Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Log4Shell Story - Wie konnte es zur "Log4Shell"- Sicherheitslücke kommen?

Lasse Fischer
Matrikelnummer: 4775329
lasse.fischer@fu-berlin.de

Betreuer/in: Prof. Dr. Lutz Prechelt
Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachter/in: Prof. Dr. Jörn Eichler

Berlin, 21. Februar 2023

Zusammenfassung

Hintergrund: Ende 2021 wurden die Sicherheitslücken CVE-2021-44228, CVE-2021-45105 und CVE-2021-45046 mit dem Spitznamen "Log4Shell" veröffentlicht. Diese konnten es Angreifern ermöglichen, beliebigen Code in Systemen auszuführen, die die beliebte Java-Bibliothek "Log4j 2" von Apache Software verwendeten.

Ziele: Ziel dieser Arbeit ist es, bestmöglich zu verstehen und glaubhaft zu machen, wie es zu dieser Fehlleistung kommen konnte.

Ergebnisse: Die "Log4Shell"-Sicherheitslücke ist ein Zusammenspiel aus mehreren korrekt implementierten und getesteten Features, die nicht zusammen in der Softwarearchitektur hätten existieren dürfen. Dass dieser Missstand entstanden ist und so lange unentdeckt blieb liegt an der enormen Ausdehnung des Projektumfanges und fehlender bzw. unvollständiger Dokumentation und daraus resultierendem Verlust des Überblicks der Projektleitung.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

21. Februar 2023

Lasse Fischer

Inhaltsverzeichnis

1 Einführung

1.1 Hintergrund

Einer der vielen Vorteile von Open Source Software [1] ist, dass ihr Code häufiger und von mehr Menschen überprüft wird. Doch haben uns die letzten Jahre gezeigt, dass auch in Open Source Software über lange Zeit Fehler unentdeckt bleiben können. Sowohl der 'Heartbleed' Bug in OpenSSL und 'ShellShock' in der Unix-Shell Bash aus dem Jahre 2014, als auch 'Drown' in OpenSSL aus dem Jahre 2016 waren schwerwiegende Fehler in Open Source Projekten. Man sollte sich demnach nicht vollständig auf die Sicherheit von Open Source Softwarebestandteilen verlassen und genau abwägen, wie sehr man welchen Systemkomponenten vertraut.

Ein weiteres Beispiel ist die Sicherheitslücke, die unter dem Namen 'Log4Shell' bekannt geworden ist. Veröffentlicht wurde diese am 10. Dezember 2021 auf der Website cve.org mit der Bezeichnung CVE-2021-44228 [1]. Sie erhielt den höchstmöglichen Vulnerability Score von 10.0 im Common Vulnerability Scoring System von NIST, des amerikanischen National Institute of Standards and Technology und ist hiermit wahrscheinlich eine der schwerwiegendsten Software-Sicherheitslücken überhaupt. Sie ermöglichte es Angreifern über speziell generierte Anfragen unter Nutzung des Java Naming and Directory Interfaces (JNDI) beliebigen Code auf dem angegriffenen System zu laden und auszuführen. JNDI wird gewöhnlich dafür verwendet, Datenbanken anzubinden oder verteilte Objekte in einem Netzwerk zu referenzieren und über remote method invocation (RMI) anzusprechen.

```
logging.info("Important Log - ${jndi:ldap://127.0.0.1/pfad/zu/
    boesartiger/java/klasse}")
```

Beispiel für eine Lognachricht, die über JNDI eine Java-Klasse von einem Server lädt.

Eingeführt wurde Log4Shell im Jahre 2013 durch Hinzufügen der Funktionalität von JNDI-Lookups im Open Source Projekt Log4j der Apache Software Foundation[2]. Dies ermöglichte es mir, tiefe Einblicke in die Entwicklungsgeschichte des Programms zu werfen, da z.B. Mailing Listen [6][5], JIRA Issue Tracking, sowie das GitHub Projekt öffentlich zugänglich sind. Es handelt sich bei Log4j um ein Logging Framework für Java, daher auch der Name Log(ging) for(4) j(ava).

In der modernen Softwareentwicklung wird Logging so gut wie überall eingesetzt, um Informationen über den Ablauf von Programmen, Anwendungen oder Systemen zu sammeln und aufzuzeichnen. Typischerweise protokolliert man Ereignisse wie Fehler, Warnungen oder andere relevante Informationen, um diese später zu analysieren oder zur Diagnose von Problemen zu nutzen. Dazu werden an entsprechenden Stellen im Programmcode Log-Nachrichten in eine dafür vorgesehene Datei geschrieben. Beispielsweise könnte bei einer Systemanmeldung der Benutzername oder die genutzte IP-Adresse in einer Logdatei mitgeschrieben werden. Log4j ist eines der am weitesten verbreiteten Logging-Frameworks für Java und wird oft entweder direkt

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228
[2] https://logging.apache.org/log4j/2.x/index.html

oder indirekt von anderen Java-Bibliotheken oder -Frameworks genutzt. Dadurch ist die Anzahl der potenziell betroffenen Systeme extrem hoch, vor allem, da schon eine einzige Schnittstelle nach außen reicht, um für den Exploit anfällig werden zu können.

Ziel dieser Arbeit ist es, bestmöglich zu verstehen und kenntlich zu machen, wie diese Sicherheitslücke enstanden ist. Dazu habe ich die vielen öffentlich zugänglichen Informationsquellen des Projektes untersucht.

1.2 Was macht Log4Shell so besonders?

Log4Shell unterscheidet sich von den anderen großen Sicherheitslücken der letzten Jahre vor allem dadurch, dass es sich hierbei nicht um einen Programmierfehler bzw. Bug handelt, sondern um ein Problem in der Softwarearchitektur des Projekts selbst. Dies ist besonders signifikant, da man Programmierfehler durch spezielle Prozesse, wie z.B. Code Reviews oder automatische Tests finden und dementsprechend beheben kann. Problematischer wird dies bei Architekturentscheidungen, da das Programm ja wie spezifiziert funktioniert.

1.3 Log4j Grundkomponenten und ihre Funktion

Um verstehen zu können, wie der Log4Shell-Exploit funktioniert, muss man zunächst die Grundbausteine von Log4j verstehen. Diese werde ich hier zunächst grob erklären und dann später an der jeweiligen Stelle genauer erläutern. Für den Logging-Prozess sind die folgenden Hauptkomponenten zuständig:

1.3.1 LogManager

Die Aufgabe des LogManagers ist es, die von der API angefragten Logger bereitzustellen. Dazu sucht er den dazugehörigen LoggerContext, welcher ihm den gefragten Logger übergibt. Sollte der gefragte Logger erstellt werden müssen, wird er dem zuständigen LoggerContext zugewiesen. Dies ist entweder der LoggerContext mit selbem Namen wie der Logger, der LoggerContext mit demselben parent package oder der root LoggerConfig.

1.3.2 LoggerContext

Der LoggerContext schreibt das Verhalten eines Loggers vor und stellt somit den Ankerpunkt des Loggingvorganges dar. Jeder LoggerContext enthält die Liste aller konfigurierten Logger sowie eine Referenz auf die zugehörige Configuration. Zu jedem LoggerContext gehört eine aktive Configuration. Es ist jedoch möglich mehrere unterschiedliche LoggerContexte in einem System zu haben, zum Beispiel in Systemen mit mehreren Anwendungen könnte jede Anwendung einen eigenen LoggerContext haben.

1.3.3 Configuration

Die Konfiguration des LoggingContexts enthält eine Liste mit allen konfigurierten Loggern, Filtern und Appendern des jeweiligen LoggerContexts, sowie den Verweis

auf die StrSubstitutor Klasse. Diese Konfiguration kann auf unterschiedlichen Wegen erzeugt werden, beispielsweise durch eine .XML, .YAML oder .JSON Datei, programmatisch durch kreieren einer ConfigurationFactory und Implementierung einer Configuration, durch API Aufrufe, um die Standardkonfiguration zu erweitern, oder durch Methodenaufruf in der internen Logger Klasse.

1.3.4 Logger

Logger werden vom LogManager erstellt und ihr Verhalten von ihrem LoggerContext vorgegeben. Ein Logger hat einen Namen sowie ein Level, über das entschieden wird, welche Arten an Events geloggt werden. Die Log Level in Log4j sind TRACE, DEBUG, INFO, WARN, ERROR, ALL und OFF, wobei als Standard ERROR eingestellt ist. Die Log-Level sind hierarchisiert, wobei die Lognachrichten immer alle Daten der niedriger eingestuften Log-Level beinhalten.

