

SZAKDOLGOZAT



MISKOLCI EGYETEM

Linux rendszerek használatát segítő eszközök fejlesztése Lua-ban

Készítette:

Kovács László

Programtervező informatikus

Témavezető:

Dr. Glavosits Tamás

MISKOLC, 2023

MISKOLCI EGYETEM

Gépészmérnöki és Informatikai Kar

Alkalmazott Matematikai Intézeti Tanszék

Szám:

SZAKDOLGOZAT FELADAT

Kovács László (CIJ404) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Linux rendszerek kezelése, szerverek/tűzfal menedzselése Linux rendszereken

A szakdolgozat címe: Linux rendszerek használatát segítő eszközök fejlesztése Lua-ban

A feladat részletezése:

A Lua egy hatékony, elterjedt és egyszerűen használható programozási nyelv. A dolgozat azt mutatja be, hogy ezen programozási nyelv segítségével hogyan készíthetők olyan eszközök, amelyek segítik a GNU/Linux rendszerek kezelését.

Ilyen jellemző feladatok például a fájlok kezelése, a szöveges fájlok feldolgozása, rendszergazdai feladatok, terminál alapú felhasználói felületek kezelése.

A nyelv egyszerűségének köszönhetően segítséget próbál jelenteni a kezdő felhasználók számára is a Linux alapú rendszerekkel való ismerkedés során.

A dolgozat összeveti az elterjedt Lua változatokat, implementációkat. Specifikálja a felhasználók számára szükséges funkciókat, bemutatja az elterjedt elérhető megoldásokat, majd azoknak a saját implementációját.

Témavezető: Dr. Glavosits Tamás (egyetemi adjunktus)

A feladat kiadásának ideje:

.....
szakfelelős

EREDETISÉGI NYILATKOZAT

Alulírott **Kovács László**; Neptun-kód: **CIJ404** a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírással igazolom, hogy *Linux rendszerek használatát segítő eszközök fejlesztése Lua-ban* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szó szerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, év hó nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

.....

dátum

.....

témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):

konzulens (dátum, aláírás):

.....

.....

.....

.....

.....

.....

3. A szakdolgozat beadható:

.....

dátum

.....

témavezető(k)

4. A szakdolgozat szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

.....

dátum

.....

témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve:

.....

dátum

.....

szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:

a bíráló javaslata:

a szakdolgozat végleges eredménye:

Miskolc,

.....

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. Bevezetés	1
2. Konceptió	3
2.1. Felhasznált programnyelv	3
2.2. Lua sajátosság: metatáblák	5
2.3. Lua sajátosság: osztályok, öröklődések implementálása metatáblák segítségével	7
2.3.1. Táblaműveletek, self kulcsszó	7
2.3.2. Konstruktor, Objektum példányosítás	9
2.3.3. Destruktor	9
2.3.4. Öröklődés	10
2.3.5. Láthatósági szabályok osztályokon belül	11
2.4. Programtól elvárt működés	12
2.4.1. Mi az a tls-crypt?	13
2.4.2. Mi az a chroot?	13
2.5. Hasonló alkalmazások	14
2.5.1. Apache2 és nginx	14
2.5.2. Tűzfal	15
2.5.3. OpenVPN	17
3. Tervezés	19
3.1. general, apt_packages, utils modulok felépítése, feladata	19
3.2. linux modul felépítése, feladatai	20
3.3. OpenVPN modulok felépítései, feladatai	21
3.4. nginx modulok felépítései, feladatai	23
3.5. apache modulok felépítései, feladatai	24
3.6. certbot modul felépítése, feladata	26
3.7. snapd modul felépítése, feladata	26
3.8. iptables modul felépítése, feladata	27
4. Megvalósítás	28
4.1. linux modul érdekességei	28
4.1.1. Processz exit codek	30
4.2. Implementációs nehézség: háttérben futó processz eredményeire való vá- rás certbot modulban	31
4.3. Konfigurációs fájlok módosításának implementációja	33
4.3.1. OpenVPN config parser-writer implementáció felépítése, használata	33
4.3.2. Apache, nginx config parser-writer implementáció felépítése és használata	34

4.4. Modulok használata Lua-ban	37
5. Tesztelés	38
5.1. OpenVPN	39
5.1.1. Konfigurálás, telepítés előtt	39
5.1.2. Konfigurálás, telepítés után	39
5.2. Webszerverek	40
5.3. iptables	41
5.3.1. Feltelepítés előtt	41
5.3.2. Feltelepítés után	41
5.3.3. Minden forgalom átirányítása OpenVPN szerveren keresztül . .	42
6. Összefoglalás	43
Irodalomjegyzék	45

1. fejezet

Bevezetés

A dolgozat célja, hogy egy olyan Lua nyelvben általam készített eszközt mutasson be, amely könnyebbé teszi GNU/Linux disztribúciókon a szerverüzemeltetést.

Habár ez az eszköz kifejezetten Linuxra az Ubuntu és Debian disztribúción való felhasználásra készült, a Lua programnyelv miatt könnyedén bővíthető több disztribúció kezelésére is. A Lua programnyelv cross-platform, így futtatható akár Linuxon és Windowson is, néhány részegységet akár Windows-kompatibilissé is lehetne tenni.

A program által támogatott kiszolgálói szoftverek közé tartozik az OpenVPN, az Apache2, továbbá az nginx. Támogatja továbbá az *iptables* csomagszűrő programot is.

Az OpenVPN napjaink egyik legmeghatározóbb VPN-szerver cross-platform implementációja. Kétféle típusa van: fizetős/freeware (Access Server) és ingyenes (Community Server) verzió. Az eszközünk a Community Server típust fogja használni.

Az Access Server előnye a Community Server-rel szemben a könnyű kezelhetőség, WebAdmin alapú felülettel való kezelés. A Community Server általában haladók számára ajánlott, mivel ott a felhasználónak önmagának kell bekonfigurálnia a szervert.

A legtöbb VPN-szolgáltató OpenVPN szervert használ szolgáltatásaihoz (például NordVPN, ExpressVPN, Avast SecureLine VPN). [29]

Érdemes manapság VPN-t használni több okból kifolyólag is [30], például:

- lehallgatás elleni védelmet biztosít (publikus hálózat esetén ez nagy előny)
- nagyobb biztonságot adhat a távoli munkához, például ha rácsatlakozunk a munkahelyi hálózatra
- IP címet tudunk változtatni vele (akár más ország IP címét is felvehetjük), ezáltal bizonyos kedvezményekben részesülhetünk

Az Apache2 és az nginx napjaink kettő legpopulárisabb webservere. [31]

Mind a két szerver platformfüggetlen, nyílt forráskódú webserverek implementáció. Alkalmazásuk teljesen ingyenes, konfigurációjuk könnyen értelmezhető. Főbb különbség köztük a processzek és szálak használata.

Az Apache2 koncepciója legfőképp processzek és szálak használatára épül, akár új szál is létrehozható egy kérés feldolgozása érdekében.

Az nginx ezzel szemben esemény (aszinkron) alapú processz, akár több kérést is feldolgozhat egy szálon.

A szerverek üzemeltetése mellett természetesen fontos a többrétegű *védelem* is, amelynek egyik alapeleme a tűzfal. A tűzfalat a Linux kernel *netfilter* csomagja tartalmazza, az *iptables/ip6tables* csomagyszűrő program segítségével konfigurálható. A dolgot ennek a programnak a kezelését is megkönnyíti.

Az Interneten sok segítség található ezen alkalmazások telepítéséről, alkalmazásáról, azonban úgy gondolom, hogy a program így is segítséget tud nyújtani.

Könnyebb, gyorsabb a szerverprogramok telepítése, konfigurálása, és nem utolsósorban a Linux kezelésében nem annyira jártas felhasználók saját maguk által létrehozott példák tanulmányozásán keresztül tudják tanulni a rendszer használatát, a szerverprogramok konfigurálását.

2. fejezet

Koncepció

2.1. Felhasznált programnyelv

A program elkészítéséhez a felhasznált programnyelv a Lua, tesztelésre és az implementációra az 5.3.5 verzió lett használva.

A **Lua** egy könnyű, magas szintű programozási nyelv, amelyet főképp a könnyű beágyazhatóság jegyében fejlesztettek ki. A legelső verzióját 1993-ban adták ki. Cross-platform, mivel implementációja Ansi C-ben íródott. Saját virtuális géppel és bytecode formátummal rendelkezik.



2.1. ábra. A Lua logója

A scriptek futtatás előtt bytecode-ra fordítódnak át, majd úgy kerülnek átadásra az interpreternek. Mi magunk is lefordíthatjuk a Lua forráskódunkat a luac bináris segítségével. Ezt akkor szokták megtenni, ha fel akarják gyorsítani a script futását, vagy ha nem szeretnék, hogy idegenek ismerjék a forráskódot.

Támogat többféle programozási paradigmát is, azonban ezek nincsenek előre implementálva, viszont a nyelv lehetőséget ad arra, hogy implementáljuk őket. Például öröklődést, osztályokat metatáblák használatával tudunk implementálni. [21]

Széleskörű C API található felhasználásához, azonban nem csak erre a nyelvre korlátozódik az API használatának lehetősége, többféle wrapper is készült a C API-hoz, a legtöbb azonban a C++ nyelvhez készült (például sol).

Az interpreterje önmagában eléggé jó teljesítménnyel rendelkezik, jónéhány benchmark kimutatása szerint a Lua élvonalon jár teljesítmény szempontjából az interpretált scriptnyelvek között. Nem csak végletekig-csiszolt "benchmarkra szánt" programokban jó a teljesítménye, hanem a való életben való felhasználása közben is. Nagyobb sebességért a LuaJIT branchet is lehet használni.

A **LuaJIT** a Lua olyan verziója, amely Just-In-Time compilert tartalmaz, alapjainban is gyorsabb, mint a Lua. A LuaJIT bytecode formátuma teljesen más, mint a sima Lua-é, gyorsabb az instrukciók dekódolása. A virtuális gépe is közvetlenül Assemblyben íródott. A LuaJIT implementáció szinten nem kötődik hivatalosan a Lua programnyelvhez, attól független, más emberek fejlesztik. [17]



2.2. ábra. A LuaJIT logója

A Lua programnyelvhez csomagkezelő is készült, amelyet LuaRocksnak hívnak, funkcionalitása hasonló a NodeJS **npm** csomagkezelőjéhez.

Előszeretettel használják a játékfejlesztők is, több neves játékban is előfordul, például World of Warcraft, PayDay 2, Saints Row széria, Crysis, Roblox, FiveM, MTA:SA. [14]

2.2. Lua sajátosság: metatáblák

Az előző szekcióban kifejtettem, hogy néhány programozási paradigma nincs előre implementálva, ezeket nekünk kell implementálnunk. Öröklődést, osztályokat metatáblák segítségével tudunk implementálni. A nyelvben minden táblának és userdata-nak lehet metatáblája. Az userdata típus adatok tárolására szolgál, a C API készíti, és az eltárolt adatok memóriacíme alapján működik (ez lehet akár az alap Lua IO library által megnyitott fájlok handleja, amelyek garbage collect esetén bezárják a file handleket a `__gc` metametódus segítségével); vagy például játékok esetében akár játékosadat, járműadat, satöbbi).

A **metatábla** egy olyan szokványos Lua tábla, amely meghatározza bizonyos műveletek viselkedését. Értelemszerűen egy metatáblát több táblára is fel lehet használni. A metatáblák legközelebbi rokonjai véleményem szerint a JavaScript nyelv Proxy-jai, azonban hasonlítanak a C++ nyelvben használt operator overloadingokra is.

A metatáblákban találhatóak meg a **metametódusok**, amelyek meghívódhatnak bizonyos műveletekkor. Nevük két alsóvonallal kezdődnek, majd a metametódus neve követi. Például: `__add`.

A metatáblák beállíthatóak a `setmetatable` funkcióval, továbbá lekérhetőek a `getmetatable` funkcióval. A metatáblákon belüli értéklekérésre célszerű a `rawget` funkciót használni, ez kikerüli a további metatáblákat az értéklekéréskor.

Alapvetőleg csak tábláknak és userdata-knak van metatáblájuk. Minden más típusú adat saját **-a típusának megfelelő-** metatáblát használ (például van külön számoknak, szövegeknek, satöbbi). Alapvetőleg egy értéknek nincs metatáblája, viszont például a beépített string library hozzáad a szöveg típusú értékeknek saját metatáblát, ezzel használható például a `string.sub`, `string.gsub` funkció is.

Néhány érdekesebb, akár a program által is felhasznált metametódusok:

- **`__add`**: hozzáadás művelete. Ha bármelyik operandus nem szám (és nem is szöveg, ami számot tartalmaz), akkor a Lua a metametódusokhoz nyúl, és ekkor próbálja ezt a metódust meghívni. Ha egyik operandusnál sem találja meg a metametódust, akkor hibát dob. A metametódus paraméterei a két operandus, és a funkciómeghívás végeredménye lesz az összeadás eredménye.
- további matematikai műveletek (**`__sub`**, **`__mul`**, **`__div`**, **`__mod`**, **`__pow`**, **`__unm`**, **`__idiv`**): hasonlóan működnek az **`__add`** metódushoz. Bitszinten is elérhetőek a bitmetódusok felülírása (például `or`, `xor`, `and`, `not`, satöbbi).
- **`__concat`**: az összefűzés (Luaban: `..`) művelete. Hasonló az **`__add`** metametódushoz, ellenben ha bármelyik operandus nem szám és nem is szöveg (vagy szám alapú szöveg), máris megpróbálja az interpreter meghívni a metametódust.
- **`__len`**: a hossz lekérdezés művelete (Luaban: `#`). Ha az objektum nem szöveg, akkor ez a metódus meg lesz hívva. Ha nincs ilyen metódus, és az objektum egy tábla, akkor az alap hossz lekérdezés operátor lép működésbe. A meghívás `return` value-ja a visszaadott érték.

