

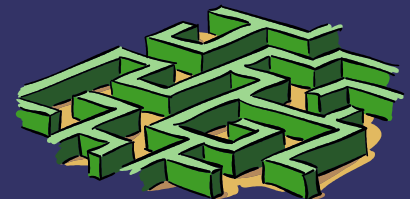
*INF329: Utvalgte emner i programutviklingsteori*

# *Sikkerhetsanalyse av programvare*

Kap. 6, «Auditing Software» (s. 115)

Kristian Harms, [harms@ii.uib.no](mailto:harms@ii.uib.no)

Presentert 21. september 2005



# Merriam-Webster:

## *Audit*

Function: noun

Etymology: Middle English, from Latin *auditus* (act of hearing), from *audire*

**1:**

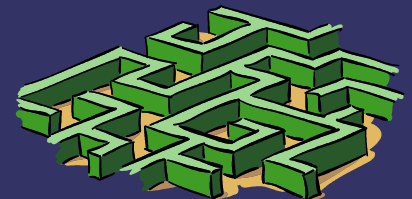
a: A formal examination of an organization's or individual's accounts or financial situation b: The final report of an audit

**2:**

A methodical examination and review

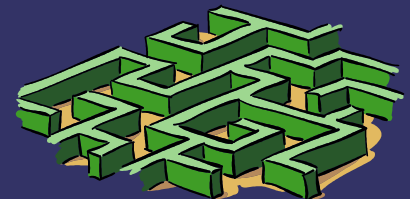
På norsk:

Security Audit = Sikkerhetsanalyse/-gjennomgang



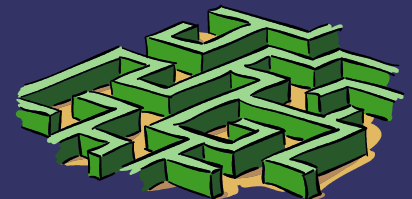
# Generelt

- ⇒ Sikkerhet tenkes på for sent
- ⇒ Gjelder også sikkerhetsanalyse
- ⇒ For fokusert på kodegjennomgang ved slutten av implementasjonsfasen
- ⇒ Bør fokusere mer på arkitekturanalyse
- ⇒ «*Det hjelper ikke å kontrollere vinduer og dører når huset mangler en vegg*»



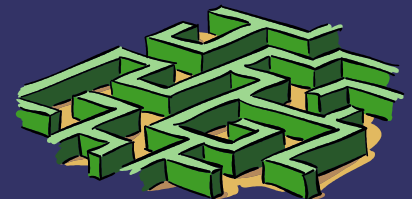
# *To typer analyse*

- ➔ Analyse av arkitekturen/designen
- ➔ Analyse av implementasjonen



# *Arkitekturanalyse: Når?*

- ➔ Begynn når man har et førsteutkast til arkitekturen for prosjektet.



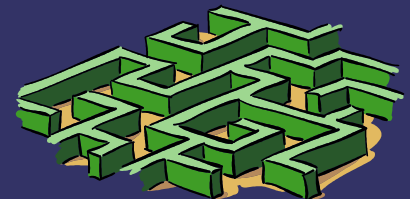
# *Arkitekturanalyse: Hvem?*

- ⇒ Noen må ha ansvaret
- ⇒ Trekk inn folk fra utenfor prosjektet
- ⇒ Unngå å bruke sikkerhetsansvarlig (kap. 2)
- ⇒ Vanskelig jobb, krever erfaring og ekspertise
- ⇒ Dann gjerne et lag med variert bakgrunn



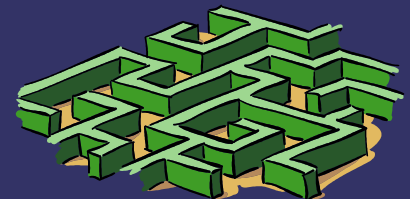
# Arkitekturanalyse: Hvordan?