1.3.5 Filter

Filter werden benutzt, um zu regulieren, welche LogEvents tatsächlich in die Logdateien geschrieben werden. Hierfür existieren drei unterschiedliche Status: accept, deny und neutral. Bei accept werden bei diesem Event keine weiteren Filter angewendet und das Event wird direkt an die Appender weitergeleitet. Bei deny wird das LogEvent ignoriert und nicht weitergegeben. Bei neutral wird das Event an den nächsten Filter weitergegeben. Sollten keine weiteren Filter existieren, wird das Event verarbeitet.

1.3.6 Appender

Der Appender in Log4j bezeichnet den Ort, an den eine LogMessage geschrieben wird. Dies kann z.B. die Konsole, eine Datei, ein Remote Server oder eine Datenbank API sein. Eine spezifische LoggerConfig kann hierbei pro Logger mehrere Appender haben, jeder einzelne loggingRequest wird dann an alle Appender eines Loggers weitergegeben.

1.3.7 Layout

Hier kann ein bestimmtes Layout für eine LogMessage angegeben werden. Das Layout wird einem Appender zugewiesen, der dieses dann für seine Nachrichten anwendet.

Das Standardpattern von Log4j ist das TTCCLayout: time, thread, category und context information.

```
%r [%t] %-5p %c %x - %m%n
```

Hierbei steht %r für die Zeit seit Programmstart in Millisekunden, %t der Thread, %p die Eventpriorität, %c für die Eventkategorie, %x die verschachtelten Kontextinformationen des aufrufenden Threads und %m für anwendungsspezifische Nachrichten die zu diesem Event gehören. %n sorgt für einen Zeilenumbruch am Ende jeder Lognachricht.

Der Log4Shell-Exploit benutzt eben diese anwendungsspezifischen Nachrichten um mit Hilfe von JNDI-Lookups Remote Code Execution (RCE) zu ermöglichen.

Das Pattern

```
pattern "%r [%t] %-5p %c - %m%n"
```

könnte z.B. folgende Lognachricht erzeugen.

```
176 [main] INFO  org.foo.Bar - Located nearest open bar.
```

1.3.8 StrSubstitutor/StrLookup

Der StrSubstitutor wurde ursprünglich aus Apache Commons Lang, einer Java-Bibliothek, die viele nützliche Funktionen bereitstellt, übernommen und für das Log4j Projekt so angepasst, dass LogEvents evaluiert werden können. Er erlaubt es Variablen aus unterschiedlichen Quellen entweder bei Initialisierung der Configuration oder beim Loggen einer Nachricht, zu evaluieren. Diese könnten z. B. aus den System Properties, der Configuration, der ThreadContextMap oder der Structured Data aus dem LogEvent stammen.

1.3.9 LogEvent

Ein LogEvent enthält alle notwendigen Kontextinformationen für eine Lognachricht. Dazu gehört sowohl die Message, der ThreadContext, als auch das LogLevel der Nachricht.

1.4 Wie funktioniert der Log4Shell Exploit?

Ein Angreifer kann den Exploit dazu nutzen, um beliebigen Code von einem Webserver auf einem Server auszuführen, der Benutzerdaten des Angreifers loggt. Dies funktioniert, weil Lookups in der Lognachricht ausgewertet werden und einer der möglichen Quellen bei einem JNDI-Lookup ein Remote Server ist, dessen URL im Lookup-Pattern angegeben wird, also eben in der Benutzereingabe. Die Java- und Log4j-Features, die diesen Exploit ermöglichen oder seine Gefährlichkeit erhöhen, sollen hier näher erläutert werden.

1.4.1 Variablensubstitution

Zur sprachlichen Einfachheit wird in dieser Beschreibung die Bezeichnung “Log4j” verwendet, wenn eigentlich nur “angreifbare Versionen von Log4j 2” gemeint sind.

Findet sich in einer Log4j-Message das Pattern `${...}` so wird es nicht direkt ins entsprechende Ziel geschrieben sondern weiter ausgewertet. Diese Syntax findet sich auch in anderen Systemen wie Mavens *pom.xml* oder modernem *JavaScript*. Um Variablen aus unterschiedlichen Quellen zu laden, wird in den Klammern eine von mehreren Quellen und der Name der Resource mit einem Doppelpunkt getrennt eingegeben. So kann der Wert der Umgebungsvariable foo geloggt werden, indem ${env:foo}

im Logstring enthalten ist. Liegt foo im ThreadContext, greift man mit `${ctx:foo}` darauf zu. Mit `${date:<datumsformatstring>}` kann der aktuelle Zeitpunkt geloggt werden.

Diese Lookups werden erst ausgeführt, nachdem die LogMessage erzeugt wurde, die Ausführung ist sogar rekursiv. Es kann also zum Zeitpunkt der Substitution nicht mehr unterschieden werden, woher das `${...}`-Pattern stammt. Eine beispielhafte, unvollständige Aufzählung von möglichen Quellen des `${...}`-Patterns sei hier gegeben geordnet von nützlichen Anwendungsfällen zu Verhalten was niemals erwünscht ist und zum Exploit führt:

1. Das Layout-Pattern der Configuration gibt, wie in 1.3.7 erklärt ein Format für LogMessages vor. Diese kann zur Variablensubstitution das `${...}`-Pattern enthalten
2. Der Formatstring (bei einem Aufruf der Form `log("hello %s!", worldobj)` nennt man den String `"hello %s!"` den 'Formatstring') enthält eine Variablensubstitution.
3. Argumente des Loggingaufrufes werden in eine String-Repräsentation umgewandelt und dann in den Formatstring des Loggingaufrufes eingefügt, der dann die LogMessage bildet.
4. Aus mehreren Argumenten des Logaufrufes: Da die Lognachricht aus dem Formatstring und allen Argumenten gebildet wird, kann das `${...}`-Pattern auch aus mehr als einem der geloggten Objekte gebildet werden.

Im ersten Fall können wir davon ausgehen, dass die Konfigurationsdateien nur für die Entwickler zugängig sind und dort keine Benutzereingaben vorkommen. Ob beim zweiten Punkt Benutzerdaten enthalten sind, hängt vom jeweiligen Aufruf des Log Calls ab. Bei sauberem Code sollte dies allerdings nicht vorkommen. Bei vielen Funktionen (in vielen Kontexten und Sprachen) ist der Formatstring das erste Argument des Interface und danach folgt eine variable Anzahl an Argumenten beliebigen Typs. Es gilt als gute Praxis (notwendig für Sicherheit, aber auch hilfreich für allgemeines defensives Programmieren), den Formatstring als Konstante festzusetzen. Dies ermöglicht es intern, falsche Verwendung der Funktion festzustellen und Bereinigungen auf den Benutzereingaben auszuführen oder bedenkliche Operationen nur mit dem Format-Teil durchzuführen. Dadurch können Szenarien wie zum Beispiel printf-Formatstring-Attacken oder SQL-Injections vermieden werden. Da in Log4j jedoch die Substitutionen zuletzt ausgeführt werden, gibt es hier keinen Sicherheitsunterschied zwischen den Aufrufen `log.info("object:", obj)` und `log.info("object:"+obj)`.

Der letzte Punkt macht deutlich, dass eine ausreichende Bereinigung der Benutzereingabe auch nicht erfolgen kann. So sind zum Beispiel für den Aufruf `log.info("objects {}{}",obj1, obj2)` Werte für die Objekte denkbar, sodass die fertige Nachricht das `${...}`-Pattern enthält, die String-Repräsentation der einzelnen Objekte aber nicht.

Ein Angreifer löst also das Substitutionsfeature von Log4j mit beliebigem Inhalt auf dem Server aus, indem er in seine Eingabe Daten in Form von `${...}` einsetzt, die der Server von ihm annimmt und loggt.

Wie dies zu arbitrary code execution führt soll in den nächsten Kapiteln geschildert werden.