- **__index**: Az egyik véleményem szerint legtöbbször felülírt metametódus. Alapját adja az **OOP** programozásnak a nyelvben. `obj[key]` alapú értéklekérdezőskor hívódhat meg. Több esetben is meghívódhat:
 - ha az objektumunk **nem** tábla
 - ha az objektumunk tábla: akkor hívódik meg, ha a táblában nem található a keresett kulcsú érték

Az `index` metametódus nem csak funkció lehet, hanem tábla is. Ha funkció, akkor meghívódik, két argumenttal: az 1. argumentum maga az objektum lesz, a 2. pedig a keresett kulcs. Tábla esetén pedig ott próbálja megkeresni az adott kulcsú értéket, ilyenkor ez a lekérdezés sem kerül ki a metatáblák meghívódását.

- **__newindex**: Működése hasonló az **__index** metametódushoz. Akkor hívódik meg, amikor egy objektumon egy `obj[key] = value` értéket beállítanak. A beállított metametódus szintén lehet funkció, vagy tábla, mint az **__index** metametódusnál. Tábla argumentum esetén azon a táblán állítja be az értéket az interpreter, funkció esetén pedig meghívódik a következő argumentumokkal: objektum, key, value. Ekkor a meghívott funkció a **rawset** funkció segítségével állíthat az objektumon értékeket (kikerülve a rekurzív metametódus meghívásokat).
- **__call**: Meghívás operátor: *func(args)*. Akkor hívódik meg ez a metametódus, ha megpróbálunk meghívni egy nem funkció értéket. Ekkor `func`-ban kerül megkeresésre és meghívásra a metametódus (triviálisan csak akkor, ha létezik). Meghívása esetén `func` a legelső argumentum, a többi argumentum pedig az eredeti meghívás argumentuma (`args`). A meghívott funkció visszatérési értékei maga a meghívás visszatérési értékei is. Ez az egyetlen metametódus, amely több értékkel is visszatérhet.
- **__gc**: Garbage collector metódus. Azelőtt hívódik meg, mielőtt a garbage collector felszabadítja az adott táblát/userdatat. Ez jó lehet például megnyitott fájlok, vagy bármi más megnyitott erőforrások bezárására.

Jó szokás a lehető legtöbb metametódust definiálni az új metatáblánkban. A **__gc** metametódus csak akkor működik, ha az összes metametódus sorban implementálva lett. Az összes metametódus részletesen megtekinthető a Lua Manualban.

Mivel a metatáblák átlagos táblák, ezért nem kötelező csak a metametódusokat tartalmazniuk, tartalmazhatnak tetszőleges adatokat is, amelyeket a bennük definiált funkciók felhasználhatnak. Például a **tostring** funkció a metatáblában meghívja a **__tostring** nevű funkciót (amely igazából egyedi metametódusnak is tekinthető), ha létezik, majd azt adja vissza, amit a funkció meghívásából visszakapott. [15]

2.3. Lua sajátosság: osztályok, öröklődések implementálása

metatáblák segítségével

Az előző szekcióban megemlített `__index` metametódussal tudunk osztályokat implementálni. Minden objektumnak van egy adott állapota, a táblának is.

A táblának van egy identitásuk, amely független a bennük tárolt értékektől, két tábla ugyan azokkal az értékekkel különböző objektumok is lehetnek, mindeközben egy tábla vagy userdata különböző értékekkel rendelkezhet különböző időkben, mégis ugyan az az objektum.

A táblának is van saját életciklusuk, amelyek függetlenek attól, hogy hol lettek létrehozva, vagy hogy ki hozta őket létre.

Az objektumoknak saját metódusai, műveletei vannak, csakúgy mint egy táblának.

2.3.1. Táblaműveletek, self kulcsszó

Példakód táblaműveletre:

```
Account = {balance = 0}
function Account.withdraw(v)
    Account.balance = Account.balance - v
end
```

Ebben a példakódban létrehozunk egy `Account` táblát, amelyben van egy `withdraw` funkció. Ez a `withdraw` funkció a globális `Account` táblára vonatkozik, onnan fog értéket levonni. Ha megváltoztatjuk a nevét a táblának, vagy `nil`-el tesszük egyenlővé, máris nem fog működni az alábbi kód.

A következő példakódban kiküszöböljük ezen problémákat, átnevezhetővé tesszük az objektumot, felülírhatóvá:

```
Account = {balance = 0}

function Account.withdraw(self, v)
    self.balance = self.balance - v
end

a1 = Account; Account = nil

a1.withdraw(a1, 100.00) — OK

a2 = {balance=0, withdraw = a1.withdraw} — itt figyelni kell arra, hogy
    ↪ az Account-ot már nille raktuk (toroltuk), ezért a1-et használunk

a2.withdraw(a2, 260.00) — OK

—alternatív hasznalati mód:
a1:withdraw(100.00) —OK
a2:withdraw(260.00) —OK
```

A self paraméter bevezetésével már nem csak arra az objektumra/táblára érvényes a funkció, hanem bármelyik másra. Azonban ez a self paraméter úgymond saját magunk által implementált, kitalált.

Viszont ebben a nyelvben létezik a **self** kulcsszó, változó. A self változó mindig az adott objektumra/táblára mutat, vagyis önmagára. Ez egy nagy segítség az **OOP** implementálása során.

A Lua nyelvben a self változót kétféle képpen használhatjuk:

1. megoldás: az előző példakódban tárgyalt 1. argument bevezetése. Abban a példakódban a self argumentumnevet igazából bármire kicserélhetjük, mivel a Lua nyelv akkor is oda fogja rakni első argumentként az objektumot, ha úgy használjuk, hogy: `Account:withdraw(v)`.

2. megoldás:

```
Account = {balance = 0}

function Account:withdraw(v)
    self.balance = self.balance - v
end

a1 = Account; Account = nil

a1.withdraw(a1, 100.00)    — OK

a2 = {balance=0, withdraw = a1.withdraw} — itt figyelni kell arra, hogy
    ↪ az Account-ot már nincsen raktuk (toroltuk), ezért a1-et használunk

a2.withdraw(a2, 260.00) — OK

—alternatív használati mód:
a1:withdraw(100.00) — OK
a2:withdraw(260.00) — OK
```

Ebben az esetben a withdraw funkciót használva mindig a saját objektumunkra fog mutatni a self változó, kivéve akkor, ha két paraméter segítségével használjuk, és az első paraméter az objektum.

Most már az objektumjainknak van állapota, identitása, metódusai. Azonban nincsenek rendes osztályok, öröklődések, láthatósági szabályok.

2.3.2. Konstruktor, Objektum példányosítás

A korábban említett `__index` metódus segítségével tudunk osztályainknak "konstruktort" írni:

```
Account = {}

function Account:new(o)
    o = o or {} — create object if user does not provide one
    setmetatable(o, self)
    self.__index = self
    return o
end

function Account:withdraw(v)
    if v > self.balance then error"insufficient funds" end
    self.balance = self.balance - v
end

function Account:deposit(v)
    self.balance = self.balance + v
end

a = Account:new{balance = 0}
a:deposit(100.00)
a:withdraw(100.00)

print(a.balance) —> 0
```

Ebben az esetben létrehozunk egy `Account` táblát/objektumot, amelynek lesz `new` funkciója, ez lesz a "konstruktorunk".

Ez a funkció úgy működik, hogy 1. argumentként kaphat egy állapot objectet/állapot táblát (ekkor a már meglévő objektum lesz állapotként felhasználva), vagy csinál egy teljesen üreset.

Az `"o"` táblára pedig saját magát, vagyis a jelenlegi objektumot (`Account`) állítja be metatáblának. Ezután pedig a `self` (`Account`) táblán beállítja az `__index` metametódus értékét saját magára (tehát tábla lesz, nem funkció). Ennek segítségével minden definiált funkció működni fog úgy, hogy a `self` megmarad saját állapotként.

Így létrehozhatjuk az `"a"` változónév alatt eltárolt új objektumunk realizációját, amely most már bármire átnevezhető, bármikor törölhető. Az `"a"` objektumnak saját állapota lesz, saját belső változókkal, ugyanakkor az előre definiált `deposit` és `withdraw` funkciók ugyan úgy működni fognak. Ha az `"a"` objektumon meghívjuk ezeket a funkciókat, akkor a `self` keyword magára az `"a"` táblára (objektumra) fog mutatni, így lesz saját állapota az objektumnak.

2.3.3. Destruktor

A korábban említett `__gc` metódus segítségével tudunk destruktort írni objektjeinknek, ez lefut, mielőtt a garbage collector felszabadítaná őket. Hasznos lehet olyan erőforrások bezárására, amelyek maguktól nem szabadulnak fel.

2.3.4. Öröklődés

Az előző, legutolsó példakódot felhasználva könnyedén tudunk öröklődést implementálni. Ne felejtsük el, hogy a **self** változó mindig önmagunkra mutat. Az alábbi példakódban megtekinthető az öröklődés mechanizmusa:

```
SpecialAccount = Account:new() —letrehozunk egy új Account objektum
    ↪ példányt, amelyet SpecialAccountként nevezünk el. Azonban a
    ↪ SpecialAccount minden olyan tulajdonsaggal rendelkezik, mint az
    ↪ Account.

s = SpecialAccount:new{limit=1000.00, balance=0} —Ezert lehet ugyan úgy
    ↪ példányosítani ot is, mint az Account objektumot.
s:deposit(100.00) —ebben az esetben a self parameter az "s"-re mutat. Az
    ↪ "s" metatáblája pedig SpecialAccount lesz. Ezáltal az "s" örökli
    ↪ a SpecialAccountot, ami pedig örökli az Accountot.
—mivel a Lua nem találja meg a deposit funkciót SpecialAccountban, ezért
    ↪ tovább megy, és az Accountban megtalálja, meghívja azt. Azonban a
    ↪ metódusok felulírhatóak az új objektumban:

function SpecialAccount:withdraw(v)
    if v — self.balance >= self:getLimit() then
        error"insufficient funds"
    end
    self.balance = self.balance — v
end

function SpecialAccount:getLimit()
    return self.limit or 0
end

s:withdraw(200.00) —[[mostmar a Lua interpreternek nem kell az Accountig
    ↪ visszamennie, mivel a SpecialAccountban a withdraw definálva van
    ↪ , ezért azt fogja meghívni.
a Lua nyelv érdekesége, hogy nem kell új osztályt létrehozni annak, hogy
    ↪ különleges viselkedést defináljunk egy adott objektumra. Felül
    ↪ lehet írni az "s" objektum funkcióit is, hogy maskepp
    ↪ viselkedjenek]]
function s:getLimit ()
    return self.balance * 0.10
end

s:withdraw(200.00) —ekkor a Lua a SpecialAccount withdrawjat fogja
    ↪ meghívni, viszont a getLimit funkció a "self" miatt az "s"
    ↪ objektum felülírt funkciója lesz, azt fogja meghívni.
```


2.3.5. Láthatósági szabályok osztályokon belül

A többi nyelvhez hasonlóan itt is lehet alapszintű láthatóságot beállítani az osztályokon belül. Alapvetőleg ez sincs implementálva a nyelvben, azonban a Lua flexibilis nyelvnek készült, ezáltal mi magunk elkészíthetjük. Egy tábla helyett két táblát fogunk használni egy objektum reprezentálására: egyet az állapotára, a másikat pedig interfacéként, amelyen keresztül interaktálni tudunk vele. A belső -állapot- táblát pedig csak a definiált funkciók érik el. Az előzőekhez hasonló példakód:

```
function newAccount(initialBalance)
    local self = {balance = initialBalance, LIM = 10000.00}

    local withdraw = function(v)
        self.balance = self.balance - v
    end

    local deposit = function(v)
        self.balance = self.balance + v
    end

    local extra = function()
        if self.balance > self.LIM then
            return self.balance*0.10
        else
            return 0
        end
    end

    local getBalance = function() return self.balance + extra() end

    return {
        withdraw = withdraw,
        deposit = deposit,
        getBalance = getBalance
    }
end

acc1 = newAccount(100.00)
acc1.withdraw(40.00)
print(acc1.getBalance())    —> 60
```

Ebben a példakódban a newAccount a konstruktorunk, amely vár egy értéket, hogy a bankszámlánk mennyi pénzzel indul. Létrehoz egy belső táblát, amely csak a belső scopen belül létezik, self néven. Ez lesz az állapot tábla. Minden benne tárolt érték "private" láthatóságúak.

