Øvelse 41a – Kontinuerlig overvågning af afhængigheder med Dependabot

Formål

Formålet med øvelsen er at opnå praktisk erfaring med automatiseret overvågning og opdatering af tredjepartsafhængigheder ved hjælp af GitHub Dependabot.

I de tidligere øvelser har I etableret en CI-pipeline, som kan bygge applikationen, udføre unit tests, scanne NuGet-afhængigheder og analysere kildekoden med CodeQL.

Nu skal I konfigurere Dependabot til løbende at kontrollere, om projektets afhængigheder har kendte sikkerhedssårbarheder, eller om der findes nyere versioner.

I skal desuden undersøge, hvordan Dependabot kan foreslå opdateringer gennem pull requests, og hvordan den eksisterende CI-pipeline kan anvendes til at kontrollere sådanne ændringer.

Efter øvelsen skal I kunne:

- Forklare formålet med automatiseret dependency management.
- Skelne mellem Dependabot alerts, security updates og version updates.
- Konfigurere Dependabot til at overvåge NuGet-afhængigheder.
- Undersøge sikkerhedsadvarsler og automatiske opdateringsforslag.
- Forklare, hvordan Dependabot og GitHub Actions kan anvendes sammen.
- Vurdere, hvorfor opdateringer stadig kræver kontrol og eventuel manuel godkendelse.

Baggrund

Moderne applikationer indeholder ofte mange tredjepartsafhængigheder.

Disse afhængigheder bliver løbende opdateret med:

- Fejlrettelser.
- Nye funktioner.
- Sikkerhedsrettelser.
- Ændringer i eksisterende funktionalitet.

En pakke, der ikke havde kendte sikkerhedssårbarheder, da applikationen blev udviklet, kan senere vise sig at indeholde en sårbarhed.

Derfor er det ikke tilstrækkeligt kun at undersøge afhængighederne én gang.

Fra NuGet Audit til Dependabot

I øvelse 40 anvendte I NuGet Audit til at identificere kendte sårbarheder i projektets afhængigheder.

NuGet Audit giver sikkerhedsadvarsler under restore og kan vise, hvilke installerede pakkeversioner der er berørt.

Dependabot kan supplere denne kontrol ved at overvåge repositoryets afhængigheder og foreslå opdateringer.

Dependabot indeholder tre centrale funktioner:

Funktion	Formål
Dependabot alerts	Rapporterer kendte sikkerhedssårbarheder i afhængigheder
Dependabot security updates	Forsøger at oprette pull requests, der retter kendte sårbarheder
Dependabot version updates	Kontrollerer efter en tidsplan, om der findes nyere pakkeversioner, og foreslår opdateringer

Det er vigtigt at skelne mellem at identificere en sårbarhed og at oprette et forslag til at rette den.

En sikkerhedsadvarsel medfører ikke nødvendigvis, at Dependabot kan oprette en fungerende opdatering.

Forudsætninger

I skal have gennemført øvelse 38–41.

Det betyder, at I har:

- En fork af undervisningsprojektet på GitHub.
- Projektet klonet lokalt.
- En fungerende GitHub Actions-pipeline.

- Automatiseret build og unit tests.
- NuGet Audit.
- Statisk sikkerhedsanalyse med CodeQL.

I skal fortsætte med det samme repository.

I skal ikke oprette et nyt repository eller et nyt CI-workflow.

Del 1 – Undersøg den eksisterende sårbare afhængighed

Vi starter med den afhængighed, som I tidligere undersøgte med NuGet Audit.

Trin 1 – Åbn projektfilen

Åbn:

`OrderManagement.Api/OrderManagement.Api.csproj`

Find følgende afhængighed:

```
<PackageReference Include="Newtonsoft.Json" Version="12.0.1" />
```

Denne version er bevidst inkluderet i undervisningsprojektet, fordi den har kendte sikkerhedssårbarheder.

Trin 2 – Genbesøg NuGet Audit

Åbn en terminal i projektets rodmappe.

