Fachhochschule Südwestfalen
University of Applied Sciences

Entwicklung eines Konfigurationsgenerators für WireGuard VPN

Ausarbeitung für das Modul *Projekt Anwendungsentwicklung* des *sechsten Fachsemesters* im Studiengang *Informatik B.Sc.*, *Sommersemester 2022*

Eingereicht bei *Prof. Dr. rer. nat. Uwe Klug*

Fachhochschule Südwestfalen am Standort Iserlohn,
Fachbereich Informatik und Naturwissenschaften

Verfasst von *Jonas Michel Berger*, Matrikelnummer: *30080342,*
E-Mail Adresse: berger.jonasmichel@fh-swf.de

Lüdenscheid, den 23.05.2022

Inhaltsverzeichnis

1. Einleitung ... 3

2. Anforderungsaufnahme ... 4

3. Anwendungsfälle ... 5

1. Neue Konfiguration erstellen ... 5

2. Bestehende Konfiguration importieren ... 5

3. Konfiguration ändern ... 5

4. Entfernung einer Konfiguration ... 6

5. Konfiguration einsehen ... 6

6. Anpassung der Netzwerkgröße ... 6

7. Änderung eines Schlüsselpaares ... 6

8. Export der Konfiguration ... 6

4. Bedienung des Programms ... 6

5. Datenmodell ... 7

6. Algorithmen ... 10

7. Implementierung ... 12

8. Testfälle ... 12

1. Import einer Konfiguration ... 12

2. Hinzufügen eines Clients ... 12

3. Entfernen von Clients ... 13

4. Anzeige von validen QR-Codes ... 13

9. Ausführung des Programms ... 13

10. Anwenderdokumentation ... 14

1. Installation ... 14

1. Nativ ... 14
2. Docker ... 15

2. Ausführung ... 15

1. Nativ ... 15
2. Docker ... 15

3. Bedienung ... 15

Quellenverzeichnis ... 15

1. Einleitung

Das Internet verbindet heutzutage Milliarden von Menschen miteinander und ermöglicht es, von nahezu jedem Ort der Welt eine Datenverbindung zu einem anderen Ziel aufzubauen. Vieles kann heute über eine Datenverbindung erledigt werden, wofür früher ein Standortwechsel notwendig war. Voraussetzung dafür ist, dass grundlegenden Schutzziele der IT-Sicherheit wie Authentizität (Echtheit) und Vertraulichkeit dauerhaft garantiert werden können. Für viele alltäglich verwendete Protokolle zur Datenübertragung wie das hypertext transfer protocol HTTP gibt es Alternativen, welche eine Transportverschlüsselung einsetzen. Diese stellt einen sicheren Tunnel zur Verfügung, durch welche Daten unverändert und vertraulich zum Ziel transportiert werden können. Leider stehen solche Alternativen nicht für jeden Anwendungsfall zur Verfügung, sodass eine Kommunikation über das Internet in solchen Fällen ein Risiko birgt. Auch besteht die Problematik, dass die Endpunkte für die Verbindungen aus dem Internet öffentlich erreichbar sein müssen. Sind diese Endpunkte durch Sicherheitslücken verwundbar, könnten Angreifer die dahinterliegenden Systeme kompromittieren. Die Sicherheitsrisiken lassen sich durch die Einführung einer zusätzlichen Sicherheitsschicht minimieren. Zusätzlich bringt diese zusätzliche Schicht den Vorteil, die unverschlüsselten Metadaten der darunterliegenden Kommunikation (Daten der Verbindungspartner, ggf. Informationen über den Inhalt der Kommunikation) ebenfalls für Außenstehende unlesbar zu machen. In der Praxis wird eine zusätzliche Sicherheitsschicht meist kostengünstig durch ein virtuelles privates Netzwerk VPN erreicht. Eine Firma kann ihren Mitarbeitern, welche von zuhause oder unterwegs arbeiten, beispielsweise den Zugriff auf das hauseigene ERP-System nur über eine VPN-Verbindung erlauben. So ist die Vertraulichkeit der Datenübertragung auch aus unsicheren Quellen, z.B. einem öffentlichen Drahtlosnetzwerk in einem Hotel, sichergestellt.

Es existieren einige Softwarelösungen für VPN. Bekannte und weitverbreitete Lösungen sind OpenVPN[1] und IPsec[2]. Der Vorteil beider Protokolle ist die hohe Verbreitung. Für viele Betriebssysteme stehen Client- und Serveranwendungen zur Verfügung. Es bestehen jedoch auch Nachteile. Beide Softwareprojekte besitzen eine große Codebasis und eine hohe Komplexität bei der Konfiguration von Verbindungen. Durch die hohe Verbreitung und das Alter der Softwareprojekte gibt es gerade bei der Einrichtung von IPsec-Verbindungen einige Fehlerursachen, welche bei der Auswahl der Verschlüsselungsprotokolle entstehen. Das im Vergleich zu OpenVPN und IPsec neue (erste Versionen wurden 2016 veröffentlicht) VPN-Protokoll WireGuard verwendet eine kleinere Codebasis und fest definierte Verschlüsselungsalgorithmen, welche auf elliptischen Kurven[3] basieren. Das für den beim