A többi funkciót ugyan abban a belső scopeban definiálja, ezért alapvetőleg nem is lennének elérhetőek kívülről, azonban egy új tábla hozódik létre, az interface tábla return valueként, mely elérhetővé teszi ebben a belső scopeban létrehozott funkciókat.

Ebben az esetben a self keyword nem a szokványos változó lesz, hanem egy elnevezett változó a belső scopeban.

A definiált funkciók folyton arra a "self" változóra fognak hivatkozni, amelyet mi definiáltunk, ezért nem fogad extra argumentet arról, hogy melyik objektum tartalmazza az állapotot.

Emiatt a ":" használatát mellőznünk kell, csak a "." használatával tudjuk használni a funkciókat. Definiáltunk egy olyan funkciót is, amely szintén "private" láthatóságú, ez az `extra` nevű funkció. [16]

2.4. Programtól elvárt működés

A program a szerverek konfigurációja közben igyekszik a jelenleg elérhető legjobb biztonsági megoldások alkalmazására, titkosítás, SSL beállítás és egyéb dolgok terén (például chroot, tls-crypt).

Az OpenVPN Community szervert a program tudja kezelni, telepíteni. A telepítést az apt-get beépített segédprogrammal végzi. Kezelni tudja az alábbi dolgokat:

- kliensek létrehozása (kulcs alapú), törlése (kulcs esetén revoke)
- kliensek számára személyre szabott .ovpn config generálás
- init.d beállítások, az eredeti daemon beállítása automatikus indításra
- külön user létrehozása a szerver futtatására
- tls-crypt használata az üzenetek titkosításához 2.4.1
- chroot használata a szerver futtatásához a megnövelt biztonság érdekében 2.4.2

Fontos kiemelni, hogy az OpenVPN implementáció a server.conf config fájlt módosítja, azonban nem ellenőrzi kifejezetten a config fájl helyességét, így nem tud config fájlt javítani sem.

Apache2 és Nginx webszervert is tud kezelni a program, ebbe beletartozik:

- telepítés apt-gettel
- honlap hozzáadása külön directoryval
- honlap törlése
- SSL certificate kezelés Let's Encrypton belül certbot segítségével, a certbot snapdn keresztül települ; up-to-date SSL beállításokra törekszik a program
- külön user a daemonoknak mind Apache2, mind nginx esetében
- a jelenlegi implementáció csak statikus weboldalt támogat, ezért egy user van egy egész daemonnak, nincs külön weboldalanként user
PHP/bármilyen szerveroldali kiegészítő esetén célszerű különböző usereket használni weboldalanként

Szintén konfigurációbeállítást végez a program a webszerverek esetében is, teljes mértékben nem tudja a config fájlok helyességét ellenőrizni, sem megjavítani őket, ha már alaphibából hibásak.

Tűzfal gyanánt az iptables nevű beépített Linux segédprogramot tudja kezelni.

Iptables funkcionalitások:

- telepítés apt-gettel ha nincs fent
- IPv4 támogatás
- Port nyitás, zárás
- Bizonyos portra csak bizonyos IP-ről csatlakozás engedélyezése
- Bizonyos IP cím felé csak bizonyos kimeneti portok felé kimenő kapcsolat engedélyezése
- Nyitott portok listázása
- Zárt portok listázása
- Engedélyezett kimenő kapcsolatok listázása
- OpenVPN szerverhez köthető NAT Forwardot listázása, kezelése
- A fenti szabályok korlátozhatóak az egyes network interfacekra
- Szabályellenőrzés, például ha engedélyezünk egy portot, és nincsenek letiltva a nem engedélyezett bejövő kapcsolatok, akkor figyelmeztetés
- Ki-be kapcsolható togglek:
 - Bejövő forgalom csak az engedélyezettek közül jöhet be - ekkor ellenőrzi, hogy SSH port az engedélyezettek között van-e, és ha nincs, akkor figyelmeztet
 - Kimenő forgalom csak az engedélyezettek felé mehet ki

2.4.1. Mi az a tls-crypt?

A tls-crypt egy OpenVPN beállítás, amely lehetővé teszi, hogy az összes csomag digitálisan legyen aláírva, továbbá titkosítva akár szimmetrikus eljárásokkal. Ez a key-exchange előtt is már alkalmazásra kerül. [23] Többféle előnye is van:

- nehezebb megkülönböztetni az OpenVPN csomagjait
- több biztonságot ad azzal, hogy a TLS kapcsolódást is titkosítja már
- Man-in-the-middle támadás ellen is védhet, továbbá DoS/DDoS támadások ellen is

2.4.2. Mi az a chroot?

A `chroot(const char *path)` egy olyan C API hívás, amely a jelenlegi gyökérkönyvtárat átváltoztatja arra, amelyet az első argumentben megadnak neki. Ez azt jelenti, hogy a jövőben az összes fájlművelet ehhez a gyökérkönyvtárhoz lesz relatív, abban az esetben, ha `/`-lel kezdődik a megnyitott fájl pathja. Kiterjed a mostani és az az összes child-processzre is a változtatás. [5]

Csak privilegizált programok tudják általában ezt a C API hívást meghívni. Vigyázni kell programozói szempontból a használatával, mivel ez igazából csak a védelem egy alapját adhatja meg, bizonyos módszerekkel kikerülhető.

2.5. Hasonló alkalmazások

Az Internet sokaságában sok hasonló funkcionalitást implementáló alkalmazást lelhetünk fel.

2.5.1. Apache2 és nginx

Többféle neves implementáció is létezik az Apache2 és nginx kezelésére.

2.3. ábra. A webserverek logói



(a) Az apache logója



(b) Az nginx logója

Legelőször személy szerint a **VHCS** vagyis a Virtual Hosting Control Systemmel találkoztam jópár évvel ezelőtt. A VHCS már nincs aktívan fejlesztve, az utolsó kiadott verzió belőle a 2.4.8-as verzió, amely 2009-ben jelent meg. [26] Többféle szervert is tudott kezelni, néhány ezek közül: Apache, ProFTPD, MySQL, Bind. A későbbiekben az ispCP projekt erre épült. [12]

Az **ispCP** projektből alakult ki az **i-MSCP**, amelynek a legutolsó kiadott verziója 2018-ban lett kiadva [7]. Ez már több featurevel rendelkezik:

- kezeli az Apache2-t és az nginx-et egyaránt
- támogatja a PHP szerver oldali webfejlesztést
- kezeli a bind DNS szerveret
- kezel Mail Transfer Agentet
- kezel Mail Delivery Agentet
- kezel adatbázisszervereket
- kezel FTP-szervereket
- pluginnal való bővítés lehetősége adott

Azonban jelenleg a dolgozat írásakor a projekt nincs aktívan fejlesztve. Az előzőleg említett VHCS, ispCP és i-MSCP szerverkezelő implementációk mindannyian nyílt forráskódúak, ingyenesen felhasználhatóak, többnyire Linux alapú rendszerekre készültek.

A következő lehetséges implementáció az **ISPConfig**. Ez is szintén nyílt forráskódú, ingyenesen felhasználható, és Linux alapú rendszerekre készült. PHP-ban íródott. Legutolsó stabil verziószám: 3.2.11, amely 2023. augusztus 8.-án lett kiadva. [9]

Sok daemont, szervert kezel [11]:

- kezeli az Apache2-t és az nginx-et egyaránt
- kezeli a PHP-t is, azonban külön fel kell rakni és beállítani a felületen

- kezeli a postfix SMTP szervert
- kezeli a Dovecot POP3/IMAP szervert
- kezeli a PureFTPd ftp szervert
- kezeli a bind, PowerDNS DNS szervert
- kezel adatbázisszervereket: MariaDB és MySQL
- többféle nyelvet is támogat

A következő táblázatban a myVestaCP implementáció kerül összehasonlítása az ISPConfig-gal. A **myVestaCP** szintén nyílt forráskódú, a **vestaCP** forkolt, továbbfejlesztett változata. A két implementáció összehasonlítása a 2.1. táblázatban látható:

2.1. táblázat. ISP Config [11] összehasonlítása myVestaCP-vel [18]

	ISPConfig	myVestaCP
Ingyenes	igen (kivéve Dokumentáció [10])	igen
Szerver kezelés támogatott OS	Kizárólag Linux support	Kizárólag Debian support
Kezelt szervertípusok	Web, SMTP, POP3/IMAP, webmail, FTP, DNS, SQL, tűzfal (iptables)	Web, SQL, DNS, POP3/IMAP, webmail, Antivirus, FTP, node.js web, tűzfal (iptables, fail2ban) [25]
Támogatott nyelvek	Többnyelvű	Angol, többi nem ismert
Web interface alapú	Igen	Igen
Testreszabhatóság	Léteznek hozzá modulok, pluginok	Léteznek hozzá modulok, pluginok, azonban nem annyira nagy a választék

2.5.2. Tűzfal

Tűzfalak terén a Linux rendszerekben a legelterjedtebb netfilter implementációk jelenleg az iptables és az nftables.



2.4. ábra. A netfilter logója - ezt használja az iptables is

Az **iptables** legelső releaseja 1998-ban jelent meg, a legutolsó stabil kiadása 2022. május 13.-án került kiadásra. C-ben íródott. [19]

Többféle táblát használ a parancsok feldolgozására, tud csomagokat szűrni (ki és bemenő, továbbá átirányított csomagokat), csomagok tartalmát módosítani, átirányítani. Tud NAT-ot is implementálni. [8]

Az **nftables** a Linux kernel 3.13-as verziójától érhető el, pontosabban 2014 január 19.-e óta. Az nftables néhány iptables-hez köthető maradandó örökséget cserél le, tudja többnyire ugyan azt a funkcionalitást, mint az **iptables**. Előnye az iptables-hez képest, hogy kevesebb kód duplikációval rendelkezik, és könnyebb az új protokollokra való kibővítése. [27] Jobban skálázható is, jobb a teljesítménye az iptablesnél, főleg akkor, ha például iptables-ban sok saját magunk által definiált chain-t (láncot) használtunk.

Jónéhány Linux disztribúcióban, például a Debian újabb verzióiban egyszerre mindkettőt használhatjuk. Az **iptables -V** parancsot lefuttatva megnézhetjük, hogy milyen iptablessal dolgozunk. Az újabb verziókon az iptables igazából nftables backendet használ, ezt a következőképp láthatjuk a 2.5. ábrán:

```
lackos@localhost:~$ sudo iptables -V
iptables v1.8.7 (nf tables)
lackos@localhost:~$
```

2.5. ábra. Debian 11-es rendszeren használt iptables, amely nftables backenddel dolgozik

Létezik iptables szabálylánc konvertáló program is, amellyel meglévő szabályainkat nftables syntaxra konvertálhatjuk.

Az elkészítendő program az **iptables** backendet használja, mivel az a régebbi rendszereken is használható, továbbá még mindig elterjedt az **nftables** mellett is. Többféle elterjedt tűzfal implementáció is van, amelyek az **iptables**-t használják fel, a következő 2.2. táblázatban kettőt fogok összehasonlítani:

2.2. táblázat. ufw összehasonlítása firewalld-vel

	ufw	firewalld
Használatának módja	CLI, de létezik hozzá GUI is	CLI, de létezik hozzá GUI is
Ingyenes	igen	igen
Parancsok formátuma	egyszerű parancsokkal rendelkezik, angol parancsok könnyű paraméterezéssel [24]	kevésbé felhasználóbarát, komplexebb parancsok [6]
Támogatott protokollok	TCP, UDP. A ping engedélyezéséhez már iptables szabályokat kell írni	/etc/protocols-ból támogatja az összeset, portnál TCP/UDP/sctp/dccp protokollt támogat
Testreszabhatóság	nincsenek hozzá pluginok	nincsenek hozzá pluginok

2.5.3. OpenVPN

A következő kétoldalas 2.3 táblázatban a saját implementációm, amely OpenVPN Community szervert használ, hasonlítom össze az OpenVPN Access Server-rel.



2.6. ábra. Az OpenVPN logója

Fontos megemlíteni, hogy OpenVPN szerver menedzser implementáció több is létezik, létezik C#-ban írt, létezik Bash Scriptben írt is. A minél nagyobb cross-platform támogatás lehetősége miatt azonban úgy gondolom mégis helyt áll a program létezése, a Lua nyelv egyszerűsége, nagyszerű cross-platform támogatása miatt. Értelemszerűen a Bash Script implementációk csak Linuxra korlátozódnak. GitHubon létezik web alapú interface is az OpenVPN Community Serverhez.