- ➔ Boken anbefaler tre faser:
  1. Innsamling av informasjon
  2. Analysering av problemer
  3. Utarbeidelse av en rapport
- ➔ Mer strukturert enn en *ad hoc* «red teaming»-fremgangsmåte



# *Arkitekturanalyse: Samling av info: Kravspesifikasjonen*

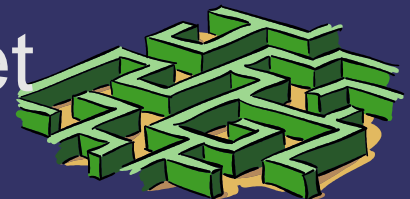
- ➔ Har en klar fremstilling av prosjektets oppgaver, konteksten for prosjektet.
- ➔ Har helst en sikkerhetspolise
- ➔ Bør utfylles av intervjuer med arkitektene osv.



# *Arkitekturanalyse: Samling av info:*

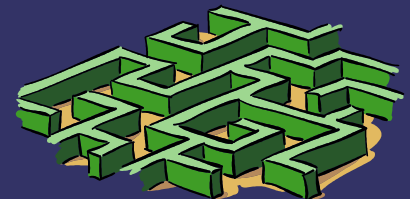
## *Fremgangsmåte*

- ➔ Forstå systemet på et høyt nivå
- ➔ Lær mest mulig om sikkerhetsrelevante ting
- ➔ Snakk med sikkerhetsansvarlig (kap. 2)
- ➔ Les all dokumentasjon for prosjektet
- ➔ Noter inkonsistens og ta dette opp med relevante personer
- ➔ Inkonsistens er en vanlig kilde til sikkerhetsfeil
- ➔ Fokuserte samtaler med utviklingslaget



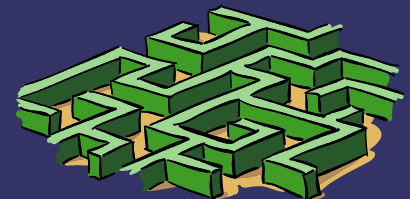
# *Arkitekturanalyse: Samling av info: Fremgangsmåte (forts.)*

- ➔ Lær å prioritere tiden
- ➔ Det er fullt mulig å skusle all tilgjengelig tid bort på én komponent
- ➔ Undersøk deler av systemet i prioritert rekkefølge
- ➔ Husk å inkludere «krympepakket» programvare, da dette medfører reelle farer: Les det du kan på BugTraq og lignende, plag kundeservice, osv



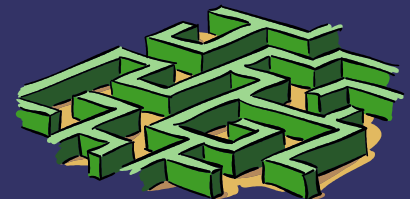
# *Arkitekturanalyse: Analyseringsfasen*

- ➔ Handler om å vurdere risiki metodisk
  - alvorlighetsgrad
  - mottiltak
  - omkostninger



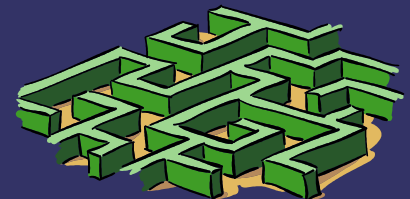
# Arkitekturanalyse: Analysefasen: Angrepstrær

- ➔ En metodisk måte å vurdere risiki
- ➔ Basert på såkalte «feiltrær»
- ➔ Man bygger et tre hvor
  - roten er et mål for en potensiell angriper
  - indre noder er mer abstrakte måter å utføre et angrep på
  - løvnodene er konkrete måter å utføre et angrep på



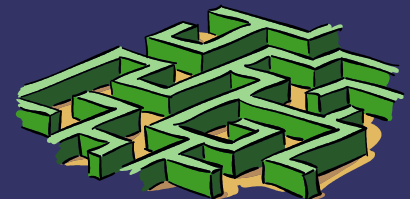
# *Arkitekturanalyse: Analysefasen: Angrepstrær (forts.)*

- ➔ Representerer beslutningsprosessen til en velinformert angriper
- ➔ Mer kunst enn videnskap: Umulig å sørge for at man har et uttømmende tre.



