Log4Shell – Schwelbrand in tausenden Servern

von Thomas Kestler, 13.12.2021, Update 16.12.2021

Einleitung

Die von vielen Java-Entwicklern genutzte Bibliothek log4j hat leider in Versionen <= 2.14.1 eine ZeroDay-Lücke (CVE-2021-44228[1]) mit deren Hilfe ein Angreifer eine externe Java-Klasse laden und ausführen kann. Viele Server und sogar große Internetplattformen waren betroffen. Es gibt zwar inzwischen Abhilfe (Upgrade der log4j Version[2], Konfigurations-änderung, Entfernen der kritische Klasse aus log4j.jar, Java-Update, etc), aber dieser Vorfall zeigt wieder einmal exemplarisch ein massives Problem in der heutigen SW-Entwicklung auf, welches ich im Anschluss detaillieren möchte.

1 https://nvd.nist.gov/vuln/detail/CVE-2021-44228

2 Inzwischen wurde die Lücke gesfixt, in Version 2.16 muss JNDI ausdrücklich aktiviert werden und man kann konfigurieren, welche Klassen von welchen Hosts per Lookup geladen werden dürfen.

JAR-Hölle

Java wurde zu einer der wichtigsten Programmiersprachen aus verschiedenen Gründen. Darunter:

- Einfache Sprache
- Typsicher, objektorientiert, mit Bounds Checks, etc.
- Plattformunabhängig
- Große Verbreitung, viele Frameworks und Bibliotheken

Gerade der letzte Punkt ist Fluch und Segen zugleich. Als Programmierer hat man eine enorme Auswahl an Bibliotheken, welche zunächst das (Programmierer-)Leben einfacher machen. Log4j wird nahezu in jedem Java-Projekt eingesetzt, es wird von vielen Frameworks als Abhängigkeit (Dependency) vorausgesetzt und die meisten Java-Applikationsserver nutzen es auch intern für das Logging. Aber noch viele weitere Bibliotheken werden heute sehr häufig eingesetzt, z. B. Apache Commons.

In der Folge werden bei der Auflösung der Abhängigkeiten, z. B. beim Einsatz von Maven, sehr viele Bibliotheken in das Projekt eingebunden und dann mit der Applikation ausgeliefert. Das Mengenverhältnis (in MB) von reinem Applikationscode zu Bibliothekscode dürfte in der Praxis in Richtung 1:10 liegen (wenn nicht noch höher).

Gerade bei Maven hat der Programmierer kaum mehr einen Überblick, welche Bibliotheken automatisch in das Projekt geladen werden. Eine Reglementierung im Sinne eines strengen Software Change and Configuration Management (SCCM) findet nur sehr selten statt. Aus diesem Grund traf diese ZeroDay-Lücke wohl auch so viele System (und das BSI hat hier Warnstufe rot vergeben).

Trügerisch ist auch die Annahme, das Bibliotheken wie log4j sicher seien, weil sie ja von so vielen Programmierern genutzt werden und als OpenSource jederzeit inspizierbar sind. Offensichtlich hat dies in diesem Fall nicht geholfen. Ob die ZeroDay-Lücke gezielt implantiert wurde oder nur aus Unachtsamkeit entstand ist egal, schließlich haben Hacker das Problem erkannt und ausgenutzt. Ob hier Geheimdienste am Werk waren, lässt sich trefflich spekulieren. Jedenfalls, wenn ich bei den Schlapphüten tätig wäre, würde ich zunächst nach Schwachstellen in häufig verwendeten OpenSource Bibliotheken suchen.

Ein Grund, warum die ZeroDay-Lücke ausgenutzt werden konnte ist, dass viele Server noch mit älteren Java8-Versionen laufen. Ab Update121 soll der Fehler behoben sein, die Frage ist, warum erst ab dann? Das Ausführen von nachträglich geladenen Klassen kann schon ein Sicherheitsproblem sein, oder? Selbst große Unternehmen kämpfen aus diversen Gründen mit einem Update von Java, u. a. wegen der geänderten Lizenzpolitik von Oracle[3].

