[PREPRINT]Sicherheit in Microservice-Umgebungen

Tobias Tefke[1]*, Ralf C. Staudemeyer[1]
[1] Hochschule Schmalkalden, Blechhammer 4-9, 98574 Schmalkalden, Germany

Zusammenfassung

Software wird permanent weiterentwickelt. Durch die Umsetzung weiterer Anwendungsfälle nimmt deren Komplexität zu. Dies ist in Bezug auf die Wartbarkeit und Sicherheit der Software kritisch zu sehen. Daher werden Lösungen gesucht, mit denen die Komplexität von Software verringert werden kann, ohne auf Funktionalität verzichten zu müssen. Eine Option besteht in der Implementierung von Microservices. Hierbei werden statt einem Programm viele Teilprogramme implementiert, die einen oder mehrere nah verwandte Anwendungsfälle umsetzen. Die Kommunikation mit und zwischen den Microservices geschieht über definierte Schnittstellen.

In Bezug auf die IT-Sicherheit ist bei der Implementierung von Microservices vor allem auf eine robuste Absicherung öffentlich verfügbarer Schnittstellen zu achten. In diesem Vortrag werden anhand eines Praxisbeispiels, einer selbst entwickelten quelloffenen Lehrplattform, Schwachstellen in der Absicherung von Microservices gezeigt. Im Anschluss wird gezeigt, wie diese Schwachstellen behoben werden können.

1 Einleitung

Bestehende Software-Applikationen werden konstant weiterentwickelt. Dies beinhaltet oft die Erweiterung bestehender und die Umsetzung neuer Anwendungsfälle. Infolgedessen nimmt die Komplexität der Applikationen zu. Gerade in Bezug auf die Sicherheit von Software ist eine hohe Komplexität kritisch zu werten.

Denn hochkomplexe Anwendungen können nur schwierig gewartet und weiterentwickelt werden, da der Zeitaufwand zur Einarbeitung und Anpassung der Software steigt. Deshalb werden Lösungen gesucht, welche einerseits die Umsetzung von Programmen mit vielen Anwendungsfällen ermöglichen, andererseits aber nicht zu komplex sind. Daher ist das Software-Architekturmuster der Microservices ist in den letzten Jahren zunehmend populärer geworden. Viele bekannte Unternehmen, wie z. B. Netflix, stellen ihre Applikationen auf Microservices um[1]. Microservices sind ein Architekturmuster, welches eine Alternative zur monolithischen Softwarearchitektur darstellt.

Ein weiteres Problem in der Software-Entwicklung stellen Legacy-Systeme dar. Hierbei handelt es sich um Programme, welche auf veralteten Technologien und Entwicklungsmethoden beruhen, aber für die Abarbeitung wichtiger Geschäftsprozesse genutzt werden. Die Wartung solcher Systeme gestaltet sich oft schwierig, insbesondere wenn zwischen den einzelnen Software-Modulen Abhängigkeiten bestehen. Schwierigkeiten in der Wartungen von Legacy-Systemen werden in [13] beschrieben. Bei der Modernisierung von Legacy-Systemen besteht u. a. die Möglichkeit, diese Systeme an eine Microservice-Architektur anzupassen.

Der Vortrag ist wie folgt gegliedert: zunächst wird das Architekturmuster des Microservices vorgestellt. Im Anschluss wird auf die IT-Sicherheit in Bezug auf Microservices, inbesondere die Absicherung von Schnittstellen eingegangen. Dies wird dann am Beispiel einer selbst entwickelten Lehrplattform gezeigt. Abschließend werden die Ergebnisse zusammengefasst.

*t.tefke@stud.fh-sm.de

[1] https://netflixtechblog.com/seamlessly-swapping-the-api-backend-of-the-netflix-android-app-3d4317155187; letzter Zugriff 28.07.2022.

2 Microservices

Im Folgenden soll das Softwarearchitekturmuster der Microservices vorgestellt werden. In Abschnitt 2.1 wird dieses Architekturmuster von monolithischer Software abgegrenzt. Im Anschluss werden in Abschnitt 2.2 die Prinzipen von Microservices vorgestellt. Danach wird in Abschnitt 2.3 gezeigt, wie Microservices kommunizieren. Abschnitt 2.4 geht auf das Deployment von Microservice-orientierter Software ein.

2.1 Abgrenzung zu monolithischer Software

Eine monolithische Software besteht aus einer ausführbaren Datei. Diese Datei enthält Module, die die Geschäftslogik der einzelnen Anwendungsfälle enthalten. Die Module sind hierbei stets zusammenhängend [5]. Die monolithische Softwarearchitektur wird in Abbildung 1 schematisch dargestellt.

