**ВВЕДЕНИЕ**

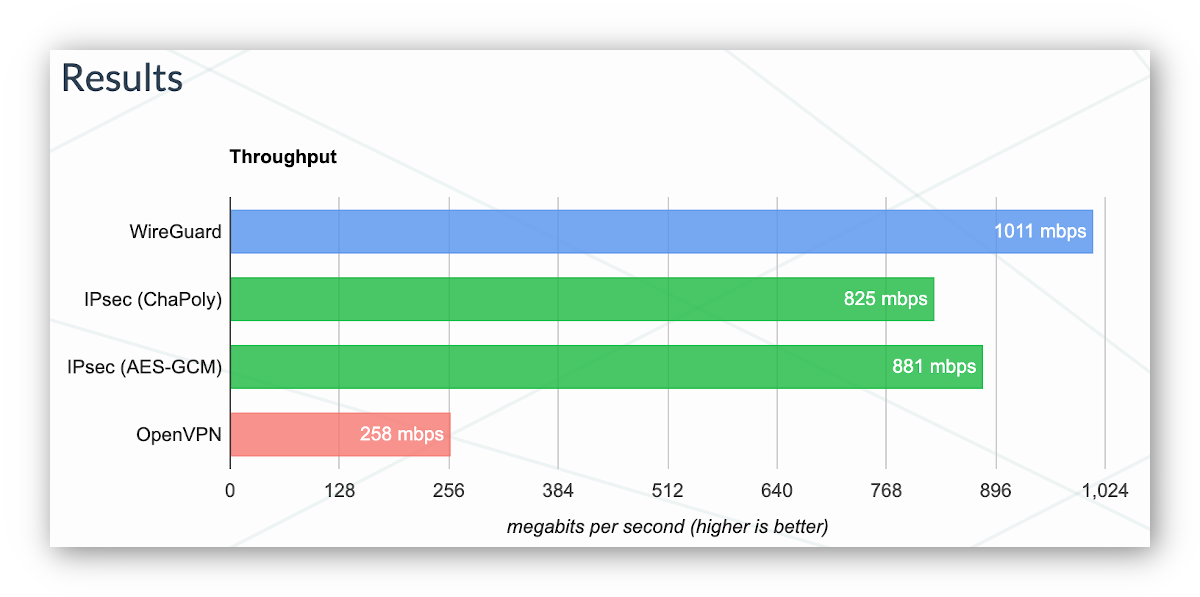
В современном мире российские пользователи сети «интернет» сталкиваются со значительными ограничениями – блокировками различных ресурсов. Причем исходить они могут как от властей РФ, так и от самих зарубежных сервисов («между молотом и наковальней»). В таких условиях актуальной становится разработка VPN-сервиса, обеспечивающего туннелирование траффика до серверов в иных юрисдикциях, обеспечивающих отсутствие ограничений.

Разрабатываемое приложение лицензируется согласно MPLv2[1] – Mozilla Public License 2.0. В отличии от GPLv3[2] - General Public License 3.0, которая, в случае заимствования кода, требует опубликовать весь проект под ней же, данная лицензия требует сохранить лицензию только для заимствованного кода, а не для всего проекта. В целом, MPLv2 выбрана как одна из самых свободных лицензий, которая, тем нее, запрещает использовать опубликованный под ней код в закрытом исходном коде.

