

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

Danksagung

Ich will mich hiermit bei meine betrieblichen Betreuer Eric Hämmerle bedanken, sowie meinen Vorgesetzten, dass diese Arbeit ermöglicht wurde. Ebenfalls bedanke ich mich bei meiner Familie und meine Freunden für die seelische Unterstützung und Motivation, die dazu geführt hat, diese Arbeit fertig zu stellen. Ebenfalls bedanke ich mich bei meinem Betreuer der DHBW, Herr Dr. Dietzsch für das Feedback.

Abstract

This paper examines the detection of insider attacks in Keycloak logs, focusing on anomalies in user behavior. To this end, a hybrid model based on an LSTM autoencoder (LSTM-AE) was developed, which was combined with one of three anomaly detection algorithms: Isolation Forest (IF), One-Class SVM (OCSVM), and DBSCAN. The aim was to compare the performance of the various hybrid models in identifying anomalies. Three different attack scenarios were simulated to represent realistic insider threats. The results show that the LSTM-AE performs particularly well in combination with Isolation Forest, followed by the combination with One-Class SVM. The model with DBSCAN achieved significantly weaker results in comparison. The study underscores the effectiveness of LSTM-AE hybrid models in anomaly detection in authentication logs and provides indications as to which algorithms are particularly suitable for practical use. Autoencoder: Consists of an encoder and a decoder. It is also a neural network. It compresses (in the encoder) and reconstructs (in the decoder) data to discover certain variables or anomalies.

Translated with DeepL.com (free version)

Zusammenfassung

In dieser Arbeit wird die Erkennung von Insider-Angriffen in Keycloak-Logs untersucht, wobei Anomalien im Benutzerverhalten im Fokus stehen. Dazu wurde ein hybrides Modell auf Basis eines LSTM-Autoencoders (LSTM-AE) entwickelt, das jeweils mit einem der drei Anomalieerkennungsalgorithmen Isolation Forest (IF), One-Class SVM (OCSVM) und DBSCAN kombiniert wurde. Ziel war es, die Leistungsfähigkeit der verschiedenen Hybridmodelle bei der Identifikation von Anomalien zu vergleichen. Es wurden drei unterschiedliche Angriffsszenarien simuliert, um realistische Insider-Bedrohungen abzubilden. Die Ergebnisse zeigen, dass das LSTM-AE in Kombination mit dem Isolation Forest besonders gut abschneidet, gefolgt von der Kombination mit One-Class SVM. Das Modell mit DBSCAN erzielte im Vergleich deutlich schwächere Ergebnisse. Die Studie unterstreicht die Effektivität von LSTM-AE-Hybridmodellen in der Anomalieerkennung von Authentifizierungs-Logs und liefert Hinweise, welche Algorithmen für die Praxis besonders geeignet sind.

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. Für sprachliche Ausbesserungen wurde die KI ChatGPT ¹verwendet.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

1	Einleitung	20
1.1	Benutzerverhalten und Anomalien	20
1.2	Kategorisierung von Anomalien	20
1.3	Regelbasierte Methoden und weitere Maßnahmen in Keycloak	23
1.4	Probleme: Grenzen der Sicherheitsmaßnahmen in Keycloak	23
1.5	Motivation: Maschinelle Lernmodelle als zukünftige Sicherheitsmaß- nahme	24
1.6	User (and Entity) Behavior Analytics (U(E)BA)	24
2	Forschungsstand, Herausforderungen und Hypothesen	25
2.1	Bisheriger Forschungsstand	25
2.2	Problematik und Begrenzung von Modellen	26
2.3	Hypothese	27
3	Methoden und Störfaktoren	29
3.1	Metriken	29
3.2	Andere, nicht angewandte Metriken	31
3.3	Störfaktoren	32
4	Angewandte Technologien	33
4.1	Scikit-learn Learn	33
4.2	Keras	33
4.3	TensorFlow	34
4.4	Deep GPU Xceleration (DGX)	34
5	Angewandte Technologie: Keycloak	34
5.1	Terminologien in Keycloak	35
5.2	Komponenten der Log-Erzeugung	37
5.3	Log-Dateien in Keycloak	40
5.4	Event-Logs	41
5.5	Aufbau der Event-Logs	41
6	Theoretischer Hintergrund der Modelle	43
6.1	LSTM	43
6.2	Autoencoder	45
6.3	Isolation Forest	47
6.4	One-Class SVM	48

6.5	DBSCAN	49
6.6	Alternative Algorithmen	50
7	Implementierung	51
7.1	Hybridmodelle mit LSTM-AE	51
7.1.1	Implementierung: Isolation Forest	56
7.1.2	Implementierung: OCSVM	56
7.1.3	Implementierung: DBSCAN	57
7.2	Alternative Architekturen	57
8	Möglichkeit 1: Generierung der Logs	59
8.1	Erster Entwurf der Angriffsfälle	59
8.2	Automatische Erstellung der Keycloak-Logs	60
8.3	Problematik dieser Lösung	62
9	Möglichkeit 2: Datenbeschaffung als Alternative	62
9.1	Anbindung der Keycloak-API	63
9.2	Selenium-Tests zur Erzeugung der Logs	64
9.3	Mögliche Angriffsszenarien basierend auf Common Vulnerabilities and Exposures (CVE) und Common Weakness Enumeration (CWE) . . .	68
9.4	Angriffsszenarien	69
9.4.1	Angriffsszenario: Versuch einer Privilege Escalation über die Admin REST API	70
9.4.2	Angriffsszenario: Account-Sabotage durch privilegierten Benutzer	71
9.4.3	Angriffsszenario: Brute-Force-Angriff über die Web-Oberfläche	72
10	Durchführung des Trainings	73
10.1	Änderung der Timestamps durch Unix Timestamp	76
10.2	Testfälle	77
11	Ergebnisse	79
11.1	Ergebnisse mit der Sequenzlänge 10	79
11.1.1	Testlauf 1	79
11.1.2	Testlauf 2	80
11.1.3	Testlauf 3	81
11.1.4	Ergebnisse mit der Sequenzlänge 25	82
11.1.5	Testdurchlauf 1	82
11.1.6	Testlauf 2	83

11.1.7 Testlauf 3	84
12 Diskussion	85
13 Fazit	87
14 Ausblick	88

Abbildungsverzeichnis

1	Protokolle für IdPs in Keycloak	36
2	Beispielhafte Konfiguration von Authentication Flows in Keycloak . .	37
3	Beispiel: Aktivierung von Quarkus-Logging in der Keycloak-Konfiguration	38
4	Log-Ausgaben aus einer Quarkus-basierten Keycloak-Instanz	39
5	Beispiel Events nach Eventtyp	42
6	Visuelle Darstellung der Funktion des AEs, Quelle: https://www.researchgate.net/figure/Structure-of-the-autoencoder_fig1_365074431 , Letzter Zugriff am: 15.07.2025 . .	46
7	Visuelle Darstellung der Funktion des AEs, Quelle: https://www.researchgate.net/publication/350551253/figure/fig1/figure-pdf/350551253-Detection-using-Isolation-Forest-18_fig3_350551253.pdf , Letzter Zugriff am: 15.07.2025	47
8	Visuelle Darstellung OCSVM. Quelle: https://www.mdpi.com/2076-3417/13/3/1734 , Letzter Zugriff am: 15.07.2025	49
9	Beispiel DBSCAN-Cluster, Quelle: https://www.baeldung.com/wp-content/uploads/sites/4/2023/04/clusters.png , Letzter Zugriff am: 15.07.2025	49
10	Skizze aller Hybridmodelle	52
11	Gesamtarchitektur LSTM-AE	54
12	Ausschnitt der generierten Logs	61
13	Daten, die an den Keycloak-Server gesendet werden müssen	63
14	Beispielhafte Darstellung der generierten Benutzer für den Testfall . .	65
15	Benötigte Rollen für den Client Nintendo	66
16	Aktuelle Uhrzeit nach dem Unix-System	76
17	Überblick der Konstanten Werte und der unabhängigen Variable . . .	78

Tabellenverzeichnis

1	Bestimmte Parameter der Modellkonfiguration	52
2	Ergebnisse für IsolationForest	79
3	Ergebnisse für One-Class SVM	80
4	Ergebnisse für DBSCAN	80
5	Ergebnisse für IsolationForest	80
6	Ergebnisse für One-Class SVM	81
7	Ergebnisse für DBSCAN	81
8	Ergebnisse für IsolationForest	81
9	Ergebnisse für One-Class SVM	82
10	Ergebnisse für DBSCAN	82
11	Ergebnisse für IsolationForest, Sequenzlänge 25	82
12	Ergebnisse für One-Class SVM, Sequenzlänge 25	83
13	Ergebnisse für DBSCAN, Sequenzlänge 25	83
14	Ergebnisse für IsolationForest, Sequenzlänge 25	83
15	Ergebnisse für One-Class SVM, Sequenzlänge 25	84
16	Ergebnisse für DBSCAN, Sequenzlänge 25	84
17	Ergebnisse für IsolationForest, Sequenzlänge 25	84
18	Ergebnisse für One-Class SVM, Sequenzlänge 25	85
19	Ergebnisse für DBSCAN, Sequenzlänge 25	85

Abkürzungsverzeichnis

ABAC: Attribute-Based Access Control

AE: Autoencoder

AUC-PR: Area Under the Precision and Recall Curve

AUC-ROC: Area Under the ROC Curve

CVE: Common Vulnerability and Exposures **CWE:** Common Weakness Enumeration **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

SPI: Serial Peripheral Interface

SSO: Single-Sign-On

UBA: User Behavior Analytics

UEBA: User and Entity Behavior Analytics

Glossar

ν Parameter der One-Class SVM, der den Anteil an erwarteten Ausreißern im Trainingsdatensatz steuert. 46

gamma Parameter der RBF-Kernel-Funktion bei der One-Class SVM; legt fest, wie stark ein einzelner Trainingspunkt die Entscheidungsgrenze beeinflusst. 47

ABAC Zugriffskontrolle basierend auf Benutzerattributen wie Rolle, Standort, Zeit etc.. 16

Access Logs Protokolle von HTTP-Zugriffen und Statuscodes. Werden häufig auf Webserver- oder Proxy-Ebene erstellt, nicht direkt in Keycloak.. 34

Account Sabotage Ein Szenario, in dem ein Benutzer absichtlich Ressourcen löscht, verändert oder auf sie zugreift, um einem System oder einem anderen Benutzer zu schaden. 59

Admin Logs Logs, die administrative Operationen wie CREATE, UPDATE oder DELETE dokumentieren. Enthalten z. B. das Feld *operationType*.. 35

AUC-PR Area Under the Precision and Recall Curve – misst die Leistung eines Modells im Umgang mit unausgeglichene Klassen.. 24

AUC-ROC Area Under the Receiver Operating Characteristic Curve – misst die Trennschärfe eines Klassifikationsmodells.. 23

Audit-Logs Logs zur Nachvollziehbarkeit administrativer Änderungen. In Keycloak Teil der Admin-Logs.. 34

Authentication Flow Ablauf, der festlegt, wie sich Benutzer oder Clients bei Keycloak authentifizieren.. 30

Authorization Code Grant OAuth2-Grant-Typ, bei dem der Benutzer über einen Code authentifiziert wird, der vom Client gegen ein Token eingetauscht wird.. 29

authType Ein Attribut in Keycloak-Logs, das angibt, über welchen Mechanismus sich ein Benutzer authentifiziert hat (z. B. password, token). 50

Balanced Accuracy Durchschnitt aus Sensitivität (Recall) und Spezifität; besonders geeignet bei unausgegleichen Klassenverhältnissen. 24

- Botnetz** Ein Zusammenschluss kompromittierter Geräte (Bots), die ferngesteuert werden, oft zur Durchführung koordinierter Angriffe wie DDoS oder zur Verbreitung von Malware.. 13
- Bottleneck** Engste Stelle eines Autoencoders, in der die komprimierte Datenrepräsentation gespeichert wird. 39
- Brute-Force-Angriff** Angriffsmethode, bei der Passwörter oder Zugangsdaten durch systematisches Ausprobieren aller möglichen Kombinationen erraten werden.. 59
- Category (Logging)** Bezeichnung des Moduls oder der Komponente, aus der ein Log stammt (z. B. `org.keycloak.events`).. 36
- Client** Anwendung oder Dienst, der Keycloak zur Authentifizierung und Autorisierung verwendet.. 28
- Client Credentials Grant** OAuth2-Grant-Typ, bei dem sich ein Client ohne Benutzerinteraktion authentifiziert.. 30
- CLIENT LOGIN** Event-Typ in Keycloak, der bei Anmeldung über einen Drittanbieter-Client (z. B. via OAuth2) generiert wird.. 37
- Client-Scope** Legt fest, welche Benutzerdaten in ein Token aufgenommen werden, wenn sich ein Benutzer über einen Client authentifiziert.. 30
- CODE TO TOKEN** Keycloak-Event, das auftritt, wenn ein Authentifizierungscode erfolgreich gegen ein Token eingetauscht wird.. 37
- Common Vulnerabilities and Exposures (CVE)** Ein öffentliches Verzeichnis bekannter Sicherheitslücken in Soft- und Hardware, das durch eindeutige Identifikationsnummern, Kurzbeschreibungen und betroffene Versionen strukturiert ist. 6, 58
- Common Weakness Enumeration (CWE)** Ein Katalog von typischen Schwachstellenmustern (Weaknesses), die potenziell zu Sicherheitslücken führen können, z. B. unzureichende Authentifizierung oder unsichere Verarbeitung von Daten. 6, 58
- CRUD** Abkürzung für Create, Read, Update, Delete; beschreibt die grundlegenden Operationen auf Datenbanken oder Entitäten in IT-Systemen. 28

Custom Logs Benutzerspezifische oder erweiterte Logs, die über Keycloak-Extensions (z. B. SPIs) erzeugt werden.. 35

DBSCAN Density-Based Spatial Clustering of Applications with Noise – Clustering-Verfahren für dichte Datenregionen.. 20

Decoder Teil eines Autoencoders, der versucht, aus der komprimierten Repräsentation die Originaldaten zu rekonstruieren. 38

Denoising Autoencoder Autoencoder, der lernt, verrauschte Eingaben zu bereinigen und zu rekonstruieren. 39

DGX Deep GPU Xceleration – NVIDIA-Hardwareplattform zur KI-Beschleunigung.. 5, 28

Encoder Teil eines Autoencoders, der Daten in eine niedrigdimensionale Repräsentation komprimiert. 38

Eps Radius in DBSCAN, der zur Definition der Nachbarschaft eines Punktes dient. 20

Event-Logs Protokolle von Benutzer- und Administratoraktionen innerhalb von Keycloak. Enthalten Informationen zu Zeitpunkt, Nutzer-ID, Realm usw.. 34

Event-Typ Bezeichnet die Art des ausgelösten Ereignisses, z. B. LOGIN, LOGOUT, TOKEN_REFRESH etc.. 32

Feature Space (Merkmalsraum) Der Raum, in dem Datenpunkte durch Merkmale abgebildet werden, häufig hochdimensional. 42

Filebeat Lightweight-Agent, der Logdateien überwacht und deren Inhalte an zentrale Systeme wie Logstash weiterleitet.. 34

Grant Type Verfahren, mit dem ein Client oder Benutzer bei einem Authentifizierungsserver ein Token anfordert.. 29

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise, eine hierarchische Erweiterung von DBSCAN. 44

Hidden State Interner Zustand eines RNNs, der Informationen über vorherige Zeitschritte speichert. 38

- Hybridmodell** Ein Modell, das zwei oder mehr unterschiedliche Verfahren kombiniert, z. B. LSTM-Autoencoder mit einem klassischen ML-Algorithmus wie Isolation Forest. 20
- IAM** Identity and Access Management – Verwaltung von Identitäten und Zugriffsrechten.. 16
- Identity Provider** Externer Anbieter, der Authentifizierungsdienste bereitstellt. 29
- Impersonate** Ein Event in Keycloak, bei dem ein Administrator die Rolle eines anderen Benutzers übernimmt; wird zur Fehlersuche oder zu Testzwecken verwendet. 57
- Insider Threat** Sicherheitsbedrohung, die von Personen innerhalb einer Organisation ausgeht, bspw. durch missbräuchliches Verhalten von Mitarbeitenden mit legitimen Zugriffsrechten.. 48
- Isolation Tree** Binärer Baum aus dem Isolation Forest-Verfahren zur Isolierung von Datenpunkten. 41
- JBoss Logging** Logging-API aus dem JBoss-Ökosystem. Bietet eine Abstraktionsschicht über verschiedene Logging-Backends wie log4j, JUL oder slf4j.. 31
- K-Means** Clustering-Verfahren, das Daten in vordefinierte Cluster basierend auf Mittelwerten aufteilt. 44
- Keycloak** Open-Source Identity-Management-Lösung mit Funktionen wie SSO, RBAC und Brute-Force-Schutz.. 5, 16
- LOF** Local Outlier Factor, ein Verfahren zur Anomalieerkennung über Vergleich lokaler Dichten. 44
- Log-Level** Gibt die Kritikalität oder den Zweck eines Log-Eintrags an (z. B. INFO, ERROR, FATAL, DEBUG).. 36
- Log4j** Beliebtes Java-Logging-Framework, bekannt durch seine Flexibilität. Wegen Sicherheitslücken (bspw. Log4Shell) kritisch betrachtet.. 32
- LOGIN ERROR** Event-Typ, der bei einem fehlgeschlagenen Benutzer-Login generiert wird.. 37

- LSTM** Long Short-Term Memory – Neuronales Netz zur Verarbeitung von Sequenzdaten mit Langzeitspeicher.. 20
- manage-users** Keycloak-Rolle, die es erlaubt, Benutzer in einem Realm zu verwalten (z. B. anlegen, bearbeiten, löschen). 49
- Matthews Correlation Coefficient (MCC)** Eine robuste Metrik, die alle Werte der Konfusionsmatrix berücksichtigt; besonders geeignet bei unausgeglichene Datensätzen. 24
- MinPts** Minimale Anzahl an Punkten innerhalb des Eps-Radius, damit ein Punkt als Kernpunkt eines Clusters gilt. 20
- Noise** Punkte in DBSCAN, die keinem Cluster zugeordnet werden können. 44
- OAuth 2.0** Autorisierungs-Framework, das sicheren Zugriff auf geschützte Ressourcen ermöglicht.. 29
- OCSVM** One-Class Support Vector Machine – Algorithmus zur Anomalieerkennung mit einseitiger Klassifikation.. 42
- OpenID Connect (OIDC)** Auth-Protokoll, das auf OAuth 2.0 basiert und die Identität von Nutzern sicher überträgt.. 29
- operationType** Feld in Admin-Logs, das die Art der Admin-Aktion angibt, z. B. CREATE, UPDATE, DELETE oder ACTION.. 37
- Privilege Escalation** Ein Angriffsszenario, bei dem sich ein Benutzer unbefugt höhere Rechte verschafft, z. B. um auf administrative Funktionen zuzugreifen. 58
- Quarkus** Modernes Java-Framework, optimiert für containerisierte und cloud-native Anwendungen. Ab Keycloak Version 17 als Basisplattform genutzt.. 31
- RBAC** Role-Based Access Control – Zugriffskontrolle auf Basis definierter Rollen.. 16
- Realm** Isolierte Umgebung in Keycloak, in der Benutzer, Rollen und Clients separat verwaltet werden.. 28
- realm-admin** Eine Administratorrolle in Keycloak, die vollständige administrative Rechte innerhalb eines bestimmten Realms gewährt. 49