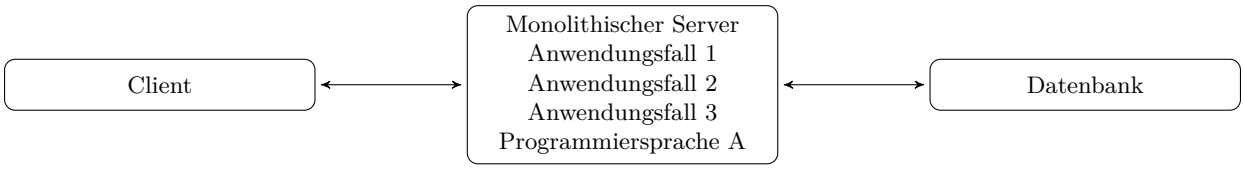

Abbildung 1: Schematische Darstellung einer monolithischen Softwarearchitektur

Die Nutzung einer monolithischen Architektur bedeutet, dass in einem monolithischen Programm stets alle Funktionen der Software zur Verfügung stehen. Wird nur ein Bruchteil der verfügbaren Funktionalitäten tatsächlich genutzt, muss im Falle eines redundanten Betriebs stets der gesamte Monolith mehrfach betrieben werden. Des Weiteren heißt dies, dass alle extern aufrufbaren Schnittstellen stets zur Verfügung stehen. Nicht genutzte Schnittstellen müssen dennoch abgesichert werden, um unbefugten Zugriff zu verhindern. Bei der Nutzung einer Microservice-Architektur kann hingegen der Betrieb auf die konkret erforderlichen Microservices begrenzt werden.

Ein wesentliches Problem in Bezug auf die Sicherheit von Programmen ist deren hohe Komplexität. Werden Fehler oder Sicherheitslücken in der Applikation bekannt, muss zunächst der fehlerhafte Quelltext identifiziert und berichtigt werden. Im Anschluss muss das Programm in Gänze neu kompiliert und ausgerollt werden. Bei Mircoservices genügt die Weiterentwicklung des betroffenen Teils. Die Probleme, die durch eine monolithische Softwarearchitektur entstehen, werden in in [5] detailliert aufgelistet.

Im folgenden Abschnitt wird die Microservice-Architektur vorgestellt.

2.2 Prinzipien von Microservices

Ein Ziel von Microservices ist es, die Zeit zwischen der (Weiter-)Entwicklung einer Software und deren Deployment zu verringern. Dies wird durch eine andere Softwarearchitektur ermöglicht: Im Gegensatz zu monolithischen Programmen, die alle Anwendungsfälle enthalten, setzt jeder Microservice einen Anwendungsfall oder mehrere nah verwandte Anwendungsfälle um. Bei Microservices handelt es sich somit zunächst um voneinander unabhängige (Teil-)Programme [5].

Microservices basieren auf drei wesentlichen Prinzipien [4]:

Begrenzter Kontext Die Funktionalitäten, die ein Microservice implementiert, sind zusammenhängend und begrenzt. Ein Microservice stellt nur einen Anwendungsfall oder mehrere eng verwandte Anwendungsfälle zur Verfügung.

Größe Da ein Microservice nur wenige Funkionalitäten (z.B. einen spezifischen Anwendungsfall) implementiert, ist der Umfang des Quelltextes überschaubar. Infolgedessen kann ein Microservice bei Bedarf relativ schnell angepasst oder neu entwickelt werden.

Unabhängigkeit Jeder Microservice kann unabhängig von anderen Microservices entwickelt und betrieben werden. Sofern Schnittstellen angepasst werden, müssen ggf. mehrere Versionen einer Schnittstelle unterstützt werden, damit Dienste, welche auf angebotene Schnittstellen zugreifen, aber noch nicht auf die neueste Version der Schnittstellen aktualisiert worden sind, weiterhin funktionsfähig bleiben.

Die verschiedenen Microservices können, falls notwendig in verschiedenen Programmiersprachen entwickelt werden. Außerdem können die einzelnen Dienste auf verschiedenen Servern betrieben werden. Hierdurch ist es möglich, stark ausgelastete Microservices zu replizieren, während bei weniger ausgelasteten Microservices auf eine Replikation verzichtet werden kann [5]. In Abbildung 2 wird eine Microservice-orientierte Softwarearchitektur schematisch dargestellt.

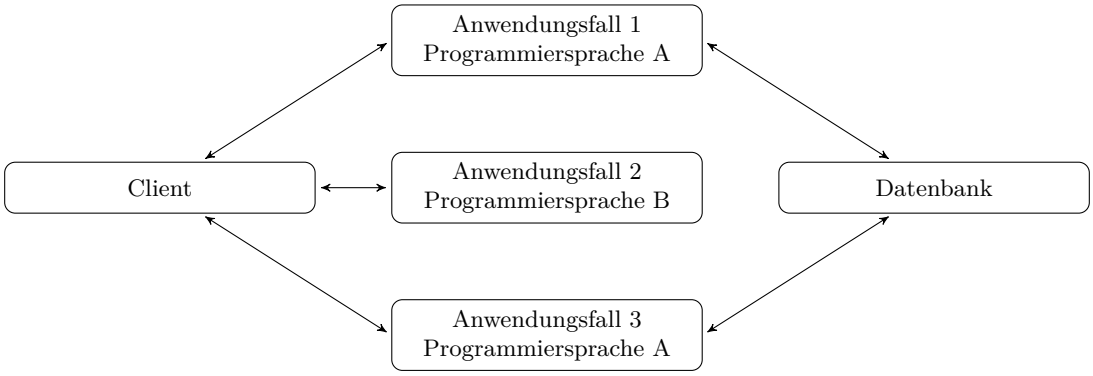

Abbildung 2: Schematische Darstellung einer Microservice-orientierten Softwarearchitektur