В настоящем проекте выделяется несколько логических частей, которые можно выделить как по технологиям, так и по «бизнес-задачам». На текущем этапе они включают: VPN-сервера (т.н. ноды), главный сервер, сайт с информацией о проекте, клиентские приложения. Предполагаемой к использованию процедурой установления VPN-соединения предлагается следующее: клиент, после авторизации и получения JWT-токенов, регистрирует свой девайс на главном сервере, сообщая ему свой главный ключ, а также запрашивает список доступных нод; клиент запрашивает у главного сервера подключение к выбранной ноде; главный сервер, сначала удостоверившись, что клиент отключен от предыдущей ноды, проведя валидацию: проверив лимиты, соответствие уровня доступа пользователя запрашиваемой ноде и удостоверившись в доступности последней, передаёт ей публичный ключ пользователя; нода, после валидации ключа (с целью защиты от инъекций), выполняет его добавление, выделяет ему адрес в приватной сети, после чего возвращает главному серверу свой публичный ключ и выделенный адрес; главный сервер добавляет к полученному от ноды ответу IP-адрес ноды и порт, прослушиваемый сервером Wireguard, после чего возвращает всю полученную модель данных клиенту; клиент устанавливает Wireguard-туннель.

Предлагается назвать приложение Vdb (Vpn-Developed-Badly), а в качестве логотипа использовать латинскую букву V на синем фоне.

**ПРОЕКТИРОВАНИЕ VPN-СЕРВЕРА**

 В первую очередь поясним упомянутое во введении превосходство протокола Wireguard над остальными. На изображении 1 представлено сравнение пропускной способности наиболее распространенных протоколов. Дополнительно замечу, что Wireguard использует в качестве протокола шифрования ChaCha20-Poly1305, обеспечивающий экономии заряда батареи на мобильных устройствах, ибо потребляет значительно меньше ресурсов на архитектуре ARM, хотя, как и AES, не имеет на ней аппаратного ускорения.

Изображение 1

Данная глава описывает прототипирование базовой единицы разрабатываемого проекта. Сам VPN сервер представляет из себя Wireguard сервис и ASP-приложение, которое, по запросу, выполняет с ним определенные манипуляции – добавляет и удаляет пиры, возвращает список клиентов и удаляет неактивных. Данные приложения запускаются в одном контейнере, основывающемся на образе Alpine Linux – полноценной операционной системе архитектуры MUSL, занимающей в готовом виде 5 (пять!) МБ, в отличии от, например, Ubuntu – занимающей «из коробки» 190 МБ. Изучение документации[5][6]7] docker показало, что разумным вариантом является запуск двух приложений в одном контейнере, нежели их разделение, т.к. в этом случае возникают проблемы с получением доступа из одного контейнера в другой, а также нарушается в целом идея docker-контейнеризации, которая подразумевает взаимодействие с контейнерами по веб-протоколам (в т.ч. HTTP, RPC). Проблема состоит в том, что Wireguard не отслеживает обновление файла конфигурации. Таким образом, если ASP и может изменить файл конфигурации Wireguard (путем, например, использования docker volumes), то это не привет к обнаружению и применению этого изменения. Наивным решением могла бы быть регулярная перезагрузка файла конфигурации, например с применением cronjob, однако это однозначно более медленно, нежели стандартный способ добавления пиров «на лету», т.е. через выполнение команды ‘wg set wg0 peer KEY allowed-ips IP’[2].

Функционал Wireguard является стандартным и состоит в прослушивании порта 51820 (стандартный порт wg), маскировании трафика (англ. masquerade), перенаправлении на внешний адрес. Пиры должны добавляться и удаляться по CL-команде от ASP-приложения.

Функционал ASP-приложения состоит в принятии запросов от удаленного сервера, проверке действительности API-ключа, выполнении действия. Само приложение, как предполагается, должно хранить ключи в виде их SHA512-хеша, не храня оригиналы, т.к. это создает концептуальную уязвимость и не соответствует принципу secure-by-design.

В контейнер также устанавливается NGINX, выполняющий функции шифрования TCP-траффика, ограничения запросов в секунду и применения белого списка IP-адресов, кеширования ответов, поддержания keep-alive соединений с ASP. Впрочем, поскольку конфигурация прокси-сервера крайне схожа в различных частях приложения, она будет описана в отдельной главе во избежание дублирования.

**РЕАЛИЗАЦИЯ VPN-СЕРВЕРА**

**Разработка WebAPI**

С помощью Powershell создадим проекты типов classlib (для управления Wireguard) и webapi (для получения команд), после чего создадим файлы dockerignore, gitignore и инициализируем git-репозиторий.