Kør:

```
dotnet package list --vulnerable --include-transitive
```

Undersøg resultatet.

Refleksion

1. Hvilken sikkerhedsrisiko er knyttet til den anvendte pakkeversion?
2. Hvad sker der, hvis der efter denne scanning offentliggøres en ny sårbarhed i en anden afhængighed?
3. Hvorfor kan det være en fordel at overvåge afhængigheder løbende frem for udelukkende under den lokale udvikling?

Del 2 – Aktiver Dependabot på GitHub

Vi skal nu undersøge og aktivere GitHubs funktioner til overvågning af afhængigheder.

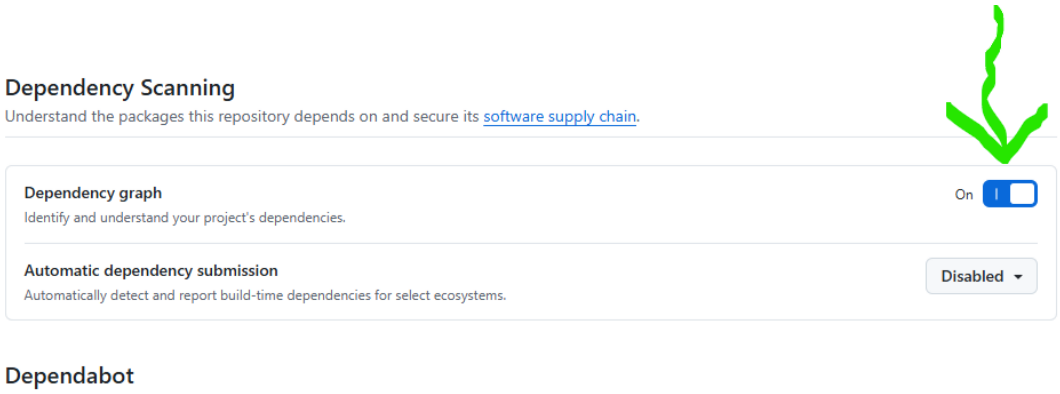
Trin 1 – Åbn repositoryets sikkerhedsindstillinger

1. Åbn jeres fork på GitHub.
2. Vælg **Settings**.
3. Find **Advanced Security** under repositoryets sikkerhedsindstillinger.

Placeringen af menupunkterne kan variere lidt afhængigt af GitHubs brugergrænseflade.

Trin 2 – Kontroller Dependency graph

Kontroller, at repositoryets **Dependency graph** er aktiveret.



Dependency graph anvendes til at registrere projektets afhængigheder og deres indbyrdes relationer.

