ДИСЦИПЛИНА	Инструменты девопс
ИНСТИТУТ	Институт перспективных технологий и индустриального программирования
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Практическая работа
ПРЕПОДАВАТЕЛЬ	Гиматдинов Дамир Маратович
СЕМЕСТР	5 семестр, 2025-2026

Практическая работа № 4

Подготовьте отчет с описанием выполненных шагов в Microsoft Office и загрузите для проверки. Шаги сопровождайте скриншотами. Отчет должен содержать ссылку на хостинг с проектом (github и т.д.).

Вступление

В этом руководстве вы познакомитесь с процессом подключения сервисов, написанных на Python и Go с использованием фреймворка gRPC, взаимной аутентификации TLS, nginx.

gRPC - это платформа удаленного вызова процедур (RPC), которая очень хорошо работает в сценариях межсервисного взаимодействия. Она использует Protocol buffers как в качестве языка определения интерфейса (IDL), так и в качестве формата обмена сообщениями. gRPC использует HTTP/2 и поддерживает протокол безопасности транспортного уровня (TLS), и он может работать без TLS - в принципе, это то, что показывает нам большинство руководств. Таким образом, связь осуществляется по протоколу h2c, по сути, в виде обычного текста HTTP/2 без шифрования TLS. Однако, когда связь осуществляется по сети общего пользования, TLS является обязательным требованием. А принимая во внимание современные угрозы безопасности, протокол TLS следует рассматривать даже для подключений к частным сетям.

1. Настройка окружения

Работа выполняется на ОС Линукс. Рекомендуется использовать дистрибутив Ubuntu 24.04.

1.1 Создание структуры проекта

Создание рабочей директории проекта:

```
mkdir python_go_grpc && cd python_go_grpc
```

Создание структуры проекта:

```
mkdir certs && mkdir client && mkdir proto && mkdir server
```

1.2 Установка Go

```
sudo apt update

sudo apt install golang
```

```
go version
```

1.3 Настройка nginx

```
sudo apt install nginx

systemctl status nginx
```

1.4 Установка Protobuf:

```
sudo apt install -y protobuf-compiler

protoc –version
```

1.5 Уставнока cfssl

```
sudo apt -y install golang-cfssl
```

2. Настройка TLS сертификатов

Чтобы начать с TLS, вам понадобятся сертификаты как для клиента, так и для сервера. Для создания самозаверяющих сертификатов я предлагаю набор инструментов PKI от CloudFlare, CFSSL.

Во-первых, вам необходимо создать Центр сертификации (CA), который будет использоваться для создания сертификатов TLS для сервера и клиента. Этот сертификат CA также используется для проверки подлинности сертификатов других сторон при установлении TLS-соединения.

Создание файла, генерирующего сертификаты:

```
touch certs.sh

nano certs.sh
```

Необходимо добавить в файл certs.sh следующее содержимое:

```
  cd certs && \
      cfssl gencert -initca ca-csr.json | cfssljson
-bare ca - && \
      cfssl gencert \
```

```
            -ca=ca.pem \
            -ca-key=ca-key.pem \
            -config=ca-config.json \
            -hostname=127.0.0.1 \
            -profile=server server-csr.json | cfssljson
-bare server && \
        cfssl gencert \
            -ca=ca.pem \
            -ca-key=ca-key.pem \
            -config=ca-config.json \
            -profile=client client-csr.json | cfssljson
-bare client
```

Перейти в директорию certs:

```
cd certs
```

Создать файл ca-config.json:

```
nano ca-config.json
```

Добавить в файл ca-config.json следующее содержимое:

```
{
    "signing": {
        "default": {
            "expiry": "168h"
        },
        "profiles": {
            "server": {
                "expiry": "8760h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth"
                ]
            },
            "client": {
                "expiry": "8760h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "client auth"
                ]
            }
        }
    }
```

```
}
```

Создать файл ca-csr.json:

```
nano ca-csr.json
```

Добавить в файл ca-csr.json следующее содержимое:

```
{
    "CN": "CA",
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "US",
            "ST": "CA",
            "L": "San Francisco"
        }
    ]
}
```

Создать файл client-csr.json:

```
nano client-csr.json
```

Добавить в файл client-csr.json следующее содержимое:

```
{
    "CN": "client",
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "US",
            "ST": "CA",
            "L": "San Francisco"
        }
    ]
}
```

Создать файл server-csr.json:

```
nano server-csr.json
```

Добавить в файл server-csr.json следующее содержимое:

```
{
    "CN": "server",
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "US",
            "ST": "CA",
            "L": "San Francisco"
        }
    ]
}
```

