

IN2000

Software Engineering og Prosjektarbeid

Vår 2026

Sikker systemutvikling

Audun Jøsang



Oversikt

- Sikker systemutvikling
 - *Security Development Lifecycle*
 - Trusselmodellering og STRIDE
 - Applikasjonssikkerhet og OWASP Top 10
 - Sikkerhet i skyen
 - Innebygd informasjonssikkerhet
 - Sikker vite-koding
 - Sikkerhet i Android

Innebygd informasjonssikkerhet og personvern

- «Innebygd» informasjonssikkerhet og personvern betyr at det tas eksplisitt hensyn til informasjonssikkerhet og personvern i hele livssyklusen til programvare og applikasjoner.
- Et viktig mål er å finne og redusere sårbarheter tidlig i utviklingsprosessen slik at det blir færre hendelser og sårbarheter å håndtere under drift.
- Microsoft så dette behovet tidlig og har vært ledende her og utviklet *Microsoft Security Development Lifecycle (SDL)*
 - Denne er nå en integrert del av programvareutviklingsprosessen hos Microsoft (og andre)
- Vi kan beskrive livssyklusen som en prosess av 7 faser:

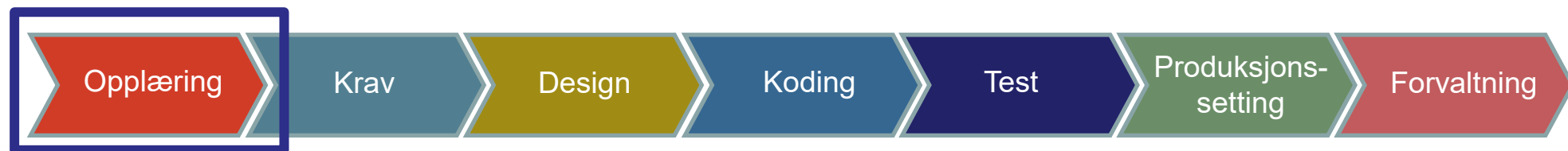


Fase 1: Opplæring

Alle som deltar i utvikling og drift av digitale tjenester og applikasjoner skal ha basiskunnskap og forståelse for

- Informasjonssikkerhet og personvern
- Risikovurderinger
- Trusselmodellering

Dette er egentlig en selvfølge. Det ville være uforsvarlig å utdanne bygningsarkitekter og ingeniører uten å gi dem kunnskap om branntrygghet, fordi arkitekter og ingeniører da ville bygget brannfeller inn i våre bygninger. På samme måte er det uforsvarlig å utdanne informatikere og dataingeniører uten obligatoriske kurs om informasjonssikkerhet, fordi de ferdigutdannede da nødvendigvis ville bygget en sårbar IKT-infrastruktur.



Politikk om IT-sikkerhet i utdanningen

2014

Kunnskapsdepartementet

SAK (Samarbeid, Arbeidsdeling og Konsentrasjon)

IT-sikkerhet bør bare undervises ved ett eneste lærested !



2019

Kunnskapsdepartementet

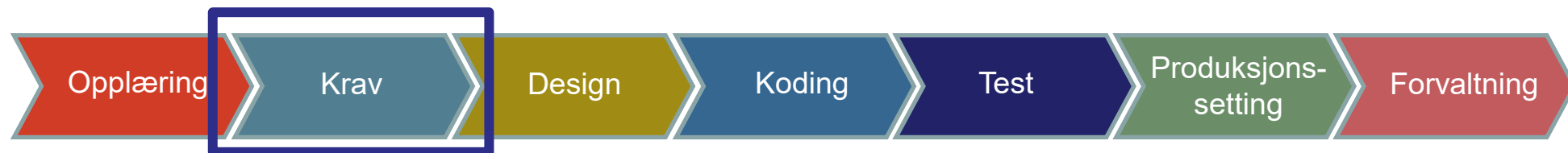
Nasjonal strategi for digital sikkerhetskompetanse 2019

Digital sikkerhet skal undervises overalt !



Fase 2: Krav om informasjonssikkerhet og personvern

- Kilder til krav om informasjonssikkerhet og personvern:
 1. Krav som følger av god praksis for adekvat sikkerhet i applikasjoner og forretningsprosesser.
 2. Krav om å begrense sikkerhetsrisiko til et akseptabelt forsvarlig nivå.
 3. Juridiske lovbestemte, regulatoriske og kontraktsmessige krav til informasjonssikkerhet og personvern.
- Sikkerhetskrav må kontinuerlig oppdateres for å gjenspeile endringer i nødvendig funksjonalitet, trusselslandskap, lover, forskrifter, reguleringer,...
- Bør gjøres tidlig i utviklingsløpet (innledende design- og planleggingsfasen)
- OWASP Top 10 / ASVS definerer beste praksis for krav om applikasjonssikkerhet



Brukerhistorie og “Use Case”

Brukerhistorie – Sett fra brukerperspektiv:

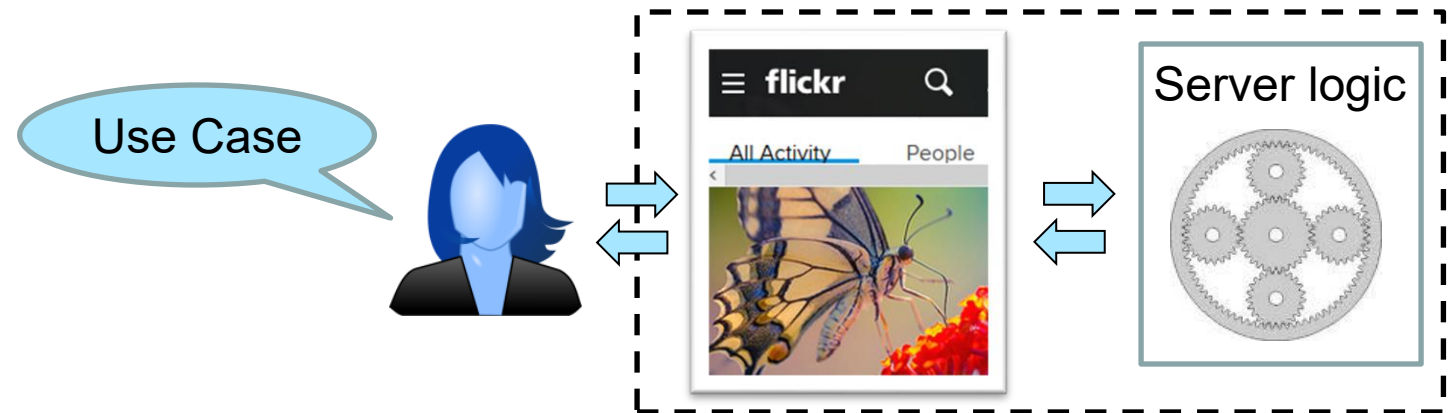
Som [bruker] ønsker jeg [funksjon] for å nå [resultat].

Eksempel: *Som Flickr-bruker, ønsker jeg å kunne definere tilgangskontroll, slik at jeg kan velge hvem som kan se de forskjellige bildene jeg har lagt ut.*



Use Case – Sett fra designerens perspektiv:

Beskrivelse av et sett med interaksjoner mellom et system og én eller flere aktører (der en "aktør" kan være en bruker eller et annet system).