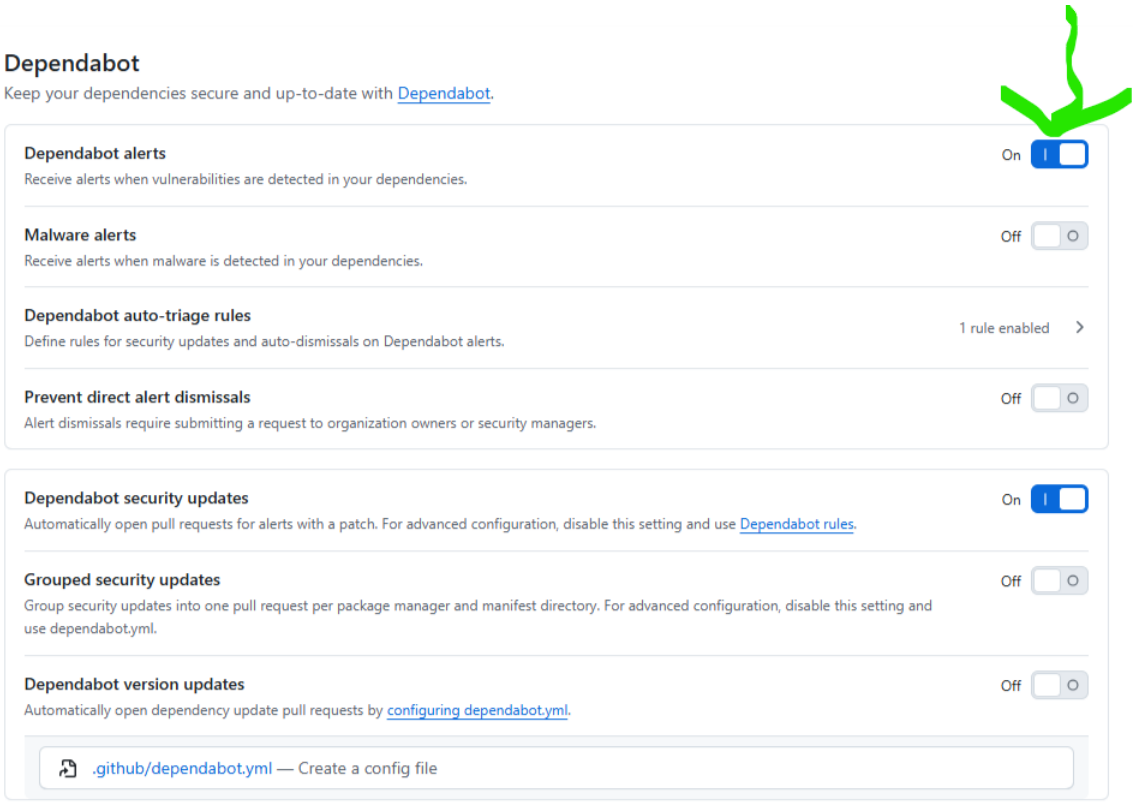
Den er en vigtig forudsætning for Dependabots sikkerhedsfunktioner.

Trin 3 – Kontroller Dependabot alerts

Find indstillingen:

Dependabot alerts

Aktiver den, hvis den ikke allerede er aktiveret.