# *Arkitekturanalyse: Analysefasen: Angrepstrær (forts.)*

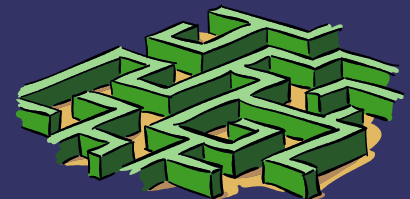
- Relevant informasjon om angrep kan legges inn:
- Tid, krefter, omkostninger, risiko og konsekvenser for angriperen, den angrepne, programvarehuset, osv.



# *Arkitekturanalyse: Analysefasen:*

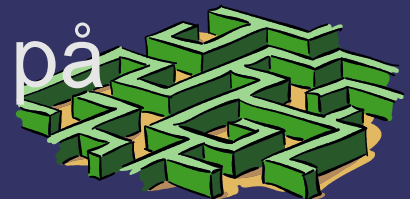
## *Hvordan lage angrepstrær*

- ➔ Fokuser på data og ressurser som kan angripes
- ➔ Modulene i prosjektet
- ➔ Moduler utenfor prosjektet (ofte krympepakket)
- ➔ Grensesnitt mellom disse er ofte svake punkter



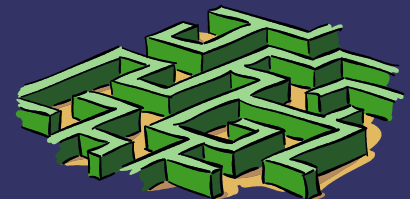
# *Arkitekturanalyse: Analysefasen: Hvordan lage angrepstrær? (forts.)*

- ⇒ Brainstorming med tavle
- ⇒ Skriv ned alle ideer til angrep man har fra før
- ⇒ Folk får alltid nye ideer på stedet
- ⇒ Til slutt:
  - fordel ideene på de tilstedeværende
  - folk lager deltrær hver for seg
  - deltrærne samles til et angrepstre på et nytt møte



# *Arkitekturanalyse: Rapportfasen*

- ➔ Til slutt trenger man noe mer lesbart
- ➔ Lag en rapport utfra angrepstreet
- ➔ Inkluder en oversikt over de aller groveste problemene med pekere inn i selve rapporten
  - fordi mange trenger nettopp denne infoen
  - velegnet til en oppsummering/introduksjon
- ➔ Gruppér problemer etter type
- ➔ Rangér problemer etter alvorlighetsgrad innad i hver gruppe

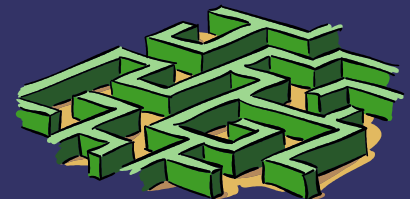


# *Arkitekturanalyse: Rapportfasen:*

## *For hvert problem:*

### ➔ Kort:

- En én linjes vurdering av alvorlighetsgrad
- Anslått omkostning ved et angrep
- Anslått risiko for angriperen
- Anslått arbeidsmengde for angriperen

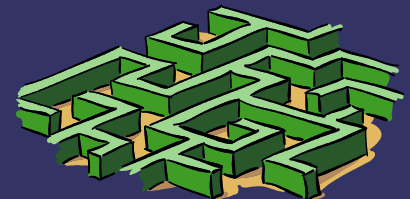


# *Arkitekturanalyse: Rapportfasen:*

## *For hvert problem (forts.):*

### ➔ Lengre:

- Forklar problemet (detaljert nok til at folk uten sikkerhetsbakgrunn skjønner det)
- Gå gjennom konsekvensene
- Hvordan det kan utbedres (foreslå gjerne konkrete forandringer i arkitekturen)