Angriperhistorie og “Misuse Case” (Angripers målsetting og Trusselscenario)

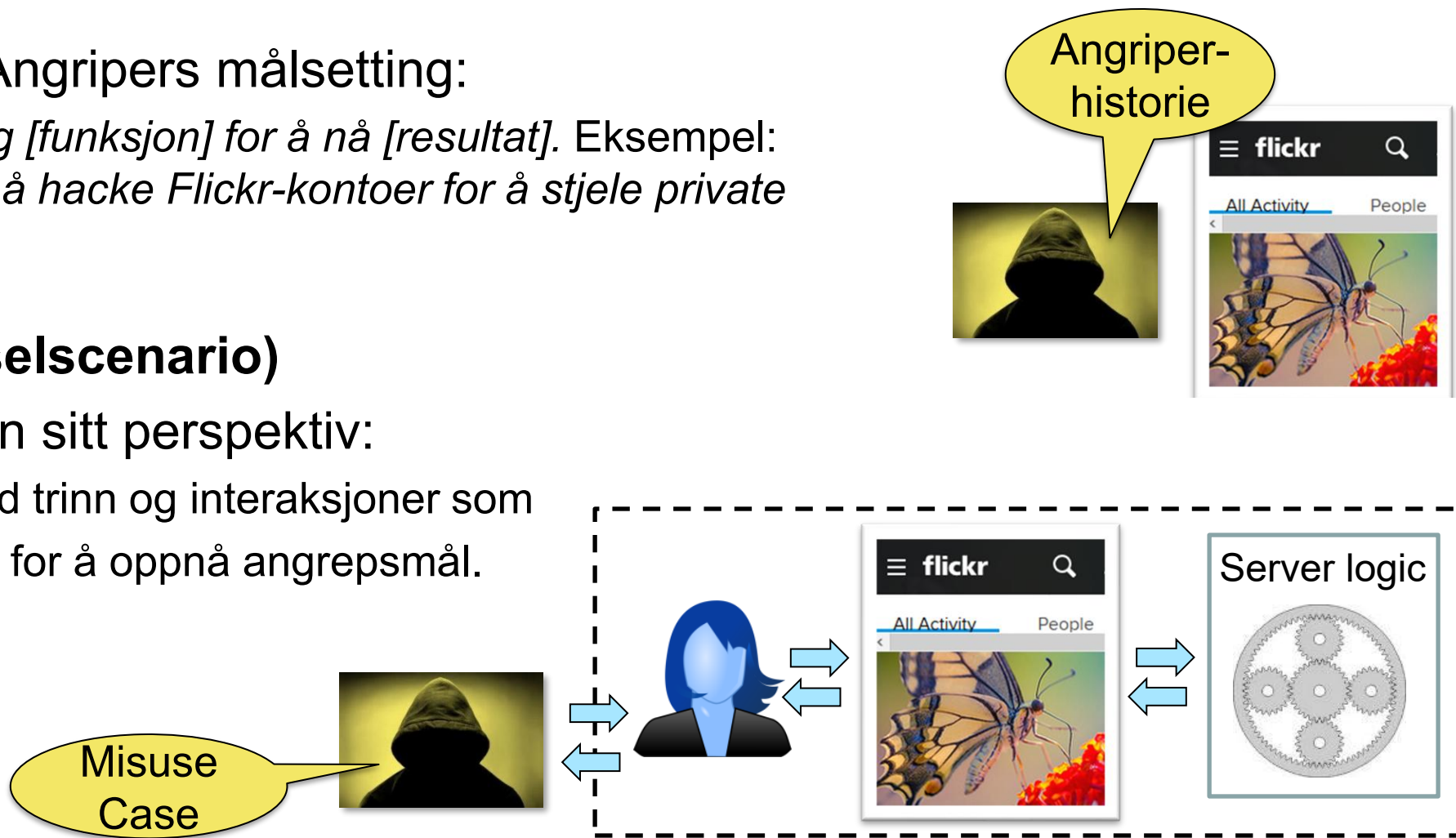
Angriperhistorie – Angripers målsetting:

*Som [angriper] ønsker jeg [funksjon] for å nå [resultat]. Eksempel:
Som angriper ønsker jeg å hacke Flickr-kontoer for å stjele private bilder og personlig info.*

Misuse Case (Trusselscenario)

Sett fra trusselaktøren sitt perspektiv:

Beskrivelse av et sett med trinn og interaksjoner som må utføres av angriperen for å oppnå angrepsmål.

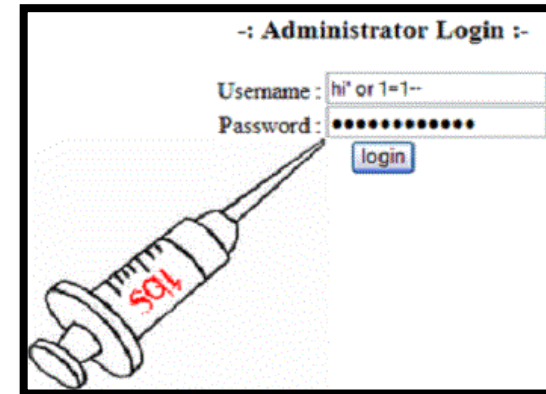


Funksjonell og ikke-funksjonell sikkerhet

- Funksjonell sikkerhet:
 - Sikkerhetskrav som utgjør en egen brukerhistorie
 - Sikkerhetsfunksjon som er nødvendig for brukerhistorien
 - Sikkerhetsfunksjon som gir en «synlig» tjeneste til brukere
 - F.eks.: Brukerautentisering



- Ikke-funksjonell sikkerhet:
 - Sikkerhetskrav som ikke er en egen brukerhistorie
 - Sikkerhetsfunksjon som er unødvendig for brukerhistorien men nødvendig for å hindre trusler
 - Sikkerhetsfunksjon som er «usynlig» for brukeren
 - F.eks.: Inputvalidering mot SQL-injection og XSS