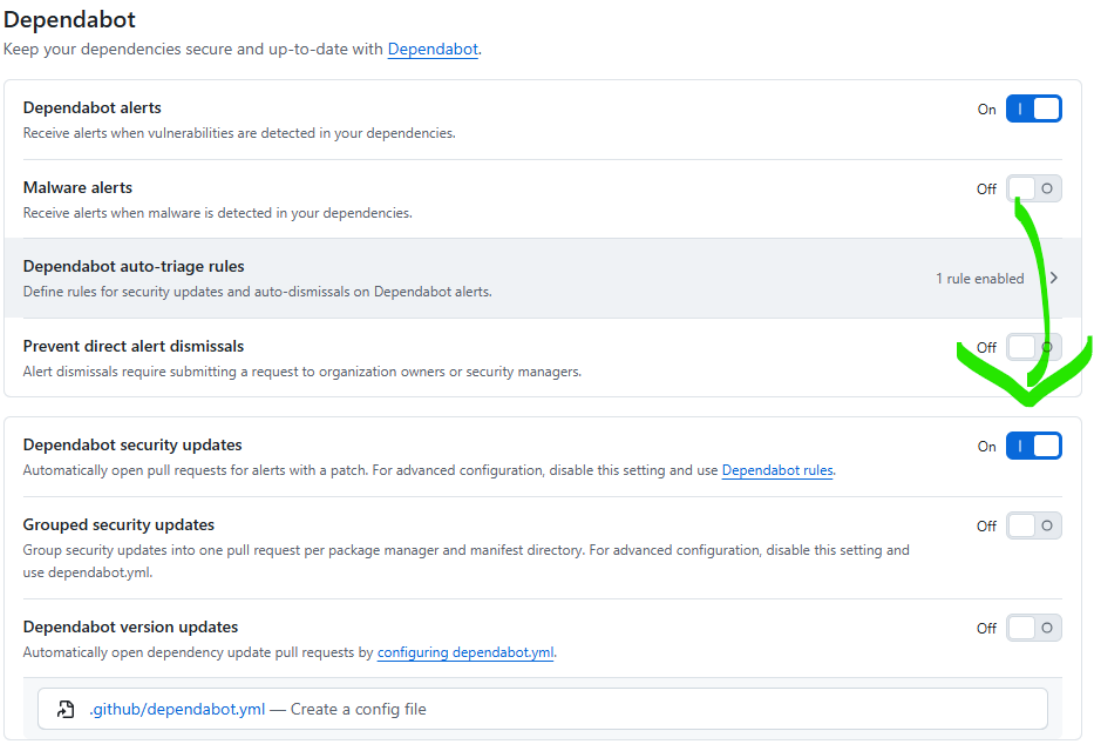


Dependabot alerts anvendes til at rapportere kendte sikkerhedssårbarheder i repositoryets afhængigheder.

Trin 4 – Aktiver Dependabot security updates

Find indstillingen:

Dependabot security updates



Aktiver den, hvis det er muligt, og den ikke allerede er aktiv.

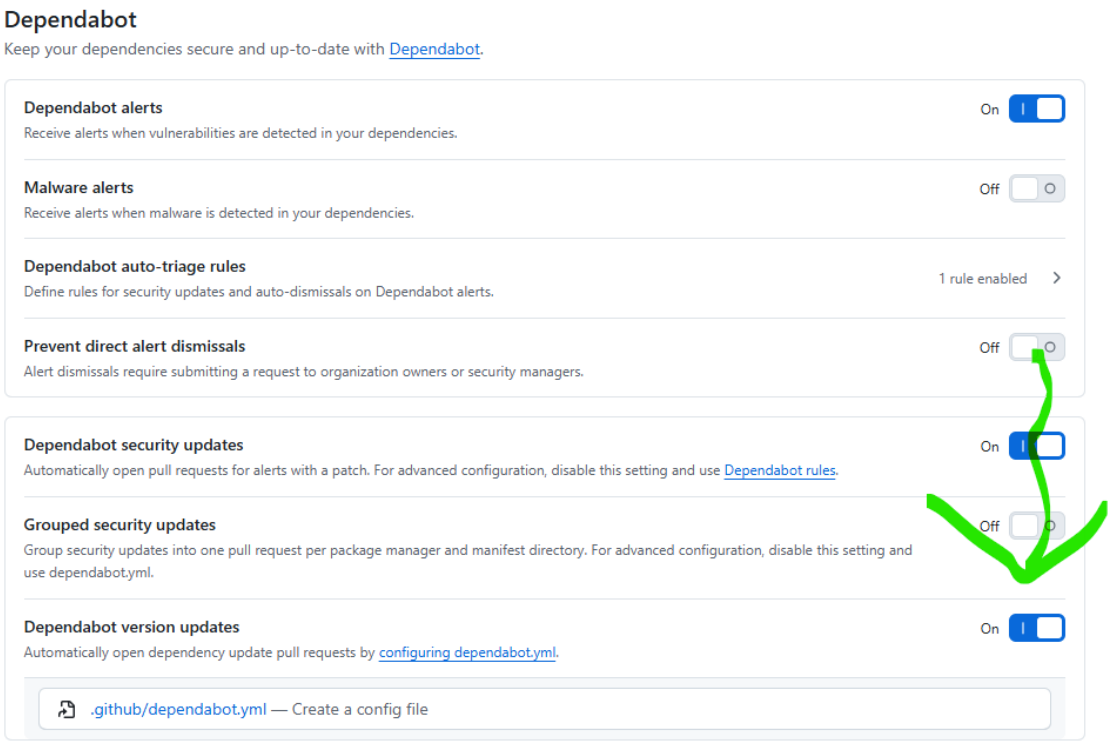
Denne funktion forsøger at oprette pull requests med opdateringer, der kan rette kendte sårbarheder.

På nogle offentlige repositories kan funktionen allerede være aktiveret, uden at den kan ændres manuelt.