- Redirect-URL** Die URL, auf die Benutzer nach erfolgreicher Authentifizierung in Keycloak weitergeleitet werden; wichtig für Sicherheit und Funktionsfähigkeit. 55
- Rekonstruktionsfehler** Differenz zwischen Originaldaten und den durch einen Autoencoder rekonstruierten Daten. 39
- Remote Logging** Versand von Logs an externe Systeme oder Services, z. B. über Syslog, HTTP oder Filebeat.. 34
- RNN** Recurrent Neural Network – Neuronales Netz zur Verarbeitung zeitabhängiger Daten.. 38
- SAML** Security Assertion Markup Language – XML-basiertes Protokoll für Authentifizierung und Autorisierung.. 29
- Scikit-learn** Eine weit verbreitete Python-Bibliothek für maschinelles Lernen mit Tools für Klassifikation, Regression, Clustering und Modellbewertung. 5, 27
- Security Logs** Protokollieren sicherheitsrelevante Ereignisse wie z. B. verdächtige Logins, Policy-Verletzungen oder Blockierungen.. 35
- Selenium** Ein Framework zur Automatisierung von Webbrowser-Interaktionen, häufig für Testzwecke genutzt. 52
- Server-Logs** Logs, die Betriebsstatus, Fehler und Debugging-Informationen des Keycloak-Servers dokumentieren.. 34
- Service Account** Konto, das einem Client gehört und ohne Benutzerinteraktion für automatisierte Vorgänge eingesetzt wird.. 31
- Session Hijacking** Ein Angriff, bei dem ein Angreifer die aktive Sitzung eines anderen Benutzers übernimmt, z. B. durch Abfangen von Session-Cookies. 59
- SLF4J** Simple Logging Facade for Java – eine Logging-Fassade, die unterschiedliche Logging-Backends wie log4j oder logback unterstützt.. 32
- Spoofing** Täuschungstechniken, bei denen sich ein Angreifer als vertrauenswürdige Instanz ausgibt, um Zugang zu Informationen oder Systemen zu erlangen.. 16
- SSO** Single Sign-On – Einmalige Authentifizierung für den Zugriff auf mehrere Anwendungen.. 16

Syslog Standardprotokoll zur Übertragung von Logmeldungen in Netzwerkkumgebungen, z. B. an zentrale Logserver.. 32

System.out / **System.err** Standard-Ausgabe-/Fehlerstreams in Java. Nicht strukturiert und für produktive Logs ungeeignet.. 32

timestamp Zeitmarke, zu der ein Event im Log registriert wurde. Wichtige Information für zeitbasierte Analysen.. 35

TOKEN_REFRESH Keycloak-Event-Typ, der beim Aktualisieren eines Zugriffstokens generiert wird.. 36

UBA User Behavior Analytics – Analyse des Benutzerverhaltens zur Erkennung von Anomalien und Sicherheitsbedrohungen.. 17

UEBA User and Entity Behavior Analytics – Verfahren zur Analyse des Verhaltens von Benutzern und technischen Entitäten mit dem Ziel, sicherheitsrelevante Anomalien zu identifizieren.. 18

User Logs Logs, die Benutzeraktionen ohne Adminrechte erfassen. Typische Eventtypen sind LOGIN, LOGOUT oder TOKEN-Operationen.. 35

Variational Autoencoder (VAE) Autoencoder-Variante, die Wahrscheinlichkeitsverteilungen in der Bottleneck-Schicht modelliert, häufig für generative Modelle. 39

view-events Keycloak-Rolle, die es erlaubt, Event-Logs eines Realms einzusehen. 55

WildFly Java-Anwendungsserver, früher Basis von Keycloak. Verwendet JBoss Logging und bietet umfangreiche Java EE-Funktionalität.. 31

Zero-Day-Angriff Angriff auf eine bislang unbekannte Sicherheitslücke, für die noch kein Patch existiert.. 17

1 Einleitung

Laut dem BSI-Lagebericht 2024 ist die Zahl von Cyberangriffen auf Unternehmen weiter gestiegen und seien so hoch wie nie zuvor [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. 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, S.2]. Systemweite Anomalien betreffen technische Parameter und Prozesse im gesamten System, wie etwa plötzliche Veränderungen im Datenverkehr, unerwartete Systemlast oder ungewöhnliche Netzwerkverbindungen [2, S.2].

Neben systemweiten Anomalien – etwa im Netzwerkverkehr oder bei Gerätezuständen – gewinnen zunehmend verhaltensbezogene Auffälligkeiten an Bedeutung. In dieser Arbeit werden Anomalien, die sich auf das Benutzerverhalten beziehen bewusst als *verhaltensbasierte Anomalien* bezeichnet. Diese beziehen sich auf das individuelle Verhalten einzelner Nutzer.

1.2 Kategorisierung von Anomalien

Nach Chandola et Al. [2, S.7] 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, bspw. 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 bis um Mitternacht nach einem Bug suchen. Wenn dies in einem Unternehmen der Standardfall ist, würden dies keine Anomalien darstellen sondern den üblichen Standardfall. Unübliche Zugriffsrechte auf sensible Daten können Anomalien sein können, 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.

Alle aufgezählten Aspekte sind in *Intrusionsangriffen* und *Insider Threat Attacks* vorhanden. Intrusionsangriffe sind von außen kommende Angriffe, welche versuchen in das Innere eines Systems zu gelangen, z.B. Brute-Force Angriffe. Insider Threat Attacks können im System selbst schon auftreten, bspw. könnte der Angreifer ein Mitarbeiter im Unternehmen selbst sein. Insider-Attacken äußern sich nach Legg et Al. [3, S.2] in den genannten Aspekten: Benutzer zeigen abweichendes Verhalten (z.B. loggt er sich viel früher ein als üblich) oder ein Benutzer zeigt ein bestimmtes 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.

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) hin-

deuten können. In regulären Betriebsabläufen sind IP-Adressen in der Regel geografisch, organisatorisch oder netz-technisch 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 z.B. auf 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. Diese Arbeit fokussiert sich auf die 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 ², ein IAM-Tool³, das unter anderem SSO (Single Sign-On) bietet⁴.

Keycloak stellt schon vorgefertigte Sicherheitsmechanismen zur Verfügung, die in größtenteils 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

Dennoch stoßen viele Sicherheitsfunktionen von Keycloak praktisch 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, hybride, 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 in den 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 ⁵ 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. In dieser Arbeit werden daher größtenteils UBA-Methoden angewandt.

Zur Echtzeitüberwachung werden dabei unter anderem Logdaten analysiert, um anhand zusammenhängender Informationen anomales Benutzerverhalten zu identifizieren.

2 Forschungsstand, Herausforderungen und Hypothesen

2.1 Bisheriger Forschungsstand

Zwar bietet Keycloak regelbasierte Sicherheitsmaßnahmen an- diese beruhen jedoch größtenteils auf vordefinierten Bedingungen. Dadurch können sie lediglich bekannte Muster erkennen und sind wenig flexibel gegenüber neuen oder komplexeren Angriffstechniken [4, S. 3]. Angreifende nutzen gezielt sogenannte *Adversarial Attacks* ⁶, 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* ⁷. 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 de-

ren Erkennung und Behebung stets Zeit. Neue Verwundbarkeiten entstehen kontinuierlich – etwa durch das Umgehen von Zertifikatsprüfungen⁸, 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 sind bislang kaum erforscht. Dabei spielen sie eine zunehmende Rolle, insbesondere zur Erkennung von *Insider Threats* durch ML-Techniken, die im Keycloak-Kontext bisher nicht aktiv adressiert oder dokumentiert sind. Die Erforschung dieser Lücken ist sowohl aus wissenschaftlicher als auch wirtschaftlicher Sicht lohnenswert.

Die obige Darstellung zeigt eine vereinfachte, aber repräsentative Einschätzung des aktuellen Sicherheitsniveaus in Keycloak.

Nun 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 [5], wobei OCSVMs sogar Anomalien präziser erkennen können als Autoencoder [6]. 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 [7]. Eine Studie von Ergen et Al. [8] zeigt, dass die Kombination von LSTM-basierten Architekturen mit OC-SVM oder SVDD⁹ zu signifikant verbesserten Erkennungsraten bei der Anomalieerkennung führt – insbesondere bei sequenziellen Daten. Andere Arbeiten wie die von Ghrib et Al. [9] untersuchen ebenfalls hybride Ansätze mit Isolation Forest- die Rechenzeit würde mit Isolation Forest kombiniert die Rechenzeit reduzieren. Auch DBSCAN wird erfolgreich zur Cluster-basierten Anomalieerkennung eingesetzt, insbesondere bei unstrukturierten oder verrauschten Daten. Insgesamt zeigt sich, dass der hybride Ansatz dieser Verfahren vorteilhaft 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 Iso-

lation Forest ist oft sensitiv gegenüber der Wahl der Parameter und muss je nach Architektur angepasst werden, wie auch in einer Studie nach Chua et Al. diskutiert wurde [10]. 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.

Hybridmodelle haben ebenfalls 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) [11]. Andere Studien zeigen, dass sich LSTM-AE mit IF ein gutes Modell bauen lässt, welches Anomalien erkennen kann und ebenfalls eine hohe Accuracy erzielt [12].

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:

Es wird erwartet, dass der LSTM-Autoencoder in Kombination mit Isolation Forest unter unterschiedlichen Sequenzlängen bessere Erkennungsmetriken erzielt als die Kombinationen des LSTM-Autoencoders mit DBSCAN bzw. OCSVM.

Die zugehörige Nullhypothese lautet:

Unter verschiedenen Sequenzlängen zeigt der LSTM-Autoencoder in Kombination mit Isolation Forest keine besseren Erkennungsmetriken als die Kombinationen mit DBSCAN oder OCSVM.

Folgende Hybridmodelle werden implementiert:

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

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

Es wird davon ausgegangen, dass LSTM-AE mit IF am besten abschneidet, weil dies in den erwähnten Studien hervorgehoben wird. Zusätzlich wurden die Einzelmodelle Isolation Forest, OCSVM und DBSCAN auf den Keycloak-Daten explorativ angewendet. Diese Evaluation ist nicht Bestandteil der zentralen Vergleichsstudie, dient jedoch als Ergänzung zur Einordnung der Ergebnisse der hybriden Modelle.

Sie ermöglicht einen informellen Eindruck davon, wie sich gängige Einzelverfahren im gegebenen Kontext verhalten, ohne dass daraus direkte Schlussfolgerungen im Sinne der zentralen Forschungsfrage abgeleitet werden.

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.

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 [13]. 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 [14], 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 komplexe Hybridmodellen miteinander verglichen werden.

3 Methoden und Störfaktoren

3.1 Metriken

Typischen Verfahren wie die Berechnung der Precision, des Recalls und des F1-Scores¹⁰ werden angewandt.

Die Accuracy wird nicht berechnet aus den folgenden Gründen:

Eine Studie weist aber daraufhin, dass das Öfteren diese 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 [15]. 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 Klassen 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.

Es muss ebenfalls erwartet werden, dass es in dieser Arbeit zu einem Klassenungleichgewicht kommen kann, da Anomalien in den Testdaten sehr wenig auftreten werden, wie es in der Realität erwartet wird. Dies ist jedoch nicht natürlich und dessen Problematik ist relativ und unterscheidet sich von Fall zu Fall [16].

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, weil der Score alleine nicht viel aussagte [17]. Es wird kritisiert, 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 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 [17]. 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 Accuracy beeinflussen. Gleichzeitig wird kritisiert, dass viele Studien auf Metriken zurückgreifen, die nicht ausreichend aussagekräftig sind.

Generell ist die Kritik, dass nur ein 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:

- **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 zurück zuführen lässt, dass beide Verfahren sich schon im Algorithmus ähnlich sind [18].

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 und dass es zu einer optimistischen Schätzung, wenn ein verzerrter Klassifikator auf einem unausgewogenen Datensatz getestet wird, führt. [19, 20].

Außer diesen Metriken wurde noch überlegt, einen sogenannten Cross-Validation-Test durchzuführen. Cross-Validation ist keine „Metrik“ wie z.B. der 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 gebliebene 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. Zudem soll sie die Ergebnisse nach jedem Training des k -Datensatzes zeigen, damit sich erkennen lässt, 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.

Ein weiterer Störfaktor ist die Parameterwahl, auf die im Abschnitt zur Umsetzung noch näher eingegangen wird. Parameter wie bspw. die Anzahl der Epochen oder auch besonders die Sequenzlänge beeinflussen das Verhalten und die Ergebnisse 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*¹¹ 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, welches 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 Daten häufig begrenzt verfügbar sind, stehen meist nur wenige Trainings- und Testdaten zur Verfügung. LSTMs gelten umgangssprachlich 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 beeinflusst.

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.

Zudem kann noch die Anzahl der Features das Ergebnis ebenfalls beeinflussen, dieses Problem wird in den späteren Kapiteln noch diskutiert wird.

4 Angewandte Technologien

4.1 Scikit-learn Learn

Scikit-learn¹² 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 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*.

4.2 Keras

Keras¹³ ist eine benutzerfreundliche, modulare und erweiterbare Open-Source-Bibliothek für die Entwicklung von Deep-Learning-Modellen in Python. Sie bietet eine einfache und intuitive API, die das schnelle Prototyping, die Implementierung und das Training neuronaler Netze ermöglicht. Keras unterstützt verschiedene neuronale Netzwerkarchitekturen, darunter Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) und Autoencoder.

4.3 TensorFlow

TensorFlow¹⁴ ist ein umfangreiches Open-Source-Framework von Google für maschinelles Lernen und Deep Learning. Es ermöglicht die effiziente Definition, Optimierung und Ausführung von numerischen Berechnungen mit Hilfe von Datenflussgraphen. TensorFlow bietet Skalierbarkeit auf verschiedenen Plattformen von mobilen Geräten bis zu großen verteilten Systemen und unterstützt sowohl CPU- als auch GPU-Beschleunigung. Keras ist in TensorFlow integriert und fungiert als High-Level-API, die den Zugang zu den leistungsfähigen Funktionen von TensorFlow erleichtert.

4.4 Deep GPU Xceleration (DGX)

Eine DGX ist ein Hochleistungsrechnersystem von NVIDIA¹⁵. 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 verfügen¹⁶.

Im weiteren Entwicklungsverlauf wurde jedoch vermehrt CPU-basiert gearbeitet, weshalb auf ein alternatives System umgestiegen wurde. Dies liegt daran, weil doch nur wenige Daten zum Trainings und Testen verwendet wurden und eine große GPU dazu nicht mehr benötigt wird.

5 Angewandte Technologie: Keycloak

Keycloak ist ein IAM-Tool mit SSO-Funktion. Es ist ein auf Rollen basiertes Identitäts- und Zugriffsmanagementsystem (RBAC). 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.

5.1 Terminologien in Keycloak

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. In diesem befindet sich der *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):

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. Es baut auf dem u. a. OIDC auf.

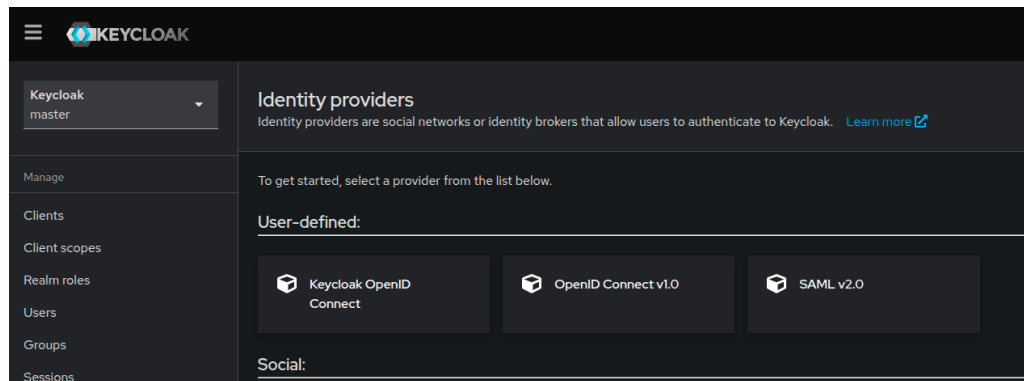


Abbildung 1: Protokolle für IdPs in Keycloak

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.