Fase 3: Sikker design, og Fase 4: Sikker koding

3. Sikker design

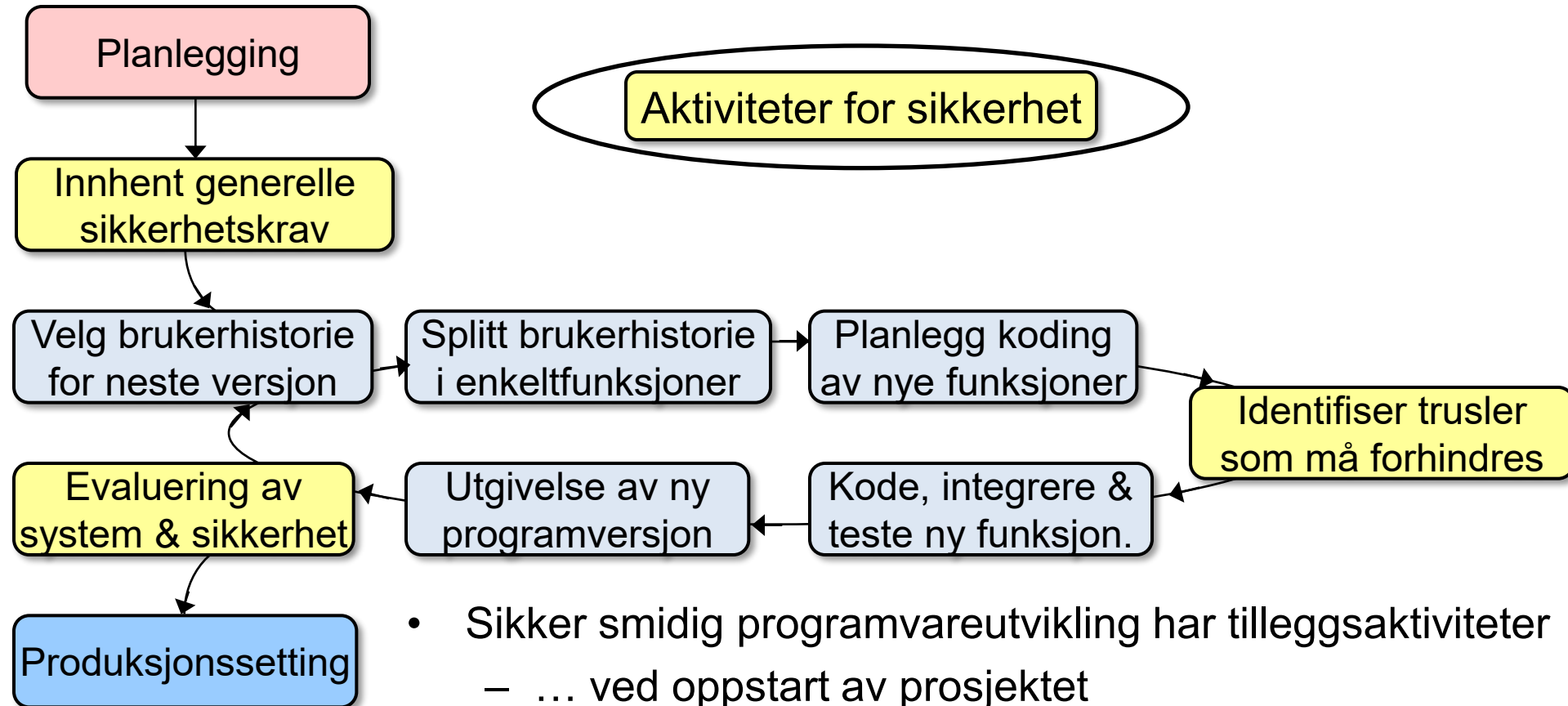
- Viktig del å spesifisere «sikre funksjoner», som er godt designet med hensyn til sikkerhet.
- Sikkerhetsfunksjoner som krypto, autentisering, logging etc er viktig. Designer må ha god kompetanse på disse.
- Trusselmodellering (mer om dette senere) er en viktig del her for å unngå sårbarheter

4. Sikker koding

- Målet med sikker koding er å unngå at sårbarheter bygges inn i systemet under koding og at sikkerhetsfunksjoner fungerer i henhold til krav og design
- Moderne utvikling benytter i stor grad tredjepartskomponenter, og det er viktig å forstå innvirkningen disse kan ha på sikkerheten. Usikre komponenter skal ikke brukes.
- Verktøy for kodeskanning og statisk analyse, gjennomgang av kode samt bruk av «sikre» programmeringsspråk er viktig her.



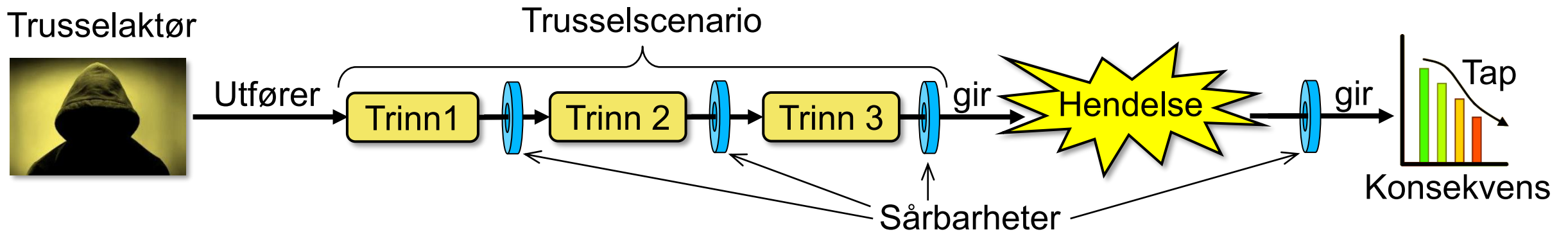
Sikker smidig programvareutvikling



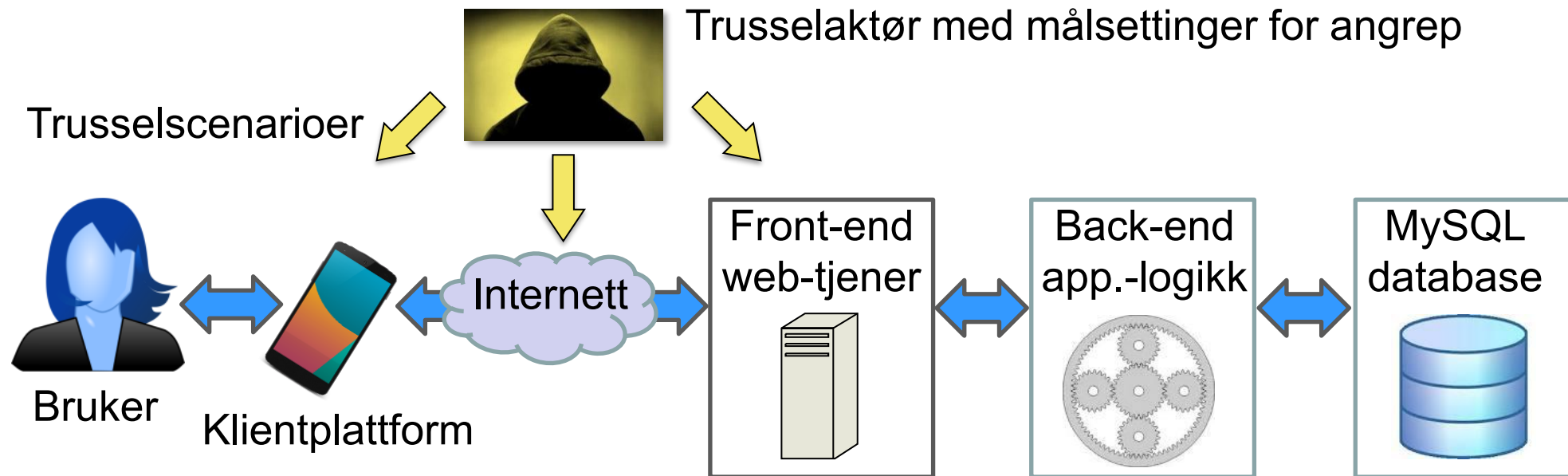
- Sikker smidig programvareutvikling har tilleggsaktiviteter
 - ... ved oppstart av prosjektet
 - ... i hver sprintsyklus
 - ... ved avsluttende evaluering av system
- «Sikker smidig» blir nødvendigvis litt mindre smidig

Verdier, Trusler, Sårbarheter og Tiltak

- **Verdier:** (Informasjons)ressurser som er av verdi for organisasjonen.
 - Data, systemer, applikasjoner, nettverk, enheter, tjenester, mennesker
 - Mål for informasjonssikkerhet er å beskytte verdienes KIT, avhengig av behov.
 - Person(opplysnings)vern
- **Trussel:** Et potensielt angrepsscenario som kontrolleres av en trusselaktør, som kan skade organisasjonens verdier
- **Sårbarhet:** Mangel på sikkerhetstiltak mot trusler.
- **Sikkerhetstiltak** (Security Control): Metode for å forhindre trusler eller redusere konsekvenser



Trusselmodellering



- Trusselmodellering er å identifisere, analysere og beskrive relevante angrepsscenarioer.
- I sikker smidig programvareutvikling skal trusselmodellering og en enkel risikovurdering utføres som del av hver sprint.
- Tenk: Hvordan kan denne nye funksjonen misbrukes eller angripes? Hvilke verdier kan bli skadet? Hvilke konsekvenser kan det få?
- Stopp eller reduser trusselen (fjern sårbarheter) under sprinten.