Trin 5 – Aktiver Dependabot version updates

Find indstillingen:

Dependabot version updates



Aktiver den, hvis den ikke allerede er aktiv.

Dette trin er særligt vigtigt, fordi I arbejder med et repository, som er oprettet ved hjælp af en fork.

På forkede repositories er versionsopdateringer ikke nødvendigvis aktiveret, selvom der findes en `dependabot.yml`-fil.

Hvis GitHub tilbyder at oprette en standardkonfiguration, kan I i stedet oprette konfigurationen lokalt i næste del.

Kontrol

Undersøg, hvilke Dependabot-funktioner der er aktiveret på jeres repository.

Refleksion

1. Hvad er forskellen mellem Dependabot alerts og Dependabot security updates?
2. Hvorfor kræver versionsopdateringer en konfiguration, som beskriver, hvad der skal overvåges?
3. Hvorfor er det relevant, at ejeren af en fork selv aktiverer automatiserede opdateringer?

Del 3 – Konfigurer Dependabot lokalt

Vi skal nu oprette en konfigurationsfil, som fortæller Dependabot, hvilke afhængigheder der skal overvåges, og hvor ofte der skal kontrolleres efter nye versioner.

Trin 1 – Opret konfigurationsfilen

Åbn jeres lokale projekt.

Opret følgende fil:

`.github/dependabot.yml`

Filen skal placeres i `.github`-mappen og ikke i `.github/workflows`.

Trin 2 – Indsæt konfigurationen

Indsæt følgende YAML-konfiguration:

```yaml
version: 2

updates:
  - package-ecosystem: "nuget"
    directories:
      - "/OrderManagement.Api"
      - "/OrderManagement.Tests"
    schedule:
      interval: "weekly"
    open-pull-requests-limit: 5
```

Gem filen.

Vi har angivet de to projektmapper eksplicit, fordi deres `.csproj`-filer indeholder NuGet-afhængigheder.

Forklaring af konfigurationen

`version: 2`

Angiver versionen af Dependabots konfigurationsformat.

Dette er ikke versionen af selve Dependabot.

`updates`

Indeholder de pakkeøkosystemer, som Dependabot skal overvåge.

`package-ecosystem: "nuget"`

Angiver, at Dependabot skal overvåge .NET- og NuGet-afhængigheder.

`directories`

Angiver de mapper, hvor Dependabot skal undersøge projektfiler og afhængigheder.

I vores projekt ligger NuGet-referencerne i:

- `/OrderManagement.Api`
- `/OrderManagement.Tests`

`schedule.interval: "weekly"`

Angiver, at Dependabot skal kontrollere efter almindelige versionsopdateringer hver uge.

Det betyder ikke, at Dependabot nødvendigvis opretter en pull request hver uge.

Hvis der ikke findes relevante opdateringer, kan kontrollen gennemføres uden at oprette en pull request.

`open-pull-requests-limit: 5`

Begrænser antallet af åbne pull requests for almindelige versionsopdateringer til fem i denne konfiguration.

Begrænsningen er ikke en generel garanti for, hvor mange sikkerhedsopdateringer Dependabot kan oprette.

Refleksion

1. Hvorfor skal Dependabot vide, hvilket package ecosystem der anvendes?
2. Hvad er formålet med at angive de mapper, som indeholder projektets afhængigheder?
3. Hvilke fordele og ulemper kan der være ved at kontrollere efter opdateringer dagligt frem for ugentligt?
4. Hvorfor kan det være relevant at begrænse antallet af åbne pull requests?

Del 4 – Commit og push Dependabot-konfigurationen

Vi skal nu overføre konfigurationen til GitHub.

Åbn en terminal i projektets rodmappe.

Kør:

```
git add .github/dependabot.yml
git commit -m "Configure Dependabot dependency updates"
```

```
git push origin main
```

Dependabot læser konfigurationen fra repositoryets default branch, som i dette projekt er `main`.