2.3. táblázat. Az OpenVPN Access Server összehasonlítása saját implementációval

	OpenVPN Access Server [2]	Saját implementáció (OpenVPN Community)
Ingyenes	nem	igen
Kliens támogatott OS	Cross-platform	Cross-platform
Szerver kezelés támogatott OS	a legtöbb támogatott	kizárólag Linux support
User autentikáció módja	webes alapú felhasználónév-jelszó, PAM, LDAP RADIUS, SAML, stb. Saját auth script is készíthető.	kizárólag kulcs alapú (ez bővíthető saját auth scriptekkel, továbbá pluginokkal, például: PAM pluginnal).
Többfaktoros hitelesítés	beépített lehetőségek, továbbá pluginokkal bővíthető lehetőségek	saját auth scripttel kivitelezhető a 2FA, vagy akár pluginként
Access Control (ACL)	beépített lehetőségek	saját auth scripttel, vagy tűzfallal kivitelezhető
Tunnel átirányítás	beépített lehetőségek: full-tunnel és split-tunnel	megoldható a kliens konfigurálásával (a <code>route</code> program használatával), továbbá szerveroldali konfigurálással (redirect-gateway)

2.4. táblázat. A 2.3 táblázat folytatása

	OpenVPN Access Server [2]	Saját implementáció (OpenVPN Community)
Support	professzionális support ticketing rendszerrel	OpenVPN community fórum, Community Ticket report
Certificatek kezelése	automata, beépített tanúsítványkezelés van, külső infrastruktúrát is lehet használni	saját, automatizált tanúsítványkezelés
Kezelőfelület	webes alapú, command line	alapvetőleg command line, azonban webes alapú felület is lehetséges 3rd party szoftverekkel

3. fejezet

Tervezés

A program az OO alapelveket igyekszik követni. Elsősorban modulokból áll, amelyek osztályként is értelmezhetők. Vannak olyan modulok, amelyekben csak a helyi scope-ban elérhető kódok vagy változók vannak, tehát "private" elérésűek.

A következőekben a modulok UML diagramját fogom külön-külön bemutatni. A következő 3.1. ábrán tekinthetők meg a láthatósági szabályok az UML diagramokon:

Láthatósági szabályok
- Private + Public

3.1. ábra. Láthatósági szabályok az UML diagramokon

3.1. general, apt_packages, utils modulok felépítése, feladata

Legelőször a general, linux, apt_packages és az utils modul került megtervezésre. Felépítésük a 3.2. ábrán látható.

general	apt_packages
+ lineEnding : string	
+ getOSType() + clearScreen() + sleep(n) + strSplit(str, sep) + deepCompare(tbl1, tbl2) + concatPaths(...) -> varargs + extractDirFromPath(path) + readAllFileContents(filePath) + trim2(s)	+ isPackageInstalled(packageName) + installPackage(packageName)

3.2. ábra. A general és apt_packages modul felépítése

A **general** modulban olyan funkciók és változók találhatók, amelyek a legtöbb OS-en működnek, és fontos szerepet töltenek be a program működése közben.

Az **apt_packages** modul Linux-specifikus kódokat tartalmaz, a modul feladata az apt program segítségével a csomagok menedzselése, telepítése.

Az **utils** modul legfőképp debugra volt használva, jelenleg épp nincs alkalmazva a program kódjában, így ezt a modult nem mutatom be.

3.2. linux modul felépítése, feladatai

A következő 3.3. számú UML diagramon a **linux** nevezetű modult fogom bemutatni, amely linux-specifikus kódrészleteket tartalmaz. Erre a modulra épül a legtöbb modul.

linux
<ul style="list-style-type: none"> + exists(file) + isDir(path) + listDirFiles(path) + mkdir(path) + deleteFile(path)) + deleteDirectory(path) + execCommand(cmd) + getServiceStatus(serviceName) + isServiceRunning(serviceName) + isProcessRunning(name) + stopService(serviceName) + startService(serviceName) + restartService(serviceName) + systemctlDaemonReload() + checkIfUserExists(userName) + createUserWithName(userName, comment, shell, homeDir) + updateUser(userName, comment, shell) + getUserHomeDir(userName) + execCommandWithProcRetCode(cmd, lines-Returned, envVariables, redirectStdErrToStdIn) + copy(from, to) + copyAndChown(user, from, to) + chown(path, userName, isDir) + chmod(path, perm, isDir)

3.3. ábra. A linux modul felépítése

A **linux** modul feladatai szerteágazóak:

- vannak benne fájl- és könyvtár manipulációs funkciók
- vannak benne parancsfuttató funkciók (amelyek közül az egyik tökéletesen kompatibilis Windows-sal is), továbbá servicet és systemctlt kezelő funkciók
- vannak benne felhasználókat módosító funkciók (felhasználók létrehozása, módosítása, felhasználó létezésének ellenőrzése, felhasználó home dir lekérése)
- vannak benne ownershipet változtató funkciók
- vannak benne fájl és mappa hozzáférési szabályokat változtató funkciók

3.3. OpenVPN modulok felépítései, feladatai

A következőekben az OpenVPN szerver kezelését kezdtem el megtervezni, implementálni. Az összes modul a program jegyzékén belül a **modules/vpnHandler** jegyzékben található meg.

Több részmodulra lett szétosztva:

- **OpenVPN**: ez maga **bootstrap** modul, ebben van OpenVPN csomagot feltelepítő funkció, szervert indító/leállító funkció, továbbá ez a modul tölti be a **server_impl** modult
- **server_impl**: ez a modul kezeli a szerverrel kapcsolatos legtöbb dolgot
előkészíti a könyvtárakat, feltelepíti az Easy-RSA Certificate kezelőt, létrehoz egy saját CA-t (**Certificate Authority**-t), szerver-oldali certificate és kulcsfájlokat generál, saját OpenVPN usert hoz létre a szervernek, tls-auth/tls-crypt kulcsot generál, a server configot beüzemeli, beállítja a server daemon auto-startot a `/etc/default/openvpn` fájlban
- **config_handler**: az OpenVPN szerver és kliens configját kezelő modul, config parselést és writeolást implementál. A parser és writer az eredeti OpenVPN parser kódja alapján épült, amely megtalálható a [22] hivatkozás alatt.
- **clienthandler_impl**: ez a modul kezeli teljesen a klienssel kapcsolatos dolgokat:
kliens certificatet, private keyt hoz létre; kliens configot hoz létre, amelybe beágyazza a generált fájlok tartalmát (így 1 db config filera van szüksége az OpenVPN kliensnek); certificate revoket támogat; továbbá le lehet kérni az összes jelenlegi klienst, amelyek valóságosak (nincsenek revokeolva)

Maga az OpenVPN **bootstrap** és **config_handler** modul nagyon egyszerű felépítésű, amely látható is a következő 3.4. és 3.5. ábrán.

OpenVPN
+ errors : table + serverImpl -> OpenVPN_server_impl module
+ isOpenVPNInstalled() + installOpenvpn() + isRunning() + stopServer() + startServer() + initDirs()

3.4. ábra. Az OpenVPN bootstrap modul felépítése

OpenVPN_config_handler
+ parseOpenVPNConfig(linesInStr) + writeOpenVPNConfig(parsedLines)

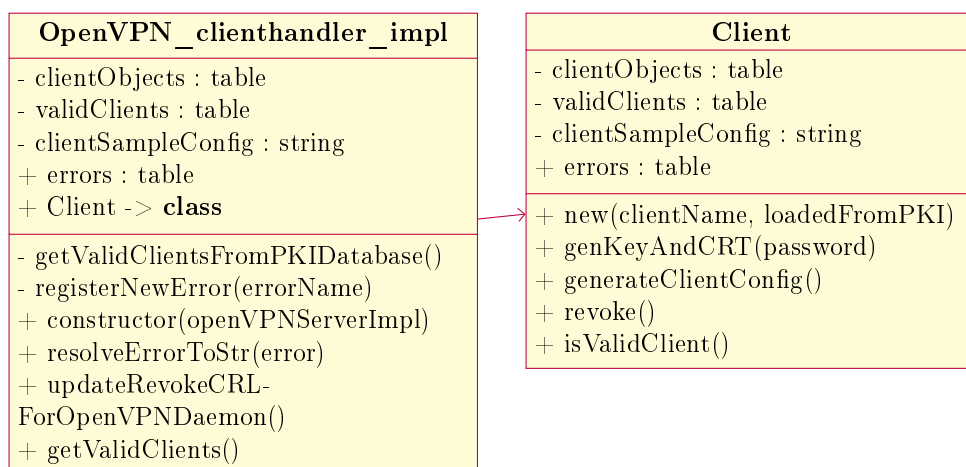
3.5. ábra. Az OpenVPN_config_handler bootstrap modul felépítése

A **server_impl** modul minden OpenVPN szerverrel kapcsolatos művelet magja, ezért sok funkcióval rendelkezik. Felépítése a 3.6. ábrán látható.

A klienseket kezelő **OpenVPN_clienthandler_impl** modulban pedig két osztály is helyet foglal az OO alapelveket követve. Felépítése a 3.7. ábrán látható.



3.6. ábra. Az OpenVPN_server_impl modul felépítése



3.7. ábra. Az OpenVPN_clienthandler_impl modul felépítése

3.4. nginx modulok felépítései, feladatai

Az nginx-et kezelő modulok is több részre lettek osztva. Az összes modul a program jegyzékén belül a **modules/nginxHandler** jegyzékben található meg. A modulok leírásai, feladataikkal:

- **nginx**: ez maga egy **bootstrap** modul, ebben van nginx csomagot feltelepítő funkció, szervert leállító/elindító funkció, továbbá ez a modul tölti be a **server_impl** modult
- **server_impl**: ez a modul kezeli a szerverrel kapcsolatos legtöbb dolgot előkészíti a könyvtárakat, létrehoz az nginx daemonnak, workereknek egy usert; támogatja automatikusan a weboldalak létrehozását, törlését; támogatja az SSL-t; minden támogatott featuret bekonfigurál automatikusan
- **config_handler**: nginx szerver konfigurációját kezelő modul, config parselést és write-olást implementál. A parser és writer az eredeti nginx parser kódja alapján épült, amely megtalálható a [20] hivatkozás alatt

Az nginx **bootstrap** modul ez esetben is egyszerű felépítésű, amely látszik a 3.8. ábráról is.

Azonban a **config_handler** modul ebben az esetben már kissé bonyolultabb, mivel az nginx syntaxa is komplikáltabb. Két osztályt tartalmaz. Felépítése a 3.9. ábrán látható.

nginx
+ errors : table + serverImpl -> nginx_server_impl module
+ isInstalled() + install() + isRunning() + stopServer() + startServer() + initDirs()

3.8. ábra. Az nginx bootstrap modul felépítése

nginx_config_handler	nginxConfigHandler
+ nginxConfigHandler -> nginxConfigHandler class	+ new(linesInStr, paramToLine)
- concatArgsProperly- ForBlockName(args)	+ getParsedLines()
- parseNginxConfig(linesInStr)	+ getParamsToIdx()
- formatDataAccordingQuoting(tbl)	+ insertNewData(dataTbl, pos)
- doPaddingWithBlock- Deepness(blockDeepness)	+ deleteData(pos) - je- lenleg nem használt
- writeNginxConfig(parsedLines)	+ toString()

3.9. ábra. Az nginx_config_handler modul felépítése

A következő 3.10. ábrán látható a `nginx_server_impl` modul felépítése:

<code>nginx_server_impl</code>
<ul style="list-style-type: none"> - <code>sampleConfigForWebsite</code> : string + <code>nginxUser</code> : string + <code>nginxUserComment</code> : string + <code>nginxUserShell</code> : string + <code>baseDir</code> : string + <code>errors</code> : table
<ul style="list-style-type: none"> - <code>registerNewError(errorName)</code> + <code>constructor(_bootstrapModule)</code> + <code>resolveErrorToStr(error)</code> + <code>formatPathInsideBasedir(path)</code> + <code>initDirs()</code> + <code>checkNginxUserExistence()</code> + <code>createNginxUser(homeDir)</code> + <code>updateExistingNginxUser()</code> + <code>getNginxHomeDir()</code> + <code>getNginxMasterConfigPathFromDaemon()</code> + <code>initializeServer()</code> + <code>createNewWebsite(websiteUrl)</code> + <code>deleteWebsite(websiteUrl)</code> + <code>getCurrentAvailableWebsites()</code> + <code>initSSLForWebsite(webUrl, certDetails)</code>

