Weihnachten mit Log4j

Log4Shell / Log4J verursachte einen großen und lauten Aufschlag, strategisch klug in der Weihnachts- und Zwischenjahreszeit. Wir waren gut und erfolgreich geschützt, aber dennoch hat uns das Thema mehrfach und einige Tage beschäftigt:

* Sind unsere Infrastrukturen geschützt?

* Sind unsere Produkte betroffen?

* Wie können wir unseren Kunden und unserer Community helfen.

Wir haben darüber ein Tagebuch geführt, dass in der Nachbetrachtung uns, und wir vermuten auch vielen Anderen, nützliche Informationen und Erkenntnisse bietet. Vor allem ist dies interessant für Leute die sich dafür interessieren, wie mit und durch Open Source und ein paar gute Prozesse mehr Sicherheit geschaffen werden kann. Denn der nächste Vorfall kommt bestimmt.

Um was geht es bei Schwachstellen eigentlich?

Es gibt aktuell hundertausende bekannter Schwachstellen, die sich nach Schweregrad (CVE Score), Auffindbarkeit (Quality of Detection- QoD Score), Ausnutzbarkeit, Verbreitung betroffenener Produkte unterscheiden. Natürlich betrifft nicht jede Schwachstelle betrifft jedes System, und nicht jede Ausnutzung der Schwachstelle ist für die Betrieber der Systeme gleich relevant. So ist zum Beispiel für manche Dienste die Verfügbarkeit besonders kritisch, dann ist eine Schwachstelle die eine Denial of Service Angriff ermöglicht, aber keine Zugriff auf Daten des Services hat, trotzdem besonders relevant. Für andere wiederum sind Verfügbarkeitsausfälle besser tolerierbar, aber nicht der Zugriff auf Daten. Viele weitere Unterscheidungen sind vorstellbar, ein paar grundsätzlich gleichermaßen geführliche Auswirkungen gibt es jedoch, die für alle Dienste gelten, wie die Übernahme von administrativen Rechten, das Einschleusen von Ransomware und dergleichen. Ebenfalls als besonders gefährlich gelten natürlich Schwachstellen, die ermöglichen das sich Schadsoftware im System einnisted.

Bei dieser Betrachtung wird relativ schnell deutlich, das nicht die Schwachstelle die eigentlich Gefahr darstellt, sondern deren Ausnutzung für die eigentlichen Angriffe. Dabei spielt die Ausnutzbarkeit eine große Rolle, und auch diese wird natürlich stark davon beeinflusst, was ich präventiv tun kann, wie schnell ich auf Bedrohungen reagiere, und wie schnell ich im Zweifel in der Lage bin eine sichere Betriebsfähigkeit wieder herzustellen.

Wie entstehen Schwachstellen?

Schwachstellen entstehen durch

- Konfiguration,
- ungeplante Wechselwirkungen,
- Programmierfehler,
- fehlerhafte Prozesse.

Wenn Schwachstellen auftreten, entzünden sich zudem in der Regel weitere Diskussionen, die sich eher im Bereich von

- Programmiersprache / Framework
- Entwicklungsmodell (Proprietär oder Open Source)
- Technologie

bewegen. Wir wollen uns hier weniger mit der Diskussion der Ursachen auseinandersetzen, sondern eher mit den Auswirkungen, und wie wir diesen begegnen können. Dazu gehört auch ob und inwieweit wir selbst Schwachstellen verhindern können – oder inwieweit wir schlimmstenfalls selbst zu Schwachstellen beitragen.

Besonders dort, wo sehr verbreitete Produkte als Standard gelten, gibt es die implizite Erwartung, das diese „sicher genug“ seien – denn schließlich werden sie von so vielen Menschen genutzt. Diese Erwartung ist nicht gerechtfertigt, im Gegenteil. Denn besonders beliebte Produkte sind natürlich auch besonders beliebte Ziele für Angriffe. Und sehr häufig kennen wir nur die uns nutzbringende Funktionen von Software, und wissen wenig darüber, wie sie entsteht.

Ein weiterer, impliziter Zusammenhang zwischen Beliebtheit und Angreifbarkeit ist: Produkte sind dann besonders beliebt, wenn sie besonders komfortabel sind. Komfort und Sicherheit müssen jedoch gut abgewogen werden. Nicht jedes Schwachstelle ermöglicht Komfort, aber ein zuviel an Komfort bedeutet auch ein größere Angreifbarkeit. Einfache Beispiele sind bequeme Passwörter, Verzicht auf Passwortmanager, Verzicht auf 2 Faktor Authentifizierung. Ein komplexeres Beispiele hierfür ist ein hoher Integrationsgrad mit weiteren Systemen, hier liegen unter anderem die Auswirkungen des Log4J Problems begründet.

Was ist besonders an Log4j

Log4J ist ein sehr einfaches, komfortables und nützliches Werkzeug in der Softwareentwicklung. Es gilt in der immer noch sehr verbreiteten Softwareentwicklung mit der Programmiersprache Java als Standard, um Zustände und Aktivitäten bei der Nutzung

einer Software zu protokollieren. Dies wird als „Logging“ bezeichnet, daher der Name Log4J. Eine Logging Funktion wird in praktisch jeder Software benötigt. Sie wird zudem nicht nur zur Entwicklungszeit genutzt, sondern auch in die Nutzungskontexte getragen, somit massiv verbreitet. Der Einsatzbereich kann sehr divers sein: Anwendungsserver in Unternehmen oder im Internet, Desktopsysteme aber auch Internetrouter, Firewallgeräte und viele mehr können betroffen sein.

Die Anwendung von Log4J ist sehr einfach und ein Bewußtsein für die möglichen Risiken war bisher nicht sehr ausgeprägt, sicher auch häufig nach dem Motto: „Was soll an Logging schon gefährlich sein?“. Zudem hat die Person, die Log4J in der Programmierung einsetzt, auch kein klare Bild davon, wie sich die Komponente in Einsatz wirklich verhält, und welche Konfigurationsparameter darauf Einfluss haben.

Log4J gibt schon etliche Jahre, ohne das größere Probleme bekannt wurden, so dass der Einsatz auch nicht mehr bewertet oder im konkreten Anwendungszenario hinterfragt wurde.

Schwachstellen können sehr unterschiedlich sein, was die Einfachheit einer Ausnutzung betrifft. Als die entscheidende Schwachstelle in Log4j gefunden wurde, zeigte sich unmittelbar, dass diese sehr leicht auszunutzen war, bei möglicherweise dramatischen Folgen.

Diese Rahmenbedingungen führen zusammen zu

- Einer extrem weiten Verbreitung in verschiedenste Anwendungsysteme.
- Einem hohen möglichen Risiko beim Einsatz der Komponente.
- Wenig Aktivitäten und Bewußtsein für risikobegrenzende Maßnahmen.
- Einfache Ausnutzung der Schwachstelle.
- Große mögliche Schäden.

Dieses Zusammenwirken machte den Log4j Vorfall zu etwas Besonderem. Wie wir damit umgehen, damit wollen wir uns in diesem Vortrag auseinandersetzen.