

Credential-Management Lab mit MySQL und HashiCorp Vault

ITT-Net-IS

2025-05-01

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Einführung in das Credential-Management | 2 |
| 1.1 | Lernziele dieses Labs | 2 |
| 2 | Vorbereitung der Laborumgebung | 2 |
| 2.1 | Voraussetzungen | 2 |
| 2.2 | Installation der benötigten Tools | 3 |
| 2.3 | Projektstruktur | 3 |
| 3 | Teil 1: Problematischer Ansatz - Hartcodierte Credentials | 3 |
| 3.1 | Schritt 1: Aufsetzen der MySQL-Datenbank mit Docker | 3 |
| 3.2 | Schritt 2: Erstellen einer einfachen Python-Anwendung mit hartcodierten Credentials | 4 |
| 3.3 | Schritt 3: Die problematische Anwendung ausführen | 5 |
| 3.4 | Schritt 4: Die Probleme diskutieren | 5 |
| 4 | Teil 2: Verbesselter Ansatz - Verwendung von Umgebungsvariablen | 5 |
| 4.1 | Schritt 1: Refaktorisieren der Anwendung zur Verwendung von Umgebungsvariablen | 5 |
| 4.2 | Schritt 2: Die verbesserte Anwendung ausführen | 6 |
| 4.3 | Schritt 3: Die verbleibenden Probleme diskutieren | 7 |
| 5 | Teil 3: Sicherer Ansatz - HashiCorp Vault | 7 |
| 5.1 | Schritt 1: Aufsetzen von HashiCorp Vault mit Docker | 7 |
| 5.2 | Schritt 2: Konfigurieren von Vault | 7 |
| 5.3 | Schritt 3: Aktualisieren der Anwendung für die Verwendung von Vault | 9 |
| 5.4 | Schritt 4: Vault initialisieren und die verbesserte Anwendung ausführen | 11 |
| 5.5 | Schritt 5: Die Vorteile von HashiCorp Vault diskutieren | 12 |
| 6 | Teil 4: Best Practices für Credential-Management | 12 |
| 6.1 | 1. Niemals Credentials im Quellcode speichern | 12 |
| 6.2 | 2. Das Prinzip der geringsten Privilegien anwenden | 13 |
| 6.3 | 3. Regelmäßige Rotation von Credentials | 13 |
| 6.4 | 4. Sichere Übertragung von Credentials | 13 |
| 6.5 | 5. Überwachung und Protokollierung | 13 |
| 7 | Teil 5: Übungen für die Lernenden | 13 |
| 7.1 | Übung 1: Implementierung eines Read-Only-Zugriffs | 13 |
| 7.2 | Übung 2: Credential-Leakage simulieren | 13 |
| 7.3 | Übung 3: Implementierung von automatischer Credential-Rotation | 13 |

| | | |
|----------|---|-----------|
| 7.4 | Übung 4: Integration mit einer CI/CD-Pipeline | 13 |
| 8 | Fazit | 13 |
| 8.1 | Zusammenfassung | 13 |
| 8.2 | Weiterführende Ressourcen | 14 |
| 9 | Anhang: Erweiterungsmöglichkeiten | 14 |
| 9.1 | Verwendung von Azure Key Vault oder AWS Secrets Manager | 14 |
| 9.2 | Integration mit Kubernetes | 14 |
| 9.3 | Multi-Environment-Setup | 14 |
| 9.4 | Erweiterte Authentifizierungsmethoden | 14 |

1 Einführung in das Credential-Management

Anwendungen benötigen häufig Zugriff auf Datenbanken und andere sensible Systeme. Dafür werden Zugangsdaten (Credentials) benötigt, die oft unsicher verwaltet werden:

- Hartcodierte Credentials im Quellcode
- Unverschlüsselte Config-Dateien
- Umgebungsvariablen ohne ausreichenden Schutz
- Unsichere Weitergabe von Zugangsdaten im Team

Diese Praktiken führen zu erheblichen Sicherheitsrisiken:

- Versehentliche Veröffentlichung sensibler Daten in Code-Repositories
- Unbefugter Zugriff auf Produktionssysteme
- Fehlende Möglichkeit zur Nachverfolgung von Credential-Nutzung
- Schwierigkeiten beim Rotieren von Zugangsdaten