[1] Webseite des Projekts OpenVPN Community: https://community.openvpn.net/openvpn
[2] RFC 4301 „Security Architecture for the Internet Protocol“: https://datatracker.ietf.org/doc/html/rfc4301
[3] RFC 7748 “Elliptic Curves for Security”: https://datatracker.ietf.org/doc/html/rfc7748

Verbindungsaufbau durchgeführten Schlüsselaustausch verwendete Curve25519 Verfahren ist für seine hohe Geschwindigkeit bekannt und wird daher auch von vielen weiteren Softwareprojekten und -protokollen (u.A. Signal, WhatsApp, Threema, Tor, SSH) verwendet. Das WireGuard Protokoll ist aufgrund der statischen Auswahl der Verschlüsselungsprotokolle selbst sehr performant. Darüber hinaus verwendetet es das verbindungslose UDP Protokoll, welches im Gegensatz zu TCP ebenfalls Geschwindigkeitsvorteile liefert. Im Jahr 2020 wurde es durch Linus Torvalds, dem Koordinator der Linux Kernel Softwareentwicklung, in den Linux Kernel integriert. Dieser lobte das Projekt als deutlich fehlerunanfälliger als die Implementierungen von OpenVPN und IPsec.

Die Konfiguration der Verbindungspartner erfolgt mittels Textdateien. Diese enthalten eine der INI-Schreibweise ähnelnde Syntax. Es gibt insgesamt etwa 15 Konfigurationsparameter.

Das WireGuard Projekt stellt lediglich Software für den Verbindungsaufbau zur Verfügung. Die Verwaltung der Konfigurationsdateien übernimmt der Anwender oder Drittsoftware.

2. Anforderungsaufnahme

Die zu entwickelnde Software soll dem Anwender den Verwaltungsaufwand vollständig abnehmen. Die Erstellung, Änderung und Entfernung von beliebigen Konfigurationen soll abstrakt über die Software gesteuert werden. Es soll möglich sein, die Software direkt an das Verzeichnis für die Konfigurationsdateien anzuschließen, sodass sowohl bestehende als auch neue Konfigurationen eingelesen, verändert abgespeichert und sofort umgesetzt werden können. WireGuard ist ein sehr schlankes Protokoll, welches sich ideal in die Umgebung eines unixbasierten Systems integriert. Obwohl Implementierungen für Windows existieren, verwenden viele Anwender das Protokoll auf einem PC, Server oder VPN-Router mit unixbasiertem System. Die Administration von unixbasierten Systemen erfolgt klassischerweise über eine Textkonsole, entweder lokal oder entfernt via SSH. Graphische Benutzeroberflächen existieren zwar für viele Distributionen und sind im Desktopbereich weit verbreitet, aber auch hier gibt es jederzeit die Möglichkeit, mittels eines Terminal Emulators verschiedenste Änderungen schnell über eine Textkonsole zu erledigen. Die Bedienung der Applikation soll daher über eine Textkonsole möglich sein. Das Programm wird durch Administratoren bedient, von welchen Grundkenntnisse der Netzwerktechnik sowie des WireGuard Protokolls vorausgesetzt werden. Daher soll die Oberfläche funktional gehalten werden. Als Programmiersprache soll Python 3 verwendet werden. Auf nahezu allen unixähnlichen Betriebssystemen ist ein entsprechender Python Interpreter und der Paketmanager pip zur einfachen Verwaltung der Abhängigkeiten und Versionsverwaltung dieser verfügbar. Der Im- und Export von Konfigurationen soll über die Konsole oder Konfigurationsdateien möglich sein. Für die Mobilbetriebssysteme Android und iOS existiert eine WireGuard App, welche Konfigurationen in einem QR-Code kodiert einlesen kann. Die Übermittlung per QR-Code hat Vorteile bzgl. Sicherheit und Geschwindigkeit und soll daher unterstützt werden. Die Ausgabe des QR-Codes

soll mittels ASCII-Zeichen auf der Konsole stattfinden. Es sollen Eingabefehler erkannt und wenn möglich verhindert werden (beispielsweise IP-Adresskonflikte). Alle eingelesenen Konfigurationen sollen in einer einheitlichen Syntax abgespeichert werden. Die Verwaltung von IP-Adressen inkl. der Einteilung der IP-Subnetze soll durch die Anwendung übernommen werden. Außerdem soll die Software die Verwaltung der Schlüsselpaare übernehmen. Es soll möglich sein, per Massenauftrag die IP-Adressen aller Verbindungspartner zu ändern sowie die Schlüsselpaare einzelner Verbindungspartner neu auszustellen (z.B. im Falle einer Kompromittierung). Die Anwendung soll für die Bedienung durch einen Netzwerkadministrator vorgesehen werden, welcher über Kenntnisse des WireGuard-Protokolls bereits verfügt und in der Lage ist, auch Änderungen mit einem Texteditor durchzuführen. Das Programm soll diesen Anwender bei Anpassungen entlasten, aber kein notwendiges Fachwissen ersetzen.

Das Anforderungsdiagramm wurde aus Gründen der Formatierung in den Anhang verschoben, siehe Seite 17.

