01-2017 | Frühling | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Java ist die beste Wahl

Einfacher programmieren
Erste Schritte mit Kotlin

Security
Automatisierte Überprüfung von Sicherheitslücken

Leichter testen
Last- und Performance-Test verteilter Systeme

4 191978 304903 01

iJUG Verbund

Der Unterschied von Java EE zu anderen Enterprise Frameworks

Kotlin ist eine ausdrucksstarke Programmiersprache, um die Lesbarkeit in den Vordergrund zu stellen und möglichst wenig Tipparbeit erledigen zu müssen

Innerhalb von Enterprise-Anwendungen spielen Sicherheits-Aspekte eine wichtige Rolle

Automatisierte Überprüfung von Sicherheitslücken in Abhängigkeiten von Java-Projekten

Johannes Schnatterer, Triology GmbH

Sicherheit in Software-Projekten ist ein schwer zu beherrschendes Thema, wie regelmäßig in den Medien erscheinende Berichte über Datenlecks beweisen. Für Software-Entwickler ist es schwer zu überblicken, welchen Einfluss ihre tägliche Arbeit an nicht unmittelbar sicherheitsrelevanten Themen auf die Sicherheit ihrer Anwendung hat.

Einen Überblick über die häufigsten Makel in Punkto Sicherheit bieten die Top Ten des Open Web Application Security Project (OWASP, *siehe „https://www.owasp.org/index.php/Top_10_2013-Top_10")*, eine Non-Profit-Organisation, die sich zum Ziel gesetzt hat, die Sicherheit im Web zu verbessern. Hier zeigt sich zudem, dass auch nicht selbst geschriebener Code Sicherheitslücken enthalten kann. Diesen kann man jedoch mit wenigen Schritten überprüfen. Der Artikel zeigt, wie man mit geringem Aufwand zumindest über die bekannten Sicherheitslücken in verwendeten Abhängigkeiten von Dritten auf dem Laufenden bleibt.

Grundlage bildet die National Vulnerability Database (NVD, siehe *„https://nvd.nist.gov")*, eine Art Datenbank für Sicherheitslücken. Sie wird gepflegt vom National Institute of Standards and Technology (NIST), eine mit der deutschen Physikalisch-Technischen Bundesanstalt (PTB) vergleichbare US-amerikanische Bundesbehörde, die auch bekannte Verschlüsselungsalgorithmen wie DES und AES standardisiert.

Die NVD beinhaltet unter anderem die Schwachstellen, die im Industrie-Standard „Common Vulnerabilities and Exposures" (CVE) erfasst werden. Die bereits erwähnte OWASP bietet eine Anwendung an, die automatisiert Jar-Dateien mit der NVD abgleicht. Dieser sogenannte „OWASP Dependency-Check" (DChc, *siehe „https://www.owasp.org/index.php/OWASP_Dependency_Check")* wurde für die Kommandozeile sowie für Ant, Maven, gradle, sbt, Jenkins und SonarQube implementiert.

Nachfolgend ein Lösungskonzept, um die Abhängigkeiten einer Java-Anwendung mithilfe von Maven und Jenkins regelmäßig auf neue Sicherheitslücken zu durchsuchen. Nach einmaliger Einrichtung sind keine weiteren Schritte mehr notwendig. Die Information über neu gefundene Sicherheitslücken erfolgt dann per E-Mail. Dabei kann es sich entweder um neu bekannt gewordene Sicherheitslücken in bestehenden Abhängigkeiten handeln oder um bereits bekannte Sicherheitslücken, die in neu hinzufügten Abhängigkeiten existieren.

Erster Schritt: Maven-Plug-in ohne Konfiguration verwenden

Eine erste Überprüfung der Abhängigkeiten der eigenen Anwendung kann ohne weitere Konfiguration in wenigen Sekunden angestoßen werden, beispielsweise mittels Maven-Plug-in (MVN, *siehe „http://jeremylong.github.io/DependencyCheck/dependency-check-maven/configuration.html")* durch „mvn org.owasp:dependency-check-maven:1.4.0:aggregate". Dabei gilt es zu beachten, dass „dependency-check-maven-plugin" zunächst eine lokale Kopie der CVE-Datenbank in Form einer H2-Datenbank herunterlädt und sie im lokalen Maven-Repository speichert. Aufgrund der Anzahl bekannter Sicherheitslücken resultiert dies in einer mehrere Hundert Megabytes umfassenden Datei. Das initiale Erzeugen kann, abhängig von verfügbarer Bandbreite und Rechenleistung, einige Minuten dauern. Nach erfolgreichem Abschluss steht der Bericht unter „target/dependency-check-report.html". Ein Beispiel für einen solchen Bericht findet man auf der Website des Projekts (DChck-Sample, *siehe „http://jeremylong.github.io/DependencyCheck/general/SampleReport.html")*.

