetingelser**
  SSG må (bidra til å) samle relevante lover og regler. Hvis virksomheten eller deres kunder er underlagt Personopplysningsloven eller lignende (amerikanske eksempler som nevnes er FFIEC, GLBA, OCC, PCI DSS, SOX, HIPAA), fungerer SSG som et samlende element for å forstå

begrensningene dette medfører for programvaren som utvikles. I enkelte tilfeller vil SSG lage en enhetlig fremgangsmåte som fjerner redundans fra overlappende krav. En formell tilnærming vil lage en kobling mellom de relevante delene av lover og regler til krav (*control statements)* som forklarer hvordan organisasjonen oppfyller disse. Alternativt kan eksisterende forretningsprosesser som drives av en juridisk avdeling eller andre grupper som driver med risikohåndtering utgjøre kjernen i arbeidet med lover og regler. Målet med denne aktiviteten er å lage et sett med veiledninger for programvaresikkerhet slik at arbeidet med å sikre etterlevelse av lover og regler gjøre så effektivt som mulig (ved å fjerne duplikater).

- **CP1.2: Identifisér krav til personvern**
  Måten programvare håndterer personidentifiserbar informasjon (PII) er ofte spesifikt regulert (f.eks. i Personopplysningsloven), men også i tilfeller hvor det ikke er det, er personvern et hett tema. SSG tar en ledende rolle i å identifisere forpliktelser med hensyn til personopplysninger; både forpliktelser som stammer fra regelverk og fra kundenes forventinger. SSG bruker denne informasjonen for å fremme god praksis for personvern. Det er verdt å merke seg at tjenesteutsetting (f.eks. til nettskyen) ikke fritar deg fra personvernforpliktelser. Virksomheter som lager programvare som behandler personopplysninger (men som ikke selv gjør det) kan tilby personvernmekanismer og veiledning for sine kunder.
- **CP1.3: Lag retningslinjer og strategi**
  SSG veileder resten av organisasjonen ved å lage eller bidra til retningslinjer for programvaresikkerhet som tilfredsstiller lovpålagte eller kundestyrte sikkerhetskrav. Retningslinjene gir en enhetlig måte for å tilfredsstille (den potensielt lange) listen av sikkerhetsaspekter som må tas hensyn til på i styringen av virksomheten. Dermed kan prosjektgrupper slippe å måtte lære alle detaljene som etterlevelse av alle relevante lover og regler innebærer, og de slipper også å måtte lære seg alle kundenes sikkerhetskrav på egen hånd. Retningslinjene fra SSG er noen ganger fokusert på fundamentale områder som håndtering av personopplysninger eller bruk av kryptografi. I noen tilfeller vil retningslinjene relatere seg direkte til SSDL og hvordan den gjennomføres i virksomheten. Arkitekturstandarder og kodingsveiledninger er ikke eksempler på programvaresikkerhetsretningslinjer, men retningslinjer som anbefaler/krever kodingsveiledninger og arkitekturstandarder for visse typer applikasjoner er gode eksempler.

2.2.2 CP Nivå 2: Tilpass intern praksis til lovpålagte krav og retningslinjer, støttet av ledelsen

Ledelsen må åpent støtte SSG og initiativet for sikker programvareutvikling, inkludert behovet for å etterleve lover og regler. Ledere for risikostyring (sikkerhetsansvarlig e.l.) må eksplisitt ta ansvar for programvarerisiko. SSG og applikasjonseiere må sørge for at tjenesteavtaler (SLA) omfatter sikkerhetsegenskaper ved leverandørenes programvareleveranser.

- **CP2.1: Identifiser personvernrelaterte data**
  Virksomheten identifiserer hvilke typer personopplysninger som lagres i deres forskjellige systemer. En oversikt over personopplysninger kan organiseres på to måter; enten med utgangspunkt i den enkelte applikasjon og hvilke personopplysninger som brukes der, eller ved å ta utgangspunkt i forskjellige typer personopplysninger, og angi hvilke applikasjoner som berører dem. Kombinert med virksomhetens forpliktelser mht. personopplysninger vil en slik oversikt veilede virksomhetens personvernplaner. For eksempel gjør det SSG i stand til å lage en liste av databaser som, hvis de ble kompromittert, ville forutsette varsling av kundene.
- **CP2.2: Krev kvittering for sikkerhetsrelaterte risikoer relatert til lovkrav**
  Virksomheten har en formell prosess for å akseptere (rest-)risiko i forbindelse med etterlevelse av krav samt plassering av ansvar der det hører hjemme, og denne prosessen må anvendes i alle programvareutviklingsprosjekter. Den som har ansvar for å akseptere restrisikoen må underskrive på tilstanden til programvaren før den slippes. For eksempel, så kan kvitteringsretningslinjene kreve at

lederen for forretningsenheten skriver under på forhold rundt lover og regler som ikke er fullstendig håndtert, eller steg i SSDL relatert til etterlevelse av lover og regler som er hoppet over. Slik kvittering må gjøres eksplisitt og bevares for ettertiden, og alle unntak må spores.

- **CP2.3: Implementer og følg opp mekanismer for å sikre etterlevelse**
  Virksomheten kan demonstrere etterlevelse av relevante lover og regler fordi intern praksis er tilpasset retningslinjer og krav utviklet av SSG. SSG følger opp kontrollmekanismer og problemområder, og sørger for at revisorer og tilsyn er fornøyde. Hvis virksomhetens programvareutviklingssyklus (SDLC) er forutsigbar og pålitelig, kan SSG ofte bare lene seg tilbake og føre regnskap med prosessen. I motsatt fall må kanskje SSG ta en mer aktiv rolle som "dommer" (En virksomhet har gjerne mer enn én SDLC, og det kan være krevende å holde styr på alle). En virksomhet som gjør dette ordentlig kan eksplisitt knytte etterlevelse av lover og regler til sin SSDL.
- **CP2.4: Tapetsér alle leverandørkontrakter med programvaresikkerhets-SLAer**
  Leverandørkontrakter inneholder et sett av *Service Level Agreement* (SLA) klausuler som sikrer at leverandøren ikke setter virksomhetens etterlevelse av lover og regler i fare, og ei heller kommer i konflikt med virksomhetenes programvaresikkerhets-initiativ. Hver ny eller fornyet kontrakt inneholder et sett med klausuler som krever at leverandøren leverer et produkt eller en tjeneste som er kompatibel med virksomhetens sikkerhetsretningslinjer.
- **CP2.5: Øk ledelsens bevissthet rundt behovet for personvern og sikkerhet**
  SSG sørger for ledelsesforankring av aktiviteter rundt personvern og etterlevelse av lover og regler. Ledelsen presenteres for forklaringer i klart språk om virksomhetens forpliktelser i forbindelse med personvern og andre lover og regler, og de potensielle konsekvensene ved å ikke møte disse forpliktelsene. For enkelte virksomheter vil en forklaring av den direkte kostnaden og sannsynlige ettervirkninger fra kompromittering av data være en effektiv måte å bringe temaet på banen. For andre virksomheter vil det være mer effektivt å få en ekstern ekspert til å gi en presentasjon for styret, ettersom enkelte ledere har større tiltro til det eksterne kontra det interne perspektivet. Et sikkert tegn på skikkelig ledelsesbevissthet er tilstrekkelig tildeling av ressurser for å få jobben gjort.

2.2.3 CP Nivå 3: Organisasjonsmessige trusler, angrep, defekter og operasjonelle forhold driver utvikling av retningslinjer og krav til leverandører

Ledelsen må sørge for at retningslinjene for programvaresikkerhet oppdateres jevnlig basert på faktiske data, og må demonstrere virksomhetens løpende etterlevelse av lover og regler. SSG, applikasjonseiere og juridisk avdeling må sørge for at leverandører leverer programvare som tilfredsstiller relevante deler av virksomhetens retningslinjer.

- **CP3.1: Gi myndighetene det de vil ha**
  SSG har informasjonen myndigheter og tilsyn trenger. En kombinasjon av skriftlige retningslinjer, beskrivelse av tiltak og artefakter samlet i utførelsen av SSDL gir SSG evnen til å demonstrere hvordan virksomheten oppfyller lover og regler uten at det må slås full alarm for hver revisjon. I noen tilfeller vil tilsyn, revisorer og toppledelsen kunne tilfredsstilles med samme typer rapporter, som ofte kan genereres direkte fra forskjellige verktøy.
- **CP3.2: Pålegg til leverandører**
  Leverandører må forholde seg til de samme regler og retningslinjer som brukes internt. Leverandører må fremskaffe bevis for at deres programvaresikkerhetspraksis holder mål. Slike bevis kan omfatte resultater fra kodegjennomgang eller penetreringstester. Leverandører kan også bekrefte at de følger vise SSDL-prosesser. I noen tilfeller kan en BSIMM-poengsum[4] eller en vBSIMM[5]-poengsum brukes til å forsikre seg om at leverandøren oppfyller organisasjonens regler og retningslinjer.

[4] BSIMM-poengsummen er tallet på BSIMM-aktiviteter en virksomhet utfører (mellom 0 og 113). Det er p.t. bare Cigital som kan beregne en offisiell BSIMM-poengsum for en virksomhet; en selvevaluering eller lignende anses ikke for godt nok.
[5] vBSIMM = BSIMM for vendors

- **CP3.3: Mat resultater fra SSDL tilbake til strategi og retningslinjer**
  Informasjon fra SSDL mates rutinemessig tilbake til prosessen med å lage retningslinjer og strategi. Retningslinjer forbedres for å finne programvarefeil tidligere eller forhindre dem fra å oppstå i det hele tatt. Blindsoner elimineres basert på trender som observeres i forbindelse med at SSDL feiler. For eksempel kan utilstrekkelig arkitekturanalyse, tilbakevendende sårbarheter, ignorerte sikkerhets-sjekkpunkter (*security gates*), eller valg av feil firma for å utføre en penetreringstest avdekke svak-heter i retningslinjene. Over tid bør retningslinjene bli mer praktiske og enkle å gjennomføre. I ytterste konsekvens tilpasser retningslinjene seg til SSDL-data (dvs. resultater fra kodegjennom-ganger, tester, etc.), og forbedrer virksomhetens effektivitet.

2.3 Opplæring og øvelser (*Training*)

Hvordan kan du vite at du kan håndtere en oppgave hvis du aldri har gjort den før? Opplæring er svaret! Øvelse gjør mester – og den som tror at han er ferdig utlært, er ikke utlært, men ferdig. Programvare-sikkerhetsgruppen (SSG) spiller en viktig rolle her!

Hovedmålene for denne praksisen er å utdanne en kunnskapsrik arbeidsstokk og rette opp feil i prosesser. Arbeidsstokken må ha rolle-basert kunnskap som spesifikt inkluderer ferdighetene som kreves for å utføre deres SSDL-aktiviteter på en hensiktsmessig måte. Opplæringen må omfatte spesifikk informasjon om rotårsaker til feil som oppdages i prosessaktiviteter og resultater.

2.3.1 T Nivå 1: Gjør tilpasset, rollebasert opplæring tilgjengelig ved behov

SSG må bygge opp interessen for programvaresikkerhet i hele organisasjonen, og tilby rollespesifikt opplæringsmateriale, inkludert datamaskinbasert opplæring, som bygger inn erfaringer fra faktiske interne hendelser.