1.1 Lernziele dieses Labs

- Verständnis für die Probleme unsicherer Credential-Verwaltung entwickeln
- Sichere Alternativen mit HashiCorp Vault kennenlernen
- Praktische Erfahrung mit einer Vault-Integration in Python-Anwendungen sammeln
- Best Practices für Credential-Management in der Entwicklung anwenden

2 Vorbereitung der Laborumgebung

2.1 Voraussetzungen

- Windows-Betriebssystem
- VSCode installiert
- Docker Desktop installiert und konfiguriert
- Grundlegende Python-Kenntnisse
- Grundlegende SQL-Kenntnisse

2.2 Installation der benötigten Tools

1. VSCode-Erweiterungen:
 - Python Extension (ms-python.python)
 - Docker Extension (ms-azuretools.vscode-docker)
 - Remote Development Extension Pack (empfohlen)
2. Benötigte Dateien herunterladen:
 - Die Lab-Dateien können von [LINK ZUM REPOSITORY] heruntergeladen werden
 - Alternativ: Die einzelnen Dateien aus diesem Dokument kopieren

2.3 Projektstruktur

```
credential-management-lab/
├── docker-compose.yml
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── config.py
│   ├── database.py
│   └── requirements.txt
├── mysql/
│   └── init.sql
├── vault/
│   └── config.hcl
└── README.md
```

3 Teil 1: Problematischer Ansatz - Hartcodierte Credentials

3.1 Schritt 1: Aufsetzen der MySQL-Datenbank mit Docker

Erstellen Sie eine 'docker-compose.yml' Datei:

```
1 version: '3'
2
3 services:
4     mysql:
5         image: mysql:8.0
6         container_name: mysql-db
7         environment:
8             MYSQL_ROOT_PASSWORD: rootpassword
9             MYSQL_DATABASE: inventory
10            MYSQL_USER: app_user
11            MYSQL_PASSWORD: app_password
12        ports:
13            - "3306:3306"
14        volumes:
15            - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
16        networks:
17            - app-network
18
```

```
19 networks:
20     app-network:
21         driver: bridge
```

Erstellen Sie die 'mysql/init.sql' Datei:

```
1 USE inventory;
2
3 CREATE TABLE products (
4     id INT AUTO_INCREMENT PRIMARY KEY,
5     name VARCHAR(100) NOT NULL,
6     price DECIMAL(10, 2) NOT NULL,
7     quantity INT NOT NULL
8 );
9
10 INSERT INTO products (name, price, quantity) VALUES
11     ('Laptop', 999.99, 10),
12     ('Smartphone', 499.99, 20),
13     ('Headphones', 99.99, 50),
14     ('Tablet', 299.99, 15);
15
16 CREATE USER 'readonly_user'@'%' IDENTIFIED BY 'readonly_password';
17 GRANT SELECT ON inventory.products TO 'readonly_user'@'%';
18
19 CREATE USER 'admin_user'@'%' IDENTIFIED BY 'admin_password';
20 GRANT ALL PRIVILEGES ON inventory.* TO 'admin_user'@'%';
```

3.2 Schritt 2: Erstellen einer einfachen Python-Anwendung mit hartcodierten Credentials

Erstellen Sie die Datei 'app/database.py':

```
1 import mysql.connector
2
3 def connect_to_database():
4     # PROBLEM: Hartcodierte Credentials im Code
5     connection = mysql.connector.connect(
6         host="mysql",
7         database="inventory",
8         user="admin_user",
9         password="admin_password" # Sensible Information im Klartext!
10    )
11    return connection
12
13 def get_all_products():
14     connection = connect_to_database()
15     cursor = connection.cursor(dictionary=True)
16     cursor.execute("SELECT * FROM products")
17     products = cursor.fetchall()
18     cursor.close()
19     connection.close()
20     return products
```

Erstellen Sie die Datei 'app/main.py':

```
1 from database import get_all_products
2
3 def show_all_products():
4     try:
5         products = get_all_products()
6         print("\n== Produkte im Inventar ==")
```

```

7         for product in products:
8             print(f"ID: {product['id']}, Name: {product['name']}, "
9                   f"Preis: €{product['price']}, Menge: {product['quantity']}")
10        except Exception as e:
11            print(f"Fehler beim Abrufen der Produkte: {e}")
12
13    if __name__ == "__main__":
14        show_all_products()