Automatisierung mit Jenkins

Um ohne weiteres Zutun über neu gefundene Sicherheitslücken informiert zu werden, bietet sich eine regelmäßig durchgeführte, automatische Überprüfung an. OWASP bietet für den CI-Server Jenkins ein Plug-in an, das die Überprüfung durchführen und deren Ergebnisse auswerten kann. Unter anderem können die gefundenen Sicherheitslücken in Jenkins visualisiert, das Ergebnis eines Jobs von der Anzahl gefundener Sicherheitslücken abhängig gemacht und die Entwicklung der Anzahl gefundener Sicherheitslücken über die Builds in einem Diagramm dargestellt werden. Diese Auswertung ist in einer Post-Build-Action im Jenkins-Job konfiguriert. Die Überprüfung der Abhängigkeiten selbst kann durch einen durch das Jenkins-Plug-in bereitgestellten Build-Step oder per Maven durchgeführt werden.

Da die Überprüfung einige Zeit in Anspruch nimmt, sollte sie nicht bei jedem Commit/Push stattfinden. Ein Nightly-Build oder ein einmal pro Woche durchgeführter Build bieten sich daher an. Die Überprüfung per Maven Goal statt mit Jenkins-Build Step hat den Vorteil, dass die zentrale Konfiguration in der „pom.xml" erfolgt, wie in *Listing 1* gezeigt.

Mit dieser Konfiguration müssen in Jenkins nur noch die folgenden Maven Goals mit Parameter aufgerufen werden: „mvn clean install

```
<properties>
  <dependency-check-format>HTML</dependency-check-format>
</properties>
<build>
  <plugins>
     <plugin>
        <groupId>org.owasp</groupId>
        <artifactId>dependency-check-maven</artifactId>
        <version>1.4.0</version>
        <configuration>
           <format>${dependency-check-format}</format>
        </configuration>
     </plugin>
  </plugins>
</build>
```