3.10. ábra. Az `nginx_server_impl` modul felépítése

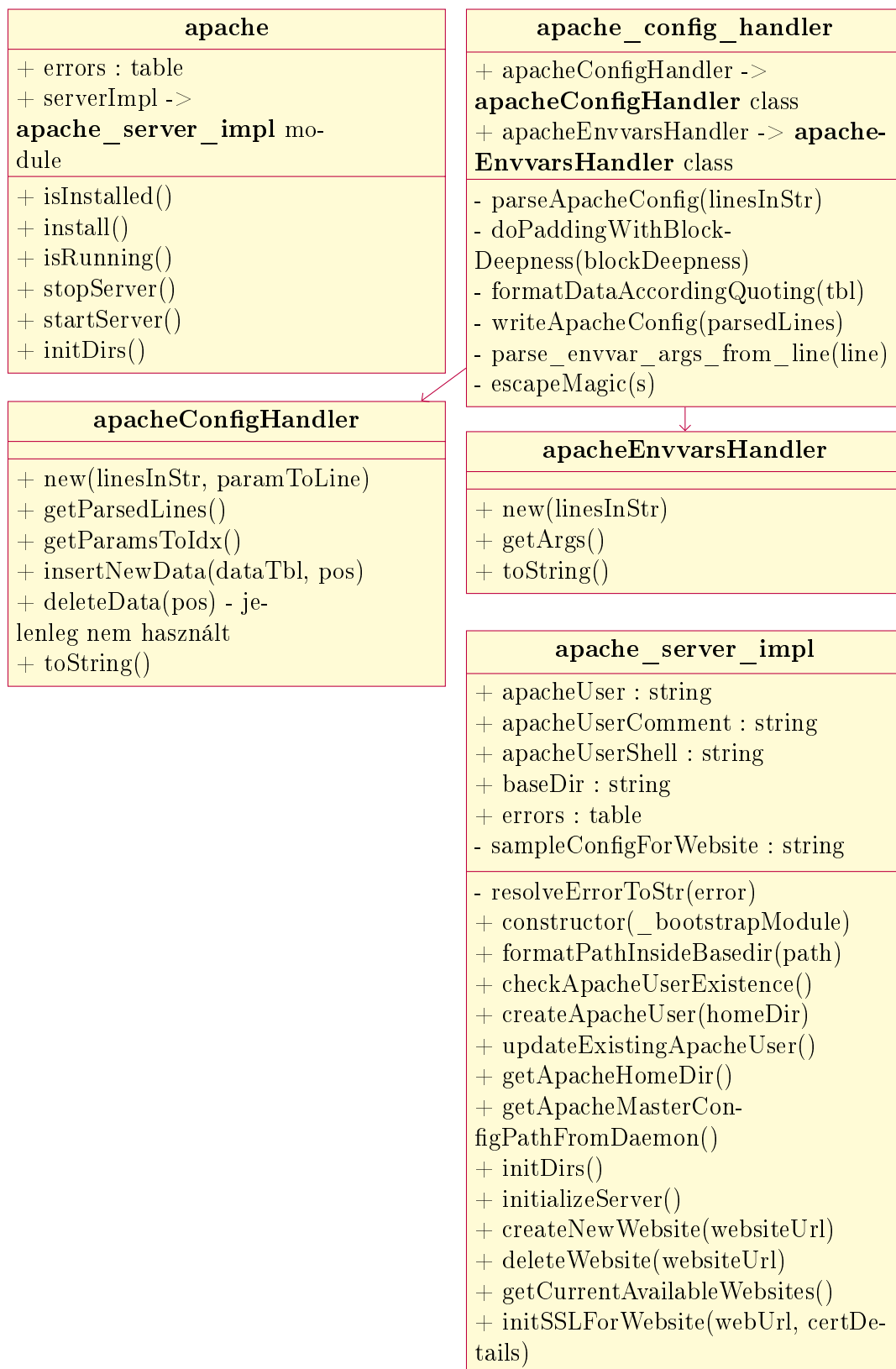
3.5. apache modulok felépítései, feladatai

Az apache kezeléséhez tartozó modulok a program jegyzékén belül a **modules/apacheHandler** jegyzékben található meg.

A modulok leírásai, feladatokkal:

- **apache**: ez maga egy `bootstrap` modul, ebben van `apache2` csomagot feltelepítő funkció, szervert leállító/elindító funkció, továbbá ez a modul tölti be a `server_impl` modult
- **server_impl**: ez a modul kezeli a szerverrel kapcsolatos legtöbb dolgot: előkészíti a könyvtárakat, létrehoz az `apache2` daemonnak, processeknek egy usert támogatja automatikusan a weboldalak létrehozását, törlését; támogatja az SSL-t; minden támogatott featuret bekonfigurál automatikusan (`envvars`-t is)
- **config_handler**: apache szerver konfigurációját kezelő modul: Config parselést és writeolást implementál. Az `apache2` szerver parsere az előzőleg tárgyalt programokhoz képest bonyolultabb, ezért teljesen az alapoktól terveztem a parsert és a writert. Ehhez az `apache2` config syntax dokumentációját vettem segítségül, amely megtalálható a [3] hivatkozás alatt. Az `envvars` nevű fájl szerkesztését implementáló class is ebben a modulban van.

Az `apache bootstrap` modul ez esetben is egyszerű felépítésű, a `config_handler` pedig három osztályt tartalmaz. A `server_impl` modul hasonló bonyolultságú az `nginx_server_impl` modulhoz. Felépítésük megtekinthető a 3.11. ábrán.



3.11. ábra. Az apache implementáció moduljainak felépítése

3.6. certbot modul felépítése, feladata

A certbot csak egy modult foglal magában. Működéséhez több modult is felhasznál. A modulban megtalálható a certbot telepítése snapd-n keresztül, symlink létrehozás, SSL certificate létrehozás HTTP-01 challenge és DNS challenge segítségével, továbbá be is konfigurálja az adott webszervereket a használatához.

Habár a certbot akár az **apt** package managerrel is felrakható, az **apt-get install certbot** parancs segítségével, maga a program weboldalán a leírás szerint snapd-vel telepítik. [4]

A különbség főképp a kettő között az, hogy az **apt** package manager deb csomagokkal dolgozik, a **snap** pedig teljes archívumokkal. Teljesen külön repository-kal dolgoznak.

Az **apt** esetében a függőségek külön rakódnak fel a csomagok felleltelítésekör, és a **deb** archívumok csak a csomagokat tartalmazzák. Ebben az esetben a frissítések késhetnek, mivel több szervezet, személy is átnézi a frissítés tartalmát.

A **snapd** esetében egy teljes archívumot tölt le a rendszer, ebben benne van a csomag összes függősége pont azzal a verzióval, amellyel a csomag fejlesztői szerették volna. Ez az archívum egy "zárt térbe" kerül kicsomagolásra, és limitált hozzáférése van magához a rendszerhez (hasonlóan a Docker-alkalmazásokhoz). Frissítéskör a fejlesztő adja ki a frissítéseket, nem validálják őket külön, ezáltal gyorsabban eljut a userekhez. [28]

A **certbot** modul UML diagramja a következő 3.12. ábrán látható:

certbot
<ul style="list-style-type: none"> - errors : table - certFileName : string - keyFileName : string - dhParamFileName : string - dryRunStr : string - kizárólag debugra
<ul style="list-style-type: none"> - registerNewError(errorName) - sleep(n) + resolveErrorToStr(error) + isCertbotInstalled() + createCertbotSymlink() + installCertbot() + getCertDdatas(domain) + trySSLCertificateCreation(method, domain, webserverType) + init()

3.12. ábra. A certbot modul felépítése

3.7. snapd modul felépítése, feladata

A snapd modul egyszerű felépítésű. Feladata maga a snapd felleltelítése az **apt** package manager segítségével, továbbá csomagok telepítése, és a már felleltelített csomagok meglétének ellenőrzése.

A **snapped** modul UML diagramja a következő 3.13. ábrán található meg:

snapped
<ul style="list-style-type: none"> + isSnappedInstalled() + installSnapped() + isPackageInstalled(packageName) + installPackage(packageName, classic)

3.13. ábra. A snapped modul felépítése

3.8. iptables modul felépítése, feladata

Ez a modul kezeli a tűzfalszabályokat az **iptables** frontend segítségével. Funkcionálitása maga az iptables telepítése, a portok nyitása/zárása, kimenő/bemenő csomagok szűrése, továbbá a NAT előkészítése az OpenVPN szerver számára.

A modul UML diagramja a 3.14. ábrán tekinthető meg:

iptables
<ul style="list-style-type: none"> - iptablesAliases : table + errors : table
<ul style="list-style-type: none"> + resolveErrorToStr(error) + isIptablesInstalled() + installIptables() + getCurrentNetworkInterfaces() + getCurrentSSHPorts() + parseCurrentRules() + getOpenPorts(interface) + deleteOpenPortRule(interface, idx) + getClosedPorts(interface) + deleteClosePortRule(interface, idx) + closePort(interface, protocol, dport, fromIP) + openPort(interface, protocol, dport, fromIP) + listAllowedOutgoingConnections(interface) + deleteOutgoingRule(interface, idx) + allowOutgoingNewConnection(interface, protocol, dip, dport) + checkIfInboundPacketsAreBeingFilteredAlready(interface, protocol) + togOnlyAllowAcceptedPacketsInbound(toggle, interface, protocol) + checkIfOutboundPacketsAreBeingFilteredAlready(interface, protocol) + togOnlyAllowAcceptedPacketsOutbound(toggle, interface, protocol) + deleteNATRules(mainInterface, tunnelInterface, forwardTblIdx, forwardTblAllIdx, postroutingTblAllIdx) + getCurrentActiveNATForOpenVPN() + initNATForOpenVPN(mainInterface, tunnelInterface, openvpnSubnet) + loadOurRulesToIptables() + iptablesToString() + initModule()

3.14. ábra. Az iptables modul felépítése

4. fejezet

Megvalósítás

4.1. linux modul érdekességei

A tervezési fázis után hasonlóan épült fel a megvalósítás sorrendje is. A legelső közt készült el a linux modul megvalósítása, mivel több modul is függött a funkcionalitásától. Néhány érdekesebb részt szeretnék bemutatni, mint például az `exists`, `isdir` implementációját:

```
function module.exists(file)
    local ok, err, code = os.rename(file, file)
    if not ok then
        if code == 13 or code == 17 then
            return true
        end
    end
    return ok, err
end

function module.isDir(path)
    return module.exists(path.."/")
end
```

Ez a kód úgy nézi meg a fájlok létezését, hogy megpróbálja a fájlokat átnevezni saját magukra. Ekkor kétféle error lehetséges: **Permission Denied** (code 13) vagy **File exists** (code 17). Jegyzékek létezését pedig úgy ellenőrzi, hogy hozzárakja a path végéhez a / jelet, mivel így biztos, hogy jegyzékre mutat a path.

Érdekes lehet még az `execCommand` és a `execCommandWithProcRetCode` funkció implementációja is. Ezek a funkciók a program gerincét képezik, a legtöbb modul használja őket.

```
function module.execCommand(cmd)
    local handle = io.popen(cmd);
    local result = handle:read("*a");
    handle:close();

    return result;
end
```

Az `execCommand` egyszerű implementációjú, amely az `io` standard library `popen` funkcióját használja meg egy processz megnyitására. Ez egy `handle`-t ad. A `read` megvárja míg lefut a processz, majd az `*a` paraméter segítségével mindent kiolvas a pipeből. A végén lezáródik a `handle`. Windows rendszeren is tökéletesen működik.

A következő funkció az `execCommandWithProcRetCode`. Itt a forráskódból kivettük az üres sorokat a kompaktabb kód érdekében. Működése nagyban hasonlít az `execCommand`-hoz, azzal a kivétellel, hogy ez az implementáció felhasznál bizonyos Bash-script elemeket. Például az `export` funkciót az environment variablek beállítására, vagy a `$?` változót a process return kód lekérésére. Támogatja az `stderr` átirányítását is `stdout`-ba (ez a `2>&1` paraméter) cross-platform módon, a többi Bash alapú megoldás kivételével. A return code-t az utolsó sorba iratja ki, ezt olvassa be magától, és vágja ki az alap program outputjából.

```
function module.execCommandWithProcRetCode(cmd, linesReturned,
  ↪ envVariables, redirectStdErrToStdIn)
  local exportCmd = "";
  if envVariables then
    for k, v in pairs(envVariables) do
      exportCmd = exportCmd.." export "..tostring(k).."="..tostring
        ↪ (v).." ";
    end
  end
  local handle = io.popen(exportCmd..cmd..tostring(
    ↪ redirectStdErrToStdIn and " 2>&1" or "").." "; echo $?");
  handle:flush();
  local overallReturn = "";
  local lastLine = "";
  local newLineChar = "\n";
  local lineNum = 0;
  for line in handle:lines() do
    overallReturn = overallReturn .. line .. newLineChar;
    lastLine = line;

    lineNum = lineNum + 1;
  end
  handle:close();
  local retCode = tonumber(lastLine);
  if lineNum == 1 then
    overallReturn = "";
  else
    overallReturn = string.sub(overallReturn, 1, #overallReturn - #
      ↪ lastLine - #newLineChar * 2); —skip return code line
  end
  if linesReturned then
    return overallReturn, retCode;
  end
  return retCode;
end
```

4.1.1. Processz exit code felhasználása

Valamennyi funkció a programban kihasználja azt, hogy a processzek egy meghatározott visszatérési értéket (pontosabban `exit code`-t) adnak vissza lefutásuk után. Ezeknek jelentőségük van, mivel bizonyos műveleteknél más értékeket adhatnak vissza attól függően, hogy milyen funkcionalitást használunk épp. Találhatunk olyan listát az Interneten, amelyek ezeket a kódokat általánosságban írják le, hogy milyen hibához kapcsolódhatnak. Általában a processzek visszatérési értékeinek jelentése valamennyire hasonlít a legtöbb táblázatban hozzákapcsolt leíráshoz. [13]

A legfontosabb, hogy általában a legtöbb processz 0-s `exit code`-val fog visszatérni, ha sikeresen lezajlott a a processz futása, hiba nélkül. Ezt több programrész is felhasználja, például:

```
function module.checkIfUserExists(userName)
  local retCodeForId = module.execCommandWithProcRetCode("id "
    ↪ ..userName);

  return retCodeForId == 0;
end
```

Ebben az esetben a `checkIfUserExists` funkció az alapján tudja egy felhasználó létezését, hogy 0-s `exit code`-val tér-e vissza az `id` processz.