Для начала введем базу для дальнейшей разработки. Из предыдущих проектов импортируем к ласс провайдера настроек[3], повышающий абстракцию использования файла appsettings.json и прочих, позволяя использовать их как набор C#-объектов в классах остальных сервисов. Дополнительно создадим класс, извлекающий переменные среды – см. приложение 2А.

Теперь опишем сервис, отвечающий за выдачу адресов. Данный сервис должен реализовывать отношения рода один-к-одному, т.е. один уникальный публичный ключ к одному уникальному IP-адресу. При необходимости повторного использования можно было бы разработать собственный класс «двунаправленного словаря», т.е. в котором, помимо ключа, каждое значение также выступает ключом и не может повторяться. Для хранения адреса используется Int32, байты из которого извлекаются методом BitConverter.GetBytes. Само хранение реализовано словарем (ключ к адресу) и дополнительным HashSet’ом, хранящим только адреса. Сам по себе данный сервис способен адресовать до (250\*256\*256). Первая цифра адреса каждого клиента выбрана за 10 (согласно RFC 1918[4]), вторая – за 6 в связи с техническими ограничениями Alpine Linux, особого значения это не имеет.

*The Internet Assigned Numbers Authority (IANA) has reserved the*

*following three blocks of the IP address space for private internets:*

*10.0.0.0 - 10.255.255.255 (10/8 prefix)*

*172.16.0.0 - 172.31.255.255 (172.16/12 prefix)*

*192.168.0.0 - 192.168.255.255 (192.168/16 prefix).*

- RFC 1918, февраль 1996

Маска подсети выбрана за /32, т.е. весь адрес относится к узлу, подсети отсутствуют, что является стандартной практикой конфигурирования Wireguard, ибо мы хотим развернуть туннель, а не сеть. Когда сервису необходимо выдать клиенту адрес – он в первую очередь пытается сгенерировать адрес на основании текущего количества клиентов, проще говоря, пытается угадать, какой адрес свободен. К сожалению, одни клиенты могут оставаться подключенными длительное время, другие же – отключаться спустя секунды, что, в действительности, порождает случайное распределение занятых адресов по адресному пространству. Это приводит к необходимости каждый раз проверять свободность того или иного адреса. В целом, данные операции не должны существенно снижать производительность, так как и поиск, и добавление в HashSet’а имеет сложность О(1). При удалении пира его публичный ключ и выделенный адрес также должны удаляться из соответствующих коллекций (что, к сожалению, будет «стоить» уже О(n)), таким образом происходит освобождение адресов. Отдельным вопросом является синхронизация реального состояния Wireguard-сервиса с WebAPI, соответствие коих, как можно понять после небольшого размышления, должна гарантироваться, в противном случае отсутствует даже возможность корректно выделять адреса, не говоря уже о более сложных функциях программы. Например, клиент может отключиться от сервера по причине потери у клиента электропитания машины, что приведет к тому, что де-факто пир будет отключен, но об этом не будет сообщено ASP-приложению. Тогда ASP-приложение должно регулярно проверять список пиров и удалять те, что уже не подключены, для этого реализован метод SyncState – принимающий «реальной состояние» и сверяющийся с ним. Код некоторых методов сервиса представлен в кратком варианте – см. приложение 3А.

Следующим этапом являет разработка авторизации. Она представляет собой т.н. промежуточное ПО (англ. middleware), предшествующее на конвейере обработки запросом конечной точке (англ. endpoint). Для начала нам потребуется сервис, который будет валидировать ключи, вычисляя их хеш и сверяя с переданным путем секретного json-файла. Его код представлен в приложении 4А – он имеет достаточно простую реализацию. Задача middleware состоит в извлечении значения заголовка Authorization, передаче его в сервис, возврат семантически корректного кода ошибки (согласно RFC 9110[5]) либо же передача запроса далее по конвейеру (в случае успешной валидации), частичный код представлен в приложении 4А.

