

FH Aachen

FB9 Medizintechnik und Technomathematik

Angewandte Mathematik
und Informatik

Seminararbeit

Untersuchung der Maßnahmen zur Gewährleistung von Sicherheit
bei Nutzung einer Redpanda Message Queue

Kilic, Meryem
Matrikel-Nr.: 3646674

Aachen, 28. November 2025

Erstprüfer: Rohde, Philipp, Prof. Dr.
Zweitprüfer: Mauthe, Christopher-Jannik

Abstract

In den letzten Jahren haben sich Message Queuing Systeme als essenzielle Komponente moderner verteilter Software-Architekturen etabliert. Sie ermöglichen asynchrone Kommunikation, erhöhen die Zuverlässigkeit von Systemen und unterstützen deren horizontale Skalierbarkeit. Das Federated-Identity-Management-Team bei CANCOM Managed Services GmbH (im Folgenden mit FIM-Team abgekürzt) befindet sich aktuell in einem Umstrukturierungsprozess, in dessen Rahmen eine bestehende Software-Lösung auf den Einsatz einer Message Queue umgestellt wird. Ziel dieser Arbeit ist es, die dabei eingesetzte Technologie zu analysieren, getroffene Entscheidungen sicherheitstechnisch einzuordnen und Handlungsempfehlungen für eine langfristig sichere Architektur abzuleiten.

Zu Beginn werden grundlegende Eigenschaften von Message Queues erläutert und deren Stärken im Vergleich zu direkten Kommunikationsmodellen herausgearbeitet. Anschließend wird die konkrete Architektur des FIM-Teams untersucht, um die getroffenen Sicherheitsmaßnahmen entlang des Entwicklungsprozesses zu bewerten und Empfehlungen auszusprechen. Als Bewertungsrahmen dienen die OWASP Top Ten 2021, ein etablierter Standard zur Identifikation und Priorisierung von Sicherheitsrisiken in Softwaresystemen. Die Analyse zeigt, dass das FIM-Team bereits ein gut durchdachtes Sicherheitskonzept verfolgt und zahlreiche Risiken aus den OWASP Top Ten adressiert. Unter anderem wird der Zugang zur Message-Queue-Infrastruktur architekturbedingt stark eingeschränkt: Externe Nutzer:innen besitzen keinen direkten Zugriff, und interne Mitarbeiter:innen interagieren ausschließlich über abgeschottete Netzwerkbereiche. Zugriffsbeschränkungen werden granular mittels Access Control Lists umgesetzt und Passwörter ausschließlich verschlüsselt übertragen. Darüber hinaus verhindern einheitliche Nachrichtenformate und definierte Verarbeitungsketten die Einspeisung unerlaubter Inhalte, während externe Software-Komponenten bewusst aus vertrauenswürdigen und regelmäßig überprüften Quellen bezogen werden, um Risiken durch kompromittierte Software zu vermeiden.

Trotz des hohen Sicherheitsniveaus wurden Optimierungspotenziale identifiziert. Besonders relevant ist die Einführung einer durchgängigen TLS-Verschlüsselung zur Absicherung der Kommunikation zwischen Systemkomponenten. Aktuell wird dieses Risiko durch eine stark abgeschottete Netzwerkarchitektur kompensiert. Perspektivisch kann eine Migration auf Kubernetes dabei unterstützen, TLS automatisiert auszurollen, ohne die interne Exklusivität des Systems aufzugeben. Außerdem gibt es konfigurationstechnisch leichtes Verbesserungspotenzial. Darüber hinaus empfiehlt es sich, verwendete Software-Versionen regelmäßig zu aktualisieren, um sicherheitsrelevante Verbesserungen zeitnah zu übernehmen. Obwohl die derzeit eingesetzten Versionen nicht veraltet sind, existieren bereits neuere Varianten, deren Nutzung zusätzliche Sicherheit bieten kann. Ergänzend wird für den Produktivbetrieb der Einsatz eines Security-Monitorings empfohlen, etwa durch die Kombination von Loki und Grafana, um sicherheitsrelevante Ereignisse systematisch und visuell nachvollziehen zu können.

Diese Arbeit stellt dem FIM-Team eine strukturierte Übersicht über bestehende und empfohlene Sicherheitsmaßnahmen zur Verfügung. Sie unterstützt die Einordnung neuer Sicherheitsansätze im Kontext der bestehenden Architektur und macht transparent, welche konkreten Risikokategorien durch welche Maßnahmen adressiert werden. Da Software-Sicherheit ein dynamisches Feld darstellt, wird im Ausblick auf die Notwendigkeit kontinuierlicher Sicherheitsanalysen hingewiesen. Zum Zeitpunkt der Erstellung dieser Arbeit wurden die OWASP Top Ten 2025 veröffentlicht, welche neue Risikofaktoren berücksichtigen. Auch insbesondere im Zusammenhang mit dem wachsenden Einsatz von KI-basierten Systemen und Large Language Models kommen stetig neue Gefahren für Softwaresysteme dazu. Dadurch wird deutlich, dass Sicherheitsmaßnahmen stetig an neue technologische Entwicklungen angepasst werden müssen.

Abschließend zeigt sich, dass eine sichere Software-Entwicklung nur durch fortlaufende sicherheitstechnische Überprüfungen gewährleistet werden kann. Die vorliegende Arbeit leistet hierzu einen Beitrag, indem sie bestehende Maßnahmen analysiert, entlang etablierter Standards bewertet und konkrete Handlungsempfehlungen zur weiteren Stärkung der Systemsicherheit formuliert.

Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen	2
2.1	Message Queuing: Grundlegende Erläuterungen	2
2.1.1	Definitionen	2
2.1.2	Arten von Message Queues	3
2.1.3	Abwägung zur Nutzung von Message Queues	4
2.2	Eingesetzte Technologien	5
2.2.1	Apache Kafka	5
2.2.2	Redpanda	6
2.2.3	Spring Boot	8
2.2.4	Terraform und OpenTofu	8
2.3	OWASP	9
2.3.1	OWASP Foundation	9
2.3.2	OWASP Top Ten	9
3	Einordnung sicherheitsrelevanter Entscheidungen auf Basis der OWASP Top Ten 2021	10
3.1	Broken Access Control	10
3.2	Cryptographic Failures	10
3.3	Injection	12
3.4	Insecure Design	13
3.5	Security Misconfiguration	15
3.6	Vulnerable and Outdated Components	16
3.7	Identification and Authentication Failures	17
3.8	Software and Data Integrity Failures	17
3.9	Security Logging and Monitoring Failures	18
3.10	Server-Side Request Forgery	19
3.11	Ergebnisse und Diskussion	19
4	Ausblick	20

1 Einleitung

Das Team „Federated-Identity-Management“ der CANCOM Managed Services GmbH, im Folgenden FIM-Team genannt, bietet seinen Kund:innen einen virtuellen Arbeitsplatz an, das sogenannte „FIM-Portal“.

Dieses Portal stellt eine zentrale Plattform bereit, über die verschiedene Dienste genutzt werden können. Dazu zählen beispielsweise Mailing-Dienste, Kalenderfunktionen, Cloud-Speicher oder weitere Anwendungen, die den Arbeitsalltag der Nutzer:innen erleichtern und die Effizienz innerhalb ihrer Organisation steigern.

Das Portal dient somit als integrative Lösung, die unterschiedliche Funktionalitäten bündelt und den Kund:innen eine einheitliche Benutzeroberfläche bietet.

Derzeit basiert das eingesetzte System auf einem stark gekoppelten verteilten Monolithen. Diese Architektur bringt mehrere Herausforderungen mit sich:

Zum einen erschwert die enge Verzahnung der Komponenten die Wartung des Systems erheblich: Änderungen in einem Bereich können unbeabsichtigte Auswirkungen auf andere Teile der Anwendung haben, wodurch Fehleranfälligkeit und Wartungsaufwand steigen.

Zum anderen gestaltet sich die Implementierung kundenspezifischer, nicht standardisierter Anwendungsfälle kompliziert, da Anpassungen häufig tiefgreifende Änderungen am gesamten System erfordern.

Möchten zwei Kund:innen im FIM-Portal beispielsweise über verschiedene Tools Cloud-Speicher nutzen, ist es mit der aktuellen Architektur sehr schwierig, auf diese beiden Anforderungen gleichzeitig einzugehen.

Insgesamt wird durch die derzeitige Struktur die Agilität bei der Weiterentwicklung des Portals deutlich eingeschränkt, was insbesondere in einem dynamischen Geschäftsumfeld problematisch sein kann.

In modernen verteilten Softwaresystemen haben sich Message Queues als zentrale Technologie etabliert. Sie ermöglichen eine skalierbare, asynchrone und fehlertolerante Kommunikation zwischen einzelnen Komponenten, ohne dass diese direkt miteinander gekoppelt sind. Dadurch können Lastspitzen besser abgefangen, Systeme resilienter gestaltet und die Verarbeitung von Events zuverlässig sichergestellt werden. Insbesondere in Microservice-Architekturen bieten Message Queues eine Möglichkeit, Daten und Ereignisse effizient zwischen verschiedenen Diensten auszutauschen und so die Gesamtarchitektur flexibler und wartbarer zu gestalten.

Vor diesem Hintergrund plant das FIM-Team, die bestehende Software-Lösung schrittweise auf den Einsatz von Message Queues für die Event-Verarbeitung umzustellen. Zukünftig sollen sowohl von Nutzer:innen initiierte Events, wie beispielsweise die Beantragung eines E-Mail Postfachs oder die Änderung eines Benutzerprofils, als auch automatisch zwischen Applikationen ausgetauschte Events über ein Message Queuing System verarbeitet werden. Ziel ist es, die Kommunikation innerhalb des Systems zu entkoppeln, die Skalierbarkeit zu erhöhen und die Grundlage für eine resilientere und wartungsfreundlichere Software-Architektur zu schaffen.

Die vorliegende Arbeit beschäftigt sich mit den sicherheitsrelevanten Designentscheidungen, die im Rahmen dieser Umstrukturierung getroffen werden müssen. Insbesondere wird untersucht, inwiefern die geplanten Entscheidungen Risiken adressieren, die in den OWASP Top Ten 2021 identifiziert wurden. Darüber hinaus werden Empfehlungen für eine sichere Gestaltung ausgesprochen, um potenzielle Sicherheitslücken frühzeitig zu minimieren und ein robustes, zukunftsfähiges System zu gewährleisten.