Подняться в директорию проекта:

```
cd
```

Запустить скрипт генерации сертификатов:

```
sh certs.sh
```

3. Настройка Protobuf

Создать файл api.proto в директории proto:

```
nano proto/api.proto
```

Добавить в файл proto/api.proto следующее содержимое:

```
syntax = "proto3";

option go_package = "server/api";

package api;

message RollDieRequest {}

message RollDieResponse {
  int32 value = 1;
}

service DiceService {
  rpc RollDie (RollDieRequest) returns (RollDieResponse) {}
}
```

4. Создать виртуальное окружение в текущей директории

```
python3 -m venv venv
```

Активировать виртуальное окружение:

```
source venv/bin/activate
```

Установить зависимости python:

```
pip install grpcio grpcio-tools
```

```
pip install django
```

5. Генерация кода Protocol buffer для клиента и сервера

Создать директорию api в директории client:

```
mkdir client/api
```

Следующим шагом является генерация кода для каждого языка из определения proto с помощью компилятора protobuf - Proc. Кроме того, для каждого языка требуется свой собственный набор зависимостей.

Установите необходимые пакеты и соберите прото-файл compile для Python:

```
python -m grpc_tools.protoc -I proto --proto_path=proto --python_out=client/api --grpc_python_out=client/api proto/api.proto
```

Скомпилированные файлы расположены в каталоге client/api. По какой-то причине компилятор protocol для Python использует абсолютный импорт в сгенерированном коде, и это должно быть исправлено:

```
cd client/api && cat api_pb2_grpc.py | sed -E 's/^(import api_pb2.*)/from client.api \\1/g' > api_pb2_grpc.tmp && mv -f api_pb2_grpc.tmp api_pb2_grpc.py
```

Установите необходимые модули и создайте файл proto для Go. protoc-gen-go и protoc-gen-go-grpc по умолчанию установлены в каталоге GOBIN.

Вы можете переопределить GOBIN и указать его в каталоге bin virtualenv — это упростит последующую очистку.

Необходимо находиться в коренвой директории проекта.

Зависимости го добавить в файл bash_profile:

```
nano ~/.bash_profile
```

Добавить в файл следующее содержание:

```
export GO_PATH=~/go
export PATH=$PATH:/$GO_PATH/bin
```

Установить зависимости для Go:

```
go install google.golang.org/protobuf/cmd/protoc-gen-go@latest

go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@latest

protoc -I. --go_out=. --go-grpc_out=. --proto_path=<полный путь до директории proto> api.proto
```

6. Настройка клиента

Вернитесь в директорию проекта:

```
cd ../../
```

Создайте django проект client в текущей директории:

```
django-admin startproject client .
```

Создайте файл client/api/client.py:

```
nano client/api/client.py
```

Наполните файл client/api/client.py следующем содержимым:

```
import grpc

from . import api_pb2, api_pb2_grpc

class Certs:
  root = None
  cert = None
```

```
    key = None

    def __init__(self, root, cert, key):
      self.root = open(root, 'rb').read()
      self.cert = open(cert, 'rb').read()
      self.key = open(key, 'rb').read()

  class Client:
    rpc = None

    def __init__(self, addr: str, crt: Certs):
          creds  =  grpc.ssl_channel_credentials(crt.root,  crt.key,
crt.cert)
      channel = grpc.secure_channel(addr, creds)
      self.rpc = api_pb2_grpc.DiceServiceStub(channel)

    def roll_die(self) -> int:
      return self.rpc.RollDie(api_pb2.RollDieRequest()).value
```

Поменяйте содержимое файла client/urls.py на следующее:

```
from django.contrib import admin
from django.urls import path

from . import views

urlpatterns = [
    #path('admin/', admin.site.urls),
    path('', views.index, name="index"),
    path('grpc', views.index, name="grpc"),
    path('nginx', views.index, name="nginx")
]
```