- **T1.1: Tilby sikkerhetsbevissthetsopplæring**
  SSG tilbyr kurs i sikkerhetsbevissthet for å fremelske en programvaresikkerhetskultur gjennom hele virksomheten. Opplæring kan gis av medlemmer fra SSG, et eksternt firma, den interne opplærings-avdelingen, eller gjennom et e-læringssystem. Kursinnholdet er ikke nødvendigvis skreddersydd for en spesifikk målgruppe. F.eks. kan alle programmerere, kvalitetskontrollører og prosjektledere delta på samme "Introduksjon til programvaresikkerhet"-kurs. Denne fellesaktiviteten kan utvides med en skreddersydd variant av et introduksjonskurs som eksplisitt adresserer den aktuelle virksomhetens kultur. Generelle introduksjonskurs som dekker grunnleggende IT-sikkerhet og høynivå program-varesikkerhetskonsepter genererer ikke tilfredsstillende resultater (tilfredsstiller ikke aktiviteten). På samme måte er det utilstrekkelig å bare tilby sikkerhetsbevissthetsopplæring til utviklere og ikke andre roller.
- **T1.5: Tilby rollespesifikk læreplan (verktøy, teknologistakker, "bug parade")**
  Programvaresikkerhetsopplæring går videre enn å bare bygge bevissthet, og gjør det mulig for "elevene" å bygge inn sikkerhetsaktiviteter i sitt arbeid. Opplæringen er skreddersydd til rollene til "elevene" som får informasjon om verktøy, teknologistakker, og type feil som er mest relevante for dem. En virksomhet kan f.eks. tilby fire spor for sine ansatte: ett for arkitekter, ett for Java-utviklere, ett for .NET-utviklere, og et fjerde for testere. Verktøyspesifikk opplæring er også noe man ofte ser som en del av pensum. Ikke glem at opplæring vil være nyttig for mange forskjellige roller i en virksomhet, inkludert QA, produktadministrasjon, ledelse, og andre.
- **T1.6: Utarbeid og bruk materiale som er spesifikt for virksomhetens historie**
  For å oppnå en sterk og varig endring i oppførsel omfatter opplæringsmaterialet hendelser som er spesifikke for virksomhetens historie. Når deltakere kan se seg selv i problemet. er det mer sannsynlig at de forstår at materialet er relevant for deres arbeid, og vet når og hvordan de skal anvende det de har lært. En måte å gjøre dette på er å bruke minneverdige angrep på virksomheten som eksempler i opplæringspensumet. Vær skeptisk til opplæring som dekker plattformer som ikke

brukes av utviklerne (Windows-utviklere bryr seg ikke om gamle Unix-problemer) eller eksempler på problemer som kun er relevant i programmeringsspråk som ikke lenger er i bruk i virksomheten (Java-utviklere trenger ikke forstå bufferoversvømmelse-angrep [*buffer overflow*] i C). Historier fra gamle dager i virksomheten kan bidra til å styre opplæringen i riktig retning kun dersom historiene fortsatt er relevante.

- **T1.7: Tilby individuell personlig opplæring på forespørsel**
  Virksomheten minsker byrden for de som skal læres opp, og reduserer kostnader ved å tilby opplæring på forespørsel for enkeltpersoner. Datamaskinbasert opplæring er det mest åpenbare valget her, og kan holdes oppdatert gjennom en abonnementsordning. Datamaskinbaserte kurs må skape engasjement og være relevante for å oppnå hensikten. For utviklere er det også mulig å tilby opplæring direkte gjennom utviklingsgrensesnitt (*IDE*) ved behov. Husk imidlertid at å bygge opp nye ferdigheter (som f.eks. kodegjennomgang) er kanskje mer egnet for tradisjonell instruktørledet opplæring.

2.3.2 T Nivå 2: Opprett programvaresikkerhetssatellitten

SSG må bygge opp og forbedre en satellitt gjennom sosiale aktiviteter, inkludert opplæring og relaterte arrangementer. SSG og ledere må sørge for at nyansatte eksponeres for organisasjonens sikkerhetskultur som en del av velkomstprogrammet. Se også SM2.3 i avsnitt 2.1.2

- **T2.5: Forbedre satellitten gjennom opplæring og arrangementer**
  SSG forsterker sitt sosiale nettverk ved å lage spesielle arrangementer for satellitten. Satellitten lærer om avanserte tema eller lytter til gjesteforelesere. Det skader ikke å tilby pizza med drikke til. Et fast telefon/videomøte dekker ikke denne aktiviteten, som handler like mye om å bygge kameratskap som å dele kunnskap eller øke organisasjonsmessig effektivitet. Det finnes ikke noen erstatning for å møtes ansikt til ansikt, selv om det bare skjer en eller to ganger i året.
- **T2.6: Inkluder sikkerhetsressurser i nyansattopplæring**
  Prosessen for å integrere nyansatte i utviklingsorganisasjonen krever en modul om programvaresikkerhet. En generell nyansattprosedyre kan omfatte ting som å velge et godt passord og sørge for at folk ikke sniker seg inn i bygningen ved å følge etter deg, men dette utvides til å dekke tema som sikker koding, SSDL og interne sikkerhetsressurser. Målet er at nyansatte skal *forbedre* sikkerhetskulturen – det er generelt mye gjennomtrekk i utviklingsvirksomheter. Selv om en generell nyansattmodul er nyttig, kan den ikke fullt ut erstatte et mer komplett introduksjonskurs i programvaresikkerhet.

2.3.3 T Nivå 3: Belønn ferdigheter og skap en karrierevei

Bygg moral. Ledelsen og SSG må sørge for at alle ansatte får tilstrekkelig anerkjennelse for å fullføre opplæringsløpet. Ledere, applikasjonseiere og SSG må tilby opplæring til leverandører og innleide utviklere som en måte å spre sikkerhetskulturen. Ledere og SSG må fortsette å bygge opp under satellittens moment ved å markedsføre sikkerhetskulturen eksternt. SSG må være tilgjengelig, i hvert fall i perioder, for de som ønsker programvaresikkerhetsbistand og veiledning. Ledelsen må sørge for at alle ansatte mottar slik opplæring minst på en årlig basis.

- **T 3.1: Belønn progresjon gjennom pensum (sertifisering eller HR)**
  Lærdom er sin egen belønning, men det å komme seg gjennom sikkerhetspensumet gir andre fordeler også. Utviklere og testere ser det å lære om sikkerhet som en fordel for karrieren. Belønningssystemet kan være formelt og føre til sertifisering eller en offisiell kreditering i personalsystemet, eller det kan være mer uformelt og gjøre bruk av motiverende grep som f.eks. anbefalingsbrev for satellitten skrevet rett før en årlig evaluering. Påvirkning av sikkerhet på karriere-progresjon kan gjøres mer åpenbar ved å involvere virksomhetens opplæringsavdeling eller personalavdeling,

men SSG burde fortsatt monitorere sikkerhetskunnskap i virksomheten, og ikke fullstendig gi fra seg kontroll eller oversikt.

- **T3.2: Tilby opplæring til leverandører eller innleide arbeidere**
  Virksomheten tilbyr sikkerhetsopplæring til leverandører og konsulentfirmaer. Det er lettere å bruke tid og penger på hjelpe leverandører til å få gjort ting skikkelig enn å prøve å finne ut hva de har rotet til senere. I beste fall får innleide samme opplæringen som gis til egne ansatte. Det er mer naturlig å lære opp individuelle konsulenter enn å utdanne hele konsulentfirmaer (og dette er følgelig et fornuftig sted å starte). Det er selvfølgelig viktig å lære opp alle som arbeider med din programvare, uansett hvor de er ansatt.
- **T3.3: Vær vert for eksterne programvaresikkerhetsarrangementer**
  Virksomheten fremhever sin sikkerhetskultur ved å arrangere konferanser og seminarer som er åpne for eksterne deltakere. Ansatte kan ha nytte av å høre andres perspektiver. Virksomheten som helhet tjener på å fremheve sin sikkerhetskredibilitet.
- **T3.4: Krev en årlig oppfrisker**
  Alle som er involvert i å lage programvare pålegges å ta et årlig oppdateringskurs i programvare-sikkerhet. Oppfriskningskurset holder ansatte oppdatert på sikkerhet, men fanger også opp nyansatte og sørger for at virksomheten ikke mister fokus pga. gjennomtrekk. SSG kan f.eks. bruke en halv dag på å gi en oppdatering på sikkerhetslandskapet og forklare endringer i retningslinjer og standarder. Oppfriskeren kan rulles ut som en del av en felles sikkerhetsdag for virksomheten (f.eks. i Nasjonal Sikkerhetsmåned) eller i forbindelse med en intern sikkerhetskonferanse.
- **T3.5: Etabler SSG kontortid**
  SSG tilbyr hjelp til alle som ber om det i en angitt lab-tid eller fast kontortid. Ved å fungere som en uformell ressurs for folk som ønsker å løse sikkerhetsproblemer, kan SSG utnytte "læringsøyeblikk" og fremheve gulroten foran pisken. Kontortiden kan f.eks. være en ettermiddag i uka på kontoret til et senior SSG medlem. Ambulerende kontortider er også en mulighet, med besøk hos bestemte produkt- eller applikasjonsgrupper etter anmodning.
- **T3.6: Identifiser satellitten gjennom opplæring**
  Satellitten begynner som en samling mennesker som er over gjennomsnittet interessert i sikkerhet og/eller har slike ferdigheter, og som er spredt utover i virksomheten. Det å identifisere denne gruppen er et steg i retning av å lage et sosialt nettverk som framskynder integrering av sikkerhet i programvareutviklingen. En måte å gjøre dette på er å notere seg mennesker som utmerker seg under kurs og opplæring. En hær av frivillige er generelt lettere å lede enn en som er tvangsutskrevet.

3 Etterretning (*Intelligence*)

Domenet **Etterretning** omfatter Angrepsmodeller; Sikkerhetsfunksjonalitet og design; og Standarder og krav.

3.1 Angrepsmodeller (*Attack Models*)

Etter et foredrag om BSIMM fikk jeg et godt spørsmål som jeg ikke kunne svare på: "Hvilket nivå skal man legge seg på?" Mitt ikke-svar var at "da må du gjøre en risikobasert vurdering", som egentlig bare er en annen måte å si "det kommer an på". I utgangspunktet hadde jeg sagt at "alle" burde kunne starte med aktivitetene på nivå 1 i hver praksis, men her skilte daværende aktivitet AM1.1 seg ut i tallmaterialet – bare 21 av de 67 undersøkte bedriftene in BSIMM-V vedlikeholder en liste av de topp N mulige angrepene mot seg selv. Dette virker som en ganske grei aktivitet å gjøre et par ganger i året, og jeg finner ingen god forklaring på hvorfor det ikke gjøres, annet enn at virksomhetene ikke opplever at det har noen nytteverdi. I BSIMM7 ble denne aktiviteten oppgradert til AM2.5.

3.1.1 AM Nivå 1: Lag kunnskapsbase med angrep og dataverdier.

SSG må identifisere potensielle angripere og dokumentere både angrepene som representerer den største bekymringen for virksomheten, og eventuelle angrep som allerede har funnet sted. SSG må kommunisere

angrepsinformasjonen til alle interessenter. Forretningsdelen av virksomheten må utforme et graderingssystem for data som SSG kan bruke for å katalogisere og prioritere applikasjoner.

Her handler det om å opprette en kunnskapsbase om angrep og dataverdier.

- **AM1.2: Lag et graderingssystem for data og katalogiser**
  Klassifiser programvaren etter graderingen på informasjonen den behandler. Dermed kan man prioritere programvare etter gradering.
- **AM1.3: Identifiser mulige angripere**
  SSG identifiserer mulige angripere for å forstå deres motivasjoner og evner. Spesifikk og kontekstavhengig informasjon er alltid mer nyttig enn å klippe og lime fra noen andres liste.
- **AM1.5: Samle inn informasjon om nye angrep i bransjen**
  SSG er proaktiv, og sørger for å lære om nye typer angrep og sårbarheter. Informasjonen finner man ved å delta på konferanser, følge med på forum der angripere ferdes, og lese relevante publikasjoner (både på papir og elektronisk).

3.1.2 AM Nivå 2: Spre informasjon om angripere og relevante angrep

AM Nivå 2 handler om å strekke seg litt lenger når det gjelder å forstå angripere og relevante angrep. SSG må samle informasjon om angrep og utvide sin kunnskap til å omfatte både høynivå angrepsmønstre og mer detaljerte "misbrukseksempler" (*abuse cases*). Angrepsmønstre må inkludere teknologispesifikk informasjon som er relevant for virksomheten.