1.5 JNDI

Das Java Naming and Directory Interface (JNDI) ist eine Java-API, die es Anwendungen ermöglicht, auf Ressourcen über ihren Namen zuzugreifen. JNDI ermöglicht es, Objekte in einen hierarchischen name space zu registrieren und von dort wieder abzurufen. Vor allem wird dies verwendet, um Ressourcen wie Datenbanken, Verzeichnisse, Email-Dienste oder andere Dienste, die in einer verteilten Umgebung verfügbar sind, zu lokalisieren und darauf zuzugreifen.

Im Falle von Log4Shell kann ein Angreifer eine speziell präparierte Lognachricht mit einem bösartigen JNDI-Namen an die Log4j-Bibliothek senden. Dabei verwendet er die Funktion von JNDI, die es erlaubt durch verschiedene Protokolle Informationen abzurufen. Eines dieser Protokolle ist das Lightweight Directory Access Protocol (LDAP). Der Angreifer kann in der JNDI-Anfrage auf einen vom Angreifer kontrollierten LDAP-Server verweisen, worauf das Zielsystem dann zugreift. Von diesem LDAP-Server kann dann beliebiger Schadcode ausgeführt werden, da die Lookup-Funktion eine Klasse auf dem Server als externe Ressource sieht und diese ausführt. Dadurch kann der Angreifer das Zielsystem übernehmen, was Log4Shell zu einer der schwerwiegendsten Sicherheitslücken der letzten Zeit macht.

2 Projektkultur

Zu Beginn meiner Arbeit stand die Hypothese, dass die Projektkultur von derartiger Natur war, dass Sicherheitsbedenken überhaupt nicht in Erwägung gezogen werden, woraufhin sich der Exploit sozusagen in das Projekt eingeschlichen hat, ohne dass an irgendeiner Stelle jemand überlegt hat, ob und falls ja, welche Auswirkungen dies für die Sicherheit des Projektes haben könnte.

Um die Projektkultur zu verstehen, habe ich zunächst damit begonnen, mir die Entwickler-Mailingliste der Projekte anzusehen. Hier ist es wichtig zu wissen, dass diese Liste sich auf zwei separate Listen aufteilt, einmal von 2001-2017 und einmal von 2017 fortlaufend bis heute. Die Entwicklung von Log4j 2 begann jedoch erst im Jahre 2012, was zugleich den letzen Release von Log4j 1 sah, welches im Jahre 2015 als “end-of-life” [3] deklariert wurde, also keine weiteren Releases, Sicherheitsupdates oder Support erhielt. Dies bedeutet, dass, für einen Zeitraum von etwa 3 Jahren, die Mailing-Liste für beide Log4j Projekte genutzt wurde.

In diesem Abschnitt werde ich die Unterschiede, die ich zwischen den beiden Projekten gefunden habe aufzeigen und versuchen zu erklären, welche Auswirkungen diese auf die Entwicklung gehabt haben könnten.

[3] https://news.apache.org/foundation/entry/apache_logging_services_project_announces

2.1 Geschichte Log4j 1

Die Geschichte von Log4j 1 begann im Jahre 1996 im Rahmen des EU-Projekts Semper. Entwickelt wurde es im IBM-Forschungslabor in Zürich und wurde später der Apache Software Foundation übergeben, dessen Teil sie bis heute ist. Einer der drei Hauptentwickler führte das Projekt unter dem Schirm von Apache bis 2006 fort, wonach er seine eigenen Projekte abseits von Apache startete.

Die Hauptgründe für seinen Weggang aus dem Log4j Projekt nannte er in einem Interview im Dezember 2021, kurz nach Bekanntwerden der Log4Shell Sicherheitslücke.

> "[...] der Nachteil für Mitwirkende ist, dass man ständig lobbyieren muss. Es ist im Grunde ein politischer Prozess, du musst Leute für deine Änderungen gewinnen. Es gibt langwierige Auseinandersetzungen per Mail. Das ist ziemlich mühsam. [4]"

Dort beschreibt er weiterhin, wie er in einem seiner weiteren Projekte auf den jetzigen quasi Projektleiter[4], Mitglied des PMC und einer der Hauptautoren von Log4j 2 traf.

> "[...] Ich arbeitete unabhängig von Apache an der Log4j-Idee, dafür dachte ich mir neue Namen aus. SLF4j und Logback. Vor allem Logback wurde schnell sehr beliebt. Daran arbeitete ich eine Zeitlang auch mit [Name des "Projektleiters"[5]], der später Log4j 2 entwickelt hat.
>
> Bei mir und ihm gerieten zwei Weltanschauungen aneinander, er hat eine liberalere Einstellung zum Einbauen neuer Funktionen, ich bin ziemlich konservativ. 95 Prozent der Änderungsvorschläge lehne ich ab. Das ist für andere frustrierend. Deshalb machte [Name des "Projektleiters"] dann auch sein eigenes Ding und veröffentlichte um 2012 Log4j 2 auf Apache[4] [...] ."

Dies zeigt besonders, dass bei der Entwicklung von Log4j 1 deutlich mehr darauf geachtet wurde, die Primärfunktion der Logging-Bibliothek zu verbessern und nicht zusätzlich zu versuchen, auch andere Aufgaben zu übernehmen - ein Trend der sich in Log4j 2 definitiv so nicht wiedererkennen lässt.

Wie im Interview bereits erwähnt, war der Ton im Projekt teilweise sehr rau, z.B. hier in der Mailing Liste, eine Diskussion darüber ob Log4j dem JSR47 Standard folgen sollte oder nicht.

> "We are a meritocracy where only your contribution counts. You stop contributing you loose influence. This is as open as gets, anywhere, any time [...]
>
> ps: Whining is not considered as contribution. [6]"

[4] Es gibt de facto keinen richtigen Projektleiter in diesem Projekt, jedoch übernimmt diese Person einen großen Teil der administrativen Tätigkeiten im Projekt und war der Hauptentwickler zu Projektbeginn.

[5] Ich habe die Namen der Personen hier absichtlich weggelassen, bei Interesse stehen sie allerdings in der angegebenen Quelle.

[6] https://lists.apache.org/thread/4cwylb9d9zfd92j9gs1mfl5oqvho8m5t

2.2 Kultur in Log4j 2

Der Nachfolger von Log4j 1 ging im Jahr 2012 in den ersten Alpha Release, doch bis zum kritischen Release 2.0beta-9 verging über ein gesamtes Jahr in dem das Projekt wuchs.

2.2.1 Sicherheit in Log4j 2

Meine Bedenken, dass Sicherheitsbelange im Projekt nicht in Erwägung gezogen werden, konnte ich nicht bestätigen.

Ich konnte feststellen, dass die Apache Foundation ein spezielles Sicherheitsteam besitzt, dessen Aufgabe darin besteht, Apache Projekte bei der Behandlung von Sicherheitsproblemen und dem Umgang von Sicherheitslücken zu unterstützen.

> "The Apache Security Team provides help and advice to Apache projects on security issues and coordinates the handling of security vulnerabilities.[7]"

Des Weiteren gibt es auch im Log4j-Projekt eine eigene 'Security' Seite [8], die alle bekannten Sicherheitslücken von Log4j enthält. Zusätzlich enthält sie weiteren Informationen über die Sicherheitslücke, wer diese gemeldet hat, welche Versionen betroffen sind, sowie Mitigationsstrategien zur Schwachstelle.

So schreibt hier etwa ein Mitglied des PMC nach Bekanntwerden einer anderen Sicherheitslücke aus dem Jahre 2017 folgendes:

> "Given the inherent security problems with Java object serialization (highlighted by CVE-2017-5645), I do suggest that we deprecate SerializedLayout and remove it as default for SocketAppender, and all other appenders which currently have it as default. (We can still keep SerializedLayout, with a warning about security issues in documentation, but users will have to enable it explicitly.) [...]
>
> [...] I know this will break some existing configurations, but given the security problems, I think that is a price we have to pay in this case." [9]

Dies sehe ich als weiteres Indiz dafür, dass die Mitglieder des Projektes Sicherheitsbedenken durchaus wahrnehmen, auch wenn dies nicht häufig spezifisch diskutiert wird.

Weitere Beispiele für Sicherheitsbewusstsein des Entwicklerteams finden sich im Jira Issue Tracker des Projektes.