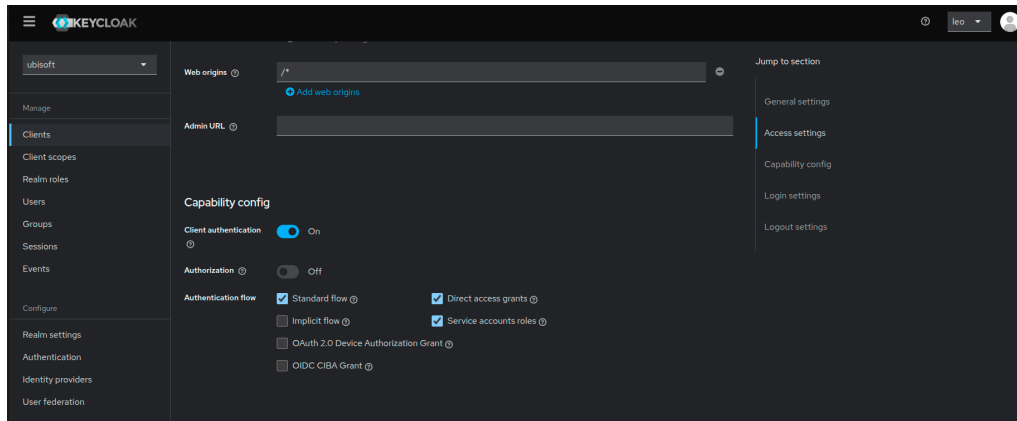


Abbildung 2: Beispielhafte Konfiguration von Authentication Flows in Keycloak

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:

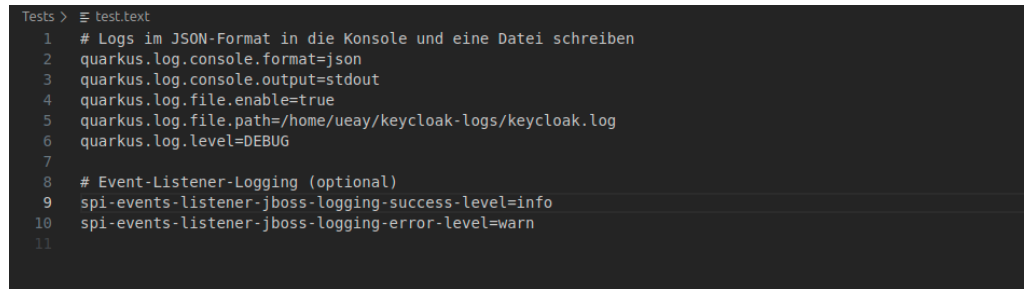
Clients können sich in Keycloak auch selbst authentifizieren – ohne Benutzerinteraktion. Wird die *Service Account*-Funktion für einen Client aktiviert, erhält dieser ein eigenes Servicekonto. Dieses Konto kann mit Rollen ausgestattet werden, um Rechte auf bestimmte Ressourcen oder Operationen zu erhalten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

5.2 Komponenten der Log-Erzeugung

Keycloak generiert Logs über verschiedenen Frameworks und Komponenten. Eines davon ist das Logging-Framework von JBoss, wobei moderne Versionen von Keycloak zusätzlich auf das Quarkus-Framework setzen. **Von WildFly zu Quarkus –**

Technologiewechsel:

Keycloak basierte in früherer Zeit **WildFly**-Anwendungsserver¹⁷, der wiederum die **JBoss Logging**-Infrastruktur verwendet. Ab der 17. Version von Keycloak wurde WildFly durch das **Quarkus**-Framework ersetzt. Quarkus unterstützt weiterhin JBoss Logging.



```
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

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

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 aus einer Quarkus-basierten Keycloak-Instanz

Insgesamt stellt JBoss Logging in Kombination mit Quarkus eine 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. Zudem 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. Obwohl Keycloak über ein umfangreiches Logging-Framework auf Basis von JBoss Logging verfügt, das in Kombination mit Quarkus eine zentrale Rolle im Systembetrieb spielt, nutzt diese Arbeit ausschließlich das Event-Logging-Subsystem, welches über die Keycloak-Admin-API zugänglich ist. Die oben beschriebenen Logging-Komponenten dienen damit in erster Linie dem Verständnis der Gesamtarchitektur, sind jedoch nicht direkt Grundlage der durchgeführten Datenanalysen.

5.3 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.

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*

5.4 Event-Logs

Analysiert in dieser Arbeit werden die Event-Logs und Admin-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.

5.5 Aufbau der Event-Logs

Folgende Features sind üblicherweise in den User Event Logs enthalten:

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

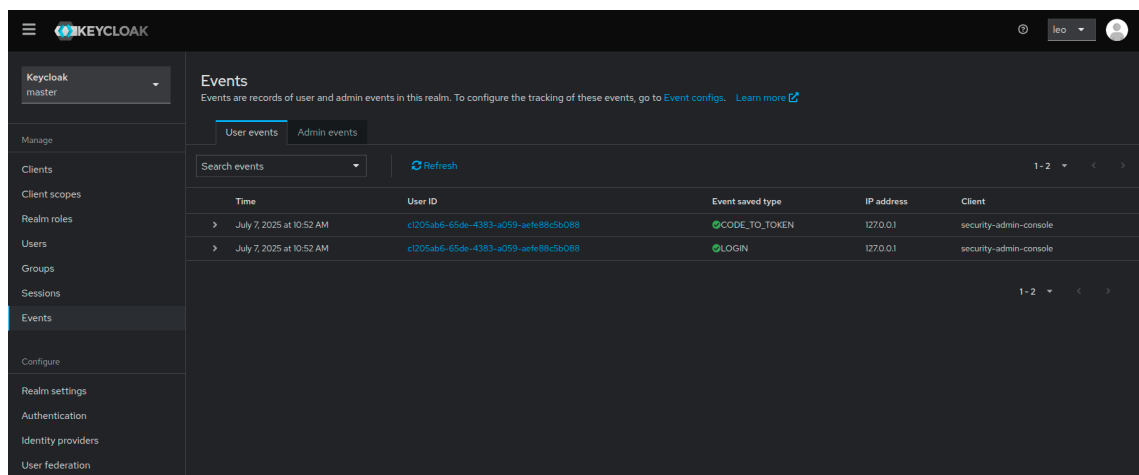
Man erkennt, dass wichtige Attribute, wie bspw. der Standort des Clients fehlen. Dies liegt an datenschutzrechtlichen Gründen, jedoch erschwert dies Anomalien in den Logs zu erkennen, weil diese auch anhand eines ungewöhnlichen Standortes (z.B. Pentagon) erkannt werden. Dies muss noch als Störfaktor genannt werden.

Anhand des Standortes können somit schon mal keine Anomalien erkannt werden. Timestamp ist einer der wichtigen Attribute, welche zeigen, zu welchem Zeitpunkt der Log erzeugt wurde. Zur Analyse zeitbasierter Daten eignet sich timestamp als aussagekräftige Information für die LSTM-AE-Hybridmodelle.

Des weiteren 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 ¹⁸. 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.

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-ae4e89c5b088	CODE_TO_TOKEN	127.0.0.1	security-admin-console
July 7, 2025 at 10:52 AM	c1205ab6-65de-4383-a059-ae4e89c5b088	LOGIN	127.0.0.1	security-admin-console

Abbildung 5: Beispiel Events nach Eventtyp

Je nach Event-Typ 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-Typen 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 ¹⁹.

6 Theoretischer Hintergrund der Modelle

6.1 LSTM

Ein LSTM ist ein spezieller Typ von rekurrentem neuronalen Netzwerk (RNN), der entwickelt wurde, um Informationen über längere Zeiträume hinweg zu speichern und verarbeiten [21].

RNNs gehören zu einer Klasse neuronaler Netzwerke, die sich durch ihre Fähigkeit auszeichnen, sequentielle Daten zu verarbeiten – also Daten, bei denen die Reihenfolge eine zentrale Rolle spielt. Dadurch eignen sie sich insbesondere für die Analyse von Zeitreihen. RNNs speichern frühere Informationen in einem sogenannten Hidden State, der bei jedem Zeitschritt aktualisiert wird. So kann der Kontext vorheriger Eingaben bei der Verarbeitung aktueller Daten berücksichtigt werden.

Ein zentrales Problem klassischer RNNs besteht jedoch darin, dass sie bei längeren Sequenzen dazu neigen, frühere Informationen „zu vergessen“. Der Gradient, also der Vektor, welcher anzeigt, in welche „Richtung“ das Modell verbessern soll, kann beim Zurück-propagieren über viele Zeitschritte hinweg sehr klein werden (sogenannter „Vanishing Gradient“), wodurch der Lerneffekt früherer Daten verloren geht.

LSTM-Netzwerke wurden entwickelt, um genau dieses Problem zu lösen [22, S.19]. Sie verfügen über eine spezielle Zellstruktur, die es ermöglicht, relevante Informationen über längere Zeiträume hinweg zu speichern. Dies wird durch drei zentrale Steuermechanismen („Gates“) innerhalb der LSTM-Zelle erreicht:

- **Forget Gate:** Entscheidet, welche Informationen aus dem Zellzustand verworfen werden sollen.
- **Input Gate:** Bestimmt, welche neuen Informationen in den Zellzustand aufgenommen werden.
- **Output Gate:** Regelt, welche Informationen aus dem Zellzustand als Ausgabe weitergegeben werden.

LSTM-Netzwerke finden in zahlreichen Anwendungsbereichen Verwendung, etwa in der Sprachverarbeitung, bei Vorhersage-Modellen oder im Kontext des Internets der Dinge (IoT) [23].

Die Wahl der Sequenzlänge in einem LSTM beeinflusst maßgeblich die Lern- und Analysefähigkeit. Sehr lange Sequenzen erschweren das Training und können die Erkennung relevanter Muster verringern, da das Modell über viele Zeitschritte hinweg Informationen behalten muss. Kürzere Sequenzen helfen dem Modell, fokussierter auf lokale Muster zu lernen, wodurch sich die Erkennungsgenauigkeit für Anomalien verbessern kann. Eine Herausforderung bei LSTMs ist die Festlegung der passenden Sequenzlänge. Hierarchische multiskalige LSTM-Modelle können dynamisch lernen, auf welchen Zeitskalen Informationen aktualisiert oder zusammengefasst werden sollen, was die flexible Handhabung langer Sequenzen ermöglicht [24]. Das hierarchisch multiskalige LSTM kann auf verschiedenen Ebenen gleichzeitig lernen, wie weit es „zurückblicken“ oder „zusammenfassen“ muss, um relevante Informationen zu behalten — und so auch sehr lange Sequenzen besser zu verarbeiten. Die Hierarchie sorgt dafür, dass Daten in Ebene gegliedert werden und Multiskalig bedeutet, dass das Modell zeitliche Abhängigkeiten auf mehreren Zeitskalen gleichzeitig betrachtet — also zum Beispiel kurze, mittlere und lange Zeiträume innerhalb der Sequenz.

Die Wahl der Sequenzlänge steht oft in engem Zusammenhang mit weiteren Modell- und Trainingsparametern wie der Batch-Größe sowie der Tiefe und Breite des Netzwerks. Eine größere Netzwerktiefe oder -breite kann theoretisch komplexere Muster erkennen, führt aber auch zu höherem Rechenaufwand und größerem Speicherbedarf, was bei langen Sequenzen schnell problematisch wird. Ähnlich kann eine zu große Batch-Größe das Training instabil machen oder den Speicher überfordern, besonders bei tiefen Modellen mit langen Sequenzen. Daher ist das Finden eines guten Gleichgewichts zwischen Sequenzlänge, Netzwerkgröße und Batch-Größe entscheidend, um effizientes Lernen und eine gute Generalisierung zu ermöglichen. Insbesondere bei längeren Sequenzen kann es sinnvoll sein, die Modellkomplexität zu reduzieren oder hierarchisch-multiskalige Architekturen zu nutzen, um die Verarbeitung effizienter

zu gestalten [25].

Da aber in dieser Arbeit nicht mit Millionen von Dateneinträgen gearbeitet wird, liegt der Fokus darauf, die Sequenzlänge gering zu halten, weil LSTMs darin scheitern, korrektes Zählverhalten auf deutlich längeren Sequenzen zu generalisieren, selbst wenn sie dies auf kürzeren Sequenzen während des Trainings scheinbar gelernt haben. [26].

6.2 Autoencoder

Ein Autoencoder ist ein spezieller Typ neuronalen Netzwerks, das aus zwei Hauptkomponenten besteht: einer Encoder- und einer Decoder-Schicht [27, S.3].

Die Encoder-Schicht komprimiert die Eingabedaten, indem sie deren Dimension reduziert und sie auf eine kompakte Repräsentation abbildet – die sogenannte Bottleneck-Schicht. Diese Engstelle zwingt das Modell dazu, die wesentlichen Merkmale der Daten in verdichteter Form zu erfassen. Dabei werden redundante oder irrelevante Informationen herausgefiltert, was es dem Autoencoder ermöglicht, die „Essenz“ der Daten zu lernen.

Die Decoder-Schicht nimmt diese komprimierte Repräsentation und rekonstruiert daraus die ursprünglichen Eingabedaten. Das Ziel des Netzwerks ist es, durch Minimierung des Rekonstruktionsfehlers – also der Differenz zwischen Eingabe und Ausgabe – eine möglichst originalgetreue Rekonstruktion zu erreichen. Der Rekonstruktionsfehler ist auch in der später vorgestellten Hybridarchitektur von zentraler Bedeutung.

Durch diese Eigenschaften eignen sich Autoencoder besonders gut für Aufgaben wie die Anomalieerkennung. Weicht eine Eingabe signifikant von den zuvor gelernten Mustern ab, gelingt die Rekonstruktion schlechter – der Rekonstruktionsfehler steigt. Dies kann als Indikator für potenzielle Anomalien genutzt werden.

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 Daten "durchzuschleusen". Um zu verhindern, dass das Modell im Bottleneck redundante Merkmale lernt, wird die Verlustfunktion erweitert. Dadurch wird das Netzwerk dazu angeleitet, möglichst vielfältige und wenig überlappende Merkmale zu lernen [28].

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. Da für diese Arbeit keine zu großen, aufwendigen Daten erstmals analysiert werden können, aufgrund mangelnder Trainingsdaten- wird ein Deep Autoencoder auch nicht zum Einsatz kommen.

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

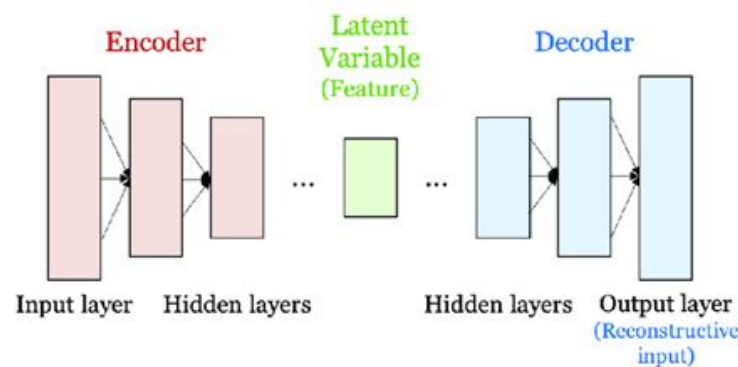


Abbildung 6: Visuelle Darstellung der Funktion des AEs, Quelle: https://www.researchgate.net/figure/Structure-of-the-autoencoder_fig1_365074431, Letzter Zugriff am: 15.07.2025

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.

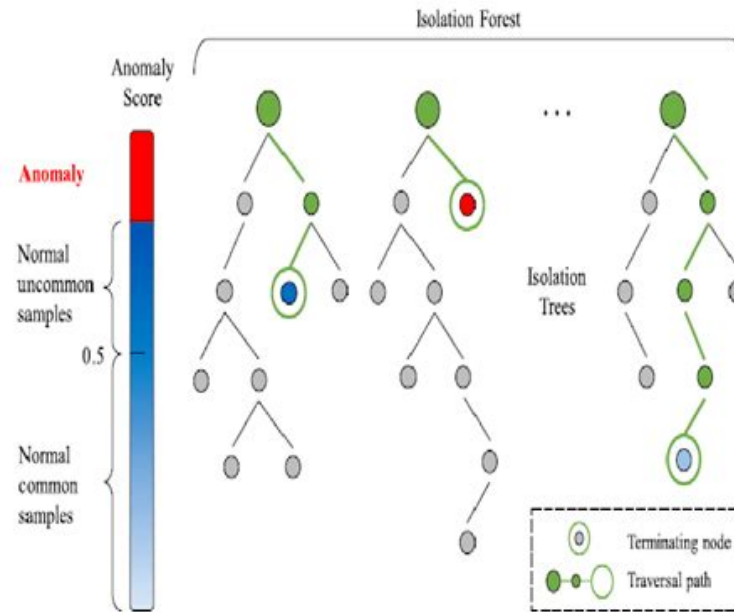


Abbildung 7: Visuelle Darstellung der Funktion des AEs, Quelle: https://www.researchgate.net/figure/Anomaly-Detection-using-Isolation-Forest-18_fig3_350551253, Letzter Zugriff am: 15.07.2025

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.

6.3 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. Isolation Forest ist im Vergleich zu anderen Algorithmen recht gut für hoch-dimensionale Daten geeignet [29].

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 [29, S.3].

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.

6.4 One-Class SVM

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 [30]. Die Methode nutzt eine nichtlineare Abbildung der Daten in einen höherdimensionalen Raum, um auch nichtlinear trennbare Daten linear trennbar zu machen: „Bei nichtlinear 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 [31]. 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.

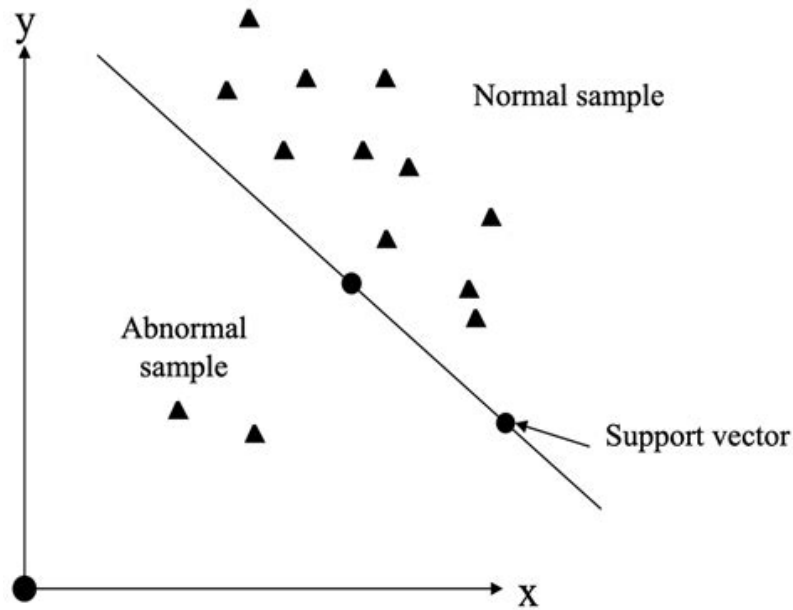


Abbildung 8: Visuelle Darstellung OCSVM. Quelle: <https://www.mdpi.com/2076-3417/13/3/1734>, Letzter Zugriff am: 15.07.2025

6.5 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 [32]. Die Wahl dieser Parameter ist entscheidend für den Erfolg der Klassifizierung.

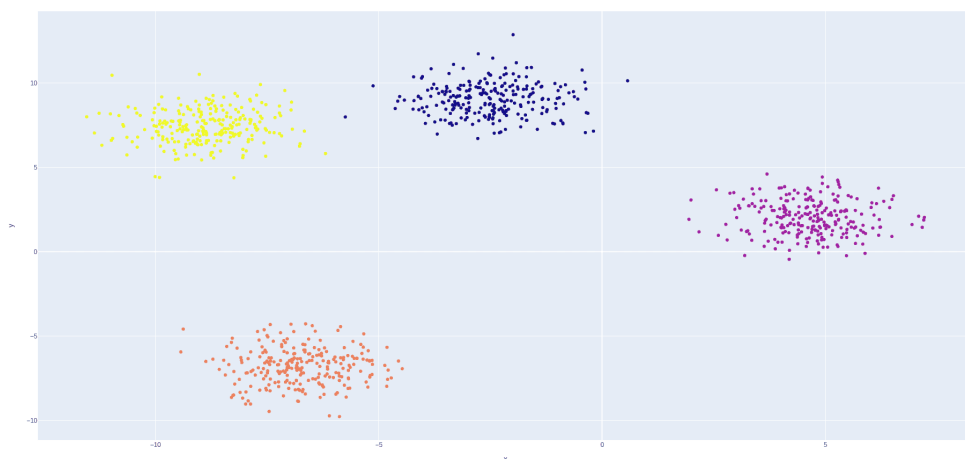


Abbildung 9: Beispiel DBSCAN-Cluster, Quelle: <https://www.baeldung.com/wp-content/uploads/sites/4/2023/04/clusters.png>, Letzter Zugriff am: 15.07.2025

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. Eine hohe Anzahl von Noise-Punkten kann darauf hinweisen, dass die Klassifikation nicht optimal gelungen ist [32].

6.6 Alternative Algorithmen

Es 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 [33]. 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-AE näher zu untersuchen.

Darüber hinaus wurden auch weitere Anomalieerkennungsverfahren wie K-Means und LOF (Local Outlier Factor) evaluiert. K-Means wird sogar als effektivster Algorithmus dargestellt im Vergleich zu anderen [34].

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- es ist auch wissenschaftlich bewiesen, dass er sich gut zur Erkennung von Anomalien eignet [35].

Kurz gefasst: Beide Verfahren eignen sich sehr gut zur Erkennung von Anomalien und wurden daher ebenfalls berücksichtigt. Einer Studie zufolge ist OCSVM in Bezug auf die Anomalie-Erkennung jedoch überlegen [36]. Zudem leidet auch LOF unter dem „Fluch der Dimensionalität“, da es auf Dichte-Schätzungen im euklidischen Raum (z. B. k -Nächste-Nachbarn-Distanzen) basiert, welche in hochdimensionalen Räumen an Aussagekraft verlieren. Dieses Problem ist bekannt und wird in der Literatur mehrfach beschrieben [37, S.366]. Die Frage, warum dennoch DBSCAN zusammen mit OCSVM und Isolation Forest gewählt wurde, obwohl diese eben-

falls betroffen sind, lässt sich mit einer Begrenzung des Arbeitsumfangs erklären. Darüber hinaus ähneln sich einige der ausgewählten Algorithmen denen, die nicht gewählt wurden: So ist K-Means beispielsweise ein Clustering-Algorithmus, ähnlich wie DBSCAN, und LOF weist Parallelen zum Isolation Forest auf.

Um eine möglichst hohe Varianz und einen ausgewogenen Vergleich zu gewährleisten, wurden daher sich ähnelnde Algorithmen bewusst ausgeschlossen. Ein weiterer Grund mitunter der Hauptgrund- weshalb beide abgelehnt wurden war die Tatsache, dass K-Means (ohne Optimierungsmechanismen) bei zu großen Datensätzen Skalierungsprobleme hat und dass LOF (ohne Optimierungsmechanismen), ebenfalls bei zu großen Datensätzen eine hohe Rechenzeit hat [38, 39].

7 Implementierung

7.1 Hybridmodelle mit LSTM-AE

Generell wurden die Modelle nach der selben Architektur gebaut:

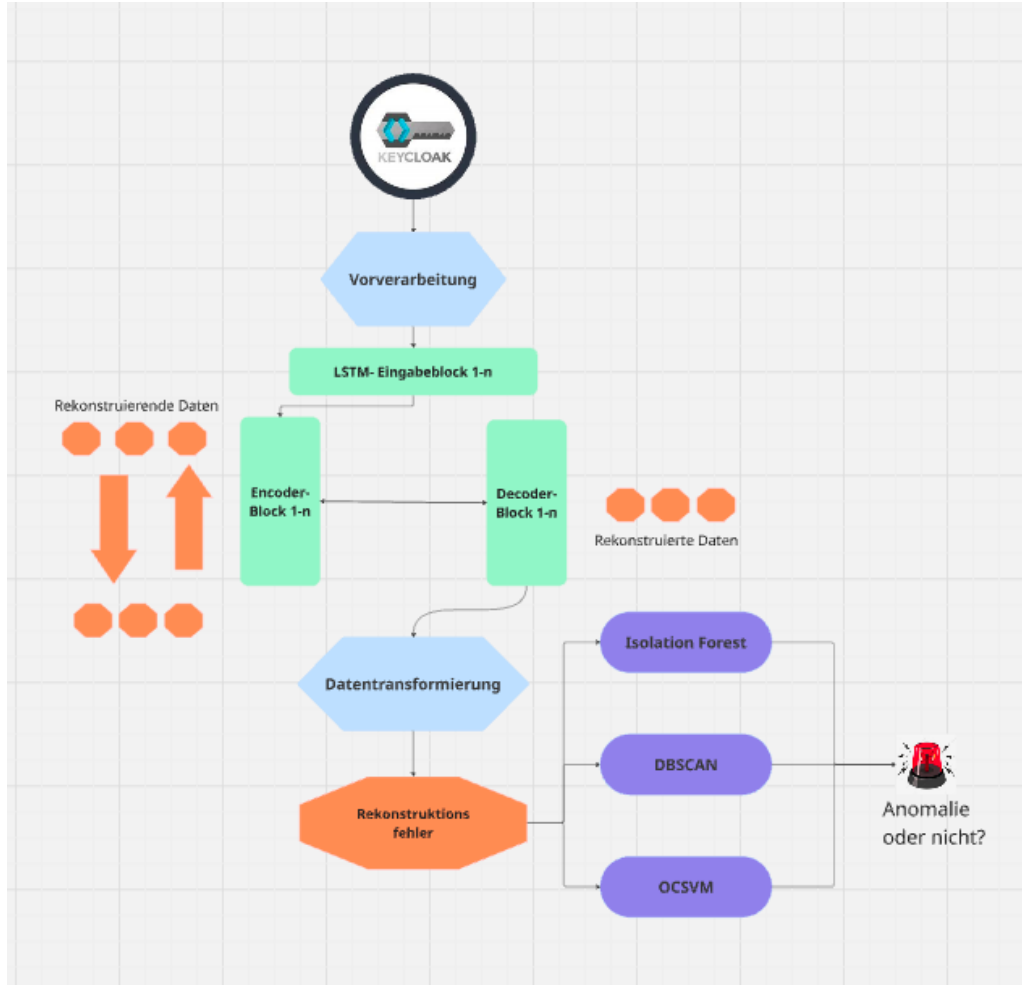


Abbildung 10: Skizze aller Hybridmodelle

man entschied sich für folgende Parameter: Man entschied sich für eine Archi-

Parameter	Wert
batch_size	32
dropout_rate	0.1
encoder_layers	[128, 64]
decoder_layers	[64, 128]
learning_rate	0.001

Tabelle 1: Bestimmte Parameter der Modellkonfiguration

tektur mit zwei Schichten im Encoder und Decoder (Tiefe) sowie 128 (Breite), um einen ausgewogenen Kompromiss zwischen Informationsreduktion zu erzielen. Diese Dimensionierung ermöglicht es dem Modell, komplexe zeitliche Muster in den Daten ausreichend zu komprimieren, ohne wesentliche Strukturen zu verlieren. Eine geringere Breite hätte hingegen das Risiko erhöht, dass relevante Informationen ver-

loren gehen. Es muss schon hier erwähnt werden, dass die Reliabilität dieser Arbeit durch die Regularisierungstechniken beeinflusst wird, was aber notwendig ist, da dies übliche Techniken sind, um Overfittung zu vermeiden.

Zudem wird die sogenannte Dropoutrate verwendet, eine Regularisierungstechnik, die Overfitting verhindert. Sie deaktiviert zufällig einzelne Neuronen während des Trainings. Die Dropoutrate wurde gering gewählt, um die Testläufe nicht zu stark zu randomisieren und die Reliabilität der Ergebnisse zu gewährleisten.

Die Daten werden zuerst in numerische Daten umgewandelt- wenn sie noch nicht numerisch sind. Absichtlich werden NaN-Werte und unklare Werte nicht bereinigt, sondern in eine eigene Kategorie gefasst, die auch ins numerische umgewandelt wird. Dies ist wichtig, da auch solche Werte Anomalien darstellen und die Bereinigung dieser Daten dafür sorgen würde, dass potenzielle Gefahrenquellen unerkannt bleiben. NaN-Werte können außerdem bspw. bei falschen Accounts oder vielleicht bei Bots auftreten, weshalb es wichtig ist, diese in der Analyse beizubehalten. Die numerischen Daten werden durch den LabelEncoder ²⁰ erzeugt. Die Daten werden danach skaliert mit dem StandardScaler ²¹.

Der LSTM-AE wird dabei nur auf normalen Daten trainiert. Dies liegt an der Natur des Autoencoders- denn dieser rekonstruiert Daten nur wieder. Wenn er lange genug trainiert wird, "lernt" er nur "normale" Daten zu konstruieren. In Testdaten, wo sich dann die Anomalien befinden, kann er schlechter rekonstruieren. Daraus entsteht dann der sogenannte Rekonstruktionsfehler.

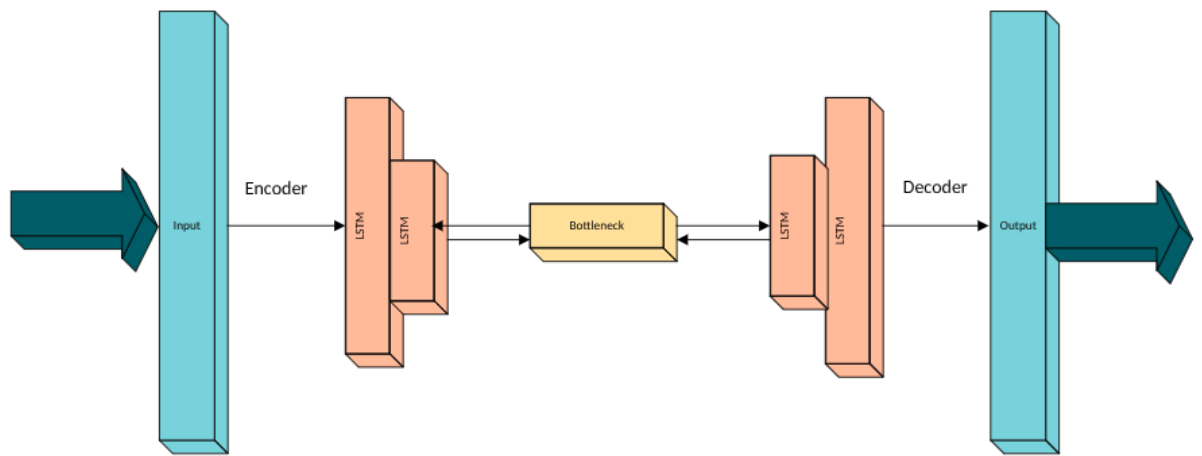


Abbildung 11: Gesamtarchitektur LSTM-AE

Dieser Trainingsfehler, bzw. Rekonstruktionsfehler, also die Daten, welche nicht klar klassifiziert werden konnten, werden danach jeweils mit einem der Anomalieerkennungs-Algorithmen (IF, OCSVM, DBSCAN) trainiert. Dies ist auch eine Vorgehensweise, welche bspw. in der Studie von Malhotra bekannt ist [40].

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.

Die Sequenzlänge wird gering gehalten, um nicht zu viele Daten zu generieren, da eine zu hohe Sequenzlänge kontraproduktiv ist. Auch die Batch-Size wird gering gehalten, damit sie mit der Sequenzlänge harmonisiert. Die Sequenzlänge gibt an, wie viele Logs gleichzeitig analysiert werden und deren Zusammenhänge erkannt werden. Die Batch-Size gibt an, wie groß die Eingabe dieser Sequenzen sein darf (weshalb sie proportional zueinander sein müssen). Für die Sequenzlängen wird in den letzten Abschnitten Testfälle angewandt.

Encoder-Decoder Struktur und Bidirektionales LSTM Die Architektur folgt dem Prinzip eines Encoder-Decoder-Systems, bei dem der Encoder die Aufgabe übernimmt, die wesentlichen Merkmale der Eingabesequenzen in einem kompakten, latenten Raum zu repräsentieren. Dieser latente Raum fasst die zeitlichen Mus-

ter und Strukturen zusammen, wodurch eine effizientere Repräsentation ermöglicht wird. Das bidirektionale LSTM im Encoder erlaubt es, Informationen nicht nur aus der Vergangenheit, sondern auch aus der Zukunft innerhalb der Sequenz zu erfassen, was besonders bei zeitlich stark korrelierten Daten von Vorteil ist. Der Decoder hingegen nutzt diese latente Repräsentation, um die ursprüngliche Sequenz zu rekonstruieren, wodurch eine präzise Nachbildung der Eingabedaten sichergestellt wird.

Verlustfunktion und Optimierung Zur Bewertung der Modellgüte wird die mittlere quadratische Abweichung (*Mean Squared Error*, MSE ²²) herangezogen, welche die Differenz zwischen Eingabe- und Ausgabesequenzen quantifiziert. MSE eignet sich zur Messung der Rekonstruktionsqualität bei kontinuierlichen Werten, da größere Fehler durch die Quadratisierung stärker gewichtet werden, was eine präzise Anpassung des Modells erzwingt. Die Optimierung erfolgt mittels des Adam-Optimierers ²³, welcher durch Lernraten für jeden Parameter besticht und somit eine schnelle und stabile Konvergenz auch bei komplexen Modellen gewährleistet. Die Lernrate stellt dabei einen wesentlichen Hyperparameter dar, der das Tempo der Gewichtsaktualisierungen steuert und einen Balanceakt zwischen schnellem Lernen und stabiler Anpassung darstellt.

Berechnung des Rekonstruktionsfehlers Die Idee, durch den Mean Square Error den Rekonstruktionsfehler zu berechnen stammt von Torabi et Al. [41, S. 5]. Die allgemein Formel zur Berechnung des Fehlers lautet wie folgt:

$$\text{Reconstruction Error} = \frac{1}{T \cdot F} \sum_{t=1}^T \sum_{f=1}^F (x_{t,f} - \hat{x}_{t,f})^2$$

Im Grunde genommen wird hier jeder Fehler als Zeitpunkt dargestellt. Jeder Zeitpunkt (I) und jedes Feature F stellt hier eine Abhängigkeit dar. Alle rekonstruierbaren Werte werden von den nicht-rekonstruierbaren über jedes Logs hinweg berechnet – und anschließend der Durchschnitt berechnet wird. Dies ist dann der gesamte Rekonstruktionsfehler.

Ermittlung des Schwellenwertes Ab wann soll das Modell eine Anomalie erkennen? Soll es schon bei einer kleinen Abweichung eines Features es als Anomalie deuten oder erst ab großen Unterschieden? Jablonski et Al. [42] haben dies untersucht und sind zu folgendem Algorithmus gekommen: Man extrahiert einfach alle Rekonstruktionsfehler aus den Daten und sortiert sie zuerst. Der optimale Schwellenwert ist genau die Mitte zwischen zwei Werten, wo der größte Abstand im sortierten

Fehlervektor gefunden wurde. Nach den Autoren sei dies ein optimaler Weg, um den Schwellenwert zu erkennen. Dieser wurde jedoch in dieser Arbeit später verworfen, weil man mit diesem Ansatz schlechtere Ergebnisse in Precision und Recall für alle Hybridmodelle erhielt. Ursache dafür ist wahrscheinlich, dass die Anomalien im Verhältnis zu Anzahl normaler Logs zu gering ist. Um Anomalien in Zeitreihen zuverlässig zu erkennen, ist es notwendig, die Rohwerte der Anomalie-Score-Funktion (die Rekonstruktionsfehler, die von Modellen wie Isolation Forest oder OCSVM berechnet werden) in binäre Entscheidungen zu übersetzen. Hierzu wird ein Threshold festgelegt: Werte oberhalb des Thresholds werden als Anomalien klassifiziert, Werte darunter als normal. Die Wahl des Thresholds hat direkten Einfluss auf die Balance zwischen Precision (wie viele als Anomalien erkannten Punkte tatsächlich Anomalien sind) und Recall (wie viele der echten Anomalien erkannt werden). Um diese Balance optimal zu berücksichtigen, verwenden wir den F1-Score, der Precision und Recall kombiniert. Der Threshold wird so gewählt, dass der F1-Score maximiert wird. Dieser Ansatz erwies sich als effektiver und es ist auch bewiesen, dass dies ein guter Ansatz ist, um den Schwellenwert zu berechnen, auch besonders bei Halb-Überwachten Lernmodellen [43].

7.1.1 Implementierung: Isolation Forest

Im Fall des Isolation Forest wird ein auf Entscheidungsbäumen basierender Algorithmus genutzt, der speziell für die Erkennung von Ausreißern in hochdimensionalen Daten entwickelt wurde. Das Modell wird auf den Fehlerwerten der Trainingsdaten trainiert und anschließend auf die Testdaten angewendet. Anstatt die standardmäßige Vorhersage (-1 = anomal, 1 = normal) direkt zu verwenden, werden die Rohscores des Modells herangezogen, um mittels einer F1-basierten Schwellenwert-Optimierung eine binäre Klassifikation zu erzeugen. Hierbei wird für jede Sequenz ein optimaler Schwellenwert ermittelt, der die F1-Metrik maximiert. Alle Punkte mit Scores oberhalb dieses Schwellenwerts werden als Anomalien (1) klassifiziert, alle anderen als normal (0). Da die Daten sequenzbasiert aufgebaut sind – beispielsweise aus Zeitreihen bestehen – erfolgt eine Anpassung der Labels: Aus den Zielwerten der Testdaten wird jeweils das Label der letzten Position jeder Sequenz extrahiert, sodass ein Vergleich zwischen den binären Vorhersagen und den tatsächlichen Labels möglich ist.

7.1.2 Implementierung: OCSVM

Für die Umsetzung der One-Class SVM wird das Modell mit einem ν -Wert initialisiert, der die erwartete Obergrenze an Ausreißern im Trainingsdatensatz fest-

legt. Zusätzlich wird kein `gamma`-Parameter gesetzt, sodass der RBF-Kernel automatisch die Standard-Einstellung (`gamma='scale'`) verwendet. Anstatt die Standardvorhersage (-1 = anomal, 1 = normal) direkt zu verwenden, werden die Rohscores der Entscheidungsfunktion genutzt, um mittels einer F1-basierten Schwellenwert-Optimierung eine binäre Klassifikation zu erzeugen. Hierbei wird für jede Sequenz ein optimaler Schwellenwert ermittelt, der die F1-Metrik maximiert. Punkte oberhalb dieses Schwellenwerts werden als Anomalien (1) klassifiziert, die übrigen als normal (0). Da die Daten sequenzbasiert aufgebaut sind – beispielsweise aus Zeitreihen bestehen – wird aus den Testlabels jeweils das Label der letzten Position jeder Sequenz extrahiert, um die binären Vorhersagen korrekt mit den tatsächlichen Labels vergleichen zu können.

7.1.3 Implementierung: DBSCAN

Das DBSCAN-Modell wird direkt auf die Rekonstruktionsfehler der Testdaten angewendet. Dabei werden keine festen Werte für `eps` und `min_samples` vorgegeben, sodass das Modell flexibel auf die vorliegenden Daten reagieren kann. Die Methode `fit_predict()` liefert für jeden Punkt entweder eine Clusterzuweisung (0, 1, ...) oder -1, wobei -1 Datenpunkte kennzeichnet, die als Rauschen bzw. Anomalien erkannt werden. Diese Labels werden anschließend in binäre Werte umgewandelt (1 = Anomalie, 0 = normal), analog zu den anderen Verfahren.

7.2 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 möglich, bei dem ein Hybridmodell zwischen IF und LSTM-AE erstellt wurde [44]. In der Studie wurde zunächst der IF verwendet, um die Rohdaten automatisch zu klassifizieren und sogenannte Pseudo-Labels für Anomalien und normale Zustände zu generieren. Diese vorläufige Labelung diente dazu, das Training des nachfolgenden LSTM-Autoencoders zu unterstützen. Das LSTM-Autoencoder-Modell wurde anschließend mit den so gelabelten Daten trainiert, um zeitliche Muster besser zu erfassen und Anomalien präziser zu erkennen. Durch diese Kombination aus einem schnellen, unüberwachten Verfahren zur Vorfilterung und einem sequenzbasierten Modell zur detaillierten Analyse konnte die Anomalieerkennung verbessert werden. 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 [45].

Es gab auch Ideen, für größere 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 Anomal 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.

Für das LSTM-AE wurde zudem noch überlegt, sogenannte *Residual Blocks* einzubauen. Diese Blöcke fügen der normalen Verarbeitung eine direkte Verbindung zwischen Ein- und Ausgangsschicht hinzu, sodass die Eingabe unverändert zum Ausgang addiert wird. Dadurch kann das Netzwerk lernen, nur die *Differenz* (Residual) zwischen Ein- und Ausgang zu modellieren. Diese Architektur erleichtert das Training tiefer Netzwerke, da sie beim Backpropagation-Prozess den Gradientenfluss verbessert und das Problem des *vanishing gradient* (verschwindender Gradient) reduziert. Ohne Residual Blöcke kann der Gradient in sehr tiefen Netzen zu klein

werden, wodurch das Lernen erschwert wird. Mit Residual-Verbindungen wird dieser Effekt abgefedert, was zu stabilerem und effizienterem Training führt. Auch nach der Studie nach Zhang et Al. ist dies bspw. gut zur Erkennung von Bildern [46].

Dennoch wurde diese Architektur verworfen, da Residual Blöcke insbesondere bei kleineren Trainingsdatensätzen, wie in dieser Arbeit mit etwa 50.000 Datenpunkten, wenig Mehrwert bieten und unter Umständen sogar kontraproduktiv sein können. Die Blöcke kommen vor allem bei sehr großen Datensätzen und tiefen Netzwerken mit mehreren Hunderttausend bis Millionen Trainingsbeispielen sinnvoll zum Einsatz.

Für den Autoencoder bestand noch die Möglichkeit, Deep Autoencoder-Modell zu verwenden, welches sogenannte „Attention Layer“ [47] hat- wurde jedoch verworfen, dafür den Testfall dieser Arbeit wie bereits erwähnt keine zu komplexe Architektur notwendig ist.

8 Möglichkeit 1: Generierung der Logs

8.1 Erster Entwurf der Angriffsfälle

Zu den häufigsten Insider Threat-Angriffen in IT-Systemen zählen laut aktuellem Forschungsstand unter anderem der Missbrauch privilegierter Rollen und Rechte, Datendiebstahl sowie die Manipulation kritischer Daten, wie etwa das Löschen wichtiger Dateien [48, S.6]. Auch Brute-Force-Angriffe gehören zu den zentralen Bedrohungen.

Für diese Arbeit wurden Angriffsszenarien ausgewählt, die in den Log-Dateien identifizierbar sind. Beispielsweise lässt sich ein Datendiebstahl in Keycloak nur schwer direkt erkennen, da Log-Dateien keine spezifischen Hinweise auf große Dateiübertragungen enthalten. Wahrscheinlicher und gleichzeitig nachvollziehbarer in den Logs sind dagegen Angriffe wie das unbefugte Erlangen von Administrator-Rollen oder Brute-Force-Angriffe. Letztere lassen sich etwa über das Event-Attribut LOGIN_ERROR identifizieren.

Die folgenden Angriffsszenarien wurden daher modelliert und in die Log-Daten integriert:

- *Anwendungsfall 1:* Brute-Force-Angriffe bei der Anmeldung
- *Anwendungsfall 2:* Löschen und Verändern sensibler Konfigurationen wie Clients, Benutzer, Passwörter und Realms

- *Anwendungsfall 3:* Erlangen privilegierter Rollen, z.B. manage-users und realm-admin

Darüber hinaus wurden bekannte Anomalien wie ungewöhnliche IP-Adressen und Login-Zeitpunkte bei der Generierung der Daten berücksichtigt. Besonderes Augenmerk liegt dabei auf einer hohen Variabilität der Angriffsformen, um die Aussagekraft der Tests zu erhöhen. In späteren Phasen dieser Arbeit wurden noch weitere Angriffe ergänzt, welche in den nachfolgenden Kapiteln erläutert werden.

8.2 Automatische Erstellung der Keycloak-Logs

Die Log-Daten werden mithilfe eines Python-Skripts automatisiert generiert.

Der erste Anwendungsfall simuliert Brute-Force-Angriffe, indem eine Abfolge mehrerer Log-Einträge mit dem Event-Typ `LOGIN_ERROR` erzeugt wird. Diese Einträge können ungewöhnliche Zeitstempel oder IP-Adressen enthalten, was jedoch nicht zwingend notwendig ist – bereits die Häufung von `LOGIN_ERROR`-Ereignissen deutet auf ein anomales Verhalten hin. Da Keycloak die Sperrung einer IP-Adresse nach einer konfigurierbaren Anzahl fehlgeschlagener Anmeldeversuche ermöglicht, wird dieser Parameter bei der Log-Erzeugung berücksichtigt. Die Anzahl der erzeugten fehlerhaften Logins variiert zufällig zwischen drei und fünfzehn Einträgen. Diese Auswahl basiert auf praktischer Erfahrung: Bereits ab drei gescheiterten Versuchen wird das Verhalten oft als verdächtig eingestuft – auch wenn es sich nicht immer um einen Angriff handelt. Ab etwa fünfzehn Fehlversuchen hingegen liegt mit hoher Wahrscheinlichkeit eine Anomalie vor. Die Zahl wird bewusst niedrig gehalten, um die Log-Sequenzen möglichst kurz und realistisch zu halten.

Im zweiten Anwendungsfall werden Log-Einträge erzeugt, die auf das Löschen oder Modifizieren von Entitäten hinweisen. Hierzu gehören beispielsweise Event-Typen wie `DELETE`, `UPDATE_PASSWORD` oder `UPDATE_PROFILE`. Es werden mindestens fünf solcher Einträge pro Sequenz erzeugt, wobei auf ausreichend Variabilität geachtet wird. Die Ereignisse werden in zufälligen Kombinationen generiert, um zu vermeiden, dass das Modell nur eine festgelegte Angriffssignatur erlernt. Würde beispielsweise das Feature `authType` ständig zusammen mit dem Event-Typ `UPDATE` auftreten, bestünde die Gefahr des Overfittings: Das Modell würde dann nur dieses konkrete Muster als Anomalie erkennen und andere Formen übersehen.

Im dritten Anwendungsfall wird überprüft, ob ein Nutzer, der ursprünglich keine privilegierten Rollen hatte, plötzlich Administratorrechte erhält. Bei der Generierung wird sichergestellt, dass die betreffende Session einen Benutzer enthält, der zu

Beginn keine Admin-Rollen besitzt und später Rollen wie admin, realm-admin oder manage-users zugewiesen bekommt – gegebenenfalls sogar die Admin-ID selbst.

Normale Benutzer-Sessions enthalten keine ungewöhnlichen IP-Adressen oder Zeitstempel. Als ungewöhnliche Uhrzeiten gelten in dieser Arbeit Zugriffe zwischen 0:00 Uhr und 6:00 Uhr morgens. Obwohl diese Definition kontextabhängig ist – etwa bei Homeoffice-Arbeit in späteren Stunden – wird in dieser Arbeit vom Regelfall ausgegangen, in dem nächtliche Aktivitäten als auffällig gelten.

Die Definition ungewöhnlicher IP-Adressen orientiert sich überwiegend an den von der IANA bereitgestellten Daten²⁴.