2 Grundlagen

2.1 Message Queuing: Grundlegende Erläuterungen

2.1.1 Definitionen

In der *Encyclopedia of Database Systems* werden „Message Queues“ wie folgt definiert:

”A message is an information sent by a sender process to a receiver process. A message queue is a mechanism that allows a sender process and a receiver process to exchange messages. The sender posts a message in the queue, and the receiver retrieves the message from the queue.” [1]

Dies bedeutet, dass in einem Message Queuing System der Sender- und Empfängerprozess nicht direkt miteinander kommunizieren. Stattdessen wird die Nachricht des Senders an die Message Queue weitergegeben und der Empfänger erhält die für ihn bestimmte Nachricht aus der Message Queue.

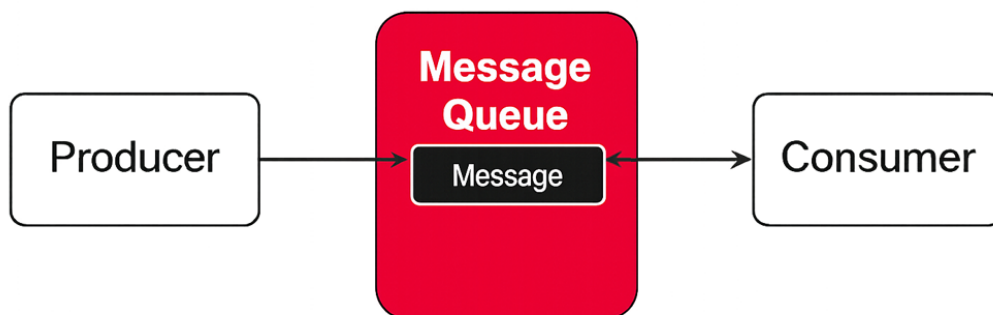


Abbildung 1: Funktionsweise eines Message Queuing Systems. Die Abbildung zeigt, wie Nachrichten vom Producer an die Queue gelangen und von dort vom Consumer abgerufen werden. Dieses Prinzip ermöglicht asynchrone und entkoppelte Kommunikation zwischen den Systemkomponenten. Erstellt mit diagrams.net

Im nachfolgenden Abschnitt werden zentrale Fachbegriffe definiert und erläutert, die im Verlauf dieser Arbeit wiederholt verwendet werden. Ziel ist es, ein einheitliches Verständnis der verwendeten Terminologie zu schaffen und zugleich Begrifflichkeiten, die in der einschlägigen Literatur teilweise synonym oder uneinheitlich verwendet werden, klar voneinander abzugrenzen. Auf diese Weise wird eine konsistente sprachliche Grundlage geschaffen, die das Verständnis der weiteren Ausführungen erleichtert.

Nachricht:

Zusammenschluss von Informationen, die von einem *Producer* an einen *Consumer* gesendet werden

Producer:

Der Senderprozess, der Nachrichten in die *Message Queue* einstellt

Consumer:

Der Empfängerprozess, der Nachrichten aus der *Message Queue* abrufen und verarbeitet

Message Queue:

Struktur, über die Programme Nachrichten austauschen können. Wird an einigen Stellen mit *Queue* abgekürzt

Message Queuing System:

Softwaresystem, welches aus einer *Message Queue* und mindestens einem *Producer* und *Consumer* besteht

2.1.2 Arten von Message Queues

Es existieren zwei gängige Arten von Message Queuing Systemen:

Point-to-Point Messaging

Das Point-to-Point-Messaging beschreibt ein Kommunikationsmodell, bei dem Nachrichten von einem Producer gezielt an einen einzelnen Consumer übermittelt werden. Nachdem eine Nachricht erfolgreich empfangen und verarbeitet wurde, wird sie aus der Queue entfernt. Dieses Modell gewährleistet, dass jede Nachricht ausschließlich einmal und nur von einem Consumer verarbeitet wird.

Grundsätzlich lassen sich zwei Ausprägungen des Point-to-Point-Messaging unterscheiden: Zum einen das *Fire-and-Forget*-Modell, bei dem der Producer eine Nachricht übermittelt, ohne auf eine Antwort zu warten, und zum anderen das *Request-Response*-Modell, bei dem der Consumer nach der Verarbeitung der Nachricht eine Rückmeldung an den Producer sendet.

Typische Anwendungsfälle ergeben sich insbesondere in Szenarien, in denen Aufgaben sequentiell abgearbeitet werden sollen, beispielsweise bei Job- oder Task-Processing-Systemen, in denen einzelne Aufgaben von jeweils einem Consumer verarbeitet werden. [2]

Message-Broker-Systeme

In Broker-basierten Nachrichtensystemen übernehmen sogenannte *Message Broker* die Vermittlung, Verarbeitung und Weiterleitung von Nachrichten zwischen Produzern und Consumern. Dabei können Message Broker mehrere Message Queues verwalten. Die Nachrichtenverteilung kann auf vordefinierten Regeln basieren und unterstützt damit eine flexible und skalierbare Architektur.

Ein Message Broker kann zudem unterschiedliche Kommunikationsmuster realisieren, darunter beispielsweise das Publish-Subscribe-Modell. Dabei veröffentlicht ein Producer (*Publisher*) eine Nachricht zu einem bestimmten Thema (*Topic*), welches vom Broker verwaltet wird.

Der Broker speichert, sortiert und verteilt die Nachricht anschließend an alle registrierten Consumer (*Subscriber*). Dieses Vorgehen ermöglicht es, identische Informationen simultan an mehrere Systeme oder Dienste zu übermitteln. Typische Einsatzgebiete solcher Broker-Systeme sind Echtzeit-Analysen, Monitoring- und Benachrichtigungssysteme, bei denen mehrere Komponenten parallel auf eingehende Daten reagieren müssen. [2]

2.1.3 Abwägung zur Nutzung von Message Queues

Vorteile

Der Einsatz von Message Queues bringt zahlreiche organisatorische und technische Vorteile mit sich und reduziert gleichzeitig mögliche Nachteile klassischer Kommunikationsmethoden.

In Message Queuing Systemen fällt zunächst auf, dass Producer und Consumer keinen direkten Kommunikationskanal besitzen. Der Nachrichtenaustausch erfolgt ausschließlich über eine zwischengeschaltete Message Queue. Dadurch entsteht eine Entkopplung der Komponenten: Änderungen in einer der beiden Anwendungen haben keine unmittelbaren Auswirkungen auf die jeweils andere.

Zudem vereinfacht sich die Architektur, da die Queue die gesamte Kommunikationslogik übernimmt und Producer wie Consumer dadurch weniger komplex sind.

Ein weiterer Vorteil liegt in der zeitlichen Asynchronität der Kommunikation. Der Producer muss nicht wissen, ob der Consumer aktuell verfügbar oder in der Lage ist, die Nachricht zu verarbeiten. Die Nachricht verbleibt in der Queue, bis sie vom Consumer abgeholt und verarbeitet wird. Ebenso muss der Consumer nicht aktiv auf eingehende Nachrichten warten, sondern kann andere Aufgaben ausführen, bis eine Nachricht auf der Queue erscheint. Dadurch wird eine flexible und ressourcenschonende Verarbeitung ermöglicht.

Zusätzlich erlaubt Message Queuing eine parallele und verteilte Aufgabenausführung: Anstatt dass ein einzelner, großer Prozess eine Aufgabe vollständig bearbeitet, können mehrere kleinere Prozesse Teilaufgaben übernehmen und ihre Ergebnisse über die Queue austauschen. Dies verbessert die Skalierbarkeit und Auslastung der Systeme.

In Message Queuing Systemen kann die Verarbeitung nachrichtengetrieben (*Message-driven*) oder ereignisgesteuert (*Event-driven*) erfolgen, wobei eine Nachricht selbst ein Ereignis darstellen kann. Diese Flexibilität erleichtert die dynamische Steuerung von Prozessen. Darüber hinaus können Nachrichten priorisiert werden, sodass wichtige Ereignisse bevorzugt verarbeitet werden. [3]

Nachteile

Neben den zahlreichen Vorteilen bringt der Einsatz von Message Queues jedoch auch einige Herausforderungen mit sich, die bei der Implementierung und dem Betrieb berücksichtigt werden sollten.

So können Nachrichten beispielsweise durch Systemausfälle oder Netzwerkfehler verloren gehen. Je nach Bedeutung der betroffenen Nachricht kann dies von einem geringfügigen Problem bis hin zu einem kritischen Systemfehler führen.

Auch die Reihenfolge der Nachrichtenverarbeitung stellt eine mögliche Schwierigkeit dar: Nachrichten, die in einer bestimmten Reihenfolge gesendet werden, können unter Umständen in abweichender Reihenfolge ankommen. In Anwendungen, in denen die korrekte Abfolge essenziell ist, kann dies zu unerwünschten Inkonsistenzen führen.

Ein weiteres Risiko besteht in sogenannten *Poison Messages*. Das sind Nachrichten, die aufgrund fehlerhafter Formate oder unvollständiger Implementierungen auf Seite der Consumer nicht verarbeitet werden können.

Darüber hinaus kann eine zu hohe Anzahl gleichzeitig versendeter Nachrichten zu einer Überlastung des Systems führen. In solchen Fällen steigt die Gefahr von Verzögerungen oder gar Systemausfällen. [4]

Entscheidung

Nach sorgfältiger Abwägung der Vor- und Nachteile hat sich das FIM-Team für den Einsatz eines Message Queuing Systems entschieden. Die genannten Herausforderungen gelten als beherrschbar und können mit geeigneten Mechanismen und Architekturentscheidungen adressiert werden. Diese Lösungsansätze werden in „3 Einordnung sicherheitsrelevanter Entscheidungen auf Basis der OWASP Top Ten 2021“ näher erläutert. Zudem unterstützt die Einführung von Message Queues das Ziel, die bisherige monolithische Systemarchitektur in eine besser wartbare Struktur zu überführen.

Außerdem bietet die Umstellung der bestehenden Architektur auf ein Message Queuing System langfristig den Vorteil, individueller auf Kundenanforderungen reagieren zu können. Durch die entkoppelte Architektur wird es zukünftig einfacher, zusätzliche Microservices, die Kund:innen nutzen möchten, in das System zu integrieren und gleichzeitig nicht benötigte Dienste zu entfernen, ohne bestehende Komponenten zu beeinträchtigen.

Diese Überlegungen bestärkten das FIM-Team darin, die Umstrukturierung hin zu einem Message Queuing System einzuleiten.

2.2 Eingesetzte Technologien

2.2.1 Apache Kafka

Die vom FIM-Team eingesetzte Queue-Technologie Redpanda basiert stark auf Apache Kafka (im Folgenden Kafka genannt) und ist zudem vollständig Kafka-kompatibel. Aus diesem Grund folgt zunächst eine grundlegende Erläuterung über die Funktionsweise von Kafka.

Kafka ist eine verteilte, fehlertolerante und persistente Streaming-Plattform, die als Queue genutzt werden kann und unter Entwicklern große Beliebtheit genießt.