> " Added a Pattern encoding format limited to just CRLF for use cases where you do not want full HTML or JSON encoding, but do want to protected[sic] against CR and/or LF injection attacks in logs. "

Woraufhin ein weiterer Mitwirkender im Projekt antwortet:

[7] https://www.apache.org/security/
[8] https://logging.apache.org/log4j/2.x/security.html
[9] https://lists.apache.org/thread/3qnhzpxl9wc86b4snclcs6fj9m60g08k

> " However, I would say that to be really safe against injection attacks, I would recommend using JsonLayout or GelfLayout. [10]"

Ein Hauptgrund für die Abwesenheit von Sicherheitsbedenken in der öffentlichen Entwickler-Mailingliste ist jedoch sehr wahrscheinlich die private Mailingliste des PMC, auf die ich leider keinen Zugriff hatte. Diese dient laut der Log4j 2 Webseite "for discussion of issues that are inappropriate for public discussion, such as legal, personal, or security issues prior to a published fix. Subscription to the list is only open (actually: mandatory) to Apache Logging's Project Management Committee [11]". Ich denke, dass sich viele der Mitwirkenden im Projekt Gedanken über sicherheitskritische Belange vor und während ihrer Arbeit am Projekt machen und insofern ein gesundes Verständnis für Codesicherheit besitzen, was solche Diskussionen weiterhin minimieren könnte. Wenn allerdings Sicherheitsthemen auftreten, wird darüber auch diskutiert, wie z.B in dieser Diskussion zur Verschlüsselung von Passwörtern.

> **PMC Member A:**
>
> Having a plain text password in a config file seems like asking for trouble. Maybe we should at least support obfuscation, or access through a -D parameter, env var?
>
> **Project Lead:**
>
> I believe there is a semi-standard way to encrypt the password. I know we encrypt our database passwords for JBoss. Maven also supports encryption of the credentials stored in settings-security.xml.
>
> **PMC Member B:**
>
> First, any environmental variable can be used here, right? Log4j replaces things like ${env.whatever}, etc. Or do I understand that incorrectly?
>
> Second, there are three connection providers for the <Jdbc> appender:
>
> <DriverManager> (URL, username, password)
>
> <DataSource> (JNDI)
>
> <ConnectionFactory> (class name + static method that returns a DataSource or Connection)
>
> There are unit tests (actually more like integration tests, but that's the case all over Log4j2 appender tests) against <Jdbc> configurations with <DriverManager> and <ConnectionFactory>. I can create additional unit tests on pass 2 against <DataSource>.
>
> Since users can use variable substitution, I see no reason to natively support password obfuscation. [12]

[10] Beide Zitate stammen von https://issues.apache.org/jira/browse/LOG4J2-1203
[11] https://logging.apache.org/log4j/2.x/guidelines.html
[12] https://issues.apache.org/jira/browse/LOG4J2-229

2.2.2 Performance

Verbesserte Performanz gegenüber ihrem Vorgänger sowie einigen konkurrierenden Lösungen ist eines der Hauptmerkmale, welche die Projektseite für Log4j 2[13] angibt. Sie werben mit "18 times higher throughput and orders of magnitude lower latency than Log4j 1.x and Logback" und teilen ihre Benchmark Performance.

Dies schien mir als gutes Indiz, dass Performance eine sehr wichtige Projekteigenschaft ist. Daher habe ich mir die Diskussionen zu neuen Features im Projekt angesehen und konnte weitere Diskussionen zum Thema Performance finden, was meine Theorie weiter bestärkte.

So zum Beispiel hier in der Diskussion zur Einführung von asynchronen Loggern:

> **PMC Member A:**
>
> [...]
>
> To pass on data to the thread that does the actual logging, I use a similar approach as LogEventProxy, with one difference that Disruptor preallocates the event objects, to avoid creating a new event instance for every call to Logger.log() on the publisher (application) side. (The Log4jLogEvent instance is created on the event handling thread on the consumer side.)
>
> I do pass on the context map using (ThreadContext.isEmpty() ? null : ThreadContext.getImmutableContext()) and may pass on the stack too with (ThreadContext.getDepth() == 0 ? null : ThreadContext.cloneStack()) because these are fairly cheap. However, not calculating Log4jLogEvent.location is a big performance win.
>
> This is still a work in progress, but preliminary tests show it is possible to get to 450 nanos per call to Logger.log. I may be able to get it down to 150 nanos by always passing a null context map and stack and using a custom clock implementation instead of System.currentTimeMillis().[14]

Aus den oben gezeigten Beispielen schließe ich, dass Performance eine sehr wichtige Rolle spielt. Ich habe zusätzlich Beispiele gefunden, in denen Sicherheitsfragen über die Performance des Programms gestellt wurden. All dies sind weitere Indizien, die meine ursprüngliche Theorie widerlegen, dass Sicherheit im Projekt nicht bedacht wird.

Auch der generelle Ton scheint in Log4j 2 kollegialer und problembezogener zu sein als im Vorgängerprojekt. Eine genauere Analyse dieser Tendenzen hätte jedoch den Rahmen dieser Arbeit überschritten, da alleine die Mailingliste von 2017 bis zum Bekanntwerden der Sicherheitslücke über 8500 Emails enthält. Dies wäre jedoch ein guter Ansatzpunkt für weitere Forschung auf dem Gebiet.

> **Project Lead:**

[13] https://logging.apache.org/log4j/2.x/index.html
[14] https://issues.apache.org/jira/browse/LOG4J2-151

> I am quite sure <PMC Member A> is well aware of the rules. He also knows me and that I would cancel the vote if anyone cast a serious -1 as that is not something we do lightly - or even if others agreed that a respin from trunk should be done.
>
> Notice that even though <PMC Member A> would prefer a different outcome he still voted +1 on the release. To me, that is a true demonstration of character. IMO, that is what makes the ASF work so well, as cumbersome as it might seem to some people. As such, I think this is the perfect time to thank <PMC Member A> for all his dedication to this project, along with the other PMC members and committers. Despite disagreements and bumps in the road, you have no idea what a pleasure it is to work with you all. [...]
>
> **PMC Member B:**
>
> Agree with <Project Lead> on all points. It's a pleasure to work with all of you and I'm proud to be part of this team!
>
> **PMC Member C:**
>
> Ditto! It's been fun working with you guys. :)
>
> **PMC Member A:**
>
> No complaints here :-) only appreciation.
>
> **PMC Member D:**
>
> Great work guys! It was a long road. [15]

In der Einstellung der 'Projektleiter' ist ein sehr starker Kontrast festzustellen. Während in Log4j 1 "95% aller Änderungsvorschläge abgelehnt" wurden, werden in Log4j 2 neue Features begrüßt, vor allem wenn diese schon als fertige Patches vorliegen, die lediglich integriert werden müssen. Wie genau neue Features in das Projekt integriert werden wird unter 3.2 erklärt.

So sagt er, dass, falls es darum geht eine schnelle Lösung für ein Problem zu implementieren, dies am besten selbst übernommen wird.

> **Project Lead:**
>
> We are all volunteers here and none of our day jobs directly pay us to work on Log4j. So if you need something quickly we highly encourage you to submit a patch with the particular enhancements or features you are looking for, along with corresponding tests. We are pretty quick about reviewing and applying good patches. If you need help we are also pretty good at responding to emails.

Ein weiteres Beispiel aus einer Diskussion zum Filtern von Zeilenumbrüchen in Layout-Patterns zeigt uns erneut, dass Änderungswünsche mit Implementierung im Projekt

[15] https://lists.apache.org/thread/w353zzgdov7m45o8yq188zdzrghhtrw2

gern gesehen sind.

> **Contributor:**
>
> This isn't really a duplicate, and isn't really resolved with `%enc{}`. Currently `%enc` forces you down HTML or JSON routes, but doesn't have a simple CRLF encoding. As such, you get HTML'ized logs, when all you might want is CRLF injection avoidance.
>
> I'm happy to provide a patch to extend the `%enc` to support a new type of encoding for just CRLF.
>
> **Project Lead:**
>
> Yes. Issues with patches are always more likely to be accepted. I should point out that instead of attaching a patch you can also create a pull request at github.

Dies hat mich zu der Überlegung gebracht, dass sich der Umfang des Projektes sehr stark aufgebläht hat, wodurch die Projektteilnehmer den Überblick über das Projekt etwas verloren haben. Dies trägt weiterhin dazu bei, dass durch Zusammenwirken von unterschiedlichen Systemkomponenten unerwünschte Nebeneffekte auftreten können. Der Projektleiter teilt seine Bedenken zu Software Bloat in seiner Email 'The shape of Log4j'.