3. Anwendungsfälle

Auf Basis der Anforderungen wurden folgende Anwendungsfälle definiert:

1. /F10/ Neue Konfiguration erstellen

Der Benutzer möchte eine neue Konfiguration, bestehend aus einer Serverkonfiguration und mindestens einer Clientkonfiguration, erstellen. Dabei werden die für die Erstellung notwendigen Daten vom Benutzer durch einen Dialog abgefragt. Das Programm bestimmt so weit wie möglich selbst weitere Konfigurationsparameter wie das Schlüsselpaar. Nach Abschluss des Dialogs liegt sofort eine lauffähige Konfiguration vor. Der Benutzer kann die Konfiguration auf das Dateisystem schreiben lassen oder weitere Einstellungen vornehmen.

2. /F20/ Bestehende Konfiguration importieren

Der Benutzer möchte eine bestehende Konfiguration im WireGuard-Verzeichnis importieren, um diese zu bearbeiten und erneut abzuspeichern. Das Programm prüft, ob eine Konfiguration vorliegt und beginnt dann mit dem Import der Konfigurationsdateien. Eventuell auftretende Fehler werden dem Benutzer gemeldet, sodass dieser geeignet eingreifen kann. Alle Konfigurationsparameter sind später über das Programm einseh- und anpassbar.

3. /F30/ Konfiguration ändern

Der Benutzer möchte eine Änderung an einer Konfiguration für den Server oder einen Client vornehmen. Nachdem die betroffene Konfiguration ausgewählt ist, erhält der Benutzer eine Eingabemöglichkeit, um anzupassende Parameter mit dessen neuem Wert einzugeben. Das Programm

übernimmt die geänderten Parameter in der Datenstruktur und passt zusammenhängende Parameter ggf. selbstständig an.

4. /F40/ Entfernung einer Konfiguration

Der Benutzer möchte eine Client- oder die Serverkonfiguration entfernen. Das Programm entfernt die Konfiguration nach Angabe durch den Benutzer aus der Datenstruktur und aus dem Konfigurationsverzeichnis.

5. /F50/ Konfiguration einsehen

Der Benutzer möchte die vollständige Konfiguration im aktuellen Zustand ausgeben lassen. Das Programm gibt sämtliche Konfigurationsparameter aus. Für einen kurzen Überblick gibt es eine weniger detailreiche Ansicht.

6. /F60/ Anpassung der Netzwerkgröße

Der Benutzer möchte die Netzwerkgröße anpassen. Dazu wird vom Benutzer eine Angabe benötigt, wie viele Hosts das interne VPN Netzwerk höchstens adressieren soll. Nach der Eingabe übernimmt das Programm die Berechnung des Subnetzes und sämtliche Änderungen an den Konfigurationen.

7. /F70/ Änderung eines Schlüsselpaares

Der Benutzer möchte über das Programm das Schlüsselpaar eines Clients oder des Programms ändern. Nach der Angabe der anzupassenden Konfiguration generiert das Programm eigenständig ein neues Schlüsselpaar und hinterlegt es an allen notwendigen Orten in der Konfiguration.

8. /F80/ Export der Konfiguration

Der Benutzer möchte die Konfiguration aus dem Arbeitsspeicher in das Dateisystem schreiben. Dabei wird die bisherige Konfiguration gesichert und überschrieben

4. Bedienung des Programms

Das Programm besitzt eine interaktive Menüführung. Diese funktioniert ähnlich wie eine Shell mit einer Eingabe per Kommandozeile. Der Status des Programms wird vor jeder Eingabe ausgegeben. Der Status enthält eine eindeutige Benennung der gerade verwendeten Programmfunktion (z.B. `Hauptmenü` oder `Client anlegen`), ggf. Details zur angeforderten Eingabe (z.B. `(Bestätigung)`) und ggf. eine Angabe, welche Eingaben akzeptiert werden (z.B. `[j/n]`). Der Benutzer gibt jeweils eine Zeile ein und erhält vom Programm danach weitere Anweisungen, Informationen oder die Möglichkeit, weitere Informationen einzugeben. Vom Programm ausgegebene Meldungen sind grundsätzlich nach ihrer Relevanz farblich gekennzeichnet. Es gibt vier Stufen: Fehlermeldungen werden in roter Farbe ausgegeben. Sie weisen auf ein akutes Problem hin, welches den Programmablauf derzeit oder in Kürze stark beeinträchtigen kann (Beispiel: WireGuard-Verzeichnis ist nicht lesbar). Warnmeldungen werden

in gelber Farbe ausgegeben. Sie weisen auf einen Mangel hin, welcher den Programmablauf nicht unterbricht, aber zu Fehlern nach Programmende führen kann (Beispiel: es wurde ein IP-Adresskonflikt erkannt). Erfolgsmeldungen werden in grüner Farbe ausgegeben. Sie informieren den Benutzer im Modus mit detaillierten Ausgaben zum Programmablauf (`DEBUG`) über erfolgreich abgeschlossene Vorgänge (Beispiel: die Konfigurationsdateien im WireGuard-Verzeichnis wurden erfolgreich importiert). Informationsmeldungen werden in blauer Farbe ausgegeben. Sie weisen den Anwender auf benötigte Eingaben (Beispiel: neben der Angabe einer IP-Adresse wird auch die Angabe einer CIDR-Maske benötigt) oder den Programmablauf (Beispiel: nach dem Abbruch eines Eingabedialogs wird zurück in das Hauptmenü gewechselt) hin. Die ausgegebenen Meldungen sind im Modus mit detaillierten Ausgaben zum Programmablauf ausführlicher ausgelegt. Der Benutzer wird hier zusätzlich über Hintergrundprozesse informiert. Nur in diesem Modus werden Erfolgsmeldungen auf der Konsole ausgegeben. Andere Fehler-, Warn- und Informationsmeldungen erscheinen häufiger als im normalen Modus.

