

# **Anomalie-Erkennung in Keycloak-Logs mittels maschinellen Lernens: Vergleich von Hybridmodellen**

Vorgelegt von:

**Ümmühan Ay**

Matrikelnummer: 7060837

Eingereicht am: 24.03.2025

Abgabedatum: 08.09.2025

Duale Hochschule Baden-Württemberg Stuttgart  
Fakultät für Informatik

Erstprüfer: Eric Hämmerle  
Zweitprüfer: Dr. Janko Dietzsch

# Abstract

## **Zusammenfassung**

## Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind in der Arbeit angegeben. Diese Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, Datum

Unterschrift

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>12</b>
1.1	Benutzerverhalten und Anomalien . . . . .	12
1.2	Kategorisierung von Anomalien . . . . .	13
1.3	Regelbasierte Methoden und weitere Maßnahmen in Keycloak . . . . .	15
1.4	Probleme: Grenzen der Sicherheitsmaßnahmen in Keycloak . . . . .	15
1.5	Motivation: Maschinelle Lernmodelle als zukünftige Sicherheitsmaßnahme . . . . .	16
1.6	User (and Entity) Behavior Analytics (U(E)BA) . . . . .	17
<b>2</b>	<b>Forschungsstand, Herausforderungen und Hypothesen</b>	<b>18</b>
2.1	Bisheriger Forschungsstand . . . . .	18
2.2	Problematik und Begrenzung von Modellen . . . . .	19
2.3	Hypothese . . . . .	20
<b>3</b>	<b>Methoden und Störfaktoren</b>	<b>21</b>
3.1	Metriken . . . . .	21
3.2	Andere, nicht angewandte Metriken . . . . .	24
3.3	Störfaktoren . . . . .	24
<b>4</b>	<b>Anforderungsdefinition und Analyse</b>	<b>26</b>
4.1	Funktionale Anforderungen . . . . .	26
4.2	Nicht-funktionale Anforderungen . . . . .	26
<b>5</b>	<b>Angewandte Technologien</b>	<b>26</b>
5.1	Scikit-learn Learn . . . . .	26
5.2	Deep GPU Xceleration (DGX) . . . . .	27
<b>6</b>	<b>Angewandte Technologie: Keycloak</b>	<b>27</b>
6.1	Terminologien in Keycloak . . . . .	27
6.2	Komponenten der Log-Erzeugung . . . . .	30
6.3	Log-Dateien in Keycloak . . . . .	33
6.4	Event-Logs . . . . .	34
6.5	Aufbau der Event-Logs . . . . .	34
<b>7</b>	<b>Theoretischer Hintergrund der Modelle</b>	<b>37</b>
7.1	LSTM . . . . .	37
7.2	Autoencoder . . . . .	37
7.3	Isolation Forest . . . . .	40

7.4	One-Class SVM . . . . .	41
7.5	DBSCAN . . . . .	42
7.6	Alternative Algorithmen . . . . .	43
<b>8</b>	<b>Implementierung</b>	<b>43</b>
8.1	Hybridmodelle mit LSTM-AE . . . . .	43
8.1.1	Implementierung: Isolation Forest . . . . .	45
8.1.2	Implementierung: OCSVM . . . . .	45
8.1.3	Implementierung: DBSCAN . . . . .	46
8.2	Alternative Architekturen . . . . .	46
<b>9</b>	<b>Möglichkeit 1: Generierung der Logs</b>	<b>47</b>
9.1	Angriffsfälle . . . . .	47
9.2	Automatische Erstellung der Keycloak-Logs . . . . .	48
9.3	Problematik dieser Lösung . . . . .	50
<b>10</b>	<b>Möglichkeit 2: Datenbeschaffung als Alternative</b>	<b>51</b>
10.1	Anbindung der Keycloak-API . . . . .	51
10.2	Selenium-Tests zur Erzeugung der Logs . . . . .	52
10.3	Mögliche Angriffsszenarien nach CVE und CWE . . . . .	57
10.4	Selenium-Tests zur Erzeugung der Angriffsszenarien . . . . .	58
<b>11</b>	<b>Durchführung des Trainings</b>	<b>60</b>
<b>12</b>	<b>Ergebnisse</b>	<b>60</b>
<b>13</b>	<b>Diskussion</b>	<b>60</b>
<b>14</b>	<b>Fazit</b>	<b>60</b>
<b>15</b>	<b>Ausblick</b>	<b>60</b>

# Abbildungsverzeichnis

1	Protokolle für IdPs in Keycloak . . . . .	28
2	Beispielhafte Konfiguration von Authentication Flows in Keycloak . .	29
3	Beispiel: Aktivierung von Quarkus-Logging in der Keycloak-Konfiguration	31
4	Log-Ausgaben, wie sie in einer Quarkus-basierten Keycloak-Instanz erzeugt werden . . . . .	32
5	Beispiel Events nach Eventtyp . . . . .	36
6	Visuelle Darstellung der Funktion des AEs, Quelle: <a href="https://www.researchgate.net/figure/Structure-of-the-autoencoder_fig1_365074431">https://www.researchgate.net/figure/Structure-of-the-autoencoder_fig1_365074431</a> . . . . .	39
7	Visuelle Darstellung der Funktion des AEs, Quelle: <a href="https://www.researchgate.net/publication/350551253/Automatic-Detection-using-Isolation-Forest-18_fig3_350551253">https://www.researchgate.net/publication/350551253/Automatic-Detection-using-Isolation-Forest-18_fig3_350551253</a> . . . . .	40
8	Visuelle Darstellung OCSVM. Quelle: <a href="https://www.mdpi.com/2076-3417/13/3/1734">https://www.mdpi.com/2076-3417/13/3/1734</a> . . . . .	42
9	Beispiel DBSCAN-Cluster, Quelle: <a href="https://www.baeldung.com/wp-content/uploads/sites/4/2023/04/clusters.png">https://www.baeldung.com/wp-content/uploads/sites/4/2023/04/clusters.png</a> . . . . .	42
10	Architektur . . . . .	44
11	Ausschnitt der generierten Logs . . . . .	50
12	Daten, die an den Keycloak-Server gesendet werden müssen . . . . .	52
13	Hier sieht man die generierten Benutzer für den Testfall . . . . .	53
14	Benötigte Rollen für den Client Nintendo . . . . .	55

# Tabellenverzeichnis

1	Bestimmte Parameter der Modellkonfiguration . . . . .	45
---	---	----



# Abkürzungsverzeichnis

**ABAC:** Attribute-Based Access Control

**AE:** Autoencoder

**AUC-PR:** Area Under the Precision and Recall Curve

**AUC-ROC:** Area Under the ROC Curve

**DBSCAN:** Density-Based Spatial Clustering of Applications with Noise **DGX:** Deep GPU Xceleration

**IAM:** Identity und Access Management

**IdP:** Identity Provider (Identitätsanbieter)

**IF:** Isolation Forest

**LSTM:** Long Short Term Memory

**MCC:** Matthews Correlation Coefficient

**MITRE ATT&CK:** MITRE Adversarial Tactics, Techniques, and Common Knowledge

**ML:** Machine Learning

**OCSVM:** One Class Support Vector Machine

**OIDC:** OpenID Connect

**RBAC:** Role-Based Access Control

**RNN:** Recurrent Neural Network

**SAML:** Security Assertion Markup Language

**SSO:** Single-Sign-On

**UBA:** User Behavior Analytics

**UEBA:** User and Entity Behavior Analytics

# Glossar

**ABAC** Zugriffskontrolle basierend auf Benutzerattributen wie Rolle, Standort, Zeit etc.. 10

**AE** Autoencoder – Neuronales Netzwerk zur komprimierten Darstellung und Rekonstruktion von Daten.. 12

**AUC-PR** Area Under the Precision and Recall Curve – misst die Leistung eines Modells im Umgang mit unausgeglichene Klassen.. 14

**AUC-ROC** Area Under the Receiver Operating Characteristic Curve – misst die Trennschärfe eines Klassifikationsmodells.. 14

**Balanced Accuracy** Durchschnitt aus Sensitivität (Recall) und Spezifität; besonders geeignet bei unausgegleichen Klassenverhältnissen. 15

**DBSCAN** Density-Based Spatial Clustering of Applications with Noise – Clustering-Verfahren für dichte Datenregionen.. 11

**DGX** Deep GPU Xceleration – NVIDIA-Hardwareplattform zur KI-Beschleunigung.. 3, 17

**Hybridmodell** Ein Modell, das zwei oder mehr unterschiedliche Verfahren kombiniert, z. B. LSTM-Autoencoder mit einem klassischen ML-Algorithmus wie Isolation Forest. 11

**IAM** Identity and Access Management – Verwaltung von Identitäten und Zugriffsrechten.. 10

**IF** Isolation Forest – Verfahren zur Anomalieerkennung durch zufällige Isolation.. 12

**Keycloak** Open-Source Identity-Management-Lösung mit Funktionen wie SSO, RBAC und Brute-Force-Schutz.. 3, 10

**LSTM** Long Short-Term Memory – Neuronales Netz zur Verarbeitung von Sequenzdaten mit Langzeitspeicher.. 11

**LSTM-Autoencoder** Ein neuronales Netzwerk, das auf Long Short-Term Memory-Zellen basiert, um Sequenzen zu kodieren und zu rekonstruieren, oft zur Anomalieerkennung in Zeitreihen. 11

. 3, 15

**Matthews Correlation Coefficient (MCC)** Eine robuste Metrik, die alle Werte der Konfusionsmatrix berücksichtigt; besonders geeignet bei unausgeglichene Datensätzen. 15

**MITRE ATT&CK** MITRE Adversarial Tactics, Techniques, and Common Knowledge – öffentliches Framework zur Kategorisierung von Angriffstechniken.. 23

**ML** Machine Learning – Teilgebiet der KI, bei dem Systeme aus Daten lernen.. 11

**OCSVM** One-Class Support Vector Machine – Algorithmus zur Anomalieerkennung mit einseitiger Klassifikation.. 21

**RBAC** Role-Based Access Control – Zugriffskontrolle auf Basis definierter Rollen.. 10

**RNN** Recurrent Neural Network – Neuronales Netz zur Verarbeitung zeitabhängiger Daten.. 19

**SSO** Single Sign-On – Einmalige Authentifizierung für den Zugriff auf mehrere Anwendungen.. 27

**Zero-Day-Angriff** Angriff auf eine bislang unbekannte Sicherheitslücke, für die noch kein Patch existiert.. 10

# 1 Einleitung

Laut dem BSI-Lagebericht 2024 ist die Zahl von Cyberangriffen auf Unternehmen durch Ransomware und DDoS-Attacken weiter gestiegen [1]. Angreifende entwickeln kontinuierlich neue Methoden, um Sicherheitsmechanismen zu umgehen und finanzielle oder strategische Vorteile zu erlangen. Herkömmliche, regelbasierte Schutzmaßnahmen – etwa Firewalls oder Brute-Force-Abwehrsysteme – stoßen dabei zunehmend an ihre Grenzen. Besonders schwer zu erkennen sind interne Angreifer, also etwa eigene Mitarbeitende, sowie koordinierte Attacken über Botnetze. Solche Bedrohungen erfordern fortschrittliche Verfahren zur Mustererkennung. Der Einsatz von Künstlicher Intelligenz (KI) in Sicherheitssystemen gewinnt daher zunehmend an Bedeutung. Maschinelle Lernalgorithmen bieten hier ein vielversprechendes Potenzial: Sie können Anomalien identifizieren und das Nutzerverhalten auf Basis theoretischer Modelle analysieren, um Angriffe frühzeitig zu erkennen.

## 1.1 Benutzerverhalten und Anomalien

Das Benutzerverhalten umfasst sämtliche Aktionen, Muster und Gewohnheiten, mit denen ein Nutzer ein IT-System verwendet. Dazu zählen unter anderem Login-Zeiten, Zugriffsrechte, genutzte Ressourcen, Netzwerkbewegungen sowie Interaktionen mit Anwendungen und Daten. Die Analyse des Benutzerverhaltens dient dazu, typische Nutzungsmuster zu identifizieren und Abweichungen (sogenannte Anomalien) zu erkennen, die auf Sicherheitsvorfälle oder Missbrauch hindeuten können.

Anomalien bezeichnen Datenmuster, die signifikant von erwarteten oder üblichen Verhaltensweisen abweichen. Mathematisch gesehen, ist eine Anomalie, ein abweichender Punkt vom "normalen" Bereich  $N_1$  und  $N_2$  [2]. Sie können beispielsweise im Netzwerkverkehr, in eingebetteten Systemen oder anderen datenverarbeitenden Bereichen auftreten. Solche Abweichungen deuten häufig auf Systemangriffe oder technische Störungen hin.

Neben systemweiten Anomalien – etwa im Netzwerkverkehr oder bei Gerätezuständen – gewinnen zunehmend verhaltensbasierte Auffälligkeiten an Bedeutung. Systemweite Anomalien betreffen technische Parameter und Prozesse im gesamten System, wie etwa plötzliche Veränderungen im Datenverkehr, unerwartete Systemlast oder ungewöhnliche Netzwerkverbindungen [2]. Verhaltensbasierte Anomalien hingegen beziehen sich auf das individuelle Verhalten einzelner Nutzer.

## 1.2 Kategorisierung von Anomalien

Nach [2] werden Anomalien in drei Hauptkategorien eingeteilt:

- **Punkt-Anomalien:** Einzelne Dateninstanzen, die im Vergleich zum Großteil der Daten als anomal gelten. Beispiel: Eine Kreditkartentransaktion, deren Betrag deutlich höher ist als die sonst üblichen Ausgaben einer Person.
- **Kontextuelle Anomalien:** Dateninstanzen, die nur in einem spezifischen Kontext als anomal betrachtet werden, ansonsten jedoch normal sind. Der Kontext wird durch sogenannte *kontextuelle Attribute* definiert, z. B. Zeit oder Ort. Die *verhaltensbezogenen Attribute* beschreiben die eigentlichen Eigenschaften der Dateninstanz. Beispielsweise kann eine Temperatur in einem bestimmten Gebiet im Winter als anomal gelten, im Sommer aber normal sein.
- **Kollektive Anomalien:** Gruppen von Datenpunkten, die zusammen eine Anomalie darstellen, obwohl einzelne Punkte für sich genommen normal erscheinen.

Dabei umfassen kontextuelle Anomalien zwei Arten von Attributen:

1. **Kontextuelle Attribute:** Bestimmen den Kontext, z. B. geografische Lage oder Zeitstempel.
2. **Verhaltensbezogene Attribute:** Beschreiben das tatsächliche Verhalten oder die Charakteristik der Dateninstanz, z. B. die gemessene Temperatur oder der Betrag einer Transaktion.

Obwohl zahlreiche Arbeiten im Bereich der Anomalieerkennung im Nutzerverhalten existieren, fehlt es an einer klaren und systematischen Definition dessen, was genau eine verhaltensbasierte Anomalie konstituiert. Stattdessen werden in der Regel Beispiele genannt, ohne dass diese in ein konsistentes semantisches Modell eingebettet werden. Diese Forschungslücke erschwert die Evaluation und Vergleichbarkeit von Verfahren erheblich – und wird auch in dieser Arbeit als Ausgangspunkt einer kritischen Einordnung aufgegriffen.

Es wird davon ausgegangen, dass Anomalien welche sich im Benutzerverhalten (Verhaltensbezogene Anomalien) äußern, unter anderem folgende sein können:

- ungewöhnliche Login-Zeiten,
- verdächtige Standorte,

- auffällige Rollenprofile,
- unübliche Zugriffsrechte,
- eine Häufung von Anmelde- oder Autorisierungsfehlern.

Ein Benutzer, welcher sich Mitternachts außerhalb der üblichen Geschäftszeit einloggt, ist ungewöhnlich und weicht vom gewöhnlichen Muster ab. Zudem ist es eher so, dass Arbeiter in der Region bleiben, wo sie arbeiten oder im Umkreis. Natürlich muss man hier moderne Szenarien beachten, z.B. tritt es öfter auf, dass Personen im Homeoffice arbeiten und auch weiter weg wohnen und auch manche Entwickler auch bis um Mitternacht nach einem Bug suchen. Wenn dies in einem Unternehmen der Standardfall ist, würden dies keine Anomalien darstellen sondern den üblichen Standardfall. Es wird davon ausgegangen, dass unübliche Zugriffsrechte auf sensible Daten durchaus Anomalien sein können, besonders, wenn ein bestimmter Benutzer eigentlich auf die Quelle keinen Zugriff haben darf oder auch plötzlich viele Rollen hat, die ihm zu viele Rechte in einem System geben. Anmeldefehler können ebenfalls auf Brute-Force Angriffe, bzw. Intrusionsangriffe hindeuten, sowie auf DoS-Angriffe.

Es handelt sich hierbei um verhaltensbezogene Anomalien. Alle genannten Aspekte sind in *Intrusionsangriffen* und *Insider Threat Attacks* vorhanden. Intrusionsangriffe sind Angriffe außerhalb eines Systems, welche versuchen in eine System zu gelangen, z.B. Brute-Force Angriffe. Insider Threat Attacks sind wiederum Attacken, welche im System selbst schon auftreten können. Bspw. könnte der Angreifer ein Mitarbeiter im Unternehmen selbst sein. Insider-Attacken äußern sich nach [3] in den genannten Aspekten: Benutzer zeigen abweichendes Verhalten (z.B. loggt er sich viel früher ein als üblich) oder ein Benutzer zeigt ein Verhalten viel zu oft. z.B. viele Dateien auf einmal herunterladen. Auch neue Attribute, die der Benutzer zuvor nicht, deutet auf eine Insider Attacke hin. Alle Beispiele sind den zuvor definierten Anomalien ähnlich. Man erkennt, dass die Begriffe welche die selben Anomalien beschreiben sollen, nicht klar definiert sind.

In dieser Arbeit wird der Begriff „verhaltensbasierte Anomalien“ mit den Erscheinungsformen von Intrusionsangriffen und Insiderattacken gleichgesetzt, um den Fokus auf sicherheitsrelevante Abweichungen im Nutzerverhalten zu legen. Diese begriffliche Vereinfachung erleichtert die Analyse und Methodik, obwohl Anomalien technisch betrachtet als Indikatoren für verschiedene Angriffstypen verstanden werden sollten.