Создайте файл client/views.py:

```
nano client/views.py
```

Наполните файл client/views.py следующем содержимым:

```
import logging

from django.shortcuts import render
from django.http import HttpResponse

#from . import api
from .api.client import Certs
from .api.client import Client

logger = logging.getLogger(__name__)


# 🎲 Game Die
# ⚀ ⚁ ⚂ ⚃ ⚄ ⚅ can be shown in text using the range U+2680 to U+2685
ICONS = ["?", "⚀", "⚁", "⚂", "⚃", "⚄", "⚅"]
```

```
   def index(request):
     grpc_addr = "127.0.0.1:8443"
     nginx_addr = "127.0.0.1:9448"

     value_a, value_b = 0, 0
     if request.method == "POST":
        crt = Certs('certs/ca.pem', 'certs/client.pem', 'certs/client-
key.pem')
             #crt   =   api.Certs('certs/ca.pem',   'certs/client.pem',
'certs/client-key.pem')
       try:
         #value_a = api.client(grpc_addr, crt).roll_die()
         value_a = Client(grpc_addr, crt).roll_die()
       except Exception as e:
         logger.exception(e)

       try:
         #value_b = api.Client(nginx_addr, crt).roll_die()
         value_b = Client(nginx_addr, crt).roll_die()
       except Exception as e:
         logger.exception(e)

     ctx = {
       "icon_a": ICONS[value_a],
       "icon_b": ICONS[value_b],
       "sign": "=" if value_a == value_b else "<" if value_a < value_b
else ">",
     }
     return render(request, "index.html", context=ctx)


   def grpc(request):
     grpc_addr = "127.0.0.1:8443"

          #crt    =    api.Certs('certs/ca.pem',    'certs/client.pem',
'certs/client-key.pem')
      crt  =  Certs('certs/ca.pem',  'certs/client.pem',  'certs/client-
key.pem')
     try:
       #value = api.Client(grpc_addr, crt).roll_die()
       value = Client(grpc_addr, crt).roll_die()
     except Exception as e:
       logger.exception(e)

     return HttpResponse('Value: ' + ICONS[value])


   def nginx(request):
     nginx_addr = "127.0.0.1:9448"

          crt    =    api.Certs('certs/ca.pem',    'certs/client.pem',
'certs/client-key.pem')
     try:
       #value = api.Client(nginx_addr, crt).roll_die()
       value = Client(nginx_addr, crt).roll_die()
     except Exception as e:
       logger.exception(e)

     return HttpResponse('Value: ' + ICONS[value])
```

Создайте файл index.html в директории templates в папке venv виртуального окружения (примерный путь: venv/lib/python3.12/site-packages/django/contrib/admin/templates/):

```
nano
venv/lib/python3.12/site-packages/django/contrib/admin/
templates/index.html
```

Добавьте в файл следующее содержимое

```
<!doctype html>

<html lang="en-us" dir="ltr">
  <head>
    <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1">
    <style>
      html {
        line-height: 1.15;
      }
      body {
        max-width: 960px;
        color: #525252;
        font-family: sans-serif;
        margin: 0 auto;
        font-size: x-large;
      }
      main {
        text-align: center;
      }
    </style>
    <script>
      // removes form resubmit popup on F5
      window.onload = function() {
          history.replaceState("", "", "/");
      }
      // favicon
       document.head.appendChild(Object.assign(document.createElemen
t("link"),     {rel:     "icon",     href:     "data:image/svg+xml,<svg
xmlns='http://www.w3.org/2000/svg'   viewBox='0   0   100   100'><text
y='.9em' font-size='90'>🎲</text></svg>"}));
    </script>
  </head>
  <body>
    <main>
      <p>{{ icon_a }} {{ sign }} {{ icon_b }}</p>
      <form method="POST">
        {% csrf_token %}
        <button type="submit">Roll it!</button>
      </form>
    </main>
  </body>
</html>
```

Проверьте отсутствие ошибок:

```
python manage.py check
```

Проверьте работу клиента:

```
python manage.py runserver
```

7. Настройка сервера Go

Создайте вспомогательный файл server/api/server.go:

```
nano server/api/server.go
```

Добавьте в файл следующее содержимое:

```
package api

import (
        "context"
        "math/rand"
        "time"
)

// Number of dots on a die
const Dots = 6

type Server struct {
        UnimplementedDiceServiceServer
        rnd *rand.Rand
}

func NewServer() *Server {
        return &Server{
                                                                rnd:
rand.New(rand.NewSource(time.Now().UnixNano())),
        }
}

func (s *Server) RollDie(ctx context.Context, req *RollDieRequest)
(*RollDieResponse, error) {
        // rand.Intn returns a value in [0, Dots) interval
        value := s.rnd.Intn(Dots) + 1
        return &RollDieResponse{Value: int32(value)}, nil
}
```