- **AM2.1: Lag angrepsmønstre (*attack patterns*) og misbrukseksempler (*abuse cases*) knyttet til potensielle angripere**
  SSG forbereder sikkerhetstesting og arkitekturanalyse ved å bygge angrepsmønstre og eksempler på mulig misbruk; disse må ikke nødvendigvis bygges fra bunnen hver gang, ettersom det kan være forhåndskonstruerte "sett" for applikasjoner av en gitt type.
- **AM2.2: Lag teknologispesifikke angrepsmønstre**
  SSG lager teknologispesifikke angrepsmønstre som samler kunnskap om angrep rettet mot spesifikke teknologier. Hvis organisasjonens web-programvare baserer seg på det aller siste i nettleser-egenskaper, kan SSG katalogisere alle særegenhetene til de mest populære nettleserne, og hvordan de kan utnyttes.
- **AM2.5: Lag og vedlikehold en liste av de topp N mulige angrepene**
  SSG kan hjelpe organisasjonen med å forstå grunnleggende egenskaper ved angrep ved å liste opp angrepene som er mest relevante for dem. Man kan sortere på angrep som må kunne håndteres "nå", "snart", eller "i fremtiden".
- **AM2.6: Samle inn og offentliggjør angrepshistorier**
  For å maksimere verdien av dyrekjøpte erfaringer, samler SSG inn og offentliggjør internt historier om angrep mot organisasjonen. Både vellykkede og mislykkede angrep kan være verdt å merke seg. Fordelen er at dette kan danne en motivasjon for programvaresikkerhet som er basert på organisasjonens virkelighet, ikke topp-ti lister fra utlandet.
- **AM2.7: Etabler et internt forum for å diskutere angrep**
  Organisasjonen har et internt forum der SSG, assosierte sikkerhetsfolk ("satellitten") og andre diskuterer angrep. Det kan være en epostliste, et internt webforum eller lignende hvor medlemmer deler siste informasjon om offentlig kjente hendelser. Detaljert analyse av relevante angrep og sårbarheter kan være spesielt nyttig hvis de leder til diskusjoner om hvordan man praktisk kan løse problemene på utviklersiden. Det å kun videresende meldinger fra offentlige fora oppnår ikke de samme resultatene som en aktiv diskusjon.

3.1.3 AM Nivå 3: Forske på nye angrepsmønster og mottiltak

På AM nivå 3 dreier det seg om å drive egen forskning på angrepsmønstre og mottiltak. SSG må forske på angrep på virksomhetens programvare for å ligge på forskudd i forhold til angrepsaktivitet. SSG må tilby kunnskap og automatisering til revisorer og testere for å sikre at deres aktiviteter gjenspeiler faktiske angrep som utføres mot virksomhetens programvare, i tillegg til potensielle angrep.

- **AM3.1: Ha en forskergruppe som utvikler nye angrepsmetoder**
  SSG har en forskergruppe som identifiserer og finner mottiltak mot nye angrepsmetoder før virkelige angripere en gang er klar over at de finnes. Dette er *ikke* en gjeng med penetreringstestere som finner nye tilfeller av en kjent type sårbarhet.
- **AM3.2: Lag og ta i bruk automatiserte verktøy som gjør det angripere vil gjøre**
  SSG utstyrer testere og revisorer med automatiserte verktøy, f.eks. hvis forskergruppen har funnet en ny type angrep, kan det være behov for et nytt verktøy. Å skreddersy verktøy til organisasjonens eksisterende teknologi og potensielle angripere er en kjempeidé.

3.2 Sikkerhetsfunksjonalitet og design (*Security Features and Design*)

En "feature" er kanskje i hovedsak en egenskap, men i de fleste tilfeller dreier det seg om funksjonalitet, så vi kjører videre med det. Hovedmålet for denne praksisen er etablering av tilpasset kunnskap om sikkerhets-funksjonalitet, -rammeverk og -mønster. Den tilpassede kunnskapen må drive arkitektur- og komponent-beslutninger.

3.2.1 SFD Nivå 1: Publisér sikkerhetsfunksjonalitet og arkitektur

SSG må tilby arkitekter og utviklere veiledning på sikkerhetsfunksjonalitet og sikkerhetsmekanismer, og SSG må bidra direkte til arkitekturgrupper.

- **SFD1.1: Lag og publisér sikkerhetsfunksjonalitet**
  Enkelte problemer løses best en gang for alle. I stedet for å la hver prosjektgruppe implementere sine egne sikkerhetsmekanismer (autentisering, rolleadministrasjon, nøkkeladministrasjon, revisjon/logg, kryptografi, protokoller), gir programvaresikkerhetsgruppen (SSG) proaktiv veiledning ved å bygge opp og publisere sikkerhetsmekanismer som andre grupper kan bruke. Prosjektgrupper drar fordel av implementasjoner som er forhåndsgodkjent av SSG, og SSG unngår å gjentatte ganger måtte spore opp subtile feil som ofte sniker seg inn i sikkerhetsmekanismer. SSG kan identifisere en eksisterende implementasjon de liker, og anbefale den som den aksepterte løsningen.
- **SFD1.2: Knytt sammen SSG og arkitektur**
  Sikkerhet er en fast del av virksomhetens programvarearkitekturdiskusjon. Arkitekturgruppen tar ansvar for sikkerhet på samme måte som de tar ansvar for ytelse, tilgjengelighet eller skalerbarhet. En måte å unngå at sikkerhet faller ut av diskusjonen er å sørge for at et SSG-medlem deltar på faste arkitekturmøter. I andre tilfeller kan virksomhetsarkitektur hjelpe SSG til å lage sikre design som integreres skikkelig i virksomhetens designstandarder.

3.2.2 SFD Nivå 2: Bygg og identifisér sikkerhetsløsninger

SSG må tilby sikker-ved-design (*secure by design*) rammeverk, og må være tilgjengelig for og kapabel til å løse designproblemer for andre.

- **SFD2.1: Bygg sikker-ved-design (*secure by design*) rammeverk og felles biblioteker**
  SSG tar en proaktiv rolle i programvaredesign ved å lage eller peke på sikker-ved-design mellom-varerammeverk eller felles biblioteker. I tillegg til å utnytte eksempelets makt vil slik mellomvare bidra til arkitekturanalyse og kodegjennomgang ettersom byggesteinene gjør det enklere å oppdage feil. SSG kan f.eks. modifisere et populært web-rammeverk som Spring for å gjøre det enkelt å

oppfylle inputvalideringskrav. Etter hvert kan SSG skreddersy kodegjennomgangsregler spesifikt for komponentene den tilbyr. Når man innfører et mellomvarerammeverk (eller en hvilken som helst ofte brukt programvare) er det avgjørende å ha en grundig sikkerhetsgjennomgang før publisering. Det bidrar ikke til bedre programvaresikkerhet hvis man oppfordrer til innføring og bruk av usikker mellomvare! Generelle åpen kildekode sikkerhetsarkitekturer, inkludert OWASP ESAPI, bør ikke i utgangspunktet regnes for å være sikre uten ytterligere konfigurering.

- **SFD2.2: Bygg opp SSGs evne til å løse vanskelige designproblemer**
  Når SSG er involvert tidlig i designprosessen, bidrar den til ny arkitektur og løser vanskelige designproblemer. Den negative virkningen sikkerhet har på andre begrensninger (tid til markedet, pris, osv.) minimaliseres. Hvis en arkitekt fra SSG er involvert i designet av en ny protokoll kan han eller hun analysere sikkerhetsaspektene til eksisterende protokoller og identifisere elementer som burde dupliseres eller unngås. Det er mer effektiv å designe for sikkerhet i utgangspunktet enn å analysere et eksisterende design med henblikk på sikkerhet og skrive om når feil oppdages. Enkelte designproblemer vil kreve spesifikk ekspertise som ikke finnes i SSG.

3.2.3 SFD Nivå 3: Aktivt gjenbruk av godkjente sikkerhetsegenskaper og sikkerhetsmekanismer og sikker-ved-design rammeverk

SSG må tilby flere modne design-mønstre (*design patterns*) tatt fra eksisterende programvare og teknologi-stakker (*technology stacks*). Ledere må sørge for at det er formell konsensus i hele virksomheten når det gjelder sikre designvalg. Ledere må også kreve at definerte sikkerhetsmekanismer og rammeverk brukes til enhver tid der dette er mulig.

- **SFD3.1: Dann et evalueringsutvalg eller en sentral komité for å godkjenne og vedlikeholde sikre designmønstre**
  Et evalueringsutvalg eller en sentral sikkerhetsdesignkomité formaliserer prosessen for å oppnå konsensus rund designbehov og sikkerhetsavveininger. I motsetning til arkitekturkomitéen er denne gruppen spesifikt fokusert på å gi sikkerhetsveiledning. Gruppen kan også periodisk gjennomgå allerede publiserte designstandarder (spesielt rundt kryptografi) for å sørge for at designavgjørelser ikke går ut på dato.
- **SFD3.2: Krev bruk av godkjente sikkerhetsmekanismer og rammeverk**
  Utviklere må hente sine sikkerhetsmekanismer fra en godkjent liste. Dette har to fordeler: Utviklere bruker ikke tid på finne opp hjulet på nytt, og de som gjennomgår koden slipper å måtte finne de samme gamle feilene i nye prosjekter. I særdeleshet, jo mer et prosjekt bruker velprøvde komponenter, jo lettere blir arkitekturanalyse og kodegjennomgang. Gjenbruk er en vesentlig fordel ved en konsistent programvarearkitektur.
- **SFD3.3: Finn og publiser modne designmønstre fra virksomheten**
  SSG fostrer sentralisert gjenbruk av design ved å finne og publisere modne designmønstre (*design patterns*) fra og gjennom hele virksomheten. En del av websiden for SSG kunne promotere positive elementer identifisert under arkitekturanalyse slik at gode ideer sprer seg. Denne prosessen burde være formalisert. Ad hoc, tilfeldig registrering er ikke tilstrekkelig. I noen tilfeller vil en sentral arkitektur- eller teknologi-gruppe legge til rette for og forbedre denne aktiviteten.

3.3 Standarder og krav (*Standards and Requirements*)

Hovedmålet for praksisen "Standarder og krav" er å lage retningslinjer for alle interessenter. Ledere og programvaresikkerhetsgruppen (SSG) må dokumentere programvaresikkerhetsvalg, og formidle dette materialet til alle som er involvert i livssyklusen for sikker programvareutvikling (SSDL), inkludert eksterne aktører.

3.3.1 SR Nivå 1: Gjør sikkerhetsstandarder og krav lett tilgjengelige

SSG må gjøre grunnleggende kunnskap tilgjengelig, inkludert sikkerhetsstandarder[6], sikre kodestandarder og krav for etterlevelse av relevante lover og regler. Ledere må sørge for at programvaresikkerhetsinformasjon holdes oppdatert og gjøres tilgjengelig for alle.