Vagy másik példaként az `mkdir` parancs 0-s (néhány táblázat szerint: `Success`) visszatérési értékkel tér vissza akkor, ha sikerült létrehozni egy új jegyzéket, vagy 1-es (néhány táblázat szerint: `Operation not permitted`) visszatérési értékkel tér vissza akkor, ha már létezik az a jegyzék, amit létre szeretnénk hozni. A programkód is ez alapján dönti el, hogy sikeres-e a jegyzék létrehozása:

```
function module.mkdir(path)
  local retCodeForMkdir = module.execCommandWithProcRetCode("mkdir "
    ↪ ..path);
  return retCodeForMkdir == 0 or retCodeForMkdir == 1; —new dir
    ↪ successfully created/already exists
end
```

Ez szintén hasonlóan működhet olyan programoknál is, amelyek nem rendszerprogramok, hanem valaki más készítette őket. Például a `certbot` is különböző processz `exit code`-val térhet vissza az adott művelet sikerességétől függően:

```
function module.trySSLCertificateCreation(method, domain, webserverType)
  ...
  local retLines, retCode = linux.execCommandWithProcRetCode("certbot
    ↪ certonly -n " ..tostring(dryRunStr).." —agree-tos —no-eff-
    ↪ email —email \"\" —webroot —webroot-path " ..tostring(
    ↪ websiteData.rootPath).." -d " ..tostring(domain), true, nil,
    ↪ true);
  ...
  local hasCertificate = retCode == 0;
  ...
end
```

4.2. Implementációs nehézség: háttérben futó processz eredményeire való várás *certbot* modulban

A program tervezése, implementálása során belefutottam egy nehézségbe, amely a Lua alapvető kialakításából fakad: a Lua alapvetően single-thread, és nem event-driven. Létezik `coroutine` beépített library, azonban az `light thread` alapú implementáció. Egyszerre csak egy coroutine futhat, és maga a coroutine kezdeményezheti azt, hogy épp suspendelve legyen (tehát visszaadja az irányítást az őt futtató kódnak, vagy más coroutinenak). Tehát a coroutine sem volt megoldás erre a nehézségre.

A *certbot* modul tervezése, implementálása közben jött elő ez a nehézség. A *HTTP-01* challenge-t könnyen lehetett implementálni úgy, hogy megvártuk az újonnan létrehozott processz futási eredményeit, mivel az támogatta a nem-interaktív módot. Azonban a *DNS-01* challenge-t nem lehet ilyen könnyen implementálni.

A probléma ott kezdődött, hogy a *certbot* maga alapvetőleg sajnos nem támogatja azt, hogy nem-interaktív módon fut ezen challenge esetében. Megoldást azonban az jelentett, hogy `-manual` módban használjuk, és megadjuk neki a preferált challenge-t (vagyis a `dns-t`), továbbá `manual-auth-hook` scriptet használunk.

Ezzel a probléma felét már sikerült orvosolnunk, azonban előjött egy újabb probléma: az `io.popen` esetén a program megvárja a processz futásának végét. Ez azonban nekünk nem megfelelő, mivel akkor teljesen befagy a program, a DNS challenge futtatásakor pedig a usernek meg kell jelenítenünk bizonyos adatokat, hogy milyen DNS rekordokat hozzanak létre a saját domainjükön és milyen értékekkel. Az `os` függvénykönyvtárban található `execute` funkció másképp működik, hátránya, hogy alapvetően ez is blocking funkció, továbbá nem is tudjuk pontosan, hogy sikerült-e a program elindítása, csak akkor, ha van `return`-ja és megvizsgáljuk a `return`-olt státusz kódot. Ezután azzal folytattam a probléma megoldását, hogy Bash script elemeket használtam fel a program futtatásához, például a `&` szimbólumot a háttérben való futáshoz, `$?` szimbólumot a státusz kód megkapásához, továbbá `#!` szimbólumot az újonnan elindított program PID-jének megkapásához. Ezt az egészet egy nagy parancsba foglaltam. A parancs több fájlt is felhasznál:

- Egy temp fájlt abból célból, hogy az újonnan létrehozott processz, továbbá a mostani processz kommunikálni tudjon egymással. Amint lefutott a *certbot*, ide kerül mentésre az exit code
- Egy másik temp fájlt abból a célból, hogy az újonnan létrehozott processz `stdout`-ját és `stderr` pipejét abba irányítsuk, így a mostani processz tudja az outputot vizsgálni
- Egy `certbot_pid.txt` fájlt, amelyből tudjuk, hogy sikeresen lefutott-e a programunk. Azt a célt is szolgálja, hogy ha esetleg megszakadt a program futása valami miatt, akkor a következő lefutáskor a `kill` parancs segítségével megszüntessük a már nem használt processt.

A kommunikáció maga a két processz között úgy történik, hogy elindul az `auth` Lua script a *certbot*-ban, amely megkapja `environmental` variablekon keresztül a *certbot*-tól az adatokat. Ezeket az adatokat a legelsőnek létrehozott temp fileba írja bele, majd ezután 1 másodpercenként folyamatosan kiolvassa a temp file tartalmát. Ez azért fontos, mert addig is blokkolja a *certbot* processzt, így nem halad tovább.

Amint a user lereadyzte a műveletet, akkor pedig ebbe a tempfájlba beleírodik a "ready" szó, majd ezután fut le csak a certbot DNS challengeje (megszakad a loop).

Lua kódban néhány részletet kiemelve így néz ki az implementáció:

```
function module.trySSLCertificateCreation(method, domain, webserverType)
    ...
    local tempFileName = os.tmpname();
    local tempFileNameForStdOut = os.tmpname();

    local certbotPIDStuff = general.readAllFileContents("certbot_pid.txt"
        ↪ );

    if certbotPIDStuff then
        os.execute("kill -9 "..tostring(certbotPIDStuff));
    end

    linux.deleteFile("certbot_pid.txt");

    local formattedCmd = "(certbot certonly -n "..tostring(dryRunStr)..
        ↪ --agree-tos --no-eff-email --email \"\" --manual --preferred-
        ↪ challenges dns --manual-auth-hook \"sh ./authenticator.sh \"
        ↪ ..tostring(tempFileName)..\"\" -d "..tostring(domain)..\" > \"
        ↪ \"..tostring(tempFileNameForStdOut)..\" 2>&1; echo $? > \"
        ↪ ..tostring(tempFileName)..\" ) & echo $! > certbot_pid.txt";
    os.execute(formattedCmd);

    if not linux.exists("certbot_pid.txt") then
        return module.EXEC_FAILED;
    end
    ...
    --fajlolvassas, cleanup
    ...
end
```

Authenticator script részlet:

```
while true do
    local fileHandle = io.open(fileName, "r");
    if fileHandle then
        local readStr = fileHandle:read("*a");
        fileHandle:close();
        if readStr:find("ready", 0, true) == 1 then
            break;
        end
    end
    sleep(1);
end
```

4.3. Konfigurációs fájlok módosításának implementációja

A konfigurációs fájlok módosításához többféle modul is implementálva lett, ezeket a Tervezés című fejezetben meg is említettem: `apache_config_handler`, `nginx_config_handler` és `OpenVPN_config_handler`. Ezek közül a modulok közül a legtöbb OO alapelveket igyekezett követni, az OpenVPN config handler kivételével.

A config parsereket és writereket hasonlóképp ugyanarra a kódbázisra építettem fel, ez az Apache szerver `envvars` fájl kezelőjének kivételével sikerült is. Mindegyik parser outputja általában két nagyobb táblából épül fel: `parsedLines` és `paramToLine`. A `parsedLines` maga az állapot tábla, abba van benne minden egyes beolvasott és értelmezett sor, benne paraméterek, kommentek vannak. A `paramToLine` tábla általában csak cache, azt mutatja meg, hogy bizonyos opciók hol vannak használva a `parsedLines` táblán belül, így nem kell átfésülni az egész `parsedLines` táblát, hanem egyszerűen elég azt felhasználni.

4.3.1. OpenVPN config parser-writer implementáció felépítése, használata

A `parsedLines` felépítése OpenVPN esetén egyszerű array, minden array elem egy tábla, amelyben a következő paraméterek szerepelhetnek:

- **params:** Ez tartalmazza az adott paramétereket, opciókat egy adott sorban. A táblában található `val` érték maga a paraméter értéke (vagy akár a config option neve), a `state` pedig kifejezi az idézőjeltípust, ha használnak (ami lehet `reading_quoted_param` " esetén, vagy `reading_squoted_param` ' esetén)
- **comment:** Ez tartalmazza az adott sorban található kommentet. Ha a `params` tábla nem létezik, akkor maga az egész sor egy komment sor, ha létezik, akkor pedig a paraméterek után van a komment elhelyezve. Üres sor, ha ez sem létezik

Kódrészlet OpenVPN server config buildelésre a programból:

```
function module.checkServerConfig(homeDir, openVPNConfigDir)
...
  local configFileContent, paramsToLines =
    ↪ config_handler.parseOpenVPNConfig(sampleConfigFileContent);
  if paramsToLines["user"] then
    local paramTbl = configFileContent[paramsToLines["user"]];
    paramTbl["params"][2].val = module["openvpn_user"];
  end
...
  configFileHandle:write(config_handler.writeOpenVPNConfig(
    ↪ configFileContent));
  configFileHandle:flush();
  configFileHandle:close();
...
end
```

Az előző oldalon látható kódrészlet egy teljesen új konfigurációs fájlt generál az OpenVPN szerver számára. Ezt úgy teszi meg, hogy egy előre megadott alap konfigurációt beolvas, majd azon módosítja a paramétereket (például az `user option-t`), utána pedig rendes konfigurációs szöveggé alakítja a módosított konfigurációt az állapot táblából.

```
# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
#
# You can uncomment this out on
# non-Windows systems.
user openvpn_serv
group openvpn_serv
```

4.1. ábra. Az átírt `user` argumentum. A példa konfigurációban `nobody` van használva

4.3.2. Apache, nginx config parser-writer implementáció felépítése és használata

Az `apache_config_handler`, és az `nginx_config_handler` modulok már OO alapelveket követnek, azonban itt is fontos a két tábla felépítése. Az OpenVPN szerverhez képest a két szerver konfigurációs fájljai bonyolultabb és kiterjedtebb syntaxú. Bár a parser-writer ugyanúgy két táblával dolgozik, megfigyelhető, hogy sokkal több paramétert tartalmaz egy adott sort leíró tábla.

A `parsedLines` felépítése Apache és nginx esetén szintén egyszerű array, minden array elem egy tábla (és egy sor), amelyben a következő paraméterek szerepelhetnek (akár több is egyszerre):

- **spacer:** Üres sort jelent. Ha ez létezik, akkor a többi lentebb sorolt property biztosan nem létezik az adott elemtáblában
- **blockStart:** Egy blokk kezdetét jelenti, a blokk azonosítóját tartalmazza. Apache esetében argumentumok is tartozhatnak hozzá (tehát ilyenkor az `args` paraméter is feldolgozódik)
- **blockEnd:** Egy blokk végét jelenti, a lezárandó blokk azonosítóját tartalmazza
- **paramName:** Adott sorban található option nevét tartalmazza (például nginx esetében `include`, Apache esetében `ServerName`), egy táblában, amely ugyan úgy épül fel, mint az `args` paraméter. Természetesen mellé feldolgozódik az `args` paraméter is.
- **comment:** Ez tartalmazza az adott sorban található kommentet. Ha az `args` tábla nem létezik, akkor maga az egész sor egy komment sor, ha létezik, akkor pedig a paraméterek után van a komment elhelyezve nginx esetén. Az Apache nem támogat kommenteket a sorok végén, az optionok és argumentumok után
- **blockDeepness:** Megadja, hogy az adott sor mennyire van eltolva indent-ügyleg
- **args:** több elem esetében is használatos paraméter (például Apache esetében `blockStart`-nál is, `paramName`-nél mindkettő implementációnál), amely egy tábla, és az adott művelethez tartalmaz paramétereket:

- **quoteStatus**: lehet **d**, ekkor duplaidézőjelben van a paraméter; lehet **s**, ekkor két sima idézőjel között van a paraméter
- **multipleLine**: Apache esetében használatos, ha egy paraméter több sort is magába ölel (Apache-ban támogatott a több sor a `\\` jelölés segítségével)
- **data**: ez maga egy string, amely a paramétert tartalmazza

A két modul használata valamelyest eltér az előző OpenVPN configot kezelő modultól. Ezek már OO alapelveket követnek, továbbá valamennyivel több funkciót is implementálnak, szélesebb körben vannak használva a programban. Például van implementálva funkció arra, hogy teljesen új datát tudjunk beszúrni bárhova, erre az OpenVPN szerver kezelő implementációja közben nem volt szükség.