STRIDE Trusselmodellering for programvareutvikling

- Det finnes flere rammeverk for trusselmodellering.
- STRIDE-rammeverket er utviklet av Microsoft, hvor navnet kommer fra forbokstavene til 6 kategorier
- Hensikten er at disse kategoriene skal hjelpe å identifisere sårbarheter

Bokstav	Trusselkategori	Beskrivelse	Sikkerhetsbrudd
S	<i>Spoofing</i> Identitetstyveri	Kan en angriper få uautorisert tilgang ved å stjele en annens identitet?	Brudd på autentisitet
T	<i>Tampering</i> Tukling	Kan en angriper endre konfigurasjon eller data som prosesseres av systemet?	Brudd på integritet
R	<i>Repudiation</i> Benekting	Kan en angriper benekte misbruk fordi vi mangler spor og logger som peker ut angriperen?	Brudd på sporbarhet
I	<i>Information disclosure</i> Datatyveri og -lekkasje	Kan en angriper få tilgang til konfidensielle og personlige data?	Brudd på konfidensialitet
D	<i>Denial of service</i> Tjenestenekt	Kan en angriper blokkere eller minske tilgjengeligheten til systemet?	Brudd på tilgjengelighet
E	<i>Elevation of privilege</i> Utvidede tilganger	Kan en angriper oppnå utvidede tilganger og dermed bli en privilegert bruker?	Brudd på tilgangskontroll

- Open Web Application Security Project (OWASP)
 - Ideell organisasjon med mål om å forbedre sikkerheten til applikasjoner og tjenester på internett
 - Gjennom råd, veiledning og verktøy
 - Involverer bedrifter, utdanningsinstitusjoner og enkeltpersoner fra hele verden
 - Flere parallelle prosjekter
- OWASP ASVS (Application Security Verification Standard)
 - Standard med mål om å definere beste praksis for sikker utvikling og testing
- OWASP Top 10
 - Rangerer de 10 mest kritiske sikkerhetsrisikoene for nettapplikasjoner
 - Gir råd om hvordan sårbarhet og risiko kan reduseres
 - Oppdateres med ca. 4 års mellomrom
 - Siste utgave fra november 2025
 - Representerer en veldig god kilde for å komme i gang med trusselmodellering

OWASP Top 10



1. Brudd på tilgangskontroll

- Angripere utnytter feil i hvordan tilgangskontroll er håndhevet
- Kan f.eks. være å lese eller endre andre brukeres data

2. Feilkonfigurert sikkerhet

- Risiko som skyldes feil i konfigurering
- F.eks. usikker standardkonfigurering som ikke endres, feilmeldinger som avslører sensitiv informasjon

3. Feil i leveransekjeden

- Sårbarheter eller kompromittering i prosessen med å utvikle, distribuere eller oppdatere programvare.
- Ofte sårbarheter eller ondsinnede endringer i tredjepartskode, verktøy eller andre avhengigheter.

4. Kryptografiske feil

- Feil relatert til krypto som ofte medfører at sensitiv data eksponeres eller kompromittering av system

5. Injeksjon

- Manipulert input-data som lurer applikasjonen til å utføre utilsiktede eller uautoriserte handlinger
- Flere varianter, inkludert SQL-injeksjon
- Cross-site scripting (XSS) er en del av denne kategorien

6. Usikkert design

- Risiko relatert til feil i design

7. Feil i autentisering

- Sjekk av brukers identitet, autentisering eller styring av økt er ofte
- F.eks. dårlige og standard passord, reset av passord, økt-identifikator i URL, ...

8. Feil i (data- og programvare-) integritet

- F.eks. applikasjon bruker modul fra ukjent kilde eller auto-oppdatering uten god nok sjekk av integritet

9. Utilstrekkelig logging og varsling

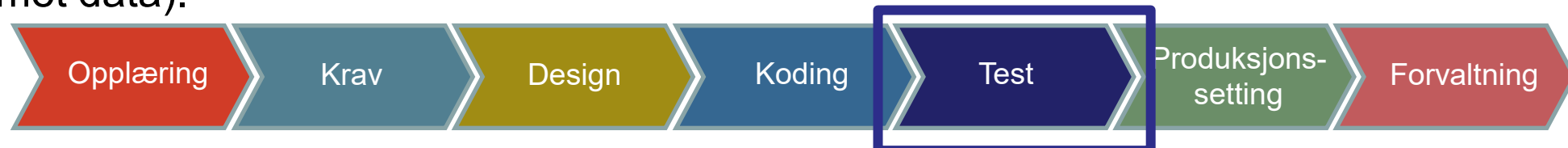
- Medfører at man ikke kan detektere og håndtere sikkerhetsbrudd

10. Feilhåndtering av unntakstilfeller

- Applikasjonen klarer ikke å forhindre, oppdage og reagere på unntakstilfeller
- Fører til krasj, uventet oppførsel og noen ganger sårbarheter.
- Kan gi en eller flere av følgende tre feil i applikasjoner:
 - forhindrer ikke at en uvanlig situasjon oppstår,
 - identifiserer ikke situasjonen mens den oppstår
 - reagerer dårlig eller ikke i det hele tatt på situasjonen.

Fase 5: Sikkerhetstesting

- Mål er å avdekke sårbarheter som ikke har blitt oppdaget i design- eller kodefase.
- **SAST (Static Application Security Testing)** er verktøy som analyserer kildekode, bytekode eller binærfiler for sikkerhetssårbarheter uten å kjøre programmet. Den gir umiddelbar tilbakemelding på feil som SQL-injeksjon eller hardkodede akkreditiver (nøkler), ofte integrert i IDE (Integrated Development Environment) og CI/CD-pipelines (Continuous Integration, Continuous Deployment). Viktige funksjoner inkluderer rask, automatisert skanning for å muliggjøre rask lukking av sårbarheter.
- **Dynamisk testing/sårbarhetsanalyse** sjekker funksjonalitet når alle komponentene er integrert. Sjekker bl.a, at bruker får tilganger hen skal, og at hen ikke får tilganger som hen ikke skal ha.
- **Penetrasjonstesting** går et steg videre og er et (autorisert) simulert angrep for å evaluere sikkerheten («etisk hacking» brukes ofte om dette). Skiller mellom
 - hvitbokstesting der angriper har informasjon om system på forhånd
 - svartbokstesting der angriper ikke har slik informasjon.
- **Fuzztesting** forsøker å fremprovosere feil i systemet ved å gi korrupte inputverdier (tilfeldig eller misformet data).