```

Erstellen Sie die Datei 'app/requirements.txt':

mysql-connector-python==8.0.32

3.3 Schritt 3: Die problematische Anwendung ausführen

1. Starten Sie die Docker-Container:

```
1 docker-compose up -d
```

1. Bauen Sie ein Docker-Image für die Anwendung:

```
1 docker build -t credential-app -f Dockerfile.app .
```

1. Führen Sie die Anwendung aus:

```
1 docker run --network credential-management-lab_app-network credential-app
```

3.4 Schritt 4: Die Probleme diskutieren

Identifizieren Sie die folgenden Probleme:

- Die Zugangsdaten sind im Quellcode sichtbar
- Bei Versionskontrolle werden die Credentials mit eingecheckt
- Bei einer Änderung der Zugangsdaten muss der Code angepasst werden
- Keine Trennung zwischen Entwicklungs-, Test- und Produktionsumgebung
- Keine Möglichkeit, die Nutzung der Credentials zu protokollieren

4 Teil 2: Verbesselter Ansatz - Verwendung von Umgebungsvariablen

4.1 Schritt 1: Refaktorisieren der Anwendung zur Verwendung von Umgebungsvariablen

Erstellen Sie die Datei 'app/config.py':

```

1 import os
2
3 # Konfiguration über Umgebungsvariablen
4 DB_HOST = os.environ.get('DB_HOST', 'mysql')
5 DB_NAME = os.environ.get('DB_NAME', 'inventory')
6 DB_USER = os.environ.get('DB_USER', 'admin_user')
7 DB_PASSWORD = os.environ.get('DB_PASSWORD') # Kein Default-Wert für Passwörter!
8
9 def validate_config():
10     if not DB_PASSWORD:
11         raise ValueError("Die Umgebungsvariable DB_PASSWORD muss gesetzt sein!")

```

Aktualisieren Sie die Datei 'app/database.py':

```
1 import mysql.connector
2 from config import DB_HOST, DB_NAME, DB_USER, DB_PASSWORD, validate_config
3
4 def connect_to_database():
5     # Überprüfen Sie, ob alle erforderlichen Konfigurationsparameter vorhanden
6     # sind
7     validate_config()
8
9     # Verwenden Sie Umgebungsvariablen statt hartcodierter Werte
10    connection = mysql.connector.connect(
11        host=DB_HOST,
12        database=DB_NAME,
13        user=DB_USER,
14        password=DB_PASSWORD
15    )
16    return connection
17
18 def get_all_products():
19    connection = connect_to_database()
20    cursor = connection.cursor(dictionary=True)
21    cursor.execute("SELECT * FROM products")
22    products = cursor.fetchall()
23    cursor.close()
24    connection.close()
25    return products
```

4.2 Schritt 2: Die verbesserte Anwendung ausführen

1. Erstellen Sie ein Dockerfile:

```
FROM python:3.9-slim

WORKDIR /app

COPY app/requirements.txt .
RUN pip install -r requirements.txt

COPY app/ .

ENV DB_PASSWORD=admin_password

CMD ["python", "main.py"]
```

1. Bauen Sie das Docker-Image:

```
1 docker build -t credential-app-env .
```

1. Führen Sie die Anwendung aus:

```
1 docker run --network credential-management-lab_app-network credential-app-env
```

4.3 Schritt 3: Die verbleibenden Probleme diskutieren

Obwohl dieser Ansatz besser ist als hartcodierte Credentials, bleiben Probleme:

- Umgebungsvariablen sind für alle Prozesse auf dem System sichtbar
- Passwörter können in Shell-Historien landen
- Keine automatische Rotation von Credentials
- Keine Protokollierung der Credential-Nutzung
- Docker-Images können Umgebungsvariablen in ihren Metadaten speichern

5 Teil 3: Sicherer Ansatz - HashiCorp Vault

5.1 Schritt 1: Aufsetzen von HashiCorp Vault mit Docker

Erweitern Sie Ihre 'docker-compose.yml' Datei:

```
1 version: '3'
2
3 services:
4   mysql:
5     image: mysql:8.0
6     container_name: mysql-db
7     environment:
8       MYSQL_ROOT_PASSWORD: rootpassword
9       MYSQL_DATABASE: inventory
10      MYSQL_USER: app_user
11      MYSQL_PASSWORD: app_password
12     ports:
13       - "3306:3306"
14     volumes:
15       - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
16     networks:
17       - app-network
18
19   vault:
20     image: hashicorp/vault:1.13
21     container_name: vault
22     ports:
23       - "8200:8200"
24     environment:
25       VAULT_DEV_ROOT_TOKEN_ID: myroot
26       VAULT_DEV_LISTEN_ADDRESS: 0.0.0.0:8200
27     cap_add:
28       - IPC_LOCK
29     networks:
30       - app-network
31
32 networks:
33   app-network:
34     driver: bridge
```