Listing 1: Maven-Konfiguration mit parametrisierbarem Format des Reports

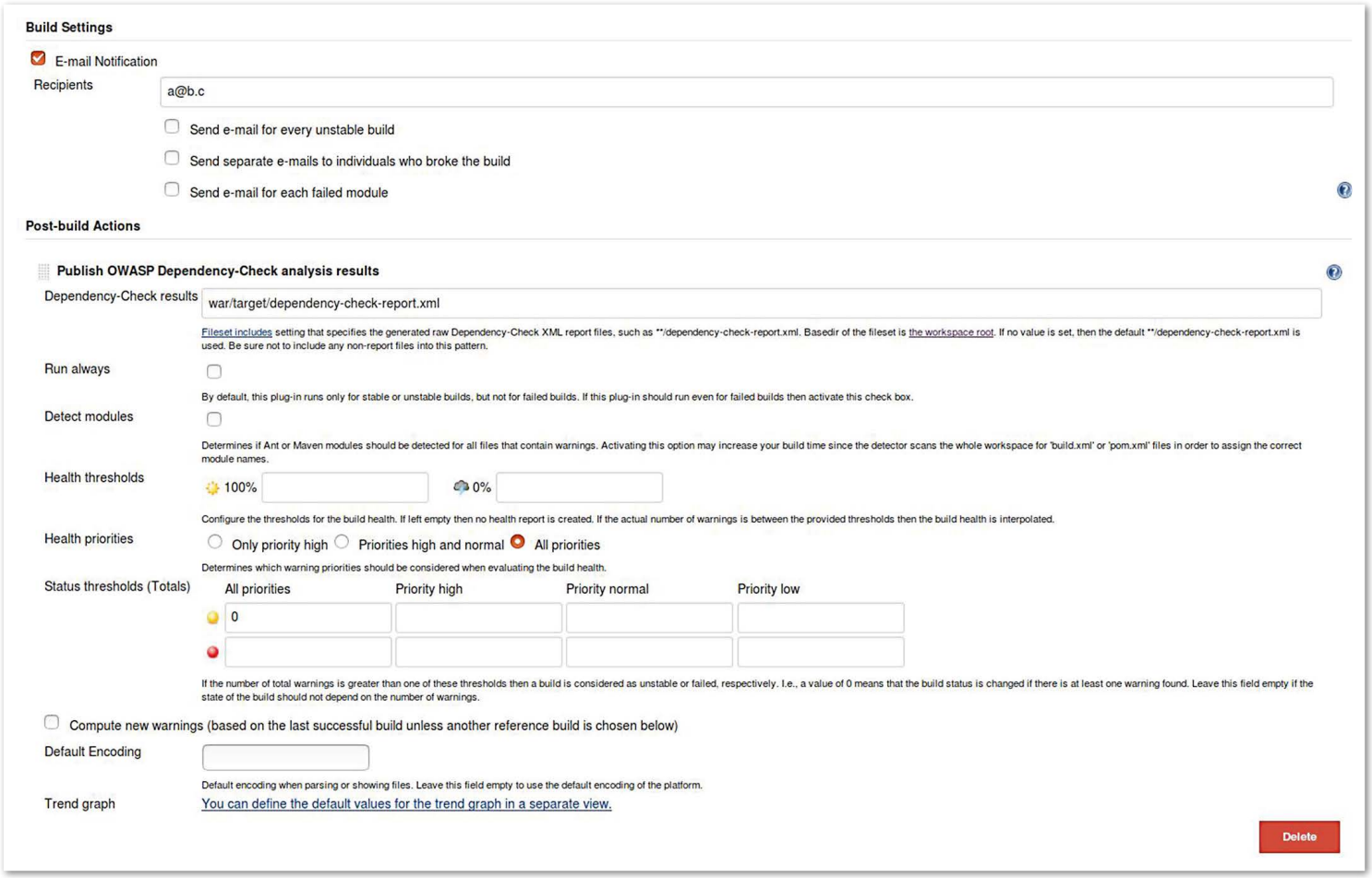

Abbildung 1: Das Ergebnis der Post-Build-Action

```
<configuration>
  <!-- Skip artifacts not bundled in distribution (provided scope) -->
  <skipProvidedScope>true</skipProvidedScope>
  <!-- Suppress false positives or dependencies that cannot be changed for specific reasons.-->
  <suppressionFile>suppress-cves.xml</suppressionFile>
</configuration>
```

Listing 2: Maven-Konfiguration mit suppressionFile und provided Scope

org.owasp:dependency-check-maven:check -Ddependency-check-format=XML". Dadurch wird eine Datei „target/dependency-check-report.xml" im Workspace generiert, die man von einer entsprechenden Post-Build-Action verarbeiten lassen kann. Abschließend muss in dieser noch konfiguriert werden, dass der Job fehlschlägt oder instabil wird, sobald eine Sicherheitslücke entdeckt wird *(siehe Abbildung 1)*. In Zusammenarbeit mit der von Jenkins angebotenen E-Mail-Benachrichtigung ist sichergestellt, dass man ohne weiteres Zutun per E-Mail über neu gefundene Sicherheitslücken informiert wird.

Die Praxis

Abschließend noch einige Details, die in der täglichen Arbeit mit dem Dependency-Check relevant sein können. Es empfiehlt sich, den Dependency-Check auf dem Maven-Modul durchzuführen, das das eigentlich veröffentlichte Artefakt (.jar, .war, .ear etc.) herstellt. Dadurch werden auch nur die wirklich relevanten Abhängigkeiten auf Sicherheitslücken durchsucht und nicht die Abhängigkeiten von Modulen, die beispielsweise nur für Tests im Einsatz sind. Abhängigkeiten im Maven-Scope-Test werden in der aktuellen Version des Maven-Plug-ins standardmäßig nicht durchsucht.

Anders sieht es mit Abhängigkeiten in den Scopes „provided" und „runtime" aus. Insbesondere bei „provided" können sie unerwünscht sein. Wenn man beispielsweise durch die Kundenumgebung an ein bestimmtes Servlet-API gebunden ist, ist man gegen die gegebenenfalls darin enthaltenen Sicherheitslücken machtlos. Insofern kann es durchaus sinnvoll sein, diesen Scope auszuschließen. Dies kann in der Konfiguration von Maven angepasst werden *(siehe Listing 2)*.