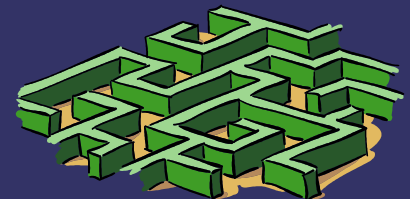
# *Implementasjonsanalyse*

- ➔ Fortsatt viktig
- ➔ To hovedpunkter:
  1. Sjekk at implementasjonen faktisk er i henhold til designen.
  2. Sjekk for svakheter som er spesifikke for implementasjon



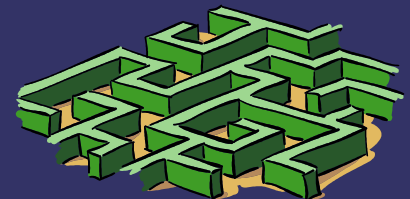
# *Implementasjonsanalyse (forts.)*

- ➔ Vanskelig!
- ➔ Koden er omfattende og kompleks
- ➔ Krever forskjelligartet ekspertise:
  - sikkerhet
  - programmering
  - dataflyt



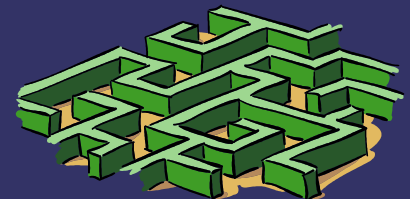
# *Implementasjonsanalyse: Fremgangsmåte*

- ➔ Snarveier må til
  - ikke praktisk å gå gjennom hele koden
- ➔ Still smarte spørsmål til utviklerne
- ➔ Se etter symptomer på problemer
  - f.eks. funksjonskall med rykte på seg
  - forøvrig dekket i andre del av boken



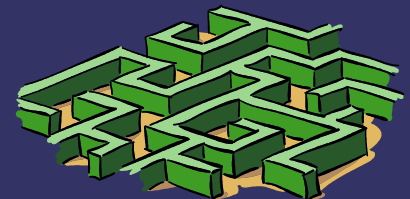
# *Implementasjonanalyse: Fremgangsmåte (forts.)*

- ➔ Fokuser på steder hvor programmet tar inn data fra bruker/komponenter:
  - nettverk
  - kommandolinje
  - miljøvariabler
  - GUI-felter
  - filer som leses
  - komponenter i og utenfor prosjektet
  - og så videre ...



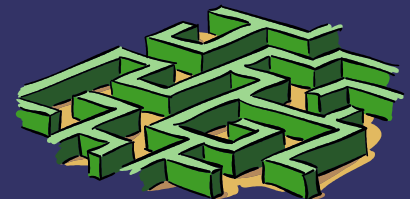
# *Implementasjonsanalyse: Når man finner problemer*

- ➔ Man kan demonstrere hullet
  - eneste måten å bevise at det eksisterer
  - arbeidskrevende
  - utelates helst
- ➔ Man kan omskrive problemområdet
  - greiest selv når man ikke er sikker på at det *er* et problem



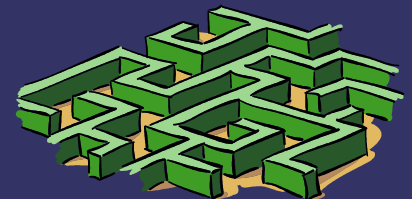
# *Implementasjonsanalyse: Automatiserte verktøy*

- ➔ Utfører en statisk sjekk av programteksten
  - dog smartere enn utstrakt bruker av *grep*
- ➔ Ser etter kjente problemområder:
  - problematiske funksjonskall
  - problematiske språklige konstruksjoner
- ➔ RATS
- ➔ Flawfinder
- ➔ ITS4



# *Implementasjonsanalyse: Automatiserte verktøy (forts.)*

- ➔ Gir lange lister med potensielle problemer
  - må gjennomgås for hånd
- ➔ Ikke et verktøy for at hvermann skal kunne gjøre ekspertenes arbeide
- ➔ Derimot et verktøy for å gjøre ekspertenes jobb enklere
- ➔ Langt fra perfekte



*Spørsmål?*

