Runbook: Kwetsbaarheidsbeheer (CVE) & Incident Response

Doel en Basisprincipe

Dit document beschrijft onze standaardwerkwijze voor het voorkomen, detecteren, analyseren en oplossen van beveiligingskwetsbaarheden (CVE's) in onze software en containers.

> **Ons gouden principe (Assume Breach):** Zodra een relevante CVE wordt gemeld, behandelen we onze systemen alsof ze daadwerkelijk kwetsbaar zijn, totdat we technisch hebben aangetoond dat dit niet zo is. Elke analyse en beslissing leggen we vast voor onze audit-trail.

Deel 1: De Fundering (Proactieve Hygiëne)

Voordat er een incident plaatsvindt, zorgen we dat ons aanvalsoppervlak zo klein mogelijk is en onze detectie sluitend is.

1.1 Dependency-beleid

- **Weten wat je gebruikt:** Voeg geen onnodige dependencies toe als de standaardbibliotheek van de taal het probleem ook oplost.
- **Cooldown-periode op updates:** Voor reguliere (niet-security) updates hanteren we een *cooldown* van minimaal 3 tot 7 dagen (bijv. via `minimumReleaseAge` in Renovate/Dependabot). Hiermee voorkomen we dat we direct slachtoffer worden van een *supply chain attack* in een net gepubliceerde package-versie.
- **Vuistregel voor "Functies kopiëren vs. Dependency importeren":**
  - **Wel overnemen (Vendor/Copy):** Heb je alleen een triviale hulpfunctie nodig van 15 regels code (bijv. simpele string-formatting of array-sortering)? Schrijf of kopieer deze dan zelf in plaats van een heel pakket binnen te halen.
  - **Nooit overnemen:** Kopieer **nooit** complexe of beveiligingsgevoelige functies (zoals parsers, HTML-sanitizers, validatie of cryptografie) los in de eigen codebase. Gekopieerde code wordt onzichtbaar voor onze CVE-scanners (*Shadow Dependency*), waardoor we geen meldingen meer krijgen als er later een lek in die functie wordt ontdekt.

1.2 Container-beleid

- **Slimme basis-images:** Gebruik waar mogelijk *distroless*, Chainguard of minimale (Alpine) base-images.
- **Minimale rechten:** Productie-containers bevatten geen onnodige OS-packages, geen shell (`sh` / `bash`), geen package managers (`apt` / `apk`) en draaien altijd als **non-root user**.

1.3 Continue Detectie & SBOM

- **Build-time scanning:** Elke CI/CD-pipeline draait taal-specifieke audits (zoals `npm audit`, `dotnet list package --vulnerable`, `pip-audit`) én een containerscan met **Trivy** of **Grype**.
- **Automatische SBOM:** Bij elke productierelease wordt automatisch een **SBOM** (Software Bill of Materials, bijv. in CycloneDX-formaat) gegenereerd en opgeslagen.
- **Runtime / Dagelijkse scan:** Omdat applicaties soms weken draaien zonder nieuwe build, worden onze bestaande container-images en SBOM's dagelijks automatisch gescand op nieuw gepubliceerde CVE's.

Deel 2: Het Incident Stappenplan (Reactief)

Volg deze 4 stappen zodra een scanner, externe melding of security-bulletin een (kritieke) CVE signaleert.

Stap 1: Vastleggen (Audit Trail starten)

Voordat je begint met onderzoeken, maak je **altijd** een Jira-ticket aan.

- **Waarom:** Dit ticket is ons formele bewijs (voor klanten, hosting en compliance zoals ISO/NIS2) dat we de melding hebben opgepakt, geanalyseerd en afgehandeld.
- **Inhoud ticket:** CVE-nummer (bijv. `CVE-2026-XXXX`), getroffen component/versie, datum van signalering en de naam van de behandelaar.

