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.