> **Project Lead:**
>
> You can view it either way. I look at your position of insisting the stuff stay in the main build as being obstructionist on the one hand and creating a bloated mess on the other.
>
> I've been complaining about the build for at least a year and finally feel the issue needs to be forced. Plus, I see us keep adding modules for things that are not related to what, in my opinion, is the main purpose of the main repo - to build a Java logging framework that meets the vast majority of users needs.
>
> Other things, like Mongo appenders, are esoteric add-ons that don't belong there. Over time I can see this growing to outlandish proportions as we keep adding things that allow you to integrate with whatever the latest thing is.
>
> [...]
>
> I have no problem with those integrations but they have no business being in the main build. I just don't see how you could possibly believe that having integrations with all those components in the main build would be a good thing.[16]

Interessanterweise fragt er im Verlauf desselben Email-Threads danach, wo im Projekt denn JNDI genutzt wird, was dafür spricht, dass er, obwohl er das Feature damals

[16] https://lists.apache.org/thread/0dfxm730p2l40v1qtp78wd9mkbttc6mb

kontrolliert und eingepflegt hatte, den Nutzen bzw. die Existenz der JNDI-Lookups vergessen hatte.

> **Project Lead:**
>
> [...]
>
> Where are we using JNDI?[17]
>
> [...]
>
> **PMC Member A:**
>
> JNDI is used in log4j-web for being able to locate a LoggingContext or configuration (forget which), and there's a JNDI lookup in log4j-core. Might be other places, but those are the main ones I can remember. [...]

Dies spricht weiterhin dafür, dass durch den immensen Umfang des Projekts selbst dem Projektleiter nicht klar war, das die JNDI-Lookups überhaupt existieren, was eine Entdeckung dieser Sicherheitslücke extrem erschwert. Zumal er als Autor der ursprünglichen Lookup-Funktionalität wohl am besten darüber Bescheid weiß.

Die Projektkultur hat demnach definitiv Einfluss darauf gehabt, dass in Log4j 2 viele neue Funktionen hinzugekommen sind. Dies hat dazu geführt, dass das Projekt sehr stark gewachsen ist, was in Log4j 1 vermutlich so nicht stattgefunden hätte. Trotzdem war ein Bewusstsein für Sicherheitsbedenken im Projekt vorhanden, auch wenn dies zumindest in der öffentlich zugänglichen Mailing-Liste nicht häufig diskutiert wurde.

3 Wie entsteht neuer Code im Projekt?

Um die Qualität ihrer Software zu gewährleisten, hat Log4j mehrere Sicherheitsmaßnahmen entwickelt um Fehler im Code zu minimieren. Neuer Code im Projekt wird von mehreren Projektmitgliedern begutachtet, es müssen Unit-Tests vom Ersteller mitgeliefert werden.[18].

3.1 Wie wird man Committer im Log4j Projekt?

Da es sich bei Log4j 2 um ein Open Source Projekt handelt, darf jeder, der möchte, Änderungsvorschläge in das Projekt einbringen. Wer sich regelmäßig in das Projekt einbringt, kann nach Abstimmung seitens des PMC, den 'Committer'-Status erhalten, welcher ihn dazu berechtigt, selbst Code in das Projekt einzuführen. Dies dauert im Regelfall ca. 6 Monate und wird bei mangelnder Beteiligung am Projekt auch wieder entzogen. Im Fall der Log4Shell Sicherheitslücke handelte es sich hier jedoch nicht um einen Committer aus dem Log4j Projekt, sondern eine externe Person, die jedoch in mehreren anderen Apache Projekten mitwirkt.

[17] https://lists.apache.org/thread/3y8ngqx22pprl16l6zdxvx7pw0oqprxr
[18] https://logging.apache.org/log4j/2.x/guidelines.html

3.2 Neue Features

Wenn jemand neue Features in das Projekt einbringen möchte, geschieht dies dadurch, dass er eine Feature-Anfrage an das Projekt stellt, in der er seine Idee vorbringt und wenn möglich seine Implementierung eben dieser. Sollte keine eigene Lösung eingebracht werden, kann es entweder sein, dass

a) jemand aus dem Projekt sich der Implementierung des Features widmet,

b) das Feature abgelehnt wird,

c) die Lösung auf einen späteren Zeitpunkt verschoben wird, oder

d) nach Beiträgen anderer Mitwirkender des Projektes gefragt wird und bei etwaigen Fragen an die Entwickler-Mailingliste verwiesen wird.

3.2.1 Abstimmungen zu neuen Versionen

Immer wenn eine neue Version der Software veröffentlicht werden soll, werden alle Änderungen seit der letzten Version gesammelt und es wird eine öffentliche Abstimmung in der Entwickler-Mailingliste abgehalten. Hierbei sind alle Projektteilnehmer angehalten, sich die Änderungen anzusehen und diese zu testen. Bindendes Stimmrecht haben hier jedoch nur PMC-Mitglieder und es muss ein sogenanntes “lazy approval” erreicht werden. Dafür haben sie die Möglichkeit entweder für die Änderungen zu stimmen (+1), ein Veto gegen die Änderungen einzulegen (-1), oder keine Präferenz zu haben (+0). Abstimmungen sind für 72 Stunden geöffnet und können bei Bedarf verlängert werden. Um den Releaseprozess fortsetzen zu können, müssen mindestens drei Mitglieder dafür gestimmt haben, ohne dass es eine Veto Stimme gab. Kommt es zu einem Veto, muss diese Person erklären, warum sie gegen den Release stimmt. Danach ändern sich die Anforderungen an das Stimmergebenis: entweder muss ein Konsens entstehen, oder es muss eine Mehrheit mit mindestens drei +1 Stimmen und mehr +1 als -1 Stimmen erreicht werden. Dies wird je nach Gegenstand der Abstimmung entschieden[19].

4 Abstimmungsverhalten

4.1 Tendenzen im Abstimmungsverhalten

Nachdem ich herausgefunden habe wie neuer Code in das Projekt einfließt, habe ich mich gefragt, ob im Falle der Einführung der kritischen JNDI-Lookups diese Maßnahmen durchgeführt und eingehalten wurden oder ob eventuell ihre eigenen Regeln nicht eingehalten wurden.

Dazu habe ich in den älteren Mailing Listen Release-Abstimmungen untersucht, um eventuelle Regelmissachtungen feststellen zu können. Hierbei habe ich keinerlei Indizien dafür gefunden, dass sie sich nicht an ihre eigenen Regeln gehalten hätten.

[19] https://logging.apache.org/log4j/2.x/guidelines.html

In diesem Beispiel zum Release der Verison 2.0-beta4 wurden Probleme festgestellt und der Releasevorgang abgebrochen, um diese Fehler zu beheben.

> **Release Manager/Project Lead:**
>
> This is a vote to release Log4j 2.0-beta4, the sixth release of Log4j 2.0.
>
> Changes in this version include:
>
> [...]
>
> **PMC Member:**
>
> [...] Here's the result I get when running mvn test on windows:
>
> `http://pastebin.com/RKKi9X1n`
>
> Flume NG Bridge fails, and several tests are skipped. Do I need to manually install any prerequisites to make these pass. Not sure if this is anything to worry about.
>
> The WARNING at the very top complains because a version is not specified for a plugin in one of the pom.xml files (i attached a patch for that), not critical, can be applied after the release.
>
> Generating the site takes a lot longer than testing for me, and produces heaps of exception traces and some error messages. However, in the end, most sites seem to be generated ok, except for flume-remote (failed) and flume-embedded (skipped). See log here: `http://pastebin.com/b1WpGaR4`
>
> I'd like a little feedback on these before voting, please. Am i doing something wrong?
>
> **Release Manager/Project Lead:**
>
> The error that concerns me is the message that says it cannot create checkpointdir target
> file-channel. Another PMC Member reported that he had a problem on windows but I haven't been able to duplicate it in my VM.
>
> I might have to switch the tests from the file channel to the memory channel just so the tests aren't sensitive to windows.
>
> I tried the fix but still got an error. I will be traveling today so I won't get a chance to look at this until this evening. I guess I will create a new candidate.
>
> Canceling the vote on RC1[20]

Ich habe allerdings bei manchen Abstimmungen festgestellt, dass sich teilweise in den vorgesehenen 72 Stunden nicht genug Mitglieder des PMC zum Thema geäußert haben und dadurch nicht genug Stimmen zusammengekommen sind, um den

[20] https://lists.apache.org/thread/1h7on8fofxx47l1y7qt4t4r6rb6jhfgk

Release der Software zu autorisieren. Daraufhin wurde mehrfach nach Stimmen oder Kommentaren gefragt, um letztlich auf die drei nötigen Stimmen zu kommen.

