ARTRAGIS

Geste de savoir

CycloneDX et dépendency track : luter contre les CVE importées de l'extérieur

18 décembre 2022

Table des matières

Introduction

L'OWASP publie une liste de dix typologies de failles de sécurité, appelé le OWASP TOP 10 . Celle qui nous intéresse aujourd'hui est la numéro 6 du classement 2021 "Les composants vulnérables et non mis à jour .

La question qui nous intéresse sera donc "Comment faire pour s'assurer que nos projets n'incluent pas de dépendances vulnérables".

En particulier, nous allons nous concentrer sur les projets Java.

1. Dependency check, les bases de CVE

1.1. Dependency Check

L'OWASP fournit un ensemble assez conséquents d'outils pour monitorer les projets, notamment vos dépendances.

Le plus ancien est dependency-check et notamment le plugin maven qui l'accompagne[1].

L'intégration à maven est aussi simple que verbeuse à un détail près que nous verrons plus tard.

Le dependency-check tournera sur le *lifecycle* `site` et fournira un rapport html que vous pourrez ensuite exploiter.

1. En vrai il a des plugins pour tous les outils de builds liés à java, y compris ant et gradle. Mais ne sachant pas les utiliser et ayant choisi de me limiter à un billet pour l'instant, ces plugins ne seront évoqués que dans cette note.

Pour avoir dans le cadre professionnel implémenté un script qui interprète le rapport HTML pour extraire les données et les mettre dans une base de données pour ensuite faire un rapport synthétique, je peux vous dire que le HTML produit est au mieux *pas optimisé*. Ceci dit il est parsable sans erreur par n'importe quelle lib, comme `BeautifulSoup4` si vous aimez python.

```
<reporting>
        <plugins>
            <plugin>
                <groupId>org.owasp</groupId>
                <artifactId>dependency-check-maven</artifactId>
                <version>7.4.1</version>
                <reportSets>
                    <reportSet>
                        <reports>
                            <report>check</report>
                        </reports>
                    </reportSet>
                </reportSets>
            </plugin>
        </plugins>
    </reporting>
```

Listing 1 – Le code minimal à mettre dans votre pom pour faire fonctionner dépendency check

Vous pouvez désormais lancer `mvn site` ou encore `mvn org.owasp:dependency-check-maven:check` et il produira un fichier html tel que celui donné en exemple sur la documentation officielle .

C'est moche mais ça fait le taf, vous avez toutes les informations nécessaires.

Par contre je vous l'annonce tout de suite, ça peut rapidement prendre du temps: les bases de données doivent être chargées et si votre CI/CD gère mal les caches[2], ce simple téléchargement peut rapidement vous prendre 5/10 minutes où le plugin met un coeur complet à 100%.

1.2. Les bases de CVE

Je profite de ce petit moment salé à propos du téléchargement des bases de CVE pour vous expliquer un peu ce que sont ces bases.

Ce sont simplement des listes de numéros de failles et les informations qui vont avec:

- numéro de faille (e.g CVE-2021–44228)
- produit affecté (exemple log4j)
- version de départ (exemple 2 2.0-beta9)
- version de fin (exemple 2.16.0)
- sévérité au format LOW/MEDIUM/HIGH/CRITICAL (par exemple Log4Shell était critique.

2. Je pense à toi bitbucket pipelines et tes surprises surprenantes sur la gestion de ce qui est caché, pour combien de temps etc.

— sévérité au format ”note /10” (log4shell était 10)

La base la plus grande (National Vulnerability Database) est au NIST, et on peut par exemple voir ce genre d'information:

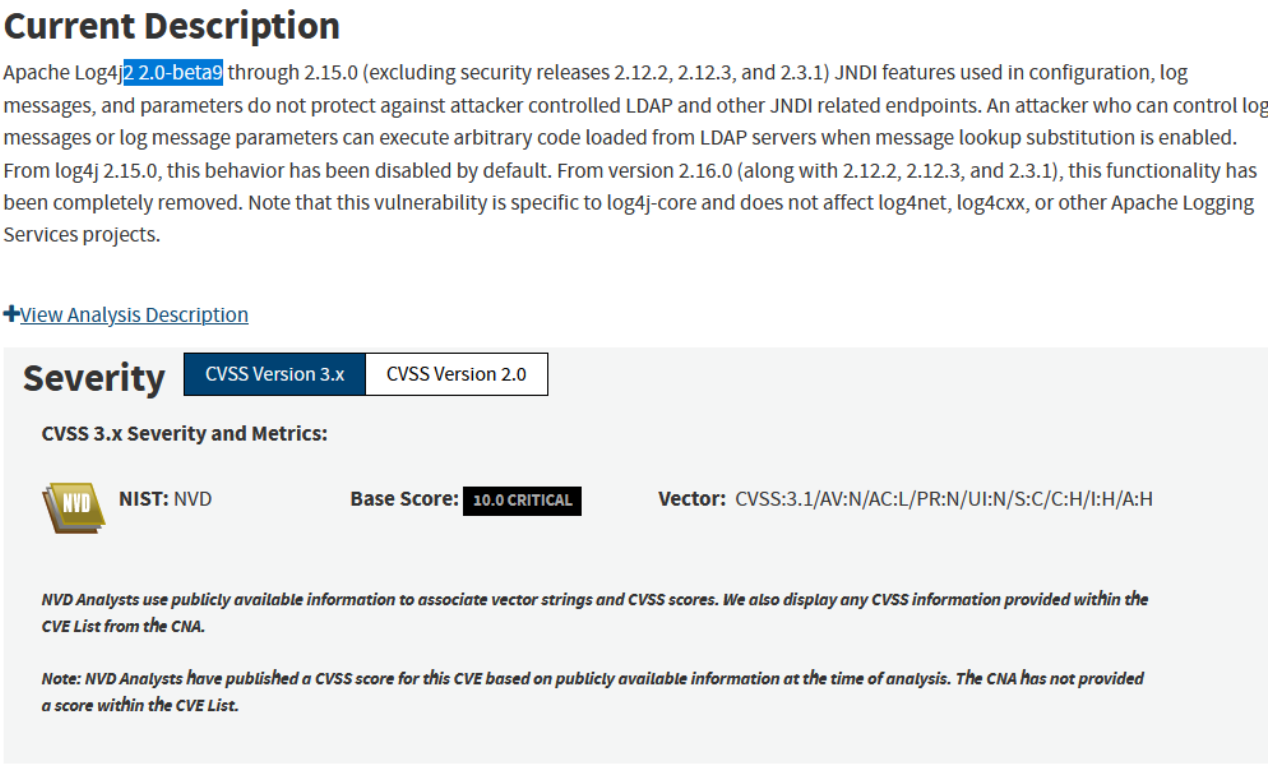


Figure 1.1. – La NVD

Mais ces derniers temps github, mais aussi Google pour ne citer qu'eux ont eu le droit de numéroter des CVE et donc de publier leurs bases, ce qui multiplie les sources pour le plugin dependency check.

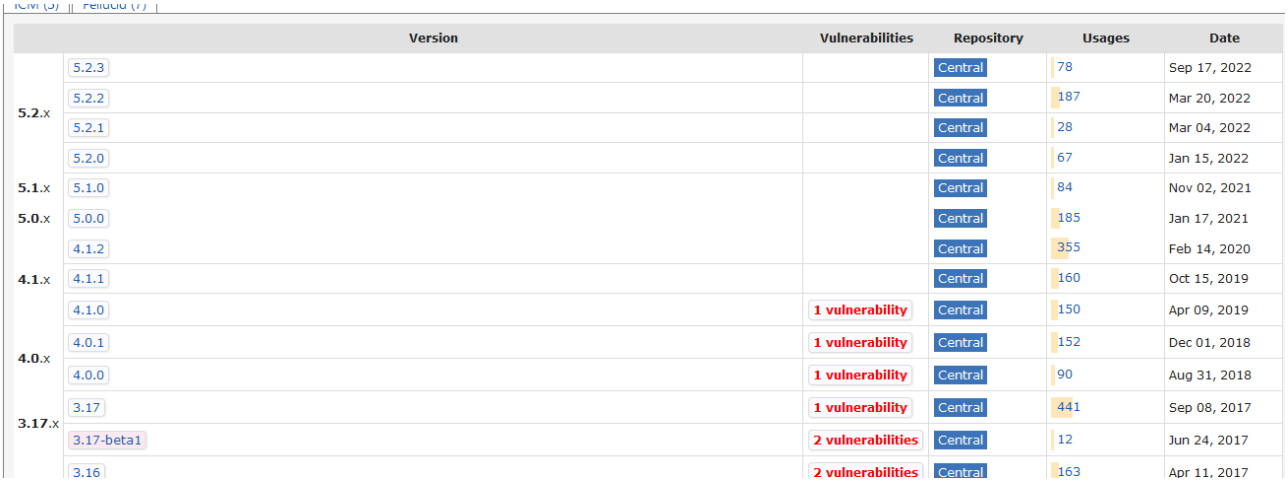
1.3. Quelques retours d'expérience sur dependency-check

L'outil fait le job pour lequel il a été conçu et... c'est tout. J'ai déjà évoqué le fait que le rapport est complexe à exploiter pour la machine, mais c'est principalement car malgré une structure ”globalement” stable, certaines bases de données ne donnent pas le résultat dans le même format et dependency-check ne fait pas vraiment le travail d'uniformiser l'output. Tout ce qui compte est que le HTML soit lisible, pas que le rapport soit exploitable par un outil d'automatisation.
D'ailleurs on remarquera qu'à cause de la différence entre les formats des bases de données, dependency-check ne met pas en avant la version qui corrige la CVE. Il faut donc lire le descriptif pour comprendre.
D'ailleurs dependency check ne se connectant pas aux dépôts maven (ou des autres techno), il est incapable de savoir que des versions plus récente existe. Il faut donc le croiser avec les rapports des plugins dependency-upgrade ou encore dependency-convergence.
D'ailleurs c'est aussi une faiblesse de dependency-check: il n'est pas rare que des dépendances transitives existent mais que vous même dépendiez d'une version non vulnérable. C'est une erreur de *dependency convergence* mais dependency-check ne se soucie pas que par exemple la version vulnérable ne sera même pas téléchargée: il considère que vous en dépendez.