Теперь разработаем «манипулятор» WireGuard сервером. Он представляет собой надстройку над исполнением обычных CL-команд. Выполнение осуществляется асинхронно с точками возврата управления (команда await) после запуска процесса и после начала чтения потока вывода вызванной команды. Его частичный код представлен в приложении 5А.

Последним этапом разработки являются непосредственно конечные точки (англ. endpoints) разрабатываемого WebAPI. Привести их все на бумажном носителе невозможно. Отмечу, что к каждой точке была добавлена валидация wg-ключа с целью защиты от инъекции с исполнением произвольного кода.

**Конфигурация NGINX**

Поскольку ASP имеет возможность авторизации по API-ключу, который передается в заголовке Authorize в формате base64, что является кодированием, а не шифрованием, необходимо защитить подключение. В конфигурации установлены только современные протоколы TLS – 1.2 и 1.3 а также запрещены все слабые шифры[6], из разрешенных – «ChaCha20\_Poly1305» и «AES256\_GCM\_SHA384». Помимо вышеназванного обеспечивается поддержание keep\_alive соединений с WebAPI, ограничение частоты запросов или разрешение запросов только с определенного адреса на основании переданных в контейнер переменных среды.

**Конфигурация Wireguard**

В конфигурации WG применим наиболее простую из обеспечивающих полную функциональность конфигурацию – программный код 1. Замечу, что PrivateKey оставлен пустым и генерируется при запуске уже непосредственно в контейнере.Применение iptables потребует выдачи контейнеру некоторых привилегий (NET\_ADMIN), ибо изначально контейнер не имеет доступа к настройкам сети. Директива MASQUERADE означает маскировку траффика, как будто он исходит от сервера, а не от удаленного клиента посредствам туннеля.

Программный код 1

[Interface]

Address = 10.0.0.0/8

PostUp = iptables -I FORWARD -i wg0 -j ACCEPT;

PostUp = iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE;

ListenPort = 51820

PrivateKey =

**Конфигурация Docker и сборка образа**

Как и планировалось, мы базируем свой образ на Alpine Linux – одной из самый легковесных разновидностей Linux, полноценный образ которой веси 5 МБ. Мы применяем так называемую multi-stage build: создаем контейнер, в который устанавливаем SDK, там производим компиляцию, после чего создаём финальный контейнер, куда копируем уже собранное приложение, а в нём добавляем только Runtime, что уменьшает финальную сборку – программный код 2.

Программный код 2

FROM alpine:3 AS base

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

COPY ./vdb\_node\_api /vdb\_node\_api

COPY ./vdb\_node\_wireguard\_manipulator /vdb\_node\_wireguard\_manipulator

RUN dotnet publish /vdb\_node\_api/vdb\_node\_api.csproj -c "Release" -r linux-musl-x64 --no-self-contained -o /app/publish

FROM base AS final

COPY --from=build /app/publish /app

RUN apk add -q --no-progress nginx

RUN apk add -q --no-progress openssl

RUN apk add -q --no-progress wireguard-tools

RUN apk add -q --no-progress aspnetcore7-runtime

COPY ./build\_alpine/pre-setup.sh ./etc/rest2wg/pre-setup.sh

COPY ./build\_alpine/pre-wg0.conf ./etc/rest2wg/pre-wg0.conf

COPY ./build\_alpine/pre-nginx-limit\_req.conf.template ./etc/rest2wg/pre-nginx-limit\_req.conf.template

COPY ./build\_alpine/pre-nginx.conf/ ./etc/nginx/nginx.conf

COPY ./build\_alpine/pre-ssl-params.conf ./etc/nginx/snippets/ssl-params.conf

COPY ./build\_alpine/pre-self-signed.conf ./etc/nginx/snippets/self-signed.conf

ENV ASPNETCORE\_ENVIRONMENT=Production

ENV REST2WG\_LIMIT\_REQ=100000;

