

▮ Øvelse 40 – Detektering af sårbare pakker i build job

▮ Information

I denne øvelse skal der arbejdes med detektering af sårbare pakker i build-pipelinen. Formålet er at vise, hvordan detektering af sårbare pakker kan håndteres i en build-pipeline.

Detektering af sårbare pakker bør egentlig ikke udføres af build-pipelinen, men af repository-hosting-tjenesten. GitHub har [Dependabot](#), der løbende kan scanne et repository for sårbare pakker (pakker med en eksisterende CVE). I praksis er det bedst at anvende repository'ets indbyggede sårbarhedsscanner såsom Dependabot. Men Dependabot kan, med det mindste interval, kun eksekveres én gang dagligt, hvilket ikke gør den særligt anvendelig i en undervisningskontekst. Derfor anvendes denne ikke i øvelsen.

Løbende automatiske scanninger for sårbare pakker er bestemt den bedste løsning, men det kan suppleres med sårbarhedstjek, når der integreres ny kode i repositoret. Det skal implementeres i denne øvelse.

▮ OBS!

`dotnet list package --vulnerable --include-transitive` returnerer statuskode 0, uanset om den finder sårbare pakker eller ej. Dette gør den mindre hensigtsmæssig i et GitHub Actions build-job, da processen kun afbrydes, hvis applikationen ikke returnerer statuskode 0. Derfor har undertegnede været nødt til at være kreativ for at få dette til at fungere. Løsningen bør kun anvendes i undervisningen og kun med det formål at vise konceptet!

Prøv at se, om du kan finde ud af, hvilket potentielle problem løsningen kan skabe, når du løser opgaven.

▮ Instruktioner

1. ▮ Kopier folderen med projektet `SqlInjection` fra [dette repo](#) ind i lokal folderen til det repo som er anvendt i de to forrige øvelse (det er blot folderen med projektet `SqlInjection` som skal kopieres ind). (Undlad og forsøg at eksever projektet, da det er bygget med en ældre .Net version)
2. ▮ Tilføj projektet til solution-filen:
3. ▮ I GitHub Actions konfigurationsfilen, tilføj følgende job som det første:

```
scan-for-vulnerable-packages:  
  runs-on: windows-latest
```

```
steps:
  - uses: actions/checkout@v4
  - name: Check for vulnerable packages
    working-directory: ./SqlInjection
    run: |
      dotnet restore
      New-Item vulnerabilities.txt
      dotnet list package --vulnerable --include-transitive | Tee-Object
vulnerabilities.txt
      $matches = Get-Content vulnerabilities.txt | Select-String -Pattern
"has the following vulnerable packages"
      if ($matches.Length -gt 0) {echo 'Vulnerable packages detected';
exit 1}
```

4. □ Push koden til GitHub.

5. □ Observer build-jobbet. Hvilke sårbarheder bliver der detekteret? Afbrydes processen? ?

□ Automatisk scanning af brugen af eksterne tredjepartspakker med kendte CVE'er sikrer, at udviklere ikke uvidende anvender pakker med kendte sårbarheder. I eksemplet eksekveres scanningen dog kun, når der tilføjes ny kode til repositoret, hvilket er u hensigtsmæssigt. Der bør kontinuerligt og automatisk scannes for brug af tredjepartspakker med CVE'er. En ny CVE kan blive opdaget, men anvendelsen af pakken med CVE'en i egen software opdages kun, hvis scanningen foretages jævnligt, og ikke kun ved ændringer i det eksisterende repository. Løbende automatiske scanninger, som f.eks. [Dependabot](#) tilbyder, sikrer, at nye CVE'er bliver opdaget, selvom der ikke er ændret noget i kildekoden. Man kan f.eks. overveje, hvad risikoen ville være, hvis man ikke foretog ændringer i sin kildekode i et helt år. Hvornår ville de nye sårbarheder i tredjepartspakker så blive opdaget?

□ Oprydning

For at rydde op, kan du blot udkommentere det job, du tilføjede tidligere i build pipelinens konfigurationsfil.

Lad dog `SqlInjection`-projektet blive i folderen sammen med de andre projekter, dette vil gøre det nemmere for dig, hvis du senere skal genbesøge øvelsen. Og folderen skal også benyttes i den næste opgave.

Når du har udkommenteret jobbet, så husk at push til GitHub igen, og sikre dig at build-pipelinen igen kører succesfuldt igennem.

□ Refleksionsspørgsmål

1. □ Hvorfor er det vigtigt at detektere sårbare tredjepartspakker i softwareudvikling?
2. □ Hvilke risici kan opstå, hvis man ikke løbende scanner og opdaterer afhængigheder?
3. □ Hvordan kan automatiserede scanninger integreres bedst muligt i en CI/CD pipeline?

4. ¶ ⚡ Hvad kan være begrænsninger eller udfordringer ved at anvende værktøjer som `dotnet list package --vulnerable` i build-jobs?
5. ¶ Hvordan kan brugen af services som Dependabot supplere denne pipeline-baserede scanning?

¶ *Detektion og håndtering af sårbare tredjepartsbiblioteker er afgørende for at opretholde sikkerheden i moderne softwareudvikling og bør automatiseres, hvor det er muligt.*

Last update: 2025-10-30 12:07:19