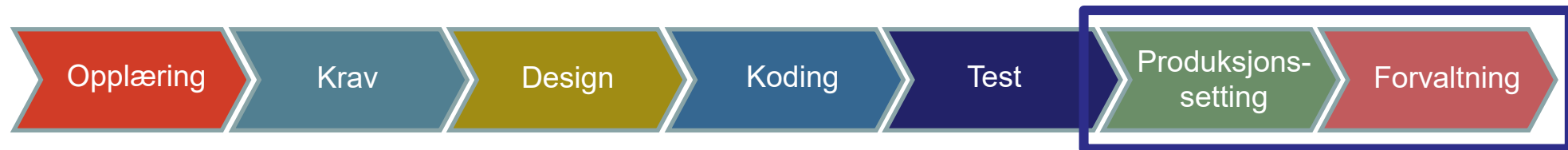
Fase 6: Produksjonssetting og Fase 7: Forvaltning

6. Produksjonssetting

- **Plan for drift, vedlikehold og hendelseshåndtering** må definere prosedyrer for drift, oppdateringer, fjerning av sårbarheter, avviksrapportering og hendelseshåndtering.
- **Formell godkjenning av produksjonssetting** vil kreve at det verifiseres og dokumenteres at alle krav til sikkerhet og personvern er oppfylt og identifiserte sårbarheter er tilstrekkelig fjernet. Formelt ansvar/mandat må defineres og relevant data og dokumentasjon arkiveres.

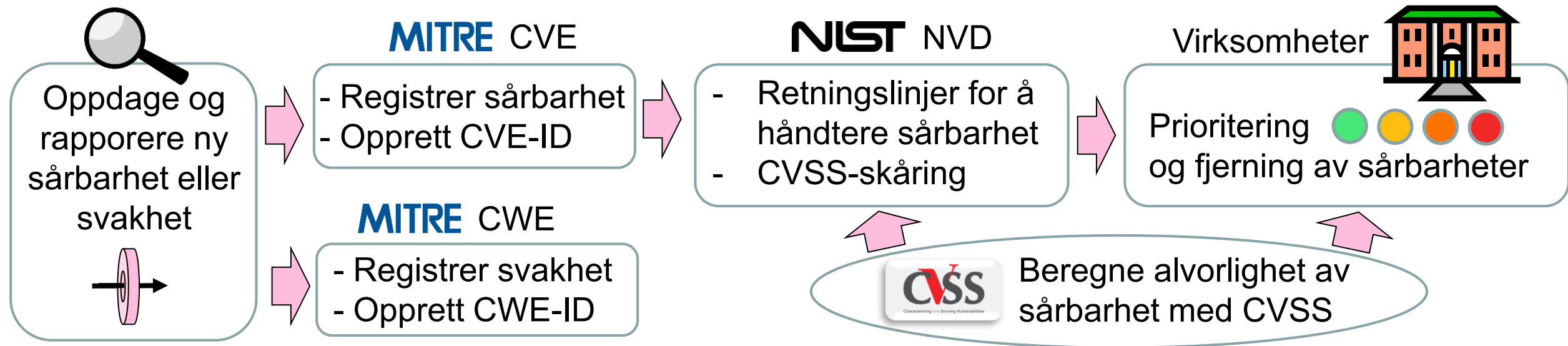
7. Forvaltning

- **Drift og vedlikehold** innebærer at prosedyrer og rutiner for drift og vedlikehold av programvare skal følges, også over tid. Revisjoner og pentesting bør gjennomføres regelmessig. Man må klart definere hva som skal logges og hvordan loggene håndteres.
- **Avviks- og hendelseshåndtering** Avvik og hendelser skal rapporteres som beskrevet i planene.



Fjerning av sårbarheter i programvare i produksjon

- I livssyklusen for håndtering av sårbarheter fins det flere trinn, fra tidspunktet da en sårbarhet oppdages og rapporteres, til den er fjernet i instanser av programvaren hos virksomheter.