A következőekben két rövid kódrészletet fogok mutatni Apache és nginx esetében is a modulok használatáról. Nginx esetében a következőképp kerül beállításra az SSL egyik beállítása:

```
function module.initSSLForWebsite(webUrl, certDetails)
...
—Redirect unencrypted connections
local blockName = 'if ($scheme != "https")';
local blockStartSchemeIdx = paramsToIdx["block:"..toString(blockName)
    ↪ ];
if not blockStartSchemeIdx then
local blockDeepness = serverNameData.blockDeepness;
configInstance:insertNewData({"comment" = " Redirect
    ↪ unencrypted connections", blockDeepness =
    ↪ serverNameData.blockDeepness}, posStart);
posStart = posStart + 1;
configInstance:insertNewData({"blockStart" = blockName, block =
    ↪ serverNameData.block, blockDeepness =
    ↪ serverNameData.blockDeepness, args = {}}, posStart);
posStart = posStart + 1;
blockDeepness = blockDeepness + 1;
configInstance:insertNewData({"paramName" =
    {
        data = 'rewrite',
    }, block = blockName, blockDeepness = blockDeepness, args =
    ↪ {{data = "^"}, {data = "https://$host$request_uri?"},
    ↪ {data = "permanent"}}
    }, posStart);
posStart = posStart + 1;
blockDeepness = blockDeepness - 1;
configInstance:insertNewData({"blockEnd" = blockName, block =
    ↪ serverNameData.block, blockDeepness =
    ↪ serverNameData.blockDeepness, args = {}}, posStart);
posStart = posStart + 1;
end
...
end
```

A kódrészlet megkeresi a 'if (\$scheme != "https")' blokkot az adott weboldal nginx-konfigurációjában. Ha nem találja meg beszúr egy kommentet, majd azután beszúr egy blokk-kezdést. A blokkba beilleszti a 'rewrite ^ https://\$host\$request_uri? permanent' parancsot, majd lezárja a blokkot a parancs után. Mindezeket indentálva teszi, így a konfiguráció jól átlátható marad. A beállítás maga azt jelenti, hogy minden HTTP-n keresztüli csatlakozást átirányít HTTPS-re.

A gyakorlatban így néz ki a beillesztett blokk, az nginx konfigurációjában:

```
# Redirect unencrypted connections
if ($scheme != "https") {
    rewrite ^ https://$host$request_uri? permanent;
}
```

4.2. ábra. A beillesztett blokk

Apache esetében is hasonlóan működik a konfiguráció módosítása. Ott azonban teljesen külön blokkot kell csinálni az SSL konfigurációnak, teljesen új beállításokkal, ezért az azt beállító kódrészlet hosszabb az nginx-implementációtól. Rövid kódrészlet az Apache SSL beállításából:

```
function module.initSSLForWebsite(webUrl, certDetails)
...
blockDeepness = blockDeepness + 1;
configInstance.insertNewData({
    blockStart = "VirtualHost",
    args = {
        {data = " *:443"}
    },
    blockDeepness = blockDeepness
});
...
configInstance.insertNewData({
    paramName = {data = "Include"},
    args = {
        {data = pathForLetsEncryptApacheConfig, quoteStatus = "d"},
    },
    blockDeepness = blockDeepness
});
...
blockDeepness = blockDeepness - 1;
configInstance.insertNewData({
    blockEnd = "VirtualHost",
    blockDeepness = blockDeepness
});
...
end
```

```
<VirtualHost *:443>
    Include "/etc/letsencrypt/options-ssl-apache.conf"
</VirtualHost>
```

4.3. ábra. A kiragadott példakód által generált konfiguráció

A példakódban a VirtualHost blokk létrehozását láthatjuk, amelybe egy Include option is beszúrásra kerül. Ezután lezárásra kerül a létrehozott VirtualHost blokk. A valóságban azonban az SSL beállítása sokkal több optiont, paramétert tartalmaz, a példaként szolgáltatott kép csak a kiragadott példakód funkcionalitását kívánja bemutatni.

4.4. Modulok használata Lua-ban

Az összes modul, továbbá a main.lua-ban található felhasználói interface is különböző modulokat használ fel. Ezeket a modulokat a **require** funkcióval lehet betölteni. A **require** funkció működése hasonló a **dofile** funkcióhoz, azonban két főbb különbség is van köztük.

Az egyik az, hogy a **require** funkció egy megadott path-on belül keresi a betöltendő fájlt, a másik pedig, a **require** funkció nem engedi ugyanazon fájl duplikált betöltését. Tehát ha már egyszer betöltöttünk egy modult, akkor nem tölti be még egyszer teljesen, hanem elcachezi egy táblában a már betöltött fájlt. Ha azonban több virtuális path-ot (például: ?/? .lua foo.lua helyett), nevet használunk ugyanazon fájlra, akkor többször is betöltődnek, mivel más lesz a fájl neve, viszont a tartalom ugyan az marad. [1]

A program kódjában a main.lua-ban van megadva, hogy milyen megadott path mentén keresi meg az adott betöltendő fájlt:

```
package.path = package.path..";modules/? .lua";
```

Ez a sor szimplán hozzáfűzi a package-k betöltésének path-jához a modules mappát.

Emiatt lehetséges az, hogy nem kell a .lua extensiont kiírnunk a **require** meghívásaink végén, továbbá, hogy tudunk modulokat betölteni például így:

```
local OpenVPNHandler = require("vpnHandler/OpenVPN");
local linux = require("linux");
```

A **require** funkció a háttérben a megadott fájl kódját futtatja le. Így működik például a **Client** class implementációja az összes modulban is, amelybe be van töltve a config handler, mivel a **Client** tábla globális változó.

Minden fájl egy adott code-scope, azonban a globális változók hozzáférhetőek azokban a modulokban is, amelyek betöltötték az adott modult. Emiatt lehetséges például a lokálisan definiált funkciók, változók szeparációja. Az adott modulok a saját maguk kódjában használják őket, viszont az őt betöltő modulok már nem tudnak hozzáférni ezekhez a funkciókhoz, változókhoz.

Modulok esetében a saját implementációmban egy lokális tábla hozódik létre (tehát alapvetőleg "private" láthatóságú), azonban mégis tudjuk használni azt. Ez azért van, mert a **require** funkció az adott kód lefutási értékét adja vissza. Így lehet akár funkcióval is visszatérni, ami konstruktorként szolgálhat (és az adja vissza a module belső táblát), vagy lehet akár a module táblával is visszatérni. Ha a module tábla nem lokális változó lenne, a modulok implementációi egymással konfliktusba kerülnének.

5. fejezet

Tesztelés

Az elkészült alkalmazás Lua-ban íródott. Legelőször root jogosultságokhoz kell jussunk, akár az `su`, vagy a `sudo` parancs használatával. A program futtatása előtt meg kell győződnünk arról, hogy legalább 5.3-as Lua verzióval rendelkezünk az adott számítógépen.

Ezt így ellenőrizhetjük:

```
# lua -v
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio
```

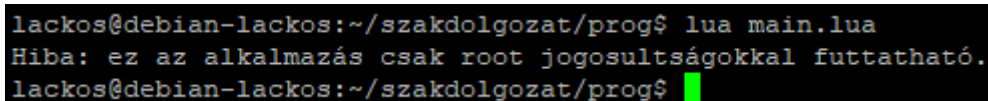
Ha esetleg nem lenne Lua feltelepítve, akkor a következő paranccsal tehetjük meg Debian/Ubuntu esetén:

```
# apt-get install lua5.4
```

Ezután nincs más dolgunk, mint lefuttatni magát az alkalmazást:

```
# lua main.lua
```

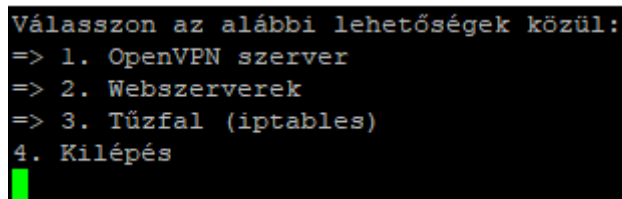
Ha nem rendelkezünk root jogosultságokkal, akkor az alkalmazás hibával tér vissza:



```
lackos@debian-lackos:~/szakdolgozat/prog$ lua main.lua
Hiba: ez az alkalmazás csak root jogosultságokkal futtatható.
lackos@debian-lackos:~/szakdolgozat/prog$
```

5.1. ábra. root jogosultságok hiányára figyelmeztető hiba

A program sikeres lefuttatása után a főmenübe érkezünk, ahol 4 lehetőségünk van:



```
Válasszon az alábbi lehetőségek közül:
=> 1. OpenVPN szerver
=> 2. Webszerverek
=> 3. Tűzfal (iptables)
4. Kilépés
```

5.2. ábra. a program főmenüje

A program összes menüjében úgy navigálhatunk, hogy beírjuk a sorszámot (például 1, vagy 1.) és ENTER-t nyomunk. A program törekszik arra, hogy minden instrukciót megadjon a felhasználó számára a használatára vonatkozóan. Ha hibába ütközik, kiírja a hiba kódját, továbbá lehetséges forrását.

5.1. OpenVPN

5.1.1. Konfigurálás, telepítés előtt

Az OpenVPN menüjébe érve ezt a képet kapjuk eleinte, ha nincs feltelepítve:

```
<=> OpenVPN szerver <=>
=> 1. Feltelepítés
2. Visszalépés
```

5.3. ábra. OpenVPN főmenü - még feltelepítés előtt

Ekkor csak szimplán kiválasztjuk az egyes menüpontot, és feltelepítődik magától az OpenVPN, ekkor frissülni fog a menü erre:

```
<=> OpenVPN szerver <=>
=> 1. Elindítás
=> 2. Szerver automatikus bekonfigurálása
3. Visszalépés
```

5.4. ábra. OpenVPN főmenü - még konfigurálás előtt

5.1.2. Konfigurálás, telepítés után

Konfiguráljuk be a szervert telepítés után. Konfigurálás után a program mappájában létrejön egy openvpn mappa, amely az easysra programot tartalmazza. Itt található meg a saját Certificate Authority-nk, amellyel a kliensek és a szerver certificatejét, private keyét kezeli a program.

Konfigurálás után több menüpont is rendelkezésünkre fog állni:

```
<=> OpenVPN szerver <=>
=> 1. Elindítás
=> 2. Szerver konfigurációjának frissítése
=> 3. Bekonfigurált kliensek listázása
=> 4. Új kliens bekonfigurálása
5. Visszalépés
```

5.5. ábra. OpenVPN főmenü - telepítés, konfigurálás után

A 3. menüpontban tudjuk megtekinteni a jelenleg már konfigurált klienseinket, további menüpontok nyílnak onnan: vissza tudjuk vonni egy kliens hozzáférését az OpenVPN szerverhez, továbbá ki tudjuk iratni a kliens konfigját. A kliens konfigja teljesen kimásolható, csak az IP-címet kell átírni benne a saját szerverünk IP-címére. Mindent tartalmaz beágyazva (a certifikatet, kulcsfájlokat, tls-crypt fájlt, satöbbit). Ha egy kliens hozzáférését visszavonjuk, akkor a hozzá generált certificatek, kulcsok revokeolásra kerülnek, és a kliens neve újra felhasználható lesz.

A 4. menüpontban tudunk új klienst létrehozni, két adatot kér be: a kliens nevét, amellyel azonosíthatjuk és egy jelszót a privát kulcsához. A jelszó megadása biztonsági okokból kötelező. A kliens nevének egyedinek kell lennie, duplikáció nem megengedett.

5.2. Webszerverek

A webszerverek menüjébe érve választhatunk **Apache** és **nginx** között is. A két webserver kezelőfelülete között nincs különbség, teljesen egy kódstruktúrára is épülnek, ezért egyben fogom bemutatni őket. Értelmszerűen ha nincs feltelepítve az adott webserver, akkor a telepítést fogja felajánlani:

```
<=> nginx szerver <=>
=> 1. Feltelepítés
2. Visszalépés
```

5.6. ábra. Webszerver főmenü - telepítés előtt

Telepítés után a következő menüpontokat láthatjuk:

```
<=> nginx szerver <=>
=> 1. Leállítás
=> 2. Jelenlegi weboldalak kezelése
=> 3. Új weboldal létrehozása
4. Visszalépés
```

5.7. ábra. Webszerver főmenü - telepítés után

A 2. menüpontban tudjuk kezelni a meglévő weboldalainkat, amint kiválasztottunk egyet, utána további menüpontokhoz jutunk:

- a legelső menüpont a weboldal törlését jelenti, ez kitörli a weboldal konfigurációját és magát a weboldalt tartalmazó `www` mappát is
- a második menüpont pedig az SSL certificatek certbot általi generálását szolgálja. Több lehetőség is van a certificatek generálására: HTTP-01 challenge, amely egy fájl automatikus elhelyezésével működik; vagy DNS-01 challenge, amelynél a saját névszerverünkönél (DNS szerverünkönél) beállítások módosítására is szükségünk van. A DNS-01-et akkor célszerű választani, ha valamiért a 80-as portot nem tudjuk használni.