Darüber hinaus gelten komplexere Abweichungen – wie Veränderungen in gewohnten Arbeitsabläufen, zeitlichen Mustern von Datei- oder Netzwerkzugriffen oder in der

Kommunikation mit untypischen Partnern – ebenfalls als Anomalien. Ungewöhnliche IP-Adressen stellen auf Netzwerkebene ein häufiges Indiz für Anomalien dar, da sie auf einen möglichen unautorisierten Zugriff, eine Spoofing-Attacke (Also Stehlen von Anmeldedaten, in dem der Angreifer eine infiltrierte Anmeldeseite beim Benutzer lädt) oder den Einsatz von anonymisierenden Diensten (z.B. VPNs oder TOR) hindeuten können. In regulären Betriebsabläufen sind IP-Adressen in der Regel geografisch, organisatorisch oder netztechnisch eingeschränkt. Weicht eine Adresse deutlich von bekannten Mustern oder erlaubten Adressräumen ab – etwa durch Zugriffe aus einem anderen Land, über Proxy-Netzwerke oder aus verdächtigen IP-Bereichen (z.B. Botnetze) – kann dies auf kompromittierte Accounts oder externe Angreifer hinweisen. Besonders im Kontext von Benutzerverhalten sind IP-basierte Auffälligkeiten oft ein erstes Signal für Insiderbedrohungen oder gestohlene Zugangsdaten.

Insgesamt bilden solche Merkmale die Grundlage moderner Systeme zur Anomalieerkennung, die auf Verfahren des maschinellen Lernens und der Statistik beruhen, um potenzielle Sicherheitsrisiken automatisiert und präventiv zu identifizieren. In dieser Arbeit liegt der Fokus auf der Erkennung von verhaltensbasierten Anomalien, bzw. auf Intrusionsangriffe und Insider Threat Attacken.

### **1.3 Regelbasierte Methoden und weitere Maßnahmen in Keycloak**

Die Intension GmbH nutzt das Produkt Keycloak<sup>1</sup>, ein IAM-Tool<sup>2</sup>, das unter anderem SSO (Single Sign-On) bietet<sup>3</sup>.

Keycloak stellt eine Reihe vorgefertigter Sicherheitsmechanismen zur Verfügung, die in der Regel regelbasiert arbeiten. Solche Maßnahmen definieren explizit, welche Aktivitäten im System erlaubt oder verboten sind. Grundlage dieser Regeln sind meist bekannte Angriffsmuster aus früheren Sicherheitsvorfällen. Zusätzlich bietet Keycloak klassische Zugriffskontrollmechanismen wie RBAC und ABAC, Brute-Force-Erkennung sowie die Möglichkeit zur Integration externer Netzwerkschutzsysteme wie Firewalls.

### **1.4 Probleme: Grenzen der Sicherheitsmaßnahmen in Keycloak**

Trotz der Vielzahl integrierter Sicherheitsfunktionen stoßen die Schutzmaßnahmen von Keycloak in der Praxis an ihre Grenzen. Regelbasierte Systeme sind statisch und

meist nur auf bekannte Muster abgestimmt. Neue, bislang unbekannte Angriffsarten (sogenannte Zero-Day-Angriffe) entziehen sich dieser Logik, da keine entsprechenden Regeln vorliegen.

Auch die Zugriffskontrollmodelle zeigen Schwächen: RBAC wird bei wachsender Komplexität und Vielzahl an Rollen schnell unübersichtlich, während ABAC zwar flexibler ist, dafür jedoch hohen Pflegeaufwand verursacht und anfällig für Fehler ist – insbesondere, wenn viele Attribute aktuell gehalten werden müssen.

Der integrierte Brute-Force-Schutz erkennt typische Fehlversuchs-Muster, ist jedoch bei modernen Angriffen mit verteilten Systemen (z. B. Botnetzen) oft ineffektiv. Firewalls bieten zwar eine erste Verteidigungslinie auf Netzwerkebene, können aber weder interne Bedrohungen noch gezielte Phishing-Angriffe zuverlässig abwehren.

Diese Schwächen verdeutlichen, dass klassische, regelbasierte Sicherheitsarchitekturen zunehmend an ihre Grenzen stoßen. Um aktuelle Bedrohungen wirksam erkennen und abwehren zu können, bedarf es dynamischer, lernfähiger Systeme, wie sie beispielsweise durch Verfahren des maschinellen Lernens und der UBA realisiert werden können.

## **1.5 Motivation: Maschinelle Lernmodelle als zukünftige Sicherheitsmaßnahme**

Die Motivation dieser Arbeit besteht darin, geeignete maschinelle Lernmodelle zu identifizieren, mit denen verhaltensbasierte Anomalien in Keycloak-Logs erkannt werden können. Maschinelle Lernmodelle stellen heute eine vielversprechende Alternative zu klassischen, regelbasierten Sicherheitsmaßnahmen dar, da sie dynamisch auf neue Bedrohungen reagieren können.

Im Gegensatz zu statischen Regeln, die lediglich bekannte Muster erfassen, sind ML-Modelle in der Lage, aus großen Datenmengen zu lernen und auch unbekannte Anomalien zu identifizieren, die auf potenzielle Angriffe hindeuten. Dadurch sind sie besonders effektiv im Umgang mit Zero-Day-Angriffen oder komplexen, sich ständig wandelnden Bedrohungsszenarien. Ein weiterer Vorteil ist die Fähigkeit zur kontinuierlichen Anpassung, sodass sich ML-Modelle automatisch auf veränderte Rahmenbedingungen einstellen, ohne dass ein manueller Eingriff notwendig ist.

Im Ausblick besteht Interesse daran, das leistungsfähigste Modell in ein Tool zu integrieren, welches von der Intension GmbH genutzt werden kann. Dadurch sollen Kosten eingespart werden, da herkömmliche maschinelle Analysetools häufig mit hohen Lizenzgebühren verbunden sind. Aufgrund datenschutzrechtlicher Vorgaben



wurde entschieden, die Keycloak-Logs selbst zu generieren, wie im entsprechenden Kapitel näher erläutert wird.

## 1.6 User (and Entity) Behavior Analytics (U(E)BA)

UEBA ist ein ganzheitlicher Ansatz zur Gewährleistung einer erstklassigen Sicherheit in einem Unternehmen und zur Erkennung von Benutzern, die eine Sicherheitsverletzung im System verursachen könnten. UEBA kann normales und abnormales Verhalten sowohl von Menschen als auch von Computern identifizieren <sup>4</sup> Die Analyse erfolgt dabei häufig unter Einsatz maschineller Lernverfahren.

User and Entity Behavior Analytics (UEBA) stellt eine Weiterentwicklung von User Behavior Analytics (UBA) dar. Während UBA sich ausschließlich auf das Verhalten von Benutzern konzentriert, berücksichtigt UEBA zusätzlich auch Entitäten wie Server, Anwendungen oder Geräte sowie deren Interaktionen mit den Nutzern.

Da sich diese Arbeit auf Keycloak und das Benutzerverhalten im Rahmen von Authentifizierungsprozessen konzentriert, erfolgt keine vollständige Berücksichtigung aller Entitäten oder ihrer Netzwerkbeziehungen. Der Fokus liegt daher auf der Anwendung klassischer UBA-Methoden.

Zur Echtzeitüberwachung werden dabei unter anderem Logdaten analysiert, um anhand zusammenhängender Informationen anomales Benutzerverhalten zu identifizieren. Im Rahmen dieser Arbeit werden insbesondere die folgenden UEBA-Techniken verwendet:

- Zeitreihenanalyse,
- Anomalieerkennung,
- maschinelles Lernen,
- Clusteranalyse.

Ziel ist es vor allem, sicherheitsrelevante Abweichungen zu erkennen – insbesondere solche, die auf Insider-Bedrohungen oder gezielte Angriffe hindeuten.

## 2 Forschungsstand, Herausforderungen und Hypothesen

### 2.1 Bisheriger Forschungsstand

Keycloak bietet, wie bereits erwähnt, regelbasierte Sicherheitsmechanismen an, die jedoch ausschließlich auf vordefinierten Bedingungen beruhen. Dadurch können sie lediglich bekannte Muster erkennen und sind wenig flexibel gegenüber neuen oder komplexeren Angriffstechniken [4]. Angreifende nutzen gezielt sogenannte *Adversarial Attacks* <sup>5</sup>, um diese vorhersehbaren Schutzmechanismen zu umgehen. Da die Regeln dieser Systeme häufig offen dokumentiert sind, lassen sich entsprechende Gegenstrategien leicht entwickeln.

Ein weiteres Beispiel ist der integrierte Brute-Force-Schutz von Keycloak, der jedoch nachweislich umgangen werden kann – etwa durch sogenannte *Timing-Angriffe* <sup>6</sup>. Dabei analysiert ein Angreifer die Antwortzeit des Systems, um Rückschlüsse auf den Authentifizierungsprozess zu ziehen. Wird beispielsweise ein korrekter Benutzername, aber ein (fast) korrektes Passwort eingegeben, kann die Antwortzeit minimal verlängert sein – was auf die sequentielle Überprüfung der Passwortzeichen hindeutet. Solche zeitlichen Unterschiede im Millisekundenbereich lassen sich mit ausreichend vielen Versuchen systematisch auswerten.

Obwohl für viele Schwachstellen bereits Gegenmaßnahmen existieren, erfordert deren Erkennung und Behebung stets Zeit. Neue Verwundbarkeiten entstehen kontinuierlich – etwa durch das Umgehen von Zertifikatsprüfungen <sup>7</sup>, was jedoch primär netzwerkbasierte Sicherheitslücken betrifft und damit außerhalb des Fokus dieser Arbeit liegt. Diese Beispiele verdeutlichen die aktuellen Grenzen der in Keycloak implementierten Schutzmechanismen.

Verhaltensbasierte Anomalien hingegen – etwa verdächtige Login-Zeiten oder Zugriffe auf untypische Ressourcen – sind bislang kaum erforscht. Dabei spielen sie eine zunehmende Rolle, insbesondere zur Erkennung von *Insider Threats*, die im Keycloak-Kontext bisher nicht aktiv adressiert oder dokumentiert sind. Die Erforschung dieser Lücken ist sowohl aus wissenschaftlicher als auch wirtschaftlicher Sicht lohnenswert. Insbesondere bei SSO-Systemen wie Keycloak fehlt es derzeit an quantifizierbaren Metriken, um etwa den Kompromittierungsstatus eines Systems systematisch zu erfassen. Audit-Logs werden oft unzureichend überwacht, wodurch legitime Sitzungen leicht kompromittiert werden können [5]. Zwar existieren kommerzielle Tools zur Unterstützung der Log-Analyse, jedoch werfen diese insbesondere im Hinblick auf

Datenschutz und Ethik neue Herausforderungen auf.

Die obige Darstellung zeigt eine vereinfachte, aber repräsentative Einschätzung des aktuellen Sicherheitsniveaus in Keycloak. Im folgenden Abschnitt wird der Stand der Forschung hinsichtlich maschineller Lernverfahren zur Anomalieerkennung im Benutzerverhalten dargelegt.

Die Literatur legt nahe, dass klassische Verfahren wie One-Class SVM (OC-SVM) und Isolation Forest besonders durch ihre Robustheit und gute Performance auf tabellarischen Daten überzeugen [6], wobei OCSVMs sogar Anomalien präziser erkennen können als Autoencoder [7]. Für sequenzielle Daten wie Zeitreihen oder Nutzerinteraktionen zeigen hingegen neuronale Netze wie Long Short-Term Memory (LSTM) und Autoencoder signifikante Vorteile, da sie zeitliche Abhängigkeiten und komplexe Muster besser erfassen können [8]. Studien wie [9] zeigen, dass die Kombination von LSTM-basierten Architekturen mit OC-SVM oder SVDD <sup>8</sup> zu signifikant verbesserten Erkennungsraten bei der Anomalieerkennung führt – insbesondere bei sequenziellen Daten. Andere Arbeiten wie [10] untersuchen ähnliche hybride Ansätze, etwa in Verbindung mit Isolation Forest, und berichten neben Leistungsgewinnen auch über eine mögliche Reduktion der Rechenzeit. Auch DBSCAN wird erfolgreich zur Cluster-basierten Anomalieerkennung eingesetzt, insbesondere bei unstrukturierten oder verrauschten Daten. Insgesamt zeigt sich, dass der domänenspezifische, kombinierte Einsatz dieser Verfahren besonders vielversprechend ist, um die Herausforderungen bei der Erkennung verhaltensbasierter Anomalien in realen Anwendungen effizient zu bewältigen.

## 2.2 Problematik und Begrenzung von Modellen

Die Literatur zeigt jedoch auch potenzielle Schwächen der genannten Netzwerke und Modelle. LSTM und Autoencoder als Hybridmodell benötigen große Datenmengen und sind anfällig für Overfitting. Zudem sind sie schwer interpretierbar. Auch Isolation Forest ist oft sensitiv gegenüber der Wahl der Parameter [11]. DBSCAN ist beispielsweise sehr empfindlich gegenüber der Wahl von Parametern wie Eps und MinPts, erkennt Cluster mit unterschiedlicher Dichte nur schlecht und kann bei hohem Rauschen versagen. Auch OCSVM ist dem gegenüber sensible bei der Wahl der Parameter.

Auch Hybridmodelle haben ihre Schwächen: Zwar verbessern sie häufig die Leistung, gleichzeitig erhöhen sie jedoch die Komplexität, was die Interpretierbarkeit erschwert. Zudem wird das Training aufwendiger, was den Einsatz in Echtzeitanwendungen erschwert.

## 2.3 Hypothese

Aus der Literatur ergibt sich, dass Kombinationen aus LSTM und Autoencoder in zahlreichen Studien als besonders robust und effektiv bewertet wurden – etwa in Szenarien zur Luftqualitätsüberwachung (99,5 % Genauigkeit) [12]. Andere Studien zeigen, dass sich LSTM-AE mit IF ein gutes Modell bauen lässt, welches Anomalien erkennen kann und ebenfalls eine hohe Accuracy erzielt [13].

Basierend auf der Literatur besteht daher Interesse an einem umfassenden Vergleich zwischen Hybridmodellen aus LSTM-AE und den weiteren genannten klassischen maschinellen Lernalgorithmen sowie an einem Vergleich dieser Modelle einzeln.

Daraus wurde folgende Hypothese abgeleitet:

*Hybridmodelle zur Anomalieerkennung, die LSTM-Autoencoder mit klassischen Verfahren kombinieren, zeigen auf dem Keycloak-Datensatz eine bessere Erkennungsgenauigkeit, Robustheit und Konsistenz als andere Hybridmodelle.*

Die zugehörige Nullhypothese lautet:

*Es gibt keinen signifikanten Unterschied in Erkennungsgenauigkeit, Robustheit und Konsistenz zwischen verschiedenen Hybridmodellen, die auf LSTM-Autoencodern kombiniert mit klassischen Algorithmen basieren, beim Einsatz auf dem Keycloak-Datensatz.*

Ziel dieser Arbeit ist es, die Leistungsfähigkeit verschiedener unüberwachter Anomalieerkennungsverfahren, insbesondere Hybridmodelle aus LSTM-Autoencodern kombiniert mit klassischen Algorithmen, anhand eines geeigneten Keycloak-Datensatzes zu evaluieren.

Folgende Hybridmodelle werden implementiert:

- LSTM-AE und Isolation Forest
- LSTM-AE und One-Class SVM
- LSTM-AE und DBSCAN

Mit verschiedenen Metriken soll geprüft werden, welches Modell hinsichtlich Klassifikationsgenauigkeit, geringster False-Positive-Rate und Konsistenz die besten Ergebnisse erzielt. Unter Konsistenz wird verstanden, welches Modell nicht zufällig, sondern anhand eines stabilen Algorithmus klassifiziert.

Die Kombination von Hybridmodellen mit zwei verbundenen Netzwerken und drei klassischen Lernalgorithmen bietet einen interessanten Vergleich, auch wenn der

Vergleich zwischen den Modellen als unausgeglichen wahrgenommen werden kann. Es wird jeweils das Hybridmodell LSTM-AE mit einem der drei Lernalgorithmen kombiniert und miteinander verglichen. Obwohl die komplexeren Hybridmodelle aufgrund ihrer Leistungsfähigkeit voraussichtlich besser abschneiden, ist dies genau der Forschungsschwerpunkt: Nicht die absolute Überlegenheit steht im Fokus, sondern die quantitative Bewertung des Mehrwerts tief ergehender Hybridarchitekturen gegenüber klassischen Anomalieerkennungsverfahren.

Tran et al. zeigen beispielsweise bereits, dass ein Vergleich von LSTM-AE mit weiteren Lernalgorithmen erfolgte und dass das Hybridmodell LSTM-AE bessere Ergebnisse erzielte [14]. Daher erscheint ein erweiterter Vergleich zwischen komplexeren Hybridmodellen, die noch klassische Lernalgorithmen verwenden, und den einfachen Algorithmen sinnvoll. Zudem kann so geprüft werden, ob komplexere Architekturen eventuell auch Nachteile haben und einfache Modelle unter bestimmten Bedingungen überlegen sind.

Zudem werden komplexere Daten verwendet, nämlich Logdaten, die Aufschluss über das Benutzerverhalten geben. Auch wenn eine Studie zeigt, dass Isolation Forest hier gute Ergebnisse erzielt [15], gibt es bislang keinen Vergleich mit Hybridmodellen.

Der Vergleich trägt somit dazu bei, bisherige Studien zu ergänzen und einen neuen wissenschaftlichen Mehrwert zu schaffen, indem einfache Modelle mit komplexeren Hybridmodellen verglichen werden.

Abschließend wird erwartet, dass die Hybridmodelle insgesamt effizienter sind als die einfachen, wobei die konkrete Architektur entscheidend für diesen Effekt ist.

## 3 Methoden und Störfaktoren

### 3.1 Metriken

Als Metrik werden die typischen Verfahren wie die Berechnung der Accuracy, der Precision, des Recalls und des F1-Scores <sup>9</sup> verwendet.

Eine Studie weist aber daraufhin, dass das Öffnen dieser Methoden angewandt werden und oft nicht valide Begründungen für die Klassifikationsgenauigkeit des Modells bietet. U.a. wies diese Studie daraufhin, dass ein Großteil, der Studien nur anhand der Accuracy der maschinellen Lernmodelle ihre Effizienz misst und andere Eigenschaften ausblenden [16, 17]. Zudem wird hingewiesen, dass durch stark unausgewogenen Klassen die Accuracy irreführend ist. Dies liegt daran, weil die Accuracy nur angibt, wie viele Datenpunkte richtig klassifiziert wurden. Wenn aber die Klas-

sen unausgeglichen sind (z.B. 99 % der Datenpunkte gehören Klasse A der Rest zu Klasse B), ist es wahrscheinlicher, dass das Modell lernt immer nur entweder sich für Klasse A oder Klasse B zu entscheiden und wählt- da es mehr Objekte von Klasse A gibt- diese dahin zu klassifizieren, obwohl es noch eine kleinere Klasse B gab, die aber nicht berücksichtigt wurde. Es wird dann zwar eine hohe gute Accuracy von 99 % erreicht, was aber bedeutet, dass 1 %, die von Klasse B ebenfalls zu Klasse A zugeordnet wurden. Dies nennt man auch Accuracy Paradoxon. Um dieses Paradoxon zu vermeiden, will man in dieser Arbeit Metriken einsetzen, welche ausschlaggebender sind.