Создать файл сервера Go:

```
nano server/server.go
```

Добавить в файл следующее содержимое:

```
package main
```

```
    import (
            "context"
            "crypto/tls"
            "crypto/x509"
            "fmt"
            "io/ioutil"
            "log"
            "net"
            "os"
            "os/signal"
            "syscall"
            "time"

            "github.com/oklog/run"
            "google.golang.org/grpc"
            "google.golang.org/grpc/codes"
            "google.golang.org/grpc/credentials"
            "google.golang.org/grpc/metadata"
            "google.golang.org/grpc/peer"
            "google.golang.org/grpc/status"

            "python_go_grpc/server/api"

    )

    func   loggingInterceptor(ctx   context.Context,   req   interface{},   info
*grpc.UnaryServerInfo, handler grpc.UnaryHandler) (interface{}, error) {
            ts := time.Now()

            peer, ok := peer.FromContext(ctx)
            if !ok {
                    return nil, status.Errorf(codes.InvalidArgument, "missing
peer")
            }
            md, ok := metadata.FromIncomingContext(ctx)
            if !ok {
                    return nil, status.Errorf(codes.InvalidArgument, "missing
metadata")
            }

            res, err := handler(ctx, req)

             log.Printf("server=%q ip=%q method=%q status=%s duration=%s user-
agent=%q",
                    md[":authority"][0],
                    peer.Addr.String(),
                    info.FullMethod,
                    status.FromContextError(err).Code(),
                    time.Since(ts),
                    md["user-agent"][0],
            )
            return res, err
    }

    func main() {

            secureAddress := "127.0.0.1:8443"
            insecureAddress := "127.0.0.1:50051"

            var g run.Group

            // Handling signals
            term := make(chan os.Signal)
            signal.Notify(term, syscall.SIGINT, syscall.SIGTERM)
            cancel := make(chan struct{})
```

```
	g.Add(
		func() error {
			select {
			case s := <-term:
				log.Printf("Received signal=%q, exiting...", s)
			case <-cancel:
				// pass
			}
			return nil
		},
		func(err error) {
			close(cancel)
		},
	)

	serverCert, err := tls.LoadX509KeyPair("certs/server.pem", "certs/server-key.pem")
	if err != nil {
		log.Printf("failed to load server cert/key: %s", err)
		os.Exit(1)
	}
	caCert, err := ioutil.ReadFile("certs/ca.pem")
	if err != nil {
		log.Printf("failed to load CA cert: %s", err)
		os.Exit(1)
	}
	caCertPool := x509.NewCertPool()
	caCertPool.AppendCertsFromPEM(caCert)
	creds := credentials.NewTLS(&tls.Config{
		Certificates: []tls.Certificate{serverCert},
		ClientCAs:    caCertPool,
		ClientAuth:   tls.RequireAndVerifyClientCert,
	})

	secureSrv := grpc.NewServer(grpc.Creds(creds), grpc.UnaryInterceptor(loggingInterceptor))
	g.Add(
		func() error {
			log.Printf("Starting gRPC server, address=%q", secureAddress)
			lis, err := net.Listen("tcp", secureAddress)
			if err != nil {
				return fmt.Errorf("failed to listen: %w", err)
			}

			api.RegisterDiceServiceServer(secureSrv, api.NewServer())
			if err := secureSrv.Serve(lis); err != nil {
				return fmt.Errorf("failed to serve: %w", err)
			}
			return nil
		},
		func(err error) {
			secureSrv.Stop()
		},
	)

	insecureSrv := grpc.NewServer(grpc.UnaryInterceptor(loggingInterceptor))
	g.Add(
		func() error {
```

```
                                        log.Printf("Starting gRPC server (h2c),
address=%q", insecureAddress)
                                lis, err := net.Listen("tcp", insecureAddress)
                                if err != nil {
                                        return fmt.Errorf("failed to listen: %w",
err)
                                }

                                        api.RegisterDiceServiceServer(insecureSrv,
api.NewServer())
                                if err := insecureSrv.Serve(lis); err != nil {
                                        return fmt.Errorf("failed to serve: %w",
err)
                                }
                                return nil
                        },
                        func(err error) {
                                insecureSrv.Stop()
                        },
                )

                if err := g.Run(); err != nil {
                        log.Print(err)
                }
        }
```