5.2 Schritt 2: Konfigurieren von Vault

Erstellen Sie ein Setup-Skript 'setup_{vault.sh}':

```
1 #!/bin/bash
2
3 # Warten, bis Vault gestartet ist
4 sleep 5
5
6 # Vault-Adresse und Token setzen
7 export VAULT_ADDR=http://vault:8200
8 export VAULT_TOKEN=myroot
9
10 # KV Secrets Engine aktivieren
11 vault secrets enable -path=secret kv-v2
12
13 # MySQL Secrets in Vault speichern
14 vault kv put secret/mysql/admin \
15     user=admin_user \
16     password=admin_password
17
18 vault kv put secret/mysql/readonly \
19     user=readonly_user \
20     password=readonly_password
21
22 # Database Secrets Engine aktivieren
23 vault secrets enable database
24
25 # MySQL-Verbindung konfigurieren
26 vault write database/config/mysql \
27     plugin_name=mysql-database-plugin \
28     connection_url="{{username}}:{{password}}@tcp(mysql:3306)/" \
29     allowed_roles="readonly,admin" \
30     username="root" \
31     password="rootpassword"
32
33 # Readonly-Rolle erstellen
34 vault write database/roles/readonly \
35     db_name=mysql \
36     creation_statements="CREATE_USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';\n
37     GRANT_SELECT_ON_inventory.* TO '{{name}}'@'%' ;" \
38     default_ttl="1h" \
39     max_ttl="24h"
40
41 # Admin-Rolle erstellen
42 vault write database/roles/admin \
43     db_name=mysql \
44     creation_statements="CREATE_USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';\n
45     GRANT_ALL_PRIVILEGES_ON_inventory.* TO '{{name}}'@'%' ;" \
46     default_ttl="1h" \
47     max_ttl="24h"
48
49 # AppRole Auth Method aktivieren
50 vault auth enable approle
51
52 # Policies erstellen
53 vault policy write readonly-policy -<<EOF
54 path "secret/data/mysql/readonly" {
55     capabilities = ["read"]
56 }
57
58 path "database/creds/readonly" {
```



```

57     capabilities = ["read"]
58 }
59 EOF
60
61 vault policy write admin-policy -<<EOF
62 path "secret/data/mysql/admin" {
63     capabilities = ["read"]
64 }
65
66 path "database/creds/admin" {
67     capabilities = ["read"]
68 }
69 EOF
70
71 # AppRoles erstellen
72 vault write auth/approle/role/readonly \
73     token_policies=readonly-policy \
74     token_ttl=1h \
75     token_max_ttl=4h
76
77 vault write auth/approle/role/admin \
78     token_policies=admin-policy \
79     token_ttl=1h \
80     token_max_ttl=4h
81
82 # AppRole IDs und Secrets abrufen und anzeigen
83 READONLY_ROLE_ID=$(vault read -format=json auth/approle/role/readonly/role-id | jq
84     -r '.data.role_id')
85 READONLY_SECRET_ID=$(vault write -format=json -f auth/approle/role/readonly/secret
86     -id | jq -r '.data.secret_id')
87
88 ADMIN_ROLE_ID=$(vault read -format=json auth/approle/role/admin/role-id | jq -r '.
89     data.role_id')
90 ADMIN_SECRET_ID=$(vault write -format=json -f auth/approle/role/admin/secret-id |
91     jq -r '.data.secret_id')
92
93 echo "Readonly_Role_ID:_$READONLY_ROLE_ID"
94 echo "Readonly_Secret_ID:_$READONLY_SECRET_ID"
95 echo "Admin_Role_ID:_$ADMIN_ROLE_ID"
96 echo "Admin_Secret_ID:_$ADMIN_SECRET_ID"

```

5.3 Schritt 3: Aktualisieren der Anwendung für die Verwendung von Vault

Aktualisieren Sie 'app/requirements.txt':

```

mysql-connector-python==8.0.32
hvac==1.1.0
python-dotenv==1.0.0