Forventet resultat

Filen skal nu være tilgængelig på GitHub:

`.github/dependabot.yml`

GitHub kan derefter anvende konfigurationen til løbende at undersøge projektets NuGet-afhængigheder.

Kontrol

Åbn filen på GitHub, og kontroller, at YAML-indrykningen og de angivne mapper er korrekte.

Del 5 – Undersøg Dependabot alerts

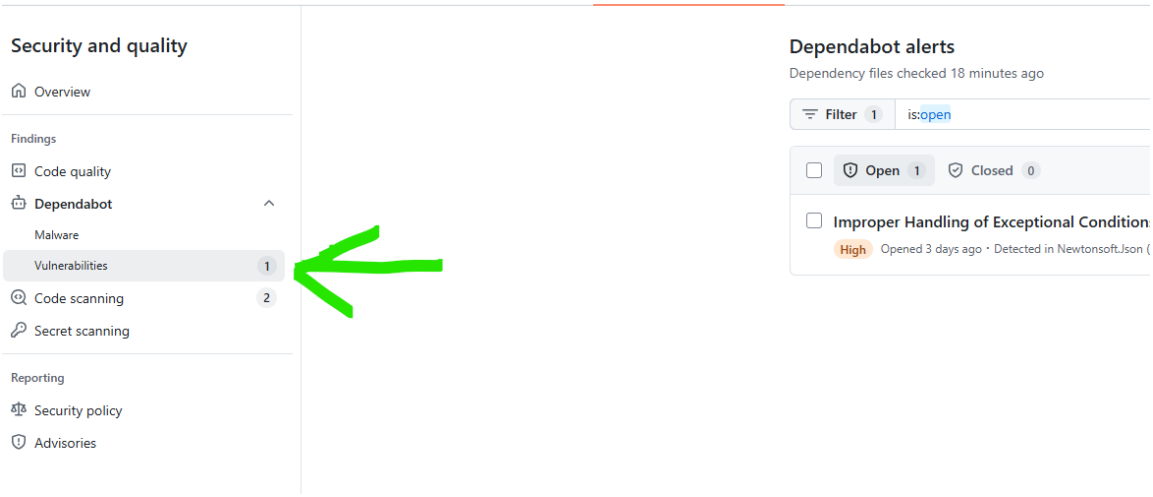
Vi skal nu undersøge, om GitHub har identificeret kendte sikkerhedssårbarheder i projektets afhængigheder.

Trin 1 – Åbn sikkerhedsadvarslerne

Åbn jeres repository på GitHub.

Gå til repositoryets **Security and quality**-område, og find:

Dependabot -> vulnerabilities



Undersøg de tilgængelige advarsler.

Trin 2 – Find den sårbare afhængighed

Undersøg, om Dependabot har rapporteret en sårbarhed i:

`Newtonsoft.Json` version `12.0.1`.

Hvis advarslen er tilgængelig, skal I åbne den og undersøge:

- Den berørte pakke.
- Den installerede version.
- Sårbarhedens alvorlighedsgrad.
- Referencen til sikkerhedsadvarslen.
- Oplysninger om eventuelle rettede versioner.

Trin 3 – Sammenlign med NuGet Audit

Sammenlign Dependabots advarsel med resultatet fra NuGet Audit i øvelse 40.

Undersøg, om værktøjerne rapporterer den samme sårbare afhængighed.

Forventet resultat

Dependabot forventes at kunne identificere den kendte sårbare NuGet-afhængighed, når GitHubs dependency graph og sikkerhedsadvarsler er korrekt aktiveret og opdateret.

Det præcise antal advarsler afhænger af GitHubs aktuelle sårbarhedsoplysninger og projektets registrerede afhængigheder.

Refleksion

1. Hvad er forskellen mellem at opdage sårbarheden gennem NuGet Audit og gennem Dependabot alerts?
2. Hvorfor kan en sikkerhedsadvarsel være relevant, selvom der ikke er foretaget nye commits i projektet?
3. Hvilke oplysninger skal en udvikler bruge for at vurdere, om en afhængighed bør opdateres?