Eine andere wies daraufhin, dass Metriken des Öfteren falsch ausgewertet werden und der Fokus alleine auf den F1-Score liegt. Dieser aber wiederum lieferte nach der Studie nur oberflächliche Ergebnisse lieferte, weil der Score alleine nicht viel aussagte [18]. Micelucci et. Al. kritisieren u.A. am F1-Score, dass dieser zwar eine gute Gewichtung zwischen Precision und Recall zeigt, jedoch an sich nicht genau angeben kann, ob nun wirklich das Modell Probleme in False-Positive-Klassifizierungen oder False-Negative-Klassifizierungen hat.

Es zeigt sich, dass verschiedene Studien ähnliche Probleme feststellen. Eine weitere Untersuchung von Fourure et al. zeigt, dass in vielen Vergleichsstudien vor allem die Klassifikationsgenauigkeit (Accuracy) betrachtet wird, während andere wichtige Metriken oft vernachlässigt bleiben. So wurde beispielsweise häufig der F1-Score als Bewertungsmaß verwendet, der jedoch stark vom Anteil an Anomalien (Kontaminationsrate) im Datensatz abhängt. Durch die Auswahl bestimmter Trainings- und Testdatensätze kann der F1-Score künstlich erhöht werden, was zu einer verzerrten Einschätzung der Modellleistung führt [19]. Fourure und Kollegen empfehlen stattdessen die Verwendung der AUC-Kurve zur Evaluierung.

Insgesamt lässt sich festhalten, dass jedes Modell individuelle Stärken und Schwächen besitzt, die die Klassifikationsgenauigkeit beeinflussen. Gleichzeitig wird kritisiert, dass viele Studien auf Metriken zurückgreifen, die nicht ausreichend aussagekräftig sind. Generell ist die Kritik, dass eine einfacher Prozentsatz nicht ausschlaggebend für die Fähigkeiten des Modells ist, dennoch werden diese als übliche Metriken angewandt. Zusätzlich aber noch, um die Störfaktoren der üblichen Metriken zu verringern, werden modernere Metriken angewandt [18]:

- **Area Under the ROC Curve (AUC-ROC):** Diese Kurve zeigt, wie sich die True-Positive-Rate und False-Positive-Rate bei verschiedenen Schwellenwerten verändern. Für jeden Schwellenwert kann man ablesen, wie viele False Positives bzw. False Negatives entstehen. Die AUC-ROC fasst die Performance über alle

Schwellenwerte zusammen.

- **Area Under the Precision-Recall Curve (AUC-PR:)** Diese Kurve zeigt die Beziehung zwischen Precision und Recall bei verschiedenen Schwellenwerten. Für jeden Schwellenwert misst man neu, wie genau (Precision) und wie vollständig (Recall) die positiven Fälle erkannt werden. So erkennt man, bei welchem Schwellenwert die Balance zwischen Precision und Recall am besten ist.
- **Matthews Correlation Coefficient (Matthews Correlation Coefficient (MCC)):** Eine umfassende Metrik, die alle vier Werte der Verwirrungsmatrix (True Positives, True Negatives, False Positives, False Negatives) berücksichtigt. MCC liefert einen Wert zwischen -1 und 1, wobei 1 perfekte Vorhersage, 0 zufällige Vorhersage und -1 vollständig falsche Vorhersage bedeutet. Besonders geeignet für unausgeglichene Datensätze.
- **Balanced Accuracy:** Der Anteil der negativen Beispiele, die fälschlicherweise als positiv klassifiziert wurden. Sie wird berechnet als 
$$FPR = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$
 und ist wichtig, um die Fehlerquote bei der Erkennung negativer Fälle zu bewerten.

Auch wenn diese Metriken mehr Informationen über die Verarbeitungsweise der Modelle schließen lassen, haben diese ebenfalls Einschränkungen. Davis et Al. haben z.B. erkannt, dass bei kleinen Datensätzen die Punkte in der AUC-ROC-Kurve und in der AUC-PR-Kurve die Ergebnisse sich recht ähnlich sind, was möglicherweise sich darauf zurückzuführen lässt, dass beide Verfahren sich schon im Algorithmus ähnlich sind [20].

Im Vergleich, zeigt sich in Studien, dass MCC und Balanced Accuracy in ihrer Methodik wenige Paradoxe oder Schwächen zeigen. Chicco et Al. zeigen z.B. dass MCC für unausgeglichene Klassen besser geeignet ist und Broderesen et Al. zeigen das Gleiche für die Balanced Accuracy [21, 22].

Außer diesen Metriken wurde noch überlegt, einen sogenannten Cross-Validation-Test durchzuführen. Cross-Validation ist keine „Metrik“ wie Genauigkeit oder F1-Score, sondern eine Methode zur Modellbewertung. Sie unterteilt dabei den Datensatz in  $k$  gleich große Teile. Das Modell wird dann  $k$ -mal trainiert, jedes mal mit  $k-1$  Teilen als Trainingsdaten. Der übrig geblieben Teil wird als Testdatensatz und Validierungsdatensatz verwendet. Der Sinn dahinter ist die Trainingsdaten so zu verteilen, dass das Modell jedes mal neu trainiert wird. Dies soll Overfitting auf

den Daten vermeiden. Durch das Training. Zudem soll sie die Ergebnisse nach jedem Training des k-Datensatzes zeigen, um zu zeigen, ob es an bestimmten Stellen Abweichungen gab, womit sich eben Overfitting erkennen lässt.

### **3.2 Andere, nicht angewandte Metriken**

Im Rahmen der Methodik wurde auch die Durchführung eines ANOVA-Tests (Analysis of Variance) in Betracht gezogen. Der ANOVA-Test prüft, ob sich die Mittelwerte von zwei oder mehr Gruppen signifikant voneinander unterscheiden.

Beispielsweise könnte man untersuchen, ob sich ein bestimmtes Merkmal – wie etwa das Persönlichkeitsmerkmal „Lieblingstier“ – in Abhängigkeit von einer Gruppenzugehörigkeit, etwa der Haarfarbe, signifikant unterscheidet. Dabei zeigt der ANOVA-Test lediglich auf, ob Unterschiede zwischen den Mittelwerten der Gruppen vorliegen, jedoch nicht, wie stark zwei Merkmale miteinander korrelieren.

Der ANOVA-Test wurde im vorliegenden Kontext nicht angewandt, da das Ziel der Arbeit nicht darin besteht, einzelne Merkmale hinsichtlich ihrer Korrelation zu Anomalien zu analysieren. Stattdessen soll untersucht werden, wie gut verschiedene Modelle Anomalien erkennen können und welches Modell dabei die besten Ergebnisse in Bezug auf die zuvor erläuterten Metriken erzielt. Die Verwendung des ANOVA-Tests würde diesen Fokus verfehlen, da er keine Aussage über die Leistungsfähigkeit der Modelle in der Anomalieerkennung trifft.

### **3.3 Störfaktoren**

Entscheidend für einen „fairen“ Vergleich ist unter anderem die Art und Weise, wie die Daten vorverarbeitet werden. Eine ungenaue Behandlung von kategorischen und numerischen Daten kann die Ergebnisse erheblich verfälschen. Zudem werden die Daten je nach Modell unterschiedlich verarbeitet. Während das LSTM-AE-Hybridmodell mehrere Verarbeitungsschritte zwischen den einzelnen Komponenten erfordert (z. B. Datenreinigung und anschließende Umwandlung in Sequenzen), benötigen die einfachen Modelle meist nur einen Verarbeitungsschritt vor dem Training.

Ein weiterer Störfaktor ist die Parameterwahl, auf die im Abschnitt zur Umsetzung noch näher eingegangen wird. Parameter wie die Anzahl der Epochen, die Batch-Größe und die Sequenzlänge beeinflussen maßgeblich das Verhalten der Modelle. Da die Modelle unterschiedlich funktionieren, müssen die Parameter individuell angepasst werden. Dadurch ist ein absolut „fairer“ Vergleich schwierig, da die Parameter



naturgemäß variieren.

Ebenso entscheidend sind die zu verarbeitenden Daten. Da diese teilweise synthetisch generiert werden, bestimmt der Anteil der als anomal gekennzeichneten Zeilen die Bewertung der Modelle. Wie bereits im Abschnitt zu den Metriken erwähnt, führt eine starke Klassenungleichheit häufig zum Accuracy-Paradoxon. In der Regel sind Anomalien in Datensätzen selten, sodass viele Modelle zunächst anomale Daten als normal klassifizieren. Moderne Metriken helfen, diesen Störeffekt abzufedern. Dennoch ist es wichtig, die Ungleichverteilung der Datensätze zu beachten.

Darüber hinaus ist es essentiell, die Daten in Trainings-, Test- und Validierungsdaten aufzuteilen, um frühzeitig *Overfitting*<sup>10</sup> erkennen zu können.

Auch die Zufälligkeit der Daten spielt eine Rolle bei der Modellverarbeitung. Die Datensätze müssen zufällig generiert sein, gleichzeitig aber einen kontinuierlichen Zusammenhang besitzen – wie es bei realen Cyberangriffen der Fall ist. Dies ist besonders wichtig für das LSTM, das auf sequenzielle Abhängigkeiten angewiesen ist. Kontextlose Datenpunkte erschweren die Analyse.

Zudem beeinflusst die verfügbare Rechenleistung die Verarbeitungsgenauigkeit der Modelle. Zu lange oder zu kurze Trainingszeiten sowie hohe Rechenintensität wirken sich auf die Leistungsfähigkeit aus. Deshalb werden in den Modellen Regulierungstechniken implementiert, um eine optimale Balance zu gewährleisten.

Da die Generierung synthetischer Logs Herausforderungen mit sich bringt, wird ebenfalls in Betracht gezogen, echte Daten zu verwenden. Diese können z. B. durch automatisierte Tests (auf die im weiteren Verlauf der Arbeit eingegangen wird) generiert werden. Auch firmeninterne Daten wurden erwogen.

Da solche realen Daten häufig begrenzt verfügbar sind, stehen meist nur wenige Trainings- und Testdaten zur Verfügung. LSTMs gelten als „datenhungrig“, da sie für die Analyse hochkomplexer zeitbasierter Daten entwickelt wurden und typischerweise eine große Menge an Logs benötigen. Ein zu kleiner Datensatz könnte daher zu einer Unterforderung des Modells führen, was die Ergebnisse erheblich beeinflussen kann.

Beim Vergleich der Modellleistungen mit synthetischen und echten Daten als Input wird erwartet, dass sich die Ergebnisse deutlich unterscheiden, da die Feature-Struktur und die logische Beschaffenheit der Logs pro Datensatz variieren.

## 4 Anforderungsdefinition und Analyse

### 4.1 Funktionale Anforderungen

Da in dieser Arbeit mehrere Modelle zur Anomalieerkennung auf Basis von Keycloak-Logdaten untersucht werden, ergeben sich folgende funktionale Anforderungen an das System:

- Die Logdaten sollen in einem geeigneten Format (z.,B. JSON) eingelesen und vorverarbeitet werden.
- Es sollen mindestens drei Hybridmodelle angewendet werden.
- Die Verfahren sollen auf den rekonstruierten Fehlerwerten basieren und eine Klassifikation in normal/anomal ermöglichen.
- Die Vorhersagen sollen mit tatsächlichen Anomalien verglichen und quantitativ bewertet werden (Precision, Recall, F1-Score, Balanced Accuracy, MCC).

### 4.2 Nicht-funktionale Anforderungen

- Das System soll modular aufgebaut sein, sodass einzelne Modelle unabhängig getestet werden können.
- Die Reproduzierbarkeit der Experimente soll durch fixierte Random Seeds gewährleistet sein, wie bspw. bei den Keycloak-Log-Daten.
- Das gesamte Setup soll in einer isolierten Entwicklungsumgebung (z.,B. Jupyter Notebook) lauffähig dokumentiert sein.

## 5 Angewandte Technologien

### 5.1 Scikit-learn Learn

Scikit-learn<sup>11</sup> ist eine weitverbreitete Open-Source-Bibliothek für maschinelles Lernen in Python. Sie stellt eine Vielzahl von Werkzeugen für klassische Machine-Learning-Aufgaben bereit, darunter Klassifikation, Regression, Clustering, Dimensionsreduktion sowie Modellbewertung und -selektion. Die Bibliothek ist insbesondere aufgrund ihrer einfachen API, klaren Struktur und umfassenden Dokumentation sowohl für Einsteiger als auch für fortgeschrittene Anwendungen geeignet. Scikit-learn bietet eine Reihe vorimplementierter Modelle wie *Isolation Forest*, *DBSCAN* und *One-Class SVM*.

## 5.2 Deep GPU Xceleration (DGX)

Eine DGX ist ein Hochleistungsrechnersystem von NVIDIA<sup>12</sup>. Es eignet sich insbesondere für das effiziente Training rechenintensiver Modelle des maschinellen Lernens, wie beispielsweise LSTM-Netze. Die DHBW stellt hierfür das Modell DGX H100 zur Verfügung, auf dem die Trainingsprozesse ausgeführt wurden. NVIDIA DGX H100-Systeme sind mit zwei Intel Xeon 8480C-Prozessoren ausgestattet, die gemeinsam über insgesamt 112 CPU-Kerne verfügen<sup>13</sup>.

Der Einsatz der DGX ist erforderlich, da herkömmliche GPU-basierte Systeme nicht über die nötige Rechenkapazität und Verarbeitungsgeschwindigkeit verfügen. Im weiteren Entwicklungsverlauf wurde jedoch vermehrt CPU-basiert gearbeitet, weshalb auf ein alternatives System umgestiegen wurde.

## 6 Angewandte Technologie: Keycloak

Keycloak ist ein IAM-Tool mit SSO-Funktion. Es bietet an, Benutzerdaten und Firmendaten sicher und strukturiert zu speichern. Jedes mal, wenn man sich über Keycloak bei einem Dienstleister anmeldet, so wird durch SSO die Anmelden- und Autorisierungsdaten an diesen weitergegeben. Keycloaks Struktur beinhaltet Bereiche, die auch Realms genannt werden, in denen sogenannte Clients angelegt werden können. Die Clients stellen die jeweiligen Anwendungen dar, die u.A. Benutzerdaten und Daten von Gruppen beinhalten.

### 6.1 Terminologien in Keycloak

Keycloak ist ein auf Rollen basiertes Identitäts- und Zugriffsmanagementsystem (RBAC). Für ein besseres Verständnis der zentralen Begriffe und Abläufe wird in diesem Abschnitt ein Überblick über die wichtigsten Terminologien und Konzepte gegeben.

#### **Realms und Clients:**

Keycloak organisiert Anwendungen in sogenannten *Realms*. Ein Realm ist eine isolierte Umgebung, in der Benutzer, Rollen und Clients verwaltet werden. Innerhalb eines Realms können sich mehrere *Clients* befinden – dies sind Anwendungen oder Dienste, die Keycloak für Authentifizierung und Autorisierung nutzen.

Standardmäßig existiert ein sogenannter *Master-Realm*, der übergeordnete Verwaltungsrechte besitzt. Er enthält den *Admin-User*, der Zugriff auf alle anderen Realms und deren Komponenten hat. Der Admin kann CRUD-Operationen (Create, Read, Update, Delete) auf Benutzer, Gruppen und Clients ausführen. Normale

Benutzer hingegen sehen nur den eigenen Realm und haben nur eingeschränkte Rechte, sofern ihnen keine speziellen Rollen zugewiesen wurden.

### **Authentifizierung über Clients und Identity Provider (IdP) (IdP):**

Benutzer können sich entweder direkt bei Keycloak oder über einen Client anmelden. Die Authentifizierung erfolgt meist über einen externen *Identity Provider* (IdP), wie z. B. Google oder Microsoft. Dieses Verfahren ermöglicht *Single Sign-On* (SSO): Der Benutzer meldet sich bei einem Drittanbieter an und ist gleichzeitig auch bei Keycloak und weiteren Diensten authentifiziert.

Für die Kommunikation zwischen IdPs und Keycloak kommen verschiedene Protokolle zum Einsatz:

- **OpenID Connect (OIDC) (OIDC):** Modernes Authentifizierungsprotokoll basierend auf OAuth 2.0. Es ermöglicht, Benutzeridentität und Profildaten sicher abzurufen.
- **SAML:** XML-basiertes, etabliertes Standardprotokoll für Authentifizierungs- und Autorisierungsdaten.
- **OAuth 2.0:** Framework zur Autorisierung, auf dem u. a. OIDC aufbaut.

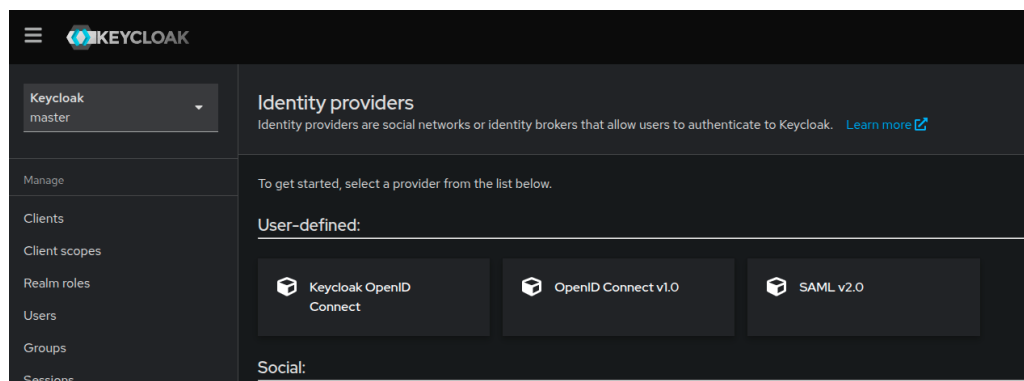


Abbildung 1: Protokolle für IdPs in Keycloak

### **Grant Typen und Authentifizierungsflüsse:**

Keycloak unterstützt verschiedene *Grant Types*, die beschreiben, wie sich Benutzer oder Clients authentifizieren:

- **Authorization Code Grant:** Benutzer wird zur Keycloak-Loginseite weitergeleitet. Nach erfolgreicher Authentifizierung erhält der Client einen Authorization Code, den er gegen ein Access Token eintauscht. Das Passwort wird dabei nie an die Client-Anwendung weitergegeben.

- **Client Credentials Grant:** Wird für Clients ohne Benutzerinteraktion verwendet. Der Client authentifiziert sich mit seiner Client-ID und einem geheimen Schlüssel (Client Secret), um ein Access Token zu erhalten.