Alternativ lassen sich bestimmte CVEs auch für einzelne Abhängigkeiten unterdrücken. Diese sind in einer speziellen Datei spezifiziert. Der XML-Code für das Unterdrücken lässt sich direkt aus dem im ersten Schritt generierten HTML-Bericht kopieren. Dies kann insbesondere sinnvoll sein, wenn es sich um False Positives handelt, also um gefundene Sicherheitslücken, die auf die Abhängigkeit gar nicht zutreffen. Ein Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<suppressions xmlns="https://jeremylong.github.io/DependencyCheck/dependency-suppression.1.1.xsd">
  <suppress>
     <!-- This Jackson-related issue seems to be related to module jackson-dataformat-xml, which is not used here.
It is considered a false positive. See https://github.com/jeremylong/DependencyCheck/issues/517 In addition it was
fixed in version 1.7.4., see https://github.com/FasterXML/jackson-dataformat-xml/issues/199 -->
     <notes><![CDATA[file name: jackson-core-2.8.1.jar]]></notes>
     <sha1>fd13b1c033741d48291315c6370f7d475a42dccf</sha1>
     <cve>CVE-2016-3720</cve>
  </suppress>
  <suppress>
     <!-- Same as other Jackson-related issue -->
     <notes><![CDATA[file name: jackson-annotations-2.8.0.jar]]></notes>
     <sha1>45b426f7796b741035581a176744d91090e2e6fb</sha1>
     <cpe>cpe:/a:fasterxml:jackson:2.8.0</cpe>
  </suppress>
</suppressions>
```

Listing 3: Beispielhaftes suppressionFile

dafür ist der Generalverdacht (CVE-2016-3270, *siehe „https://github.com/jeremylong/DependencyCheck/issues/517“)* gegen alle Module des Jackson-Frameworks, obwohl nur in einem eine Sicherheitslücke auftritt. Es ist daher generell sinnvoll, jede gefundene Sicherheitslücke auf Korrektheit zu überprüfen. In jedem Fall sollte kommentiert werden, weshalb ein CVE unterdrückt wird, um zu späteren Zeitpunkten die Gründe für den Ausschluss nachvollziehen zu können. In den *Listings 2 und 3* wird diese Möglichkeit für den Ausschluss aufgezeigt.

Fazit

Der Artikel zeigt, wie man mit wenigen Schritten dauerhaft über bekannte Sicherheitslücken von Abhängigkeiten informiert bleibt. Dies sorgt nicht für absolute Sicherheit, entschärft aber mit geringem Aufwand einen der zehn gängigsten Makel. Insofern ist dies uneingeschränkt für alle Java-Projekte zu empfehlen.

Die hier beschriebene, auf Maven und Jenkins basierende Lösung stellt eine Möglichkeit für das Herausfinden von Sicherheitslücken dar. Aufgrund der vielen verfügbaren Implementierungen des OWASP-Dependency-Checks lassen sich vergleichbare Lösungen auch für viele andere Tools realisieren. Ein komplett lauffähiges Beispiel, das die oben beschriebenen Fälle sowie eine Abhängigkeit mit nicht ausgeschlossener Sicherheitslücke enthält, steht bei GitHub *(siehe „https://github.com/triologygmbh/dependency-check“)*.

Johannes Schnatterer
johannes.schnatterer@triology.de

Johannes Schnatterer ist Solution Architect bei der TRIOLOGY GmbH in Braunschweig. Technologisch ist er dort in den Bereichen „Java EE“ und „Web“ tätig und versucht, mit besonderem Fokus auf Qualität, Open-Source-Enthusiasmus, einem Hauch von Pedantismus und der Pfadfinderregel die IT-Welt jeden Tag ein bisschen besser zu machen.

Veranstaltungen der im iJUG organisierten Java User Groups auf Erfolgskurs

Mit einer Rekordbeteiligung von rund 1.700 Teilnehmern fand am 7. Juli 2016 zum 19. Mal das Java Forum in Stuttgart statt. Die Java User Group Stuttgart hatte 49 Vorträge in sieben parallelen Tracks organisiert. Zudem waren 34 Aussteller vor Ort, darunter auch der Interessenverbund der Java User Groups e.V. (iJUG). Abends gab es die Gelegenheit, sich bei verschiedenen BoF-Sessions (Birds of a Feather) mit Gleichgesinnten zu treffen, um über ein bestimmtes Thema zu diskutieren und sich auszutauschen.

Am 15. und 16. September 2016 waren in Berlin die Berlin Expert Days. Der Verein Berlin Expert Days e.V. wurde im Jahr 2010 mit dem Ziel gegründet, eine Plattform zum Informationsaustausch anzubieten. Als offen geführter Verein steht eine solide Basis bereit, um die Unabhängigkeit von Herstellern und Dienstleistern zu gewährleisten. Auf der diesjährigen Veranstaltung wurden mehr als 500 Teilnehmer auf den 44 Vorträgen begrüßt.

Das Java Forum Nord öffnete am 20. Oktober 2016 in Hannover seine Pforten. Die eintägige, nicht-kommerzielle Konferenz in Norddeutschland mit Themenschwerpunkt Java ist für Entwickler und Entscheider. Mit mehr als 25 Vorträgen in parallelen Tracks und einer Keynote wurde ein vielfältiges Programm zu einem unschlagbaren Preis geboten, der regionale Bezug bietet zudem interessante Networking-Möglichkeiten. Gestaltung und Organisation wurde von den lokalen Java User Groups (Bremen, Göttingen, Hamburg, Hannover, Kassel, Ostfalen) in Kooperation mit der Java User Group Deutschland e.V./Sun User Group Deutschland e.V. als offiziellen Veranstaltern durchgeführt. Es kamen 400 Besucher.

Auch das Datum für die JavaLand 2017 steht bereits fest. Sie findet vom 28. bis 30. März 2017 an gewohnter Stätte im Phantasialand Brühl statt.