- **SR1.1: Lag sikkerhetsstandarder**
  Programvaresikkerhet krever mye mer enn sikkerhetsmekanismer, men sikkerhetsmekanismer er også en del av jobben. SSG møter virksomhetens behov for sikkerhetsveiledning ved å lage standarder som forklarer den aksepterte måten å følge retningslinjene og utføre spesifikke sikkerhetsorienterte operasjoner. En standard kan f.eks. beskrive hvordan man utfører autentisering ved bruk av J2EE eller hvordan man bestemmer ektheten til en programvareoppdatering. Standarder kan rulles ut på flere forskjellige måter. I noen tilfeller kan standarder og veiledninger bli automatisert i utviklingsmiljøer (f.eks. bygget inn i en IDE som Eclipse). I andre tilfeller kan veiledninger eksplisitt kobles til kodeeksempler for å gjøre dem mer praktiske og relevante.
- **SR1.2: Lag en sikkerhetsportal**
  Virksomheten har et sentralt plassert område for informasjon om programvaresikkerhet. Dette er typisk en intern webside som administreres av SSG. Folk henvender seg til siden for det siste nye innen sikkerhetsstandarder og krav, i tillegg til andre ressurser som gjøres tilgjengelig av SSG. En interaktiv Wiki er bedre enn en statisk side med veiledningsdokumenter som sjelden endrer seg. Virksomheter kan øke verdien på slikt materiale med epostlister og fysiske møter.
- **SR1.3: Oversett begrensninger fra lover og regler til krav**
  Det som skal til for å etterleve lover og retningslinjer oversettes til programvarekrav for individuelle prosjekter. Dette er en krumtapp i virksomhetens etterlevelsestrategi – ved å representere begrensningene eksplisitt som krav blir det å demonstrere etterlevelse en håndterlig oppgave. F.eks. hvis virksomheten rutinemessig lager programvare som prosesserer kredittkort-transaksjoner, kan PCI DSS [10] spille en rolle i SSDL i kravfasen. I andre tilfeller kan teknologistandarder laget for internasjonal interoperabilitet inneholde sikkerhetsveiledninger. Det å representere disse standardene som krav hjelper med sporbarhet og synlighet i tilfelle av revisjon.

3.3.2 SR Nivå 2: Kommuniser formelt godkjente standarder internt og til leverandører

Ledere må sørge for at det er en formell prosess som brukes til å lage standarder som er spesifikk til teknologistakker. Ledere, SSG og produkteiere må sørge for at SLAer er forsterket med kontraktsspråk som er godkjent av egne jurister. SSG må sørge for at all åpen kildekode er identifisert i virksomhetens kode.

- **SR2.2: Lag en vurderingskomite for standarder**
  Virksomheten lager en komite for å formalisere prosessen som brukes til å utvikle standarder, og for å sørge for at alle interessenter har mulighet til å komme med innspill. Vurderingskomiteen kan operere slik at den oppnevner en "forkjemper" for enhver foreslått standard. Det påligger forkjemperen å demonstrere at standarden oppfyller sine mål, og å innhente godkjenning og oppfølging fra vurderingskomiteen. Ansvaret for å opprette og styre vurderingskomiteer for standarder tas noen ganger av de som er ansvarlige for virksomhetsarkitektur eller virksomhetsrisiko (f.eks. sikkerhetsansvarlig).
- **SR2.3: Lag standarder for teknologistakker**
  Virksomheten standardiserer på spesifikke teknologistakker. For SSG betyr dette redusert arbeidsbelastning ettersom de ikke trenger å utforske nye teknologirisikoer for hvert nye prosjekt. Ideelt sett vil virksomheten lage en sikker basiskonfigurasjon for hver teknologistakk, noe som ytterligere reduserer arbeidsmengden som kreves for å bruke stakken på en trygg måte. En stakk kan omfatte et operativsystem, en database, en applikasjonstjener, og et kjøremiljø for et administrert språk.

[6] Her tenker vi mer på interne standarder enn ISO-standarder etc.

Sikkerhetsfrontlinjen (dvs. den vidunderlige nye verden) er et godt sted å starte for å sikre forankring i ledelsen og blant utviklerne. For tiden er mobilteknologistakker og nettskybaserte teknologistakker to områder hvor det lønner seg å spandere ekstra oppmerksomhet på sikkerhet.

- **SR2.4: Identifiser åpen kildekode**
  Det første steget mot å styre risikoen introdusert av åpen kildekode er å identifisere de åpen kildekode komponentene som er i bruk. Det er ikke uvanlig å oppdage gamle versjoner av komponenter med kjente sårbarheter, eller flere versjoner av den samme komponenten. Automatiserte verktøy for å finne åpen kildekode er en måte å tilnærme seg denne aktiviteten. En prosess som kun baserer seg på at utviklere må be om lov genererer ikke tilfredsstillende resultater. På neste modenhetsnivå innlemmes denne aktiviteten av en retningslinje som begrenser bruk av åpen kildekode.
- **SR2.5: Lag standardtekster for SLAer**
  SSG samarbeider med juridisk avdeling for å lage standard SLA-tekst som kan brukes i kontrakter med leverandører og konsulenter. Juridisk avdeling forstår at slik standardtekst bidrar til å unngå problemer i forbindelse med personvern og etterlevelse av lover og regler. I henhold til avtalen må leverandører og konsulenter oppfylle virksomhetens programvaresikkerhetsstandarder. Standardteksten kan identifisere programvaresikkerhetskontroll-løsninger for leverandører som vBSIMM-målinger eller BSIMM poengsum.

3.3.3 SR Nivå 3: Krev risikostyringsavgjørelser for bruk av åpen kildekode

Ledere og SSG må demonstrere at åpen kildekode som brukes i virksomheten er gjenstand for de samme risikostyringsprosessene som kode som lages internt, og sørge for at alle relevante standarder kommuniseres til tredjepartsleverandører.

- **SR3.1: Kontroller risiko fra åpen kildekode**
  Virksomheten har kontroll over sin eksponering for risiko som kommer fra bruk av åpen-kildekode-komponenter. Bruk av åpen kildekode kan være begrenset til predefinerte prosjekter. Slik bruk kan også være begrenset til åpen kildekode-versjoner som har vært gjenstand for en sikkerhetsgjennomgang av SSG, fått uakseptable sårbarheter fjernet, og gjort tilgjengelig kun via interne tjenere. Juridisk avdeling er ofte en spydspiss i arbeidet med å kontrollere åpen kildekode pga. det "virale" lisensproblemet knyttet til GPL-kode[7]. Det å få juridisk avdeling til å forstå informasjonssikkerhetsrisikoer kan bidra til å bevege en virksomhet i retningen av å praktisere fornuftig åpen kildekode-hygiene.
- **SR3.2: Kommuniser standarder til leverandører**
  SSG samarbeider med leverandører for å lære dem opp og promotere virksomhetens sikkerhetsstandarder. Et sunt forhold til en leverandør kan ikke garanteres gjennom kontraktsspråk alene. SSG involverer seg med leverandørene, diskuterer leverandørens sikkerhetspraksis, og forklarer i konkrete vendinger (snarere enn advokatspråk) hva virksomheten forventer fra leverandøren. Hver gang en leverandør annekterer virksomhetens sikkerhetsstandarder er det en klar seier. Når en virksomhets SSDL er offentlig tilgjengelig er det mye enklere å kommunisere programvare-sikkerhetsforventninger. På samme måte kan forventninger gjøres veldig klare ved å dele intern praksis og tiltak (dokumentasjon av formelle eller uformelle BSIMM-målinger kan også bidra til dette).
- **SR3.3: Bruk standarder for sikker koding**
  Standarder for sikker koding hjelper utviklere til å unngå de mest åpenbare feilene, og etablerer grunnreglene for kodegjennomgang. Standarder for sikker koding er nødvendigvis spesifikke for et programmeringsspråk, og kan adressere bruk av populære rammeverk og biblioteker. Hvis virksomheten allerede har kodestandarder for andre formål, bør standardene for sikker koding bygge på disse. Et tydelig sett med standarder for sikker koding er en god måte å rettlede både manuell og

[7] http://www.gnu.org/licenses/

automatisert kodegjennomgang, i tillegg til å forbedre sikkerhetsopplæringen med relevante eksempler.

4 SSDL Trykkpunkter (*SSDL Touchpoints*)

Domenet **SSDL Trykkpunkter** henter navnet sitt fra Gary McGraws samling av god praksis for programvaresikkerhet [5], og omfatter Arkitekturanalyse; Kodegjennomgang; og Sikkerhetstesting.

4.1 Arkitekturanalyse (*Architecture Analysis*)

Hovedmålet for praksisen Arkitekturanalyse er kvalitetskontroll. De som utfører arkitekturanalyse må sørge for at strukturelle sikkerhetsfeil oppdages og korrigeres. Programvarearkitekter må sørge for at standarder følges, og at godkjente sikkerhetsløsninger gjenbrukes.

Det kan være ekstra nyttig å gjøre arkitekturgjennomgang med mer enn en arkitekt til stede for å beskrive arkitekturen. Da kan viktige funn gjerne komme av seg selv, ettersom forståelsen av arkitekturen varierer og dette fort blir åpenbart med flere til stede. Både nyttig, og potensielt interessant å bivåne!

4.1.1 AA Nivå 1: Utfør risikodrevne arkitektur-gjennomganger, ledet av SSG.

Virksomheten må gjøre et lettvekts risikoklassifiserings-skjema for programvare tilgjengelig. Programvaresikkerhetsgruppen (SSG) må begynne å lede arkitekturanalyseaktiviteter, spesielt i forbindelse med høyrisiko-applikasjoner, som en måte å bygge opp kompetanse internt og demonstrere verdien på designnivå.

- **AA1.1: Foreta gjennomgang av sikkerhetsmekanismer**
  For å komme i gang med arkitekturanalyse, fokuser prosessen på en gjennomgang av sikkerhetsmekanismer. Sikkerhetsbevisste evaluatorer identifiserer først sikkerhetsmekanismer i en applikasjon (autentisering, aksesskontroll, bruk av kryptografi, osv.), og studerer så designet mens de leter etter problemer som kan få disse mekanismene til å feile eller på annet vis vise seg utilstrekkelige. For eksempel, et system som kunne utsettes for et angrep som ga forhøyede privilegier pga. utilstrekkelig aksesskontroll, eller et system som lagret usaltede passordhasher ville begge identifiseres i denne type gjennomgang. På høyere modenhetsnivåer vil denne aktiviteten komme i skyggen av en grundigere tilnærming til arkitekturanalyse som ikke kun er fokusert på mekanismer. I noen tilfeller kan bruk av virksomhetens sikker-ved-design-komponenter strømlinjeforme denne prosessen.
- **AA1.2: Foreta designgjennomgang for høyrisiko-applikasjoner**
  Virksomheten lærer fordelene ved arkitekturanalyse ved å se håndfaste resultater for et utvalg høyrisiko-applikasjoner med høy profil. Evaluatorene må ha noe erfaring med utføring av arkitekturanalyse og bryting av arkitekturen som studeres. Hvis SSG ikke enda er satt opp for å utføre en arkitekturanalyse i dybden, bruker den konsulenter for å løse denne oppgaven. Ad hoc gjennomgangsparadigmer som hviler tungt på ekspertise kan brukes her, men i det lange løp skalerer det ikke.
- **AA1.3: La SSG lede gjennomgangsaktiviteter**
  SSG tar en lederrolle rolle i gjennomføring av arkitekturanalyse for å begynne å bygge opp virksomhetens evne til å avdekke designfeil. Det å bryte en arkitektur er tilstrekkelig mye av en kunst til at SSG må være dyktig til det før de kan overlate jobben til arkitektene, og dyktighet krever øvelse. SSG kan ikke oppnå suksess alene, de heller, og trenger sannsynligvis hjelp fra arkitektene eller utviklerne for å kunne forstå designet. Med et klart design for hånden kan SSG utføre analysene med et minimum av interaksjon med prosjektgruppen. På høyere modenhetsnivåer flyttes ansvaret for å lede gjennomgangsaktiviteter til programvarearkitekter. Tilnærminger til arkitekturanalyse (og trusselmodellering) utvikler seg over tid. Ikke forvent å etablere en prosess og så bruke den til evig tid.