Ursprünglich wurde Kafka in Scala entwickelt und basiert auf dem TCP-Protokoll.

Nachrichten in Kafka werden in sogenannte Topics organisiert, wobei jedes Topic in mehrere Partitionen unterteilt werden kann. Diese Partitionen werden auf unterschiedlichen Brokern verteilt gespeichert. Durch die Partitionierung wird eine gleichmäßige Verteilung der Nachrichten erreicht, was die Parallelität des Systems deutlich erhöht.

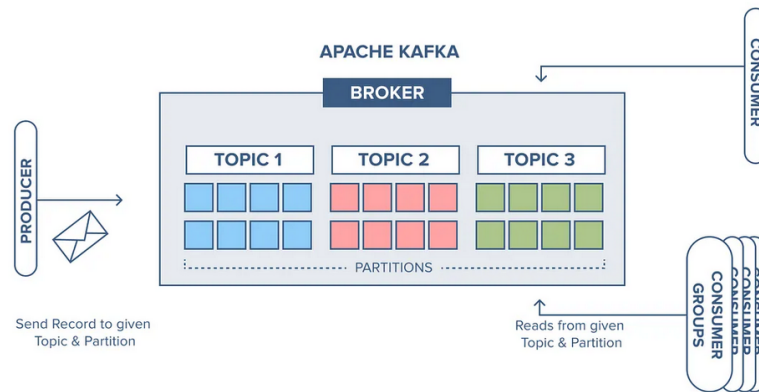


Abbildung 2: Zusammenspiel von Brokern, Topics, Partitionen, Producern und Consumern in Kafka [5]

Producers veröffentlichen Nachrichten, während Consumer Nachrichten aus den Partitionen lesen. Innerhalb einer Partition werden Nachrichten sequenziell abgelegt und mit einem eindeutigen Offset nummeriert, wodurch die Reihenfolge der Nachrichten garantiert ist. Der Offset wird vom Consumer verwaltet, sodass Nachrichten bei Bedarf erneut gelesen werden können. Mehrere Consumer innerhalb einer Consumer-Gruppe können parallel auf unterschiedliche Partitionen zugreifen, wodurch die Verarbeitung großer Datenmengen skaliert wird.

Insgesamt lässt sich die Struktur der Kafka-Architektur als eine vertiefte Erweiterung des grundsätzlichen Aufbaus eines Message Queuing Systems verstehen (siehe „2.1.1 Definitionen“). Jedoch ist bei Kafka die Message Queue selbst keine einzelne große Entität, sondern in Topics und Partitionen unterteilt, damit Consumer effizienter die für sie relevanten Nachrichten auslesen können.

Kafka verfolgt eine Peer-to-Peer-Architektur, bei der alle Broker denselben Status aufrechterhalten. Die hohe Verfügbarkeit wird durch Replikation gewährleistet: Jede Partition kann mehrere Replikate besitzen, die auf unterschiedlichen Brokern gespeichert und synchronisiert werden. Es gibt jeweils eine Leader-Replik und mehrere Follower-Replikate. Schreib- und Lesezugriffe erfolgen über den Leader. Fällt ein Broker aus, wird automatisch ein neuer Leader aus den Follower-Replikaten gewählt. Kafka unterstützt unterschiedliche Nachrichtenliefergarantien, darunter *at-least-once*, *at-most-once* sowie *exactly-once* in Zusammenarbeit mit dem Zielsystem.

Kafka ist für eine hohe Durchsatzrate bekannt, die unter anderem dadurch erzielt wird, dass Nachrichten möglichst selten kopiert werden und häufiger aus dem Page Cache gelesen werden. [6]

Historisch war Kafka auf ZooKeeper angewiesen, um Koordination, Leader Election und Metadatenmanagement zu übernehmen. Ab Version 4.0 nutzt Kafka jedoch den sogenannten KRaft-Modus, der ZooKeeper vollständig ersetzt und die Metadaten intern verwaltet. [7]

2.2.2 Redpanda

Redpanda ist ebenfalls eine verteilte Streaming-Datenplattform, die als Queue genutzt werden kann.

Redpanda ist Kafka-API-kompatibel und wurde als Alternative für Kafka konzipiert. Allerdings besitzt Redpanda eine deutlich einfachere, ressourceneffizientere Architektur. [8]

Kernprinzip ist ein Log-System, in dem Ereignisse in Topics organisiert und in Partitionen verteilt gespeichert werden. Producer schreiben asynchron in diese Topics, Consumer lesen sie unabhängig voneinander aus. Redpanda garantiert, dass Daten langlebig gespeichert werden und die Reihenfolge innerhalb jeder Partition eingehalten wird. [9]

Technisch unterscheidet sich Redpanda zu Kafka vor allem durch die Implementierung in C++ mit einem Thread-per-Core-Modell und den Verzicht auf Komponenten wie die Java Virtual Machine (kurz JVM). [10]

Vorteilhaft bei der Nutzung von Redpanda gegenüber Kafka sei die deutlich geringeren Latenz, insbesondere bei hohen Lasten und sogenannten *Tail-Latencies* (z. B. im 99. oder 99,99 Perzentil). Durch die Implementierung in C++ und den Einsatz eines Thread-per-Core-Modells, bei dem ein CPU-Kern für lediglich einen Thread zuständig ist, kann Redpanda die verfügbare Hardware wesentlich effizienter nutzen als Kafka. Hinzu kommt, dass Redpanda im Gegensatz zu Kafka vollständig auf die JVM verzichtet und dadurch den Garbage-Collection-bedingten Overhead reduziert. Dieser führt bei Kafka häufig zu unregelmäßigen Antwortzeiten. In eigenen Leistungsanalysen gibt Redpanda an, in einigen Szenarien eine bis zu zehnfach geringere Latenz als Kafka zu erzielen, wenn beide Systeme unter vergleichbaren Bedingungen betrieben werden. [11]

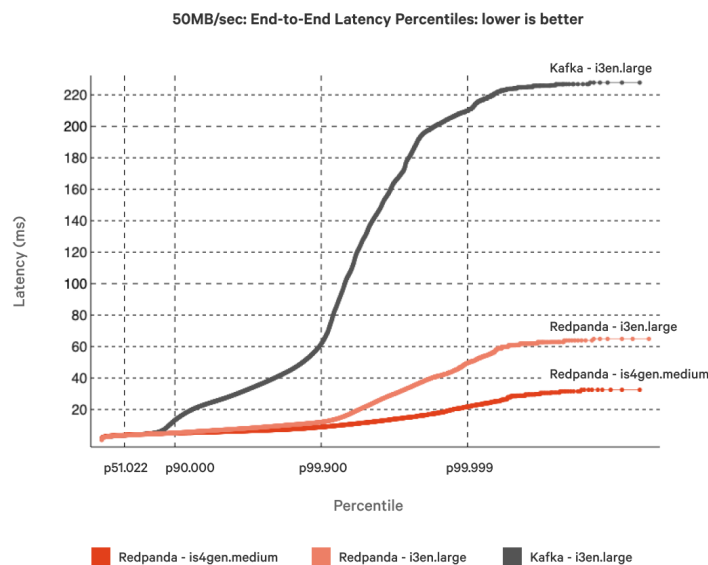


Abbildung 3: Vergleich der auftretenden Latenzen zwischen Redpanda und Kafka [11]

Außerdem benötigte Redpanda von Anfang an keinen externen Koordinator wie ZooKeeper, da alle Funktionen direkt in den Broker integriert sind. Dadurch entfällt die Komplexität, die Kafka in früheren Versionen durch ZooKeeper hatte, wodurch sich der Betrieb vereinfacht.

Auch wirtschaftlich kann Redpanda laut Hersteller Vorteile bieten. Aufgrund der geringeren Systemanforderungen und der vereinfachten Betriebsprozesse erfolge eine Reduzierung der Gesamtbetriebskosten (*Total Cost of Ownership, TCO*) um bis zu Faktor sechs im Vergleich zu Kafka. Diese Einsparungen resultieren sowohl aus einem geringeren Infrastrukturbedarf als auch aus weniger Aufwand für Administration und Monitoring. [12]

Unabhängige Analysen von Jack Vanlightly [13] betonen, dass die von Redpanda veröffentlichten Benchmark-Ergebnisse mit Vorsicht zu interpretieren sind, da sie unter vom Hersteller gewählten Testbedingungen entstanden sind. In seinen Untersuchungen zeigt sich, dass die tatsächliche Leistungsdifferenz stark von der jeweiligen Workload, den Hardwareparametern und der Konfiguration abhängt.

Insgesamt lässt sich festhalten, dass Redpanda in einigen Szenarien Kafka latenz-technisch übertreffen kann. Daher stellt Redpanda eine häufig genutzte Alternative zu klassischen Kafka-Architekturen dar.

Die vereinfachte Konfiguration ist einer der Hauptgründe, weswegen sich das FIM-Team letztendlich für Redpanda entschieden hat.

2.2.3 Spring Boot

Spring Boot ist ein Framework, das die Entwicklung, Konfiguration und den Betrieb von Java-basierten Anwendungen vereinfacht. Es stellt eine umfassende Laufzeitumgebung bereit, die auf dem Spring Framework aufbaut, und ermöglicht durch Auto-Konfiguration, eingebettete Server und einheitliche Projektstrukturen die schnelle Erstellung eigenständiger, produktionsreifer Anwendungen. Darüber hinaus unterstützt Spring Boot die Integration von Testumgebungen, wodurch sich Komponenten und Anwendungskontexte effizient überprüfen lassen. [14]

Das FIM-Team nutzt Spring Boot in einem selbstgeschriebenen Modul namens *Queue-Kit*. Spring Boot dient in diesem Modul nicht als eigenständige Laufzeitumgebung, sondern als technologische Grundlage zur Bereitstellung der Spring-Infrastruktur. Queue-Kit fungiert als wiederverwendbare Bibliothek, die zentrale Strukturen für das Messaging bereitstellt, darunter standardisierte Nachrichtenformate und Event-Handler-Strukturen.

Durch die Integration von Spring Boot wird eine einheitliche und konfigurationsarme Einbindung dieser Komponenten in andere Spring-Boot-basierte Microservices ermöglicht. Die Nutzung von Dependency Injection und Auto-Konfiguration erleichtert dabei die Verwaltung und Erweiterbarkeit der Messaging-Funktionalitäten.

2.2.4 Terraform und OpenTofu

Terraform ist ein Tool für *Infrastructure as Code* (kurz IaC) von HashiCorp, mit dem sich Infrastrukturressourcen deklarativ beschreiben, bereitstellen, vernichten und versionieren lassen. [15]

Das FIM-Team setzt Terraform ein, um zentrale Komponenten des Message Queuing Systems, wie beispielsweise Topics, zu erstellen und zu verwalten. Dadurch können die Komponenten konsistent, reproduzierbar und nachvollziehbar bereitgestellt werden und der Betrieb kann automatisiert stattfinden.