```
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 12: Ausschnitt der generierten Logs

Bei der Generierung der Log-Daten wurde auch berücksichtigt, dass neben eindeutig normalen und anomalen Sessions auch uneindeutige Sequenzen erzeugt werden, die als *Noise* klassifiziert werden können. Solche Fälle treten in realen Systemen häufig auf – etwa wenn ein normaler Nutzer temporär priorisierte Rollen übernimmt oder häufig Passwörter ändert, ohne dass ein tatsächlicher Angriff vorliegt. Um eine zu starke Sensitivität der Modelle gegenüber solchen Mustern zu vermeiden, wird bewusst ein gewisser Anteil an Rauschen erzeugt.

8.3 Problematik dieser Lösung

Die Analyse basierte ausschließlich auf den wichtigsten, verfügbaren Log-Daten, die typische Systemaktivitäten und deren Merkmale widerspiegeln. Eine tiefere Untersuchung der internen Struktur und des vollständigen Aufbaus der Event-Typen in Keycloak war aufgrund fehlender umfassender Dokumentation und eingeschränktem Zugriff auf administrative Systemressourcen nicht möglich. Diese Einschränkungen begrenzen die Tiefe und Genauigkeit der Auswertung und können die Aussagekraft der Ergebnisse beeinflussen.

Eine zentrale Herausforderung bestand in der realistischen Modellierung anomaler Sessions. Dabei musste unterschieden werden, ob eine Session normal, anormal oder als sogenannter *Noise* einzustufen ist — also als legitimes, aber ungewöhnliches Verhalten, das fälschlich als Anomalie interpretiert werden könnte. Da echte Anomalien in der Realität selten auftreten, wurde bei der Datengenerierung heuristisch eine Auftrittswahrscheinlichkeit von 2, % für anomale Sessions angenommen. Dadurch ist nicht gewährleistet, dass die erzeugten Daten eine realistischere Verteilung widerspiegeln.

Ein weiterer Nachteil der synthetischen Generierung liegt darin, dass zufällig zusammengesetzte Ereignismuster entstehen können, die keinem realistischen Angriff entsprechen. Um dem entgegenzuwirken, wurde eine Logik implementiert, die grundlegende Systemregeln von Keycloak berücksichtigt: So darf ein Benutzer mit eindeutiger ID nur in einem Realm existieren, es gibt jeweils nur einen Administrator mit explizit definierten Rollen und höherer Priorität, und normale Benutzer können keine Operationen ausführen, die mit dem Attribut *operationType* (CRUD-Operationen) verbunden sind — diese bleiben ausschließlich Administratoren vorbehalten. Zudem wird sichergestellt, dass jeder Benutzer genau einer Session und einem Realm zugeordnet ist, sofern es sich nicht explizit um einen Angreifer handelt.

Aufgrund dieser Einschränkungen sowie der im späteren Verlauf beobachteten Resultate wurde der vorgestellte Ansatz zwar modellartig getestet, letztlich jedoch verworfen.

9 Möglichkeit 2: Datenbeschaffung als Alternative

Da die Generierung von Log-Daten mit hohem Aufwand verbunden ist, wurde stattdessen ein sinnvollerer Ansatz gewählt: die Beschaffung realer Keycloak-Daten (also

direkt aus Keycloak selbst erzeugt). Aus datenschutzrechtlichen Gründen werden hierbei keine Daten aus Unternehmen verwendet. Stattdessen wurde die Idee verfolgt, eine lokale Keycloak-Instanz aufzusetzen und darauf basierend eigene Log-Daten zu generieren. Da auch dieser Weg zeitintensiv ist, wurde zusätzlich erwogen, mittels sogenannter Selenium-Tests sowohl Angriffsszenarien als auch gewöhnliche Keycloak-Daten zu simulieren. Als Grundlage zur Definition normaler Keycloak-Logs dient dabei eine Orientierung an firmeninternen Log-Daten.

9.1 Anbindung der Keycloak-API

Um Zugriff auf die Keycloak-Logs zu erhalten, ist es erforderlich, eine eigene Keycloak-Instanz zu installieren und sich mit Administratorrechten anzumelden.

Der Zugriff auf die Event-Logs erfolgt über die Keycloak-API, welche insbesondere bei der verwendeten Quarkus-Distribution eingesetzt wird. Standardmäßig benötigt man hierfür entsprechende Administrator-Rollen, um sensible Informationen wie Benutzer-Event-Logs und Administrator-Event-Logs abrufen zu können. Zunächst müssen die korrekten Authentifizierungsdaten vorliegen, um die erforderlichen Zugriffsrechte zu erhalten.

Für die Authentifizierung wurde der sogenannte *Client Credentials* Grant-Typ gewählt. Dabei erfolgt die Anmeldung über einen registrierten Client, wobei neben dem Client-Secret weitere Zugangsdaten übermittelt werden.

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

Abbildung 13: Daten, die an den Keycloak-Server gesendet werden müssen

Durch diesen Vorgang wird ein Zugriffstoken generiert, welches für den Abruf der Event-Logs benötigt wird. Das Token enthält Informationen zu den Zugriffsrechten des jeweiligen Administrators. So müssen beispielsweise die Rechte zum Verwalten von Benutzern enthalten sein, damit diese auch im Token ausgewiesen werden. Anschließend wird mit diesem Token eine weitere Anfrage an die API gestellt, um die Event-Logs abzurufen. Die erhaltenen Daten werden in einer separaten JSON-Datei gespeichert.

Weitere Details zur Keycloak-API sind in der offiziellen Dokumentation zu finden
25.

9.2 Selenium-Tests zur Erzeugung der Logs

Selenium ist ein Framework zur automatisierten Ausführung von Webbrowser-Aktionen und als Python-Paket verfügbar. Es ermöglicht, Interaktionen mit der Benutzeroberfläche realitätsnah zu simulieren, sodass der Eindruck entsteht, ein Mensch bediene den Browser. Die wesentlichen Komponenten sind:

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

Für die Erstellung der Logs wird ausschließlich der WebDriver verwendet, der 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 gestaltet sich einfach: Es wird der zu verwendende Browser („Driver“) definiert und die Tests werden auf dem Port ausgeführt, auf dem die lokale Keycloak-Instanz erreichbar ist.

Ein Nachteil dieser Methode liegt in der konstanten IP-Adresse, da alle Anfragen vom selben Host ausgehen. Dies kann dazu führen, dass das Modell lernt, mehrere Benutzer mit einer gemeinsamen IP-Adresse zu assoziieren, was in realen Szenarien als Anomalie interpretiert werden könnte. Daher wird die IP-Adresse vor der Weiterverarbeitung aus den Events entfernt. Diese Maßnahme reduziert zwar eine potenzielle Störquelle, verringert jedoch auch die Datenvielfalt für das Modell – besonders deshalb, weil die IP-Adresse als Feature gilt, welche mit Anomalien assoziiert sein kann.

Außerdem erfolgt die Log-Erzeugung in kurzen zeitlichen Abständen, wodurch die zeitliche Streuung reduziert wird. Da das verwendete LSTM-Autoencoder-Modell zeitbasierte Muster analysiert, kann dies die 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 jedoch die Generierung geringerer Datenmengen (z. B.

über 10.000 Events), was jedoch für das LSTM-Autoencoder-Modell ausreichend ist für das Training. Da die Logs direkt durch die definierten Tests erzeugt und gespeichert werden, entfällt die Notwendigkeit einer separaten manuellen Log-Generierung.

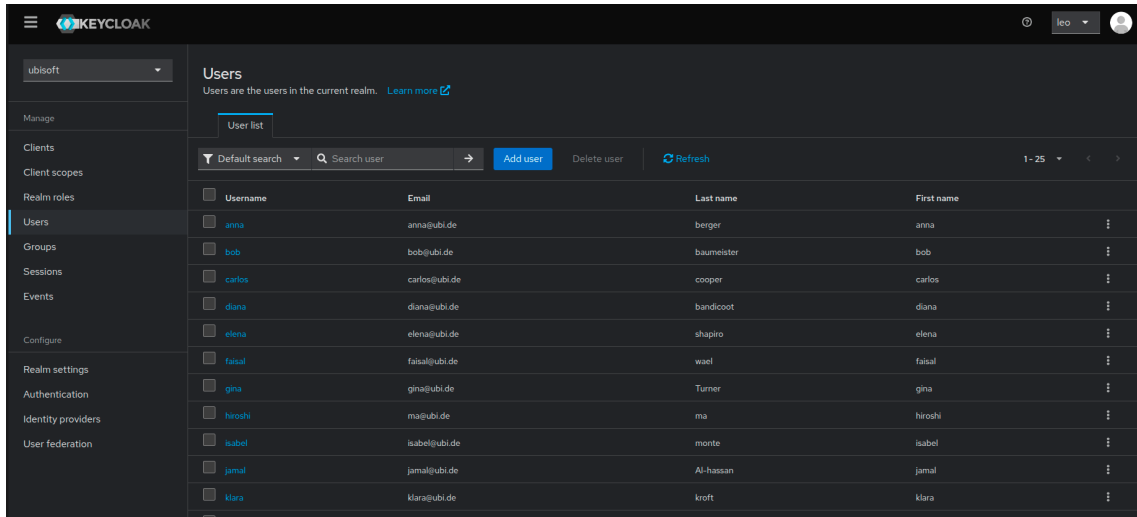


Abbildung 14: Beispielhafte Darstellung der generierten Benutzer für den Testfall

Zur Generierung der normalen Trainingsdaten wurden zuvor vier Realms erstellt:

- 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 Administrator, 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 den Benutzern, sondern den jeweiligen Clients zugewiesen werden. Dies liegt in der Architektur von Keycloak begründet: 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 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 ein vollständiges Profil (Vorname, Nachname, E-Mail) erforderlich ist und vorhanden ist – andernfalls würde die Authentifizierung fehlschlagen.