- **AA1.4: Bruk et risiko-spørreskjema for å rangere applikasjoner**
  For å legge til rette for arkitekturanalyse og andre prosesser, bruker SSG et risiko-spørreskjema for å samle inn grunnleggende informasjon om hver applikasjon slik at det kan danne grunnlaget for et risiko-klassifisering og -prioriterings regime. Spørsmål kan inkludere "Hvilke(t) programmerings-språk er applikasjonen skrevet i?", "Hvem bruker applikasjonen?", og "Håndterer applikasjonen sensitive personopplysninger?" Et kvalifisert medlem av utviklergruppen fyller ut spørreskjemaet. Spørreskjemaet er kort nok til å kunne fylle ut i løpet av et par timer. SSG kunne bruke svarene til å gruppere applikasjonen som høy, medium eller lav risiko. Ettersom det kan være lett å tilpasse seg et risiko-spørreskjema, er det viktig å utføre noen punktsjekker for validitet og nøyaktighet. En overdreven avhengighet av selv-rapportering og automatisering kan virke mot sin hensikt og gjøre denne aktiviteten impotent.

4.1.2 AA Nivå 2: Spre bruk av dokumentert arkitekturanalyse-prosess.

SSG må legge til rette for bruk av arkitekturanalyse gjennom hele virksomheten ved å gjøre seg selv tilgjengelig som en ressurs og mentor. SSG må definere en arkitekturanalyseprosess basert på et felles arkitekturbeskrivelse-språk og standard angrepsmodeller.

- **AA2.1: Definer og bruk arkitekturanalyse-prosess**
  SSG definerer og dokumenterer en prosess for å gjennomføre arkitekturanalyse, og anvender den i de designgjennomgangene de foretar. Prosessen inkluderer en standardisert tilnærming for å tenke rundt angrep og sikkerhetsegenskaper. Prosessen er definert grundig nok til at folk utenfor SSG kan læres opp til å utføre den. Man må særlig vie oppmerksomhet til dokumentasjon av både arkitekturen som vurderes og eventuelle feil som avdekkes. Stammekunnskap teller ikke som en definert prosess. Microsofts STRIDE og Cigitals ARA er eksempler på en slik prosess. Bemerk at også disse to metodikkene for arkitekturanalyse har utviklet seg betydelig over tid. Pass på å aksessere oppdaterte kilder for informasjon om arkitekturanalyse, ettersom mange tidlige publikasjoner er utdaterte og ikke lenger anvendbare.
- **AA2.2: Standardiser arkitekturbeskrivelser (inkludert dataflyt)**
  Virksomheten bruker et format man har blitt enige om for å beskrive arkitektur, inkludert en måte å representere dataflyt. Dette formatet, kombinert med en arkitekturanalyseprosess, gjør arkitektur-analyse gjennomførbart for folk som ikke er sikkerhetseksperter. En standard arkitekturbeskrivelse kan forbedres til å gi et eksplisitt bilde av informasjonsverdier som trenger beskyttelse. Standardiserte ikoner som brukes konsistent i UML-diagrammer, Visio-maler og tusjtavle-klotring er spesielt nyttige.

4.1.3 AA Nivå 3: Bygg opp kunnskap i arkitekturgruppen om revidering og forbedring av sikkerhetsfeil

Programvarearkitekter i hele virksomheten må lede analyseaktiviteter, og må bruke analyseresultater for å lage og oppdatere arkitekturmønstre som er sikre.

- **AA3.1: La programvarearkitekter lede gjennomgangsaktiviteter**
  Programvarearkitekter i hele virksomheten leder arkitekturanalyseprosessen mesteparten av tiden. SSG kan fortsatt bidra til arkitekturanalyse i en rådgivende rolle, eller i spesielle tilfeller. Denne aktiviteten krever en godt forstått og veldokumentert arkitekturanalyseprosess. Til og med i slike tilfeller er det vanskelig å oppnå konsistens, ettersom det å bryte en arkitektur krever så mye erfaring.
- **AA3.2: Mat analyseresultater inn i standard arkitekturmønstre**
  Feil som identifiseres i løpet av arkitekturanalyse mates tilbake til sikkerhetsdesignkomiteen (se SFD 3.1 i avsnitt 3.2.3) slik at tilsvarende feil kan unngås i framtiden gjennom forbedrede designmønstre. Sikkerhetsdesignmønstre kan samvirke på overraskende måter som bryter sikkerheten.

Arkitekturanalyseprosessen bør derfor anvendes også når verifiserte designmønster er i standard bruk.

- **AA3.3: Gjør SSG tilgjengelig som en arkitekturanalyse-ressurs eller mentor**
  For å kunne bygge opp en evne til å utføre arkitekturanalyse utenfor SSG, averterer SSG seg som en ressurs eller mentor for grupper som ber om hjelp til å gjennomføre sin egen designgjennomgang, og leter proaktivt opp nye prosjekter å involvere seg i. SSG besvarer spørsmål rundt arkitekturanalyse i kontortiden, og i noen tilfeller kan de også tilordne noen som kan sitte ved siden av arkitekten hele den tiden det tar å gjennomføre analysen. I det tilfellet det dreier seg om høyrisiko applikasjoner eller produkter , spiller SSG en mer aktiv mentor-rolle.

4.2 Kodegjennomgang (*Code Review*)

Hovedmålet for praksisen "Kodegjennomgang" er kvalitetskontroll. De som utfører kodegjennomgang må sørge for at sikkerhetsfeil oppdages og korrigeres. SSG må sørge for at standarder blir fulgt og at godkjente sikkerhetsmekanismer gjenbrukes.

4.2.1 CR Nivå 1: Bruk manuell og automatisert kodegjennomgang med sentralisert rapportering

Programvaresikkerhetsgruppen (SSG) må gjøre seg tilgjengelig ovenfor andre for å øke bevisstheten om og etterspørsel etter kodegjennomgang. SSG må kjenne sin besøkelsestid, og utføre kodegjennomgang på høyrisiko-applikasjoner når den kan bli involvert i prosessen. SSG må så bruke kunnskapen som tilegnes til å informere virksomheten om typen feil som oppdages. Ledelsen må gjøre kodegjennomgang obligatorisk for alle programvareprosjekter. SSG må tvinge gjennom bruk av sentraliserte rapporteringsverktøy for å samle inn kunnskap om tilbakevendende feil, og kanalisere denne informasjonen inn i strategi og opplæring.

- **CR1.2: La SSG utføre ad hoc kodegjennomgang**
  SSG utfører ad hoc kodegjennomgang for høyrisiko-applikasjoner på en opportunistisk måte. F.eks. kunne SSG følge opp designgjennomgangen for høyrisiko-applikasjoner med en kodegjennomgang. Erstatt ad hoc gjennomgang med en systematisk tilnærming på høyere modenhetsnivåer. SSGs gjennomgang kan involvere bruk av spesifikke verktøy og tjenester, eller den kan være manuell.
- **CR1.4: Bruk automatiserte verktøy sammen med manuell gjennomgang**
  Integrer statisk analyse i kodegjennomgangs-prosessen for å gjøre kodegjennomgang mer effektiv og mer konsistent. Automatiseringen erstatter ikke menneskelige vurderinger, men hjelper til med å definere prosessen og bidrar med sikkerhetsekspertise til evaluatorer som ikke er sikkerhetseksperter. En virksomhet kan bruke en ekstern tjenesteleverandør som en del av en formell kodegjennomgangs-prosess for programvaresikkerhet. Denne tjenesten burde eksplisitt kobles til en større "sikker programvareutviklings-livssyklus" (SSDL) som anvendes som en del av programvareutviklingen, og ikke bare en "huk av sikkerhetsboksen" på veien til utrulling.
- **CR1.5: Gjør kodegjennomgang obligatorisk for alle prosjekter**
  Kodegjennomgang er et obligatorisk sjekkpunkt for alle prosjekter som hører inn under SSG. Manglende kodegjennomgang eller uakseptable resultater vil føre til stopp i utrullingsprosessen. Mens alle prosjekter må utsettes for kodegjennomgang, kan prosessen variere mellom forskjellige typer prosjekter. Gjennomgangen for lavrisiko-prosjekter kan basere seg tungt på automasjon, mens kodegjennomgang for høyrisiko-prosjekter kanskje ikke har noen øvre grense for hvor mye tid som kan brukes. I de fleste tilfeller vil et kodegjennomgang-sjekkpunkt med en minimum akseptabel standard tvinge prosjekter som ikke består til å bli reparert og re-evaluert før de kan leveres.
- **CR1.6: Bruk sentralisert rapportering til å lukke kunnskaps-sirkelen og drive opplæring**
  Feilene som finnes gjennom kodegjennomgang spores i et sentralt arkiv. Dette arkivet gjør det mulig å lage oppsummerende rapporter og trend-rapporter for virksomheten. SSG kan bruke rapportene for å demonstrere framgang og som en pådriver for opplæringspensumet. Informasjon fra kodegjennom-gang kan inkluderes i et sikkerhetssjef-nivå instrumentpanel som omfatter informasjon fra andre

deler av sikkerhetsorganisasjonen. På samme måte kan informasjon fra kodegjennomgang mates inn i et prosjektsporingssystem på tvers av utviklingsorganisasjonen som samler sammen et antall forskjellige programvaresikkerhetsresultater (f.eks. penetreringstester, sikkerhetstester, svart-boks tester, hvit-boks tester, etc.). Ikke glem at individuelle feil utgjør glimrende opplæringseksempler.

4.2.2 CR Nivå 2: Følg opp standarder via kodegjennomgangsprosessen

SSG må påvirke utvikler-oppførsel ved å følge opp kodestandarder vha. automatiserte verktøy og verktøymentorer. SSG må kombinere automatiserte evalueringsteknikker med skreddersydde regler for å effektivt finne problemer.

- **CR2.5: Tilordn verktøymentorer**
  Mentorer er tilgjengelige for å vise utviklere hvordan de skal få det meste ut av kodegjennomgangsverktøyer. Hvis SSG har mest erfaring med verktøyene, kan den bruke kontortid til å hjelpe utviklere med å etablere den riktige konfigurasjonen eller komme i gang med tolkning av resultater. Alternativt kan noen fra SSG jobbe med en utviklergruppe gjennom hele gjennomgangen de utfører. En kan bevege seg fra sentralisert bruk av et verktøy til en modell som er distribuert utover i utviklingsorganisasjonen over tid gjennom bruk av verktøymentorer.
- **CR2.6: Bruk automatiserte verktøy med skreddersydde regler**
  Tilpass statisk analyse til lokale forhold for å forbedre effektivitet og redusere falske positiver. Bruk spesialtilpassede regler for å finne feil som er spesifikke for organisasjonens kodestandarder eller spesiell mellomvare. Skru av tester som ikke er relevante. Den samme gruppen som tilbyr verktøymentoring vil sannsynligvis lede tilpasningen. Skreddersydde regler kan eksplisitt knyttes til korrekt bruk av teknologistakker (positiv korrelasjon), og unngåelse av feil som ellers ofte oppdages i virksomhetens kodebase (negativ korrelasjon).
- **CR2.7: Lag en "topp N" feil-liste (virkelige data foretrukket)**
  SSG vedlikeholder en liste over de viktigste feilene som må elimineres fra virksomhetens kode, og bruker den til å drive frem endringer. Listen bidrar til å fokusere virksomhetens oppmerksomhet mot de feilene som betyr mest. En generisk liste kan lastes ned fra offentlige kilder, men en liste er mye mer verdifull dersom den er spesifikk for virksomheten, og basert på virkelige data hentet fra kodegjennomgang, testing, og virkelige hendelser. SSG kan oppdatere listen regelmessig, og publisere en "mest ettersøkt"-rapport. Noen virksomheter bruker flere verktøy og data fra virkelige kodebaser for å bygge opp "topp N"-lister, uten å begrense seg til en bestemt tjeneste eller verktøy. En potensiell fallgruve med en "topp N"-liste er problemet med å "lete etter nøklene kun under gatelyset". F.eks. vil "OWASP top ten" sjelden reflektere en virksomhets feilprioriteter. Det å kun rangere dagens feiloversikt etter antall tilfeller per feil vil ikke resultere i en tilfredsstillende "topp N"-liste, ettersom disse dataene endrer seg så ofte.