Aufgrund des im August 2023 von HashiCorp angekündigten Lizenzwechsels von Terraform von der *Mozilla Public License 2.0* (kurz MPL 2.0) zur *Business Source License 1.1* (im Folgenden BSL 1.1) ergibt sich für viele Unternehmen eine erhöhte rechtliche Unsicherheit hinsichtlich der zukünftigen Nutzung. [16] [17]

Da die BSL 1.1 keine von der *Open Source Initiative* anerkannte Open-Source-Lizenz ist, birgt sie insbesondere für Anbieter mit kommerziellen Nutzungsszenarien potenzielle Einschränkungen. Aus diesem Grund befindet sich das FIM-Team derzeit im Prozess der Migration auf OpenTofu, eine durch die Linux Foundation verwaltete, vollständig offene Fork von Terraform, die unter einer freien Lizenz weiterentwickelt wird. [18] [19]

OpenTofu bietet dabei Kompatibilität zu bestehenden Terraform-Konfigurationen sowie langfristige Lizenzsicherheit.

2.3 OWASP

Es gibt kein universell anwendbares Maß zur Bewertung der Sicherheit von IT-Systemen, da diese sehr unterschiedlich strukturiert sind und verschiedene sicherheitstechnische Anforderungen erfüllen müssen. Etablierte Ansätze bieten jedoch Orientierung. Ein prominentes Beispiel sind die OWASP Top Ten.

2.3.1 OWASP Foundation

Die *Open Worldwide Application Security Project* (OWASP) Foundation ist eine gemeinnützige Organisation, die sich der Verbesserung der Sicherheit von Software verschrieben hat.

OWASP betreibt eine Vielzahl von community-geführten Open-Source-Projekten, darunter Code, Dokumentation und Standards. Die Organisation veranstaltet Bildungs- und Trainingskonferenzen sowie Workshops, um Entwicklungs- und Sicherheitsexpertise zu verbreiten. Die Mitgliederbasis umfasst zehntausende Einzelpersonen.

Das Ziel von OWASP ist es, Software so zu gestalten, zu betreiben und zu warten, dass sie als vertrauenswürdig gelten kann. Der Leitgedanke spiegelt sich in ihrer Vision „No more insecure software“ wider.

Alle Ressourcen, Projekte und Veranstaltungen sind offen zugänglich und verfolgen Prinzipien wie Transparenz, Innovativität, Unparteilichkeit und Globalität.[20]

2.3.2 OWASP Top Ten

Die *OWASP Top Ten* ist ein weithin anerkanntes, oft als Standard genutztes Dokument zur Sensibilisierung für Webanwendungssicherheit. Sie bietet eine konsensbasierte Übersicht über die aktuell kritischsten Risiken für Webanwendungen.

Anders als eine rein statistische Rangliste wird sie aus einer Kombination von tatsächlichen Daten (etwa aus Schwachstellenreports) und Community-Feedback erstellt.

Ihr Ziel ist es, als Einstiegspunkt für Entwickler und Sicherheitsteams zu dienen. Sie verdeutlicht, welche Kategorien von Schwachstellen besonders häufig auftreten und welche priorisiert behandelt werden sollten.

Platzierung	2017	2021	2025
1	Injection	Broken Access Control	Broken Access Control
2	Broken Authentication	Cryptographic Failures	Security Misconfiguration
3	Sensitive Data Exposure	Injection	Software Supply Chain Failures
4	XML External Entities (XEE)	Insecure Design	Cryptographic Failures
5	Broken Access Control	Security Misconfiguration	Injection
6	Security Misconfiguration	Vulnerable and Outdated Components	Insecure Design
7	Cross-Site Scripting (XSS)	Identification and Authentication Failures	Authentication Failures
8	Insecure Deserialization	Software and Data Integrity Failures	Software and Data Integrity Failures
9	Using Components with Known Vulnerabilities	Security Logging and Monitoring Failures	Logging & Alerting Failures
10	Insufficient Logging & Monitoring	Server-Side Request Forgery (SSRF)	Mishandling of Exceptional Conditions

Tabelle 1: OWASP Top Ten der Jahre 2017, 2021 und 2025 [21] [22]

Die OWASP Top Ten werden etwa alle vier Jahre aktualisiert. Dabei können Kategorien samt Platzierung beibehalten werden oder je nach Veränderung ihrer Relevanz umplatziert werden. Teilweise bekommen Risiken neue Namen, wodurch sie konkretisiert werden. Mit der Zeit entstehen durch technologischen Fortschritt oder häufigeres Vorkommen neue Kategorien. Ältere Risiken, deren Vorkommen beispielsweise durch verbesserte Technologie schwindet, können ihre Platzierung in den OWASP Top Ten verlieren.

Da die OWASP Top Ten 2025 erst zum Ende der Verfassung dieser Arbeit (November 2025) verkündet wurden, bezieht sich diese Arbeit auf die zum Zeitpunkt der Verfassung aktuellste Version aus dem Jahr 2021.

Um die Sicherheit des vom FIM-Team angebotenen Redpanda Message Queuing Systems zu untersuchen und zu bewerten, werden in den folgenden Abschnitten die OWASP Top Ten 2021 entsprechend ihrer Platzierung näher beleuchtet. Dabei werden die vom Team getroffenen oder geplanten Maßnahmen dahingehend eingeordnet, inwiefern sie die Sicherheitsrisiken aus den OWASP Top Ten 2021 adressieren. Gegebenenfalls werden Empfehlungen ausgesprochen, um eine verbesserte Sicherheit zu erreichen.

3 Einordnung sicherheitsrelevanter Entscheidungen auf Basis der OWASP Top Ten 2021

3.1 Broken Access Control

Unter *Access Control* werden alle Maßnahmen zusammengefasst, deren Aufgabe es ist, den Zugriff auf eine Applikation einzuschränken und nur berechtigten Personen oder Anwendungen Zugangsrechte zu erteilen.

Broken Access Control beschreibt Ausfälle und Fehler in der Zugangsbeschränkung oder das komplette Fehlen solcher Maßnahmen. 2021 war Broken Access Control das häufigste Sicherheitsrisiko für Webapplikationen. Dieses Problem sorgt für etliche Gefahren, zum Beispiel können so Unbefugte auf das System zugreifen und Daten manipulieren, löschen oder sensible Daten unerlaubterweise weitergeben. [23] [24]

Besonders in verteilten Systemen mit vielen Microservices, wie sie in Message Queuing Systemen üblich sind, kann bereits ein einzelner fehlerhafter Zugriffspunkt erhebliche Sicherheitsrisiken verursachen.

Da im Kontext der Nutzung des vorliegenden Redpanda Message Queuing Systems alle Nachrichten zentral über die ihnen zugewiesenen Topics versendet werden und die Topics eine gemeinsame Vermittlungsschicht darstellen, besteht die wichtigste Maßnahme zur Vermeidung von Broken Access Control in der Zugriffsbeschränkung auf diese Topics.

Das FIM-Team regelt dies durch den Einsatz von *Access Control Lists*, die über Terraform bereitgestellt werden.

Mithilfe dieser Terraform-Ressourcen kann festgelegt werden, welche Anwendungen auf welche Topics zugreifen dürfen und von welchen Hosts aus der Zugriff möglich ist.

Darüber hinaus lässt sich die Art des Zugriffsrechts definieren. Mögliche Berechtigungen sind beispielsweise Lese- oder Schreibrechte.

Im FIM-Team bestehen Richtlinien darüber, welche Anwendungen welche Arten von Zugriffsrechten erhalten, um den Zugang so exklusiv wie möglich zu halten und das System bestmöglich zu schützen. Auf die genannten Anwendungen wiederum haben nur ausgewählte Mitarbeiter:innen des FIM-Teams Zugriff, um diesen Zugangspunkt ebenfalls zu limitieren.

3.2 Cryptographic Failures

Cryptographic Failures beschreiben Sicherheitsrisiken, die aufgrund unzureichender Verschlüsselung von sensiblen Daten wie zum Beispiel Passwörtern entstehen.

Sind diese mit veralteten Algorithmen verschlüsselt oder fehlt die Verschlüsselung komplett, wird von einem Cryptographic Failure gesprochen. Neben der Kodierung von Daten an sich umfassen Cryptographic Failures auch die (nicht) vorhandene Verschlüsselung des Transportkanals. [25]

Eine etablierte Methode, um den Transportkanal zwischen Client und Server zu verschlüsseln, ist das *Transport-Layer-Security*-Protokoll (kurz TLS). Vereinfacht funktioniert dieses Protokoll wie folgt:

Zunächst sendet der Client eine Nachricht an den Server, in welcher er unter anderem einen Teil für die Berechnung eines geheimen Schlüssels übermittelt. Auf diese Nachricht antwortet der Server mit seinem Berechnungsanteil. Zusätzlich weist sich der Server beim Client mit einem digitalen Zertifikat aus. Solche digitalen Zertifikate werden von sogenannten *Certificate Authorities* (kurz CA) ausgestellt. Anschließend überprüft der Client dieses Zertifikat auf seine Gültigkeit. Ist das Zertifikat gültig, können Client und Server aus den zwei Schlüsselkomponenten einen gemeinsamen geheimen Schlüssel berechnen. Auf diese Weise wird zwischen Client und Server eine geschützte Verbindung aufgebaut. Ab diesem Moment nutzen Client und Server den gemeinsam ausgehandelten Schlüssel für eine sichere Kommunikation. [26]