Enfin, la syntaxe des fichiers d'exclusion est… verbeuse et parfois un peu complexe à gérer. Ce qui fait que certains faux positifs restent ad vitam dans les rapports car les retirer prend plus de temps que de les ignorer.

1.4. Petite note pour mvnrepository.com

Il y a quelques temps le site mvnrepository.com s'est mis à afficher les vulnérabilités des dépendances directement.



	Version	Vulnerabilities	Repository	Usages	Date
5.2.x	5.2.3		Central	78	Sep 17, 2022
	5.2.2		Central	187	Mar 20, 2022
	5.2.1		Central	28	Mar 04, 2022
	5.2.0		Central	67	Jan 15, 2022
5.1.x	5.1.0		Central	84	Nov 02, 2021
5.0.x	5.0.0		Central	185	Jan 17, 2021
	4.1.2		Central	355	Feb 14, 2020
4.1.x	4.1.1		Central	160	Oct 15, 2019
	4.1.0	1 vulnerability	Central	150	Apr 09, 2019
4.0.x	4.0.1	1 vulnerability	Central	152	Dec 01, 2018
	4.0.0	1 vulnerability	Central	90	Aug 31, 2018
3.17.x	3.17	1 vulnerability	Central	441	Sep 08, 2017
	3.17-beta1	2 vulnerabilities	Central	12	Jun 24, 2017
	3.16	2 vulnerabilities	Central	163	Apr 11, 2017

FIGURE 1.2. – Exemple avec Apache POI

Cependant mvnrepository.com n'affiche pas TOUTES les vulnérabilités, en comparant avec les rapports fournis par dépendency-check et cyclonedx (ça vient), ma principale hypothèse est que ce site ne regarde pas toutes les bases.

2. cyclonedx, BOM BOM BOM BOM

Nous l'avons vu au dessus, le rapport fourni par dependency-check est exploitable par l'humain mais pas par la machine. Et c'est un soucis, car en terme de réactivité, une machine surpasse l'humain.
Pour pouvoir être plus automatisable, le mieux c'est probablement de voir plus large que juste les CVE. Justement, il existe depuis longtemps dans le monde de l'entreprise le concept de "Software Bill Of Material", souvent abrégé SBOM.
Le SBOM consiste à lister dans des formats intéropérables l'ensemble des risques associés aux composants logiciels :

- vulnérabilités
- licences
- copyright
- compatibilité…

Et c'est là qu'arrive CyclonDX. Il s'agit d'un système de SBOM créé par l'OWASP, toujours eux.
Comme pour dependency check, il y a un plugin maven . Le but de ce plugin est de produire un *Bill Of Material* au format XML ou JSON. `mvn org.cyclonedx:cyclonedx-maven-plugin:makeAggregateBom` suffira à le produire.
Le format étant structuré pour être traité par une machine, il suffit d'avoir un bon logiciel.

En parlant de bon logiciel…

3. Dependency-track pour tracker les dependency

L'OWASP (on ne les arrête plus), fournit donc un tool qui vous permettra de tirer le meilleur de votre *Bill Of Material*: dependency track. A ne pas confondre avec dependency-check, qui check et ne track pas.
dependency-track donc, est un service web autohébergeable qui se nourrit de votre BOM, des bases de données de vulnérabilités, des bases de licenses, des ressources déposées sur les principaux dépôts (pas que maven) et vous permet de monitorer l'état des lieux des dépendances de votre projets.
Comme c'est une interface Web, vous pouvez y mettre une authentification, avec une gestion par groupe. Ils sont compatibles avec les principaux SSO, donc pour les entreprises c'est assez cool.
Ensuite vous allez pouvoir suivre l'état de vos logiciels. Toutes les bases de données sont prises en compte.
Par contre il vous faudra tuner les choses car sinon trop de remontées inutiles sont faites, surtout si vous avez des dépendances optionnelles.
Surtout si vous n'êtes pas affectés par une vulnérabilité (dans le sens: vous avez bien la dépendance vulnérable mais vous n'utilisez pas la partie qui l'est), il suffit de le noter et de marquer votre logiciel comme non affecté. Le tout sera sauvegardé et n'importe qui pourra en tirer un rapport qui présente bien en terme administratif.