4.2.3 CR Nivå 3: Bygg en automatisert kodegjennomgangsfabrikk med skreddersydde regler

SSG må bygge opp en evne til å finne og luke ut spesifikke feil i hele kodebasen.

- **CR3.2: Bygg en fabrikk**
  Kombiner evalueringsresultater slik at resultater fra flere analyseteknikker mates inn i en enhetlig rapporterings- og forbedringsprosess. SSG kan f.eks. skrive kommandofiler som kaller flere deteksjonsprogrammer automatisk, og kombinerer resultatene til et format som kan konsumeres av en enkelt nedstrøms gjennomgang og rapporteringsløsning. Analysemotorer kan kombinere statisk og dynamisk analyse. Den kinkige biten av denne aktiviteten er å normalisere sårbarhetsinformasjon fra forskjellige kilder som bruker forskjellig terminologi. I noen tilfeller kan en CWE-lik[8] tilnærming

[8] https://cwe.mitre.org/

bidra med nomenklatur. Kombinasjon av flere kilder kan bidra til bedre informasjonsgrunnlag for risikohåndteringsavgjørelser.

- **CR3.3: Bygg en evne til å utrydde spesifikke feil fra hele kodebasen**
  Når en ny type feil oppdages skriver SSG regler for å finne den, og bruker reglene til å identifisere alle tilfeller av den nyoppdagede feilen i hele kodebasen. Det er mulig å utrydde feilen fullstendig uten å vente på at hvert prosjekt skal nå kodegjennomgangsfasen i livssyklusen. En virksomhet med kun en håndfull programvareapplikasjoner vil kunne håndtere denne aktiviteten enklere enn virksomheter som utvikler et stort antall applikasjoner.
- **CR3.4: Automatiser deteksjon av ondartet kode**
  Automatisert kodegjennomgang brukes for å identifisere farlig kode som er skrevet av ondsinnede interne utviklere eller konsulenter. Eksempler på ondartet kode som man kan se etter inkluderer: bakdører, logiske bomber, tidsinnstilte bomber, skjulte kommunikasjonskanaler, obfuskert programlogikk og dynamisk kodeinjeksjon. Selv om automatiserte verktøy med "fabrikkinnstillinger" kan finne noen generelle konstruksjoner som ser ondartede ut, vil det fort bli nødvendig med skreddersydde regler for statiske analyseverktøy for å kodifisere godkjente og uakseptable kodemønstre i virksomhetens kodebase. Manuell kodegjennomgang for ondartet kode er en god start, men er ikke tilstrekkelig for å gjennomføre denne aktiviteten.
- **CR3.5: Tving gjennom og følg opp kodestandarder**
  Et brudd på virksomhetens sikre kodestandarder er tilstrekkelig grunn til å forkaste et stykke kode. Kodegjennomgang er objektiv – den kan ikke reduseres til en diskusjon om hvorvidt dårlig kode kan utnyttes av angripere. Den delen av standarden som følges opp kan starte med noe så enkelt som en liste med forbudte funksjoner. I noen tilfeller publiseres kodestandarder som utvikler-retningslinjer spesifikke til teknologistakker (f.eks. retningslinjer for C++ eller Spring) som så følges opp i kodegjennomgangsprosessen eller direkte i det integrerte programmeringsmiljøet (IDE). Bemerk at retningslinjer kan være positive ("gjør det på denne måten") men også negative ("ikke bruk dette APIet).

4.3 Sikkerhetstesting (*Security Testing*)

Hovedmålet for praksisen Sikkerhetstesting er kvalitetskontroll (*quality assurance* – QA) som gjøres i løpet av utviklingssyklusen. De som utfører sikkerhetstesting må sørge for at sikkerhetsfeil oppdages og korrigeres. Programvaresikkerhetsgruppen (SSG) må sørge for at standarder blir fulgt og at godkjente sikkerhetsmekanismer gjenbrukes.

4.3.1 ST Nivå 1: Forbedre QA utover det funksjonelle perspektivet

Kvalitetskontrollen (QA) må inkludere funksjonell kant- og grenseverdi-testing i testregimet. Testsamlinger (*test suites*) må også omfatte funksjonell sikkerhetstesting.

- **ST1.1: Sørg for at QA støtter testing av grenseverdier og tilstander**
  QA-gruppen går utover ren funksjonell testing til å utføre grunnleggende angrepstester. De undersøker ytterkanter og enkle grensetilfeller. Ingen angrepsferdigheter kreves. Når QA forstår verdien i å komme videre forbi standard funksjonell testing med akseptable innverdier, begynner de langsomt å bevege seg mot å "tenke som en angriper". En diskusjon rundt testing i grenseland fører naturlig til tanken om at en angriper kommer til å føle seg fram i ytterkantene med vilje. Hva skjer når du taster inn feil passord igjen og igjen?
- **ST1.3: La sikkerhetsmekanismer og sikkerhetskrav drive testene**
  Testere blinker ut deklarative sikkerhetsmekanismer utledet fra sikkerhetskrav og sikkerhetsegenskaper, og tester disse funksjonelt. En tester kan f.eks. prøve å aksessere administratorfunksjonalitet som en upriviligert bruker, eller verifisere at en brukerkonto låses etter et antall mislykkede innloggingsforsøk. For det meste kan sikkerhetsmekanismer testes på samme måte som andre programvaremekanismer. Sikkerhetsmekanismer basert på krav slik som låsing av brukerkonto,

transaksjonsbegrensninger, rettigheter, og så videre testes også. Selvfølgelig er ikke programvare-sikkerhet sikkerhetsprogramvare, men det er lett å komme i gang hvis man starter med mekanismer.

4.3.2 ST Nivå 2: Integrér angriperens perspektiv i testplaner

QA må integrere "black-box" sikkerhetstestverktøy i prosessen. QA må bygge testsamlinger (*test suites*) for funksjonelle sikkerhetsegenskaper. SSG må dele sin sikkerhetskunnskap og testresultater med QA.

- **ST2.1: Integrer "svart-boks" sikkerhetsverktøy i QA-prosessen**
  Virksomheten bruker et eller flere "svart-boks" (*black box*) sikkerhetstestverktøy som en del av kvalitetssikringsprosessen. Verktøyene er verdifulle fordi de bygger inn angriperens perspektiv, om enn på en generisk måte. Det finnes forskjellige kommersielle verktøy som er tilgjengelige for web-applikasjoner, og det finnes fuzzing-rammeverk[9] for de fleste nettverksprotokoller. I enkelte situasjoner vil andre grupper samarbeide med SSG for å anvende verktøyene. For eksempel kan et testlag kjøre verktøyet, men gå til SSG for å få hjelp til å tolke resultatene. Uavhengig av hvem som kjører "svart-boks"-verktøyet bør testingen integreres skikkelig i QA-syklusen til SSDL.
- **ST2.4: Del sikkerhetsresultater med QA**
  SSG deler rutinemessig resultater fra sikkerhetsgjennomganger med QA-avdelingen. Etter hvert lærer QA-ingeniører seg sikkerhetstankegangen. Bruk av sikkerhetsresultater for å informere og utvikle spesifikke testmønstre kan være en kraftig mekanisme som fører til bedre sikkerhetstesting. Denne aktiviteten tjener på en svært teknisk, ingeniør-fokusert QA-funksjon.
- **ST2.5: Inkludér sikkerhetstester i QA-automatisering**
  Sikkerhetstester kjører side om side med funksjonelle tester som en del av automatisert regresjons-testing. Det samme automatiserte rammeverket inneholder begge deler. Sikkerhetstesting er en del av rutinene. Sikkerhetstester kan drives med utgangspunkt i misbrukseksempler (*abuse cases)* identifisert tidligere i livssyklusen (se avsnitt 3.1.2), eller så kan sikkerhetstester utledes fra kreative tilpasninger av funksjonelle tester.
- **ST2.6: Utfør fuzzing spesialtilpasset til applikasjonens API**
  Testingeniører skreddersyr et fuzzing-rammeverk til virksomhetens APIer. De kan begynne fra grunnen av eller bruke en eksisterende fuzzing-verktøykasse, men skreddersømmen går utover å lage tilpassede protokollbeskrivelser eller filformat-maler. Fuzzing-rammeverket har en innebygd forståelse av applikasjonsgrensesnittene det gjør kall mot. Automatiserte test-rammeverk utviklet spesifikt for bestemte applikasjoner kan representere gode steder å integrere fuzzing.

4.3.3 ST Nivå 3: Levér risiko-basert sikkerhetstesting

QA må inkludere sikkerhetstesting i automatiserte samlinger av regresjonstester (*regression suites*), dvs. tester som utføres ved endringer i koden. SSG må sørge for denne sikkerhetstestingen, og hvor dypt man skal gå må veiledes av kunnskap om kodebasen og den assosierte risikoen. I tillegg må man foreta aggressiv testing som simulerer angriperens perspektiv.

- **ST3.3: La risikoanalyse-resultater drive testene**
  Testere bruker resultater fra arkitekturanalyse for å stake ut retningen på arbeidet. For eksempel, hvis arkitekturanalysen konkluderer at "sikkerheten i systemet avhenger av at transaksjonene er atomiske og ikke kan avbrytes/forstyrres halvveis", så vil avbrutte transaksjoner bli et hovedmål i sikkerhets-testingen. Sikkerhetstester som disse kan utvikles i henhold til risikoprofil – høyrisiko feil først.
- **ST3.4: Utnytt dekningsanalyse**
  Testere måler dekningen av deres sikkerhetstester for å identifisere kode som ikke aktiveres. Kode-

[9] Fuzzing er en automatisert testing av grensesnitt der store mengder (mer eller mindre tilfeldige) input-data mates inn i en applikasjon for å se om uønsket oppførsel (kræsj, etc.) kan framprovoseres. Se http://pages.cs.wisc.edu/~bart/fuzz/ for mer informasjon.

dekning driver økt sikkerhetstestings-dybde. Standard "svart-boks" sikkerhetstestingsverktøy oppnår svært lav kodedekning, og etterlater hoveddelen av programvaren som skal testes uutforsket. Ikke la dette skje med dine tester. Bruk av standard metrikker for dekning som funksjonsdekning, linje-dekning, eller dekning av flere tilstander er i orden.

- **ST3.5: Begynn å bygge og bruke angrepsorienterte sikkerhetstester (*abuse cases*)**
  Testere begynner å inkorporere testtilfeller basert på misbrukseksempler (*abuse cases)* fra SSG. Testere beveger seg forbi å bare verifisere funksjonalitet, og antar angriperens perspektiv. F.eks. kan testere periodisk prøve å gjenskape hendelser fra virksomhetens historie. Misbrukseksempler (a*buse cases* og *misuse cases*[10]*)* basert på angriperes perspektiv kan også drives fra sikkerhetsinstrukser, angrepsinformasjon og retningslinjer. Dette tar virksomheten opp på neste nivå, fra å teste mekanismer til å prøve å bryte programvaren som testes.

5 Utrulling (*Deployment*)

Domenet **Utrulling** omfatter Penetreringstesting; Programvaremiljø; og Konfigurasjonsstyring og sårbarhetsstyring.

5.1 Penetreringstesting (*Penetration Testing*)

SINTEF utførte i 2015 på oppdrag fra Difi en modenhetsstudie av programvaresikkerhet i offentlige virksomheter [5][8], og i denne studien pekte penetreringstesting seg ut som en aktivitet som generelt sett ikke gjøres som en del av utviklingen – dette er overraskende når man tenker på hvor mange verktøy som er fritt tilgjengelige. Noen kunne hevde at den jevne utvikler ikke har tilstrekkelig kompetanse til å drive penetreringstesting, men vårt svar er at da er det på tide at de lærer seg noen nye triks! BSIMMs "far" Gary McGraw har uttalt [9] at utviklere ikke skal penetreringsteste sin egen kode, men det er kanskje vanskelig å unngå i en gjennomsnittlig norsk virksomhet hvor antallet ansatte tilsier at alle utviklere må gjøre mange forskjellige oppgaver.

