README für das LDAP-Gruppenberechtigungs-Skript

Übersicht

Das Skript ldap_group_permissions.sh dient dazu, LDAP (Lightweight Directory Access Protocol)-Berechtigungen für verschiedene Abteilungen und Benutzergruppen innerhalb einer Organisation zu konfigurieren. Es wendet Zugriffssteuerungslisten (ACLs) an, um die entsprechenden Berechtigungen für jede Gruppe auf dem LDAP-Server festzulegen.

Das Skript verwendet den Befehl ldapmodify, um LDAP-Einträge zu ändern und die richtigen Zugriffsrechte für jede Gruppe festzulegen, sodass die Benutzer die entsprechenden Berechtigungen für den Zugriff auf ihre jeweiligen Ressourcen erhalten.

Funktionen

- **Automatisierte LDAP-Konfiguration**: Das Skript wendet automatisch LDAP-Zugriffsberechtigungen für vordefinierte Gruppen an.
- **Anpassbare Berechtigungen**: Jede Abteilung oder Gruppe hat spezifische Berechtigungen (z.B. lesen, schreiben, verwalten).
- **Unterstützt mehrere Abteilungen**: Berechtigungen werden für IT, Management, Verwaltung, Personalabteilung und weitere Gruppen konfiguriert.
- **Einfach erweiterbar**: Neue Gruppen können leicht hinzugefügt werden, indem das Array DEPARTMENTS erweitert und die erforderlichen Berechtigungen definiert werden.

Verwendung

Voraussetzungen

1. Stelle sicher, dass du Zugriff auf den LDAP-Server haben und über die notwendigen Admin-Zugangsdaten verfügen, um Änderungen vornehmen zu können.
2. Installiere das Dienstprogramm ldapmodify auf Ihrem System, falls es noch nicht installiert ist:
3. Vergewissere dich, dass der LDAP-Server läuft und korrekt konfiguriert ist.
4. Das Skript muss mit ausreichenden Rechten ausgeführt werden, um LDAP-Einträge zu ändern.

Konfiguration des Skripts

Bevor du das Skript ausführen, stelle sicher, dass die folgenden Parameter korrekt im Skript gesetzt sind:

- **LDAP-Admin-Zugangsdaten**: Passe die Variablen LDAP_ADMIN und LDAP_PASSWORD an, um die tatsächlichen Zugangsdaten für den LDAP-Admin-Benutzer zu hinterlegen.
- **LDAP Base DN**: Setzedie Variable BASE_DN auf den korrekten Basis-Domainnamen (DN) der Organisation.

  LDAP_ADMIN="cn=admin,dc=yourdomain,dc=com"

  LDAP_PASSWORD="YourAdminPassword"

  BASE_DN="dc=yourdomain,dc=com"

Abteilungs- und Gruppenberechtigungen

Das Skript enthält eine vordefinierte Liste von Abteilungen/Gruppen und deren entsprechende Berechtigungen im Array DEPARTMENTS. Jede Abteilung erhält spezifische Berechtigungen:

- **IT**: Lese-, Schreib- und Ausführungsrechte.
- **IT-Management**: Vollständige Administratorrechte.
- **LDAP-Administratoren**: Vollzugriff auf die LDAP-Datenbank.
- **Web-Administratoren**: Schreibrechte (für Docker-bezogene Aufgaben).
- **HR, Buchhaltung, Verwaltung**: Zugriff auf ihre eigenen Abteilungsordner.
- **Geschäftsführung**: Lesezugriff auf eigene und spezielle Abteilungsordner.
- **DAU (Benutzer mit geringerem Zugang)**: Nur Authentifizierungsrechte.

Ausführung des Skripts

1. Öffnen eine Terminal-Sitzung und navigiere zum Verzeichnis, in dem sich das Skript befindet.
2. Stelle sicher, dass das Skript ausführbar ist:

   chmod +x permissions.sh

3. Führe das Skript aus:

   sudo bash permissions.sh

Das Skript wendet dann die entsprechenden Berechtigungen für jede Gruppe auf dem LDAP-Server an.

Beispiel

Ein Beispiel für die Verwendung von Berechtigungen für die Gruppe "it" könnte wie folgt aussehen:

```
apply_group_permissions "it" "read,write,execute"
```

Dieser Befehl gewährt der Gruppe "it" Lese-, Schreib- und Ausführungsrechte auf die entsprechenden LDAP-Einträge.

Lizenz

Dieses Skript steht unter der MIT-Lizenz.