Die oben vorgestellten Prinzipien definieren Microservices im Wesentlichen. In der Literatur besteht Uneinigkeit darüber, ob Microservices als komplett neue Softwarearchitektur angesehen werden sollen (wie z.B. in [6]) oder ob sie als Form von Service-Orientierten Architekturen (SOA) betrachtet werden sollen (z.B. in [11]). In der Übersichtsarbeit von Zimmermann [15] werden die Eigenschaften von SOAs mit denen von Microservices verglichen. In diesem Vergleich kommt Zimmermann zu dem Ergebnis, dass der Unterschied zwischen Microservices und SOAs nicht in der Architektur begründet ist, sondern in den konkreten Technologien, die zur Entwicklung und zum Deployment verwendet werden.

Um auf die Funktionalitäten von Microservices zugreifen zu können, ist es notwendig, dass diese eine Schnittstelle anbieten. Da ein einzelner Microservice immer nur einen Teil der Gesamtfunktionalität erbringt, ist es ggf. auch erforderlich, dass Microservices auf andere Microservices zugreifen. Im folgenden Abschnitt wird beschrieben, welche Protokolle zur Kommunikation mit und zwischen Microservices genutzt werden können.

2.3 Kommunikation mit und zwischen Microservices

Um auf die von Microservices angebotenen Schnittstellen zuzugreifen, müssen diese Möglichkeiten zur Interprozesskommunikation bereitstellen [3, 5]. Bei der Kommunikation von und mit Microservices muss zwischen interner und externer Kommunikation unterschieden werden.

Bei interner Kommunikation kommunizieren Microservices untereinander. Die für interne Kommunikation angebotenen Schnittstellen können nur von anderen Microservices angesprochen werden. Externe Schnittstellen können dahingegen auch von Benutzern des Gesamtprogramms aufgerufen werden [4].

Des Weiteren kann die Kommunikation synchron oder asynchron sein. Bei asynchroner Kommunikation können Microservices bei einem Message-Broker bestimmte Events abonnieren. Tritt das Event ein, erhalten die Microservices eine entsprechende Nachricht. Protokolle zum asynchronen Nachrichtenaustausch sind z.B. AMQP, STOMP, RabbitMQ, ZeroMQ und MQTT [3, 6].

Synchrone Kommunikation zeichnet sich dadurch aus, das Anfragen sofort beantwortet werden. Hierfür bieten sich unter anderem Protokolle wie HTTP-basiertes REST oder Thrift an. Zur Definition von Schnittstellen bietet es sich an, eine Interface Definition Language (IDL) zu nutzen [3, 6].

Eine populäre IDL ist OpenAPI[2]. Mit Tools wie Swagger[3] ist es sogar möglich, automatisiert Programmgerüste zu generieren, die in OpenAPI definierte Schnittstellen anbieten oder nutzen. Die Website OpenAPI.Tools[4] stellt einen Überblick über verfügbare OpenAPI-Werkzeuge bereit.

Neben der Definition und Implementierung der Schnittstellen, ist ein gemeinsames Deployment der Microservices notwendig, sodass diese im Verbund die definierten Funktionalitäten erbringen können. Hierauf wird im nächsten Abschnitt eingegangen

2.4 Deployment von Microservices

Zum Deployment von Microservices bieten sich Container an. Bei einem Container handelt es sich um eine vom Betriebssystem isolierte Laufzeitumgebung [1, S. 239–241]. Eine populäre Software zur Erstellung und Verwaltung von Containern ist Docker[5].

Docker ist aus folgenden Gründen zum Deployment von Microservices geeignet [8]:

Automatisierung Jeder Microservice kann in einen Docker-Container gekapselt werden. Mit Dockerfiles[6] ist es möglich, ein Skript zu schreiben, mit dem der entsprechende Container automatisch erstellt wird.

Unabhängigkeit Docker-Container können unabhängig voneinander entwickelt werden. Die Vernetzung mit anderen Containern findet erst im Deployment statt[7].

Portabilität Docker ist für verschiedene Betriebssysteme verfügbar[8]. Dockerfiles können von allen unterstützten Betriebssystemen ohne Änderungen interpretiert werden.

Leichtgewichtigkeit Docker nutzt zur Isolation der Container sogenannte Namespaces[9]. Hierbei handelt es sich um eigene Sichten auf den Userspace, welche durch den Kernel bereitgestellt werden[10]. Somit ist der Ressourcenverbrauch zum Bereitstellen eines Containers wesentlich geringer als das Betreiben einer virtuellen Maschine, wofür ein komplettes Betriebssystem virtualisiert werden muss.

Sicherheit Durch die Kapselung von Diensten in Containern können diese von anderen Microservices und Programmen isoliert werden. Sollte die Kontrolle über einen Container übernommen werden, muss ein Angreifer immer noch die Isolation des Betriebssystems übernehmen. Sollte ein Container eine Sicherheitslücke aufweisen, kann dieser nach dem Finden und Absichern des Angriffsvektors innerhalb von Sekunden neu ausgerollt werden.