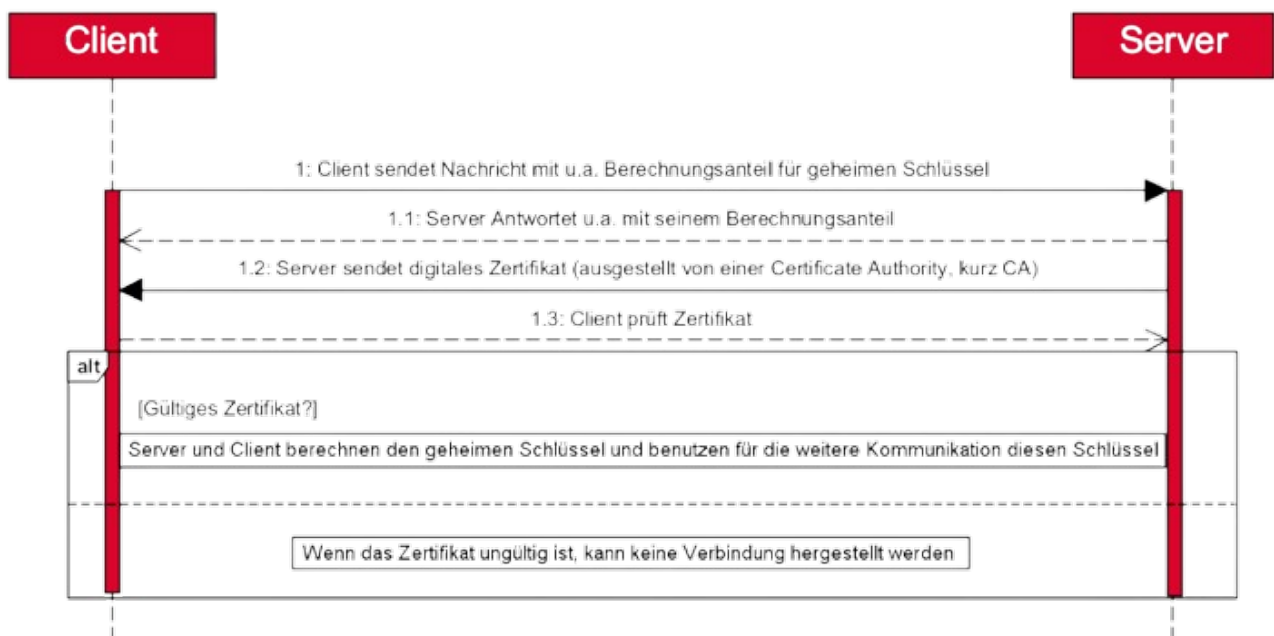


Abbildung 4: Grober Ablauf eines TLS-Handshakes. Erstellt mit Visual Paradigm

Bei der Einrichtung des Terraform-Kafka-Providers für das Message Queuing System fällt auf, dass das FIM-Team eine solche TLS-Verschlüsselung nicht nutzt.

Listing 1: Auszug aus der Definierung des Kafka-Providers

```
1 provider "kafka" {
2     ...
3     tls_enabled = false
4     ...
5 }
```

Um die Ursache für den Verzicht auf eine TLS-Verschlüsselung herauszufinden, habe ich meinen zuständigen Kollegen zu dieser Thematik befragt und folgende Rückmeldung erhalten:

Das FIM-Team besäße aktuell keine eigene CA, um digitale Sicherheitszertifikate auszustellen, welche für eine TLS-Verschlüsselung benötigt werden. Eine manuelle Ausstellung solcher Zertifikate wäre nur schwer wartbar und fehleranfällig. Beispielweise müssten bei einer manuellen Zertifikatsverteilung jedes Mal händisch neue digitale Zertifikate ausgestellt werden, sobald ihre Gültigkeit abläuft, was eine unzumutbare Arbeit darstelle.

Eine öffentliche CA könne ebenfalls nicht genutzt werden, da das Message Queuing System nur über das firmeninterne Netz zu erreichen sei und Externe keinen Zugriff auf das System hätten. Aufgrund dessen, dass das Kommunikationssystem der Message Queue in einer von äußeren Netzwerken aus nicht erreichbaren Zone läuft, ist die aktuelle Lösung für dieses Problem strenge Firewall-Richtlinien und die Übermittlung von verschlüsselten Passwörtern (siehe „3.7 Identification and Authentication Failures“).

Langfristig möchte das FIM-Team für sein Message Queuing System Kubernetes nutzen, ein Tool, mit dem containerisierte Applikationen verwaltet werden können. [27]

Kubernetes kann unter anderem genutzt werden, um digitale Zertifikate und TLS-Verschlüsselungen zu verwalten. [28] Daher bietet es sich an, beim Umstieg auf Kubernetes die Transportweg-Verschlüsselung zukünftig auch über dieses Tool zu lösen.

3.3 Injection

*Inject*ions bezeichnen Sicherheitsrisiken, die entstehen, wenn nicht validierte Eingaben von Nutzer:innen ausgewertet werden. Eine bekannte Unterkategorie sind *SQL Inject*ions: Gibt ein:e Nutzer:in Daten ein, die Teile von SQL-Befehlen enthalten, und werden diese Eingaben ohne Überprüfung verarbeitet, kann diese:r unbefugt Datenbankeinträge abrufen, verändern oder löschen.

Aus diesen Gründen stellen *Inject*ions eine ernstzunehmende Bedrohung dar. [29]

Im Message Queuing System des FIM-Teams wird kein direkter Input von Nutzer:innen erfasst. Diese können lediglich durch Interaktion mit dem FIM-Portal Events initiieren, beispielsweise durch Registrierung.

Diese Events müssen anschließend in ein einheitliches Nachrichtenformat überführt werden, bevor sie auf der Queue abgelegt werden. Dieses Nachrichtenformat sieht wie folgt aus:

Listing 2: Aufbau einer Nachricht

```
1 {
2     "eventId": "UUIDv4",
3     "eventType": "string",
4     "eventTypeVersion": "int",
5     "triggerSource": "string",
6     "timestamp": "ISO8601",
7     "body": {},
8     "source": "string",
9     "metadata": {
10         "traceId": "UUIDv4",
11         "version": "string"
12     }
13 }
```

Auf diese Weise wird Code Injections durch Nutzer:innen vorgebeugt.

Des Weiteren können durch die Nutzung eines einheitlichen Nachrichtenformats einige der in „2.1.3 Abwägung zur Nutzung von Message Queues“ genannten Nachteile abgemildert werden: So lassen sich beispielsweise Poison Messages vermeiden, und mithilfe eines Timestamps in der Nachricht kann eine korrekte Abarbeitungsreihenfolge der Nachrichten sichergestellt werden.

3.4 Insecure Design

Insecure Design ist eine im Vergleich zu anderen OWASP-Platzierungen sehr breit gefächerte Kategorie. Mit ihr werden alle sicherheitstechnisch ungünstigen Designentscheidungen bezeichnet, die bereits in der Planungsphase einer Software ein unsicheres Konzept ergeben. Dies führt dazu, dass selbst bei korrekter Implementierung des Konzepts Sicherheitslücken im System nicht zu vermeiden sind. Ein Beispiel hierfür ist die fehlende Validierung der Eingaben von Nutzer:innen. [30]

CANCOM Managed Services GmbH verfolgt unter anderem den nachfolgenden Ansatz, um sicheres Software-Design zu gewährleisten:

„Sicherheit [muss] bereits Bestandteil der Planungsphase sein. Idealerweise sollte [...] die Software so entworfen werden, dass Sicherheitsaspekte integrale Bestandteile der System- und Softwarearchitektur darstellen.“ [31]

Diese Richtlinie orientiert sich an der *Secure-by-Design*-Philosophie, bei der Sicherheitsmechanismen von Anfang an berücksichtigt werden, im Gegensatz zur nachträglichen Integration am Ende der Implementierung.

Um überprüfen zu können, inwiefern das Message Queuing System diese Richtlinie befolgt und potenzielle Sicherheitsrisiken im Design zu erkennen, muss das Design zunächst systematisch untersucht und aufbereitet werden.

Das Message Queuing System des FIM-Teams besteht im Kern aus drei Topics:

1. **userEvents:** userEvents ist für die Speicherung und Verarbeitung von Events verantwortlich, die Nutzer:innen betreffen. Beispiele hierfür sind Registrierung und Login.
2. **processEvents:** Das Topic processEvents wird für interne Prozesskoordination und Statusüberprüfung genutzt.
3. **deadLetterQueue:** Können Nachrichten aufgrund von Systemfehlern nicht verarbeitet werden, landen sie auf der deadLetterQueue, um entsprechend behandelt zu werden.

Diese Struktur sorgt dafür, dass fehlerhafte und damit auch potenziell sicherheitstechnisch riskante Nachrichten von den restlichen Nachrichten sofort isoliert werden. Dadurch können zum einen diese Nachrichten separat behandelt werden, zum anderen stören sie nicht den Workflow. Zudem bekämpft diese Struktur in „2.1.3 Abwägung zur Nutzung von Message Queues“ behandelte Nachteile.

Die Nachrichten auf den Topics werden von sogenannten *Workern* verarbeitet. Worker sind Anwendungen, die ihrem Aufgabenbereich entsprechende Nachrichten aus der Queue auslesen und je nach Nachrichteninhalt Aktionen ausführen können. Falls anschließend Aufgaben anfallen, die von anderen Workern erfüllt werden müssen, kann der Worker diese Events als Nachricht verpackt auf der Queue ablegen, damit die zuständigen Worker diese Neben-Events bearbeiten können. Dies bedeutet, dass je nach Szenario Worker sowohl Producer als auch Consumer sein können.

Auffällig an der Architektur ist, dass `userEvents` sehr viele Nachrichten umfasst, da jedes durch Nutzer:innen ausgelöste Event auf diesem Topic abgelegt wird. Dadurch müssen viele Worker auf dieses Topic zugreifen. Dies bedeutet, dass bereits ein korruptierter Worker ein Sicherheitsrisiko für `userEvents` darstellen würde. Diese Designentscheidung könnte unter Umständen auch als Broken Access Control aufgefasst werden. Zwar ist der Befall eines Workers aufgrund der Abschottung des Message Queuing Systems gegenüber externen Netzwerken sehr unwahrscheinlich, jedoch könnte es langfristig sicherer sein, das `userEvents`-Topic in mehrere Partitionen aufzuteilen.

Die designtechnisch sicherheitsrelevantesten Worker sind der *auditWorker* und der *dlqWorker*. Deren Funktionsweise und Aufgabenbereich lässt sich folgendermaßen beschreiben:

Die Aufgabe des `auditWorkers` ist es, alle Nachrichten zu Analyse- und Nachweiszwecken gegenüber Kund:innen für eine gewisse Zeit zu speichern.

Der `dlqWorker` liest Nachrichten aus der `deadLetterQueue` aus, die aufgrund von System- oder Nachrichtenfehlern dort gelandet sind.