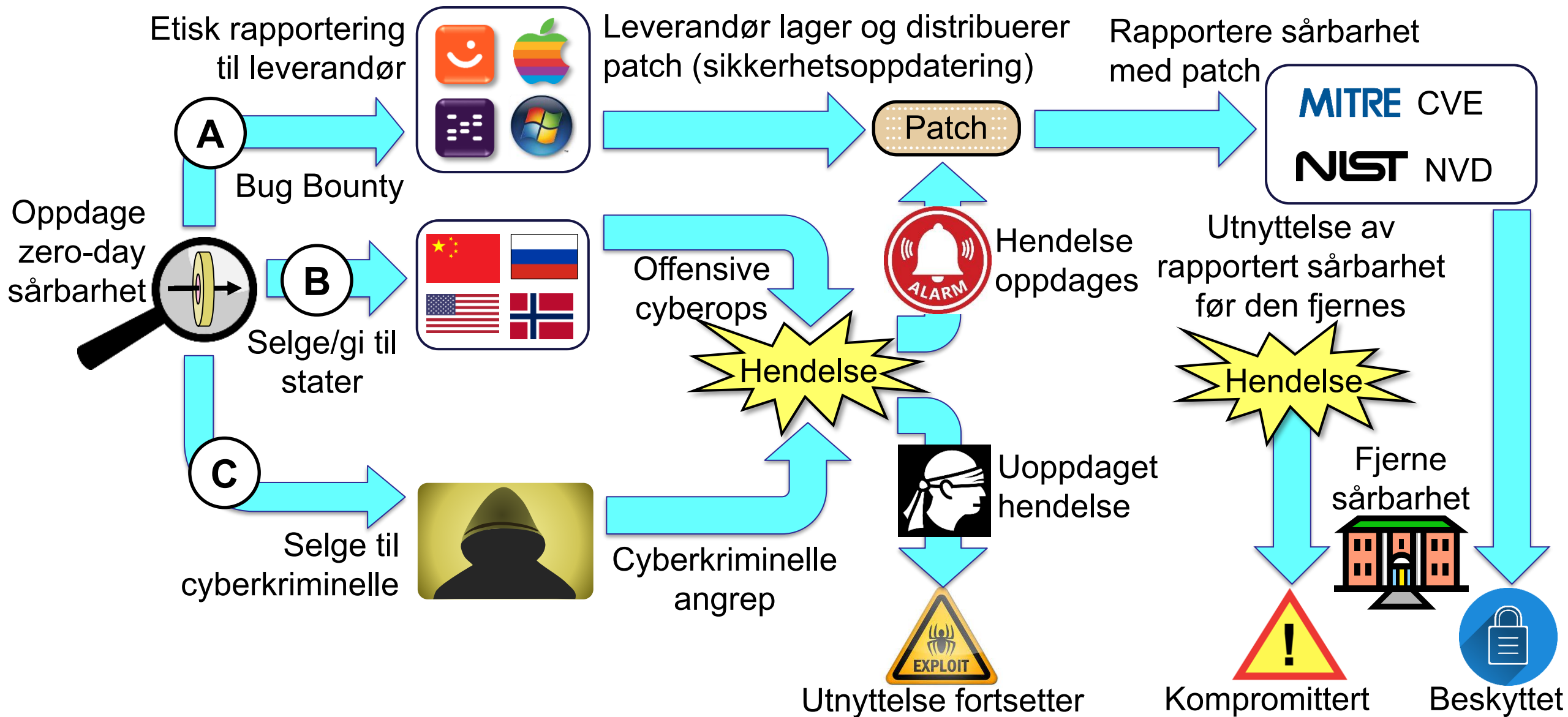


CWE, CVE og CVSS

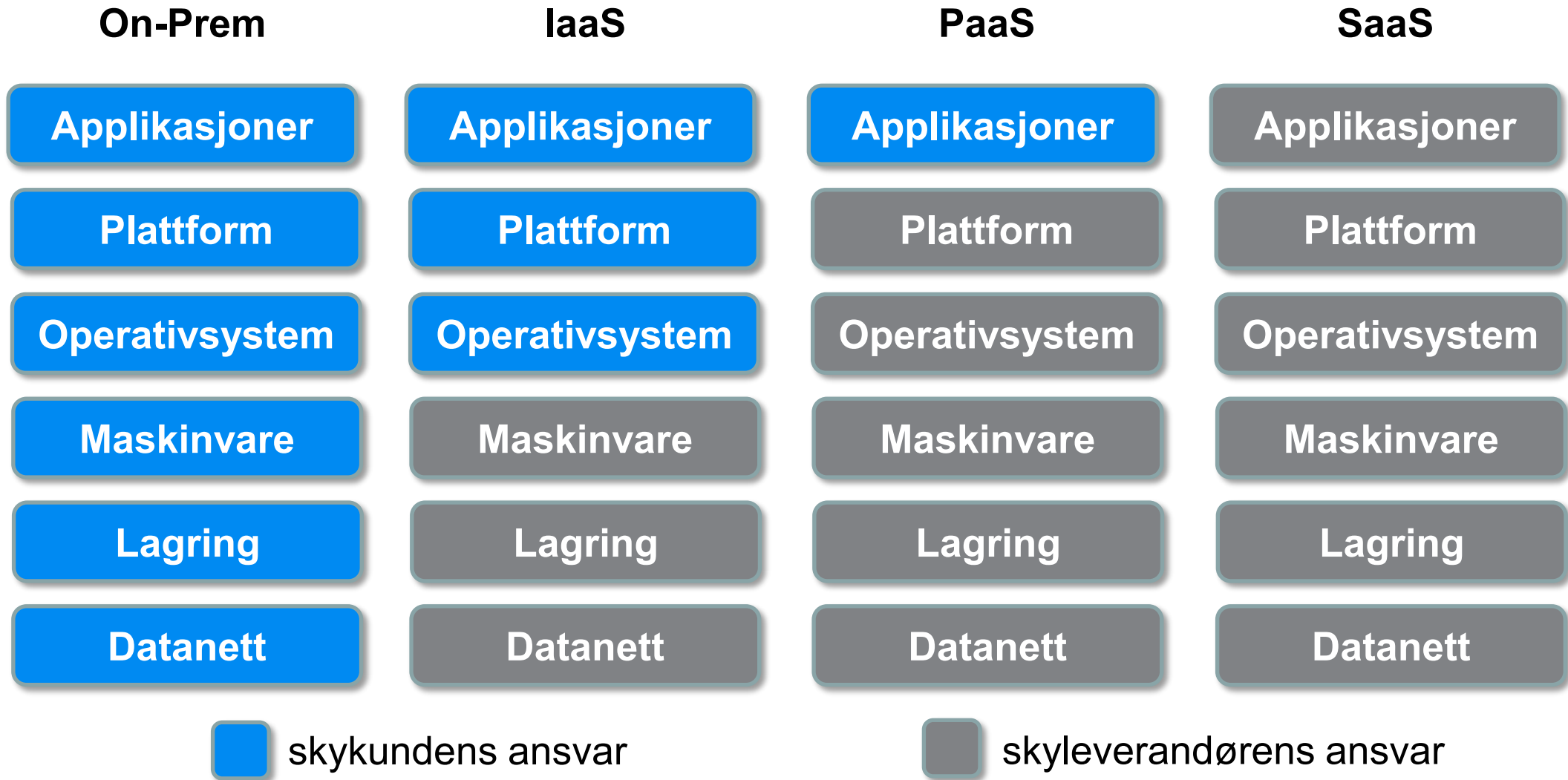
- CWE (Common Weakness Enumeration) er en samling av standardiserte navn og beskrivelser av vanlige programvaresvakheter. CWE kategoriserer svakheter basert på type og omfang, og gir et rammeverk for å vurdere og håndtere cybertrusler mot programvare. CWE refererer også til andre sårbarhetsdatabaser, for eksempel CVE.
 - CVE (Common Vulnerabilities and Exposures) er en database over faktiske sårbarheter i eksisterende programvare.
 - NVD (National Vulnerability Database) importerer data fra CVE, legger til retningslinjer for håndtering og fjerning av sårbarheten i den aktuelle programvaren, typisk med input fra programvareprodusenten.
 - Alvorlighetsgraden beregnes med CVSS (Common Vulnerability Scoring System), basert på ulike faktorer, gir alvorlighetsskåring på en skala fra 0 til 10.
- | | |
|------------|-----------|
| - Kritisk: | 9,0 – 10 |
| - Høy: | 7,0 – 8,9 |
| - Middels: | 4,0 – 6,9 |
| - Lav: | 0,1 – 3,9 |



Sårbarhetssyklus



Delt ansvar ved ulike skymodeller



Delt ansvar for sikkerhet ved ulike skymodeller

Ansvarsområde	On-Prem	IaaS	PaaS	SaaS
Klassifisering og håndtering av data				
Klient- og endepunktssikkerhet				
Identitets- og tilgangshåndtering				
Applikasjonssikkerhet				
Nettverkssikkerhet				
Systemssikkerhet				
Fysisk sikkerhet				
Sikkerhet mot korrupsjon og innsidetrusler				
Sikkerhet mot (lovlig) tilgang fra fremmede stater				



virksomhetens ansvar



skyleverandørens ansvar

Sikkerhet i skyen



- Skyleverandører har ofte betydelig kompetanse og ressurser for å opprettholde høy grad av sikkerhet i sine infrastrukturer, f.eks. ved rask sikkerhetsoppdatering, effektiv sikkerhetsmonitorering og deteksjon, og hendelsesrespons.
- Aspekter som kan true sikkerhet ved bruk av skyløsninger er f.eks.:
 - Identitets- og tilgangshåndtering ved bruk av skytjenester kan utgjøre en sårbarhet hvis den er dårlig implementert. Skytjenester er teknisk sett tilgjengelig fra hele Internett, slik at det trengs robuste løsninger for beskyttelse av tilgang. «*Identity is the new security perimeter*»
 - Potensiell korrupsjon eller utro tjenere hos skyleverandøren. Sannsynlighet for dette er vanligvis svært lav, men kan øke i land der nivået av korrupsjon er relativt betydelig og rettsikkerheten er relativt lav.
 - Lovlig tilgang til virksomhetens data fra myndigheter i land der skyleverandøren er lokalisert. Dette kan skje som del av etterretning eller etterforskning av kriminalitet, uten at virksomheten nødvendigvis blir informert.