Darüber hinaus verfügen Clients dann über Service Accounts. 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 automatisierte Vorgänge im Hintergrund zu ermöglichen.

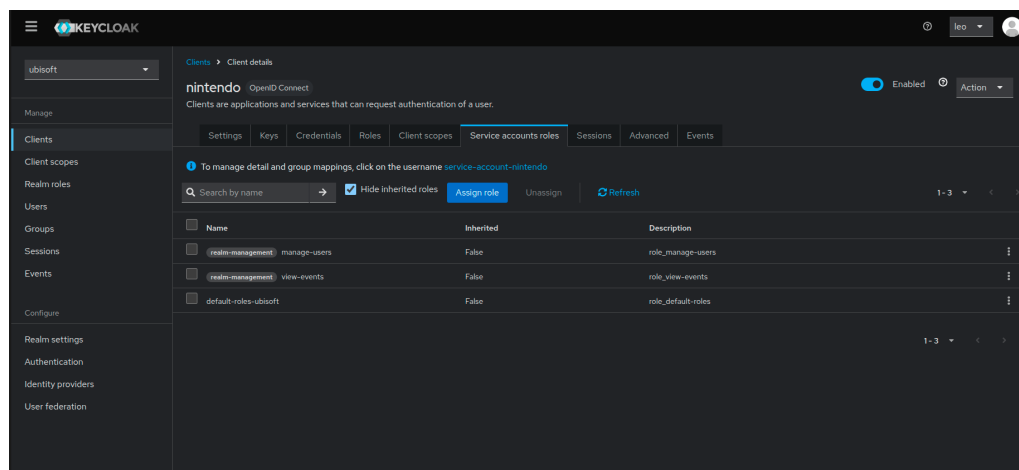


Abbildung 15: Benötigte Rollen für den Client Nintendo

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“ nach dem Vergleich mit den firmeninternen Keycloak-Logs:

- Anmeldung über Keycloak direkt
- Anmeldung über einen Client
- Token-Aktualisierung (Refresh Token)
- Abmeldung (Logout), auch nach Ablauf einer Session
- Fehlgeschlagene Login-Versuche (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 beziehungsweise Clients – wurden in der Log-Generierung bewusst nicht automatisiert. Der Grund hierfür liegt in der vergleichsweise hohen Komplexität und der geringen Häufigkeit dieser 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 potenzielle 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 Durchlauf werden insgesamt 30 Sessions (Bei den ersten 10000 Loggs waren es noch 15, wurde aber erhöht, um schneller Logs zu erzeugen) 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 am Tag zwischen 2.500 und 5.000 Log-Einträge erzeugt werden.

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 10 aufeinanderfolgende Tage zu konzentrieren, ausgeschlossen ist dabei das Wochenende. Die Sessions werden dabei nacheinander mit variablen Abstand ausgeführt.

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

9.3 Mögliche Angriffsszenarien basierend auf Common Vulnerabilities and Exposures (CVE) und Common Weakness Enumeration (CWE)

Common Vulnerabilities and Exposures (CVE) stellt eine umfassende Datenbank bekannter Sicherheitslücken dar, die in verschiedenen Softwaresystemen dokumentiert sind. Für Keycloak existieren ebenfalls zahlreiche veröffentlichte Schwachstellen²⁶

CVE-Einträge liefern jedoch keine Lösungen, sondern dienen der transparenten und neutralen Information über Art, Ursache und potenzielle Auswirkungen der jeweiligen Sicherheitslücke. Jeder Eintrag erhält eine eindeutige Identifikationsnummer, eine Kurzbeschreibung sowie (sofern verfügbar) Angaben zur betroffenen Softwareversion. Die Pflege und Dokumentation erfolgt weltweit durch Organisationen, Forschungseinrichtungen sowie unabhängige Sicherheitsexperten.

Im Gegensatz dazu beschreibt die Common Weakness Enumeration (CWE) systematisch bekannte Schwachstellenmuster (Weaknesses), die potenziell zu Sicherheitslücken führen können. Während CVEs konkrete, bereits gefundene oder ausgenutzte Schwachstellen auflisten, klassifiziert CWE abstrakte Fehlerklassen, wie beispielsweise „unzureichende Rechteprüfung“ oder „unsichere Tokenverarbeitung“. Beide Kataloge sind essentiell für die systematische Entwicklung und Interpretation von Sicherheitstests.

Obwohl viele der bekannten Probleme bereits behoben wurden, zeigen sich Schwachstellen in Keycloak zunehmend wiederkehrend und entwickeln sich mit der Zeit weiter.

Im Rahmen dieser Arbeit wurden mittels Selenium gezielt sicherheitsrelevante Angriffsszenarien nach CVE und CWE und anderen Quellen (die schon im vorherigen

Kapiteln bzgl. des ersten Entwurfes der Angriffsszenarien beschrieben wurden), gegen Keycloak simuliert und getestet. Es wird dabei folgendes Angriffsszenario umgesetzt:

CVE-2024-3656 - Privilege Escalation in Keycloak Bei CVE-2024-3656 ²⁷ handelt es sich um eine Sicherheitslücke in Keycloaks Admin REST API, bei der unzureichende Berechtigungsprüfungen es einem Nutzer mit niedrigen Rechten ermöglichen, administrative Endpunkte aufzurufen. Dadurch können sensible Daten wie Benutzerlisten eingesehen und administrative Aktionen durchgeführt werden, ohne dass entsprechende Rechte vorliegen. Die Ursache liegt in fehlerhaften oder fehlenden Zugriffskontrollen auf kritischen API-Endpunkten. Die Schwachstelle wurde in neueren Keycloak-Versionen behoben, darunter auch die Version 26.1.0, welche in dieser Arbeit verwendet wird. Andere Angriffsfälle konnten nicht umgesetzt werden, da manche Features die dafür verlangt werden in aktuellen oder früheren Versionen nicht mehr erhältlich sind. Auch wenn dieser Angriff fehlschlägt- sorgt er dafür, dass viele Logs mit anomalen Charakter generiert werden.

Da CVE-Datenbanken keine vollständigen Exploitbeschreibungen liefern, wurde im Rahmen der Simulation eine technische Interpretation der Schwachstellen vorgenommen. Zudem wurden noch weitere Angriffe ausgeführt, welche Typisch sind für einen Angriff und auch schon in den bekannten Insiderangriffsfällen genannt wurde:

- **Account Sabotage-Test:** Löschen oder Verändern wichtiger Benutzerinformationen.
- **Brute Force Angriff**

9.4 Angriffsszenarien

Die im vorherigen Abschnitt beschriebenen Angriffsszenarien wurden im Rahmen dieser Arbeit mithilfe von Selenium-Tests implementiert und in den generierten Logs dokumentiert. Für jedes Szenario wurde ein konkreter Ablauf definiert, um die jeweilige Sicherheitslücke zu simulieren.

Hier werden die Angriffsfälle aufgelistet welche direkt in der Keycloak-Konsole ausgeführt wurden, also nicht als Selenium-Test- außer das erste Szenario des Testfalls *Privilege Escalation*, dieses wird auch als Selenium-Test ausgeführt.

9.4.1 Angriffsszenario: Versuch einer Privilege Escalation über die Admin REST API

In diesem Szenario wird untersucht, ob ein technisch privilegierter, aber nicht vollständig administrativer Benutzer in Keycloak durch gezielte API-Aufrufe seine eigenen Berechtigungen ausweiten kann. Ziel ist es, das Verhalten von Keycloak bei einer möglichen Rechteausweitung über die REST-Schnittstelle zu evaluieren.

Ziel: Unautorisierte Erweiterung der Benutzerrechte durch Ausnutzung der REST API

Betroffener Realm: Ubisoft

Angreiferrolle: Benutzer mit der Rolle `manage-users`

Betroffener Benutzer: bob (versucht sich selbst administrative Rechte zu geben)

Ablauf:

1. Der Benutzer `bob` meldet sich regulär im System an und verfügt über ein gültiges Access Token.
2. Mit diesem Token führt er einen `GET`-Request auf `/admin/realms/ubisoft/users` aus, um die Benutzerliste des Realms zu erhalten.
3. Die Abfrage ist erfolgreich – bob erhält Zugriff auf die vollständige Liste und identifiziert seine eigene Benutzer-ID.
4. Anschließend versucht er über einen `POST`-Request an `/admin/realms/ubisoft/users/{id}` sich selbst die Rolle `realm-admin` zuzuweisen.
5. Der Request schlägt fehl – die Serverantwort ist `HTTP 401 Unauthorized`.

Technische Umsetzung: Die Requests wurden mithilfe von `curl` direkt an die REST API gesendet. Dabei wurde ein korrekt signiertes Access Token verwendet, das bob durch einen Login mit seiner `manage-users`-Rolle erhalten hatte.

Erwartetes Verhalten:

- Die Benutzerliste kann durch Rollen wie `manage-users` teilweise eingesehen werden, was ein potenzielles Datenschutzrisiko darstellt.
- Die unautorisierte Zuweisung privilegierter Rollen wird vom Server korrekt abgelehnt.

- Der Zugriff auf administrative Rollen über die REST API ist in Keycloak Version 26.1 restriktiver als in früheren Versionen, was diese Art von Privilege Escalation wirksam verhindert.

Sicherheitsrelevanz: Das Szenario zeigt, dass selbst privilegierte Benutzer mit erweiterten Rechten (z. B. `manage-users`) keine vollständigen administrativen Aufgaben übernehmen können. Dennoch bleibt der Zugriff auf sensible Informationen wie Benutzerlisten ein Risiko. Frühere Keycloak-Versionen waren anfällig für ähnliche Angriffe (z. B. CVE-2024-3656). Seit Version 26.1 wurden entsprechende Sicherheitsmechanismen gestärkt, um Missbrauch über die REST API zu verhindern.

9.4.2 Angriffsszenario: Account-Sabotage durch privilegierten Benutzer

In diesem Szenario wird untersucht, inwieweit ein privilegierter Benutzer (z. B. ein Administrator) absichtlich oder versehentlich Schaden an Benutzerkonten anrichten kann. Ziel ist es, das Verhalten von Keycloak bei kritischen administrativen Operationen zu beobachten und zu analysieren, welche Aktionen protokolliert werden.

Ziel: Simulierte Account-Sabotage durch einen legitimen Admin im Realm

Betroffener Realm: Sega

Angreiferrolle: Benutzer mit Administratorrechten (z. B. `realm-admin` oder `manage-users`)

Betroffene Benutzer: Admin in Sega (Profiländerung) sowie drei zufällige Benutzer (Löschung)

Ablauf:

1. Ein Angreifer kennt die Credentials des Admins im Realm **Sega** und ist bereits mit dessen Credentials eingeloggt.
2. Der Angreifer ändert das Passwort des Admins in Sega.
3. Anschließend löscht der Admin drei beliebige Benutzerkonten im selben Realm.
4. Der falsche Admin meldet sich ab.

Technische Umsetzung: Die Ausführung erfolgt entweder manuell über die Keycloak Admin Console oder automatisiert mit Hilfe von **Selenium**.

Erwartetes Verhalten:

- Die Profiländerungen und das Zurücksetzen des Passworts sollten als `UPDATE_PASSWORD` im **User-Event-Log** erscheinen.
- Die Löschung der Benutzerkonten wird im **Admin-Event-Log** als `DELETE` Operation registriert.
- Alle Aktionen sind über den Zeitstempel, Benutzer und IP-Adresse nachvollziehbar.
- *LOGOUT* wird ebenfalls in den Logs angezeigt

Sicherheitsrelevanz: Dieses Szenario demonstriert, wie gefährlich ein missbrauchter Admin-Zugang sein kann. In realen Umgebungen sollten Änderungen durch Admins auditiert und mit mehrstufigen Genehmigungsmechanismen oder Alerting-Systemen abgesichert sein.

9.4.3 Angriffsszenario: Brute-Force-Angriff über die Web-Oberfläche

In diesem Szenario wird ein klassischer Brute-Force-Angriff simuliert, bei dem ein Benutzername fest vorgegeben ist und systematisch unterschiedliche Passwörter ausprobiert werden. Ziel ist es zu prüfen, ob ein erfolgreiches Login durch mehrfaches Raten von Passwörtern möglich ist und ob Keycloak entsprechende Fehlversuche protokolliert.

Ziel: Test der Account-Sicherheit gegenüber einfachen Brute-Force-Angriffen über das Login-Formular

Betroffener Benutzer: admin

Realm: Dynamisch gewählt (aus mehreren Realms auswählbar)

Passwortliste: Eine Liste gängiger schwacher Passwörter (z. B. admin123, 123456, password, qwerty, welcome, etc.)

Ablauf:

1. Es wird eine Verbindung zur Keycloak-Login-Seite des jeweiligen Realms hergestellt.
2. Der Benutzername wird in das Formular eingetragen (**admin**).

3. Es wird nacheinander jedes Passwort aus der vorgegebenen Liste eingegeben und die Login-Schaltfläche betätigt.
4. Das Skript prüft, ob eine Fehlermeldung erscheint oder ob der Login erfolgreich war.
5. Bei Erfolg wird der Versuch gestoppt.

Technische Umsetzung: Die Automatisierung erfolgt mittels `Selenium WebDriver` in Kombination mit einer Python-Testfunktion. Die Login-Seite wird jeweils mit einem neuen Passwort geladen und automatisiert ausgefüllt.

Erwartetes Verhalten:

- Keycloak sollte nach mehreren Fehlversuchen Gegenmaßnahmen ergreifen (z. B. Account-Sperrung, Rate Limiting).
- Im Event-Log sollten die fehlgeschlagenen Login-Versuche als `LOGIN_ERROR` mit dem jeweiligen Benutzer verzeichnet sein.

Brute Force und Privilege Escalation sind Angriffe, die fehlschlagen. Nur Account Sabotage ist erfolgreich. Dies liegt an der Umsetzung (Brute Force durch Selenium-Tests) und daran, dass der erwähnte CVE Fall für neuere Keycloak Versionen nicht mehr funktioniert durch neue Sicherheitsfunktionen- dennoch stellen sie erkennbare, anomale Logs dar.

Sicherheitsrelevanz: Ein solcher Angriff simuliert reale Angriffsvektoren durch Bots oder Skripte. Das Vorhandensein und die Konfiguration von Mechanismen wie *Brute Force Detection*, Lockout Policies oder IP-Rate-Limits sind für die Sicherheit von zentraler Bedeutung.

10 Durchführung des Trainings

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.

Über einen Zeitraum von 12 Tagen (2.5 Wochen, nur Werktage) wurden ausschließlich „normale“ Logs generiert. An den darauffolgenden Tagen fanden gezielte Angriffsszenarien statt. Mithilfe eines Skripts wurden anschließend sämtliche Event-Logs aus allen Realms zusammengeführt.

Die kombinierten Logs wurden zunächst im Verhältnis 70 % zu Trainingsdaten und jeweils 15 % zu Test- und Validierungsdaten aufgeteilt. Die Anomalien wurden zufällig auf die Test- und Validierungsdaten verteilt, dabei jedoch in zeitlich korrekter Reihenfolge einsortiert. Da die Anomalien erst nachträglich erzeugt wurden, mussten die Timestamps entsprechend angepasst werden.

Eine rein zufällige Vergabe der Timestamps war dabei nicht möglich, da bestimmte Angriffsszenarien einen klaren zeitlichen Ablauf erfordern. Die Timestamps wurden daher gezielt so angepasst, dass sie diesen inhaltlichen Zusammenhang korrekt abbilden, ohne am Ende des Datensatzes zu kumulieren. Aufgrund der überschaubaren Anzahl an Anomalien (ca. 50–70 Einträge) konnte dieser Schritt manuell umgesetzt werden.

Bevor die anomalen Logs mit den normalen vermischt wurden, fügte das Skript Labelinformationen hinzu: In der Datei `attacks_logs.py` wurden die anomalen Ereignisse aus Keycloak extrahiert und mit dem Label "1" versehen, während in der Datei `normal_test_logs.py` die regulären Log-Einträge mit dem Label "0" markiert wurden.

Abschließend wurde die finale Datei in zwei Hälften aufgeteilt; in eine Validierungs- und einer Testdatei.

Das LSTM-AE wird dabei nur mit normalen Daten trainiert. Wie viele Trainingsdaten an sich auch nötig sind, ist ein weitgehend offenes Problem. Es gibt nur wenige Studien, die sich systematisch mit der Frage beschäftigt haben, welche Datenmenge „angemessen“ ist, um ein neuronales Netzwerk zuverlässig zu trainieren.

Nach der Studie von Golestaneh et al. [49] wurde eine theoretische Fehlerabschätzung für bestimmte Netzwerke abgeleitet. Demnach skaliert der Fehler ε eines solchen Netzwerks nicht mit der klassischen „parametrischen“ Rate von $\varepsilon \sim 1/n$, sondern mit einer langsameren, linearen Rate:

$$\varepsilon(n) \sim \frac{1}{\sqrt{n}}$$

Dabei ist n die Anzahl der Trainingsbeispiele. Diese Formel bedeutet, dass eine Verdopplung der Datenmenge den Fehler nicht halbiert, sondern nur um den Faktor

$1/\sqrt{2}$ verringert. Dennoch erlaubt diese Abschätzung eine grobe Einschätzung der notwendigen Datenmenge.

Wendet man diese Formel auf eine Trainingsmenge von $n = 10.000$ Log-Zeilen an, ergibt sich ein theoretischer Fehler von etwa

$$\varepsilon(10.000) \approx \frac{1}{\sqrt{10.000}} = 0,01$$

Dieser Wert gilt zwar nicht als absolute Fehlermarke (da der tatsächliche Fehler von Modellarchitektur, Datenqualität und Regularisierung abhängt), er ist jedoch in vielen praktischen Fällen bereits hinreichend gering, um eine sinnvolle Modellierung zu ermöglichen. Somit kann ein LSTM-Autoencoder bei geeigneter Modellwahl und sauberem Datenvorverarbeitung bereits mit etwa 10.000 normalisierten Log-Zeilen ein brauchbares Modell für Anomalieerkennung liefern.