CMD ["bash", "-c",

"chmod +x /etc/rest2wg/pre-setup.sh && /etc/rest2wg/pre-setup.sh"]

**Точка входа контейнера**

Рассмотрим точку входа контейнера, коей является bash-скрипт ‘pre\_setup.sh’. Предшествующая ему команда ‘chmod +x /etc/rest2wg/pre-setup.sh’ исправляет потенциальную неспособность оболочки распознать файл как исполняемый. Данный скрипт выполняет несколько задач. Он генерирует приватный ключ Wireguard, если таковой отсутствует; генерирует self-signed x509 сертификат, если таковой отсутствует, для шифрования траффика посредствам NGINX; исправляет значения переменных среды, если пользователь задал их некорректно; запускает приложения. Файл частично представлен в программном коде 3.

Программный код 3

**if** ! test -e "/etc/wireguard/wg0.conf"; then

cp /etc/rest2wg/pre-wg0.conf /etc/wireguard/wg0.conf

wg genkey >> /etc/wireguard/wg0.conf

fi

**if** ! test -e /etc/ssl/private/nginx-selfsigned.key then

openssl req -x509 -nodes -days 36500 -newkey rsa:2048   
 -subj "/CN=US/C=US/L=Miami" -keyout /etc/ssl/private/nginx-selfsigned.key   
 -out /etc/ssl/certs/nginx-selfsigned.crt

fi

**if** !((REST2WG\_LIMIT\_REQ > 0)) && ((REST2WG\_LIMIT\_REQ <= 9999999)); then

REST2WG\_LIMIT\_REQ=100000

fi

**if** ! test -e "/etc/nginx/snippets/nginx-limit\_req.conf"; then

cp /etc/rest2wg/pre-nginx-limit\_req.conf.template   
 /etc/nginx/snippets/nginx-limit\_req.conf

echo "${REST2WG\_LIMIT\_REQ}r/s;" >> /etc/nginx/snippets/nginx-limit\_req.conf

fi

**if** (testvar='all' && [[ $REST2WG\_ALLOWED\_IP = $testvar ]]); then

unset testvar;

**elif** !(ipcalc -n "${REST2WG\_ALLOWED\_IP}"); then

REST2WG\_ALLOWED\_IP="all";

fi

**if** ! test -e "/etc/nginx/snippets/while\_list.conf"; then

echo "allow ${REST2WG\_ALLOWED\_IP}; deny all;"

> /etc/nginx/snippets/while\_list.conf

fi

echo "Spinning up the Nginx reverse-proxy..."

nginx

echo "Spinning up the Wireguard service..."

wg-quick up wg0 && wg show wg0

echo "Spinning up the ASP WebAPI..."

dotnet /app/vdb\_node\_api.dll --no-launch-profile

tail -f /dev/null

**ИСТОЧНИКИ**

1. Текст лицензии Mozilla Public License [Электронный ресурс] -   
   mozilla.org/en-US/MPL/2.0/ (дата обращения: 01.06.2023)
2. Текст лицензии GNU General Public License [Электронный ресурс] -   
   www.gnu.org/licenses/gpl-3.0.en.html (дата обращения: 01.06.2023)
3. Репозиторий siteblog [Электронный ресурс] - [github.com/LuminoDiode/siteblog/blob/main/backend/Services/SettingsProviderService.cs](https://github.com/LuminoDiode/siteblog/blob/main/backend/Services/SettingsProviderService.cs) (дата обращения: 01.06.2023)
4. Международный стандарт RFC 1918 [Электронный ресурс] -   
   rfc-editor.org/rfc/rfc1918 (дата обращения: 01.06.2023)
5. Международный стандарт RFC 9110 [Электронный ресурс] -   
   rfc-editor.org/rfc/rfc9110 (дата обращения: 01.06.2023)
6. Документация Docker [Электронный ресурс] -   
   [docs.docker.com/config/containers/multi-service\_container/](https://docs.docker.com/config/containers/multi-service_container/)   
   (дата обращения: 01.06.2023)
7. Сайт от программировании [Электронный ресурс] -   
   [stackoverflow.com/questions/50845198/periodically-running-docker-command](https://stackoverflow.com/questions/50845198/periodically-running-docker-command)   
   (дата обращения: 01.06.2023)
8. Сайт от программировании [Электронный ресурс] -   
   [serverfault.com/questions/1101002/wireguard-client-addition-without-restart](https://serverfault.com/questions/1101002/wireguard-client-addition-without-restart)   
   (дата обращения: 01.06.2023)
9. Рекомендации по разработке от SSL Labs [Электронный ресурс] -   
   github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices   
   (дата обращения: 01.06.2023)