A 3. menüpontban tudunk új weboldalakat létrehozni, ehhez magára csak a weboldal címére van szükség. Automatikusan létrehozza a program a weboldal konfigurációját, a weboldal `www`-dirjét, továbbá egy `index.html` fájlt:

```
lackos@debian-lackos:~/szakdolgozat/prog$ ls -l /home/nginx-www/websiteconfigs/
total 4
-rw-r--r-- 1 nginx-www nginx-www 2275 Nov 23 13:59 lszlo.ltd.conf
lackos@debian-lackos:~/szakdolgozat/prog$ ls -l /home/nginx-www/wwwdatas/
total 4
drwxr-xr-x 2 nginx-www nginx-www 4096 Nov 23 13:58 lszlo.ltd
lackos@debian-lackos:~/szakdolgozat/prog$
```

5.8. ábra. Weboldal konfigurációjának helye, weboldal `www`-dirje `nginx` esetén

5.3. iptables

5.3.1. Feltelepítés előtt

Az iptables menüpontot választva, ha nincs még feltelepítve a frontend, akkor felajánlja a program:

```
<=> iptables <=>
=> 1. Feltelepítés
2. Visszalépés
█
```

5.9. ábra. iptables feltelepítése előtt

5.3.2. Feltelepítés után

Feltelepítés után több menüpont is a szemünk elé tárul:

```
<=> iptables <=>
=> 1. Nyitott portok
=> 2. Zárt portok
=> 3. Port nyitása
=> 4. Port zárása
=> 5. Kifelé irányuló új megengedett kapcsolat létrehozása
=> 6. Engedélyezett kimenő kapcsolatok
=> 7. Interface alapú togglek
=> 8. OpenVPN NAT setup
9. Visszalépés
█
```

5.10. ábra. iptables főmenüje

A menüpontok magukért beszélnek. Minden menüpont kiválasztása után felugrik egy interface kiválasztó felület, amellyel az adott funkcionalitást leszűkíthetjük egy interfacerre.

```
Válasszon egy interfacet a továbblépés előtt:
=> 1. Összes (mindegyikre vonatkozik egyszerre)
=> 2. enp0s17
3. Visszalépés
█
```

5.11. ábra. iptables - interface választása

Ha az összes interfacet választjuk, akkor a program szigorúan azt kezeli, amikor egy szabály ténylegesen mindenre érvényes. Ez azt jelenti iptables esetén, hogy nem adunk meg interfacet hozzá (nem manuálisan adja hozzá a program minden egyes interfacehoz az adott szabályt). Ez azt jelenti, hogy az adott szabály az interfacek változása esetén is megmarad.

Érdemes ezt figyelembevenni a program használatakor, mert például a nyitott portok lehet, hogy az "all" (vagyis az összes) interfacerre vonatkozóan vannak kinyitva, és így nem mutatja ki másik interface-n való nyitott portok lekérdezésekor.

Port nyitáskor három adatra lesz szükségünk az interface kiválasztása után: a protokollra (tcp/udp/all), a port számára és a bejövő IP-címre. Ha a bejövő IP-cím üresen marad, bármely IP tud csatlakozni erre a portra.

Port záráskor is három adatra lesz szükségünk az interface kiválasztása után: a protokollra (tcp/udp/all), a port számára és a bejövő IP-címre. Ha a bejövő IP-cím üresen marad, mindegyik IP le lesz tiltva erről a portról.

Kifelé irányuló új kapcsolat engedélyezésekor is három adat lesz szükséges:

a protokoll (tcp/udp/all), a port számára és a külső IP-címre. Ha a port száma üresen marad, akkor a teljes IP-címet engedélyezi kifelé irányuló új kapcsolatként.

A "nyitott portok", "zárt portok" és "engedélyezett kimenő kapcsolatok", továbbá a "OpenVPN NAT setup" menüpont alatt tudjuk ellenőrizni a már meglévő beállításainkat. Ezekben a menüpontokban tudjuk törölni is a már létrehozott szabályokat is.

Az interface alapú togglek menüpontban van két, a működés szempontjából nagyon fontos beállítás: itt lehet beállítani, hogy minden bejövő, vagy kimenő kapcsolat szűrve legyen-e. Ha egy adott portot kinyitunk, és nem tiltjuk le az összes nem engedélyezett bejövő kapcsolatot, akkor a port nyitás szabály jelenleg épp nem lesz effektív (azonban ígyis megmarad a későbbiekre). Szintén ez vonatkozik a kimenő kapcsolatokra is: ha hozzáadunk egy engedélyezett kimenő kapcsolatot, de nem tiltjuk le a nem engedélyezett kimenő kapcsolatokat, akkor a szabály nem lesz effektív.

Az OpenVPN NAT setup menüpontban tudunk NAT-szabályokat létrehozni az OpenVPN szerverünkhöz, ha feltelepítettük és bekonfiguráltuk a program segítségével. Automatikusan felismeri, ha már van meglévő NAT szabályunk létrehozva, kilistázza azokat és törölni is tudjuk őket.

5.3.3. Minden forgalom átirányítása OpenVPN szerveren keresztül

A NAT bekonfigurálása után, ha minden forgalmat át akarunk irányítani az OpenVPN szerverünkön keresztül a kliens felől, ne felejtjük el a `net.ipv4.ip_forward` flaget beállítani a Linux szerveren. Alapértelmezésként a program csak kommentként adja hozzá azt az option-t az OpenVPN szerver konfigurációjához, amely átirányít minden forgalmat a VPN csatornára, ezt kell uncommentelnünk:

```
# nano /etc/openvpn/server/openvpn_serv.conf # ez az alapértelmezett
## elérése az OpenVPN szerver konfigurációnak
```

Maga a beállítás:

```
#push "redirect-gateway def1 bypass-dhcp"
```

A `net.ipv4.ip_forward` flaget pedig a `sysctl.conf` szerkesztésével tudjuk bekapcsolni:

```
# nano /etc/sysctl.conf
# sysctl -p
```

Ha nincs a flag a fájlban, írjuk bele: `"net.ipv4.ip_forward = 1"` ha van, akkor pedig kapcsoljuk be (írjuk át 1-re).

6. fejezet

Összefoglalás

A dolgozat célja volt, hogy egy olyan Lua nyelvben általam készített eszközt mutasson be és készítsen el, amely könnyebbé teszi GNU/Linux disztribúciókon a szerverüzemeltetést.

Úgy gondolom, hogy ezt a célt sikeresen teljesítettem. Sikerült egy olyan eszköztárat létrehoznom, amely a mindennapokban megkönnyítheti a szerverüzemeltetést azzal, hogy néhány műveletet átvesz a felhasználótól, például: webszerverek kezelését, tűzfal kezelését, OpenVPN szerver kezelését. Véleményem szerint az elkészült implementáció a gyakorlatban is megállja a helyét, éles környezetben, valós feladatok ellátására is használható lenne.

A megvalósítás során sok érdekes (és néhányszor akadályozó) dologgal találkoztam, és habár nem szakdolgozatom során találkoztam először a Lua programozási nyelvvel, úgy érzem sikerült a szakdolgozat megírása során a már meglévő tudásomat bővíteni.

Igyekeztem a program implementálása során minőségi és könnyen átlátható kódot létrehozni, amely későbbi továbbfejlesztési lehetőséget hordozhat magában.

A tervekhez képest a funkcionalitás kissé elmaradt, azonban úgy gondolom, hogy így is jól használható az elkészült alkalmazás. Lehetne még tovább fejleszteni a programot, például:

- Command line argumentumok létrehozásával, hogy támogassa az automatizációt. Ennek megfelelően különböző process exit codekra lenne szükség
- A kezelőfelülethez lehetne Terminal UI támogatást létrehozni, hogy méginteraktívabb legyen a program, ezzel például támogathatná az egérekattintásokat is, és az egyszerűbb navigációt
- További szerverek támogatása
- Automatikus logértelmezés, olvasás (például connection log, iptables block log, access log)
- OpenVPN webadmin felület implementálásával
- Teljeskörű webadmin felület implementálásával
- Több Linux disztribúció támogatásával (különböző csomagkezelő programok támogatása)

-
- Esetleges Windows OS támogatással. Habár maga az implementáció egyelőre csak Linux disztribúciókhoz készült el, a programozási nyelvnek köszönhetően az elkészült szoftver platformfüggetlen lehetne. A jövőben akár ugyanebben a kódbázisban teljesen más platformok támogatását könnyedén lehetővé lehetne tenni.

Úgy gondolom, a felsorolt továbbfejlesztési lehetőségekkel való kibővítéssel akár a jelenleg elérhető népszerű eszközök alternatívája is lehet.

Irodalomjegyzék

- [1] 8.1 – the require function. <https://www.lua.org/pil/8.1.html>. megtekintve: 2023. november 23.
- [2] Access server features overview. <https://openvpn.net/vpn-server-resources/openvpn-access-server-features-overview/>. megtekintve: 2023. november 6.
- [3] Apache http server version 2.4 - configuration files. <https://httpd.apache.org/docs/2.4/configuring.html>. megtekintve: 2023. november 6.
- [4] Certbot instructions for debian 10. <https://certbot.eff.org/instructions?ws=apache&os=debianbuster>. megtekintve: 2023. november 6.
- [5] chroot man page. <https://man7.org/linux/man-pages/man2/chroot.2.html>. megtekintve: 2023. november 6.
- [6] firewalld man page. <https://firewalld.org/documentation/man-pages/firewall-cmd.html>. megtekintve: 2023. november 6.
- [7] i-mscp website. <https://i-mscp.net/>. megtekintve: 2023. november 6.
- [8] iptables man page - debian. <https://manpages.debian.org/unstable/iptables/iptables.8.en.html>. megtekintve: 2023. november 6.
- [9] Ispconfig blog. <https://www.ispconfig.org/blog/>. megtekintve: 2023. november 6.
- [10] Ispconfig documentation. <https://www.ispconfig.org/documentation/>. megtekintve: 2023. november 6.
- [11] Ispconfig website. <https://www.ispconfig.org/>. megtekintve: 2023. november 6.
- [12] ispcp weboldal. <http://isp-control.net/>. megtekintve: 2023. november 6.
- [13] List of common exit codes for gnu/linux. <https://slg.ddnss.de/list-of-common-exit-codes-for-gnu-linux/>. megtekintve: 2023. november 6.
- [14] Lua about. <https://www.lua.org/about.html>. megtekintve: 2023. május 16.
- [15] Lua metatables and metamethods (2.4). <https://www.lua.org/manual/5.3/manual.html>. megtekintve: 2023. november 5.
- [16] Lua oop. <https://www.lua.org/pil/16.html>. megtekintve: 2023. november 5.

-
- [17] LuaJit. <https://luajit.org/luajit.html>. megtekintve: 2023. május 16.
 - [18] myvestacp website. <https://www.myvestacp.com/#details>. megtekintve: 2023. november 6.
 - [19] netfilter news. <https://netfilter.org/news.html>. megtekintve: 2023. november 6.
 - [20] nginx config parser. https://github.com/nginx/nginx/blob/master/src/core/nginx_conf_file.c. megtekintve: 2023. november 6.
 - [21] Object oriented programming. <http://lua-users.org/wiki/ObjectOrientedProgramming>. megtekintve: 2023. május 16.
 - [22] Openvpn options parser. <https://github.com/OpenVPN/openvpn/blob/master/src/openvpn/options.c>. megtekintve: 2023. november 6.
 - [23] Reference manual for openvpn 2.4. <https://openvpn.net/community-resources/reference-manual-for-openvpn-2-4/>. megtekintve: 2023. november 6.
 - [24] ufw ubuntu page. <https://help.ubuntu.com/community/UFW>. megtekintve: 2023. november 6.
 - [25] Vestacp docs. <https://vestacp.com/docs/>. megtekintve: 2023. november 6.
 - [26] Virtual hosting control system. https://de.wikipedia.org/wiki/Virtual_Hosting_Control_System. megtekintve: 2023. november 6.
 - [27] What comes after 'iptables'? its successor, of course: nftables'. <https://developers.redhat.com/blog/2016/10/28/what-comes-after-iptables-its-successor-of-course-nftables#>. megtekintve: 2023. november 6.
 - [28] What's the difference between snap and apt on linux? <https://raspberrytips.com/snap-vs-apt/>. megtekintve: 2023. november 6.
 - [29] Josh. What is openvpn? how it works and when to use it in 2022. <https://www.allthingssecured.com/vpn/faq/what-is-openvpn/>. megtekintve: 2023. május 15.
 - [30] Maile McCann and Rob Watts. What is a vpn used for? 9 vpn uses in 2023. <https://www.forbes.com/advisor/business/software/why-use-a-vpn/>. megtekintve: 2023. május 15.
 - [31] W3Techs. Usage statistics of web servers. https://w3techs.com/technologies/overview/web_server. megtekintve: 2023. május 15.

Adathordozó használati útmutató

Az adathordozó felépítése egyszerű. Tartalmazza a következő jegyzékeket és fájlokat:

- **latex** jegyzék: Ebben található maga a dolgozat \LaTeX fájljai, benne a fejezetekkel és a mellékletekkel (képekkel)
- **prog** jegyzék: Ez tartalmazza magát a programot. A prog jegyzék felépítése:
 - **openvpn** jegyzék: Ebben található a saját Certificate Authoritynk (EasyRSA), ami mind a kliensek, és a szerver certificate & private key kezeléséért felelős. Runtime hozódik létre, a program használata során
 - **easy_rsa_install_cache** jegyzék: Runtime hozódik létre, az EasyRSA archívuma található benne
 - **modules** jegyzék: Itt található maga a program moduljai .lua fájlokban, továbbá ezeken belül is vannak modulokat tartalmazó aljegyzékek
 - **authenticator.lua** fájl: Certbot használatához szükséges fájl, a DNS-01 challenge implementálásáért felelős
 - **authenticator.sh** fájl: Egyszerű shell script, amely meghívja az `authenticator.lua` fájlt a Lua interpreterével
 - **main.lua** fájl: Ez a fájl tartalmazza maga a program kezelőfelületét, továbbá a program implementálása során felmerülő tesztelési kódokat
- **dolgozat.pdf** fájl: Az elkészült dolgozat PDF formátumban