Die MIT-Lizenz ist eine sehr einfache und populäre Open-Source-Lizenz, die Entwicklern erlaubt, den Code frei zu verwenden, zu modifizieren und weiterzugeben. Dabei gelten folgende Bedingungen:

1. **Erlaubnisse**: Der Code darf kostenlos verwendet, kopiert, verändert und weitervertrieben werden, auch in kommerziellen Projekten.
2. **Pflicht**: In jeder Kopie oder jedem abgeleiteten Werk muss der ursprüngliche Copyright-Hinweis sowie die Lizenz erhalten bleiben.
3. **Haftungsausschluss**: Der ursprüngliche Autor übernimmt keine Haftung für Schäden, die durch die Verwendung des Codes entstehen.

Technische README für suricata_config.sh

Übersicht

Das Bash-Skript suricata_config.sh installiert und konfiguriert die Netzwerküberwachungssoftware **Suricata** auf einem Linux-System für das Netzwerkinterface enp0s8. Suricata ist ein leistungsstarkes Tool zur Erkennung von Eindringversuchen, zur Überwachung von Netzwerkverkehr und zur Durchführung von Netzwerkforensik. Dieses Skript automatisiert die Installation, Konfiguration und Aktivierung des Suricata-Dienstes sowie den Download und die Einbindung der Regeln.

Skript-Funktionen

Das Skript führt folgende Schritte aus:

1. **Systemaktualisierung**: Aktualisiert die Systempakete, um sicherzustellen, dass alle installierten Pakete auf dem neuesten Stand sind.
2. **Installation von Suricata**: Installiert die neueste Version von Suricata über das Paketverwaltungstool apt.
3. **Konfiguration von Suricata**: Konfiguriert Suricata so, dass es das Netzwerkinterface enp0s8 überwacht.
4. **Herunterladen von Suricata-Regeln**: Lädt die neuesten Bedrohungsregeln von **Emerging Threats** herunter.
5. **Einbindung der Regeln**: Fügt die heruntergeladenen Regeln in die Konfigurationsdatei von Suricata ein.
6. **Start und Aktivierung des Dienstes**: Startet den Suricata-Dienst und stellt sicher, dass er beim Systemstart automatisch ausgeführt wird.
7. **Statusüberprüfung**: Zeigt den Status des Suricata-Dienstes an.
8. **Protokollüberwachung**: Gibt den Befehl zur Überwachung der Suricata-Logs aus.

Voraussetzungen

- **Betriebssystem**: Das Skript wurde für Debian-basierte Systeme (wie Ubuntu) entwickelt.
- **Netzwerkinterface**: Es wird standardmäßig das Interface enp0s8 verwendet. Du kannst dies jedoch im Skript anpassen, falls dein System ein anderes Interface verwendet.
- **Root-Rechte**: Du benötigst Root- oder Sudo-Rechte, um das Skript auszuführen, da es Systemänderungen vornimmt (z.B. Installation von Paketen, Änderungen an Konfigurationsdateien).

Verwendung

1. Skriptvorbereitung

1. Kopiere das Skript in eine Datei mit dem Namen suricata_config.sh.
2. Stelle sicher, dass das Skript ausführbar ist :
   chmod +x suricata_config.sh

2. Ausführung des Skripts

Führe das Skript mit folgenden Schritten aus:
sudo ./suricata_config.sh

Das Skript wird die Schritte in der folgenden Reihenfolge durchführen:

1. **Systemaktualisierung**: Führt apt-get update und apt-get upgrade aus, um sicherzustellen, dass dein System auf dem neuesten Stand ist.
2. **Suricata-Installation**: Installiert das Suricata-Paket.
3. **Suricata-Konfiguration**: Konfiguriert Suricata so, dass es den Netzwerkverkehr auf enp0s8 überwacht.
4. **Regeln-Download**: Lädt die **Emerging Threats**-Regeln für Suricata herunter.
5. **Regeln in Konfiguration einbinden**: Bindet die heruntergeladenen Regeln in die suricata.yaml-Konfigurationsdatei ein.
6. **Dienststart**: Startet den Suricata-Dienst und aktiviert ihn für den automatischen Start.
7. **Dienststatus**: Zeigt den aktuellen Status von Suricata an.
8. **Log-Anzeige**: Gibt den Befehl aus, um die Suricata-Protokolle in Echtzeit zu überwachen.

Beispielausgabe des Skripts

Updating the system...