5. Datenmodell

Zur besseren Verständlichkeit des Datenmodells ist nachfolgend eine Beispielkonfiguration abgebildet:

Dateiinhalt der Serverkonfiguration `wg0.conf`:

```
[Interface]
# Name = Serverkonfiguration
Address = 10.10.10.1/24
ListenPort = 12345
PrivateKey = cHZQta30/Z4Zijb/nCeYxkkR/u8ep1vwGuy5xYT2708=

[Peer]
# Name = Mein iPhone
AllowedIPs = 10.10.10.2/32
PublicKey = Ew8gAiOdwOIoltpCwDYrLFzMJb/Jl3oB7GO1k4JqgzY=
```

Dateiinhalt der Clientkonfiguration `iphone.conf`:

```
[Interface]
# Name = Mein iPhone
Address = 10.10.10.2
PrivateKey = kC/Wd4Ws65DX4FXCsZBkHdNAbKVpe86JJbIA2LK0slY=

[Peer]
# Name = Serverkonfiguration
AllowedIPs = 10.10.10.1
```

```
Endpoint = mein-vpn-server:12345
PublicKey = 0dAwIB3Ji96GYdlesA+iCNxhB7NElkFf7DZ4GWyaEFI=
```

Für die Persistierung der Konfigurationsparameter im Arbeitsspeicher wurde eine Datenstruktur auf Basis von zwei Klassen entwickelt.

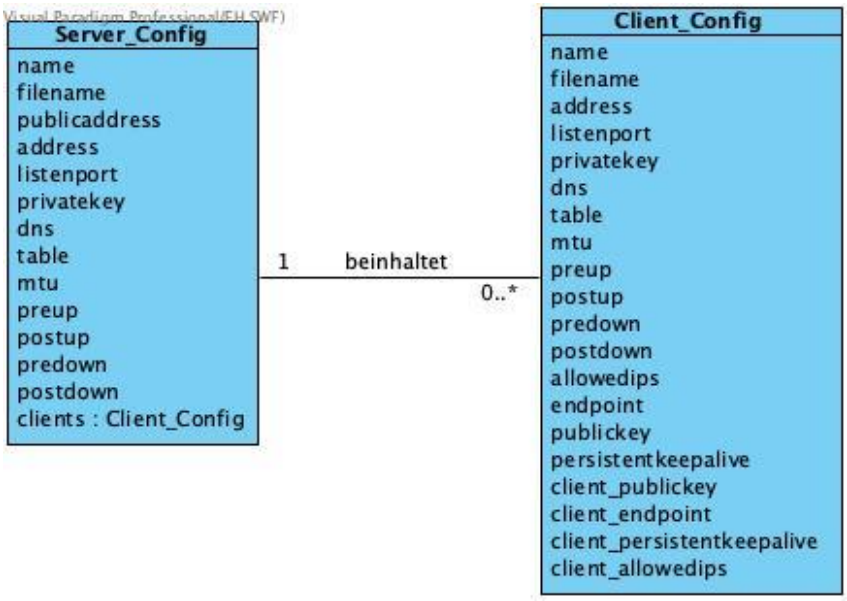

Abbildung 1: Klassendiagramm

Wird eine neue Konfiguration angelegt oder erstellt, wird ein Objekt der Klasse `Server_Config` erstellt. Dieses beinhaltet die Daten der Sektion `Interface` der Serverkonfiguration, den zusätzlichen Parameter `name` für die Bezeichnung, den Parameter `filename` für den Dateinamen sowie eine Liste `clients`, welche Objekte der Klasse `Client_Config` aufnehmen kann. Die Klasse `Server_Config` enthält im Gegensatz zur Klasse `Client_Config` keine Parameter der Sektion `Peer`. Dies hat den Grund, dass eine Serverkonfiguration mehrere Sektionen `Peer` enthält, welche jeweils einem Client zugeordnet werden. Es liegt daher im Hinblick auf die Datenstruktur näher, die Sektionen `Peer` der Serverkonfiguration gemeinsam mit den weiteren Daten des Clients zu speichern. Die Konfigurationsparameter aus der Sektion `Interface` der Serverkonfiguration sind einmalig hinterlegt.