Da diese beiden Worker Zugriff auf eine große Anzahl unterschiedlicher Nachrichten haben, müssen sie besonders geschützt werden. Dies wird durch eine grundlegende Abschottung des Systems von externen Netzwerken gewährleistet. Dies wird dadurch realisiert, dass Zugang zum Message Queuing System und damit auch zu den Workern strikt eingeschränkt ist: Nur ausgewählte Mitarbeiter:innen des FIM-Teams können auf diese zugreifen. Hinzu kommt, dass der Zugriff ausschließlich aus dem internen CANCOM-Netz heraus möglich ist. Langfristig wäre es empfehlenswert, besonders die Kommunikation dieser beiden Worker über TLS zu verschlüsseln

Auch Fehlermeldungen können in einem Softwaresystem aufgrund unbedachter Strukturierung eine Angriffsfläche darstellen, wenn sie beispielsweise sensible Daten preisgeben. Daher ist es sinnvoll, das Konzept des FIM-Teams in Zusammenhang mit Fehlerbehandlung näher zu betrachten:

Listing 3: Standardisierte Fehlernachricht im Message Queuing System des FIM-Teams

```
1 {
2   "eventId": "550e8400-e29b-41d4-a716-446655440000",
3   "eventType": "dlqEvent",
4   "eventTypeVersion": "1.0",
5   "triggerSource": "Error",
6   "timestamp": "2024-01-15T10:30:00Z", "source": "queue-kit",
7   "body": {
8     "originalMessage": " {\"eventId\":\"...\", \"eventType\": \"userEvents\", ...} ",
9     "errorMessage": "Failed to process user registration: null pointer exception"
10  },
11  "metadata": {
12    "traceId": "550e8400-e29b-41d4-a716-446655440001",
13    "version": "1.0"
14  }
15 }
```

Fehlernachrichten des FIM-Teams unterscheiden sich von klassischen Fehlermeldungen dadurch, dass sie nicht für die Anzeige bei Nutzer:innen vorgesehen sind. Das standardisierte Nachrichtenformat dient dazu, fehlerhafte oder fehgeschlagene Nachrichten in der `deadLetterQueue` abzulegen, um sie anschließend nach einem vordefinierten Verfahren vom `dlqWorker` weiterverarbeiten zu lassen. Durch diese Standardisierung und die Abschottung gegenüber Nutzer:innen stellen die Fehlernachrichten im Message-Queue-System keine Sicherheitsbedrohung dar.

Mithilfe dieser Designentscheidungen beugt das FIM-Team bereits automatisch vielen Gefahren von externer Seite vor. Mit einer möglichen Partitionierung von `userEvents` und der Einführung von TLS konnte ein Verbesserungspotenzial ermittelt werden.

3.5 Security Misconfiguration

Security Misconfiguration beinhaltet Sicherheitslücken, die durch fehlende oder unzureichende Konfiguration entstehen. Klassische Beispiele wären das Beibehalten von Standardpasswörtern oder das Offenhalten von ungenutzten Ports. [32]

Die in „3.2 Cryptographic Failures“ angesprochene fehlende TLS-Verschlüsselung stellt neben einem Cryptographic Failure auch eine Security Misconfiguration dar. Diese fehlende Verschlüsselung des Transportwegs bildet eine Schnittstelle zwischen zwei Kategorien der OWASP Top Ten. Dies zeigt auf, dass die Platzierungen ineinander übergehen und sich gegenseitig ergänzen können.

Eine vom FIM-Team konfigurierte Sicherheitsmaßnahme ist das SASL-Framework. Dieses Framework beugt Gefahren der OWASP-Kategorie Identification and Authentication Failures vor. Daher folgen Erläuterungen zu diesem Thema im gleichnamigen Kapitel.

Innerhalb des Message Queuing Systems nutzt das FIM-Team diverse Ports. Im Folgenden gilt es, deren Nutzen näher zu beleuchten und ihre Notwendigkeit zu bewerten:

Der Port 9092 wird verwendet, um die Kommunikation des Systems mit den Kafka-Endpunkten zu ermöglichen.

Für die Interaktion mit dem System über einen HTTP-Proxy kommt Port 8082 zum Einsatz. Damit sich mehrere Broker intern austauschen können, wurde ein Kommunikationskanal über Port 33145 eingerichtet. Da derzeit jedoch nur ein Broker im Einsatz ist, wird dieser Port momentan nicht aktiv genutzt. Für die zukünftige Skalierung ist geplant, mehrere Broker bereitzustellen, um Ausfällen vorzubeugen. Daher ist es sinnvoll, diesen Port bereits jetzt offen zu halten.

Für den Produktivbetrieb wurde Port 9644 konfiguriert, um Security-Monitoring einzurichten und eine Verbindung zu Prometheus herzustellen (siehe „3.9 Security Logging and Monitoring Failures“).

Redpanda bietet seinen Nutzer:innen die Möglichkeit, eine sogenannte *Schema Registry* zu verwenden. Die Verbindung zu dieser Registry ist derzeit über Port 8081 konfiguriert. Die Schema Registry dient der Überprüfung und Validierung von Nachrichten anhand definierter Schemata. Nachrichten, die nicht dem vorgegebenen Schema entsprechen, werden dabei nicht weiterverarbeitet.

Trotz der Konfiguration dieses Ports nutzt das FIM-Team die Schema Registry nicht, da sie nicht kostenfrei verfügbar ist. Stattdessen kommt ein selbstgeschriebenes Nachrichtenformat zur Validierung zum Einsatz (siehe „3.3 Injection“). Aus diesem Grund kann dieser Port entfernt werden, um das potenzielle Angriffsrisiko zu verringern.

Eine weitere Konfigurations-Instanz im Message Queuing System ist die Verwaltung von Benutzer-Credentials, mit denen auf das System zugegriffen werden kann. Derzeit müssen die entsprechenden Zugangsdaten in Terraform-Dateien hinterlegt werden.

Eine sicherere Alternative wäre, die Credentials als GitLab CI/CD-Variablen zu speichern. Dies ist momentan jedoch nicht möglich, da der verwendete GitLab-Runner keinen Zugriff auf Redpanda hat.

Nach der geplanten Migration des Systems auf Kubernetes kann dieses Problem voraussichtlich behoben werden, indem Kubernetes die Verwaltung geheimer Credentials übernimmt.

Insgesamt sind die bestehenden Konfigurationen robust, auch wenn in einigen Bereichen noch Optimierungspotenzial besteht.

3.6 Vulnerable and Outdated Components

Vulnerable and Outdated Components bezeichnen die Verwendung veralteter Software-Komponenten und die damit verbundenen Risiken.

Ältere Software-Versionen enthalten häufig bekannte Sicherheitslücken und sollten daher regelmäßig aktualisiert werden.

Es liegt in der Verantwortung der Nutzer:innen von externen Komponenten, auf aktuelle Versionen umzusteigen, um sich vor möglichen Cyberangriffen zu schützen, die durch Schwachstellen in älteren Versionen ausgenutzt werden können. [33]

Zur Veranschaulichung der Häufigkeit dieses Problems: In einer Studie wurden stichprobenartig über 133.000 Websites untersucht, die externe JavaScript-Bibliotheken verwendeten. Das Ergebnis: 37% dieser Websites nutzten Bibliotheken, die so veraltet waren, dass mindestens eine bekannte Sicherheitslücke bestand. [34]

Dies zeigt deutlich, dass die Nutzung veralteter Komponenten ein weit verbreitetes und ernstzunehmendes Problem darstellt.

Das FIM-Team nutzt diverse externe Ressourcen für die Umsetzung des Message Queuing Systems. Diese wurden in „2.2 Eingesetzte Technologien“ näher beleuchtet. Nun gilt es, die genutzten Versionen auf Aktualität zu überprüfen:

Die Untersuchung startet mit der Queue-Technologie Redpanda. Um die Strategie nachzuvollziehen, durch welche die Nutzung der aktuellsten stabilen Redpanda-Version gewährleistet wird, habe ich den zuständigen Arbeitskollegen im FIM-Team befragt und folgende Auskunft erhalten:

Mithilfe einer sogenannten *Puppet-Class*, einem Konfigurations-Tool, werde sichergestellt, dass bei Installation und Einrichtung von Redpanda stets die aktuell stabile Version verwendet wird. Dieser Ansatz sorgt für ein hohes Maß an Konsistenz und wirkt potenziellen Risiken durch veraltete Versionen entgegen.

Die *Queue-Kit-Library* (siehe „2.2.3 Spring Boot“) nutzt Spring Boot Version 3.5.5, welche zum Zeitpunkt der Verfassung dieses Abschnitts (29.10.2025) eine ziemlich aktuelle Version ist. Allerdings gibt es mit Version 3.5.7. eine bereits aktuellere Version. [14]

Version 3.5.7 beinhaltet unter anderem Aktualisierungen verschiedener Abhängigkeiten, deren sicherheitsrelevante Bedeutung jedoch nicht in jedem Fall eindeutig nachvollziehbar ist. [35]

Dennoch ist es für das FIM-Team ratsam, auf die neuere Spring-Boot-Version umzusteigen, um potenziellen Risiken vorzubeugen und von Verbesserungen der Plattform zu profitieren.

Darüber hinaus setzt das FIM-Team derzeit die Version 0.9.0 des Terraform-Kafka-Providers ein. Zum Zeitpunkt der Erstellung dieses Abschnitts (29.10.2025) liegt jedoch bereits Version 0.13.1 vor. [36] Es wird daher empfohlen, zeitnah auf eine aktuellere, von OpenTofu unterstützte Version zu aktualisieren, um langfristige Kompatibilität und Sicherheit zu gewährleisten.

3.7 Identification and Authentication Failures

Identification and Authentication Failures beinhalten alle Sicherheitslücken, die dafür sorgen, dass sich Unberechtigte als eine andere Instanz mit anderen Zugangsrechten als sie selbst ausgeben können, um Zugang zu ein System zu bekommen.

Können Angreifer:innen aufgrund von schwachen Passwortrichtlinien über Brute-Force-Attacken die Zugangsdaten von Nutzer:innen erraten, wäre dies ein klassisches Beispiel für ein Identification and Authentication Failure. [37]

Da Nutzer:innen des FIM-Portals nicht direkt auf die Queue zugreifen können, gilt es, eine Ebene tiefer den Zugang der Worker auf die Queue zu analysieren.

Jeder Worker besitzt einen sogenannten User, mit dem er sich bei der Queue anmelden muss, um auf diese zuzugreifen. Der Login erfolgt über das *Simple-Authentication-and Security-Layer-Framework* (kurz SASL).

SASL ist ein Framework, das die klare Identifizierung von Clients über einen verschlüsselten Mechanismus ermöglicht.[38]

Der für die Verschlüsselung konkret eingesetzte Mechanismus ist SCRAM-SHA-256. [39] Über diesen Algorithmus werden die Passwörter, mit denen sich Worker anmelden, verschlüsselt übertragen.

Auf diese Weise wird dafür gesorgt, dass jeder Worker eindeutig identifiziert werden kann und keine Gefahr besteht, dass sich Unbefugte über ein Identification and Authentication Failure Zugriff verschaffen können.

3.8 Software and Data Integrity Failures

Als *Software and Data Integrity Failures* werden Schwachstellen bezeichnet, die durch fehlende Überprüfung der Authentizität von verarbeiteten Daten oder genutzten externen Komponenten auftreten.