Die **Client Authentifizierungsmethode** legt fest, wie der Client gegenüber dem Server seine Identität nachweist – z. B. mittels Client Secret oder Zertifikat. Während der Grant-Typ die Art der Anmeldung bestimmt, definiert die Client-Authentifizierungsmethode die technische Form des Identitätsnachweises.

Zusätzlich existieren in Keycloak sogenannte *Authentication Flows*, die den Ablauf der Authentifizierung steuern:

- **Standard Flow:** Meist Authorization Code Grant mit Weiterleitung zur Login-Seite.
- **Implicit Flow:** Gibt direkt ein Token zurück, wird aber aus Sicherheitsgründen nicht mehr empfohlen.
- **Direct Access Grant:** Direkte Anmeldung über Benutzername und Passwort, z. B. bei Skripten oder mobilen Apps.
- **Device Authorization Grant:** Für Geräte ohne Eingabemöglichkeit, z. B. Smart TVs.
- **CIBA (Client-Initiated Backchannel Authentication):** Asynchrone Authentifizierung, z. B. über Push-Benachrichtigungen.

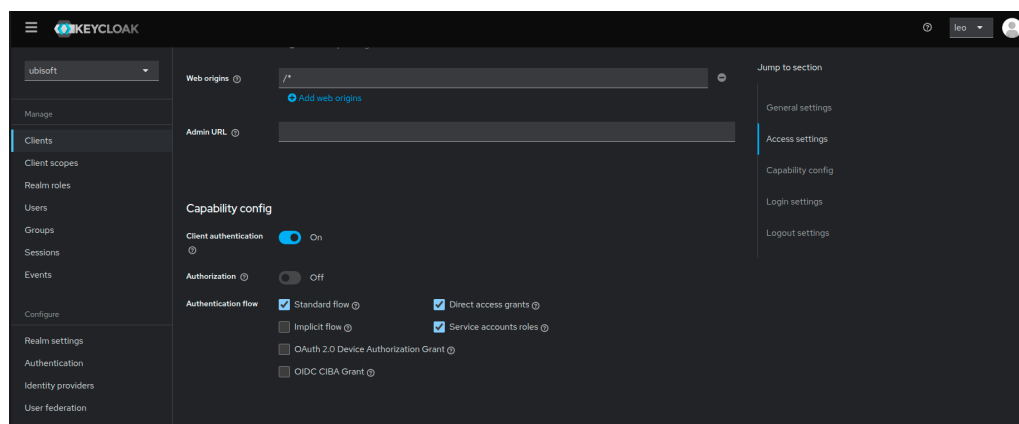


Abbildung 2: Beispielhafte Konfiguration von Authentication Flows in Keycloak

### Client-Scope:

Der *Client-Scope* definiert, welche Benutzerdaten in das Token aufgenommen werden, wenn sich ein Benutzer über einen Client anmeldet. Typische Informationen

sind Name, Nachname und E-Mail. Diese Daten werden verschlüsselt übertragen, jedoch besteht ein gewisses Risiko bei Missbrauch oder Diebstahl des Tokens.

#### **Rollenmodell:**

Keycloak unterscheidet zwei zentrale Rollentypen:

- **Realm-Rollen:** Gelten global innerhalb eines Realms. Sie eignen sich für übergreifende Rechte, z. B. „admin“ für alle Clients.
- **Client-Rollen:** Spezifisch für einen einzelnen Client. Sie erlauben feinere Zugriffskontrolle innerhalb der jeweiligen Anwendung.

#### **Service Account Roles:**

Clients können sich in Keycloak auch selbst authentifizieren – ohne Benutzerinteraktion. Wird die *Service Account*-Funktion für einen Client aktiviert, erhält dieser ein eigenes Servicekonto. Dieses Konto kann mit Rollen ausgestattet werden, um Rechte auf bestimmte Ressourcen oder Operationen zu erhalten – z. B. für automatisierte Backend-Operationen.

## **6.2 Komponenten der Log-Erzeugung**

Keycloak generiert Logs über eine Kombination aus verschiedenen Frameworks und Komponenten. Grundlage bildet das Logging-Framework von JBoss, wobei moderne Versionen von Keycloak zusätzlich auf das Quarkus-Framework setzen. **Von Wild-**

#### **Fly zu Quarkus – Technologiewechsel:**

Ursprünglich basierte Keycloak auf dem **WildFly**-Anwendungsserver<sup>14</sup>, der wiederum die **JBoss Logging**-Infrastruktur verwendet. Ab Keycloak Version 17 wurde WildFly durch das **Quarkus**-Framework ersetzt – ein modernes Java-Framework, das speziell für containerisierte und cloud-native Anwendungen entwickelt wurde. Quarkus unterstützt ebenfalls JBoss Logging, sodass bestehende Logging-Mechanismen weiterhin funktionieren.

```

Tests > test.txt
1 # Logs im JSON-Format in die Konsole und eine Datei schreiben
2 quarkus.log.console.format=json
3 quarkus.log.console.output=stdout
4 quarkus.log.file.enable=true
5 quarkus.log.file.path=/home/ueay/keycloak-logs/keycloak.log
6 quarkus.log.level=DEBUG
7
8 # Event-Listener-Logging (optional)
9 spi-events-listener-jboss-logging-success-level=info
10 spi-events-listener-jboss-logging-error-level=warn
11

```

Abbildung 3: Beispiel: Aktivierung von Quarkus-Logging in der Keycloak-Konfiguration

### Rolle von JBoss Logging:

JBoss Logging dient in Keycloak als zentrale Logging-API. Es abstrahiert verschiedene Logging-Backends (wie Log4j, JUL, SLF4J) und ermöglicht die strukturierte Ausgabe sicherheitsrelevanter Informationen – z. B.:

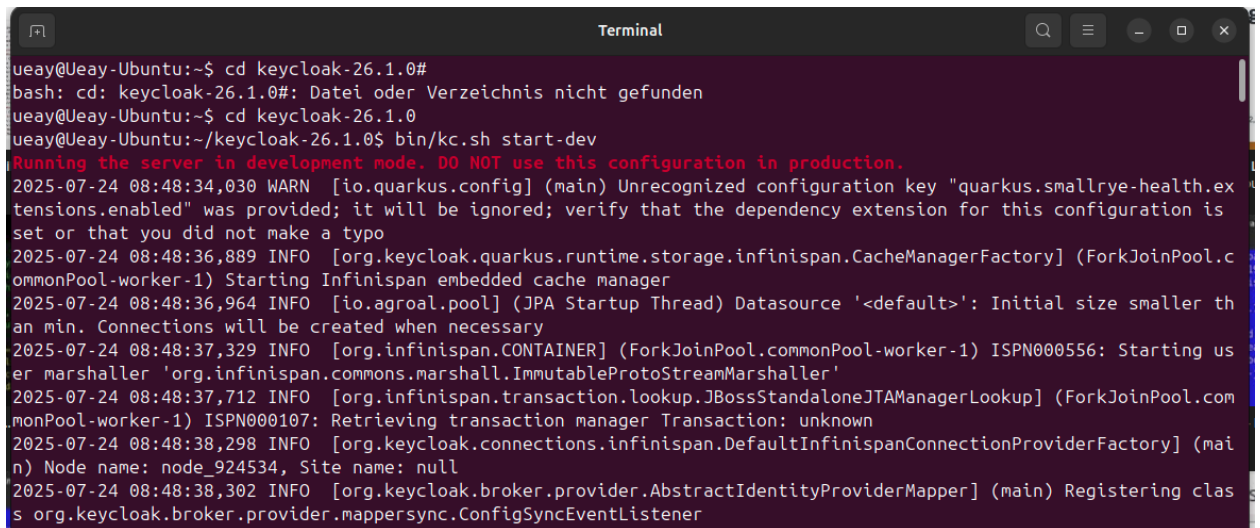
- Realm-ID
- Benutzer-ID
- IP-Adresse
- Event-Typ

Der Vorteil liegt in der Flexibilität: Entwickler müssen sich nicht auf ein spezifisches Logging-Backend festlegen, sondern können über die einheitliche JBoss-API verschiedene Ausgabekanäle konfigurieren (Konsole, Datei, Syslog etc.).

### Warum nicht Log4j, Logback oder JUL?

Obwohl es im Java-Ökosystem viele Logging-Frameworks gibt, verzichtet Keycloak bewusst auf deren direkte Verwendung:

- **Log4j / Logback:** Mächtig, aber externe Abhängigkeiten und teils Sicherheitsrisiken (z. B. Log4Shell).
- **JUL (java.util.logging):** Eingebaut, aber funktional begrenzt.
- **Slf4j:** Nur eine Fassade – erfordert Backend wie Logback oder Log4j.
- **System.out / System.err / System.err:** Nicht strukturiert, ungeeignet für produktive Logs.



```
ueay@Ueay-Ubuntu:~$ cd keycloak-26.1.0#
bash: cd: keycloak-26.1.0#: Datei oder Verzeichnis nicht gefunden
ueay@Ueay-Ubuntu:~$ cd keycloak-26.1.0
ueay@Ueay-Ubuntu:~/keycloak-26.1.0$ bin/kc.sh start-dev
Running the server in development mode. DO NOT use this configuration in production.
2025-07-24 08:48:34,030 WARN [io.quarkus.config] (main) Unrecognized configuration key "quarkus.smallrye-health.extensions.enabled" was provided; it will be ignored; verify that the dependency extension for this configuration is set or that you did not make a typo
2025-07-24 08:48:36,889 INFO [org.keycloak.quarkus.runtime.storage.infinispan.CacheManagerFactory] (ForkJoinPool.commonPool-worker-1) Starting Infinispan embedded cache manager
2025-07-24 08:48:36,964 INFO [io.agroal.pool] (JPA Startup Thread) Datasource '<default>': Initial size smaller than min. Connections will be created when necessary
2025-07-24 08:48:37,329 INFO [org.infinispan.CONTAINER] (ForkJoinPool.commonPool-worker-1) ISPN000556: Starting user marshaller 'org.infinispan.commons.marshall.ImmutableProtoStreamMarshaller'
2025-07-24 08:48:37,712 INFO [org.infinispan.transaction.lookup.JBossStandaloneJTAManagerLookup] (ForkJoinPool.commonPool-worker-1) ISPN000107: Retrieving transaction manager Transaction: unknown
2025-07-24 08:48:38,298 INFO [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) Node name: node_924534, Site name: null
2025-07-24 08:48:38,302 INFO [org.keycloak.broker.provider.AbstractIdentityProviderMapper] (main) Registering class org.keycloak.broker.provider.mappersync.ConfigSyncEventListener
```

Abbildung 4: Log-Ausgaben, wie sie in einer Quarkus-basierten Keycloak-Instanz erzeugt werden

### Vorteile der aktuellen Logging-Architektur:

- **Nahtlose Integration:** JBoss Logging ist direkt in Quarkus und die Keycloak-Architektur eingebettet.
- **Sicherheitskontext:** Strukturierte Logs enthalten kontextuelle Daten, die für Auditing und Sicherheitsanalysen wichtig sind.
- **Zentrale Konfiguration:** Log-Level, Ausgabeformate und Ziele können zentral über Konfigurationsdateien gesteuert werden.
- **Konsistenz:** Einheitliches Logging-Verhalten über alle Komponenten hinweg.

Insgesamt stellt JBoss Logging in Kombination mit Quarkus eine robuste, flexible und erweiterbare Logging-Lösung dar, die sowohl Sicherheitsanforderungen als auch Betriebsanforderungen gerecht wird. Ein weiterer Baustein ist SLF4J (Simple Logging Facade for Java), das vor allem in vielen externen Bibliotheken der Java-Welt als Logging-Fassade genutzt wird. Quarkus unterstützt SLF4J indirekt, indem es mit einem passenden Backend wie logback oder log4j2 kombiniert werden kann. In Keycloak selbst wird SLF4J jedoch nicht direkt verwendet, da stattdessen vollständig auf JBoss Logging gesetzt wird. Für erweiterte oder spezielle Logging-Anforderungen lassen sich Frameworks wie Log4j2 oder Logback über Custom Extensions oder SPIs einbinden. Diese sind nicht standardmäßig in Keycloak aktiv,



können aber in Eigenentwicklungen verwendet werden – etwa wenn besondere Formatierungen oder externe Integrationen erforderlich sind. Einfache Ausgaben über `System.out` oder `System.err` kommen in Keycloak kaum zum Einsatz und dienen allenfalls zu Debug-Zwecken in sehr frühen Initialisierungsphasen. Für produktive Umgebungen sind sie ungeeignet, da sie weder strukturiert noch steuerbar sind und nicht in zentrale Log-Management-Systeme integriert werden können. Besonders nützlich ist die Möglichkeit, JSON-basiertes Logging in Quarkus zu aktivieren. Dadurch lassen sich strukturierte Logdaten erzeugen, die sich ideal für den Einsatz in zentralisierten Logging-Stacks wie ELK (Elasticsearch, Logstash, Kibana) oder Loki mit Grafana eignen. Die JSON-Ausgabe wird direkt über eine Property aktiviert (`quarkus.log.console.json=true`). Neben dem klassischen System-Logging bietet Keycloak ein eigenes Audit- und Event-Logging-Subsystem, das unabhängig vom allgemeinen Logsystem funktioniert. Mithilfe des `EventListener-SPI` können verschiedene Ereignisse – etwa Admin-Änderungen oder Benutzeraktionen – gezielt erfasst werden. Die Ausgabe kann dabei in Form einfacher Logs, in Syslog-Formate oder in benutzerdefinierte Kanäle erfolgen. Abschließend unterstützt Keycloak die Integration in externe Logging-Systeme durch Syslog oder Remote Logging. So lassen sich Logs über Tools wie Filebeat an zentrale Systeme weiterleiten (z.B. Logstash oder Cloud-Dienste wie AWS CloudWatch, Google Stackdriver oder Azure Monitor) oder Kubernetes. Dies ist insbesondere in containerisierten oder skalierenden Cloud-Umgebungen entscheidend für Monitoring, Fehleranalyse und Sicherheitsüberwachung.

### 6.3 Log-Dateien in Keycloak

Logging, bzw. Protokollierung ist ein essentieller Bestandteil für die Überwachung von IT-Systemen. Für das IT-Monitoring dienen Logs zur Verhinderung und Detektion von Anomalien, sei es im Netzwerk, in einem IoT-Gerät oder eben in IAM-Tools. In Keycloak werden verschiedene Log-Arten unterteilt, wobei die Logs für das Netzwerk, die Access Logs u.A. nicht direkt zu Keycloak gehören. Keycloak unterteilt Logs in folgende Kategorien:

- Server-Logs: Betriebsstatus, Fehler, Debugging-Infos
- Event-Logs: Benutzer- und Admin-Events (Event-System)
- Audit-Logs: Nachvollziehbarkeit von Änderungen (Admin Events)
- Access Logs: Zugriffsversuche und HTTP-Statuscodes (HTTP Layer / Reverse Proxy)

- Security Logs: Sicherheitsrelevante Ereignisse und Warnungen
- Custom Logs: Erweiterungen und Plugins können eigene Logs erzeugen (SPI)

Die Logs werden dabei von unterschiedlichen Instanzen in Keycloak erzeugt. Die Server-Logs werden vom Keycloak-Server selbst erzeugt und weisen die Kategorie *org.keycloak.services* auf. Sie werden mit dem Quarkus-Framework erzeugt.

Event-Logs und Audit Logs werden vom Event-System in Keycloak erzeugt, der unter der Kategorie *org.keycloak.events* zu erkennen ist.

Access Logs werden auf der Anwendungs- und Netzebene erzeugt- hauptsächlich außerhalb Keycloaks und sind gekennzeichnet als *io.quarkus.http*

## 6.4 Event-Logs

Analysiert in dieser Arbeit werden die Event-Logs. Grund hierfür ist, dass diese wichtige Features enthalten, welche das Benutzerverhalten beschreiben. Server-Logs beziehen sich zu sehr auf Netzbasierte Logs und andere Logs, wie Audit-Logs sind Unterkategorien von Events-Logs. Da keine SPI verwendet wird und nur die einzelne Keycloak-Instanz untersucht wird, fallen Custom Logs ebenfalls weg als Teil der Trainingsdaten und Security Logs werden nur erzeugt, wenn es die dazu gehörigen Komponenten dazu gibt, welche hier auch nicht angewandt werden.

Keycloak unterteilt unter Anderem zwei Arten von Events-Logs:

- Admin Logs(Audit Logs)
- User Logs (Event Logs)

Die Admin Logs enthalten alle Logs bzgl. den Operationen des Admins und die User Logs nur die Logs von Benutzern, die nicht Admin sein.

## 6.5 Aufbau der Event-Logs

Folgende Features sind stets in den User Event Logs enthalten:

- timestamp
- log\_level

- category
- type
- ipAddress
- realmName
- realmId
- clientId
- userId

Man erkennt, dass wichtige Attribute, wie bspw. der Standort des Clients fehlen. Dies liegt an datenschutzrechtlichen Gründen, jedoch erschwert dies Anomalien in den Logs zu erkennen, weil diese anhand des Standortes erkannt werden. Dies muss noch als Störfaktor erkannt werden. Anhand des Standortes können somit schon mal keine Anomalien erkannt werden. Timestamp ist einer der wichtigen Attribute, welche zeigen, um welche Zeit der Log erzeugt wurde. Zur Analyse zeitbasierter Daten eignen sich timestamp als aussagekräftige Information für die LSTM-AE-Hybridmodelle.

Desweiteren unterteilt Keycloak noch sogenannte Log-Level. Diese zeigen, ob eine bestimmte Operation erfolgreich war oder nicht oder geben andere Informationen bzgl. einer Operation. Bekannte Log-Level sind z.B. INFO, FATAL oder ERROR. Alle Arten finden sich in der Dokumentation <sup>15</sup>. Das Log-Level (z.B. FATAL, ERROR, INFO) bestimmt die Art des Logs, z.B. ob es zum Debugging dient oder eine Warnung erscheint, entweder ein Fehler oder eine Info.

”Category (Logging)” beschreibt hier die Art des Logs. Da in dieser Arbeit mit Keycloak-Logs gearbeitet wird, kommen insbesondere der JBossLogger <sup>16</sup> und die JPA-Komponente <sup>17</sup> von Keycloak zum Einsatz.

Des weiteren von Bedeutung in den Logs haben die Ereignis-Typen: Bspw. ist LOGIN ein Ereignis-Typ bei dem der Benutzer sich anmeldet. Dies wird in Keycloak auch regelmäßig dokumentiert. Andere Ereignistypen sind bspw. TOKEN\_REFRESH, bei dem der Benutzer einen neuen Token bekommt.