Del 6 – Undersøg Dependabots opdateringsforslag

Dependabot kan oprette pull requests, når der findes relevante opdateringer.

Vi skal nu undersøge, hvordan disse forslag indgår i den eksisterende udviklingsproces.

Trin 1 – Undersøg Dependabots opdateringsstatus

Åbn jeres repository på GitHub.

Naviger til:

Insights → Dependency graph → Dependabot

Her kan I undersøge, hvilke afhængigheder Dependabot er konfigureret til at overvåge.

Undersøg også eventuelle logfiler fra Dependabots seneste kontrol.

Trin 2 – Undersøg pull requests

Åbn fanen:

Pull requests

Undersøg, om der er oprettet forslag fra Dependabot.

Hvis der findes en relevant pull request, skal I åbne den.

Undersøg:

- Hvilken afhængighed foreslås opdateret?
- Hvilken version anvendes aktuelt?
- Hvilken ny version foreslås?
- Hvilke filer ændres?
- Indeholder pull requesten oplysninger om sikkerhed eller kompatibilitet?

Samlet refleksion

Besvar følgende spørgsmål med udgangspunkt i jeres egne observationer:

1. Hvorfor er løbende overvågning af tredjepartsafhængigheder vigtig?
2. Hvorfor skal sikkerhedsopdateringer stadig testes og vurderes?
3. Hvordan kan de forskellige automatiserede sikkerhedskontroller bidrage til en mere sikker softwareudviklingsproces?

Resultat og evaluering

Øvelsen er gennemført, når I kan demonstrere følgende:

- I har undersøgt og aktiveret de relevante Dependabot-funktioner.
- I har oprettet `.github/dependabot.yml`.
- I har konfigureret overvågning af NuGet-afhængigheder.
- I har committed og pushed konfigurationen til GitHub.
- I har undersøgt Dependabot alerts.
- I kan forklare, hvordan Dependabot og CI-pipelinen understøtter hinanden.

Det er ikke et krav, at Dependabot har oprettet en pull request, hvis der ikke findes en relevant opdatering på tidspunktet for øvelsen.

Det centrale er, at I kan **konfigurere og undersøge automatiseret dependency management og forklare, hvordan opdateringer kan indgå i en sikker udviklingsproces**.

Afslutning på CI/CD- og DevSecOps-forløbet

I øvelse 38–41a har I gradvist udvidet en simpel CI-pipeline med forskellige kontroller.

I begyndte med automatisk build og tilføjede derefter unit tests, dependency scanning og statisk sikkerhedsanalyse.

Til sidst har I konfigureret løbende overvågning og forslag til opdatering af tredjepartsafhængigheder.

Samlet har I arbejdet med:

1. **Continuous Integration:** Automatisk build ved kodeændringer.
2. **Automatiseret test:** Kontrol af applikationens funktionalitet.
3. **Dependency scanning:** Identifikation af kendte sårbarheder.
4. **Static Application Security Testing:** Analyse af applikationens kildekode.
5. **Dependency management:** Løbende overvågning og vedligeholdelse af afhængigheder.

Dette illustrerer, hvordan automatiserede sikkerhedskontroller kan integreres i softwareudviklingsprocessen.

Samtidig har I set, at automatisering ikke i sig selv garanterer sikker software.

Resultaterne skal fortsat fortolkes, prioriteres og anvendes som grundlag for sikkerhedsmæssige beslutninger.

Litteratur og dokumentation

- *Secure by Design*, kapitel 8 – Leveraging your delivery pipeline for security.
- GitHub – Dependabot quickstart
- GitHub – About the dependabot.yml file
- GitHub – Configuring Dependabot alerts
- GitHub – Configuring Dependabot version updates
- GitHub – Configuring Dependabot security updates

Last update: 2026-11-10 20:57:39