Hovedmålet med praksisen "Penetreringstesting" er kvalitetskontroll av programvare som er i ferd med å bli rullet ut. De som utfører penetreringstesting må sørge for at sikkerhetsdefekter oppdages og korrigeres. Programvaresikkerhetsgruppen (SSG) må sørge for at standarder blir fulgt, og at godkjente sikkerhets-mekanismer gjenbrukes.

5.1.1 PT Nivå 1: Oppdater kode etter penetreringstestingsresultater

Ledere og SSG må initiere penetreringstestingsprosessen, med interne eller eksterne ressurser. Ledere og SSG må sørge for at oppdagede feil adresseres, og at alle er gjort kjent med fremdriften.

- **PT1.1: Bruk eksterne penetreringstestere for å finne problemer**
  Mange virksomheter er ikke villige til å adressere programvaresikkerhet før det er ugjendrivelig bevis for at virksomheten ikke på noe magisk vis er immun for problemet. Hvis sikkerhet ikke har vært prioritert, demonstrerer eksterne penetreringtestere at virksomhetens kode trenger hjelp. Penetreringstestere kan hentes inn for å bryte en høyprofil applikasjon for å markere poenget. Over tid flyttes fokuset for penetreringstesting fra "jeg fortalte deg at koden din er råtten" til en røyktest og sjekk av at ting ikke er helt på jordet som utføres før leveranse. Eksterne penetreringstestere bidrar med et nytt sett øyne på problemet.
- **PT1.2: Mat resultater tilbake til feilhåndterings- og opprettings-systemet**
  Resultater fra peneteringstesting mates tilbake til utviklerne via et etablert system for å håndtere feil og oppretting, og utviklerne responderer via deres feilhåndterings- og utgivelsesprosess. Øvelsen demonstrerer virksomhetens evne til å forbedre sikkerhetstilstanden. Mange virksomheter begynner

[10] *Abuse cases* og *misuse cases* er nesten det samme, men sistnevnte er formalisert som en del av UML, og har en egen grafisk notasjon som brukes i såkalte *misuse case diagrams*.

å fremheve den kritiske viktigheten av å ikke bare identifisere, men også fikse sikkerhetsproblemer. En måte å sikre oppmerksomhet er å legge til et sikkerhetsflagg i feilsporing og feilhåndteringssystemet. Evolverende DevOps og integrerte arbeidsgruppestrukturer eliminerer ikke behovet for formaliserte feilhåndteringssystemer.

- **PT1.3: Bruk penetreringstestingsverktøy internt**
  Virksomheten oppretter en intern penetreringstestings-kapabilitet som gjør bruk av verktøy. Denne kapabiliteten kan være en del av SSG, eller del av en spesielt opplært gruppe et annet sted i virksomheten. Verktøyene forbedrer effektiviteten og repeterbarheten av testprosessen. Verktøyene kan omfatte hyllevare-produkter, standard nettverkspenetreringsverktøy som forstår applikasjonslaget, og håndskrevne kommandofiler.

5.1.2 PT Nivå 2: Planlegg jevnlig penetreringstesting utført av informerte penetreringstestere

SSG må legge til rette for en penetreringstestingsaktivitet som periodisk utføres på alle applikasjoner. SSG må dele sin sikkerhetskunnskap og testresultater med alle penetreringstestere.

- **PT2.2: Utstyr penetreringstestere med all tilgjengelig informasjon**
  Penetreringstestere, enten de er interne eller eksterne, bruker all tilgjengelig informasjon om målet. Penetreringstestere kan utføre dypere analyser og finne mer interessante problemer når de har tilgang på kildekode, designdokumenter, og resultater fra arkitekturanalyse og kodegjennomgang. Gi penetreringstesterne alle artefakter du har laget i løpet av din SSDL. Hvis penetreringstesteren ikke ber om koden trenger du en ny penetreringstester.
- **PT2.3: Planlegg periodiske penetreringstester for å dekke opp alle applikasjoner**
  Test alle applikasjonene som faller inn under SSG regelmessig, dvs. etter en bestemt tidsplan (som kan knyttes til kalenderen eller utgivelsessyklusen). Testingen tjener som en sjekk på at ting ikke er helt på jordet (*sanity check*), og hjelper til med å sikre at gårsdagens kode ikke er sårbar for dagens angrep. Høyprofil-applikasjoner vil kanskje penetreringstestes en gang i året. Ett viktig aspekt ved periodisk testing er å forsikre seg om at problemene som identifiseres i en penetreringstest faktisk er ordnet, og at de ikke lurer seg tilbake i kodebasen.

5.1.3 PT Nivå 3: Utfør dypdykk-penetreringstesting

Ledere må sørge for at virksomhetens penetreringstestingskunnskap holder følge med fremskrittene som angriperne gjør. SSG må dra nytte av organisasjonsmessig kunnskap for å skreddersy penetreringstestingsverktøy.

- **PT3.1: Bruk eksterne penetreringstestere til å gjøre dypdykk-analyse**
  Virksomheten bruker eksterne penetreringstestere for å gjøre dypdykk-analyse for kritiske prosjekter og for å introdusere friske tanker i SSG. Disse testerne er eksperter og spesialister. De løfter virksomheten opp med siste nytt fra angriperens perspektiv, og de har en merittliste som demonstrerer erfaring med å knekke samme type programvare som testes. Dyktige penetreringstestere vil alltid knekke systemet. Spørsmålet er om de demonstrerer nye typer tenkning rundt angrep som kan være nyttig når man skal designe, implementere og herde nye systemer. Det å lage nye typer angrep fra trusselinformasjon og misbrukseksempler (*abuse cases)* forhindrer sjekkliste-baserte tilnærminger som bare ser etter kjente typer problemer (se avsnitt 3.1.2).
- **PT3.2: La SSG skreddersy penetreringstestingsverktøy og kommandofiler**
  SSG enten lager penetreringstestingsverktøy fra grunnen av, eller tilpasser offentlig tilgjengelige verktøy slik at de kan angripe virksomhetens systemer på en mer omfattende og effektiv måte. Verktøy forbedrer effektiviteten til penetreringstestingsprosessen uten å ofre dybden av problemer SSG kan identifisere. Verktøy som kan tilpasses er alltid å foretrekke over generelle verktøy.

5.2 Programvaremiljø (*Software Environment*)

I vår modenhetsstudie av programvaresikkerhet i offentlige virksomheter [5] fant vi at de aller fleste hadde god tiltro til egen modenhet på når det gjelder Programvaremiljø. Dette er ikke overraskende, ettersom det stort sett dreier seg om tradisjonelle mekanismer for sikring av datamaskiner og nettverk, noe som i langt større grad er en del av vanlig praksis enn det programvaresikkerhet er.

Hovedmålet for praksisen Programvaremiljø er endringsledelse. De som er ansvarlige for programvare-miljøet må sørge for at de er i stand til å gjøre autoriserte endringer, og avdekke uautoriserte endringer. Ledere må sørge for at virksomhetens retningslinjer blir fulgt.

5.2.1 SE Nivå 1: Sørg for at programvaremiljøet støtter programvaresikkerhet

Driftsgruppen sørger for at de nødvendige sikkerhetsmekanismene for datamaskiner og nettverk fungerer, og overvåker programvare proaktivt, inkludert input til applikasjoner.

- **SE1.1: Overvåk input til applikasjonene**
  Virksomheten overvåker input til programvaren den kjører for å oppdage angrep. For web-kode kan en *Web Application Firewall* (WAF) gjøre jobben. Programvaresikkerhetsgruppen (SSG) kan ha ansvaret for røkt av dette systemet. Det er ikke en del av denne aktiviteten å respondere på angrep. Tannløse WAFer som lager loggfiler kan være nyttige dersom noen gjennomgår loggene periodisk. På den annen side, en WAF som ikke overvåkes lager ingen lyd når en applikasjon faller i skogen.
- **SE1.2: Sørg for at grunnleggende system- og nettverkssikkerhetsmekanismer er på plass**
  Virksomheten tilbyr et solid fundament for programvare ved å sørge for at grunnleggende system- og nettverkssikkerhetsmekanismer er på plass. Det er vanlig at driftssikkerhetsgruppen har ansvar for oppgaver som patching av operativsystemer og vedlikehold av brannmurer. Å fokusere på programvaresikkerhet før du har nettverkssikkerheten på plass blir som å ta på seg buksene før man tar på seg undertøy.

5.2.2 SE Nivå 2: Bruk publiserte installasjonsveiledninger og kodesignering

SSG må sørge for at programvareutviklingsprosesser beskytter integriteten til koden. SSG må sørge for at veiledninger for installasjon og vedlikehold av applikasjoner er utarbeidet for bruk av driftsgruppen.

- **SE2.2: Publisér installasjonsveiledninger**
  Livssyklusen for sikker programvareutikling (SSDL) krever at det opprettes en installasjons-veiledning for å hjelpe operatører til å installere og konfigurere systemet på en korrekt og sikker måte. Hvis spesielle trinn er nødvendige for å oppnå en sikker installasjon, er disse trinnene skissert i installasjonsveiledningen. Veiledningen bør omfatte diskusjon av hyllevare-komponenter. I noen tilfeller er installasjonsveiledninger distribuert til kunder som kjøper programvaren. Evolverende DevOps og integrerte gruppestrukturer eliminerer ikke behovet for skrevne retningslinjer. Selvfølgelig er "sikker med standardinstillinger" alltid den beste måte å gjøre det på.
- **SE2.4: Bruk kodesignering**
  Virksomheten bruker kodesignering for programvare som gis ut på tvers av tillitsgrenser (dvs. til andre organisasjoner). Kodesignering er spesielt nyttig for å beskytte integriteten til programvare som forlater virksomhetens kontroll, slik som hyllevare-applikasjoner eller tykke klienter. Det faktum at enkelte mobile plattformer krever at applikasjonskode er signert er ingen indikasjon på institusjonell bruk av kodesignering.

5.2.3 SE Nivå 3: Beskytt klient-side kode og overvåk oppførselen til programvare aktivt

SSG må sørge for at all kode som forlater virksomheten er beskyttet. Driftsgruppen må overvåke oppførselen til programvaren.