Installing Suricata...

Configuring Suricata...

Downloading Suricata rules...

Adding rule files to the configuration...

Starting Suricata...

Checking the status of Suricata...

suricata.service - LSB: Suricata Intrusion Detection Service

Loaded: loaded (/etc/init.d/suricata; generated)

Active: active (running)

You can monitor Suricata logs using:

tail -f /var/log/suricata/suricata.log

Suricata installation and configuration completed.

Protokollüberwachung

Nachdem das Skript die Installation abgeschlossen hat, kannst du die Suricata-Protokolle überwachen, um sicherzustellen, dass alles ordnungsgemäß funktioniert:

tail -f /var/log/suricata/suricata.log

Anpassungen

Falls du ein anderes Netzwerkinterface als enp0s8 verwenden möchtest, kannst du die folgende Zeile im Skript anpassen:

sudo sed -i 's/^\(interface:\)./\1 enp0s8/' /etc/suricata/suricata.yaml*

Ersetze enp0s8 durch das gewünschte Interface (z.B. eth0 oder wlan0).

Lizenz

Dieses Skript steht unter der MIT-Lizenz.

Die MIT-Lizenz ist eine sehr einfache und populäre Open-Source-Lizenz, die Entwicklern erlaubt, den Code frei zu verwenden, zu modifizieren und weiterzugeben. Dabei gelten folgende Bedingungen:

1. **Erlaubnisse**: Der Code darf kostenlos verwendet, kopiert, verändert und weitervertrieben werden, auch in kommerziellen Projekten.
2. **Pflicht**: In jeder Kopie oder jedem abgeleiteten Werk muss der ursprüngliche Copyright-Hinweis sowie die Lizenz erhalten bleiben.
3. **Haftungsausschluss**: Der ursprüngliche Autor übernimmt keine Haftung für Schäden, die durch die Verwendung des Codes entstehen.

README - Sicherheitsanwendungen konfigurieren: aa_config.py

Übersicht

Das Skript aa_config.py automatisiert die Konfiguration mehrerer Sicherheitsanwendungen und die Einrichtung von Firewall-Regeln auf einem Linux-Server. Es richtet die **Uncomplicated Firewall (UFW)** ein und verwaltet die Dienste **Snort**, **Suricata**, **ClamAV** und **RKHunter**, um den Server zu schützen und den Netzwerkverkehr sowie Systemintegrität zu überwachen.

Voraussetzungen

- **Python 3.6+**: Das Skript erfordert mindestens Python 3.6.
- **Administratorrechte**: Root- oder sudo-Berechtigungen sind notwendig, um Firewall-Einstellungen zu ändern und Systemdienste zu verwalten.
- **Sicherheitsanwendungen**: Folgende Pakete müssen auf dem Server installiert sein:
  - **UFW**: Firewall
  - **Snort**: Intrusion Detection System
  - **Suricata**: Netzwerk-Sicherheitsüberwachung
  - **ClamAV**: Antivirus-Software
  - **RKHunter**: Rootkit-Scanner

Installationsbefehle:

```
sudo apt-get install ufw snort suricata clamav clamav-daemon rkhunter
```

Funktionsweise

1. Shell-Befehle ausführen (run_command)

Die Funktion run_command(command) übernimmt die Ausführung von Shell-Befehlen im Terminal. Sie prüft, ob der Befehl erfolgreich abgeschlossen wurde, und gibt eine Fehlermeldung aus, falls dies nicht der Fall ist.

```
def run_command(command):
    try:
        subprocess.run(command, check=True, shell=True)
        print(f"Erfolgreich ausgeführt: {command}")
    except subprocess.CalledProcessError as e:
        print(f"Fehler bei der Ausführung von {command}: {e}")
```

2. UFW konfigurieren (configure_ufw)

Diese Funktion richtet die Firewall ein, um den Zugriff auf bestimmte Dienste und Ports zu ermöglichen, während andere Verbindungen blockiert werden. Die Konfiguration umfasst:

- **Blockieren** eingehender Verbindungen
- **Erlauben** ausgehender Verbindungen
- Freigabe von Ports für:
    - **SSH** (22/tcp)
    - **LDAP** (389/tcp)
    - **LDAPS** (636/tcp)
    - **HTTP** (80/tcp)
    - **HTTPS** (443/tcp)
    - **ClamAV Daemon** (3310/tcp)
    - **Wazuh-Agent** (55000/tcp)

```
def configure_ufw():
run_command("sudo ufw default deny incoming")
run_command("sudo ufw default allow outgoing")
run_command("sudo ufw allow 22/tcp")
run_command("sudo ufw allow 389/tcp")
run_command("sudo ufw allow 636/tcp")
run_command("sudo ufw allow 80/tcp")
run_command("sudo ufw allow 443/tcp")
run_command("sudo ufw allow 3310/tcp")
run_command("sudo ufw allow 55000/tcp")
run_command("sudo ufw logging on")
run_command("sudo ufw enable")
run_command("sudo ufw status verbose")
```

3. Sicherheitsdienste konfigurieren

Das Skript konfiguriert und startet verschiedene Sicherheitsdienste:

Snort konfigurieren: Intrusion Detection System

```
def configure_snort():
```

```
        run_command("sudo systemctl enable snort")
        run_command("sudo systemctl start snort")
```

Suricata konfigurieren: Alternative zu Snort für Netzwerküberwachung

```
    def configure_suricata():
        run_command("sudo systemctl enable suricata")
        run_command("sudo systemctl start suricata")
```

ClamAV konfigurieren: Antivirus-Dienst zur Bedrohungserkennung

```
    def configure_clamav():
        run_command("sudo systemctl enable clamav-daemon")
        run_command("sudo systemctl start clamav-daemon")
```

RKHunter konfigurieren: Überprüfung des Systems auf Rootkits

```
        def configure_rkhunter():
            run_command("sudo rkhunter --update")
            run_command("sudo rkhunter –propupd")
```

Hauptfunktion (main())

Die Hauptfunktion führt alle oben beschriebenen Konfigurationen nacheinander aus.

```
    def main():
            print("Starte die Konfiguration von SSH, LDAP und
    Sicherheitsanwendungen...")
            configure_ufw()
           configure_snort()
            configure_suricata()
            configure_clamav()
            configure_rkhunter()
            print("Konfiguration abgeschlossen.")
```

Nutzung

1. **Skript ausführen**:

   sudo python3 aa_config.py

Das Skript muss mit sudo-Rechten ausgeführt werden, um Firewall-Einstellungen und Systemdienste zu verwalten.

- **Statusüberprüfung**:
- UFW-Status:

  sudo ufw status verbose

- ***Status der Sicherheitsdienste:***

  sudo systemctl status snort

  - ***sudo systemctl status suricata***
  - ***sudo systemctl status clamav-daemon***

Fehlerbehebung

- **Fehler bei der Ausführung von Befehlen**: Wenn Befehle fehlschlagen, stellt sicher, dass die entsprechenden Pakete installiert sind und das Skript mit Administratorrechten ausgeführt wird.
- **Überprüfen der Systemprotokolle**: Bei Problemen mit Diensten kannst du die Systemprotokolle einsehen:

  sudo journalctl -xe

Lizenz

Dieses Skript steht unter der MIT-Lizenz.

Die MIT-Lizenz ist eine sehr einfache und populäre Open-Source-Lizenz, die Entwicklern erlaubt, den Code frei zu verwenden, zu modifizieren und weiterzugeben. Dabei gelten folgende Bedingungen:

1. **Erlaubnisse**: Der Code darf kostenlos verwendet, kopiert, verändert und weitervertrieben werden, auch in kommerziellen Projekten.
2. **Pflicht**: In jeder Kopie oder jedem abgeleiteten Werk muss der ursprüngliche Copyright-Hinweis sowie die Lizenz erhalten bleiben.
3. **Haftungsausschluss**: Der ursprüngliche Autor übernimmt keine Haftung für Schäden, die durch die Verwendung des Codes entstehen.

README - Firewall-Konfiguration: configure_ufw.py

Übersicht

Das Skript `configure_ufw.py` automatisiert die Konfiguration der **Uncomplicated Firewall (UFW)** für die Verwendung mit verschiedenen Diensten, einschließlich LDAP, Snort, ClamAV, Fail2ban und Docker. Es richtet standardmäßige Sicherheitsregeln ein und öffnet spezifische Ports für diese Dienste. Ziel des Skripts ist es, den Zugriff auf kritische Dienste zu steuern und die System-Sicherheit zu erhöhen.

Voraussetzungen

- **Python 3.6+**: Das Skript setzt die Verwendung von Python-Version 3.6 oder höher voraus.
- **UFW**: Die Uncomplicated Firewall (UFW) muss auf dem System installiert und konfiguriert sein
- **Administratorrechte**: Da das Skript Firewall-Regeln ändert, ist `sudo` oder Root-Zugriff erforderlich.