Auf den Punkt der Sicherheit wird in den nächsten Abschnitten näher eingegangen.

3 IT-Sicherheit in Bezug auf Microservices

Bei Microservices handelt es sich, wie in den vorherigen Abschnitten herausgearbeitet, um ein Software-Architekturmuster. In diesem kommunizieren die einzelnen Dienste über teilweise öffentlich erreichbare Schnittstellen. In Bezug auf die IT-Sicherheit stellt sich hierbei unter anderem die Frage der Absicherung dieser.

[2] `https://spec.openapis.org/oas/latest.html`; letzter Zugriff 12.08.2022.
[3] `https://swagger.io/`; letzter Zugriff 12.08.2022.
[4] `https://openapi.tools`; letzter Zugriff 12.08.2022.
[5] `https://www.docker.com/`; letzter Zugriff 28.07.2022.
[6] `https://docs.docker.com/engine/reference/builder/`; letzter Zugriff 12.08.2022.
[7] `https://docs.docker.com/network/`; letzter Zugriff 12.08.2022.
[8] `https://docs.docker.com/get-docker/`; letzter Zugriff 12.08.2022.
[9] `https://docs.docker.com/engine/security/userns-remap/`; letzter Zugriff 12.08.2022.
[10] `https://www.linux.com/news/understanding-and-securing-linux-namespaces/`; letzter Zugriff 12.08.2022.

Dennoch darf nicht in Vergessenheit geraten, dass die IT-Sicherheit einer Entität nur so stark ist wie das schwächste Glied der Kette. Hierbei muss berücksichtigt werden, dass Programme stets auf (Standard-)Bibliotheken und Betriebssystemschnittstellen zugreifen. Daher muss neben einer sicheren Softwareentwicklung auch darauf geachtet werden, dass die Server, auf denen die Dienste betrieben werden, sicher sind.

In [14] wird eine Microservice-Sicherheitstaxonomie vorgestellt. Diese beinhaltet folgende Ebenen:

Hardware Die verwendete (Server-)Hardware muss vertrauenswürdig sein. Hardwarebugs und -Backdoors stellen Bedrohungen in Bezug auf die Sicherheit der verwendeten Hardware dar.

Virtualisierung Die einzelnen Dienste müssen voneinander separiert sein, es darf nicht möglich sein aus Containern oder Hypervisoren auszubrechen.

Cloud Heutzutage werden Programme oft bei Dienstleistern "in der Cloud" betrieben. Den Dienstleistern darf es nicht möglich sein, auf Dienste von Kunden zuzugreifen oder diese zu beeinflussen.

Kommunikation Verwendete Kommunikationsprotokolle müssen abgesichert sein. Es darf Dritten nicht möglich sein, die Integrität der ausgetauschten Daten zu verletzen. Des Weiteren muss deren Vertraulichkeit gewahrt werden.

Dienst/Applikation Der Microservice selbst muss sicher und robust sein. Die Sicherheit des Programms kann z. B. mit Codeanalysen und Eingabeprüfungen verbessert werden.

Orchestrierung Die Orchestrierung der verschiedenen Microservices, die gemeinsam Anwendungsfälle implementieren, darf nicht Dritten zugänglich sein.

In diesem Vortrag werden wir uns auf die Elemente der Taxonomie beschränken, auf die Softwareentwickler direkt Einfluss nehmen können. Dies sind insbesondere die Ebenen der Kommunikation und des Dienstes/der Applikation. In Abschnitt 3.1 stellen wir Methoden vor, die zur Entwicklung sicherer Microservices genutzt werden können. Anschließend gehen wir in Abschnitt 3.2 auf das sichere Deployment von Containern in Docker ein.

3.1 Entwicklung sicherer Microservices

Um Microservices sicher zu entwickeln, ist es notwendig, deren Schnittstellen nach außen abzusichern. In Abschnitt 3.1.1 wird hierauf detailliert eingegangen. Da Microservices oft Abhängigkeiten untereinander besitzen, ist es außerdem erforderlich, den Ausfall eines abhängigen Dienstes rechtzeitig zu erkennen und entsprechend zu reagieren. Ein populäres Entwurfsmuster, welches dies ermöglicht, wird in Abschnitt 3.1.2 vorgestellt.

3.1.1 Authentifikation

Da Microservices oft HTTP-Anfragen zur Kommunikation nutzen, bieten sich zur Authentifikation von Nutzern Verfahren an, welche bereits durch das HTTP-Protokoll ermöglicht werden werden. Hierfür existieren bereits eine Vielzahl an Verfahren, welche auf der Website der IANA aufgelistet werden[11].

Eine weitere Möglichkeit zur Sicherstellung der Authentizität von Nutzern ist die Nutzung von JSON Web Token (JWT). JWT zeichnen sich dadurch aus, das diese direkt an HTTP Anfragen angehängt werden können. JWT ermöglichen es, verifizierbare Informationen zwischen mehreren Parteien auszutauschen. Hierzu werden die auszutauschenden Informationen in JavaScript Object Notation (JSON) codiert. Im Anschluss kann ein Message Authentication Code (MAC) generiert werden, mit welchem die Integrität der enthaltenen Informationen nachgewiesen werden kann. Die ausgetauschten Daten können, falls gewünscht, verschlüsselt werden [9].