Stap 2: Impact bepalen (Triage)

Ga ervan uit dat we geraakt zijn, tot je het tegendeel bewijst. Loop de volgende checklist af:

1. **Aanwezigheid checken (SBOM):** Raadpleeg de SBOM's en dependency-trees. In welke projecten, branches en draaiende omgevingen zit deze specifieke versie van de dependency (direct of transitief)?
2. **CVE inhoudelijk doornemen:** Lees de officiële CVE-beschrijving en de advisories van de leverancier (GitHub Security Advisory / NVD). Onder welke condities treedt het lek op?

3. **Bereikbaarheid (Reachability):** Zit de kwetsbare functie daadwerkelijk in een codepad dat door onze applicatie wordt aangeroepen? Is dit onderdeel bereikbaar vanaf het internet of alleen intern?
4. **Dreigingsniveau bepalen (CVSS + EPSS):**
    - Hoe hoog is de theoretische ernst (CVSS-score)?
    - Is er al actieve exploit-code in omloop (check de **EPSS-score** of CISA KEV-lijst)?

> **Beslispunt na Stap 2:**
>
> - **Geen impact (Niet geraakt / Niet bereikbaar):** Documenteer in het Jira-ticket *waarom* de CVE niet exploiteerbaar is in onze context. Sluit het ticket af met status *Niet van toepassing / Geaccepteerd (False Positive)*. Je bent nu klaar.
> - **Wel impact (Of twijfel):** Ga direct door naar Stap 3.

Stap 3: Misbruik voorkomen & Oplossen

Afhankelijk van de ernst en of er al een patch beschikbaar is, kiezen we de snelste route om het risico weg te nemen:

- **Optie A: Directe isolatie (Als er actief misbruik is en nog geen code-fix)**
    - **Tijdelijke mitigatie (Virtual Patching):** Blokkeer het aanvalspatroon in de WAF/Reverse Proxy, pas netwerkregels aan, of schakel de kwetsbare functionaliteit uit via een *feature flag* of configuratie.
    - **Noodrem ("Alles uitzetten"):** Als een kritieke omgeving actief gevaar loopt (bijv. een openstaande Remote Code Execution zonder mogelijke WAF-blokkade) en data dreigt te lekken, haal de betreffende service of omgeving gecontroleerd offline in afstemming met de verantwoordelijke.
- **Optie B: Code-herstel (Spoed-release / Hotfix)**
    - **Pad 1 - Upgraden:** Is er een gepatchte versie van de dependency beschikbaar? Werk deze bij, draai de test-suite en zet een spoed-release (hotfix) door naar productie.
    - **Pad 2 - Downgraden:** Zit het lek alleen in een recent geïntroduceerde versie en is er nog geen fix (of zorgt een upgrade voor *breaking changes*)? **Downgrade** de dependency naar de laatst bekende veilige versie en maak een spoed-release.

> *Controleer na de deployment met Trivy/Grype en een nieuwe SBOM of de kwetsbare versie daadwerkelijk uit de productie-container is verdwenen.*

Stap 4: Stakeholders informeren & Afronden

Heldere communicatie en dossiervorming sluiten het incident af:

1. **Intern & Hosting:** Informeer interne stakeholders (development, management, support) en indien relevant de hostingpartij over de impact en het tijdstip van de (spoed)release of tijdelijke downtime.
2. **Klanten informeren:**
    - Moeten klanten zelf actie ondernemen (bijv. een update installeren of wachtwoorden roteren) of is er sprake van impact op beschikbaarheid/data? Informeer hen proactief en transparant.
    - Betreft het een achterliggende fix zonder impact of actie voor de klant? Communiceer dit pas **nadat** de fix veilig op productie staat (bijvoorbeeld via de release notes).
3. **Bewijslast afronden in Jira:** Vul het Jira-ticket aan met:
    - De genomen maatregelen (link naar de Pull Request / Commit en Release-versie).
    - Tijdstip waarop de fix live is gegaan.
    - Eventuele afspraken om herhaling te voorkomen.