```

Erstellen Sie eine neue Datei 'app/.env':

```

VAULT_ADDR=http://vault:8200
VAULT_ROLE_ID=<admin_role_id>
VAULT_SECRET_ID=<admin_secret_id>

```

Erstellen Sie eine neue Datei 'app/vault_client.py':

```

1 import os
2 import hvac
3 from dotenv import load_dotenv
4
5 load_dotenv()
6
7 def get_vault_client():
8     """Verbindung zum Vault-Server herstellen und authentifizieren"""
9     client = hvac.Client(url=os.environ.get('VAULT_ADDR', 'http://vault:8200'))
10
11     # Mit AppRole authentifizieren
12     role_id = os.environ.get('VAULT_ROLE_ID')
13     secret_id = os.environ.get('VAULT_SECRET_ID')
14
15     if not role_id or not secret_id:
16         raise ValueError("VAULT_ROLE_ID und VAULT_SECRET_ID müssen gesetzt sein")
17
18     # AppRole-Authentifizierung durchführen
19     client.auth.approle.login(
20         role_id=role_id,
21         secret_id=secret_id
22     )
23
24     return client
25
26 def get_static_credentials(path):
27     """Statische Credentials von Vault abrufen"""
28     client = get_vault_client()
29     response = client.secrets.kv.v2.read_secret_version(path=path)
30     return response['data']['data']
31
32 def get_dynamic_credentials(path):
33     """Dynamische Credentials von Vault abrufen"""
34     client = get_vault_client()
35     response = client.read(path)
36     return response['data']

```

Aktualisieren Sie 'app/database.py':

```

1 import mysql.connector
2 from vault_client import get_static_credentials, get_dynamic_credentials
3
4 def connect_with_static_credentials():
5     """Verbindung mit statischen Credentials herstellen"""
6     # Credentials aus Vault abrufen
7     creds = get_static_credentials('mysql/admin')
8
9     # Mit den abgerufenen Credentials verbinden
10    connection = mysql.connector.connect(
11        host="mysql",
12        database="inventory",
13        user=creds['user'],
14        password=creds['password']
15    )
16    return connection
17
18 def connect_with_dynamic_credentials():
19     """Verbindung mit dynamischen Credentials herstellen"""
20     # Dynamische Credentials für die Admin-Rolle erstellen

```

```

21 creds = get_dynamic_credentials('database/creds/admin')
22
23 # Mit den dynamisch erstellten Credentials verbinden
24 connection = mysql.connector.connect(
25     host="mysql",
26     database="inventory",
27     user=creds['username'],
28     password=creds['password']
29 )
30 return connection
31
32 def get_all_products(use_dynamic=True):
33     """Alle Produkte aus der Datenbank abrufen"""
34     if use_dynamic:
35         connection = connect_with_dynamic_credentials()
36     else:
37         connection = connect_with_static_credentials()
38
39     cursor = connection.cursor(dictionary=True)
40     cursor.execute("SELECT * FROM products")
41     products = cursor.fetchall()
42     cursor.close()
43     connection.close()
44     return products

```

Aktualisieren Sie 'app/main.py':

```

1 from database import get_all_products
2 import argparse
3
4 def show_all_products(use_dynamic):
5     try:
6         products = get_all_products(use_dynamic=use_dynamic)
7         print("\n=== Produkte im Inventar ===")
8         for product in products:
9             print(f>ID: {product['id']}, Name: {product['name']}, "
10                  f"Preis: €{product['price']}, Menge: {product['quantity']}")
11
12         credential_type = "dynamischen" if use_dynamic else "statischen"
13         print(f"\nErfolgreich mit {credential_type} Credentials verbunden!")
14     except Exception as e:
15         print(f"Fehler beim Abrufen der Produkte: {e}")
16
17 if __name__ == "__main__":
18     parser = argparse.ArgumentParser(description='Credential Demo mit HashiCorp Vault')
19     parser.add_argument('--static', action='store_true', help='Statische Credentials verwenden')
20     args = parser.parse_args()
21
22     show_all_products(not args.static)

```

5.4 Schritt 4: Vault initialisieren und die verbesserte Anwendung ausführen

1. Starten Sie die Docker-Container:

```
1 docker-compose up -d
```

1. Führen Sie das Vault-Setup aus:

```
1 docker cp setup_vault.sh vault:/tmp/  
2 docker exec vault sh -c "chmod +x /tmp/setup_vault.sh&&/tmp/setup_vault.sh"
```