Auch eine weitere Studie von Götz et al. [50] beschäftigt sich mit der Relation zwischen der Komplexität eines neuronalen Netzwerks (gemessen an der Anzahl der trainierbaren Parameter N_p) und der Anzahl der notwendigen Trainingsbeispiele N_{tr} . Dabei wurde gezeigt, dass kein starres Verhältnis wie $N_{tr} \geq N_p$ erforderlich ist, um gute Ergebnisse zu erzielen. Stattdessen weisen die Autoren nach, dass auch mit deutlich weniger Trainingsbeispielen eine stabile Modellleistung möglich.

In ihren Experimenten mit Convolutional Neural Networks (CNNs) konnten selbst bei einem Verhältnis $N_{tr}/N_p < 1$ noch vergleichbare Klassifikationsleistungen erzielt werden, solange die Reduktion der Netzwerkkomplexität kontrolliert erfolgt. Übertragen auf den vorliegenden Fall bedeutet dies: Bei einem moderat komplexen LSTM-Autoencoder mit z.B. einigen zehntausend Parametern können bereits 10.000 Trainingsbeispiele (z. B. Log-Zeilen) ausreichen, um ein funktionierendes Modell zu erhalten – sofern geeignete Regularisierung und Modellarchitektur verwendet werden.

Die Studie stützt damit ebenfalls die Annahme, dass in vielen praktischen Szenarien bereits Datensätze im Bereich von 10.000 bis 50.000 Einträgen als ausreichend gelten können, um eine robuste Anomalieerkennung durchzuführen-wobei man natürlich beachten muss, dass in jeder Studie der Fokus auf einem anderen Netzwerk lag. Bspw. basiert Golestanehs Formel auf ReLU-basierte Netzwerke. Dennoch will man diese Studien als Hinweis dafür wahrnehmen, möglichst viele Daten zu generieren, auch wenn es im Rahmen dieser Arbeit die Anzahl der wahrscheinlich nur im fünfstelligen Bereich liegen wird.

Auf Grundlage dieser Erkenntnisse wird ein Testfall mit ca. 39000 Logs erstellt.

Für den Testfall wurde mit 50 und 100 Epochen trainiert. Für den Test befinden sich in den Test - und Validierungsdaten befinden sich ca. 31 Log-Einträge, welche durch je eines der fünf Angriffsszenarien entsteht.

10.1 Änderung der Timestamps durch Unix Timestamp

Da die Timestamps manuell geändert werden müssen, muss gewährleistet sein, dass diese zufällig vom 22.07, ca. 14 Uhr bis zum 08.08, 17 Uhr erzeugt wurden- und demnach verändert werden. Es wurden folgende Zeitpunkte pro Angriffsszenario modelliert:

- *Brute Force Angriff*: Umstellung auf den 06.08 um 13 Uhr Abends
- *Privilege Escalation*: Umstellung auf den 07.08 um 16 Uhr
- *Account Sabotage*: Umstellung auf den 08.08 um 11 Uhr Morgens

Diese Zeitpunkte werden auf der Webseite Unix Timestamp ²⁸ verändert.

The Current Epoch Unix Timestamp

Enter a Timestamp

1754567250

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Convert →

1754567596

SECONDS SINCE JAN 01 1970. (UTC)

1:53:18 PM

Copy

Enter a Date & Time

Year Month Day Hour (24 hour) Minutes Seconds

2025 08 07 11 47 30

Convert →

The current epoch translates to

Abbildung 16: Aktuelle Uhrzeit nach dem Unix-System

Die erste Attacke dauert ca. 1-2 Minuten, die zweite ca. 5-10 Minuten und die letzte im ca. 20 Minuten. Diese Zeit wurde so bestimmt, dass sie sich für den jeweiligen Angriff eignen. Die Angriffe werden mit Absicht an Zeitpunkte durchgeführt, welche zwischen den normalen Logs mit Abstand sind, damit das LSTM-AE auch seinen Einsatz daran zeigen kann. Denn bei zu kurzen Sequenzen und auch plötzlichen, würde das LSTM-AE nur bei einer viel zu kurzen Sequenzlänge an Aussagekraft verlieren.

10.2 Testfälle

Aufgrund der ungleichmäßigen Verhältnis zwischen Anomalien und normalen Logs und der geringen Anzahl an Logs wurde beschlossen Testfälle mit unterschiedlichen Architekturen durchzugehen. Es werden folgende Sequenzlängen validiert:

- Sequenzlänge 10
- Sequenzlänge 25
- Sequenzlänge 50

Die Wahl der unterschiedlichen Sequenzlängen dient dazu, die Sensitivität des Modells auf verschiedene zeitliche Zusammenhänge zu testen. Kürzere Sequenzen erfassen lokale Muster, während längere Sequenzen komplexere zeitliche Abhängigkeiten abbilden können. Dadurch kann das Modell besser an die unterschiedlichen Charakteristika der Log-Daten angepasst werden.

Die Labels kennzeichnet jeden Log-Eintrag als Anomalie (1) oder normal (0). Alle weiteren Log-Merkmale wie Eventtyp oder Zeitstempel werden als unabhängige Variablen in das Modell eingespeist.

Die Architektur selbst beinhaltet schon Störfaktoren. Tiefe, Breite und auch die Batch-Size beeinflussen viel von dem, wie die Modelle insgesamt arbeiten.

Zur Übersicht wurden Tabellen erstellt, welche das Verhältnis zu den abhängigen und unabhängigen Variablen zeigen soll:

Variablen der Hybridmodelle:

Die Architektur des LSTM-Autoencoders wird bewusst minimal gehalten. Die Sequenzlänge wird auf 10 Log-Einträge gesetzt, um lokale zeitliche Muster effizient zu erfassen, ohne die Netzwerkkomplexität unnötig zu erhöhen. Die Batch-Size wird konstant auf 32 gehalten, da sie einen guten Kompromiss zwischen stabiler Gradientenberechnung und Trainingsgeschwindigkeit bietet und bei der Gesamtzahl von 39.000 Logeinträgen ausreichend robuste Updates ermöglicht.

Die Encoder- und Decoder-Layer sind jeweils mit [128, 64] bzw. [64, 128] Neuronen konfiguriert, wodurch das Modell leichtgewichtig bleibt und Overfitting vermieden wird, während gleichzeitig die typischen Muster in den Log-Sequenzen zuverlässig erfasst werden können. Dropout wird deaktiviert, um die Reproduzierbarkeit der Ergebnisse zu gewährleisten, sodass jeder Trainingsdurchlauf unter identischen Bedingungen erfolgt. Die Lernrate wird auf 0,001 gesetzt, ein bewährter Startwert, der stabile Konvergenz in Kombination mit der gewählten Architektur und Batch-Size

Testvariable- Unabhängige Variable	Konstanten
<ul style="list-style-type: none"> - Sequenzlänge 	<ul style="list-style-type: none"> - Tiefe des Netzwerkes - Breite des Netzwerkes - Batch-Size - Lernrate - Dropout-Rate - Art, wie der Schwellenwert berechnet wird - Art, wie der Rekonstruktionsfehler berechnet wird - Skalierung - Numerische Vorverarbeitung (Kein One Hot Encoding) - Epochenanzahl - Optimierer-Typ (Adam) - Verlustfunktion

Abbildung 17: Überblick der Konstanten Werte und der unabhängigen Variable

erlaubt.

Weitere Faktoren wie die Berechnung des Rekonstruktionsfehlers, der Schwellenwert, die Skalierung (StandardScaler), die Anzahl der Trainings-Epochen (50), der Optimierer (Adam) und die Verlustfunktion (MSE) werden konstant gehalten, um einen fairen Vergleich der Modelle zu gewährleisten. Die gewählten Konstanten können zwar die erzielbaren Höchstwerte im Vergleich zu anderen Studien begrenzen, gewährleisten jedoch, dass die Analyse zuverlässig und nachvollziehbar bleibt.

Kurz gesagt: Die abhängige Variable hier stellen die Metriken dar in den Logs dar, die unabhängige die Sequenzlänge.

11 Ergebnisse

11.1 Ergebnisse mit der Sequenzlänge 10

11.1.1 Testlauf 1

Metrik	Wert
Best Threshold (F1)	0.1861
Precision	0.9519
Recall	0.7734
F1	0.8534
ROC-AUC	0.9630
AUC-PR	0.7412
Matthews Correlation Coefficient (MCC)	0.8553
Balanced Accuracy	0.8863

Tabelle 2: Ergebnisse für IsolationForest

Metrik	Wert
Best Threshold (F1)	4999.3471
Precision	0.9364
Recall	0.8047
F1	0.8655
ROC-AUC	0.9109
AUC-PR	0.7577
Matthews Correlation Coefficient (MCC)	0.8654
Balanced Accuracy	0.9017

Tabelle 3: Ergebnisse für One-Class SVM

Metrik	Wert
Best Threshold (F1)	2.0099
Precision	1.0000
Recall	0.1484
F1	0.2585
ROC-AUC	0.5742
AUC-PR	0.1670
Matthews Correlation Coefficient (MCC)	0.3817
Balanced Accuracy	0.5742

Tabelle 4: Ergebnisse für DBSCAN

11.1.2 Testlauf 2

Metrik	Wert
Best Threshold (F1)	0.1850
Precision	0.9903
Recall	0.7969
F1	0.8831
ROC-AUC	0.9638
AUC-PR	0.7936
Matthews Correlation Coefficient (MCC)	0.8862
Balanced Accuracy	0.8984

Tabelle 5: Ergebnisse für IsolationForest

Metrik	Wert
Best Threshold (F1)	5444.4513
Precision	0.9903
Recall	0.7969
F1	0.8831
ROC-AUC	0.9184
AUC-PR	0.7936
Matthews Correlation Coefficient (MCC)	0.8862
Balanced Accuracy	0.8984

Tabelle 6: Ergebnisse für One-Class SVM

Metrik	Wert
Best Threshold (F1)	2.0995
Precision	1.0000
Recall	0.1250
F1	0.2222
ROC-AUC	0.5625
AUC-PR	0.1441
Matthews Correlation Coefficient (MCC)	0.3502
Balanced Accuracy	0.5625

Tabelle 7: Ergebnisse für DBSCAN

11.1.3 Testlauf 3

Metrik	Wert
Best Threshold (F1)	0.1934
Precision	0.9196
Recall	0.8047
F1	0.8583
ROC-AUC	0.9584
AUC-PR	0.7443
Matthews Correlation Coefficient (MCC)	0.8574
Balanced Accuracy	0.9016

Tabelle 8: Ergebnisse für IsolationForest

Metrik	Wert
Best Threshold (F1)	4082.3180
Precision	0.9043
Recall	0.8125
F1	0.8560
ROC-AUC	0.9168
AUC-PR	0.7389
Matthews Correlation Coefficient (MCC)	0.8542
Balanced Accuracy	0.9053

Tabelle 9: Ergebnisse für One-Class SVM

Metrik	Wert
Best Threshold (F1)	2.0106
Precision	1.0000
Recall	0.2188
F1	0.3590
ROC-AUC	0.6094
AUC-PR	0.2358
Matthews Correlation Coefficient (MCC)	0.4637
Balanced Accuracy	0.6094

Tabelle 10: Ergebnisse für DBSCAN

11.1.4 Ergebnisse mit der Sequenzlänge 25

11.1.5 Testdurchlauf 1

Metrik	Wert
Best Threshold (F1)	0.1764
Precision	0.9290
Recall	0.6305
F1	0.7512
ROC-AUC	0.9408
AUC-PR	0.6015
Matthews Correlation Coefficient (MCC)	0.7574
Balanced Accuracy	0.8142

Tabelle 11: Ergebnisse für IsolationForest, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	6275.1389
Precision	0.9401
Recall	0.6305
F1	0.7548
ROC-AUC	0.8532
AUC-PR	0.6085
Matthews Correlation Coefficient (MCC)	0.7622
Balanced Accuracy	0.8144

Tabelle 12: Ergebnisse für One-Class SVM, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	2.2846
Precision	1.0000
Recall	0.0442
F1	0.0846
ROC-AUC	0.5221
AUC-PR	0.0848
Matthews Correlation Coefficient (MCC)	0.2059
Balanced Accuracy	0.5221

Tabelle 13: Ergebnisse für DBSCAN, Sequenzlänge 25

11.1.6 Testlauf 2

Metrik	Wert
Best Threshold (F1)	0.1735
Precision	0.9811
Recall	0.6265
F1	0.7647
ROC-AUC	0.9410
AUC-PR	0.6306
Matthews Correlation Coefficient (MCC)	0.7771
Balanced Accuracy	0.8130

Tabelle 14: Ergebnisse für IsolationForest, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	5370.7273
Precision	0.9634
Recall	0.6345
F1	0.7651
ROC-AUC	0.8549
AUC-PR	0.6269
Matthews Correlation Coefficient (MCC)	0.7747
Balanced Accuracy	0.8167

Tabelle 15: Ergebnisse für One-Class SVM, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	2.1895
Precision	1.0000
Recall	0.0442
F1	0.0846
ROC-AUC	0.5221
AUC-PR	0.0848
Matthews Correlation Coefficient (MCC)	0.2059
Balanced Accuracy	0.5221

Tabelle 16: Ergebnisse für DBSCAN, Sequenzlänge 25

11.1.7 Testlauf 3

Metrik	Wert
Best Threshold (F1)	0.1793
Precision	0.9634
Recall	0.6345
F1	0.7651
ROC-AUC	0.9362
AUC-PR	0.6269
Matthews Correlation Coefficient (MCC)	0.7747
Balanced Accuracy	0.8167

Tabelle 17: Ergebnisse für IsolationForest, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	5983.1203
Precision	0.9467
Recall	0.6426
F1	0.7656
ROC-AUC	0.8553
AUC-PR	0.6235
Matthews Correlation Coefficient (MCC)	0.7725
Balanced Accuracy	0.8205

Tabelle 18: Ergebnisse für One-Class SVM, Sequenzlänge 25

Metrik	Wert
Best Threshold (F1)	2.1376
Precision	1.0000
Recall	0.0482
F1	0.0920
ROC-AUC	0.5241
AUC-PR	0.0887
Matthews Correlation Coefficient (MCC)	0.2150
Balanced Accuracy	0.5241

Tabelle 19: Ergebnisse für DBSCAN, Sequenzlänge 25

12 Diskussion

Die Ergebnisse zeigen deutliche Unterschiede in der Klassifikationsleistung zwischen den drei getesteten Verfahren und den beiden Sequenzlängen. Die Sequenzlänge beeinflusst die Feature-Repräsentation der Daten, was sich direkt auf die Trennbarkeit der Klassen auswirkt. Eine kürzere Sequenzlänge (10) kann feinere Anomaliesignale erfassen, führt jedoch auch zu einer höheren Varianz und potenziellen Fehlklassifikationen. Längere Sequenzen (25) glätten die Daten stärker, wodurch kurzzeitige Abweichungen schwerer erkennbar sind, stabile Muster jedoch konsistenter erfasst werden. Dieser Effekt könnte erklären, warum die Werte bei längerer Sequenzlänge leicht abfallen.

Die Ergebnisse der Tests für die Sequenzlänge 10 und 25 zeigen deutliche Unterschiede zwischen den drei untersuchten Modellen. Sowohl IsolationForest als auch die One-Class SVM liefern insgesamt gute Ergebnisse, während DBSCAN deutlich

schlechter abschneidet.

Bei IsolationForest und OCSVM liegen die Werte für F1, Precision und Recall konstant hoch. Das bedeutet, dass das Modell die meisten Anomalien richtig erkennt, gleichzeitig aber nur wenige normale Werte fälschlicherweise als Anomalien einstuft. Die ROC-AUC-Werte, die anzeigen, wie gut das Modell zwischen normalen und ungewöhnlichen Daten unterscheiden kann, sind ebenfalls sehr hoch. Auch der Matthews-Koeffizient bestätigt die gute Leistung dieses Modells.

Die One-Class SVM zeigt ein ähnliches Bild: Die Präzision ist sehr hoch und der Recall leicht besser als bei IsolationForest. Das heißt, dieses Modell erkennt etwas mehr Anomalien, hat dafür aber etwas niedrigere Werte bei der Gesamttrennschärfe (ROC-AUC).

DBSCAN hingegen zeigt deutlich schwächere Ergebnisse. Die Präzision ist zwar perfekt, doch der Recall ist sehr niedrig. Das bedeutet, dass DBSCAN fast keine Anomalien erkennt, weshalb das Modell für diese Daten und die gegebene Sequenzlänge nur bedingt geeignet ist.

Insgesamt lässt sich festhalten, dass für Sequenzlänge 10 und 25 derzeit IsolationForest und OCSVM die stabilsten und zuverlässigsten Modelle sind, wobei OCSVM in den Metriken MCC, F1-Score und Balanced Accuracy im Schnitt die höchsten Werte erzielte. Insgesamt erzielte IF jedoch im Gesamtschnitt die höchsten Werte unter allen Metriken- die Hypothese wurde also widerlegt. DBSCAN ist für diese Konfiguration nicht geeignet. Je nach Zielsetzung – ob man eher Wert auf das Erkennen möglichst vieler Anomalien (Recall) legt oder auf die Gesamtgenauigkeit – kann zwischen IsolationForest und One-Class SVM gewählt werden.

DBSCAN schneidet in beiden Szenarien deutlich schlechter ab. Da das Verfahren auf Dichtecustering basiert setzt es voraus, dass Anomalien in dünn besiedelten Regionen des Feature-Raums liegen. Bei hochdimensionalen Sequenzdaten verschwimmen diese Dichteunterschiede, wodurch die Clusterstruktur schwer erkennbar ist. Eventuell wurden zu wenige "Noises" eingebaut, wodurch es zu starke "normale" Logs gibt. Jedoch muss zudem erwähnt werden, dass es zu wenige Anomalien gibt, welche möglicherweise nicht erkannt werden konnten und daher nicht geclustert werden und nur ein Cluster entsteht. Dies deutet aber eher darauf hin, dass sich die Anomalien sogar in der Hinsicht den normalen Daten ähnlich sind. Ein weiterer Test wurde auch durchgeführt, jedoch nicht dokumentiert, bei dem DBSCAN mit den Parametern $\text{eps} = 0.05$ und $\text{min_samples} = 2$ schon mal eine Precision von über 70 Prozent erzielte, jedoch wurde dies nicht verwendet, da man gezielt ohne Parameter pro Algorithmus arbeiten wollte. Es wurde darauf geachtet, dass für jeden Algorithmus

der gleiche Ansatz benutzt wurde, um den Schwellenwert zu messen (durch Maximierung des F1-Scores) und dass so durch ein fairer Vergleich ermöglicht wurde. Mit einem anderen Verfahren hätte DBSCAN aber möglicherweise besser abgeschnitten-wurde jedoch nicht evaluiert und getestet.