Die Klasse `Client_Config` enthält, wie die Serverkonfiguration, alle Parameter einer Sektion `Interface`. WireGuard unterscheidet intern nicht zwischen Servern und Clients. Beide haben denselben Umfang der Konfigurationsparameter. Um eine Verbindung aufzubauen, muss diese von einem Verbindungspartner initiiert werden. Dies wird üblicherweise vom Client übernommen. Der Server antwortet auf Verbindungsanfragen und schließt den Handshake ab. Zusätzlich enthält die Klasse `Client_Config` die zwei Parameter `name` und `filename`, welche denselben Zweck erfüllen, wie die gleichnamigen Parameter in der Klasse `Server_Config`. Zusätzlich enthält die Klasse der

Clientkonfigurationen, wie oben erwähnt, die Daten aus der dem Client zugeordneten Sektion `Peer` der Serverkonfiguration. Diese schließt u. A. den öffentlichen Schlüssel mit ein. Der öffentliche Schlüssel ist Teil eines Schlüsselpaars für eine asymmetrische Verschlüsselung. Beim Importieren einer existierenden Konfiguration werden Server- und Clientkonfiguration getrennt eingelesen. Die Zuordnung der Daten erfolgt anhand des privaten Schlüssels des Clients, mit welchem der öffentliche Schlüssel berechnet werden kann, welcher in einer Sektion `Peer` der Serverkonfiguration hinterlegt ist.

Erläuterung der WireGuard-Konfigurationsparameter der Sektion `Interface`:

`address` enthält die IP-Adresse des Verbindungspartners (Server oder Client) im VPN-Netzwerk.

`listenport` enthält die Portnummer, auf welcher öffentlich gelauscht wird. Die Definition einer Bindungsadresse ist von den Entwicklern bisher nicht vorgesehen. Der WireGuard-Dienst lauscht daher auf allen Netzwerkschnittstellen auf dem angegebenen Port.

`privatekey` enthält den privaten Schlüssel des Verbindungspartners. Der private Schlüssel wird bei der asymmetrischen Verschlüsselung zur Entschlüsselung verwendet.

`dns` definiert einen zu verwendenden Namensserver bei aktiver VPN Verbindung.

`table` gibt an, ob eine spezielle Routingtabelle auf dem System verwendet werden soll.

`mtu` gibt die maximale Paketgröße an. Mit diesem Wert kann die Übertragungsgeschwindigkeit optimiert oder Fehler bei bestimmten Verbindungstypen (z.B. DSL oder Mobilfunk) vermieden werden.

`preup`, `predown`, `postup` und `postdown` geben an, welche zusätzlichen Programme vor oder nach Verbindungsauf- und abbau ausgeführt werden sollen.

`allowedips` wirkt sich auf die Routingtabelle des Verbindungspartners aus. Mit dem Parameter kann beeinflusst werden, welche Geräte über die VPN Verbindung erreichbar sein sollen. Weiterhin kann eine Standardroute (z.B. für einen abgesicherten Zugang zum Internet in einem unsicheren Netzwerk) gesetzt werden.

`endpoint` gibt an, unter welcher öffentlichen Adresse der Verbindungspartner erreichbar ist. Dieser Parameter wird im Normalfall für die Initiierung der Verbindung durch den VPN-Client benötigt.

`publickey` gibt, anders als der Parameter privatekey, nicht den lokalen öffentlichen Schlüssel an, sondern den öffentlichen Schlüssel des anderen Verbindungspartners. Der öffentliche Schlüssel wird bei asymmetrischer Verschlüsselung zur Verschlüsselung verwendet.

`persistentkeepalive` gibt an, in welchem Rhythmus ein Lebenssignal an den Verbindungspartner gesendet werden soll. Das WireGuard Protokoll ist zustandslos. Werden keine Daten gesendet, findet zwischen den beiden Partnern überhaupt kein Datentransfer statt. Befindet sich ein Verbindungspartner hinter einem NAT-Router, sollte ein Wert für `persistentkeepalive` gesetzt werden, um den Eintrag in der NAT-Tabelle aktiv zu halten.

6. Algorithmen

Beschreibung der Funktionsweise des Imports von vorhandenen Konfigurationsdateien:
Alle Konfigurationsdateien, welche importiert werden sollen, werden vom Parser verarbeitet und abgespeichert. Viel Programmcode des Parsers lässt sich in anderen Programmkomponenten in leicht veränderter Form wiederfinden. Der Importprozess besteht aus Gründen der Lesbarkeit und Nachvollziehbarkeit aus mehreren Funktionen. Die Funktion `import_configurations()` bildet eine Liste mit existierenden Konfigurationsdateien und erkennt, welche die Serverkonfiguration ist. Im ersten Schritt werden sämtliche Clientkonfigurationen eingelesen. Danach folgt die Serverkonfiguration. Die Reihenfolge ist bewusst gewählt, da die Clientkonfigurationen die privaten Schlüssel enthalten, mit welchen die dazugehörigen öffentlichen Schlüssel berechnet werden können. Die Sektionen `Peer` der Serverkonfigurationen enthalten bereits die öffentlichen Schlüssel der Clients. So können Clientkonfigurationen und die Sektionen `Peer` der Serverkonfiguration während des Imports zusammengeführt werden. Die Funktion `parse_and_import(peer)` übernimmt dabei das Auslesen einer Konfigurationsdatei eines Clients oder Servers, dessen Dateipfad im Parameter `peer.filename` des `Client_Config` oder `Server_Config` Objekts zu finden ist. Nach den Parameterprüfungen beginnt der eigentliche Prozess des Parsens:

Zuerst wird anhand der Klasse von `peer` erkannt, ob es sich um eine Server- oder Clientkonfiguration handelt. Der Parameter `is_server` wird entsprechend auf den Wert True oder False gesetzt.