**ПРИЛОЖЕНИЕ 1А  
Неполный код скрипта создания начальной структуры проекта**

dotnet new globaljson;

dotnet new classlib -o vdb\_node\_wireguard\_manipulator;

dotnet new webapi -o vdb\_node\_api;

dotnet new sln --name vdb\_node;

dotnet sln add vdb\_node\_wireguard\_manipulator/vdb\_node\_wireguard\_manipulator.csproj;

dotnet sln add vdb\_node\_api/vdb\_node\_api.csproj;

New-Item dockerfile;

dotnet new gitignore;  
git init;

**ПРИЛОЖЕНИЕ 2А  
Неполный код класса EnvironmentProvider**

private const string ENV\_ALLOW\_NOAUTH = "REST2WG\_ALLOW\_NOAUTH"; // названия ПС

public bool? ALLOW\_NOAUTH { get; init; } = null; // извлеченные значения ПС

private readonly ILogger<EnvironmentProvider> \_logger; // логгер

public EnvironmentProvider(ILogger<EnvironmentProvider> logger) { // конструктор

\_logger = logger;

ALLOW\_NOAUTH = ParseBoolValue(ENV\_ALLOW\_NOAUTH);

}

private bool? ParseBoolValue(string EnvName) // функция извлечения и парсинга ПС

{

string? str = Environment.GetEnvironmentVariable(EnvName);

if (str is not null) {

if (str.Equals("true", StringComparison.InvariantCultureIgnoreCase))

\_logger.LogInformation($"{EnvName}={true}."); return true;

if (str.Equals("false", StringComparison.InvariantCultureIgnoreCase))

\_logger.LogInformation($"{EnvName}={false}."); return false;

\_logger.LogWarning(GetIncorrectIgnoredMessage(EnvName)); }

\_logger.LogInformation($"{EnvName} was not present.");

return null;

}

**ПРИЛОЖЕНИЕ 3А  
Неполный код класса IpDedicationService**

private const byte FirstIpByteStart = 10;

private const byte SecondIpByteStart = 6;

private const byte NetworkMask = 32;

private Dictionary<string, int> \_dedicatedAddresses; // публичный ключ к адресу

private readonly HashSet<int> \_usedAddresses; // коллекция всех занятых адресов

public string EnsureDedicatedAddressForPeer(string pubKey) {

if (\_dedicatedAddresses.TryGetValue(pubKey, out int address)) return IndexToString(address);

else {

int addr = \_dedicatedAddresses.Count; // сначала попытаемся угадать свободный адрес

if (\_usedAddresses.Contains(addr)) {

for (addr = 0; addr < MaxClients; addr++) {

if (!\_usedAddresses.Contains(addr)) {

AddPeer(pubKey, addr); return IndexToString(addr); } } }

AddPeer(pubKey, addr); return IndexToString(addr); }

}

private bool AddPeer(string pubKey, int addressIndex) {

\_dedicatedAddresses.Add(pubKey, addressIndex);return \_usedAddresses.Add(addressIndex);