Weitere Gründe könnten unter anderem darin liegen, dass die Tiefe und Breite des Modells möglicherweise zu gering war und das Training zudem nur über 50 Epochen erfolgte. Außerdem wurde das Verhältnis der Features nur unzureichend berücksichtigt. Die abhängige Variable stellt zwar das Label dar, welches den Log als anomal oder normal klassifiziert. Die IP-Adresse wurde jedoch entfernt, ebenso wie Features, die stärker auf Anomalien hinweisen könnten, wie etwa der Standort (in den Logs nicht angegeben) und die Zeit. Die Angriffsszenarien mussten daher innerhalb normaler, unauffälliger Zeiträume, beispielsweise mittags, stattfinden, um die Fähigkeiten des LSTM-Autoencoders wirklich testen zu können. Zu viele aufeinanderfolgende Anomalie-Punkte würden es dem LSTM-AE zu leicht machen. Daher spielen diese Features nur noch eine untergeordnete Rolle bei der Erkennung von Anomalien. Stattdessen beeinflussen Features wie der Name, der Eventtyp und besonders die anderen unabhängigen Variablen das Ergebnis. Es wurde jedoch nicht untersucht, in welchem Ausmaß einzelne Features dazu führen, dass die Modelle einen Log als anomal oder normal klassifizieren. Eventuell hätte zusätzlich ein ANOVA-Test durchgeführt werden sollen.

13 Fazit

Insgesamt zeigen die Ergebnisse, dass Hybridmodelle in der Lage sind, Anomalien in Keycloak-Logs zuverlässig zu erkennen. Beim direkten Vergleich der Kombinationen mit DBSCAN, OCSVM und Isolation Forest erzielte das Hybridmodell mit OCSVM die besten Ergebnisse in den untersuchten Metriken. Dabei spielen sowohl die Architektur des LSTM-Autoencoders als auch die gewählte Sequenzlänge eine entscheidende Rolle für die Erkennungsleistung, insbesondere angesichts des stark unbalancierten Datensatzes.

Die Methodik stellt sicher, dass sensible Unternehmensdaten nicht verwendet werden: Die Trainings- und Testdaten basieren auf generierten Logs, während durch Selenium-basierte Tests und die Keycloak-API authentische Log-Muster abgebildet werden. Insgesamt unterstreichen die Ergebnisse das Potenzial von LSTM-Autoencodern in Kombination mit klassischen Anomalieerkennungsverfahren für die automatisierte Überwachung von Keycloak-Logs und demonstrieren, dass die Wahl der Modellkombination und der Hyperparameter maßgeblich für die Performance ist.

14 Ausblick

In Zukunft könnten weitere Algorithmen oder alternative Architekturen in die Untersuchung aufgenommen werden. Mit einer größeren Menge an Trainings- und Testdaten ließe sich die Architektur weiter optimieren und verallgemeinern. Auf Basis der gewonnenen Erkenntnisse wäre es denkbar, ein praxistaugliches Anomalie-Erkennungstool zu entwickeln. Die konkrete Leistungsfähigkeit hängt jedoch stark vom jeweiligen Testszenario ab.

In dieser Arbeit wurde mit rund 39,000 Logeinträgen gearbeitet, wobei zwei feste Sequenzlängen und eine spezifische Architektur verwendet wurden. Unter anderen Bedingungen könnten sich die Erkennungsergebnisse deutlich unterscheiden. So hätte beispielsweise ein LSTM-AE mit Isolation Forest in einem anderen Szenario möglicherweise bessere Ergebnisse erzielt als OCSVM oder DBSCAN.

Daher erscheint es sinnvoll, weitere Tests mit komplexeren Angriffsszenarien, zusätzlichen Trainingsdaten und alternativen Methoden zur Schwellenwertbestimmung durchzuführen.

Bis dahin jedoch würden schon genug Studien veröffentlicht werden, welche ein ähnliches Thema schon behandeln.

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] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009, abgerufen am 10. Juni 2025. [Online]. Available: <https://doi.org/10.1145/1541880.1541882>
- [3] P. Legg, O. Buckley, M. Goldsmith, and S. Creese, “Visualizing the insider threat: challenges and tools for identifying malicious user activity,” in *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 2015, pp. 1–8, abgerufen am 10. Juni 2025.
- [4] B. Walzl, G. Bonczek, and F. Matthes, “Rule-based information extraction: Advantages, limitations, and perspectives,” in *Proceedings of the International Conference on Legal Knowledge and Information Systems (JURIX)*, Garching bei München, Germany, 2017, chair for Software Engineering for Business Information Systems. [Online]. Available: <https://www.matthes.in.tum.de>
- [5] C. Zhou and R. C. Paffenroth, “Deep learning for anomaly detection: A review,” *IEEE Access*, vol. 9, pp. 5789–5813, 2021, abgerufen am 23. Juni 2025. [Online]. Available: <https://arxiv.org/abs/2007.02500>
- [6] 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, abgerufen am 16. Juni 2025. [Online]. Available: https://www.researchgate.net/publication/378403423_A_Comparative_Study_of_Deep_Neural_Networks_and_Support_Vector_Machines_for_Unsupervised_Anomaly_Detection_in_Cloud_Computing_Environments
- [7] E. Demir and C. Karaoğlu, “A comparative performance analysis of lstm autoencoder and timegpt models in time series anomaly detection,” *ResearchGate*, 2024, abgerufen am 13. Juni 2025. [Online]. Available: https://www.researchgate.net/publication/385301485_A_Comparative_Performance_Analysis_of_LSTM_Autoencoder_and_TimeGPT_Models_in_Time_Series_Anomaly_Detection

- [8] T. Ergen and S. S. Kozat, “Unsupervised and semi-supervised anomaly detection with lstm neural networks,” *arXiv preprint arXiv:1710.09207*, 2017, abgerufen am 20. Juni 2025. [Online]. Available: <https://arxiv.org/abs/1710.09207>
- [9] Z. Ghrib, R. Jaziri, and R. Romdhane, “Hybrid approach for anomaly detection in time series data,” in *2020 IEEE International Conference on Intelligent Systems and Computer Vision (ISCV)*. IEEE, 2020, pp. 1–7, abgerufen am 20. Juni 2025.
- [10] W. Chua, A. L. D. Pajas, C. S. Castro, S. P. Panganiban, A. J. Pasuquin, M. J. Purganan, R. Malupeng, D. J. Pingad, J. P. Orolfo, H. H. Lua, and L. C. Velasco, “Web traffic anomaly detection using isolation forest,” *Informatics*, vol. 11, no. 4, p. 83, 2024, abgerufen am 23. Juni 2025. [Online]. Available: <https://www.mdpi.com/2227-9709/11/4/83>
- [11] Y. Wei, J. Jang-Jaccard, W. Xu, F. Sabrina, S. Camtepe, and M. Boulic, “Lstm-autoencoder-based anomaly detection for indoor air quality time series data,” *IEEE Sensors Journal*, vol. 23, no. 4, 2023, abgerufen am 25. Juni 2025.
- [12] 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)*. World Scientific, 2020, pp. 589–596, abgerufen am 25. Juni 2025.
- [13] D. N. Tran and S. Thomassey, “Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management,” *International Journal of Information Management*, vol. 57, p. 102282, 2021, abgerufen am 30. Juni 2025. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S026840122031481X>
- [14] G. Yan, Y. Zheng, and C. 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)*. IEEE, 2021, pp. 115–121, abgerufen am 30. Juni 2025. [Online]. Available: https://www.researchgate.net/publication/308409790_Detecting_Anomalous_User_Behavior_Using_an_Extended_Isolation_Forest_Algorithm_An_Enterprise_Case_Study

- [15] F. Provost, T. Fawcett, and R. Kohavi, “The case against accuracy estimation for comparing induction algorithms,” 1998, abgerufen am 28. Juni 2025. [Online]. Available: https://www.researchgate.net/publication/2373067_The_Case_Against_Accuracy_Estimation_for_Comparing_Induction_Algorithms
- [16] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002, abgerufen am 28. Juni 2025. [Online]. Available: <https://dl.acm.org/doi/10.5555/1293951.1293954>
- [17] D. Fourure, M. Mazurowski, J. Amar, and A. Derreumaux, “Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol,” *arXiv preprint arXiv:2106.16020*, 2021, abgerufen am 23. Juni 2025. [Online]. Available: <https://arxiv.org/abs/2106.16020>
- [18] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240, abgerufen am 28. Juni 2025. [Online]. Available: <https://dl.acm.org/doi/10.1145/1143844.1143874>
- [19] D. Chicco and G. Jurman, “The matthews correlation coefficient (mcc) is more informative than cohen’s kappa and brier score in binary classification assessment,” *BMC Genomics*, vol. 21, no. 1, p. 6, 2020, abgerufen am 28. Juni 2025. [Online]. Available: <http://ieeexplore.ieee.org/document/9440903>
- [20] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The balanced accuracy and its posterior distribution,” in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 3121–3124, abgerufen am 28. Juni 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/5597285>
- [21] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, abgerufen am 27. Juni 2025. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6795963>
- [22] R. C. Staudemeyer and E. R. Morris, “Understanding lstm – a tutorial into long short-term memory recurrent neural networks,” *arXiv preprint arXiv:1909.09586*, 2019, tutorial Paper. [Online]. Available: <https://arxiv.org/abs/1909.09586>
- [23] X. Wei, J. Liu, and L. Zhang, “Lstm autoencoder-based anomaly detection for indoor air quality data,” *arXiv preprint arXiv:2204.06701*, 2022, abgerufen am 13. Juni 2025. [Online]. Available: <https://arxiv.org/abs/2204.06701>

- [32] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” pp. 226–231, 1996, abgerufen am 27. Juni 2025. [Online]. Available: <https://dl.acm.org/doi/10.5555/3001460.3001507>
- [33] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, “Hierarchical density estimates for data clustering, visualization, and outlier detection,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–51, 2015, abgerufen am 05. August 2025. [Online]. Available: <https://doi.org/10.1145/2733381>
- [34] B. V. Sánchez Vinces, E. Schubert, A. Zimek, and R. L. F. Cordeiro, “A comparative evaluation of clustering-based outlier detection,” *Data Mining and Knowledge Discovery*, 2025, abgerufen am 05. August 2025. [Online]. Available: <https://doi.org/10.1007/s10618-024-01086-z>
- [35] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: Identifying density-based local outliers,” in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, May 2000, pp. 93–104, abgerufen am 05. August 2025.
- [36] E. H. Budiarto, A. E. Permanasari, and S. Fauziati, “Unsupervised anomaly detection using k-means, local outlier factor and one class svm,” *Department of Electrical Engineering and Information Technology, Universitas Gadjah Mada*, 2025, yogyakarta, Indonesia.
- [37] A. Zimek, E. Schubert, and H.-P. Kriegel, “A survey on unsupervised outlier detection in high-dimensional numerical data,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 5, pp. 363–387, 2012, abgerufen am 05. August 2025.
- [38] R. Mussabayev and R. Mussabayev, “Comparative analysis of optimization strategies for k-means clustering in big data contexts: A review,” *Journal of Big Data*, 2023, abgerufen am 05. August 2025.
- [39] A. unbekannt, “Analysis of the effectiveness and efficiency of lof algorithm for anomaly detection in large datasets,” in *Proceedings in SpringerLink*, 2023, effizienzprobleme bei großen Datensätzen mit LOF. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-50959-9_43
- [40] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, “Lstm-based encoder-decoder for multi-sensor anomaly detection,” *arXiv*

- preprint *arXiv:1607.00148*, 2016, abgerufen am 25. Juni 2025. [Online]. Available: <https://arxiv.org/abs/1607.00148>
- [41] H. Torabi, S. L. Mirtaheri, and S. Greco, “Practical autoencoder based anomaly detection by using vector reconstruction error,” *Journal of Big Data*, vol. 10, no. 1, pp. 1–21, 2023.
 - [42] A. Jablonski and K. Mendrok, “Automatic threshold setting for anomaly detection,” *AGH University of Krakow, Faculty of Mechanical Engineering and Robotics*, 2023, preprint.
 - [43] K. Wilkinghoff and K. Imoto, “F1-ev score: Measuring the likelihood of estimating a good decision threshold for semi-supervised anomaly detection,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 256–260. [Online]. Available: <https://arxiv.org/pdf/2312.09143.pdf>
 - [44] C. Y. Priyanto, Hendry, and H. D. Purnomo, “Combination of isolation forest and lstm autoencoder for anomaly detection,” in *2021 International Conference on Artificial Intelligence and Mechatronics Systems (AIMS)*. IEEE, 2021, abgerufen am 04. August 2025.
 - [45] G. Pu, L. Wang, J. Shen, and F. Dong, “A hybrid unsupervised clustering-based anomaly detection method,” in *Proceedings of the 2021 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 2021, abgerufen am 27. Juni 2025.
 - [46] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 770–778, 2016, abgerufen am 04. August 2025.
 - [47] S. A. Najafi, M. H. Asemani, and P. Setoodeh, “Attention and autoencoder hybrid model for unsupervised online anomaly detection,” *arXiv preprint arXiv:2301.XXXX*.
 - [48] A. P. Singh and A. Sharma, “A systematic literature review on insider threats,” 2022, abgerufen am 20. Juni 2025. [Online]. Available: <https://arxiv.org/abs/2212.05347>
 - [49] P. Golestaneh, M. Taheri, and J. Lederer, “How many samples are needed to train a deep neural network?” *arXiv preprint arXiv:2101.01235*, 2021, department of Mathematics, Computer Science, and Natural Sciences,

University of Hamburg, Germany. [Online]. Available: <https://arxiv.org/abs/2101.01235>

- [50] T. I. Götz, S. Göb, S. Sawant, X. F. Erick, T. Wittenberg, C. Schmidkonz, A. M. Tomé, E. W. Lang, and A. Ramming, “Number of necessary training examples for neural networks with different number of trainable parameters,” *Neurocomputing*, 2023, affiliations: Clinic of Nuclear Medicine, University Hospital Erlangen; CIML Group, University of Regensburg; Fraunhofer IIS, Erlangen; IEETA, Universidade de Aveiro; University Hospital Erlangen; Technical University of Applied Sciences Amberg-Weiden. [Online]. Available: <https://doi.org/10.1016/j.neucom.2023.02.087>

Fußnotenverzeichnis

- 1 <https://chatgpt.com/>
- 2 Keycloak Project: *Open Source Identity and Access Management*, <https://www.keycloak.org/>, letzter Zugriff: 30. Juli 2025.
- 3 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., letzter Zugriff: 30. Juli 2025.
- 4 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>, letzter Zugriff: 30. Juli 2025.
- 5 Quora, „What Are the Biggest Challenges Facing the Cybersecurity Industry?“, Forbes, 15. September 2017, <https://www.forbes.com/sites/quora/2017/09/15/what-are-the-biggest-challenges-fa>, letzter Zugriff: 30. Juli 2025.
- 6 Sciforce, „Adversarial Attacks Explained and How to Defend ML Models Against Them“, Medium, veröffentlicht am 20. Januar 2020, <https://medium.com/sciforce/adversarial-attacks-explained-and->, letzter Zugriff: 30. Juli 2025.
- 7 National Vulnerability Database (NVD), „CVE-2024-4629 Vulnerability Details“, <https://nvd.nist.gov/vuln/detail/CVE-2024-4629>, letzter Zugriff: 30. Juli 2025.
- 8 CVEDetails, „CVE-2025-3501 Details“, <https://www.cvedetails.com/cve/CVE-2025-3501>, letzter Zugriff: 30. Juli 2025.
- 9 Lukas Ruff, „Deep-SVDD-PyTorch“, GitHub Repository, <https://github.com/lukasruff/Deep-SVDD-PyTorch>, letzter Zugriff: 30. Juli 2025.
- 10 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.
- 11 Überanpassung des Modells an die Trainingsdaten, wodurch die Generalisierbarkeit auf neue Daten leidet.
- 12 Scikit-learn: Machine Learning in Python. Offizielle Dokumentation, verfügbar unter: <https://scikit-learn.org/stable/>, letzter Zugriff: 30. Juli 2025.
- 13 Keras: The Python Deep Learning API. Offizielle Dokumentation, verfügbar unter: <https://keras.io/>, letzter Zugriff: 30. Juli 2025.
- 14 TensorFlow: An End-to-End Open Source Machine Learning Platform. Offizielle Dokumentation, verfügbar unter: <https://www.tensorflow.org/>, letzter Zugriff: 30. Juli 2025.
- 15 NVIDIA Corporation, „DGX Systems“, <https://www.nvidia.com/en-us/data-center/dgx-systems/>, letzter Zugriff: 30. Juli 2025.

- 16 NVIDIA Corporation, „Introduction to DGX H100 User Guide“, technische Spezifikationen, <https://docs.nvidia.com/dgx/dgxm100-user-guide/introduction-to-dgxm100.html>, letzter Zugriff: 30. Juli 2025.
- 17 WildFly Project, <https://www.wildfly.org/>, letzter Zugriff: 30. Juli 2025.
- 18 Keycloak Documentation, „Server Logging“, <https://www.keycloak.org/server/logging>, letzter Zugriff: 30. Juli 2025.
- 19 Keycloak Community, „JpaAdminEventQuery.java“, GitHub Repository, <https://github.com/keycloak/keycloak/blob/main/model/jpa/src/main/java/org/keycloak/events/jpa/JpaAdminEventQuery.java>, letzter Zugriff: 30. Juli 2025.
- 20 <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>
- 21 <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>
- 22 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
- 23 <https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html>
- 24 Internet Assigned Numbers Authority (IANA), „IANA IPv4 Special Registry“, <https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml>, letzter Zugriff: 30. Juli 2025.
- 25 Keycloak Documentation, „REST API Reference“, <https://www.keycloak.org/docs-api/latest/rest-api/index.html>, letzter Zugriff: 30. Juli 2025.
- 26 MITRE Corporation, „CVE Search Results for Keycloak“, <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=keycloak>, letzter Zugriff: 30. Juli 2025.
- 27 <https://www.cve.news/cve-2024-3656/>
- 28 <https://www.unixtimestamp.com/>