Im Anschluss wird die Datei zeilenweise eingelesen. Enthält eine Zeile nur oder keine Leerzeichen sowie keine weiteren Zeichen, wird mit der nächsten Zeile fortgefahren. Laut INI-Syntax dürfen beliebig viele Leerzeilen an allen Stellen in der Konfigurationsdatei vorkommen. Enthält eine Zeile eine Sektion (erkennbar an den durch eckige Klammern eingeschlossenen Buchstaben), erfolgt direkt eine weitere Fallunterscheidung. Dort wird geprüft, ob es sich um eine Sektion `Peer` handelt und das übergebene `peer`-Objekt von der Klasse `Server_Config` stammt. Dies stellt einen Sonderfall dar: die Parameter der beiden Sektionen `Interface` und `Peer` sind bekannt. Jeder Parameter kann einer Sektion

zugeordnet werden. Daher ist eine Beachtung der Sektionen durch den Parser häufig nicht notwendig, die Zeilen `[Interface]` und `[Peer]` stellen lediglich eine optische Trennung für den Benutzer dar und können vom Parser ignoriert werden. Eine Ausnahme bildet der erwähnte Sonderfall: eine Sektion `Peer` einer Serverkonfiguration kommt mehrfach vor, sobald mehr als ein Client angebunden ist. Daher muss eine Sektion `Peer` einer Serverkonfiguration erst vollständig erfasst und gespeichert werden, bevor eine weitere Sektion `Peer` einer Serverkonfiguration den Parser durchlaufen kann.

Für die Erfassung aller Parameter einer Sektion `Peer` wurde zu Anfang der Funktion ein Objekt der Klasse `Peer` erzeugt. Dieses enthält Objektparameter für alle möglichen Parameter der Sektion `Peer`. Das Peer-Objekt wird beim Start einer Sektion `Peer` eines Servers sowie zum Funktionsende von `parse_and_import()` (d.h. nach Abschluss der Aufnahme einer Datei vom Dateisystem) durch die Funktion `assign_peer_to_client(client_data, peer)` „geleert“ und die enthaltenen Parameter werden in die Objektparameter `client_`[…] des `Client_Config` Objekts geschrieben. Im Anschluss wird mit dem Parsing fortgefahren.

Kommentarspalten werden erkannt. Der erste Kommentar wird als Name hinterlegt. Dabei wird ein ggf. voranstehendes `Name =` abgeschnitten. Folgende Kommentare werden ignoriert.

Nun folgt die Erkennung eines Name-Wert Paares. Gemäß INI-Syntax dürfen Name, Zuweisungsoperator `=` und Wert beliebig viele Leerzeichen trennen. Dies wird durch einen entsprechenden regulären Ausdruck unterstützt. Name und Wert werden mit der Methode `strip()` von voranstehenden und nachfolgenden Leerzeichen bereinigt. Danach erfolgt ein Abgleich mit der Liste der bekannten Konfigurationsparameter. Ist der Parameter Teil der Sektion `Peer` und handelt es sich um eine Serverkonfiguration, tritt der oben genannte Sonderfall erneut ein und der Konfigurationsparameter wird für die spätere Zuweisung durch `assign_peer_to_client()` zurückgestellt. Handelt es sich um einen bekannten Konfigurationsparameter, wobei der Sonderfall nicht eintritt, wird der Wert des Parameters im entsprechenden gleichlautenden Objektparameter von `peer` hinterlegt. Schließlich erfolgt ein Abgleich mit der Liste der notwendigen Parameter, welche in jeder Konfiguration vorkommen müssen. Ist der Parameter enthalten, wird dieser von der Liste entfernt. Nach Abschluss des Einlesevorgangs einer Datei und damit einer Konfiguration wird auf übrige notwendige Parameter geprüft, welche in der Konfiguration nicht enthalten waren. Falls solche Parameter existieren, wird eine entsprechende Warnmeldung mit allen fehlenden Parametern ausgegeben.

Schließlich erfolgen noch zwei Prüfungen in der Funktion `import_configurations()`, welche die Funktion `parse_and_import()` zur Verarbeitung jeder entdeckten Konfigurationsdatei

aufgerufen hatte: der Konfigurationsparameter `address` der Serverkonfiguration wird in ein `IPv4Interface` Objekt umgewandelt, welches neben einer IP-Adresse auch eine Netzmaske enthält. So kann nach dem Import sämtlicher Konfigurationen geprüft werden, ob es sich um einen von der IANA für die private Verwendung vorgesehenen Adressbereich handelt und ob die IP-Adressen sämtlicher Clientkonfigurationen im Subnetz der Serverkonfiguration enthalten sind. In beiden Fällen werden entsprechende Warnmeldungen an den Benutzer gesendet, falls Probleme festgestellt werden.

7. Implementierung

Bei der Implementierung wurde Wert auf eine hohe Wartungsfreundlichkeit des Codes gelegt. Daher sind die verschiedenen Klassen und Programmfunktionen in eigene Dateien aufgeteilt worden. Jede Datei besitzt `import` Statements, welche in Standardbibliotheken, Bibliotheken von Drittanbietern und eigenen Modulen unterteilt wurden. Die Formatierung entspricht dem weitverbreiteten Standard PEP 8 (Python Enhancement Proposal, zu deutsch etwa Vorschlag für Python-Erweiterung). Der Code wurde mit mehreren Werkzeugen für statische Code Analyse (darunter pylint sowie mit dem integrierten Werkzeug der für die Entwicklung verwendeten IDE Jetbrains PyCharm Professional) auf syntaktische und semantische Fehler geprüft.