[11] `https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml`; letzter Zugriff 08.08.2022.

JWT gelten als URL-sicher [9]. Ruft ein Programm eine externe Schnittstelle eines Microservices über eine HTTP-Anfrage auf, kann das JWT als Parameter an die URL angehängt werden. Der Microservice kann durch die Verifizierung des Tokens die Authentizität des Clients prüfen.

Nachdem sich ein Nutzer bei einem Anmeldedienst authentifiziert hat, kann dieser dem Nutzer ein JWT ausstellen, welches dieser zur Authentifikation bei anderen Microservices verwenden kann. Bei der Generierung eines JWT lässt sich ein technisches Ablaufdatum einstellen, um so eine regelmäßige Re-Authentifizierung einzufordern. Des Weiteren können im JWT weitere Informationen hinterlegt werden, beispielsweise die Rechte, die dem Nutzer gewährt werden sollen [9].

Weitere Möglichkeiten in der Authentifikation bestehen u. a. in der Nutzung von OpenID Connect und OAuth 2.0. Bei der Authentifikation muss darauf geachtet werden, die Token sicher, d.h. verschlüsselt zu übertragen, um zu verhindern, dass andere Netzwerkknoten die Token mitlesen können.

In [7] wird vorgeschlagen, sogenannte Sidecars einzuführen. Hierbei handelt es sich um einen separaten Microservice, der sicherheitsrelevante Operationen (wie z.B. die Authentifikation) vornimmt. Sobald sich ein Nutzer bei einem Microservice authentifizieren möchte, wird diese an das Sidecar delegiert.

Der wesentliche Vorteil in dieser Delegation sicherheitsrelevanter Operationen besteht darin, dass die Funktionalität des Sidecars zum einen von sämtlichen Microservices genutzt werden kann. So müssen entsprechende Funktionen zur Absicherung der Microservices nicht in jedem Microservice separat definiert werden [7]. Da zu erwarten ist, dass das Sidecar stark genutzt wird, bietet sich hier eine Replikation an.

Im folgenden Abschnitt wird darauf eingegangen, wie ein Microservice mit Ausfällen von Microservices, die er zur Erbringung seiner Dienste benötigt, umgehen kann.

3.1.2 Fehlertoleranz

Zwischen verschiedenen Microservices können Abhängigkeiten bestehen. Hieraus folgt, dass der Ausfall eines Microservices zu (Teil-)Ausfall anderer Microservices führen kann. Diese kaskadierenden Ausfälle führen dazu, dass die Microservices die spezifizierten Dienstleistungen (teilweise) nicht mehr anbieten können. Allerdings sollten Microservices trotz Teilausfällen des Systems weiterhin so viel Funktionalität wie möglich erbringen können [14].

Um Ausfallkaskaden zu vermeiden, bietet sich das Entwurfsmuster des Circuit Breakers [10] an. Hierbei werden die Antworten von Anfragen an andere Dienste gesammelt. Im Anschluss wird ausgewertet, wie viele Anfragen pro Zeiteinheit erfolgreich beantwortet worden. Hieraus lassen sich Schwellwerte errechnen, auf deren Basis der Circuit Breaker einen der folgenden drei Zustände annehmen kann [10]:

Geschlossen Ist der Circuit Breaker geschlossen, werden alle Anfragen an einen anderen Microservice weitergeleitet. Der Circuit Breaker ist in diesem Zustand, wenn er davon ausgeht, dass der andere Microservice ordnungsgemäß funktioniert.

Geöffnet Ist der Circuit Breaker geöffnet, wird keine Anfrage weitergeleitet. Stattdessen wird sofort eine Fehlermeldung zurückgegeben. Dieser Zustand wird angenommen, wenn von einem Ausfall des anderen Microservices ausgegangen wird.

Halboffen Ist der Circiut Breaker halboffen, wird ein Teil der Anfragen an den anderen Microservice weitergeleitet. Die anderen Anfragen werden mit einer Fehlermeldung beantwortet. Dieser Zustand wird genutzt, um nach einem Ausfall wieder zu einem funktionsfähigen Stand zurückzukehren, ohne einen anderen Microservice mit einer Flut an Anfragen zu überlasten.

Abgesehen von einer sicheren Softwareentwicklung ist auch ein sicheres Deployment von Containern wichtig, um die Sicherheit des Gesamtsystems gewährleisten zu können. Hierauf wird im folgenden Abschnitt näher eingegangen.

3.2 Sicheres Deployment von Containern

Sobald Container ausgerollt werden, ist auf ein sicheres Deployment zu achten, um eine Einhaltung der Schutzziele (v.a. CIA-Prinzipien, siehe [2, S. 2–4]) zu gewährleisten. Die OWASP Foundation hat ein Cheat Sheet zum sicheren Betrieb von Docker-Containern veröffentlicht[12]. In diesem Cheat Sheet wird auch auf einige Analysetools verwiesen, welche u.a. Schwachstellen in Dockerfiles automatisiert identifizieren können. Eine Docker Top 10-Liste ist ebenfalls in Arbeit[13].