- **SE3.2: Bruk kodebeskyttelse**
  For å beskytte immaterielle rettigheter og gjøre det vanskeligere å lage *exploits*, oppretter virksomheten barriærer mot omvendt utvikling. Obfuskeringsteknikker kan anvendes som en del av produksjonsbygging og utgivelsesprosessen. Ved å anvende platform-spesifikke mekanismer som *Data Execution Prevention* (DEP), *Safe Structured Error Handling* (SafeSEH), og *Address Space Layout Randomization* (ASLR) kan man gjøre utvikling av *exploits* vanskeligere.
- **SE3.3: Anvend overvåking av applikasjonsoppførsel og diagnostikk**
  Virksomheten overvåker oppførselen til utrullet programvare og ser etter tegn på at den ikke oppfører seg som den skal eller andre symptomer på angrep. Denne aktiviteten går utover maskin- og nettverksovervåkning ved å se etter problemer som er spesifikke for programvaren, som f,eks, indikasjoner på svindel. Inntrengningsdeteksjons- og anomalideteksjonsprogramvare på applikasjonslaget kan fokusere på en applikasjons interaksjon med operativsystemet (gjennom systemkall), eller på applikasjonens interaksjon med de forskjellige typer data som den konsumerer, oppretter, eller manipulerer.
- **SE 3.4: Bruk applikasjonscontainere**
  Virksomheten bruker applikasjonscontainere for å understøtte sine programvaresikkerhetsmål. De viktigste driverne for å bruke containere er forenklet utrulling, tettere kobling mellom applikasjoner og avhengigheter, og isolasjon uten overheaden som er involvert ved å rulle ut et fullt operativsystem på en virtuell maskin. Containere tilbyr et hensiktsmessig sted hvor sikkerhetsmekanismer kan anvendes og oppdateres på en konsistent måte. Containere som brukes i utvikling eller testmiljø uten referanse til sikkerhet teller ikke.
- **SE 3.5: Bruk orkestrering for containere og virtualiserte miljøer**
  Virksomheten bruker automatisering for å skalere utrulling av containere og virtuelle maskiner på en disiplinert måte. Orkestreringsprosesser drar fordel av innebygde og ettermonterte sikkerhetsmekanismer for å sørge for at hver utrullede container og virtuelle maskin tilfredsstiller forhåndsbestemte sikkerhetskrav. Ved å angi sikkerhetsoppførsel på en aggregert måte tillater man hurtige endringer når behovet oppstår. Orkestreringsplattformer er selvfølgelig også programvare som på sin side krever sikkerhetsoppdateringer og konfigurasjon. Hvis du bruker Kubernetes, sørg for at du oppdaterer Kubernetes.
- **SE 3.6: Forbedre applikasjonsoversikt med operasjons-stykkliste**
  En liste av applikasjoner og deres plassering i produksjonsmiljøer er avgjørende informasjon for enhver veldrevet virksomhet (se [CMVM 2.3]). I tillegg vil et manifest som detaljerer komponenter, avhengigheter, konfigurasjoner, eksterne tjenester, osv. for all produksjonsprogramvare gjøre virksomheter i stand til å sikre alle tingene, dvs. å reagere med smidighet ettersom angripere og angrep utvikler seg, regulatoriske krav endrer seg, og antallet enheter som må oppdateres vokser til nye høyder. Oversikt over alle komponenter som finnes i kjørende programvare - enten de er i private datasentre, i skyer, eller solgt som bokser - gjør det mulig å respondere innen rimelig tid når uheldige hendelser inntreffer.
- **SE3.7: Sørg for grunnleggende skysikkerhet**
  Du har selvfølgelig stålkontroll på grunnleggende sikkerhetsmekanismer i datamaskiner og nettverk (se [SE 1.2]), men noen må sørge for at grunnleggende sikkerhetskrav oppfylles i skyen også. Etter som verden blir mer og mer "programvaredefinert", må du eksplisitt implementere sikkerhetsmekanismer og kontrollmekanismer (noen av disse kan være innebygd) i skyen minst like godt som de som bygges med kabler og fysisk maskinvare. Ingenting er så automatisk som det kan virke.

5.3 Konfigurasjonsstyring og sårbarhetsstyring (*Configuration Management and Vulnerability Management*)

Det overordnede målet for praksisen "konfigurasjons- og sårbarhetsstyring" er *endringshåndtering* – holde styr på autoriserte endringer, og avdekke og/eller forhindre uautoriserte endringer og aktiviteter. Applikasjonseiere må sørge for at virksomhetens retningslinjer blir fulgt.

5.3.1 CMVM Nivå 1: Få det som observeres i driften til å drive utviklingen

SSG må støtte opp om hendelseshåndtering. SSG må bruke data fra driften til å foreslå endringer i SSDL og utvikleroppførsel.

- **CMVM1.1**: **Opprett eller opprett kontakt med hendelseshåndtering**
  SSG er forberedt til å respondere på sikkerhetshendelser. Hvis organisasjonen har en hendelses-håndteringsgruppe (CSIRT[11] eller lignende), sørg for å opprette kontakt mellom denne og utviklerne. Hvis ikke, opprett en egen gruppe (med utgangspunkt i SSG) for å håndtere hendelser som har innvirkning på programvare. Regelmessige møter mellom SSG og CSIRT kan bidra til at informasjonen flyter begge veier. I mange tilfeller har programvaresikkerhetsinitiativer oppstått fra CSIRTer som innså at programvaresårbarheter var årsaken til mange av deres problemer.
- **CMVM1.2**: **Identifiser programvarefeil som oppdages ved overvåkning av driften, og mat disse tilbake til utviklingsavdelingen**
  IDS og andre sikkerhetsverktøy kan oppdage uregelmessigheter som har sammenheng med programvare som er utviklet internt, f.eks. angrep som utnytter spesifikke sårbarheter. Det er viktig at slik informasjon flyter tilbake til utviklerne, slik at feilene kan rettes og hullene tettes. På lengre sikt ønsker man å bruke disse dataene til å endre oppførselen til utviklerne. Innholdet i produksjons-logger kan være avslørende (eller kan avdekke behov for forbedret logging). I noen tilfeller kan det fungere å opprette en mulighet til å legge inn hendelsesinformasjon direkte i et eksisterende feil-håndteringssystem (ofte med en egen mulighet til å flagge sikkerhetsfeil spesielt). Poenget er å lukke informasjonssløyfen og sørge for at sikkerhetsfeil blir fikset. I beste fall kan SSDL-prosesser bli forbedret basert på data observert i driften.

5.3.2 CMVM Nivå 2: Sørg for at krisehåndtering er tilgjengelig når applikasjoner er under angrep

Ledere og SSG må understøtte krisehåndtering ved pågående angrep på applikasjoner. Ledere og SSG må opprette og vedlikeholde en oversikt over all kode. SSG bruker data fra driften til å styre evolusjon av SSDL og utvikleroppførsel.

- **CMVM2.1**: **Ha et system for å gjøre hurtige krise-endringer i koden når en applikasjon er under angrep**
  Virksomheten er i stand til å gjøre hurtige endringer i koden når en applikasjon er under angrep. En hurtigrespons-gruppe samarbeider med applikasjonseierne og SSG for å studere koden og angrepet, finne en løsning, og rulle en oppdatering ut i produksjon. Ofte utgjøres respons-gruppen av selve utviklingsgruppen. Brannøvelser og ad-hoc-løsninger teller ikke; en veldefinert prosess kreves for denne aktiviteten.
- **CMVM2.2**: **Hold styr på programvarefeil som er funnet i driften gjennom hele opprettingsprosessen**
  I tillegg til at feil som finnes mates tilbake til utviklerne, så legges de inn i feilhåndteringssystemet og spores gjennom hele utbedringsprosessen. Dette kan ivaretas i form av en toveis forbindelse mellom de som finner feil og de som fikser feil. Sørg for at sløyfen er fullstendig lukket! Muligheten til å sette et sikkerhetsflagg i feilhåndteringssystemet kan hjelpe til med sporingen.
- **CMVM2.3**: **Opprett en katalog over applikasjoner for driften**
  Organisasjonen må ha et kart over hvor programvare er rullet ut – hvis en kodebit må endres, kan de som driver systemene identifisere alle stedene hvor endringen må installeres. Noen ganger er felleskomponenter som deles mellom flere prosjekter marker spesielt, slik at hvis en feil oppstår i en applikasjon så kan andre applikasjoner som deler den samme komponenten fikses i samme slengen.

[11] Computer Security Incident Response Team – se http://infosec.sintef.no/informasjonssikkerhet/2014/06/god-praksis-for-hendelseshandtering/

5.3.3 CMVM Nivå 3: Lag en tett tilbakekoblingssløyfe mellom drift og utvikling

SSG må sørge for at SSDL både adresserer kodefeil som oppdages i driften, og inkluderer forbedringer som eliminerer assosierte rotårsaker.

- **CMVM3.1: Fiks alle programvarefeil som oppdages i driften**
  Virksomheten fikser alle feil som oppdages, ikke bare de som har generert feilrapporter. Dette medfører behov for muligheten til å gjennomgå hele kodebasen når nye typer feil ser dagens lys (se CR3.3 i avsnitt 4.2.3). En måte å tilnærme seg dette på er å lage et regelsett som generaliserer en feil til noe som kan søkes etter med et automatisert kodegjennomgangsverktøy.
- **CMVM3.2: Utvid organisasjonens SSDL til å forhindre feil som oppdages i driften**
  Det er selvfølgelig bra å være i stand til å oppdage feil etter at programvare er satt i drift, men de er enda bedre å unngå å introdusere dem i det hele tatt. Erfaringer fra driften leder til endringer i SSDL. SSDL er styrket for å unngå at feil som er funnet i driften re-introduseres i senere iterasjoner. For å gjøre dette systematisk kan "post mortem"-prosessen etter en hendelse inkludere et eget steg om "tilbakemelding til SSDL". Dette fungerer best når en rotårsaksanalyse identifiserer hvor i programvareutviklingsprosessen en feil var introdusert eller glapp forbi uten å bli stoppet. En ad-hoc-prosess er ikke tilstrekkelig for denne aktiviteten.
- **CMVM3.3**: **Simulér programvarekriser**
  SSG kjører beredskapsøvelser hvor man tester evnen til å identifisere og håndtere spesifikke trusler, evt. hvor man antar et en kritisk komponent er kompromittert, og evaluerer organisasjonens evne til å håndtere situasjonen. Når simuleringer modellerer vellykkede angrep er det viktig å se på tin det tar for å rydde opp. I alle tilfeller må simuleringer fokusere på sikkerhetsrelevante programvarefeil, og ikke naturkatastrofer eller andre hendelser som ofte dekkes av tradisjonelle beredskapsøvelser. Hvis datasenteret er i ferd med å brenne opp, er det rimelig å anta at SSG ikke kommer til å være i første linje i innsatsstyrken.
- **CMVM3.4**: **Skuddpremie på programvarefeil**
  Virksomheten inviterer til innsending av sårbarhetsrapporter fra eksterne aktører (f.eks. forskere), og betaler en skuddpremie for hver verifiserte og aksepterte sårbarhet de mottar. Utbetalinger vil typisk være på en glidende skala i koblet til flere faktorer som f.eks. sårbarhetstype (mulighet til å kjøre kode utenfra kan være verdt $10000, CSRF kanskje bare $750), hvor lett sårbarheten er å utnytte (*exploits* som virker gir høyere utbetalinger), eller spesifikke tjenester og programvareversjoner (tjenester som er i vid bruk, eller spesielt kritiske tjenester gir høyere utbetalinger). Ad-hoc-aktiviteter eller aktiviteter av kortere varighet som f.eks. *capture-the-flag*-konkurranser er ikke tilstrekkelig for denne aktiviteten.

6 Referanser

[1] Gary McGraw, Sammy Migues og Jacob West: "The Building Security In Maturity Model", versjon 9, oktober 2018, http://www.bsimm.com

[2] Robert Pirsig: "Zen and the Art of Motorcycle Maintenance", Perennial; Reprint edition (April 1, 1999)

[3] Martin Gilje Jaatun: "Hunting for Aardvarks: Can Software Security be Measured?", Multidisciplinary Research and Practice for Information Systems, LNCS Vol. 7465, pp 85-92 http://sislab.no/ijsse/measure.pdf

[4] Gary McGraw: "Software [In]security: You Really Need a Software Security Group", InformIT, Dec 21, 2009, http://www.informit.com/articles/article.aspx?p=1434903

[5] http://www.swsec.com/resources/touchpoints/

[6] Gary McGraw, Tweet https://twitter.com/cigitalgem/status/681875316655144961 , 29. desember 2015

[7] Martin Gilje Jaatun, Inger Anne Tøndel, Daniela Soares Cruzes: "Modenhetskartlegging av programvaresikkerhet i offentlige virksomheter", SINTEF Rapport A26860, 2015

[8] Martin Gilje Jaatun , Daniela S. Cruzes, Karin Bernsmed, Inger Anne Tøndel, Lillian Røstad: "Software Security Maturity in Public Organisations", in *Information Security*, LNCS Volume 9290, pp 120-138, 27 August 2015

[9] Gary McGraw, Tweet https://twitter.com/cigitalgem/status/657771609860743168 , 24. oktober 2015

[10] PCI DSS https://www.pcisecuritystandards.org/security_standards/

Teknologi for et bedre samfunn

www.sintef.no