8. Testfälle

Um die Funktion der Software zu validieren, wurden diverse Testfälle definiert.

1. Import einer Konfiguration

Auf dem Dateisystem befinden sich im WireGuard-Verzeichnis (Dateipfad wird über die statische Variable `WG_DIR` definiert) einige Dateien für die Server- und Clientkonfiguration einer bestehenden Konfiguration. Die Dateien werden über den Menüpunkt `1` in den Arbeitsspeicher importiert. Das Programm muss bei einer bestehenden Konfiguration im Arbeitsspeicher den Benutzer warnen und eine Bestätigung einholen, bevor die Konfiguration überschrieben wird. Die Dateien werden nach der Bestätigung durch den Benutzer importiert und müssen im Anschluss über den Menüpunkt `2` vollständig ausgegeben werden können.

2. Hinzufügen eines Clients

Der Benutzer möchte einen neuen Client zur Konfiguration hinzufügen und wählt den Menüpunkt `3`. Besteht bislang keine Konfiguration im Arbeitsspeicher, muss das Programm den Benutzer informieren und eine Bestätigung anfordern. Nach der Bestätigung durch den Benutzer muss ein Dialog angezeigt werden, in welchem eine Bezeichnung, die IP-Adresse sowie weitere Konfigurationsparameter abgefragt werden. Existiert keine Serverkonfiguration, müssen dieselben Daten für die Anlage einer neuen Serverkonfiguration abgefragt werden. Das Programm ergänzt die eingegebenen Daten um einen privaten Schlüssel sowie die Konfiguration der Parameter `allowedIPs` und `endpoint`.

3. Entfernen von Clients

Wird die Entfernung eines Clients über den Menüpunkt `4` angefordert, muss ein Dialog angezeigt werden, welcher die ID des Clients aus der Auflistung von Menüpunkt `2` abfragt. Sind keine Konfigurationen im Arbeitsspeicher hinterlegt, muss der Benutzer gewarnt und zurück zum Hauptmenü geleitet werden. Nach der Eingabe einer ID muss zunächst geprüft werden, ob eine Konfiguration mit dieser Nummer vorliegt. Falls ja, muss die Konfiguration entfernt werden. Falls nicht, muss der Benutzer gewarnt werden und zurück zum Hauptmenü geleitet werden. Wurde die ID `0` eingegeben, muss die Serverkonfiguration nach einer Bestätigung durch den Benutzer inklusive aller Clientkonfigurationen entfernt werden. Der Status des Programms muss im Anschluss dem entsprechen, in welchem es sich nach einem Start befindet. Insbesondere müssen erneut Warnungen bzgl. nicht vorhandener Konfiguration ausgegeben werden, wenn der Benutzer beliebige Menüpunkte außer `1` und `3` aufruft.

4. Anzeige von validen QR-Codes

Wurde eine Konfiguration eingegeben, kann der Benutzer mit dem Menüpunkt `8` einen QR-Code mit einer beliebigen Konfiguration ausgeben lassen. Dazu wird zunächst die ID der gewünschten Konfiguration abgefragt. Nach der Eingabe muss geprüft werden, ob die Konfiguration vorliegt. Sind keine Konfigurationen im Arbeitsspeicher hinterlegt oder liegt keine Konfiguration mit der ID vor, muss der Benutzer gewarnt und zurück zum Hauptmenü geleitet werden. Falls eine Konfiguration mit der ID vorliegt, muss ein QR-Code auf der Konsole angezeigt werden. Dieser muss mit der WireGuard-App für Android und iOS eingelesen werden können. Das Programm ist lediglich für die Ausgabe der Zeichen inklusive der Abtrennung von weiteren Inhalten auf der Konsole verantwortlich, es ist durch den Benutzer für eine passende Auflösung und kontrastreiche Farbauswahl (idealerweise schwarz-weiß) zu sorgen.

9. Ausführung des Programms

Das Programm ist für die Ausführung auf unixoiden Systemen außer macOS vorgesehen. Das schließt u. A. Linux Derivate sowie Versionen von des Unix-Derivats BSD mit ein. Voraussetzungen an das Hostsystem sind vorhandene Installationen von Python 3 und WireGuard, sowie die Möglichkeit, Pakete aus dem Python Package Index zu installieren. Idealerweise übernimmt dies ein Paketmanager wie pip. Alternativ steht ein Dockerfile zur Verfügung, mit welchem ein Image auf Basis des Linux Derivats Debian erstellt werden kann. In diesem Fall entfallen die Anforderungen an die Python-Umgebung des Hostsystems. Bei der Anwendung auf nicht unixoiden Systemen wie Windows können trotz einer Ausführung in einem Docker-Container unvorhergesehene Fehler auftreten. Da die Software ausschließlich unter Linux getestet wurde, wird die Anwendung unter Windows nicht unterstützt.