Skriptfunktionalitäten

`configure_ufw()`

Die Hauptfunktion des Skripts konfiguriert UFW mit folgenden Regeln:

1. **Aktivierung von UFW**:
    - UFW wird aktiviert, falls es nicht bereits aktiv ist.
    - Alle **eingehenden Verbindungen** werden standardmäßig blockiert, um das System zu schützen.
    - **Ausgehende Verbindungen** werden standardmäßig erlaubt, um normale Kommunikation nach außen zu ermöglichen.
2. **Spezifische Dienste und Ports**:
    - **LDAP und LDAPS**: Öffnen der Ports 389 (LDAP) und 636 (LDAPS) für eingehende Verbindungen.
    - **ClamAV**: Öffnen des Ports 3310 für den ClamAV-Daemon, falls externe Verbindungen notwendig sind.
    - **SSH**: SSH-Zugriff wird erlaubt, um sicherzustellen, dass Remote-Verbindungen weiterhin möglich sind.
    - **Docker**: Öffnen des Docker API Ports 2375, falls erforderlich. Weitere Docker-spezifische Ports können je nach Bedarf konfiguriert werden.
3. **Dienste ohne spezifische Regeln**:

- **Snort**: Da Snort den Netzwerkverkehr passiv überwacht, benötigt es keine direkten Port-Freigaben.
- **Rkhunter**: Arbeitet intern und benötigt ebenfalls keine spezifischen Firewall-Regeln.
- **Fail2ban**: Fail2ban erstellt eigene Firewall-Regeln, um verdächtige IP-Adressen zu blockieren.

4. **Überprüfung des UFW-Status**:
    - Nach Abschluss der Konfiguration zeigt das Skript den aktuellen Status von UFW und die angewendeten Regeln an.

Hauptfunktion (`main()`)

Die Hauptfunktion initialisiert die Konfiguration und gibt während des gesamten Prozesses Statusmeldungen aus.

Ausführung

1. **Firewall konfigurieren**: Führe das Skript mit Administratorrechten aus, um die Firewall-Regeln anzuwenden:

   sudo python3 configure_ufw.py

Überprüfung der Konfiguration: Nach Abschluss des Skripts kannst du den Status der UFW auch manuell überprüfen:

sudo ufw status verbose

Anpassungen

- **Weitere Ports freigeben**: Um zusätzliche Ports für bestimmte Dienste oder Anwendungen zu öffnen, kann der entsprechende UFW-Befehl hinzugefügt werden:

  subprocess.run(['sudo', 'ufw', 'allow', 'port_num/tcp'], check=True)

- **Docker-spezifische Ports**: Docker-Container können je nach Anwendung andere Ports verwenden, die zusätzlich freigegeben werden müssen

Fehlersuche

- **UFW ist bereits aktiviert**: Falls UFW bereits aktiviert ist, wird das Skript dies erkennen und die Konfiguration fortsetzen.

- **Fehlende Administratorrechte**: Ohne Root- oder `sudo`-Zugriff kann das Skript keine Firewall-Regeln ändern. Stelle sicher, dass das Skript mit Administratorrechten ausgeführt wird.

Lizenz

Dieses Skript steht unter der MIT-Lizenz.

Die MIT-Lizenz ist eine sehr einfache und populäre Open-Source-Lizenz, die Entwicklern erlaubt, den Code frei zu verwenden, zu modifizieren und weiterzugeben. Dabei gelten folgende Bedingungen:

1. **Erlaubnisse**: Der Code darf kostenlos verwendet, kopiert, verändert und weitervertrieben werden, auch in kommerziellen Projekten.
2. **Pflicht**: In jeder Kopie oder jedem abgeleiteten Werk muss der ursprüngliche Copyright-Hinweis sowie die Lizenz erhalten bleiben.
3. **Haftungsausschluss**: Der ursprüngliche Autor übernimmt keine Haftung für Schäden, die durch die Verwendung des Codes entstehen.

README - create_users.py

Übersicht

Das Skript create_users.py dient dazu, eine Reihe von Benutzern in einem LDAP-Verzeichnisdienst (Lightweight Directory Access Protocol) automatisch hinzuzufügen. Es überprüft, ob ein Benutzer bereits existiert, und fügt diesen bei Bedarf hinzu. Die Benutzerinformationen sind fest im Skript definiert und umfassen wichtige LDAP-Attribute wie uid, cn, sn, gidNumber, uidNumber und homeDirectory.