> **Release Manager/Project Lead:**
>
> 48 hours have passed with no votes or discussion.
>
> [...]
>
> This vote has now been open 5 days and needs one more +1 vote from a PMC member. Again, votes from everyone are welcome.
>
> [...]
>
> Today this vote will have been open 1 week. I would ask that you please find the time to review it and vote. [21]

Die Erwartungshaltung einen Release innerhalb von nur 3 Tagen zu testen könnte unter Umständen zu hastigem Testen eines Releases führen. Dies habe ich in meiner Recherche jedoch nur selten gesehen und halte es zwar für möglich, aber unwahrscheinlich.

Ein weiteres Beispiel für einen Release gemäß der Release Policy von Apache [2] findet sich hier, wo ein Release ordnungsgemäß getestet wird.

> **Release Manager/Project Lead:**
>
> This is a vote to release Log4j 2.11.0, the next version of the Log4j 2 project.
>
> Changes in this version include:
>
> [...]
>
> **PMC Member A:**
>
> +1
>
> From the src zip: ASC, SHA1 OK.
>
> Maven Apache RAT check ... OK.
>
> mvn clean install ... OK
>
> mvn clirr:check -pl log4j-api ... OK
>
> mvn clirr:check -pl log4j-1.2-api ... OK
>
> Using:
>
> [...]
>
> **PMC Member B:**
>
> +1
>
> Site review:

[21] https://lists.apache.org/thread/cnbl7dprs8bhpsr6cy124js2jdth0f88

* Front page header: "h3 Integrating with Application Servers"

* MongoDB component link is a 404

* JIRA report is a blank page

* Surefire report on homepage is a 404

* Wiki link in the header should point to Confluence now

Artifacts check out.

Built and tested with:

[...]

PMC Member C:

+1

checksums good, site looks good (other than the issues already pointed out by others)

Release Manager/Project Lead:

Here is my +1. I will fix the web site issues before I publish the site.[22]

4.2 Der Kritische JNDI-Lookup Patch

Dies führte mich zu der Frage, ob die Mitglieder des PMC bei der Abstimmung zum fraglichen Lookup-Patch eventuell unter Zeitdruck standen und dies zu einer hastigen Entscheidung geführt haben könnte. Dies konnte ich jedoch nach Inspizieren der Abstimmung zur Version 2.0-beta9 [23] nicht verifizieren.

Auch meine Bedenken, dass eventuell diejenigen Mitglieder des PMC dem Release zugestimmt haben, die sonst nicht viele Release- Reviews durchführen, stellte sich als falsch heraus. Tatsächlich wurde die Einführung des JNDI-Lookup-Features während der Abstimmung überhaupt nicht diskutiert, da bei Release einer neuen Version vor allem Maven Build Tests, sowie die für einen Release in der Apache Foundation notwendigen Lizenzen und automatisierte Tests überprüft werden, um gemäß ihrer Release Policy [24] sicherzustellen, dass alle Releaseartefakte signiert sind und alle notwendigen Dateien enthält, um vom Endnutzer gebaut werden zu können.

Zwar sagt die Release Policy weiterhin über das Release-Paket “[...]and that package together with its signature must be tested prior to voting +1 for release [...]” jedoch ist der Umfang eines Releases zu groß, um alle darin befindlichen Änderungen in 72 Stunden gründlich zu testen. Deshalb wird darauf gesetzt, dass der tatsächliche Code Review stattgefunden hat, bevor dieser für den Release freigegeben, beziehungsweise

[22] https://lists.apache.org/thread/scvglnhlyfo0dn6c7n88p6055bqdd2vm
[23] https://lists.apache.org/thread/wgoc8br1x92ylqxj1vg33dwvr960pvkq
[24] https://www.apache.org/legal/release-policy.html#approving-a-release

im Release inkludiert wurde. Zusätzlich sollen so vermutlich möglichst viele unterschiedliche Kombinationen von Systemen getestet werden, da die Mitglieder auf unterschiedlichen Plattformen und Betriebssystemen testen. Sollten sich bei diesen Tests jedoch Fehler auftun, geschieht es häufiger, dass der Release abgebrochen, der Fehler behoben und der Prozess von vorne begonnen wird.

> **PMC Scala Release Agent:**
>
> Hello all, this is the first release candidate for the first stand alone release of the log4j-api-scala modules under their own repository. In order to decouple the version from the log4j-core version release train, we've decided to jump right ahead to 11.
>
> **PMC Member A:**
>
> I just tested this, and it works. But I encountered one issue, when testing from an SBT project, the transitive dependency for log4j-api did not work, I had to specify log4j-api explicitly. When testing from a Maven project, it worked as expected. Have anyone else tried this from SBT? Is this a bug in SBT, or have we done something wrong in the packaging?
>
> [...]
>
> **PMC Member A:**
>
> When running RAT check, it fails with:
>
> Unapproved licenses:
>
> README.md src/site/resources/js/jquery.min.js
>
> I got rid of jquery issue by copying the RAT config in pom.xml from the main project.
>
> For some reason the RAT check does not complain about README.md in the main repo. I cannot see why. <Project Lead>, can you help?
>
> [...]
>
> **PMC Scala Release Agent:**
>
> Thanks for all the polishes! This vote ends without enough votes to pass. I'll make an RC2 when I get a chance.[25]

Auch die Sorge, dass bei diesem Release weniger Personen als sonst getestet haben, hat sich als falsch heraus gestellt - es fand jedoch nur eine Diskussion über Probleme der Lizenzen des Releases statt.

Insgesamt werden hier die intern aufgestellten Regeln zu Abstimmungen eingehalten. Da jedoch eine Diskussion aller Abstimmungen hier den Rahmen sprengen würde, findet sich eine unvollständige Liste mit von mir eingesehen Abstimmungen sowie kurzen Beschreibungen im Anhang.

[25] https://lists.apache.org/thread/lm51kwlkskk01h2pjv1tkgg6yz0wfm17

5 Lookups in Log4j 2

5.1 Einführung von Lookups in Log4j 2

Obwohl die Funktionalität des Log4Shell-Exploits erst im Juli 2013 Teil des Log4j-Projekts wurde, wurde der Grundstein dafür bereits Mitte Oktober 2011 implementiert und war damals Teil der `PatternLayout` Klasse. In diesem Commit[26] wurden die Lookup-Funktion für Umgebungsvariablen sowie Funktionalität für RegEx (Reguläre Ausdrücke) zur Ersetzung des Layout-Patterns hinzugefügt.

Die damalige Klasse `PatternLayout` hatte die Aufgabe, die LogMessage nach dem in der Configuration gespeicherten Layout zu formatieren sowie die weiteren Felder wie den timestamp oder ThreadContext hinzuzufügen, eventuell konfigurierte RegEx-Ersetzungen darauf auszuführen und abschließend Variablensubstitution durch Lookups auszuführen. Das Problem liegt hierbei (wie in 1.4.1 beschrieben), dass diese Substitution Benutzereingaben enthalten kann, auf denen dann Lookups ausgeführt werden können.

Jonathan Michael Edwards beschreibt diese Designentscheidungen in seinem Artikel [3][27] als 'questionable'. Weiterhin stellt er hier die Frage, warum diese Variablensubstitution nicht ausschließlich in der Configuration geschieht. Außerdem merkt er an, dass selbst wenn diese Funktionalität gewünscht ist, die Frage aufgeworfen wird, warum dies dann nicht einfach auf dem Format der Nachricht und nicht derer Argumente angewendet wird, da diese ja nicht vertrauenswürdige Daten beinhalten könnten.