}

public bool DeletePeer(string pubKey){

if (!\_dedicatedAddresses.TryGetValue(pubKey, out int address)) { return false; }

else { usedAddresses.Remove(address); return \_dedicatedAddresses.Remove(pubKey); }

}

public void SyncState(Dictionary<string, int> keyToIpActulState) {

\_dedicatedAddresses = keyToIpActulState; \_usedAddresses.Clear();

foreach (int ip in \_dedicatedAddresses.Values) \_usedAddresses.Add(ip); }

**ПРИЛОЖЕНИЕ 4А  
Неполный код классов MasterAccountsService и AuthorizationMiddleware**

public sealed class MasterAccountsService {

private readonly List<byte[]> \_mastersKeyHashes;

public MasterAccountsService(SettingsProviderService settingsProvider) {

\_mastersKeyHashes = settingsProvider.MasterAccounts

.Select(x => Convert.FromBase64String(x.KeyHashBase64)).ToList();

}

public bool IsValid(string keyBase64) {

var search = SHA512.HashData(Convert.FromBase64String(keyBase64));

return \_mastersKeyHashes.Any(k => k.SequenceEqual(search));

}

}

public sealed class AuthorizationMiddleware : IMiddleware

{

private readonly MasterAccountsService \_accountsService;

private readonly ILogger \_logger;

public AuthorizationMiddleware(MasterAccountsService accountsService, ILogger logger) {

\_accountsService = accountsService;

\_logger = logger;

}

public Task InvokeAsync(HttpContext context, RequestDelegate next)

{

var header = context.Request.Headers.Authorization; string? key;

try { key = header.Single(); } /\* Acc. to RFC, Authorize may appear only once \*/

catch (InvalidOperationException) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization header seen more than once."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest));

}

if (string.IsNullOrEmpty(key)) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key was null or empty."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest));

}

try {

if (!\_accountsService.IsValid(key)) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key was not found on the server."));

return Task.FromResult(new StatusCodeResult(401));

}

}

catch{

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key format was not valid."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest));

}

return next(context); /\* key was successfully validated \*/

}}

**ПРИЛОЖЕНИЕ 5А  
Неполный код класса WgCommandsExecutor**

public static class WgCommandsExecutor

{

public static string? LastSeenInterfacePubkey

=> WgStatusStreamParser.LastSeenInterfacePubkey;

private static async Task<string> RunCommand(string command, string fileName = @"wg") {

return await (await RunCommandStream(command, fileName)).ReadToEndAsync();

}

private static async Task<StreamReader> RunCommandStream(string command, string fileName = @"wg") {

var psi = new ProcessStartInfo();

psi.FileName = fileName;

psi.Arguments = command;

psi.RedirectStandardOutput = true;

psi.UseShellExecute = false;

psi.CreateNoWindow = true;

var process = Process.Start(psi);

if (process is null) {

throw new AggregateException("Unable to perform the command");

}

await process.WaitForExitAsync();

return process.StandardOutput;

}

private static string GetAddPeerCommand(string pubKey, string allowedIps) {

return $"set wg0 peer \"{pubKey}\" allowed-ips {allowedIps}";

}

private static string GetRemovePeerCommand(string pubKey) {

return $"set wg0 peer \"{pubKey}\" remove";

}

private static string GetWgShowCommand(string wgInterfaceName = null!) {

return wgInterfaceName is null ? "show" : $"show {wgInterfaceName}";

}

public static async Task<string> AddPeer(string pubKey, string allowedIps) {

return await RunCommand(GetAddPeerCommand(pubKey, allowedIps));

}

public static async Task<string> RemovePeer(string pubKey) {

return await RunCommand(GetRemovePeerCommand(pubKey));

}

public static async Task<IEnumerator<WgShortPeerInfo>> GetPeersListEnumerator()

{

return WgStatusStreamParser.ParsePeersFromStreamShortly(

await RunCommandStream(GetWgShowCommand()));

} }