Time	User ID	Event saved type	IP address	Client
July 7, 2025 at 10:52 AM	c1205ab6-65de-4383-a059-ae7e88c5b088	CODE_TO_TOKEN	127.0.0.1	security-admin-console
July 7, 2025 at 10:52 AM	c1205ab6-65de-4383-a059-ae7e88c5b088	LOGIN	127.0.0.1	security-admin-console

Abbildung 5: Beispiel Events nach Eventtyp

Je nach Eventtyp werden Features angehängt oder weggelassen, z.B. wird beim Login noch in den Logs die Token-Url, die Art der Verbindung und auch die Art des Authentifizierungsprotokolls angegeben. "Type" steht hier für den Event-Typen. Die wichtigsten Event Type sind dabei folgende:

- LOGIN: Log, der generiert wird, wenn sich ein Benutzer erfolgreich einloggt über Keycloak selbst.
- CODE TO TOKEN: Wird erzeugt, sobald der Benutzer sich über seine Anmeldedaten auf dem Keycloak-Server authentifiziert hat und von diesem einen Access Token erhalten hat.
- CLIENT LOGIN: Log, der generiert wird, sobald sich der Benutzer über einen Drittanbieter einloggt.
- LOGIN ERROR: Fehler beim Login über Keycloak
- CODE TO TOKEN ERROR: Anmeldedaten war falsch, Server lehnt Anmeldedaten von Benutzer ab

In den Admin Logs wiederum sind noch Features enthalten, welche in den normalen User Event Logs nicht vorkommen, u.A. 'operationType': Diese beinhalten typische CRUD-Operationen wie UPDATE, CREATE, DELETE und ACTION <sup>18</sup>.

## 7 Theoretischer Hintergrund der Modelle

### 7.1 LSTM

Long Short-Term Memory (LSTM) ist ein Recurrent Neural Network (RNN), welches dazu entwickelt wird zeitabhängige Datenabfolgen zu erkennen und zu analysieren [23]. RNNs sind eine spezielle Klasse von neuronalen Netzwerken, welche dazu entwickelt wurden, sequentielle Daten zu verarbeiten, also Daten bei den die Reihenfolge der zu verarbeitenden Daten entscheiden ist. daher eignen sie sich gut Zeitreihen zu analysieren. RNNs können die nacheinander folgenden, abhängigen Daten anhand eines sogenannten Hidden Errors analysieren: Dabei wird im Hidden Error die zuvor analysierten Daten behalten und mit den nächsten Daten in einem gemeinsamen Kontext verarbeitet. Der Hidden State wird bei jedem Zeitschritt aktualisiert und bildet somit einen gemeinsamen Kontext für die Verarbeitung der aktuellen und vorherigen Eingaben. Das Problem an RNNs ist jedoch, dass bei längeren Datensequenzen die Erinnerung der zu weit liegenden Daten verblasst. LSTMs gleichen diese Nachteile aus: Sie besitzen über eine Zellstruktur, die gewährleistet, dass auch zu alte Datensequenzen noch erinnert werden. Dies wird durch diese drei Bestandteile gewährleistet [24]:

- **Forget Gate:** Entscheidet, welche Informationen aus der Zellzustand gelöscht werden sollen.
- **Input Gate:** Steuert, welche neuen Informationen in den Zellzustand aufgenommen werden.
- **Output Gate:** Bestimmt, welche Informationen aus der Zelle als Ausgabe weitergegeben werden.

LSTM jedoch haben auch ihre Kapazitäten und auch hier können längere Sequenzen trotz diesen Gates für Verarbeitungsschwierigkeiten sorgen. Probleme wie das Vergessen weiter zurückliegender Informationen oder eine ineffiziente Nutzung des Speichers können weiterhin auftreten – wenn auch in deutlich abgeschwächter Form im Vergleich zu Standard-RNNs [24]. LSTM werden in vielen Bereichen, wie z.B: in IoT eingesetzt [25].

### 7.2 Autoencoder

Ein Autoencoder ist eine spezielle Form eines neuronalen Netzwerks, das aus zwei Hauptbestandteilen besteht: der Encoder- und der Decoder-Schicht [18].

Die Encoder-Schicht komprimiert die Eingabedaten und reduziert ihre Dimensionen auf eine kleinere Repräsentation, die sogenannte Bottleneck-Schicht. Diese Bottleneck-Schicht ist der engste Teil des Netzwerks und zwingt das Modell, die wichtigsten Merkmale der Daten in verdichteter Form zu lernen. Dadurch werden irrelevante oder redundante Informationen herausgefiltert, was den Autoencoder befähigt, die Essenz der Eingabedaten zu erfassen.

Die Decoder-Schicht nimmt diese komprimierte Repräsentation und versucht, die ursprünglichen Daten daraus möglichst genau zu rekonstruieren. Das Netzwerk lernt durch Minimierung des Rekonstruktionsfehlers, also der Differenz zwischen Eingabedaten und rekonstruierten Daten. Dieser Fehler ist bedeutend für die spätere Hybridarchitektur.

Diese Fähigkeit macht Autoencoder besonders nützlich für Aufgaben wie Anomalieerkennung. Wenn die Rekonstruktion schlecht gelingt (hoher Rekonstruktionsfehler), deutet dies darauf hin, dass die Eingabedaten von den gelernten Mustern abweichen und somit potenziell eine Anomalie darstellen.

**Bottleneck und seine Bedeutung** Der Bottleneck ist entscheidend, da er die Kapazität des Autoencoders steuert. Ein zu kleiner Bottleneck kann dazu führen, dass wichtige Informationen verloren gehen und die Rekonstruktion schlecht wird. Ein zu großer Bottleneck hingegen erlaubt es dem Netzwerk, einfach die Identität zu lernen und Daten "durchzuschleusen", wodurch der Autoencoder seine Fähigkeit zur Merkmalskompression verliert.

**Varianten und Architekturen** Neben dem klassischen Autoencoder gibt es weitere Varianten, z.B. den Variational Autoencoder (VAE) (VAE), der probabilistische Modelle nutzt und generative Fähigkeiten besitzt, oder den Denoising Autoencoder, der lernt, verrauschte Eingaben zu bereinigen. Außerdem können Autoencoder aus mehreren Schichten bestehen (deep Autoencoder), um komplexere Merkmale zu extrahieren.

**Vor- und Nachteile** Autoencoder bieten den Vorteil, dass sie Merkmale automatisch extrahieren können, ohne dass explizite Labels benötigt werden. Dadurch eignen sie sich gut für unüberwachte Lernaufgaben. Zudem ermöglichen sie eine effiziente Dimensionsreduktion und Datenkompression. Allerdings benötigen Autoencoder zum Trainieren große Datenmengen. Bei einem zu stark eingeschränkten Bottleneck besteht zudem die Gefahr, dass sie eine triviale Identitätsfunktion lernen. Weiterhin ist der Rekonstruktionsfehler nicht immer ein perfekter Indikator für

Anomalien, insbesondere bei sehr komplexen Daten. Schließlich kann das Training solcher Modelle rechnerisch sehr aufwendig sein.

**Anwendungsbereiche** Autoencoder finden Anwendung in der Bildkompression, Anomalieerkennung in Netzwerken oder Produktionsprozessen, Feature Learning vor Klassifikationsaufgaben oder generativer Modellierung.

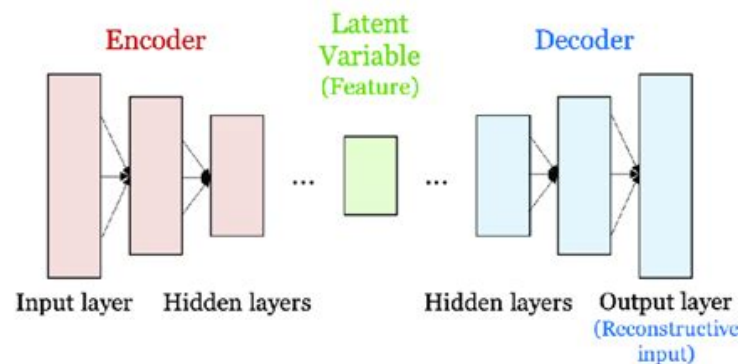


Abbildung 6: Visuelle Darstellung der Funktion des AEs, Quelle: [https://www.researchgate.net/figure/Structure-of-the-autoencoder\\_fig1\\_365074431](https://www.researchgate.net/figure/Structure-of-the-autoencoder_fig1_365074431)

Autoencoder werden für Anomalieerkennung oder auch Dimensionsreduktion verwendet.

**Mit dem LSTM kombiniert** Autoencoder lassen sich besonders gut mit Long Short-Term Memory (LSTM)-Netzwerken kombinieren, wenn es um sequenzielle oder zeitabhängige Daten geht. Während der klassische Autoencoder oft für statische Daten verwendet wird, können LSTM-Autoencoder zeitliche Abhängigkeiten erfassen und lernen, wie sich Daten über Zeit entwickeln.

Ein LSTM-Autoencoder nutzt LSTM-Zellen im Encoder und Decoder, um komplexe zeitliche Muster zu erfassen und zu rekonstruieren. Diese Kombination ist besonders nützlich bei Anomalieerkennung in Zeitreihen, wie etwa in der Überwachung von Maschinenzuständen, Finanzdaten oder Netzwerksicherheit. Anomalien fallen hier oft durch signifikant erhöhte Rekonstruktionsfehler auf, da die zeitlichen Muster nicht gut rekonstruiert werden können.

Durch die Fähigkeit von LSTM, langfristige Abhängigkeiten zu modellieren, und die Komprimierung des Autoencoders entsteht ein mächtiges Werkzeug für die Analyse und das Verständnis komplexer, sequenzieller Daten.

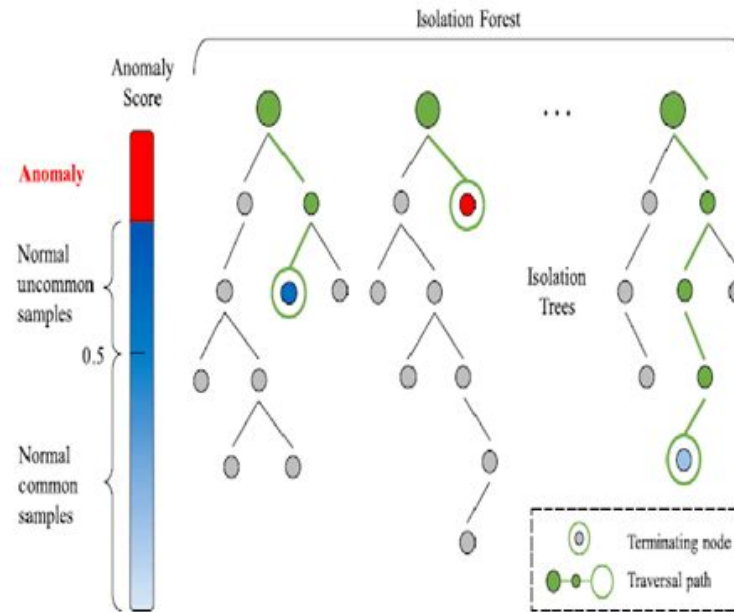


Abbildung 7: Visuelle Darstellung der Funktion des AEs, Quelle: [https://www.researchgate.net/figure/Anomaly-Detection-using-Isolation-Forest-18\\_fig3\\_350551253](https://www.researchgate.net/figure/Anomaly-Detection-using-Isolation-Forest-18_fig3_350551253)

### 7.3 Isolation Forest

Isolation Forest ist ein unüberwachter Lernalgorithmus, welcher durch Partitionen Daten voneinander trennen kann, indem er binäre Bäume erzeugt. Er unterteilt die Daten so lange, bis keine Isolierung der Daten mehr möglich ist. Durch dieses Vorgehen lassen sich auch Anomalien erkennen: Die Anomalien werden je danach erkannt, wie isoliert die Daten eingeteilt wurden. Je isolierter die Datenpunkte aufgeteilt wurden, desto wahrscheinlicher ist es, dass es eine Anomalie ist. Wenn sich Datenpunkte in ihrer Form ähneln, so würde der Algorithmus diese nicht zu stark isolieren. Lassen sich diese jedoch stark anhand ihrer Merkmale isolieren, so ist es wahrscheinlicher, dass diese keiner üblichen Menge eingeteilt werden kann [26].

Der Algorithmus erzeugt Isolation Trees, welche aus Blattknoten oder aus einem inneren Knoten besteht, der eine Testregel enthält, die die Daten in zwei Gruppen aufteilt. Die Testregel besteht aus einem Wert aus der Auswahl eines Merkmals  $q$  und einem Schwellenwert  $p$ , so dass alle Datenpunkte, deren Wert für  $q$  kleiner als  $p$  ist, in den linken Kindknoten kommen, und alle anderen in den rechten Kindknoten.

Zum Aufbau eines solchen Baums nimmt man eine Stichprobe von Datenpunkten und teilt sie rekursiv auf, indem man zufällig ein Merkmal und einen Split-Wert auswählt. Dieser Prozess wird so lange wiederholt, bis eine maximale Baumhöhe



erreicht ist, nur noch ein Datenpunkt übrig ist, oder alle Datenpunkte im aktuellen Teilbaum identisch sind [26].

Das Ergebnis ist ein vollständiger binärer Baum, bei dem jeder Datenpunkt in einem Blattknoten isoliert wird.

Wenn alle Datenpunkte unterschiedlich sind, dann gilt:

- Es gibt genauso viele Blätter wie Datenpunkte ( $n$ ).
- Die Anzahl der inneren Knoten ist  $n - 1$ .
- Insgesamt hat der Baum  $2n - 1$  Knoten.
- Der Speicherbedarf wächst also linear mit der Anzahl der Datenpunkte.

Isolation Forest unterteilt jedoch zufällig die Datenpunkte, wobei man die Sinnhaftigkeit des Algorithmus hinterfragen kann.

## 7.4 One-Class SVM

Mit dem rasanten Wachstum des Internets ist die Sicherheit von Informationssystemen zu einem global wichtigen Thema geworden. Effektive Methoden zur Erkennung von Eindringversuchen sind daher dringend erforderlich. Da viele Angriffe durch die Analyse von System-Logdaten erkennbar sind, wurde ein Ansatz entwickelt, der auf One-Class Support Vector Machines (OC-SVM) basiert und mit abstrahierten Benutzeraudit-Logs aus dem DARPA-Datensatz von 1999 trainiert wurde [27]. Die Methode nutzt eine nichtlineare Abbildung der Daten in einen höherdimensionalen Raum, um auch nichtlinear trennbare Daten linear trennbar zu machen: „Bei nicht-linear trennbaren Daten können wir sie durch eine nichtlineare Abbildung in einen hochdimensionalen Raum transformieren, so dass die Datenpunkte linear trennbar sind.“

Ein zentrales Problem bei One-Class Support Vector Machines (OCSVM) besteht darin, dass der Ursprung im Feature Space (Merkmalsraum) als Repräsentant für Anomalien dient, obwohl dies in der Realität nicht immer zutrifft [28]. In ihrer Arbeit erklären die Autoren, dass die Trennung der Zielklasse vom Ursprung (also von den Ausreißern) häufig missverstanden wird. Sie schlagen eine geometrische Interpretation vor, bei der die Zielklasse vom übrigen Raum getrennt wird, insbesondere wenn ein Gauß-Kernel verwendet wird.

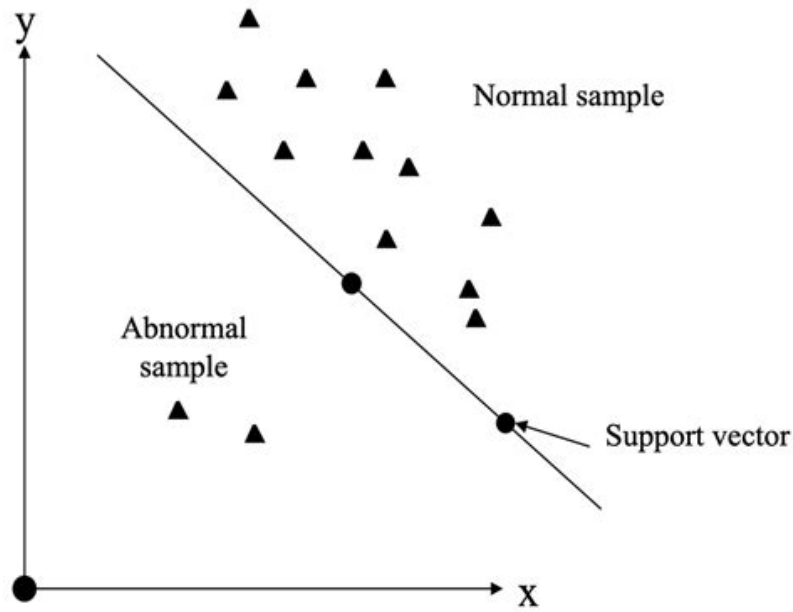


Abbildung 8: Visuelle Darstellung OCSVM. Quelle: <https://www.mdpi.com/2076-3417/13/3/1734>

## 7.5 DBSCAN

DBSCAN ist ein dichte-basiertes Clusteringverfahren, dass Datenpunkte anhand ihrer Dichte zu anderen Punkten klassifizieren kann. Die Dichte wird durch die Anzahl der Nachbarpunkte innerhalb eines Radius (Eps) gemessen, wobei mindestens eine Mindestanzahl von Punkten (MinPts) in diesem Radius vorhanden sein muss [29]. Die Wahl dieser Parameter ist entscheidend für den Erfolg der Klassifizierung.