Eine weitere wichtige Maßnahme zum sicheren Betreiben von Microservices ist eine gute Isolation des Programms vom Host. Möglichkeiten, die bestehende Isolation zu verbessern werden in Abschnitt 3.1.2 vorgestellt. Im Anschluss, in Abschnitt 3.2.2 wird darauf eingegangen, wie die Vertraulichkeit der Kommunikation von Microservices sichergestellt werden kann.

3.2.1 Isolation von Containern

Beim Deployment von Containern werden, wie bereits in Abschnitt 2.4 beschrieben, Namespaces erstellt. Hierbei handelt es sich um unterschiedliche Sichten auf das Betriebssystem. Ein Container ist also nicht in der Lage, auf andere Dienste, die auf einem Rechner laufen, zuzugreifen.

Mit Technologien wie AppArmor oder SELinux besteht die Möglichkeit, in Containern zusätzliche Sicherheitsrichtlinien zu definieren[14]. Hiermit können weitere Zugriffsbeschränkungen durchgesetzt werden.

Außerdem können Container in voneinander isolierten Netzwerken betrieben werden[7]. Somit besteht die Möglichkeit, Netzwerke zu erstellen, in welchen sich ausschließlich die Microservices befinden, welche aufgrund von Abhängigkeiten miteinander kommunizieren müssen. Dies verhindert, dass ein Angreifer im Falle der Übernahme eines Containers Anfragen an alle auf dem Host laufenden Container stellen kann.

Im Falle eine Übernahme ist es wichtig, diese schnell zu erkennen und gegenzusteuern, um weitere Schäden zu verhindern. Hierfür bieten sich Monitoring-Tools an. Tools wie Weave Scope[15] und FlowTap [12] sind speziell auf Container-Umgebungen optimiert.

Im folgenden Abschnitt wird noch auf die Gewährleistung der Vertraulichkeit bei der Kommunikation über Netze eingegangen.

3.2.2 Gewährleistung der Vertraulichkeit

Um die Vertraulichkeit in der internen und externen Kommunikation von Microservices zu gewährleisten, muss der Zugriff auf den Inhalt der ausgetauschten Nachrichten beschränkt werden. Hierfür bietet sich v. a. die Nutzung von Verschlüsselung an.

Externe Schnittstellen können hierbei mit TLS abgesichert werden. Um die Kommunikation zwischen Containern, also interne Kommunikation, zu verschlüsseln, können diese Mutual TLS (MTLS) nutzen. Hierbei wird in einem Deployment zusätzlich zu den Mircoservices eine Public Key-Infrastruktur (PKI) aufgesetzt. Diese stellt Schlüssel bereit, welche zum vertraulichen Austausch von Nachrichten zwischen Containern genutzt werden können.

Es existieren verschiedene Ansätze zur Bereitstellung der PKI, welche in [14] vorgestellt werden. Im Anschluss wird auf das sichere Deployment von Microservices am Beispiel einer von uns entwickelten Lehrumgebung eingegangen.

[12] `https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html`; letzter Zugriff 12.08.2022.

[13] `https://owasp.org/www-project-docker-top-10/`; letzter Zugriff 12.08.2022.

[14] `https://docs.docker.com/engine/security/`; letzter Zugriff 12.08.2022.

[15] `https://github.com/weaveworks/scope`; letzter Zugriff 13.08.2022.

4 Microservices am Praxisbeispiel einer Lehrumgebung

Im Folgenden sollen Sicherheitsprobleme in Microservices am Beispiel einer Lehrumgebung vorgestellt werden. Diese wird in Abschnitt 4.1 kurz präsentiert. Im Anschluss wird in Abschnitt 4.2 knapp auf die Ergebnisse einer Sicherheitsanalyse der Lehrumgebung eingegangen.

4.1 Kurzvorstellung der Lehrumgebung und der verwendeten Programme

Studenten unserer Hochschule haben im Rahmen eines IT-Sicherheitskurses auf Basis der quelloffenen Software WorkAdventure[16] eine Plattform entwickelt, welche zur dezentralen Lehre genutzt werden kann. Hierbei bewegen die Teilnehmer einer Lehrveranstaltung einen Charakter über eine Karte. Sobald mehrere Charaktere nah beieinander stehen, öffnet sich ein Kommunikationskanal zwischen den jeweiligen Nutzern. Bestimmten Bereichen der Karte könnnen Funktionalitäten, sogenannte Aktionen, zugewiesen werden. Diese werden beim Betreten des Bereiches im Browser des Nutzers ausgeführt.