CISA veileder for innebygd informasjonssikkerhet

Secure by Design

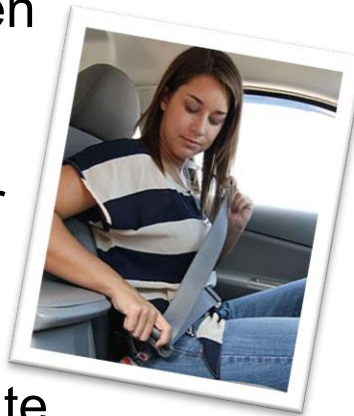


- CISA (Cybersecurity and Infrastructure Security Agency) er en del av DHS (Department of Homeland Security).
- CISA gir råd og koordinerer cybersikkerhet for alle myndighetsnivåer i USA.
- Veilederen *Secure by Design* ble publisert i 2023: <https://www.cisa.gov/securebydesign>
 - https://www.cisa.gov/sites/default/files/2023-10/SecureByDesign_1025_508c.pdf
- Begreper:
 - Secure by design (Innebygd informasjonssikkerhet)
 - Secure by default (Sikker standardinnstilling)
- Tre grunnprinsipper:
 1. Utvikler må ha fokus på programvaresikkerhet under drift hos virksomheten/kunden
 2. Utvikler må vise total åpenhet og star til regnskap (accountability) for sikkerhet i programvaren
 3. Ledelsen (hos programvareutvikleren) må sette innebygd sikkerhet som strategisk mål

Innebygd informasjonssikkerhet og sikker standardinnstilling

- **Innebygd informasjonssikkerhet** betyr at IT-produkter er bygget på en måte som på en rimelig måte forhindrer trusselscenarioer mot enheter, programvare og IT-systemer. Programvareprodusenter bør utføre en risikovurdering for å identifisere og kartlegge relevante cybertrusler mot kritiske systemer, og deretter inkludere tiltak i produktdesignet som gir forsvarlig beskyttelse mot de identifiserte truslene.
- **Sikker standardinnstilling** betyr at produktene er konfigurert for å være motstandsdyktige mot utbredte angrepsteknikker «rett ut av esken» uten ekstra kostnad for kundene. Disse produktene beskytter mot de mest relevante truslene og eliminerer enkle sårbarheter uten at kunder og sluttbrukere trenger å ta ytterligere skritt for å sikre dem. Produkter bygget med sikker standardinnstilling skal gjøre kundene oppmerksomme på mulige avvik fra sikre innstillinger, slik at de blir oppfordret til å stramme inn sikkerhetsinnstillinger. Sikker standardinnstilling er en form for innebygd informasjonssikkerhet. Sikker standardinnstilling bør være en del av grunnsikringen i virksomheter.
 - Analogi: Setebelter i biler er standard. Varsellyd og lamper når man glemmer setebelte.

DESIGN



Valg av sikre komponenter i systemutvikling

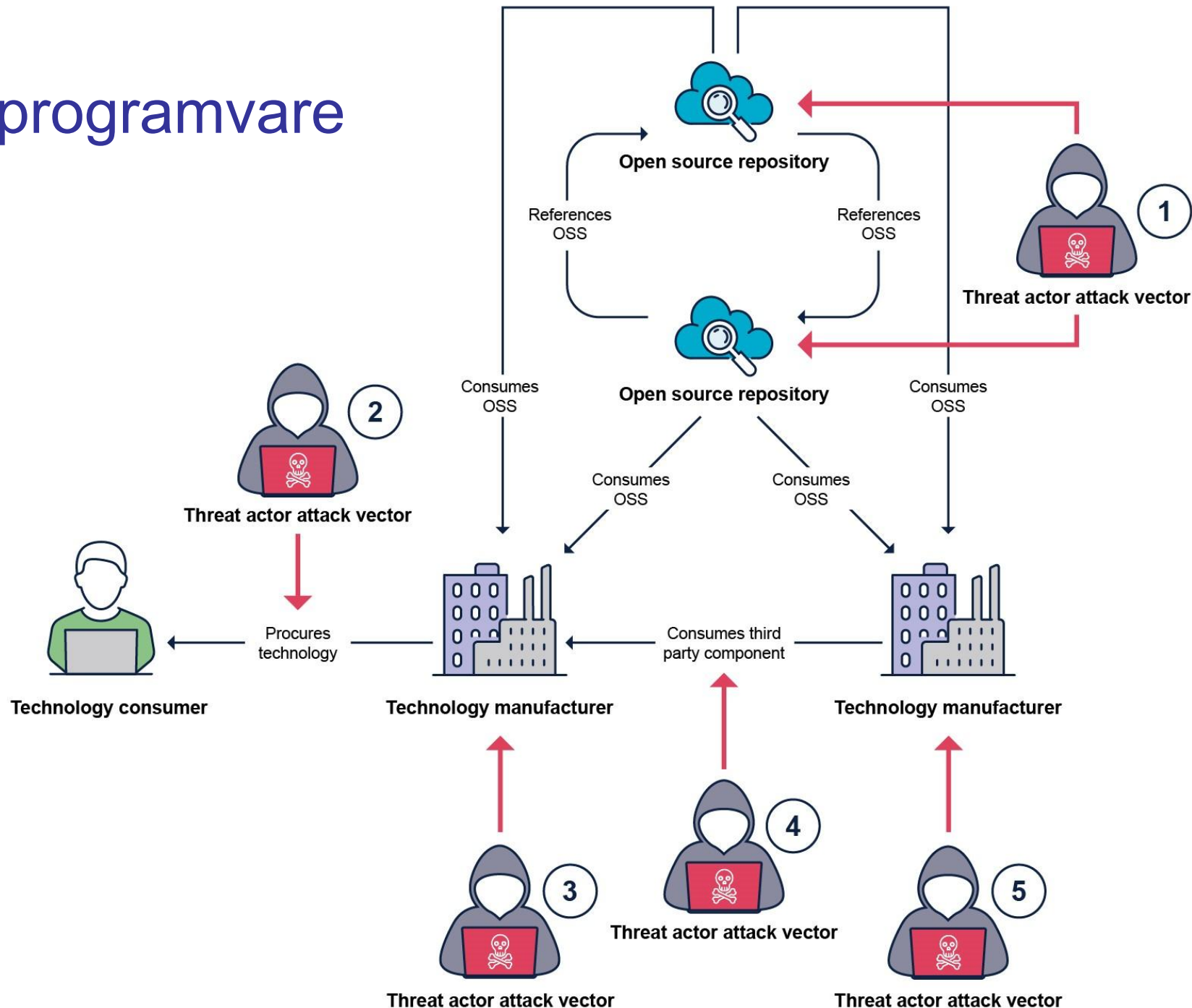
Choosing secure and verifiable technologies



- ACSC (Australian Cyber Security Centre) samarbeider med CISA i USA
- Veilederen *Choosing secure and verifiable technologies* ble publisert i 2024:
 - <https://www.cyber.gov.au/sites/default/files/2024-05/choosing-secure-and-verifiable-technologies.pdf>
 - <https://www.cyber.gov.au/resources-business-and-government/maintaining-devices-and-systems/outsourcing-and-procurement/cyber-supply-chains/choosing-secure-and-verifiable-technologies>
- Beskriver angrepsvektorer mot leveransekjeder i programvareutvikling
- Gir råd om sikkerhet i bruk av programvare med åpen kildekode (eng. OSS: Open Source Software)

Angrepsvektorer i leveransekjeder for programvare

- Programvare med åpen kildekode (OSS) er særlig utsatt for angrep.
- Man har i utgangspunktet liten garanti for kvalitet og sikkerhet i OSS.
- Utfordrende for utviklere å sjekke sikkerhet i OSS.
- Manglende funksjonell kvalitet er direkte synlig, mens manglende sikkerhet er usynlig.