Создать файл go.mod:

```
nano go.mod
```

Добавить в файл следующее содержимое:

```
module python_go_grpc

go 1.22

require (
        github.com/oklog/run v1.1.0
        google.golang.org/grpc v1.70.0
        google.golang.org/protobuf v1.36.5
)

require (
        github.com/cilium/ebpf v0.11.0 // indirect
        github.com/cosiner/argv v0.1.0 // indirect
        github.com/cpuguy83/go-md2man/v2 v2.0.6 // indirect
                github.com/derekparker/trie  v0.0.0-20230829180723-
39f4de51ef7d // indirect
        github.com/go-delve/delve v1.24.1 // indirect
                github.com/go-delve/liner  v1.2.3-0.20231231155935-
4726ab1d7f62 // indirect
        github.com/golang/protobuf v1.5.4 // indirect
        github.com/google/go-dap v0.12.0 // indirect
        github.com/hashicorp/golang-lru v1.0.2 // indirect
        github.com/inconshreveable/mousetrap v1.1.0 // indirect
        github.com/mattn/go-colorable v0.1.13 // indirect
        github.com/mattn/go-isatty v0.0.20 // indirect
        github.com/mattn/go-runewidth v0.0.13 // indirect
        github.com/rivo/uniseg v0.2.0 // indirect
        github.com/russross/blackfriday/v2 v2.1.0 // indirect
```

```
        github.com/spf13/cobra v1.9.1 // indirect
        github.com/spf13/pflag v1.0.6 // indirect
             go.starlark.net  v0.0.0-20231101134539-556fd59b42f6  //
indirect
        golang.org/x/arch v0.11.0 // indirect
            golang.org/x/exp  v0.0.0-20230224173230-c95f2b4c22f2  //
indirect
        golang.org/x/net v0.32.0 // indirect
        golang.org/x/sync v0.10.0 // indirect
        golang.org/x/sys v0.28.0 // indirect
                    golang.org/x/telemetry   v0.0.0-20241106142447-
58a1122356f5 // indirect
        golang.org/x/text v0.21.0 // indirect
                 google.golang.org/genproto  v0.0.0-20200526211855-
cb27e3aa2013 // indirect
        gopkg.in/yaml.v3 v3.0.1 // indirect
)
```

Установить дополнительные зависимости Go:

```
go mod download google.golang.org/grpc

go mod download github.com/golang/protobuf

go mod tidy
```

Проверить работу сервера Go:

```
go run server/server.go
```

9. **Настройка nginx**

Создать файл nginx.conf:

```
nano nginx.conf
```

Добавить в файл следующее содержимое:

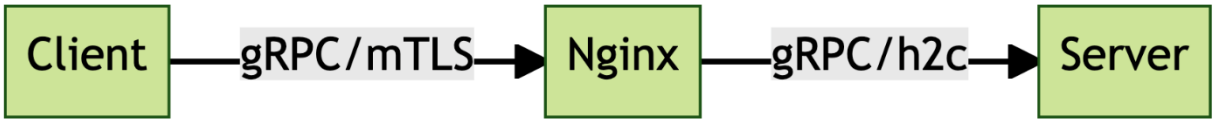
```
events {
    worker_connections  1024;
}

# Do not use it in production!
daemon on;
master_process on;

http {

    upstream grpcservers {
        server 127.0.0.1:50051;
    }
```

```
        server {
            listen 127.0.0.1:9445 ssl http2;
            error_log /dev/stdout;
            access_log /dev/stdout;

            # Server's tls config
            ssl_certificate     certs/server.pem;
            ssl_certificate_key certs/server-key.pem;

            # mTLS part
            ssl_client_certificate certs/ca.pem;
            ssl_verify_client      on;

            location / {
                grpc_pass grpc://grpcservers;
            }
            location /nginx_status {
                    stub_status on;
                    allow 127.0.0.1;
                    deny all;
            }
        }
    }
```

Проверить работу nginx:

```
sudo nginx -p $(pwd) -c nginx.conf
```

10. Запуск клиента, сервера, nginx



Проверить клиент:

```
python manage.py check
```

Запустить клиент:

```
python manage.py runserver
```

Чтобы оставить сервер нажмите Ctrl + C.

Чтобы освободить занимаемые клиентом порты, найти процессы запущенные python:

```
ps
```

Чтобы остановить процесс запустите команду:

```
sudo kill -15 <PID процесса>
```

Запустите сервер вместе с клиентом:

```
go run server/server.go & python manage.py runserver
```

Перейдите по адресу клиента в браузере и проверьте его работу.

Чтобы освободить занимаемые клиентом и сервером порты, найти и остановить процессы запущенные python и go.

Запустите сервер, клиент и nginx:

```
python manage.py runserver & go run server/server.go
& nginx -p $(pwd) -c nginx.conf
```

Перейдите в браузере на адрес клиента и проверьте работу.

Чтобы освободить занимаемые nginx порта, необходимо остановить службу, затем запустить:

```
sudo systemctl stop nginx
```

```
sudo systemctl start nginx
```

Запустите сразу несколько клиентов на нескольких разных портах вместе с сервером и nginx. Для этого команде `runserver` передайте порт через опцию `-p.`