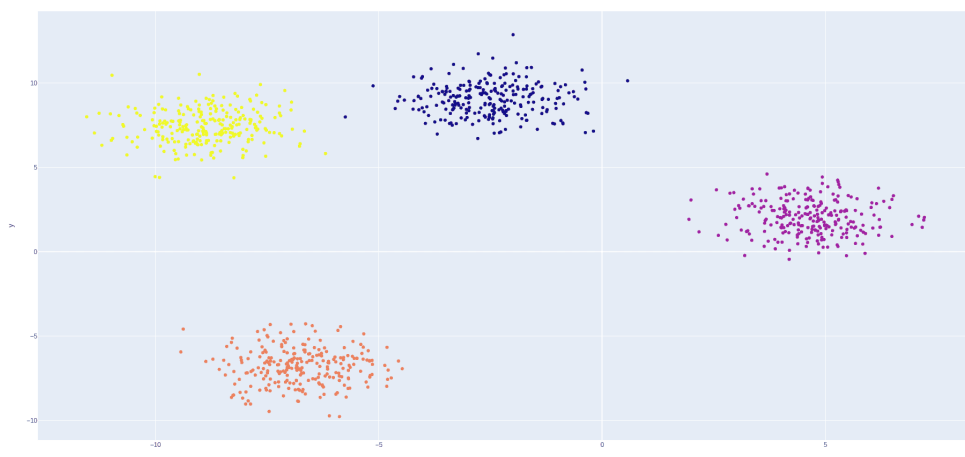


Abbildung 9: Beispiel DBSCAN-Cluster, Quelle: <https://www.baeldung.com/wp-content/uploads/sites/4/2023/04/clusters.png>

Der Algorithmus läuft folgendermaßen ab: Man startet von einem bestimmten Punkt  $p$ . Daraufhin werden alle Punkte gesucht, welche sich in der Nähe von  $p$  befinden. Sobald eine Mindestanzahl von  $Eps$  erreicht ist, die auch durch  $MinPts$  festgelegt wird, so bildet sich ein Cluster. Danach werden alle Punkte die nun zu einem Cluster gehören, aus einer Liste an verbleibenden Punkten herausgenommen. Daraufhin wird  $p$  zufällig an einen weiteren Ort im Raum platziert, wo noch kein Cluster gebildet wurde. Dies geschieht solange, bis keine Punkte in der Liste übrig sind. Die restlichen Punkte, die keinem Cluster zugeordnet sind, heißen *Noises*. Wenn es zu viele Noises gibt, heisst das, dass nicht effizient klassifiziert wurde [29].

## 7.6 Alternative Algorithmen

Wie bereits im Forschungsstand erwähnt, wurde als Alternative zu DBSCAN auch das hierarchische Verfahren HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) in Betracht gezogen. HDBSCAN ist in vielerlei Hinsicht robuster als DBSCAN, insbesondere bei der Identifikation von Clustern in hochdimensionalen oder verrauschten Daten. Ein Vorteil von HDBSCAN ist, dass es keine globale Dichte-Schwelle benötigt und sich besser an lokale Datenstrukturen anpasst. Aufgrund der derzeit noch begrenzten Anzahl an Studien und praktischen Erfahrungswerten zu HDBSCAN im konkreten Anwendungsbereich wurde jedoch auf den Einsatz verzichtet. Für zukünftige Arbeiten wäre es durchaus interessant, HDBSCAN in Kombination mit LSTM-Autoencodern (LSTM-AE) näher zu untersuchen.

Darüber hinaus wurden auch weitere Anomalieerkennungsverfahren wie K-Means und LOF (Local Outlier Factor) evaluiert. Beide Methoden sind grundsätzlich für die Anomalieerkennung geeignet, insbesondere bei kleineren und klar strukturierten Datensätzen. Allerdings stoßen K-Means und LOF bei großen, komplexen oder verrauschten Datenmengen an ihre Grenzen. K-Means ist dabei empfindlich gegenüber Ausreißern und erfordert die Festlegung der Clusteranzahl vorab. LOF erkennt Ausreißer, indem es die lokale Dichte eines Datenpunkts mit der seiner Nachbarn vergleicht und Punkte mit stark abweichender Dichte als potenzielle Anomalien markiert. Allerdings wird LOF mit zunehmender Datenmenge zunehmend ineffizient.

# 8 Implementierung

## 8.1 Hybridmodelle mit LSTM-AE

Generell wurden die Modelle nach der selben Architektur gebaut:

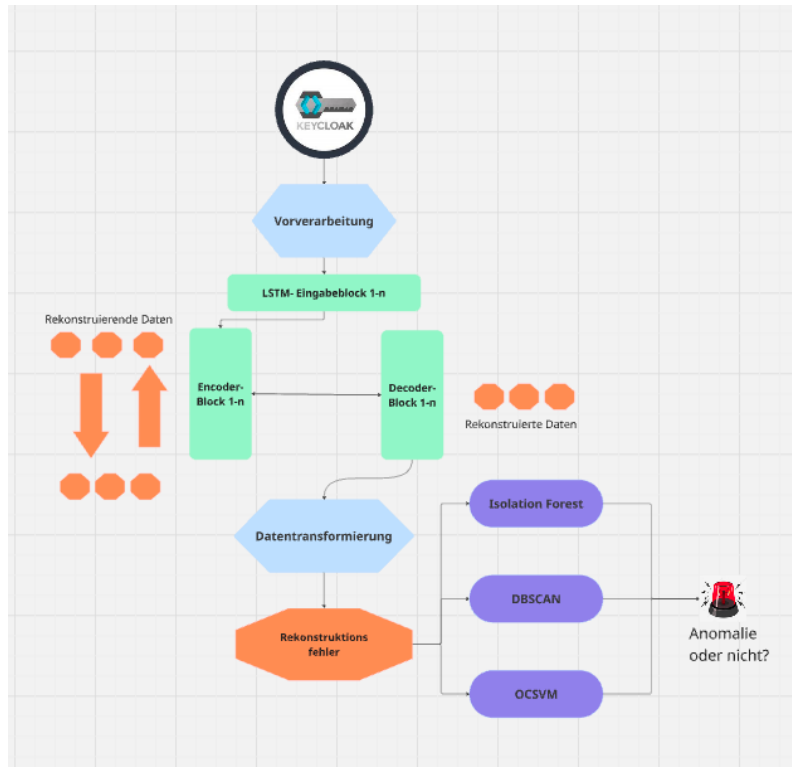


Abbildung 10: Architektur

Die Daten werden zuerst in kategorische oder numerische Daten aufgeteilt. Absichtlich werden NaN-Werte und unklare Werte nicht bereinigt, sondern in eine eigene Kategorie gefasst. Dies ist wichtig, da auch solche Werte Anomalien darstellen und die Bereinigung dieser Daten dafür sorgen würde, dass potenzielle Gefahrquellen unerkannt bleiben. NaN-Werte können außerdem bspw. bei falschen Accounts oder vielleicht bei Bots auftreten, weshalb es wichtig ist, diese in der Analyse beizubehalten. Zudem ist der LSTM-AE sehr gut darin verschieden Daten, sei es kategorisch oder numerisch zu erkennen, weshalb solch eine Vorverarbeitungsweise gewählt wurde.

Die Logs werden separat je nach Training, Test und Validierung in Dateien gepackt und dann zu Beginn ausgelesen. Zuerst werden diese Daten vorverarbeitet. Die Features werden danach automatisch in numerische, kategorische oder nicht definierte Daten eingeteilt. Die eingeteilten und bereinigten Daten werden danach wieder zusammengefügt. Alle drei Dateien, also Evaluierungs-, Test- und Trainingsdateien werden dabei einzeln bereinigt und zusammengefügt. Darauf hin werden alle Daten mit der standardisiert und skaliert. Da der LSTM-AE nur mit sequentiellen Daten arbeiten kann, wurden diese Dateien in Sequenzen aufgeteilt.

Es wurden folgende Parameter bestimmt:

Parameter	Wert
seq_length	15
batch_size	128
encoder_layers	[256, 128, 64]
decoder_layers	[64, 128, 256]
dropout_rate	0.2
learning_rate	0.0001

Tabelle 1: Bestimmte Parameter der Modellkonfiguration

Die Trainingsfehler, bzw. Rekonstruktionsfehler, also die Daten, welche nicht klar klassifiziert werden konnten, werden danach jeweils mit einem der vorimplementierten Modelle trainiert. Dies ist auch eine Vorgehensweise, welche schon angewandt wurde [30]. Die Implementierung erfolgt in Form einer modularen Pipeline. Zunächst wird der LSTM-Autoencoder sequenziell trainiert und die Rekonstruktionsfehler für die Daten berechnet. Anschließend werden diese Fehler unabhängig voneinander als Eingabe für die verschiedenen Anomalieerkennungsmodelle (Isolation Forest, One-Class SVM, DBSCAN) verwendet. Dadurch kann jedes Modell separat trainiert, getestet und evaluiert werden, was eine modulare Struktur gewährleistet, auch wenn die gesamte Umsetzung in einem Jupyter Notebook erfolgt.

### 8.1.1 Implementierung: Isolation Forest

Die Trainingsfehler, also die Daten, welche nicht klar klassifiziert werden konnten, werden danach jeweils mit einem der vorimplementierten Modelle trainiert. Im Fall des Isolation Forest wird ein auf Entscheidungsbäumen basierender Algorithmus genutzt, der speziell für die Erkennung von Ausreißern in hochdimensionalen Daten entwickelt wurde. Dabei wird das Modell auf den Fehlerwerten der Trainingsdaten fitte, um anschließend auf die Testdaten angewendet zu werden. Die Vorhersage klassifiziert dabei jeden Datenpunkt als normal (1) oder anomal (-1). Diese Ergebnisse werden binär kodiert, um eine einfache Auswertung zu ermöglichen. Da die zugrunde liegenden Daten sequenzbasiert aufgebaut sind – beispielsweise aus Zeitreihen bestehen – wird zusätzlich ein Abgleich mit den tatsächlichen Labels vorgenommen. Hierzu wird aus den Zielwerten der Testdaten die letzte Klasse jeder Sequenz extrahiert, um sie mit den Vorhersagen vergleichen zu können.

### 8.1.2 Implementierung: OCSVM

Für die Umsetzung der One-Class SVM wird das Modell mit einem sehr niedrigen  $\nu$ -Wert von 0.005 initialisiert. Dieser Wert steuert die erwartete Rate an Ausrei-

ßern im Trainingsdatensatz. Zusätzlich wird der Parameter `gamma` auf 50 gesetzt, was die Einflussweite einzelner Datenpunkte auf die Entscheidungsgrenze definiert. Anschließend wird das Modell mit den Fehlerwerten der Trainingsdaten trainiert, die zuvor durch ein Rekonstruktionsmodell (z. B. LSTM-Autoencoder) erzeugt wurden. Nach dem Training wird die One-Class SVM auf die Fehler der Testdaten angewendet. Die Methode `predict()` liefert Vorhersagen in Form von 1 (normal) oder -1 (anomal). Diese Vorhersagen werden daraufhin in binäre Werte (1 = Anomalie, 0 = normal) umgewandelt. Um die Klassifikation korrekt auszuwerten, wird wie auch bei den anderen Verfahren für jedes Sequenzfenster das zugehörige Label aus den ursprünglichen Testlabels extrahiert. Daraus entsteht ein Label-Vektor, der mit den Vorhersagen verglichen wird.

### 8.1.3 Implementierung: DBSCAN

Im Fall von DBSCAN handelt es sich um ein dichtebasiertes Clusteringverfahren, das keine explizite Trainingsphase benötigt. Das Modell wird direkt auf die Fehlerwerte der Testdaten angewendet. Als Parameter wird `eps = 0.05` gewählt, um die maximale Nachbarschaftsdistanz für die Punkte zu begrenzen, sowie `min_samples = 40`, was die Mindestanzahl an Nachbarn definiert, um als Kernpunkt eines Clusters zu gelten. Die Methode `fit_predict()` gibt für jeden Punkt entweder eine Clusterzuweisung (0, 1, ...) oder -1 zurück – letzteres kennzeichnet Datenpunkte, die als Rauschen bzw. Anomalien erkannt wurden. Analog zu den anderen Verfahren werden die -1-Werte in binäre Anomalien umgewandelt. Die tatsächlichen Labels werden erneut sequenzbasiert aus dem vollständigen Label-Vektor extrahiert. Abschließend wird ein Classification Report erzeugt, um die Qualität der durch DBSCAN erkannten Anomalien gegenüber den echten Anomalien zu evaluieren.

## 8.2 Alternative Architekturen

Es bestand die Möglichkeit, vorher eines der Algorithmen zu trainieren und danach die selben Daten nochmals in den LSTM-AE. Es ist also der umgekehrte Weg, welcher auch in einer Masterarbeit verwendet wurde, bei dem ein Hybridmodell zwischen IF und LSTM-AE erstellt wurde [?]. Desweiteren gibt es auch Architekturen, welche nur DBSCAN und OCSVM benutzen, wobei die Daten zuerst von OCSVM verwendet werden und später durch DBSCAN geclustert werden [31].

Es gab auch Ideen, für größerer Hybridmodelle. U.A. wurde eine große Kombination aus LSTM-AE, DBSCAN und IF erstellt. Die Daten werden demnach zuerst vom LSTM-AE verarbeitet. Die Fehler daraus werden geclustert durch DBSCAN,

falls es auch unterschiedliche Fehlerarten gibt. schlussendlich entscheidet IF, welche Anormal sind und welche nicht.

Diese Alternativen wurden jedoch nicht umgesetzt, da sie zu komplex sind, insbesondere, weil in dieser Arbeit nur verhaltensbasierte Anomalien verarbeitet werden, welche sich in Event-Logs zeigen. Solch ein komplexes Modell wurde auch erwogen, weil zu Beginn diskutiert wurde, Modelle in Cloud-Umgebungen zu verwenden, weshalb ein robustes Modell notwendig ist, um große Datenmengen effizient zu verarbeiten und daraus Anomalien zu erkennen. Schlussendlich entschied man sich jedoch für eine Kombination aus LSTM-Autoencoder und jeweils einem klassischen Algorithmus, da so der Vergleich der einzelnen Verfahren deutlich klarer und nachvollziehbarer bleibt.

Darüber hinaus ermöglichen einfachere hybride Modelle eine bessere Interpretierbarkeit und Wartbarkeit, was in produktiven Systemen von großer Bedeutung ist. Komplexe Modelle bringen häufig einen erheblichen Mehraufwand bei der Parametrisierung und dem Training mit sich, der in Bezug auf den Erkenntnisgewinn und die praktische Anwendung nicht immer gerechtfertigt ist.

Zudem ist zu beachten, dass verhaltensbasierte Anomalien in Event-Logs oftmals durch zeitliche Muster und Sequenzen charakterisiert sind, was der LSTM-Autoencoder durch seine Fähigkeit zur Modellierung von Zeitreihen besonders gut abdeckt. Die ergänzenden klassischen Algorithmen dienen hier vor allem zur Unterstützung bei der Erkennung von Ausreißern oder Clustern, wodurch eine effiziente und dennoch aussagekräftige Erkennung gewährleistet wird.

Durch die klare Trennung der Methoden in der gewählten Architektur lassen sich zudem die jeweiligen Stärken und Schwächen besser analysieren und bewerten, was den wissenschaftlichen Vergleich und die spätere Optimierung erleichtert.

## **9 Möglichkeit 1: Generierung der Logs**

### **9.1 Angriffsfälle**

Nach jetzigem Stand gehören zu den häufigsten Insider Threat-Attacken in IT-Systemen u.A. der Missbrauch von hohen Privilegien, Rollen und Recht, Datendiebstahl, sowie der Manipulation wichtiger Daten, wie bspw. dem Löschen von wichtigen Dateien [32]. Brute-Force Angriffe gehören ebenfalls zu den wichtigsten Attacken. Es wurden jene Angriffsmöglichkeiten umgesetzt, welcher auch am wahrscheinlichsten auftreten und auch in den Logs erkennbar sind. Datendiebstahl bspw.

lässt sich nicht direkt erkennen in Keycloak, weil in den Log-Dateien wenig zum Upload großer Dateien etwas angegeben wird. Es wird erwartet, dass Angriffe, wie dem Stehlen von Rollen in Keycloak eher auftreten würden, sowie Brute-Force Attacken, weshalb diese Anwendungsfälle umgesetzt werden. Sie sind zudem in den Logs bspw., durch Attribute wie dem Event Typ LOGIN\_ERROR erkennbar.

Da der Fokus dieser Arbeit auf dem Gebiet des anomalen Benutzerverhaltens liegt, werden bewusst Angriffsfälle wie DDoS-Attacken nicht berücksichtigt. Von daher werden nur folgende Angriffsfälle modelliert in den Logs:

- *Anwendungsfall 1:* Brute-Force Attacken bei der Anmeldung
- *Anwendungsfall 2:* Löschen und Änderungen an wichtigen Clients, Usern, Passwörtern und Realms
- *Anwendungsfall 3:* Erlangung wichtiger Rollen des Admins, wie bspw. manage-users, und realm-admin
- *Anwendungsfall 4:* Häufiges Abfragen derselben Quelle

Zudem werden bekannte Anomalien wie ungewöhnlichen IP-Adressen und ungewöhnliche Anmeldezeitpunkten bei der Generierung berücksichtigt. Es wird darauf geachtet, hohe Variabilität bei den Angriffen zu gewährleisten.

## 9.2 Automatische Erstellung der Keycloak-Logs

Die Logs werden durch ein Python-Skript erzeugt.

Der erste Anwendungsfall wurde so umgesetzt, dass eine Abfolge von mehreren Logs erzeugt wird, welche den Event Typen LOGIN\_ERROR enthalten. Die Logs können ungewöhnliche timestamps und/oder IP-Adressen enthalten, was aber nicht immer der Fall sein muss, weil schon die Abfolge dieses Event Types anomal ist. Da in Keycloak generell selbst bestimmt werden kann, ab wie vielen Anmeldeversuchen, die IP-Adresse gesperrt wird, muss bei der Generierung der Logs ein Raum angegeben werden dazu. Es werden dabei ca. zwischen 3 bis mehrere 15 Log-Abfolgen zufällig erzeugt. Der Grund für diese Auswahl besteht aus Erfahrung: Ab 3 fehlgeschlagenen Anmeldeversuchen wird das Verhalten als anomal gedeutet, auch wenn es häufig nicht der Fall ist, ab 15 sollte dann schon es eine Anomalie sein, wie alles darüber. Man entschied sich dabei die Zahl gering zu halten, damit die Sessions nicht zu lang werden.

Für den zweiten Anwendungsfall sind Features, welche auf Updates oder dem Löschen



von Entitäten deuten, wie z.B. DELETE, UPDATE\_PASSWORD oder auch UPDATE\_PROFILE. Auch hier wird eine Anzahl an Logs gewählt, welche auf das anomale Verhalten hindeuten. Es werden mindestens 5 Abfolgen erzeugt, welche eines der erwähnten Event Typen beinhaltet. Hierbei wird auf Variabilität geachtet, für mehrere Anwendungsmöglichkeiten, damit keine zu strikte Abfolge erzeugt wird. Denn wenn das Modelle stets das gleiche Angriffsmuster erkennt mit den selben Features, dann kann es zu Overfitting führen und das Modelle kann keine neue Angriffe erkennen. Bspw. werden die Logs so erzeugt, dass die Features unterschiedlich zusammengesetzt werden können. Wenn ein Feature „authType“ bspw. mit dem Event Typ UPDATE ständig auftritt, dann ist es wahrscheinlich, dass das Modell stets diesen Zusammenhang nur kennt und keine weiteren.