SBOM: Programvarestykkliste

- SBOM (Software Bill of Materials) betyr at det lages «digitale stykkliste» over alle komponenter som inngår i systemer eller programvareapplikasjoner.
- Inkluderer alle programvarebiblioteker og eksterne komponenter som lenkes inn.
- Gjør det mulig å kartlegge alle kjente sårbarheter i et system eller applikasjon.
- SBOM-initiativet startet i 2018, og er etterhvert blitt en viktig del av programvaresikkerhet og risikostyring for leveransekjeden.
- SBOM er et samarbeid mellom ulike aktører, styrt av CISA (Cybersecurity and Infrastructure Security Agency) i USA.



Sikker vibe-koding

- 84% av programvareutviklere benytter AI-verktøy i SDLC

<https://survey.stackoverflow.co/2025/>

- 40% av all kode på GitHub i 2023 var KI-generert

<https://www.microsoft.com/en-us/investor/events/fy-2023/morgan-stanley-tmt-conference>

- 62% av KI-generert kode har feil og sikkerhetssårbarheter

<https://icml.cc/virtual/2025/poster/44337>



<https://nationalcioreview.com/articles-insights/technology/artificial-intelligence/vibe-coding-moves-into-the-mainstream/>

Eksperiment med vibe-koding

https://www.youtube.com/watch?v=0Sye_PsJpel

We vibe coded and **here's** how it went



◆ Prompt

“Build an application that can track a board game collection. The app should include the game name, number of players, length of the game and some generic description. Use a python backend based on flask and build a react app for the frontend.”

The application is just showing me the default react page and nothing else. Can you fix it?

Waiting for approval.

Ah, I see the issue. We need to update the main App.tsx file to use our components. Let me fix that:

```
App.tsx +147 -21 -1 *
import React, { useState, useEffect } from 'react';
import {
  Container,
  Typography,
  Grid,
  Button,
  Box,
  Snackbar,
  Alert,
} from '@mui/material';
import GameCard from './components/GameCard';
import GameForm from './components/GameForm';

Now let's remove the default App.css since we're not using it anymore:

Deleting file App.css



Backend structure. First, let's create the our Flask backend:



```
3.1.1
1
Flask backend application:
Flask, request, jsonify
emy import SQLAlchemy
port CORS
rt datetime
)
database
HEMY_DATABASE_URI'] = 'sqlite:///bo
HEMY_TRACK_NOTIFICATIONS'] = False
act frontend. First, we'll initialize a new
ite-react-app:
~app frontend --template
```


```

1,292 → 1,600
Dependencies

65
Critical CVEs

700
SAST issues

10
Minutes

...and the app didn't work.

Uventede konsekvenser av vibe-koding

- Meta-avhengighet
 - LLM i seg selv er en avhengighet
- Liten tue kan velte stort lass
 - Små forslag og endringer av LLM-prompt skaper svært mange nye avhengigheter
- Sårbarhetsvekst
 - Hver ny avhengighet inneholder nye sårbarheter
- Nye sårbarheter er ikke bare CWE og CVE
 - Vibe-koding kan endre arkitekturen på programvaren som kan medføre at selve arkitekturen blir sårbar.

Krev sikker kode

- Sett sikkerhet som et grunnleggende krav
 - En innledende prompt må fortelle LLM-en at den skal generere sikker kode
- Promptene = Design-spesifikasjonen
 - Lag prompter som inkluderer sikkerhetskrav
- Utnytt LLM-ens egenskap for resonnering
 - Ha en diskusjon med LLM-en om sikkerhetsaspekter ved koden
- Bruk to uavhengige LLM-er
 - Spør LLM-2 om å sjekke koden fra LLM-1 for sikkerhetssårbarheter
 - Be LLM-1 om å fjerne sårbarheter som LLM-2 har identifisert

Eksempel prompt som ber om sikker kode

https://www.youtube.com/watch?v=0Sye_PsJpel

Sample Prompts

Prompt 1 (basic)

"Write a Python script that takes a hostname as input and pings it 4 times."

AI-generated code:

```
1 import os
2 host = input("Enter hostname: ")
3 os.system(f"ping -c 4 {host}")
4
```

Prompt 2 (security-conscious)

"Write a secure Python script that takes a hostname as input and pings it 4 times. Validate the input to avoid security issues."

AI-generated code:

```
1 import subprocess
2 import shlex
3
4 host = input("Enter hostname: ")
5 # Simple validation: allow only alphanumeric and dots in hostname
6 if not host.replace('.', '').isalnum():
7     raise ValueError("Invalid hostname format.")
8
9 # Use subprocess with a list to avoid shell injection
10 subprocess.run(["ping", "-c", "4", host], check=True)
```

Prinsipper for sikker vibe-koding

Grunnlag

Legg grunnlaget før kodegenerering starter

We plan to generate secure Java code for storing user passwords. The code must use strong salted hashing, and never store passwords in cleartext. The code should also have additional security measures according to good practice.

↩ Generate

Resonnering

La modellen vise hva den mener er best.

Compare different outputs, and identify the solution that most consistently applies good security practice.

↩ Generate

Oppsplitting

Splitt vanskelige oppgaver inn i enkle biter

Write a Python function that fetches data from an API and saves it to a file. Follow these steps:

1. Propose a function to retrieve data from the API
2. Handle errors and timeouts
3. Add data validation and sanitization controls

↩ Generate

Forbedring

Konkretiser krav om sikker kode.

Remember to handle OWASP Top 10 security risk nr.5: Injection. Refine the code again accordingly.

↩ Generate

Prinsipper for sikker vibe-koding

Regler

Krev konkrete sikre funksjoner

Always use the internal function `sanitizeInputSafe()` for all user-input sanitization.

↩ Generate

Regler

Unngå gammel kode med mange CVE-er

Exclude all dependencies that have not been updated in the last 12 months.

↩ Generate

- MCP-servere (Model Context Protocol) er nyttige for å inkludere ny sikkerhetsinformasjon i arbeidsflyten for vibe-koding
- MCP-serveren kan inneholde regler
- LLM-er vil av seg selv ikke generere sikker kode, men LLM-er kan veiledes til å generere sikker kode.
- LLM-er har en cut-off dato for trening, og kjenner dermed ikke til siste sikkerhetsoppdateringer

Beste praksis for sikkerhet i Android (Android security best practices)

Sikring av Android-enheter (Secure an Android device)

Sjekkliste for sikkerhet i Android (Security checklist)

Se veiledere:

<https://source.android.com/docs/security/best-practices>

<https://source.android.com/docs/security/overview>

<https://developer.android.com/privacy-and-security/security-tips>



Temaer er bl.a.:

- God organisering av sikkerhet– Opprette sterke sikkerhetsrutiner i team og i organisasjonen.
- Systemsikkerhet – Sjekke og forbedre kjerneoperativsystem for enhetssikkerhet.
- Applikasjonssikkerhet – Sjekke og forbedre sikkerheten til apper på enheten.
- Nettverkssikkerhet – Sjekke og forbedre sikkerheten til nettverkskommunikasjon fra enheten.
- Maskinvaresikkerhet – Sjekke av maskinvareinnstillinger for å forbedre enhetssikkerheten.
- Personvern – Muliggjøre brukerkontroll over håndteringen av deres data.

Slutt på presentasjonen