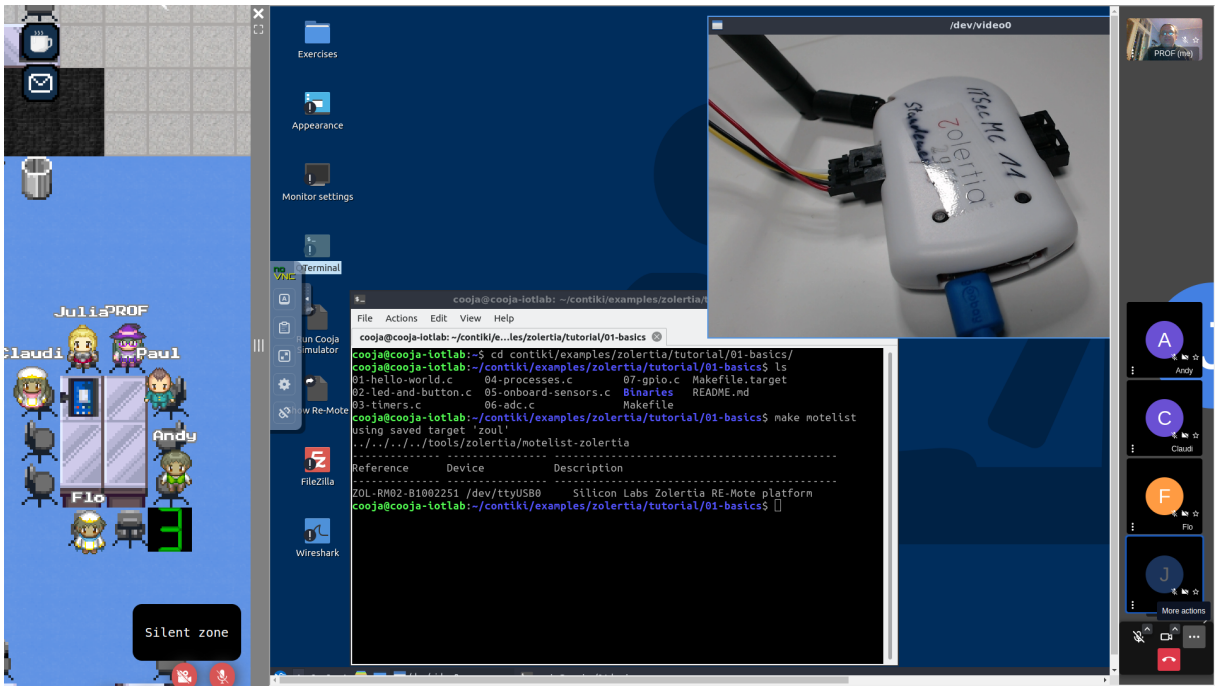

Abbildung 3: Gemeinsames Programmieren von Microcontrollern in der Lehrplattform.

Für unsere Lehrplattform wurden die Aktionen um Funktionalitäten erweitert, die für die Lehre wichtig sind. Dies beinhaltet unter anderem virtuelle Gruppenarbeitsplätze, an denen Studierende gemeinsam das Programmieren von Microcontrollern erlernen können (siehe Abbildung 3). Hierzu werden virtuelle Maschinen betrieben, die beim Betreten des Arbeitsplatzes mittels noVNC[17] in WorkAdventure eingebunden werden. Des Weiteren wurde BigBlueButton[18] in WorkAdventure eingebunden, um Online-Vorlesungen in einem virtuellen Vorlesungsraum durchführen zu können.

Die Entwicklung war darauf fokussiert, eine möglichst sichere Plattform zu entwickeln. Im folgenden wird auf die Ergebnisse einer eigenen Sicherheitsanalyse eingegangen.

[16] `https://workadventu.re/`; letzter Zugriff 28.07.2022.
[17] `https://novnc.com/info.html`; letzter Zugriff 28.07.2022.
[18] `https://bigbluebutton.org/`; letzter Zugriff 28.07.2022.

4.2 Sicherheitsanalyse der verwendeten Programme

Für die Realisierung der Lehrplattform wird eine Microservice-Architektur genutzt. Hierbei wurde der Fokus besonders auf die Sicherheit der genutzen Programme gelegt. Daher wurde auch eine Sicherheitsanalyse der eingebundenen Open-Source-Programme durchgeführt.

Hierbei wurden verschiedene Angriffsvektoren gefunden, von denen zwei nun exemplarisch vorgestellt werden:

So existierten in mehreren Microservices HTTP-Schnittstellen, über die Debug-Informationen abgerufen werden konnten. Diese Informationen enthalten unter anderem die IP-Adressen der mit WorkAdventure verbundenen Nutzer. Die entsprechenden HTTP-Schnittstellen werden zwar durch einen Authentifizierungsmechanismus geschützt. In der Open-Source-Variante des Microservices wurde hierfür jedoch ein Standardtoken genutzt, welches im Code der Applikation hinterlegt war. Somit konnte jeder die Debug-Informationen mit einem Client wie curl[19] abfragen.

In einem anderen Fall war es sogar möglich, mit einer speziellen Anfrage einen Microservice zum Absturz zu bringen. Hierbei wurde zwar erkannt, dass die Anfrage fehlerhaft ist und eine entsprechende Fehlermeldung generiert. Allerdings wurde diese Fehlermeldung im Anschluss nicht korrekt behandelt und führte zum Absturz des Microservice.