Natürlich ist der Nachteil dieser Art von Log-Generierung dass auch Logs erzeugt werden, welche ein Recht schwaches oder stupides Angriffsmuster haben, welche eher nicht auf einen Angriff, sondern auf zufällige Auswahl von Ereignissen hindeuten, welche keinen Zusammenhang haben. Um das zu verhindern, wurde eine Logik eingebaut, die dafür sorgt, dass die Regeln im System Keycloak nicht widersprüchlich sind. Bspw. darf es einen Benutzer mit eindeutiger ID nur in einem Realm geben und den Admin auch nur einmal in ganz Keycloak. Es wird fest definiert, welche Rollen dem Admin zugeordnet sind und damit eine hohe Priorität haben und welche Rollen auch für normale Benutzer gelten. Es wurde auch darauf geachtet, dass der normale Benutzer keine Features wie dem „operationType“ benutzen kann, also alle CRUD-Operationen auf den Entitäten. Dieses Feature sind nur in den Admin-Logs enthalten. Es wurde auch klar definiert, dass ein Benutzer genau einer Session angehört und auch nur einem Realm, es sei denn, es ist ein Angreifer.

Der 3. Anwendungsfall prüft, ob es mehrere Admins gibt und ob ein eigentlich normaler Benutzer plötzlich Admin wird. Hier wurde auch bei der Generierung der Logs darauf geachtet, dass die Sessions dabei einen Benutzer wählen, welcher zu Beginn keine Admin-Rollen hat und plötzlich mehrere Admin-Rollen bekommt, bzw. auch selbst die Admin-Id erhält. Die Rollen, die dabei überprüft werden sind bspw. admin, realm-admin und manage-users.

Der 4. Anwendungsfall beinhaltet mehrere Logs, welche die Admin-Operation GET stets enthalten auf die gleiche Quelle. Dies kann ein Benutzer oder ein Client sein.

Normale User-Sessions enthalten keine anomalen IP-Adressen oder ungewöhnliche timestamps. Ein ungewöhnlicher timestamps in diesem Fall sind die Uhrzeiten zwischen Abend 0 Uhr bis 6 Uhr Morgens. Natürlich ist dies stets Interpretationssache, welche Uhrzeiten normal und welche unnormale sind. Es kann auch Benutzer ge-

ben, welche gerne im Homeoffice auch Abend arbeiten und eigentlich damit kein anormales Verhalten zeigen. Jedoch wird hier in dieser Arbeit vom Regelfall ausgegangen, also dass es anormal ist, wenn ein Benutzer Mitternachts eingeloggt ist.

Die Bestimmung von unnormalen IP-Adressen wurde überwiegend aus Quellen der IANA<sup>19</sup> übernommen.

```
Tests > {} test_logs.json > ...
23 {
24   "authDetails": {
25     "ipAddress": "43.243.100.34",
26     "realmId": "aws",
27     "clientId": "security-admin-console",
28     "sessionId": "b9c7bbdf-4b98-478d-af91-93c31f9a457a"
29   },
30   "operationType": "CREATE",
31   "resourceType": "groups",
32   "resourcePath": "groupss/d75a63cb-1754-49fa-9b29-ac419e9f1bca",
33   "label": 0
34 },
35 {
36   "timestamp": "2025-07-20T19:00:27.042229+02:00",
37   "log_level": "error",
38   "category": "org.keycloak.events",
39   "type": "LOGIN_ERROR",
40   "realmId": "telekom",
41   "realmName": "telekom",
42   "clientId": "admin-cli",
43   "ipAddress": "198.205.39.117",
44   "error": "invalid_credentials",
45   "username": "Luca",
46   "details": {
47     "auth_type": "oauth2-client-credentials",
48     "connection": "openid-connect",
49     "scope": "phone",
50     "grant_type": "authorization_code",
51     "refresh_token_id": "9bce28bc-e800-4650-8461-f04c5bbf5099",
52     "code_id": "9d363895-f4b3-45b3-8e96-834ecb61a42b"
53   },
54   "label": 1
55 },
56 {
57   "timestamp": "2025-07-20T19:06:36.534525+02:00",
```

Abbildung 11: Ausschnitt der generierten Logs

Bei der Generierung der Log-Daten wurde auch beachtet, dass nicht nur Normale und Anormale Benutzer-Sessions erzeugt werden, sondern auch Sessions, welche sich nicht klar zuordnen lassen. Dies wird dann auch *Noise* genannt. Noise tritt des Öfteren in realen Anwendungsfällen auf, weil auch normale Benutzer manchmal bspw. priorisierte Rollen erhalten müssen aus Management-Gründen oder auch zu oft das Passwort ändern aus eigenen Bedenken der Sicherheit. Die Modelle würden dazu tendieren diese als Anormal zu deuten. Dies wird hier verhindert, indem noch im Script mit Absicht Noises erzeugt werden.

### 9.3 Problematik dieser Lösung

Bei der Generierung der Logs wurden nur die wichtigsten Features erzeugt, welche auf notwendig, sind um Anomalien zu finden. Im Rahmen der Arbeit war es nicht

möglich, detaillierte Informationen über die genaue Anlage und Struktur der Event Typen im Keycloak-System zu erhalten, da hierzu keine umfassende Dokumentation vorlag und der Zugriff auf administrative Ressourcen eingeschränkt war. Daher wurde die Analyse auf die verfügbaren und typischen Log-Daten gestützt, die generische Features und Muster der Systemaktivitäten widerspiegeln. Diese Einschränkung wird im weiteren Verlauf berücksichtigt und limitiert die Tiefe der Auswertung.

Eine Herausforderung dieser Arbeit war, die Wahrscheinlichkeit zu bestimmen, wann eine Session Anormal ist, wann normal und wann es einfach Noise, also normale Logdaten, welche aber Anomalien ähneln, ist. Da Anomalien selten sind wurde ein Wert für die Wahrscheinlichkeit des Auftretens einer Anomalie 2 Prozent bestimmt. Da diese Entscheidung auf Intuition basiert, wird nicht gewährleistet, dass die Daten keine realen Daten abbilden können. Dies beeinflusst das Ergebnis.

Aufgrund dieser Nachteile und den resultierenden Ergebnissen, wurde dieser Ansatz an den Modellen getestet, wird aber verworfen.

## **10 Möglichkeit 2: Datenbeschaffung als Alternative**

Da die Generierung der Daten eine hohe Aufwendigkeit benötigt, wird stattdessen der sinnvollere Ansatz verwendet, echte Keycloak-Daten zu beschaffen. Jedoch werden es keine Daten aus Unternehmen, da man keine datenschutzrechtlichen Risiken eingehen will. Stattdessen wurde die Idee angenommen, über eine lokale Keycloak-Instanz selbst Log-Daten zu generieren. Da dieser Weg zeitaufwendig ist, wurde in Betracht gezogen, über sogenannte Selenium-Tests Angriffsszenarien zu inszenieren und ebenfalls durch Selenium-Tests gewöhnliche Keycloak-Daten zu generieren. Was dabei als normale Keycloak-Logs zu verstehen ist, wird dadurch herausgefundenen, sich an firmeninternen Logs zu orientieren.

### **10.1 Anbindung der Keycloak-API**

Um Zugriff auf die Keycloak-Logs zu erhalten, ist es notwendig, selbst eine Keycloak-Instanz aufzusetzen und selbst als Admin zu navigieren. Um nun an die Logs aus Keycloak, also die die aus Quarkus hauptsächlich aus generiert werden, zu kommen, muss man zuerst die benötigten Ressourcen finden. Standard gemäß benötigt man Admin-Rollen, um an sensible Informationen wie User Event Logs und den Admin Event Logs selbst zu gelangen. Man benötigt zunächst einmal die korrekten Authen-

tifizierungsmittel, damit man auch Rechte erlangt, um auf die Events zugreifen zu dürfen.

Der gewählte Grant Typ hierfür ist *Client Credentials*. Hierbei meldet man sich über einen beliebigen Client an. man benötigt noch das Client-Secret und schickt dieses mit den anderen Daten ab.

```
def get_token():
    data = {
        'client_id': client_id_local,
        'client_secret': client_secret_local,
        'grant_type': 'client_credentials'
    }
```

Abbildung 12: Daten, die an den Keycloak-Server gesendet werden müssen

Man erhält dadurch den Token, welcher benötigt wird, um nun an die Event Logs zu importieren. Im Token sind die Zugriffsrechte geschildert, die der jeweilige Admin hat, er benötigt die Rollen, die dazu benötigt ist, Benutzer zu verwalten, welche dann auch im Token aufgelistet ist. Nun wird eine weitere URL aufgerufen, bzw., ein Request wird gestellt, um die Event Logs zu holen, wobei der Token als Autorisierungsnachweis mitgegeben wird. Die geladenen Event Logs werden danach in einer separaten JSON-Datei gespeichert.

Genauere Informationen über die Keycloak API lassen sich in der Doku finden <sup>20</sup>.

## 10.2 Selenium-Tests zur Erzeugung der Logs

Selenium ist ein Framework zur automatisierten Ausführung von Webbrowser-Aktionen und in Python als Paket verfügbar. Es ermöglicht, Interaktionen mit der Benutzeroberfläche realitätsnah zu simulieren – so, als würde ein Mensch den Browser bedienen. Die wesentlichen Komponenten sind:

- **Selenium WebDriver:** Steuert den Browser programmgesteuert – das zentrale Element der Tests.
- **Selenium Grid:** Ermöglicht die parallele Ausführung von Tests auf verschiedenen Maschinen und Browsern.
- **Selenium IDE:** Ein Recording-Tool im Browser (nicht empfohlen für produktive oder komplexe Tests).

Für die Erstellung der Logs wird ausschließlich der WebDriver verwendet, welcher automatisch den Chrome-Browser startet. Die Wahl fiel auf Selenium, da es sich unkompliziert mit Keycloak verbinden lässt und sich gut für automatisierte Tests auf dieser Plattform eignet. Die Einrichtung ist einfach: Man definiert den zu verwendenden Browser ("Driver") und führt die Tests auf dem Port aus, auf dem die lokale Keycloak-Instanz verfügbar ist.

Ein Nachteil dieser Methode besteht in der konstanten IP-Adresse – alle Anfragen stammen vom selben Host. Das führt dazu, dass das Modell unter Umständen lernt, dass mehrere Benutzer eine gemeinsame IP nutzen, was in realen Szenarien möglicherweise auf eine Anomalie hindeutet. Zudem erfolgt die Log-Erzeugung in kurzen zeitlichen Abständen, was die zeitliche Streuung reduziert. Da das LSTM-AE-Modell zeitbasierte Muster analysiert, kann dies seine Fähigkeit zur Anomalieerkennung einschränken. Zwar werden auch Tests zu ungewöhnlichen Uhrzeiten durchgeführt (z.B. abends), diese erzeugen jedoch nur wenige auffällige Logs.

Trotz dieser Einschränkungen überwiegt der Vorteil, dass die erzeugten Logs den realen Keycloak-Logs sehr nahekommen – ein zentraler Aspekt, insbesondere wenn wirtschaftlich verwertbare Erkenntnisse aus dieser Arbeit gewonnen werden sollen. Die Testumgebung ermöglicht die Generierung großer Datenmengen (z.B. über 100.000 Events), wodurch das LSTM-AE-Modell ausreichend Trainingsdaten erhält. Da die Logs direkt durch die definierten Tests erzeugt und gespeichert werden, entfällt die Notwendigkeit einer separaten manuellen Log-Generierung.

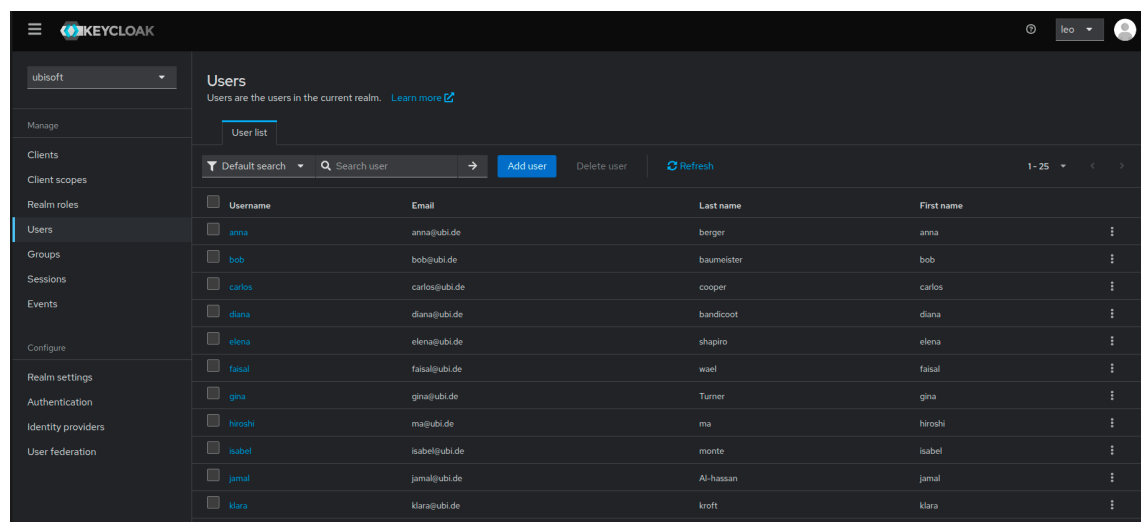


Abbildung 13: Hier sieht man die generierten Benutzer für den Testfall

Es wurde zur Generierung der normalen Trainingsdaten zuvor vier Realms er-

stellt:

- *ubisoft*
- *Nintendo*
- *Sega*
- *aws*

Pro Realm erfolgt die Benutzeranmeldung über einen jeweils definierten Client.

Im Realm *ubisoft* wurden 25 synthetische Benutzer erstellt, deren Namen alphabetisch generiert wurden (jeder Buchstabe des Alphabets außer „X“ ist vertreten). Für diesen Realm wurde der Client *ps3* angelegt. Dieser ist erforderlich, damit sowohl Administratoren Zugriff auf die User-Events erhalten als auch Benutzer sich über diesen Client authentifizieren können. Um sicherzustellen, dass während der Selenium-Tests nicht auf den Master-Realm zugegriffen wird, muss für den Client eine passende Redirect-URL definiert werden.

In den weiteren Realms – *Sega*, *Nintendo* und *aws* – wurden jeweils zehn Benutzer angelegt. Die zugehörigen Clients lauten: *megadrive* (Sega), *wii* (Nintendo) und *aws-console* (aws).

Jeder Realm verfügt über mindestens einen Administrator mit der Rolle *realm-admin*. Ein Admin-Benutzer, der für einen bestimmten Client Benutzer anlegen oder verwalten möchte, benötigt darüber hinaus die Rollen *manage-users* und *view-events*. Wichtig ist hierbei, dass diese Rollen nicht direkt dem Benutzer zugewiesen werden, sondern dem jeweiligen Client. Der Grund dafür liegt in der Architektur von Keycloak: Clients besitzen eigene Zugriffsrechte, die ihnen bestimmte Operationen ermöglichen. So erlaubt die Rolle *manage-users* dem Client, Benutzer im Realm programmatisch zu verwalten, während *view-events* erforderlich ist, um Zugriff auf die Ereignisprotokolle zu erhalten – essenziell für Logging und Fehlersuche.

Im Rahmen dieser Arbeit erzeugt ein Selenium-Test automatisch neue Benutzerkonten in Keycloak und weist ihnen jeweils ein Passwort zu. Diese Daten werden pro Realm einmalig in einer JSONL-Datei gespeichert, wobei Benutzername und Passwort gemeinsam als Schlüssel-Wert-Paar abgelegt werden.

Damit sich die erzeugten Benutzer über die automatisierten Tests in Keycloak anmelden können, muss in der Konfiguration festgelegt werden, dass kein vollständiges Profil (Vorname, Nachname, E-Mail) erforderlich ist – andernfalls würde die Authentifizierung fehlschlagen.

Darüber hinaus verfügen Clients über sogenannte Service Accounts – spezielle technische Benutzerkonten, die es ihnen ermöglichen, sich selbstständig zu authentifizieren, beispielsweise über den OAuth2 Client Credentials Grant. Die Service-Account-Rollen bestimmen dabei die verfügbaren Rechte. Ohne diese Rollen könnte der Client zwar eine Verbindung herstellen, hätte jedoch keinen Zugriff auf Benutzerverwaltung oder Logdaten. Die korrekte Konfiguration der Service-Account-Rollen ist daher zwingend notwendig, um im Hintergrund automatisiert operieren zu können.

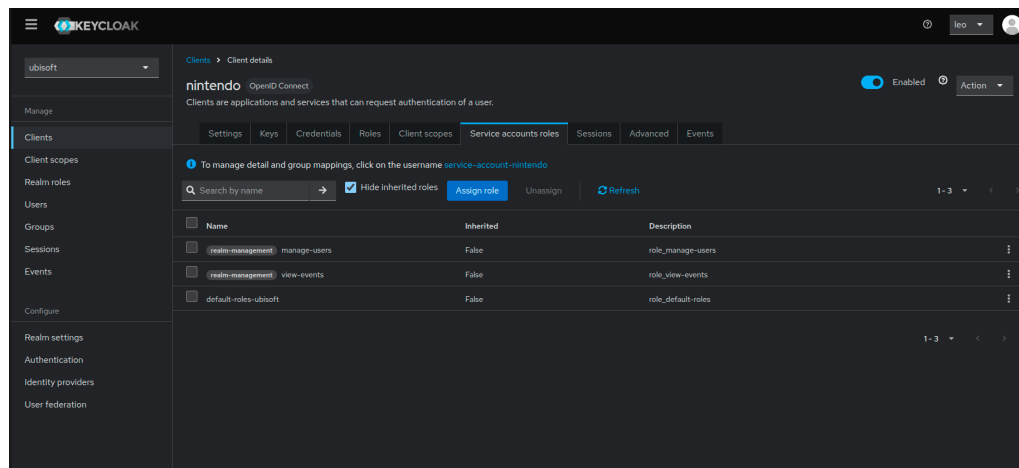


Abbildung 14: Benötigte Rollen für den Client Nintendo

Ab einer bestimmten Menge an Trainingsdaten werden die generierten Benutzer im Rahmen des Selenium-Tests dazu gebracht, sich regulär anzumelden. Die dabei entstehenden sogenannten „gewöhnlichen“ Logs umfassen unter anderem Anmeldevorgänge und basieren auf realen firmeninternen Protokollen. Diese bestehen typischerweise aus den Event-Typen LOGIN, CLIENT\_LOGIN und CODE\_TO\_TOKEN. Folgende Aktivitäten gelten laut interner Log-Analyse als „typisch“:

- Anmeldung über Keycloak direkt
- Anmeldung über einen Client
- Token-Aktualisierung (Refresh Token)
- Abmeldung (Logout), auch nach Ablauf einer Session
- Fehlgeschlagene Logins (kommen häufiger vor)

Weitere Aktionen – wie das Anpassen des Benutzerprofils (z.B. E-Mail-Änderung, Passwort-Zurücksetzung), das Löschen eines Clients oder das Zuweisen von Rollen

an andere Benutzer bzw. Clients – wurden in der Log-Generierung bewusst nicht automatisiert. Der Grund liegt in der vergleichsweise hohen Komplexität und der geringen Häufigkeit solcher Vorgänge. Da Keycloak primär als SSO-Dienst genutzt wird, wäre eine regelmäßige Durchführung solcher Aktionen im automatisierten Testkontext unrealistisch und würde die Aussagekraft der „normalen“ Logs verzerren.