Wird die Software nativ, also nicht in einem Docker-Container, betrieben, wird die Verwendung eines Python venv (virtual environment) empfohlen. Die Verwendung hat mehrere Vorteile: es können mehrere voneinander unabhängige Python-Installationen in verschiedenen Python Versionen und mit verschiedenen Paketen auf demselben Host installiert werden. Außerdem wird die Sicherheit erhöht, wenn die Applikation mit administrativen Berechtigungen (unter Linux als Benutzer root, bzw. mit dem Befehl `sudo`) in einer abgeschlossenen Umgebung betrieben wird. Die Ausführung mit administrativen Berechtigungen ist in den meisten Fällen notwendig, wenn auf das Verzeichnis von WireGuard zugegriffen werden muss, da dieses für andere Benutzer standardmäßig nicht beschreibbar ist.

10. Anwenderdokumentation

1. Installation

1. Nativ

Zu installierende Abhängigkeiten: WireGuard (https://www.wireguard.com/install/), Python 3, pip (optional), Python venv (optional)

Unter Ubuntu 22.04 werden alle benötigten Abhängigkeiten mit dem Befehl `sudo apt install pip wireguard python3.10-venv` installiert.

Nur bei Verwendung des Python venv: es ist notwendig, eine virtuelle Umgebung zu erstellen. Dies erledigt der Befehl `python3 -m venv wireguard-mgmt-env`. Der Name kann frei gewählt werden. Es wird ein Ordner mit dem Namen des venv im Verzeichnis angelegt, in welchem sich der Benutzer derzeit befindet. Der Befehl source `wireguard-mgmt-env/bin/activate` richtet den Zugriff auf das venv ein. Von nun an befindet sich der Benutzer in der virtuellen Umgebung und kann beispielsweise mit pip notwendige Abhängigkeiten des Programms installieren.

Mit pip werden die benötigten Abhängigkeiten colorama, cryptography, ipaddress und qrcode installiert. Dies kann automatisiert unter Ubuntu 22.04 mit dem Befehl `sudo pip install --no-cache-dir -r requirements.txt` erledigt werden. Der Benutzer muss sich dazu im dem Ordner befinden, in welchem sich die Datei `requirements.txt` befindet. Schließlich sollten die Inhalte der Datei `constants.py` geprüft und ggf. auf das eigene System angepasst werden. Hier ist insbesondere der Parameter `WG_DIR` zu beachten.

Im Anschluss kann das Programm mittels `python3 -m main` ausgeführt werden. Der Benutzer muss sich im dem Verzeichnis befinden, in welchem die Datei `main.py` vorhanden ist. Das Hauptmenü wird angezeigt.

2. Docker

Zu installierende Abhängigkeiten: WireGuard (https://www.wireguard.com/install/), Docker (https://docs.docker.com/engine/install/)

Zuerst muss in das Verzeichnis gewechselt werden, in welchem sich die Datei `Dockerfile` befindet. Es wird ein Docker Image `wg-config-mgmt` erstellt: `docker build -t wg-config-mgmt ./`

Das Image kann in einem Container ausgeführt werden. Der Befehl `docker run -it --rm -v /etc/wireguard:/res wg-config-mgmt` legt einen Container im Hintergrund an und verknüpft das standardmäßig verwendete WireGuard-Verzeichnis auf dem Hostsystem. Falls dies vom Standard abweicht, muss der Pfad `/etc/wireguard` hinter `-v` entsprechend angepasst werden. Das Hauptmenü wird angezeigt. Wird das Programm beendet, beendet sich der Docker Container ebenfalls. Durch die Angabe des `docker run` Parameters `--rm` wird dieser nach dem Programmende automatisch vom System entfernt.

2. Ausführung

1. Nativ

Das Programm kann mit dem Befehl `python3 -m main` ausgeführt werden. Wird das Python venv verwendet, muss vorher mit dem Befehl `source wireguard-mgmt-env/bin/activate` in dieses gewechselt werden.

2. Docker

Sobald das Image erstellt wurde, kann eine neue Programminstanz mit dem Befehl `docker run -it --rm -v /etc/wireguard:/res wg-config-mgmt` erstellt werden.

3. Bedienung

Sobald das Hauptmenü angezeigt wird, kann mit den Anpassungen begonnen werden. Zuerst sollte eine Konfiguration, falls vorhanden, importiert (Menüpunkt `1`) oder neu erstellt (Menüpunkt `3`) werden. Im Anschluss können Konfigurationen ergänzt, angepasst oder verändert werden. Einige Funktionen erfordern die Eingabe einer ID. Eine Übersicht über die IDs der vorhandenen Clients kann über den Menüpunkt `2` abgerufen werden. Sofern vorhanden, können Vorgaben in eckigen Klammern mit der Enter-Taste übernommen werden.

Quellenverzeichnis

Sweeting, Nick (2022): Some Unofficial WireGuard Documentation, [online] https://docs.sweeting.me/s/wireguard (abgerufen am 20.05.2022)

van Rossum, Guido; Warsaw, Barry; Coghlan, Nick (2001): Style Guide for Python Code, [online] https://peps.python.org/pep-0008/ (abgerufen am 20.05.2022)

Abbildungsverzeichnis

Abbildung 1: Klassendiagramm 8

Anhang

Anforderungsdiagramm