Die identifizierten Angriffsvektoren basieren im Wesentlichen auf folgenden Problemen:

- Standard- und Debugzugänge, welche aufgrund fehlender Dokumentation für Systemadministratoren im Produktiveinsatz nicht abgeschaltet werden
- fehlende Authentifizierung von Nutzern bei der Annahme von Anfragen
- öffentliches Anbieten von Schnittstellen, die nur intern zugänglich sein sollten
- fehlende Prüfung von Werten, die bei Anfragen übergeben werden

Neben Softwarefehlern wurden auch Fehler in der Konfiguration der Container gefunden. So wurde ein Großteil der Programme, welche in Containern gekapselt waren mit root-Rechten. Dies konnte mit einer Rekonfiguration der Container gelöst werden.

Außerdem enthielten viele Quelltext eigenen Quelltext zur Nutzerauthentifizierung. Dies hat zur Folge, dass die Geheimnisse, welche zur Authentifikation genutzt werden, über viele Microservices gestreut sind. Die Lösung besteht hierbei in der Implementierung eines Sidecar-Microservices, welches die Authentifikationsdienste übernimmt.

Des Weiteren wurde festgestellt, dass sich alle Container innerhalb eines Netzwerkes befinden und die Kommunikation zwischen den Containern unverschlüsselt erfolgte. Um dieses Problem zu lösen, wird im Swarm-Modus MTLS aktiviert. Des Weiteren werden die Microservices in mehrere Subnetze gelegt, um so unabhängige Teile des Gesamtprojektes voneinander zu isolieren.

5 Fazit

In diesem Artikel wird auf mögliche Sicherheitsprobleme in Microservice-Umgebungen aufmerksam gemacht. Hierzu wurden zunächst die Charakteristika der Microservice-Architektur vorgestellt. Dabei wurde auch auf die Kommunikation zwischen den einzelnen Microservices und deren Deployment eingegangen. Danach werden mögliche Sicherheitsprobleme erläutert. Dies wird auch anhand von in unserer Plattform gefundenen Schwachstellen veranschaulicht. Hierbei werden geeignete Maßnahmen zur Lösung der beschriebenen Probleme vorgestellt.

Referenzen

[1] Christian Baun. *Betriebssysteme kompakt*. Springer Vieweg, 2017. DOI: https://doi.org/10.1007/978-3-662-53143-3.

[19] https://curl.se/; letzter Zugriff 28.07.2022.

[2] Matt Bishop. *Introduction to Computer Security*. Addison-Wesley, 2004. ISBN: 0-321-24744-2.

[3] Ramaswamy Chandramouli. „Security Strategies for Microservices-based Application Systems“. In: *NIST Special Publication* 800.204 (Aug. 2019). DOI: 10.6028/NIST.SP.800-204.

[4] Nicola Dragoni u. a. „Microservices: How To Make Your Application Scale“. In: *Perspectives of System Informatics*. Hrsg. von Alexander K. Petrenko und Andrei Voronkov. Cham: Springer International Publishing, 2018, S. 95–104. ISBN: 978-3-319-74313-4.

[5] Nicola Dragoni u. a. „Microservices: Yesterday, Today, and Tomorrow“. In: *Present and Ulterior Software Engineering*. Hrsg. von Manuel Mazzara und Bertrand Meyer. Cham: Springer International Publishing, 2017, S. 195–216. DOI: 10.1007/978-3-319-67425-4_12.

[6] Martin Fowler und James Lewis. „Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr“. In: *Objektspektrum* 1.2015 (2015), S. 14–20.

[7] Kasun Indrasiri und Prabath Siriwardena. „Microservices Security Fundamentals“. In: *Microservices for the Enterprise: Designing, Developing, and Deploying*. Berkeley, CA: Apress, 2018, S. 313–345. ISBN: 978-1-4842-3858-5. DOI: 10.1007/978-1-4842-3858-5_11. URL: https://doi.org/10.1007/978-1-4842-3858-5_11.

[8] David Jaramillo, Duy V Nguyen und Robert Smart. „Leveraging microservices architecture by using Docker technology“. In: *SoutheastCon 2016*. 2016, S. 1–5. DOI: 10.1109/SECON.2016.7506647.

[9] M. Jones, J Bradley und N. Sakimura. *RFC 7519. JSON Web Token (JWT)*. online. Mai 2015. DOI: 10.17487/RFC7519.

[10] Fabrizio Montesi und Janine Weber. *Circuit Breakers, Discovery, and API Gateways in Microservices*. 2016. DOI: 10.48550/ARXIV.1609.05830.

[11] Sam Newman. *Building Microservices. Designing Fine-grained Systems*. Ö'Reilly Media, Inc.", 2015. ISBN: 978-1-491-95035-7.

[12] Yuqiong Sun, Susanta Nanda und Trent Jaeger. „Security-as-a-Service for Microservices-Based Cloud Applications“. In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. 2015, S. 50–57. DOI: 10.1109/CloudCom.2015.93.

[13] Aparna Vijaya und Neelanarayanan Venkataraman. „Modernizing legacy systems: a re-engineering approach“. In: *International Journal of Web Portals (IJWP)* 10.2 (2018), S. 50–60.

[14] Tetiana Yarygina und Anya Helene Bagge. „Overcoming Security Challenges in Microservice Architectures“. In: *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. 2018, S. 11–20. DOI: 10.1109/SOSE.2018.00011.

[15] Olaf Zimmermann. „Microservices tenets“. In: *Computer Science-Research and Development* 32.3 (2017), S. 301–310.