Voraussetzungen

- **Python 3.6+**: Das Skript wurde für Python-Versionen ab 3.6 entwickelt.
- **LDAP-Dienst**: Ein funktionierender LDAP-Server muss konfiguriert sein.
- **LDAP-Berechtigungen**: Der ausführende Benutzer benötigt Administratorrechte, um Benutzer in LDAP hinzuzufügen. Dies wird durch den Befehl ldapadd ermöglicht.
- **LDAP-Tools**: Die LDAP-Befehle ldapsearch und ldapadd müssen auf dem System verfügbar sein.

Skriptfunktionalitäten

Benutzerliste

Im Skript ist eine vordefinierte Liste von Benutzern hinterlegt, welche dem LDAP-Verzeichnis hinzugefügt werden sollen. Jeder Benutzer hat folgende Attribute:

- **uid**: Eindeutige Benutzerkennung
- **cn**: Vollständiger Name (Common Name)
- **sn**: Nachname (Surname)
- **userPassword**: Passwort des Benutzers
- **gidNumber**: Gruppen-ID des Benutzers
- **homeDirectory**: Heimatverzeichnis des Benutzers
- **uidNumber**: Eindeutige Benutzer-ID

Funktionen

check_user_exists(uid)

Diese Funktion überprüft, ob der Benutzer bereits im LDAP-Verzeichnis existiert, indem ein LDAP-Suchbefehl (ldapsearch) ausgeführt wird.

- Rückgabe: True, falls der Benutzer existiert, sonst False.

add_user(user)

Diese Funktion erstellt und führt den Befehl ldapadd aus, um einen neuen Benutzer im LDAP-Verzeichnis anzulegen. Die Benutzerdaten werden im LDIF-Format (LDAP Data Interchange Format) übergeben.

- Bei Erfolg wird eine Erfolgsmeldung ausgegeben.
- Bei Fehlern während der Ausführung wird eine Fehlermeldung ausgegeben.

main()

Die Hauptfunktion des Skripts durchläuft die vordefinierte Liste von Benutzern. Für jeden Benutzer wird überprüft, ob er bereits existiert. Falls nicht, wird er dem LDAP-Verzeichnis hinzugefügt.

Ausführung

1. **Installiere die benötigten LDAP-Tools**:

   bash
   Code kopieren
   sudo apt-get install ldap-utils

2. **LDAP-Server konfigurieren**: Stelle sicher, dass ein LDAP-Server läuft und die Verbindung ordnungsgemäß funktioniert.

3. **Skript ausführen**:

   python3 create_users.py

Anpassungen

- **Benutzerliste erweitern**: Um weitere Benutzer hinzuzufügen, erweitere einfach die Liste users im Skript mit den entsprechenden Informationen.
- **LDAP-Bind-Daten anpassen**: Falls der Administrator-DN (Distinguished Name) oder der Servername abweicht, muss die Zeile in der Funktion add_user entsprechend angepasst werden:

  ["ldapadd", "-x", "-D", "cn=admin,dc=hamster,dc=panzer", "-W"]

Fehlersuche

- **Benutzer existiert bereits**: Wenn ein Benutzer bereits im LDAP-Verzeichnis existiert, wird eine Meldung im Terminal angezeigt und der Benutzer wird übersprungen.
- **Fehler beim Hinzufügen eines Benutzers**: Falls beim Hinzufügen eines Benutzers ein Fehler auftritt, wird eine Fehlermeldung ausgegeben. Überprüfe, ob der LDAP-Dienst läuft und die richtigen Berechtigungen vorhanden sind.

Lizenz

Dieses Skript steht unter der MIT-Lizenz.

Die MIT-Lizenz ist eine sehr einfache und populäre Open-Source-Lizenz, die Entwicklern erlaubt, den Code frei zu verwenden, zu modifizieren und weiterzugeben. Dabei gelten folgende Bedingungen:

1. **Erlaubnisse**: Der Code darf kostenlos verwendet, kopiert, verändert und weitervertrieben werden, auch in kommerziellen Projekten.
2. **Pflicht**: In jeder Kopie oder jedem abgeleiteten Werk muss der ursprüngliche Copyright-Hinweis sowie die Lizenz erhalten bleiben.
3. **Haftungsausschluss**: Der ursprüngliche Autor übernimmt keine Haftung für Schäden, die durch die Verwendung des Codes entstehen.