Richtig problematisch wurde dies jedoch erst mit der vorher genannten Einführung von JNDI-Lookups als Teil von Log4j 2.0-beta9. Hier hatte ein Nutzer eine Feature-Anfrage[28] an das Projekt gestellt, welche die Unterstüzung von JNDI- Lookups in der Configuration des Projekts ermöglichen sollte. Wie in Open Source Projekten nicht unüblich hatte der Nutzer an seine Anfrage einen Patch mit der von ihm gewünschten Funktionalität angehängt. Auch in der Dokumentation der Funktion, die der Nutzer beigelegt hatte, wurde ausschließlich über die Lookup-Funktion in der Configuration gesprochen, jedoch nicht darüber, dass diese auch bei Lognachrichten benutzbar ist. Der Patch wurde vom Projektleiter selbst überprüft und wurde mit der nächsten Version Teil des Projekts.

Ob zwischen dem ursprünglichen Entwickler und dem Projektleiter zusätzliche Kommunikation stattgefunden hat, oder ob in der projektinternen Mailingliste über etwaige Sicherheitsbedenken gesprochen wurde, konnte ich leider nicht herausfinden.

5.2 Probleme von Nutzern und deren Behebung

Etwas mehr als ein Jahr später im November 2014 stellt ein Nutzer von Log4j ein Problem fest, wenn er in Verbindung mit Apache Camel, einem weiteren Open Source Framework, versucht, Dateinamen in der Form `"?fileName=${date:now:yyyyMMdd-`

[26] https://github.com/apache/logging-log4j2/commit/a538903a9b5e100b1c3e3a8aec0462b1cb86765c
[27] Kapitel 2011-10-19 – Message Pattern Lookup functionality committed
[28] https://issues.apache.org/jira/browse/LOG4J2-313

`HHmmss}ID.${id}.gz"` anzugeben, da dies in Apache Camel so zulässig ist, in Log4j jedoch als Lookup ausgewertet wird und zu einem Fehler führt.

Daraufhin wurde er gebeten, dieses Problem im Log4j Jira Issue Tracker vorzustellen. Dies geschah unverzüglich [29], wurde jedoch erst fast anderthalb Jahre später vom Projektteam bearbeitet. Die folgende Diskussion ist meiner Meinung nach eine der wichtigsten im Verlauf der Sicherheitslücke, da hier eine gute Chance bestanden hätte, das Problem zu erkennen und so zu lösen, dass es unschädlich gemacht worden wäre.

Hier merkt eines der PMC Mitglieder an, dass sie bei einer eventueller Deaktivierung von Lookups einen Unterschied zwischen Lookups in Lognachrichten und Lookups in der Configuration machen sollten.

> **PMC Member A:** Thinking out-loud: If we provide/document a way to disable lookups, then we should probably make the distinction between lookups being used in configuration files and in messages. IOW, you might want to use a lookup in an XML config file but not in messages. [...]

Die Antwort des Projektleiters darauf war die Einführung einer Option für das Layout der Lognachricht, in welcher der %m Parameter des Layouts um das Argument `{nolookups}` erweitert wurde, was für dieses Layout-Pattern die Lookups abschalten würde.

> **Project Lead:**
>
> IMO, the correct solution is to add an option to `%msg` such as `%msg{raw}` or `%msg{nosubst}`. It would be very easy to just use that to set a boolean value to check in MessagePatternConverter so that instead of
>
> ```
> if (config != null){
> ```
>
> we do
>
> ```
> if (config != null && isSubstitution){
> ```
>
> Then the user could control the behavior by specifying the converter option that disables it. Note that by combing this with PatternSelectors they could enable this just for events from Camel's loggers and leave it enabled for others.
>
> **PMC Member B:**
>
> I like this idea of having an option that disables lookups. Additional naming idea: `%msg{nolookup}`.

Diese Methode die Lookups abschalten zu können ist zwar sehr einfach umzusetzen, wird jedoch sehr aufwendig wenn es darum geht, viele unterschiedliche Pattern deaktivieren zu können, woraufhin ein weiterer Nutzer nach der Möglichkeit fragt, eine globale Variable einzuführen, die Lookups in allen Lognachrichten deaktiviert.

> **User:**

[29] https://issues.apache.org/jira/browse/LOG4J2-905

> In addition to this option, will you add a global option to disable lookups in messages ? i.e. in log4j configuration and/or via system property ?
>
> **Project Lead:**
>
> We have no plans to do that as this should give you exactly what you need.

Das Entwicklungsteam scheint jedoch zur Lösung des Problems hier den Weg des geringsten Widerstandes zu gehen, in dem sie eine simple Lösung des akuten Problems wählen, jedoch ohne das grundlegende architektonische Problem der Lookups zu lösen. Dennoch ist dieser Ansatz nicht abwegig, da die Implementierung von neuen Funktionen aufgrund eines relativ kleinen Teams von Freiwilligen sehr langsam ist, was auch daran zu erkennen ist, dass die Bearbeitung dieses Problems erst anderthalb Jahre nachdem es berichtet wurde begonnen hat. Zusätzlich spricht der Projektleiter bei einer anderen Featureanfrage davon, dass die Implementierung von neuen Funktionen wenigstens ein paar Tage und höchstens ein Jahr dauern würde.

> **Project Lead:** All that said, it will certainly be longer than days and shorter than a year before I would get to this. This issue is one that I find interesting so it would probably be at, or near, the top of my list.[30]

Die tatsächliche Implementierung einer globalen Variable zur Deaktivierung von Message Lookups wurde über ein Jahr später in Angriff genommen, um nicht für jedes einzelne MessageLayoutPattern einzeln die `%m{nolookups}` setzen zu müssen[31]. Hierbei wurde die Variable `LOG4J_FORMAT_MSG_NO_LOOKUPS` Umgebungsvariable eingeführt, sowie die Alternative `log4j.formatMsgNoLookups`, die als property gesetzt werden konnte. Unglücklicherweise wurde diese in der Dokumentation fälschlich als `FORMAT_MESSAGES_PATTERN_DISABLE_LOOKUPS` [32] angegeben, was letztlich dazu führte, dass erste Vermeidungsstrategien für den Exploit fehlschlugen, da die Umgebungsvariable zwar gesetzt werden konnte, aber keine Funktion inne hatte.

Ein weiteres Problem der Lookup-Funktionalität in Log4j war, dass die Tatsache, dass sie existiert, nur sehr schwer zugänglich war. In der ursprünglichen Dokumentation der JNDI-Lookups war zumindest von Message-Lookups keine Rede.

> It would be really convenient to support JNDI resource lookup in the configuration.[33]

Erste Hinweise hierfür finden sich, wie in [3] beschrieben, erst in der Dokumentation zur Bestrebung des Projektes 'garbage collection free' zu werden, also möglichst zu vermeiden, dass zur Programmlaufzeit Ressourcen aufgewendet werden müssen, um nicht mehr benötigte Speicherbereiche wieder freizugeben. Hierzu besagt das Jira-Issue dass alle Nachrichten, welche ein `"${...}"` beinhalten, zu einem String umgewandelt und, wie in 1.4.1 beschrieben, mittels Variablensubstitution ersetzt werden.

> Messages containing `"${"` will be converted to a String and StrSubstitutor

[30] https://issues.apache.org/jira/browse/LOG4J2-2464
[31] https://issues.apache.org/jira/browse/LOG4J2-2109
[32] https://github.com/apache/logging-log4j2/pull/127/commits/cd55baa26c34f2f54aee57ba7cb725d2b81724a1
[33] https://issues.apache.org/jira/browse/LOG4J2-313

> will be used to replace occurences of variables with their matching values. Multiple temporary objects are created during this process.[34].

Es dauerte schließlich bis Dezember 2020, bis ein weiterer Nutzer nach Problemen mit Message Lookups dokumentierte, dass alle Lognachrichten standardmäßig auf Lookups untersucht werden, und wie dies abzuschalten ist[35].

All dies zeigt, dass die Funktionalität der Lookups in Lognachrichten über lange Zeit sehr schlecht dokumentiert war, was bei Nutzern zu ungewolltem Verhalten der Software geführt hat. Die Bearbeitung dieser Probleme erfolgte aufgrund von mangelnder Arbeitskapazität jedoch sehr langsam, was auch dazu geführt haben könnte, dass einfache Lösungen gegenüber einer aufwendigeren Behebung der grundlegenden Ursache bevorzugt wurden.

6 Fazit

In meiner Arbeit habe ich untersucht, welche Ursachen der Log4Shell Sicherheitslücke zugrunde liegen. Dabei habe ich zunächst untersucht, wie im Projekt mit dem Thema Sicherheit umgegangen wird und habe dabei festgestellt, dass die Projektmitglieder durchaus auf das Thema Sicherheit achten, dies aber öffentlich selten angesprochen wird.