Stattdessen werden solche seltenen Aktionen manuell während des Testbetriebs von einzelnen Benutzern durchgeführt. Um dabei in die „Rolle“ eines anderen Benutzers zu schlüpfen, wird der Event-Typ IMPERSONATE verwendet. Die Verwendung von Impersonation während der automatisierten Testausführung wird vermieden, da sie eine potentielle Anomalie darstellt. Ziel ist es, lediglich gelegentlich und gezielt ungewöhnliche Aktionen zu erzeugen – etwa eine Passwort- oder E-Mail-Änderung, das Anlegen oder Löschen eines Benutzers oder das Zuweisen von Rollen. Dies reduziert den manuellen Aufwand bei der Erzeugung seltener Ereignisse, ohne die Gesamtdatenbasis zu verzerren.

Pro Tag werden insgesamt 15 Sessions durchgeführt, wobei jede Session durch den häufigen Einsatz von Refresh Tokens mindestens zehn Einträge im Log erzeugt. Die Sitzungsdauer wird dabei durch die Konfiguration des jeweiligen Realms bestimmt. Aus praktischer Erfahrung ergibt sich eine durchschnittliche Dauer von etwa 35 Minuten pro Session – realistisch im Rahmen von 10 Minuten bis zu einer Stunde. So können pro Tag zwischen 2.500 und 5.000 Log-Einträge erzeugt werden.

Zwar ließen sich innerhalb von fünf Tagen theoretisch bis zu 25.000 Log-Ereignisse erzeugen, jedoch ist der zeitliche Abstand zwischen den Sessions begrenzt. Ereignisse, die typischerweise erst nach längerer Zeit auftreten, fehlen somit in der Testbasis. Eine spätere Erweiterung ist möglich, jedoch wurde aufgrund der Bearbeitungsfrist beschlossen, die Generierung auf fünf aufeinanderfolgende Tage zu konzentrieren. Die Sessions werden dabei parallel ausgeführt.

Da sämtliche Tests auf demselben Rechner stattfinden, ist die Quell-IP-Adresse für alle Benutzer identisch. Dies könnte bei längerem Training als Anomalie erkannt werden. Daher wird die IP-Adresse vor der Weiterverarbeitung aus den Events entfernt. Diese Maßnahme reduziert zwar eine potenzielle Störquelle, verringert aber gleichzeitig auch die Datenvielfalt für das Modell.

Nichtsdestotrotz stellt dieses Verfahren eine vorteilhafte Methode zur Log-Generierung dar, da die erzeugten Ereignisse den realen Keycloak-Logs sehr nahekommen.



## 10.3 Mögliche Angriffsszenarien nach CVE und CWE

Common Vulnerabilities and Exposures (CVE) stellt eine Auflistung bzw. Datenbank aller bekannten Sicherheitslücken in einem System dar. Für Keycloak existieren ebenfalls zahlreiche dokumentierte Schwachstellen <sup>21</sup>.

CVEs bieten jedoch keine Lösungen an, sondern dienen der transparenten und neutralen Information über die Art, Ursache und mögliche Auswirkungen der jeweiligen Sicherheitslücke. Jede CVE erhält eine eindeutige Identifikationsnummer, eine Kurzbeschreibung sowie (sofern verfügbar) Angaben zur betroffenen Softwareversion. Die Dokumentation erfolgt weltweit durch Organisationen, Forschungseinrichtungen und unabhängige Sicherheitsforscher.

Common Weakness Enumeration (CWE) hingegen beschreibt systematisch bekannte Schwachstellenmuster (Weaknesses), die potenziell zu Sicherheitslücken führen können. Während CVEs auf bereits konkret ausgenutzte oder gefundene Schwachstellen hinweisen, beschreibt CWE eher abstrakte Klassen von Fehlern, wie z.,B. „unzureichende Rechteprüfung“ oder „unsichere Tokenverarbeitung“. Beide Kataloge sind essentiell, um systematische Sicherheitstests zu entwerfen und zu interpretieren.

Viele der genannten Probleme wurden zwar schon gelöst, jedoch häufen sich die Probleme von Jahr zu Jahr. Im Rahmen dieser Arbeit wurden mit Hilfe von Selenium gezielt sicherheitsrelevante Szenarien gegen Keycloak getestet und simuliert. Dabei lassen sich die folgenden Fälle konkreten CVE- und CWE-Einträgen zuordnen:

- **Privilege Escalation Test:** Bei diesem Test wurde überprüft, ob sich ein normaler Benutzer unbefugt Zugriff auf administrative Bereiche verschaffen kann. Die Logik entspricht dem bekannten Schwachstellenmuster **CWE-269: Improper Privilege Management**. In Bezug auf Keycloak wurde eine vergleichbare Schwachstelle mit **CVE-2019-10160** dokumentiert, bei der Benutzer durch unzureichende Prüfung ihrer Rollen Zugriff auf administrative Endpunkte erhalten konnten.
- **Token Manipulation Test:** In diesem Test wurde ein gültiges OAuth2 Access Token manipuliert (z. B. durch Abschneiden und Ändern von Zeichen), um zu prüfen, ob das System die Signaturprüfung korrekt durchführt. Die zugrundeliegende Schwachstelle ist unter **CWE-347: Improper Verification of Cryptographic Signature** klassifiziert. Eine konkrete Verwundbarkeit, bei der ein ähnlicher Fall auftrat, ist **CVE-2020-13957**, bei dem manipulierte Tokens fälschlicherweise akzeptiert wurden.

- **Denial of Service (DoS) Test:** Mittels zahlreicher, aufeinanderfolgender Anfragen wurde versucht, das System durch Last zu destabilisieren. Diese Art der Schwachstelle fällt unter **CWE-400: Uncontrolled Resource Consumption**. Eine reale CVE mit Bezug auf Keycloak ist **CVE-2021-41117**, bei der durch komplexe Gruppenrollenstrukturen ein ressourcenintensiver Zustand ausgelöst werden konnte, was ebenfalls zu einem DoS führen konnte.

Diese Tests orientieren sich an bekannten Sicherheitslücken und dienen dazu, die Robustheit des Systems unter realistischen Angriffsbedingungen zu validieren. Da CVE- und CWE-Datenbanken bewusst keine vollständigen Exploitbeschreibungen enthalten, musste im Rahmen der Simulation eine technische Interpretation der Schwachstellen erfolgen.

Des weiteren werden noch folgende Angriffsszenarien durchgespielt, welche auch bei den generierten Logs durchgespielt wurden:

- **Session Hijacking:** Missbrauch einer Session jemand anderes
- **Account Sabotage:** Viele, Wichtige Ressourcen werden gelöscht oder verändert oder zugegriffen
- **Brute Force Attacke (Brute-Force-Angriff)**

## 10.4 Selenium-Tests zur Erzeugung der Angriffsszenarien

Auch für die Selenium-Tests wurden die Angriffsszenarien wie in den generierten Logs umgesetzt. Für jeden Test, bzw. Angriffsszenario wurde ein Ablauf konstruiert. Für den Fall einer **”Privilege Escalation”** wurde folgender Fall konstruiert:

Im Ersten Fall versucht ein Nicht-Admin durch die Admin-Url zur Ansicht des Admins in Keycloak zu gelangen. Die würde schon fehlschlagen, wenn sich der Benutzer mit normalen Daten anmeldet.

Der zweite Fall ist, dass der Admin selbst einem Benutzer zu viele Rechte gibt, oder bzw. der Benutzer selbst ist, nur eben hat der Benutzer die Admin-Credentials in Erfahrung gebracht:

1. Man loggt sich in Keycloak direkt als der jeweilige Admin des zugehörigen Realms ein, (z.B. loggt sich Admin Anna in Realm Ubisoft ein).
2. Admin geht zum Tab users und klickt auf den jeweiligen Benutzer, welcher zu viele Rechte erhalten soll
3. Man klickt auf das Feld ”Role Mapping”

4. Es wird auf den Button 'Assign Role' gedrückt
5. Sensible Rollen werden dem Benutzer zugefügt (z.B. "realm-admin")
6. Man klickt auf den Button 'assign'

Für den Fall '**Account Sabotage**' wurde folgendes Vorgehen entschieden:

1. Man meldet sich an als Admin
2. Als Admin werden viele Instanzen auf einmal gelöscht: Ein Client mit all seinen Benutzern

Für den letzten Fall versucht ein gewöhnlicher Benutzer ständig Zugriff auf eine Ressource zu ergattern, die er nicht haben dürfte.

Für den Fall '**Session Hijacking**' wurde folgendes Vorgehen konstruiert:

1. Ein Angreifer loggt sich mit gültigen Zugangsdaten eines legitimen Benutzers in das System ein.
2. Der Angreifer liest die Session-Cookies bzw. die Sitzungstoken des eingeloggten Benutzers aus.
3. In einem zweiten Browser oder einer neuen Session werden diese Cookies übernommen und gesetzt.
4. Der Angreifer ruft mit der übernommenen Sitzung eine geschützte Ressource auf, die nur dem legitimen Benutzer zugänglich sein sollte.
5. Dadurch wird die aktive Benutzersitzung übernommen und unautorisierter Zugriff auf geschützte Bereiche ermöglicht.

Für den Fall "**Token Manipulation**" wurde folgendes Vorgehen konstruiert:

1. Ein Benutzer loggt sich mit gültigen Zugangsdaten im System ein.
2. Das gültige Access-Token des Benutzers wird ermittelt.
3. Das Token wird absichtlich verändert (z.B. durch Verfälschung der letzten Zeichen).
4. Mit dem manipulierten Token wird eine Anfrage an die geschützte Admin-API gesendet.
5. Das System sollte die Anfrage ablehnen und einen 401 Unauthorized-Status zurückgeben, um die Token-Manipulation zu erkennen.

- 11 Durchführung des Trainings
- 12 Ergebnisse
- 13 Diskussion
- 14 Fazit
- 15 Ausblick

# Literatur

- [1] Bundesamt für Sicherheit in der Informationstechnik (BSI). Die lage der it-sicherheit in deutschland 2023. <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2023.pdf>, 2024. Abgerufen am 10. Juni 2025.
- [2] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):1–58, 2009.
- [3] Phil Legg, Oliver Buckley, Michael Goldsmith, and Sadie Creese. Visualizing the insider threat: challenges and tools for identifying malicious user activity. In *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pages 1–8. IEEE, 2015.
- [4] Valerio Arnaboldi and Charles Morisset. Techniques for use in intrusion detection systems. In *Proceedings of the International Conference on Industrial Engineering and Operations Management (IEOM)*, Dubai, UAE, March 10–12 2020. IEOM Society International.
- [5] Louis Jannett, Maximilian Westers, Tobias Wich, Christian Mainka, Andreas Mayer, and Vladislav Mladenov. Sok: Sso-monitor — the current state and future research directions in single sign-on security measurements. In *2024 9th IEEE European Symposium on Security and Privacy (Euro&SP)*. IEEE, IEEE, 2024.
- [6] Chuan Zhou and Randy C. Paffenroth. Deep learning for anomaly detection: A review. *IEEE Access*, 9:5789–5813, 2021.
- [7] S. Arjunan and Others. A comparative study of deep neural networks and support vector machines for anomaly detection in cloud monitoring data. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 2024. Bewertung von AE, LSTM, IF und OCSVM auf Cloud-Daten.
- [8] Emre Demir and Cenk Karaoğlu. A comparative performance analysis of lstm autoencoder and timegpt models in time series anomaly detection. *Research-Gate*, 2024.
- [9] Tolga Ergen and Suleyman S Kozat. Unsupervised and semi-supervised anomaly detection with lstm neural networks. *arXiv preprint arXiv:1710.09207*, 2017. LSTM kombiniert mit OCSVM/SVDD für sequentielle Anomalieerkennung.

- [10] Zeineb Ghrib, Rakia Jaziri, and Rim Romdhane. Hybrid approach for anomaly detection in time series data. In *2020 IEEE International Conference on Intelligent Systems and Computer Vision (ISCV)*, pages 1–7. IEEE, 2020.
- [11] Wilson Chua, Arsenn Lorette Diamond Pajas, Crizelle Shane Castro, Sean Patrick Panganiban, April Joy Pasuquin, Merwin Jan Purganan, Rica Malupeng, Divine Jessa Pingad, John Paul Orolfo, Haron Hakeen Lua, and Lemuel Clark Velasco. Web traffic anomaly detection using isolation forest. *Informatics*, 11(4):83, 2024.
- [12] Yuanyuan Wei, Julian Jang-Jaccard, Wen Xu, Fariza Sabrina, Seyit Camtepe, and Mikael Boulic. Lstm-autoencoder-based anomaly detection for indoor air quality time series data. *IEEE Sensors Journal*, 23(4), 2023.
- [13] P. H. Tran, C. Heuchenne, and S. Thomassey. An anomaly detection approach based on the combination of lstm autoencoder and isolation forest for multivariate time series data. In *Developments of Artificial Intelligence Technologies in Computation and Robotics: Proceedings of the 14th International FLINS Conference (FLINS 2020)*, pages 589–596. World Scientific, 2020.
- [14] Du Nguyen Tran and Sylvestre Thomassey. Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. *International Journal of Information Management*, 57:102282, 2021.
- [15] Guozhi Yan, Yulong Zheng, and Chuan Lin. Detecting anomalous user behavior using an extended isolation forest algorithm: An enterprise case study. In *2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*, pages 115–121. IEEE, 2021.
- [16] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. 1998.
- [17] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.
- [18] Umberto Michelucci. An introduction to autoencoders. Online; accessed 2025-06-12, January 2022. <mailto:umberto.michelucci@toelt.ai>.
- [19] Damien Fourure, Maciej Mazurowski, Juan Amar, and Antoine Derreumaux. Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol. *arXiv preprint arXiv:2106.16020*, 2021.

- [20] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.
- [21] Davide Chicco and Giuseppe Jurman. The matthews correlation coefficient (mcc) is more informative than cohen’s kappa and brier score in binary classification assessment. *BMC Genomics*, 21(1):6, 2020.
- [22] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition*, pages 3121–3124. IEEE, 2010.
- [23] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019. Tutorial Paper.
- [24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [25] Xiaoli Wei, Jiawei Liu, and Lei Zhang. Lstm autoencoder-based anomaly detection for indoor air quality data. *arXiv preprint arXiv:2204.06701*, 2022.
- [26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.
- [27] Kun-Lun Li, Hou-Kuan Huang, Sheng-Feng Tian, and Wei Xu. Improving one-class svm for anomaly detection. In *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, pages 2–5, Wan, November 2003. School of Computer & Information Technology, Northern Jiaotong University. Beijing, China, 100044. Faculty of Mathematics and Computer Science, Hebei University, Baoding, China, 071002.
- [28] Abdenour Bounsiar and Michael G. Madden. One-class support vector machines revisited. *Unpublished manuscript*, 2025. Discusses geometric motivation for OCSVM with Gaussian kernels.
- [29] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.

- [30] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.
- [31] Guo Pu, Lijuan Wang, Jun Shen, and Fang Dong. A hybrid unsupervised clustering-based anomaly detection method. In *Proceedings of the 2021 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 2021.
- [32] Angad Pal Singh and Ankit Sharma. A systematic literature review on insider threats, 2022.



# Fußnotenverzeichnis

- 1 <https://www.keycloak.org/>
- 2 IAM steht für *Identity and Access Management* und beschreibt Systeme zur Verwaltung digitaler Identitäten und Zugriffskontrollen. Vgl. z.B. R. Müller: *Identity Management: Konzepte, Technologien, Standards und Praxis*, 2. Auflage, Springer Vieweg, 2016.
- 3 Single Sign-On ermöglicht einem Nutzer den Zugriff auf mehrere Anwendungen mit nur einer Anmeldung. Siehe: B. G. Blakley et al.: *An Introduction to Identity Federation*, IBM Redbooks, 2007. <https://www.redbooks.ibm.com/abstracts/sg247208.html>
- 4 <https://www.forbes.com/sites/quora/2017/09/15/what-are-the-biggest-challenges-facing-the-cyber-world/>
- 5 <https://medium.com/sciforce/adversarial-attacks-explained-and-how-to-defend-ml-models-against-them-d76f7d013b18>
- 6 <https://nvd.nist.gov/vuln/detail/CVE-2024-4629>
- 7 <https://www.cvedetails.com/cve/CVE-2025-3501>
- 8 <https://github.com/lukasruff/Deep-SVDD-PyTorch>
- 9 Definitionen der Metriken: **Accuracy** =  $\frac{TP+TN}{TP+TN+FP+FN}$  gibt den Anteil korrekt klassifizierter Beispiele an. **Precision** =  $\frac{TP}{TP+FP}$  misst den Anteil korrekt als positiv klassifizierter Beispiele an allen als positiv vorhergesagten. **Recall** =  $\frac{TP}{TP+FN}$  zeigt, wie viele der tatsächlichen Positiven korrekt erkannt wurden. **F1-Score** =  $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$  ist das harmonische Mittel von Precision und Recall. Dabei stehen TP, TN, FP, FN für True Positives, True Negatives, False Positives und False Negatives.
- 10 Überanpassung des Modells an die Trainingsdaten, wodurch die Generalisierbarkeit auf neue Daten leidet.
- 11 Scikit-learn: Machine Learning in Python. Offizielle Dokumentation, verfügbar unter: <https://scikit-learn.org/stable/>
- 12 Siehe Definition laut NVIDIA: <https://www.nvidia.com/en-us/data-center/dgx-systems/>
- 13 Siehe technische Spezifikationen: <https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html>
- 14 <https://www.wildfly.org/>
- 15 <https://www.keycloak.org/server/logging>
- 16 <https://docs.jboss.org/process-guide/en/html/logging.html>
- 17 <https://docs.spring.io/spring-data/jpa/reference/index.html>
- 18 <https://github.com/keycloak/keycloak/blob/main/model/jpa/src/main/java/org/keycloak/events/jpa/JpaAuthProvider.java>
- 19 <https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml>
- 20 <https://www.keycloak.org/docs-api/latest/rest-api/index.html>
- 21 <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=keycloak>