1. Notieren Sie die ausgegebenen Role IDs und Secret IDs und aktualisieren Sie die '.env'-Datei.

2. Bauen Sie ein Docker-Image für die Vault-Anwendung:

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

```
COPY app/requirements.txt .
```

```
RUN pip install -r requirements.txt
```

```
COPY app/ .
```

```
CMD ["python", "main.py"]
```

1. Bauen Sie das Docker-Image:

```
docker build -t credential-app-vault .
```

1. Führen Sie die Anwendung mit dynamischen Credentials aus:

```
docker run --network credential-management-lab_app-network credential-app-vault
```

1. Führen Sie die Anwendung mit statischen Credentials aus:

```
docker run --network credential-management-lab_app-network credential-app-vault --st
```

5.5 Schritt 5: Die Vorteile von HashiCorp Vault diskutieren

- Keine Passwörter im Code oder in Umgebungsvariablen
- Temporäre, dynamisch generierte Credentials mit begrenzter Lebensdauer
- Automatische Rotation von Credentials
- Detaillierte Zugriffskontrolle über Policies
- Protokollierung aller Credential-Zugriffe
- Zentrale Verwaltung von Secrets für verschiedene Systeme
- Unterstützung für verschiedene Authentifizierungsmethoden

6 Teil 4: Best Practices für Credential-Management

6.1 1. Niemals Credentials im Quellcode speichern

- Trennung von Code und Konfiguration
- Verwendung von Secret-Management-Lösungen wie HashiCorp Vault

6.2 2. Das Prinzip der geringsten Privilegien anwenden

- Nur die minimal notwendigen Berechtigungen vergeben
- Verschiedene Benutzer für verschiedene Zugriffsstufen

6.3 3. Regelmäßige Rotation von Credentials

- Automatisierte Rotation mit Tools wie HashiCorp Vault
- Kurzlebige, dynamisch generierte Credentials verwenden

6.4 4. Sichere Übertragung von Credentials

- Immer verschlüsselte Verbindungen verwenden (TLS/SSL)
- Vermeidung von unverschlüsselten E-Mails oder Messaging-Diensten

6.5 5. Überwachung und Protokollierung

- Alle Zugriffe auf Credentials protokollieren
- Ungewöhnliche Zugriffsversuche überwachen

7 Teil 5: Übungen für die Lernenden

7.1 Übung 1: Implementierung eines Read-Only-Zugriffs

Modifizieren Sie die Anwendung, um mit einem Read-Only-Benutzer zu arbeiten.

7.2 Übung 2: Credential-Leakage simulieren

Simulieren Sie ein versehentliches Commit von Credentials und diskutieren Sie die Konsequenzen.

7.3 Übung 3: Implementierung von automatischer Credential-Rotation

Erweitern Sie die Anwendung, um mit abgelaufenen Credentials umzugehen und neue anzufordern.

7.4 Übung 4: Integration mit einer CI/CD-Pipeline

Diskutieren Sie, wie Vault in einer CI/CD-Pipeline verwendet werden kann, ohne Credentials preiszugeben.

8 Fazit

8.1 Zusammenfassung

- Unsichere Credential-Management-Praktiken stellen ein erhebliches Sicherheitsrisiko dar
- HashiCorp Vault bietet eine sichere und flexible Lösung für das Credential-Management
- Best Practices wie das Prinzip der geringsten Privilegien und regelmäßige Credential-Rotation sind entscheidend

8.2 Weiterführende Ressourcen

- HashiCorp Vault Dokumentation
- OWASP Cheat Sheet zu Secret Management
- Twelve-Factor App - Coqnfing
- Python hvac Bibliothek

9 Anhang: Erweiterungsmöglichkeiten

9.1 Verwendung von Azure Key Vault oder AWS Secrets Manager

Alternative Cloud-basierte Secret-Management-Lösungen

9.2 Integration mit Kubernetes

Verwendung von Vault mit Kubernetes über den Vault Injector

9.3 Multi-Environment-Setup

Verschiedene Konfigurationen für Entwicklung, Test und Produktion

9.4 Erweiterte Authentifizierungsmethoden

Implementierung von TLS- oder JWT-basierter Authentifizierung