Daraus habe ich geschlussfolgert, dass sich meine Hypothese eines mangelnden Sicherheitsverständnisses seitens der Projektteilnehmer nicht bewahrheitet hat.

Anschließend habe ich untersucht, ob der Unterschied in der Kultur des Projektes im Vergleich zum Vorgängerprojekt Auswirkungen auf die Sicherheitslücke gehabt hat. Da in Log4j 2 neue Features begrüßt wurden, während in Log4j 1 die meisten Änderungsvorschläge abgelehnt wurden, hat dieser Kulturunterschied maßgeblich dazu beigetragen, dass das Log4j 2 Projekt stark anwuchs. Dies führte dazu, dass der Projektleiter, der die Einführung des JNDI-Features damals durchgeführt hatte, vergaß, dass diese Funktionalität überhaupt existierte.

Erschwerend kam hinzu, dass die Funktionalität der JNDI-Lookups in Lognachrichten über weite Teile des Projektzeitrahmens sehr schlecht dokumentiert war. So wurde selbst nachdem mit dem Feature Probleme bei Nutzern auftraten der zugrunde liegende Code nicht ausreichend überprüft, um diese Sicherheitslücke zu erkennen und zu schließen. Denn schon die Art der ursprünglichen Implementierung enthielt einige fragwürdige Architekturentscheidungen, welche diese Sicherheitslücke ermöglichten. Auf meine Anfragen bezüglich eines Interviews mit den beteiligten Personen des PMC, um Klarheit für die Gründe hinter diesen Entscheidungen zu erlangen, erhielt ich leider keine Antwort. Dies wäre ein möglicher Ansatzpunkt für weiterführende Forschung auf diesem Gebiet. Bei der vorläufigen Behebung eines der Nutzerprobleme wurde eine wenig arbeitsintensive Lösung gewählt, was wahrscheinlich auf der immensen Größe des Projektes beruhte. Hier hätte eine besserere Diskussion dazu führen können, dass die grundlegende Architektur der Funktionalität überarbeitet

[34] https://issues.apache.org/jira/browse/LOG4J2-1297
[35] https://github.com/apache/logging-log4j2/pull/450/

und die Sicherheitslücke damit geschlossen wird. Auch die spätere Einführung einer globalen Variable zur Abschaltung der Lookups in Lognachrichten hätte dazu beitragen können. Ein Fehler in der Dokumentation dieser Option führte jedoch dazu, dass die ersten Mitigationsstrategien des Exploits fehlschlugen.

Beim Sichten von Abstimmungen im Log4j 2 Projekt konnte ich nicht feststellen, dass die vom Projekt aufgestellten Regeln zur Veröffentlichung von neuen Softwareversionen zu irgendeinem Zeitpunkt missachtet wurden.

Daraus schließe ich, dass die Sicherheitslücke Log4Shell dadurch entstanden ist, dass fragwürdige Designentscheidungen getroffen wurden, die aufgrund der enormen Ausdehnung des Projektes unentdeckt geblieben sind. Hierbei interagierten mehrere Funktionen in wenig bis schlecht dokumentierter Weise, um die Sicherheitslücke zu ermöglichen. Auch bei späteren Nutzerproblemen wurde diese Problematik nicht erkannt. Selbst nachdem Schritte zur Umgehung von unerwünschtem Verhalten bei Nutzern eingeleitet wurden, blieb vermutlich aus Aufwandsgründen eine Restrukturierung der Funktionalität aus.

6. Fazit

Literatur

[1] *10 advantages of open source for the enterprise*. URL: https://opensource.com/article/17/8/enterprise-open-source-advantages.

[2] *Best Practices Release Creation Process*. URL: https://www.apache.org/legal/release-policy.html.

[3] Jonathan Michael Edwards. *Log4j vulnerability: What the FAQ happened?* 15. Dez. 2021. URL: https://jedwidz.hashnode.dev/log4j-vulnerability-what-the-faq-happened.

[4] Ruth Fultherer. *Log4j wurde 1997 in der Schweiz entwickelt – der Erfinder erzählt*. 16. Dez. 2021. URL: https://www.nzz.ch/technologie/log4j-wurde-1997-in-der-schweiz-entwickelt-der-erfinder-erzaehlt-ld.1660571.

[5] *Log4J Developer Mailing List 2000-2017*. URL: https://lists.apache.org/list?log4j-dev@logging.apache.org.

[6] *Log4J Developer Mailing List 2017-2023*. URL: https://lists.apache.org/list.html?dev@logging.apache.org.

A Anhang A

A.1 Liste der untersuchten Abstimmungen

25.08.07	log4j 1.2.15-rc6	Diskussion mit anschließendem Release	https://lists.apache.org/thread/4lkycvqh8n0cz150ck3nntcjt3rpkw9l
25.02.05	log4j 1.2	Hitzige Diskussion mit Veto, danach lazy majority voting, keine Änderungen	https://lists.apache.org/thread/346gd3ohjznyd6n1vd84q9lgqf0krskw
25.08.12	Log4j 2.0-alpha2	Kurze Diskussion mit anschließendem Release	https://lists.apache.org/thread/scvglnhlyfo0dn6c7n88p6055bqdd2vm
20.01.13	Log4j 2.0-beta4 rc1	Abstimmung, finden eines Fehlers, Versuch Problem zu beheben, Abstimmung abgebrochen	https://lists.apache.org/thread/1h7on8fofxx47l1y7qt4t4r6rb6jhfgk
28.01.13	Log4j 2.0-beta4 rc2	Release nach Testen	https://lists.apache.org/thread/j82249vmd4r3dxhydpbpc0tw490ftdrf
14.09.13	Log4J 2.0-beta9	Kurze Diskussion über Releaseartefakte mit anschließendem Release	https://lists.apache.org/thread/wgoc8br1x92ylqxj1vg33dwvr960pvkq
15.04.17	log4j-api-scala 11.0 rc1	Testing und abschließendem Nicht-release nach Problemen	https://lists.apache.org/thread/lm51kwlkskk01h2pjv1tkgg6yz0wfm17
27.08.17	Log4j 2.9.0-rc1	Release nach Testen	https://lists.apache.org/thread/c94hpl894mo0o87hsxwj9k8wtf7gwqr0
13.11.17	Apache Chainsaw 2.0.0-rc1	Abstimmung mit Testen & Bug Fixes, kein Release	https://lists.apache.org/thread/r91gcwr44pcconk6l20ov2l66hbfw1md
23.11.17	Log4j 2.10.0-rc1	Abstimmung mit Veto, danach lazy majority und Release	https://lists.apache.org/thread/ol81h48xwvjnzgmmpm4h0do2m83xhs7m
20.01.18	Apache Chainsaw 2.0.0-rc2	Abstimmung mit Testen und anschließendem Release	https://lists.apache.org/thread/3063mc8q8mht8kpc6s64stvbkotolqc7
18.03.18	Log4j 2.11.0-rc1	Abstimmung mit Testen und anschließendem Release	https://lists.apache.org/thread/m3hy8sdxr28otm4k1f553xn71b93wpfy

Continued on next page

Tabelle 0: (Continued)

30.04.18	Migrate git repos to gitbox	Abstimmung mit kurzer Erläuterung	https://lists.apache.org/thread/n6cm68w76m9qb15vdnzkxrt5ox1vpq27
21.06.18	Log4j Audit 1.0.0 RC1	Langsamer Abstimmungsprozess mit anschließendem Release	https://lists.apache.org/thread/cnbl7dprs8bhpsr6cy124js2jdth0f88
24.02.20	Log4j 2.13.1	Veto ohne Release aufgrund von Dependency-Fehlern	https://lists.apache.org/thread/tzxjw52w3ty2rdt8d3fbzmlcmrj1ntwb

B Glossar

ASF - Apache Software Foundation

Committer - Person mit Rechten, um eigenen Code in das Log4j-Projekt zu laden

Commit - Programmteile, die zusammen als Arbeitspacket in ein Projekt eingefügt werden

Exploit - Schadrogramm oder Abfolge von Tätigkeiten, die Schwachstellen oder Fehlfunktionen in anderer Software ausnutzt

PMC - Project Management Committee

Refactoring - Umgestaltung oder Umbenennung von Codeteilen