Können Daten unbemerkt verändert werden oder werden Bibliotheken aus unsicheren Quellen benutzt, bietet dies die Möglichkeit, Schadsoftware in ein System einzuschleusen. [40]

Da die verwendeten Technologien (siehe „2.2.3 Eingesetzte Technologien“) entweder aus verlässlichen und regelmäßig geprüften Quellen stammen oder als eigene Bibliotheken entwickelt wurden, entsteht in diesem Bereich kein relevantes Sicherheitsrisiko.

Die Datenintegrität wird gewährleistet, indem Zugriffsrechte auf das Message Queuing System nur sehr restriktiv vergeben werden und die Datenverarbeitung nach klar definierten Schemata erfolgt, welche in der Queue-Kit-Library definiert sind. In diesem Zusammenhang enthält das standardisierte Nachrichtenformat des FIM-Teams die Attribute *triggerSource* und *source*. Diese beinhalten den Grund für die Erstellung der Nachricht und von welcher Anwendung sie stammt. Dadurch kann im Zweifelsfall der Ursprung einer Nachricht nachvollzogen werden.

Aus diesen Gründen weist das Message Queuing System kaum eine Angriffsfläche für Sicherheitsrisiken im Bereich Software and Data Integrity Failures auf.

3.9 Security Logging and Monitoring Failures

Unter *Security Logging and Monitoring Failures* versteht man die Probleme, die entstehen, wenn sicherheitsrelevante Ereignisse, wie zum Beispiel fehlgeschlagene Login-Versuche, gar nicht oder nur unzureichend protokolliert werden. [41]

Dadurch bleiben potenzielle Gefahren und Angriffsmuster unentdeckt, sodass keine geeigneten Gegenmaßnahmen eingeleitet werden können.

Um diesem Problem entgegenzuwirken, ist es wichtig, relevante Ereignisse in Form von Logs festzuhalten. Das kann durch verschiedene Tools ermöglicht werden. Eine Toolchain, die dem FIM-Team bereits bekannt ist und somit auch für die Überwachung des Message Queuing Systems in Frage kommt, ist die Kombination aus Loki und Grafana.

Loki ist ein von Grafana Labs entwickeltes Open-Source-System zur zentralen Speicherung und Analyse von Log-Daten. Im Gegensatz zu klassischen Log-Management-Systemen indiziert Loki nicht den gesamten Log-Inhalt, sondern ausschließlich Metadaten wie Label oder Zeitstempel. Dadurch kann es große Mengen von Logs effizient und ressourcenschonend verarbeiten und speichern. [42]

Loki wurde konzipiert, um eng mit Prometheus zusammenzuarbeiten, einem verbreiteten Monitoring-System für Metriken.

Diese enge Integration ermöglicht es, Metriken und Logs gemeinsam auszuwerten, wodurch sich Fehlerursachen und Systemzustände präziser nachvollziehen lassen.

Grafana ist eine ebenfalls von Grafana Labs entwickelte Open-Source-Plattform zur Visualisierung und Analyse von Daten aus verschiedenen Quellen. [43] Sie bietet die Möglichkeit, Dashboards zu erstellen, die Metriken, Logs und Traces übersichtlich darstellen. Durch die Integration unterschiedlicher Datenquellen, wie beispielsweise Loki oder Prometheus, ermöglicht Grafana eine einheitliche Sicht auf die Systemlandschaft. Darüber hinaus können Alerts definiert werden, die bei bestimmten Schwellwerten oder Log-Mustern Benachrichtigungen auslösen, um Probleme oder Risiken frühzeitig zu erkennen.

Die Kombination von Grafana und Loki bietet eine leistungsfähige Lösung für Log-Monitoring und Fehleranalyse. Während Loki die effiziente Erfassung, Speicherung und Abfrage von Logs übernimmt, stellt Grafana die visuelle Aufbereitung und interaktive Analyse dieser Daten bereit. Dadurch können Administrator:innen und Entwickler:innen sowohl vergangene als auch aktuelle Ereignisse zentral überwachen. Besonders vorteilhaft ist die Möglichkeit, Logs und Metriken in einem gemeinsamen Dashboard in einen Zusammenhang zu setzen. Beispielsweise kann bei einem Anstieg der CPU-Auslastung direkt nachvollzogen werden, welche Log-Ereignisse zeitgleich aufgetreten sind. Dies verbessert nicht nur die Transparenz komplexer Systeme, sondern beschleunigt auch die Fehlersuche und reduziert Ausfallzeiten.

Insgesamt bietet diese Toolchain eine solide Struktur, um riskantes Verhalten schnell zu erkennen und entsprechend zu handeln und kann dem FIM-Team zur Nutzung empfohlen werden.

3.10 Server-Side Request Forgery

Server-Side Request Forgery beschreibt einen Vorfall, in dem ein:e Nutzer:in unautorisiert den Server dazu bringt, an interne oder externe Ziele Anfragen zu schicken.

Auf diese Weise kann ein:e Nutzer:in beispielsweise unbefugt Anfragen an interne URLs senden, die Daten enthalten, die ausschließlich für Administrator:innen bestimmt sind. [44]

Da im Message Queuing System Nutzer:innen nie direkt mit der Queue kommunizieren, ist es für sie design-technisch nicht möglich, über den Server unzulässige Ziele anzusteuern.

Daraus resultierend stellt Server-Side Request Forgery keine weitere Gefahr für das Message Queuing System dar.

3.11 Ergebnisse und Diskussion

Bereits in der aktuellen Phase der Umstrukturierung lässt sich feststellen, dass das FIM-Team viele Maßnahmen getroffen hat, um seinen Kund:innen ein sicheres FIM-Portal mit Message-Queue-Technologie anzubieten.

Der Zugang wird architekturbedingt stark eingeschränkt, indem Nutzer:innen gar keinen direkten Zugriff auf die Queue haben und ausgewählte Mitarbeiter:innen des FIM-Teams eingeschränkten Zugriff über das interne CANCOM-Netz haben.

Des Weiteren werden die Passwörter, mit denen sich die Worker im Message Queue System anmelden, nur verschlüsselt übermittelt und nicht im Klartext abgespeichert.

Die Worker sind ebenfalls in ihrem Handeln eingeschränkt: Über Access Control Lists wird den Workern ganz klar zugewiesen, auf welche Topics sie welche Art des Zugriffs (zum Beispiel Lese- oder Schreibzugriff) haben.

Zusätzlich sorgt ein einheitliches Nachrichtenformat und vordefiniertes Bearbeitungskonzept dafür, dass kein Unerlaubter Inhalt ins System gelangen kann.

Außerdem achtet das FIM-Team darauf, dass externe Software-Komponenten aus verlässlichen, regelmäßig überprüften Quellen stammen und dahingehend das System keinem Risiko durch Nutzung unzuverlässiger Software ausgesetzt.

Für den zukünftigen Produktivbetrieb ist es ratsam, die Queue-Kommunikation mit TLS-Verschlüsselung zu versehen. Diese lässt sich bei der Migration des Systems auf Kubernetes einrichten.

Weiterhin kann darüber nachgedacht werden, inwiefern eine Unterteilung des userEvents-Topics umsetzbar wäre.

Zudem kann die Verbindung zur Schema Registry über Port 8081 getrennt werden, da diese nicht genutzt wird.

Des Weiteren sollte darauf geachtet werden, bei genutzten externen Komponenten wenn möglich auf aktuellere Versionen umzusteigen. Zwar sind die derzeit genutzten Versionen nicht stark veraltet, allerdings kann der Umstieg auf noch aktuellere Versionen sicherheitstechnische Vorteile bieten.

Zur Überwachung sicherheitsrelevanter Ereignisse im Produktivbetrieb bietet sich die Nutzung von Loki in Kombination mit Grafana an, um auffälliges Verhalten sowohl in Form von Logs als auch grafisch festhalten und nachvollziehen zu können.

Insgesamt verfolgt das FIM-Team ein durchdachtes Sicherheitskonzept, welches viele der in den OWASP Top Ten 2021 beschriebenen Risiken adressiert und durch die ausgesprochenen Empfehlungen optimiert werden kann.

4 Ausblick

Durch die in dieser Arbeit formulierten Empfehlungen kann das Sicherheitskonzept des FIM-Teams gezielt erweitert und weiter gestärkt werden, um gegenüber Sicherheitsbedrohungen noch widerstandsfähiger zu sein.

Diese Arbeit bietet dem FIM-Team eine zusammengefasste Übersicht über die bereits umgesetzten Sicherheitsmaßnahmen und welche, die zukünftig eingesetzt werden können. Diese Maßnahmen wurden im Rahmen dieser Ausarbeitung konsequent anhand eines bestehenden Standards, den OWASP Top Ten 2021, geprüft und bewertet.

Das FIM-Team kann dieses Dokument künftig bei Sicherheitsfragen oder während der Planung neuer Sicherheitsmaßnahmen zu Rate ziehen. Dadurch lassen sich geplante Maßnahmen in Bezug auf das bestehende Sicherheitskonzept besser einordnen.

Außerdem wird nachvollziehbar, welche konkreten Sicherheitsrisiken durch die jeweiligen Maßnahmen adressiert werden können.

Da Software-Entwicklung eine sehr dynamische Disziplin darstellt, können sich die damit verbundenen Sicherheitsrisiken im Laufe der Zeit verändern.

Während einige Gefahren lange Zeit bestehen bleiben, kann sich ihre Häufigkeit und damit ihre Relevanz mit der Zeit verändern. Mit fortschreitender Technologie können sogar ganz neue Risiken entstehen.

Gegen Ende der Verfassung dieser Arbeit (November 2025) wurden die OWASP Top Ten 2025 veröffentlicht. [22]

Viele Sicherheitsrisiken der OWASP Top Ten 2021 bleiben mit teilweise anderen Platzierungen als zuvor und Namensänderungen in den OWASP Top Ten 2025 bestehen. Beispielsweise ist Security Misconfiguration von Platz fünf auf Platz zwei aufgestiegen.

Zwei Kategorien sind 2025 neu hinzugekommen:

Software Supply Chain Failure auf Platz drei stellt eine Erweiterung von Vulnerable and Outdated Components dar und umfasst ein breiteres Spektrum an genutzten unsicheren Abhängigkeiten innerhalb und außerhalb eines Softwaresystems.

Unter dem Punkt *Mishandling of Exceptional Conditions*, der auf Platz zehn liegt, werden Unsicherheiten thematisiert, die aus einer fehlenden oder unzureichenden Behandlung ungewöhnlicher Randfälle in Softwaresystemen resultieren.

Abgesehen von den OWASP Top Ten ergeben sich aktuell durch die Weiterentwicklung und Popularität von *Large Language Models* (kurz LLMs) wie zum Beispiel ChatGPT einige neue Chancen und Gefahren.