Das grundlegende Problem ist jedoch, dass in den allermeisten Projekten sehr viele Bibliotheken unkritisch eingebunden werden.

3 Ab JDK17 hat Oracle kürzlich wieder kostenlose Nutzung zugesichert, wenn auch mit Einschränkungen. https://blogs.oracle.com/java/post/free-java-license

Herausforderung für SW-Architekten

Die aktive Verwaltung von Fremdbibliotheken (im Sinne eines SCCM) ist eine anspruchsvolle Aufgabe für SW-Architekten (genauer: Enterprise Architect, Application Architect).

Ein Enterprise Architect benötigt einen Katalog, welche Bibliotheken in welchen Versionen in den Applikationen genutzt werden. So kann er(sie/es) im Notfall schnell betroffene Applikationen identifizieren. Alternativ kann der EA auch von dieser Kleinarbeit entlastet werden und der IT-Betrieb übernimmt dies.

Ein Application Architect sollte als Gatekeeper fungieren und ständig im Blick behalten, welche Bibliotheken in welchen Versionen eingesetzt werden (können). OSGi hatte ein wirkungsvolles Modell zur Verwaltung von Modulen und zur Steuerung was wann eingebunden wird. Daher hatte ich daran auch Gefallen gefunden. Außer in RCP-Projekten kam es jedoch nur selten zum Einsatz. Mit Java 9 wurde Jigsaw etabliert. Nur muss man die Modularisierung dann auch konsequent umsetzen und nicht doch JARs über den Classpath laden.

Die konsequente Nutzung von Modularisierung durch Jigsaw (ab Java 9) ist empfehlenswert. Aber auch hier kann es zum dynamischen Nachladen von Klassen durch einfache Konfigurationsänderungen kommen.

Ein theoretischer Ansatz wäre ein Stripping aller Fremdbibliotheken mit Entfernen aller nicht benötigten Klassen. In der Praxis wäre dies aber mit enorm viel Arbeit verbunden, weil man den Vorgang ja mit jeder neuen Version für jede Bibliothek wiederholen müsste (außer man automatisiert dies mit Hilfe von Whitelists). Oftmals werden umfangreiche Bibliotheken eingebunden von denen dann tatsächlich nur wenige Klassen tatsächlich genutzt werden.

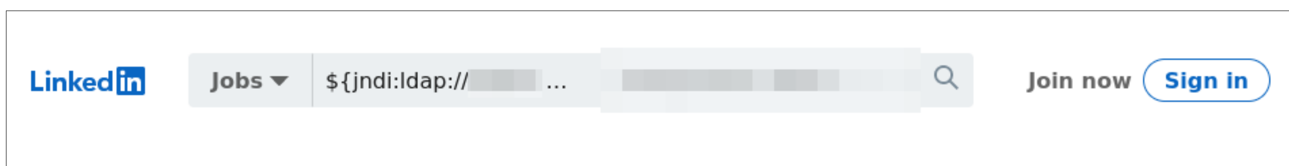
Zusammenfassung

Wichtig ist eine Schärfung des Problembewusstseins bei SW-Architekten und Programmierern (und auch im IT-Betrieb). Fremdbibliotheken sollten nicht kritiklos eingesetzt werden. OpenSource ist keine Garantie für Sicherheit[4].

Ach ja, und eine Überprüfung bzw. ein Quoting der Eingabeparameter vor Übergabe an das Logging hätte auf vielen betroffenen Servern und Plattformen vor Schaden bewahrt. Wenn es einfach reicht in ein Eingabefeld einen magischen String wie

```
${jndi:ldap://127.0.0.1:1389/...}
```

zu kopieren, dann macht man es Angreifern aber auch unnötig leicht.



Man muss auch überlegen, ob man solche Detailinformationen – falls überhaupt für das Debugging relevant – nicht besser den Kontext in die Message zusteuert. Die Idee, es über den Formatstring zu ermöglichen, hat eben genau diese Lücke gerissen[5].

4 Ungeachtet dessen, bleibe ich Fan von OpenSource
5 Wie oben erwähnt, ist die Lücke inzwischen geschlossen worden (aktuell V 2.16)