Im Bereich *Automatic Exploit Generation* bieten LLMs beispielsweise die Möglichkeit, Schwachstellen in Softwaresystemen automatisiert und schneller zu erkennen und so gegen diese vorzugehen. [45] Gleichzeitig kann dies auch von Angreifern genutzt werden, um Sicherheitslücken in fremden Systemen ausfindig zu machen und gezielt auszunutzen.

Abschließend zeigt sich, dass eine sichere Software-Entwicklung nur durch kontinuierliche Sicherheitsanalysen gewährleistet werden kann, die regelmäßig auf die Erkennung und Vermeidung bereits bekannter und neuer Gefahren ausgelegt werden müssen.

Diese Arbeit hat hierzu einen Beitrag geleistet, indem bestehende Sicherheitsmaßnahmen untersucht und Empfehlungen zur weiteren Verbesserung ausgearbeitet wurden.

Literaturverzeichnis

- [1] S. Bouchenak und N. de Palma, „Message Queuing Systems“, in *Encyclopedia of Database Systems*, L. Liu und M. T. Özsu, Hrsg. New York, NY: Springer New York, 2018, S. 2232–2233, ISBN: 978-1-4614-8265-9. DOI: 10.1007/978-1-4614-8265-9_1548. Adresse: https://doi.org/10.1007/978-1-4614-8265-9_1548.
- [2] R. Kotha, „The Impact of Messaging Queues on Real-Time Data Exchange in Large-Scale applications“, *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH AND CREATIVE TECHNOLOGY*, Jg. 7, Nr. 3, Mai 2021. DOI: 10.5281/zenodo.14227322. Adresse: <https://doi.org/10.5281/zenodo.14227322>.
- [3] IBM, *Main features and benefits of message queuing*. Adresse: <https://www.ibm.com/docs/en/ibm-mq/9.4.x?topic=queuing-main-features-benefits-message> (besucht am 29.10.2025).
- [4] A. Vinci, „Common Problems in Message Queues [With Solutions]“, Medium. (Jan. 2023), Adresse: <https://medium.com/@vinciabhnav7/common-problems-in-message-queues-with-solutions-f0703c0bd5af> (besucht am 29.10.2025).
- [5] K. Rawal, „Kafka Broker, Kafka Topic, Consumer and Record Flow in Kafka“, Medium. (Sep. 2020), Adresse: <https://kajalrawal.medium.com/kafka-broker-kafka-topic-consumer-and-record-flow-in-kafka-ec55104977b8> (besucht am 29.10.2025).
- [6] G. Fu, Y. Zhang und G. Yu, „A Fair Comparison of Message Queuing Systems“, *IEEE Access*, Jg. 9, S. 421–432, 2020, ISSN: 2169-3536. DOI: <https://doi.org/10.1109/access.2020.3046503>.
- [7] S. Hämmerle, „Goodbye ZooKeeper: Kafka schlägt neue Wege ein“, IONOS. (Feb. 2025), Adresse: <https://ionos.blog/goodbye-zookeeper-kafka-schlaegt-neue-wege-ein/> (besucht am 29.10.2025).
- [8] *High-perf Agentic Data Plane & Streaming / Redpanda*, Redpanda. Adresse: <https://www.redpanda.com/> (besucht am 29.10.2025).
- [9] *How Redpanda Works*, Redpanda. Adresse: <https://docs.redpanda.com/25.1/get-started/architecture/> (besucht am 29.10.2025).
- [10] D. Flora, „When to choose Redpanda instead of Apache Kafka“, Redpanda. (Juni 2023), Adresse: <https://www.redpanda.com/blog/when-to-choose-redpanda-vs-kafka> (besucht am 29.10.2025).
- [11] T. Stevens, „Redpanda vs. Kafka: A performance comparison“, Redpanda. (Okt. 2022), Adresse: <https://www.redpanda.com/blog/redpanda-vs-kafka-performance-benchmark> (besucht am 29.10.2025).
- [12] T. Stevens, „Redpanda vs. Apache Kafka (TCO Analysis)“, Redpanda. (Okt. 2022), Adresse: <https://www.redpanda.com/blog/is-redpanda-better-than-kafka-tco-comparison> (besucht am 29.10.2025).
- [13] J. Vanlightly, „Kafka vs Redpanda Performance - Do the claims add up?“ (Mai 2023), Adresse: <https://jack-vanlightly.com/blog/2023/5/15/kafka-vs-redpanda-performance-do-the-claims-add-up> (besucht am 29.10.2025).
- [14] *Spring Boot*, Spring. Adresse: <https://docs.spring.io/spring-boot/index.html> (besucht am 29.10.2025).
- [15] G. Lindemulder und M. Kosinski, „Was ist Terraform?“, IBM. (), Adresse: <https://www.ibm.com/de-de/think/topics/terraform> (besucht am 29.10.2025).

- [16] „HashiCorp adopts the Business Source License for future releases of its products“, HashiCorp. (Aug. 2023), Adresse: <https://www.globenewswire.com/news-release/2023/08/10/2723189/0/en/HashiCorp-adopts-the-Business-Source-License.html> (besucht am 29.10.2025).
- [17] „Terraform license change explained“, Spacelift. (Aug. 2024), Adresse: <https://spacelift.io/blog/terraform-license-change> (besucht am 29.10.2025).
- [18] „Linux Foundation Launches OpenTofu: A New Open Source Alternative to Terraform“, The Linux Foundation. (Sep. 2023), Adresse: <https://www.linuxfoundation.org/press/announcing-opentofu> (besucht am 29.10.2025).
- [19] *Open-Source Infrastructure as Code*, OpenTofu. Adresse: <https://opentofu.org> (besucht am 29.10.2025).
- [20] *About the OWASP Foundation*, OWASP. Adresse: <https://owasp.org/about/> (besucht am 29.10.2025).
- [21] *Welcome to the OWASP Top 10 - 2021*, OWASP. Adresse: https://owasp.org/Top10/A00_2021_Introduction/ (besucht am 17.11.2025).
- [22] *Introducing the OWASP Top Ten:2025*, OWASP. Adresse: https://owasp.org/Top10/2025/0x00_2025-Introduction/ (besucht am 07.11.2025).
- [23] *A01:2021 - Broken Access Control*, OWASP. Adresse: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (besucht am 29.10.2025).
- [24] E.-T. EL Ghachi und B. Raouyane, „Web Application Security: A Review of Broken Access Control in Web Applications“, in *Advances on Intelligent Computing and Data Science II*, F. Saeed, F. Mohammed, E. Mohammed, S. Basurra und M. Al-Sarem, Hrsg., Cham: Springer Nature Switzerland, 2025, S. 25–34, ISBN: 978-3-031-91351-8. DOI: 10.1007/978-3-031-91351-8. Adresse: <https://doi.org/10.1007/978-3-031-91351-8>.
- [25] *A02:2021 - Cryptographic Failures*, OWASP. Adresse: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (besucht am 29.10.2025).
- [26] A. Jonker und T. Krantz. „What Is Transport Layer Security (TLS)?“, IBM. (), Adresse: <https://www.ibm.com/think/topics/transport-layer-security> (besucht am 30.10.2025).
- [27] *Kubernetes*, Kubernetes. Adresse: <https://kubernetes.io/> (besucht am 30.10.2025).
- [28] *Manage TLS Certificates in a Cluster*, Kubernetes. Adresse: <https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster/> (besucht am 30.10.2025).
- [29] *A03:2021 - Injection*, OWASP. Adresse: https://owasp.org/Top10/A03_2021-Injection/ (besucht am 29.10.2025).
- [30] *A04:2021 - Insecure Design*, OWASP. Adresse: https://owasp.org/Top10/A04_2021-Insecure_Design/ (besucht am 29.10.2025).
- [31] *Richtlinie zum sicheren Softwareentwicklungslebenszyklus (SDLC) CANCOM Managed Services GmbH*, firmeninternes PDF, nicht öffentlich verfügbar, CANCOM Managed Services GmbH.
- [32] *A05:2021 - Security Misconfiguration*, OWASP. Adresse: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/ (besucht am 29.10.2025).
- [33] *A06:2021 - Vulnerable and Outdated Components*, OWASP. Adresse: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/ (besucht am 29.10.2025).

- [34] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson und E. Kirda, „Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web“, in *Proceedings 2017 Network and Distributed System Security Symposium*, Ser. NDSS 2017, Internet Society, 2017. DOI: 10.14722/ndss.2017.23414. Adresse: <http://dx.doi.org/10.14722/ndss.2017.23414>.
- [35] *spring-projects / spring-boot*, Spring Boot. Adresse: <https://github.com/spring-projects/spring-boot/releases> (besucht am 29.10.2025).
- [36] *kafka*, HashiCorp. Adresse: <https://registry.terraform.io/providers/Mongey/kafka/0.9.0/docs/resources/topic> (besucht am 29.10.2025).
- [37] *A07:2021 - Identification and Authentication Failures*, OWASP. Adresse: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ (besucht am 29.10.2025).
- [38] K. Zeilenga und A. Melnikov, *Simple Authentication and Security Layer (SASL)*, RFC 4422, Juni 2006. DOI: 10.17487/RFC4422. Adresse: <https://www.rfc-editor.org/info/rfc4422>.
- [39] T. Hansen, *SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms*, RFC 7677, Nov. 2015. DOI: 10.17487/RFC7677. Adresse: <https://www.rfc-editor.org/info/rfc7677>.
- [40] *A08:2021 - Software and Data Integrity Failures*, OWASP. Adresse: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/ (besucht am 29.10.2025).
- [41] *A09:2021 - Security Logging and Monitoring Failures*, OWASP. Adresse: https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/ (besucht am 29.10.2025).
- [42] *Loki Documentation*, Grafana Labs. Adresse: <https://grafana.com/docs/loki/latest/> (besucht am 29.10.2025).
- [43] *Grafana Documentation*, Grafana Labs. Adresse: <https://grafana.com/docs/grafana/latest/> (besucht am 29.10.2025).
- [44] *A10:2021 - Server-Side Request Forgery*, OWASP. Adresse: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/ (besucht am 29.10.2025).
- [45] W. Peng, L. Ye, X. Du u. a., „PwnGPT: Automatic Exploit Generation Based on Large Language Models“, in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, W. Che, J. Nabende, E. Shutova und M. T. Pilehvar, Hrsg., Vienna, Austria: Association for Computational Linguistics, Juli 2025, S. 11 481–11 494, ISBN: 979-8-89176-251-0. DOI: 10.18653/v1/2025.acl-long.562. Adresse: <https://aclanthology.org/2025.acl-long.562/>.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

„Untersuchung der Maßnahmen zur Gewährleistung von Sicherheit bei Nutzung einer
Redpanda Message Queue“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzu- bewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Meryem Kilic

Aachen, 28. November 2025

Unterschrift der Studentin / des Studenten: M. Kilic