



eduVPN

WireGuard in eduVPN

Report

Nick Aquina
SURF, Utrecht
Fontys University of Applied Sciences, Eindhoven



INTERNSHIP REPORT

FONTYS UNIVERSITY OF APPLIED SCIENCES

HBO-ICT

Data student:	
Family name, initials:	Aquina, N
Student number:	
project period: (from – till)	31 August 2020 – 22 January 2021
Data company:	
Name company/institution:	SURF
Department:	Team Security
Address:	Kantoren Hoog Overborch, 3511 EP Utrecht, Moreelsepark 48
Company tutor:	
Family name, initials:	Spoor, R
Position:	(Tech) Product Manager
University tutor:	
Family name, initials:	Vos, A
Final report:	
Title:	WireGuard in eduVPN
Date:	12 January 2021

Approved and signed by the company tutor:

Date: 12 January 2021

Signature:

Preface

This report is written for my internship for Fontys. The internship was done at SURF for the eduVPN project. My task was to build a proof of concept in which WireGuard is integrated into eduVPN. This internship took place from September 2020 until January 2021.

I would like to thank Arno Vos for his guidance and feedback throughout this internship.

I would also like to thank Rogier Spoor for guiding me throughout this internship and inviting me to meetings which gave me a valuable insight into cyber security and technological issues facing members of SURF.

And last, but not least, I would like to thank François Kooman for all technical support, advice and code reviews which helped improve the project.

All [blue text](#) can be clicked to open a hyperlink.

Contents

Preface	1
Summary	4
Introduction	5
Free software	5
The company (SURF)	6
Project	7
Context / Initial situation	7
Project goal	7
Assignment	7
Constraints	8
Development strategy	8
How does eduVPN work?	9
Dictionary	9
Research strategy	9
What components are used in eduVPN and how do they communicate with each other?	10
eduVPN Client (Phone)	10
Portal	10
VPN server API	10
OpenVPN-Daemon	10
OpenVPN	11
Node	11
RADIUS, LDAP, SAML,	11
How does eduVPN with WireGuard work?	11
Process	12
Sprint 1 (Initial improvements)	12
Dictionary	12
Introduction	12
Research strategy	12
Implementation	12
Demo and review	14
Sprint 2 (Admin features)	16
Introduction	16
Research strategy	16
Implementation	16
Demo and review	19
Sprint 3 (Deployment and packaging)	19
Dictionary	19
Introduction	20
Research strategy	20
Implementation	20
Demo and review	21

Sprint 4 (Android APP)	21
Dictionary	21
Introduction	21
Research strategy	22
Implementation	22
Demo and review	27
Sprint 5 (IP management)	27
Introduction	27
Research strategy	27
Implementation	27
Demo and review	28
Tasks for complete WireGuard support	29
Multiple WireGuard servers	29
WireGuard-Daemon security	29
Secure connection between portal and WireGuard-Daemon	29
WireGuard-Daemon authentication	29
TCP	29
Configuration expiry	30
eduVPN Profiles	30
IPv6	30
Support in all eduVPN clients	30
Consider using the pre-shared symmetric key mode	30
Recommendations	31
Implement the tasks for complete WireGuard support	31
Drop support for CentOS	31
Improve Android app test speed	31
Conclusions	32
Evaluation	33
Appendices	34
Initial setup request details	34
Connection setup request details	37

Summary

SURF is a cooperation which provides different services for all its members. Its members include universities like Fontys, research labs and other institutions. One of these services is eduVPN. eduVPN is used (by for example students) to protect their communication on public Wi-Fi networks and to access their institutions' network from home. eduVPN is continually improving their service and one opportunity to improve eduVPN is to use WireGuard instead of OpenVPN. eduVPN is built using OpenVPN. WireGuard is a new VPN implementation, which should be faster than OpenVPN.

Integrating WireGuard completely requires more time than available in this internship, so the task is to build a proof of concept. This means that the basic functionality should be available. During this internship the WireGuard integration on the server side was improved and extended. This includes:

- WireGuard configurations can now be deleted.
- eduVPN WireGuard software depends on fewer dependencies.
- Admins can now see who is connected using WireGuard.
- Admins can now disable WireGuard users.
- eduVPN with WireGuard support is now available in distributions packages for Fedora and Debian.
- eduVPN with WireGuard support can now be easily deployed using a deployment script.

On the client side nothing had yet been done. During this internship support for WireGuard was added in the eduVPN Android app.

There are still tasks that have to be done to completely integrate WireGuard into eduVPN, but a proof of concept has been successfully built.

Introduction

When working from home you want to have access to all digital services that you have access to when you are physically on your university or company. eduVPN provides software which makes this possible. When more people start to work from home, this software has to scale with the amount of users. The part of eduVPN which is most important when scaling, is the underlying software that is used for VPN connections: OpenVPN. OpenVPN is a full-featured VPN solution that has been around since 2001¹. In 2016 a new VPN implementation was released to the public: WireGuard. WireGuard aims to be simpler, faster, leaner, and more useful, than other VPN implementations².

Using WireGuard instead of OpenVPN could mean a faster eduVPN service. Some work has already been done by another intern to support WireGuard. In this report, the process of improvements done and extensions made is described.

Free software

eduVPN is free software. Both the server side and all eduVPN clients are free software. Free as in price, but more importantly also free as in freedom. Everyone has the freedom to run eduVPN, study it, change it and redistribute eduVPN with or without changes. These freedoms enable people to share and cooperate, which are essential for a society.

In the spirit of free software, all changes done during this internship were published online (with permission), so other people can use and improve the changes in the projects used in this internship:

- [WireGuard Daemon](#)
- [Portal with WireGuard support](#)
- [WireGuard Debian package builder](#)
- [Android client with WireGuard support](#)

¹Wikipedia contributors. (2021, January 6). OpenVPN. Wikipedia. <https://en.wikipedia.org/wiki/OpenVPN>

²Donenfeld, J. A. (n.d.). WireGuard: fast, modern, secure VPN tunnel. Retrieved 4 January 2021, from <https://www.wireguard.com/>

The company (SURF)

This internship was done for SURF. SURF is a cooperation between more than 100 education and research institutions like universities to collaborate on digital projects³. Fontys is also a member of SURF⁴. A member of SURF can use the services provided by SURF. SURF consists of multiple operating companies, these companies are currently being merged into SURF⁵.

- [SURFnet](#): Manages the national research and education network (NREN), [eduroam](#) and eduVPN.
- SURFMarket: Manages software licenses, cloud services and digital content for institutions.
- SURFSara: Manages high-performance computing (HPC) services, data management and storage, cloud, e-science support and visualization.

One of projects managed by SURFnet is a VPN service, that has to be improved in this internship. This VPN service is used by students of universities to protect their communications on public Wi-Fi networks and to access the university network from home.

³The organisation and management of SURF. (n.d.). SURF.nl. Retrieved 6 September 2020, from <https://www.surf.nl/en/about-surf/the-organisation-and-management-of-surf>

⁴Manen, F. (n.d.). Overview of the members of SURF. SURF.nl. Retrieved 6 September 2020, from <https://www.surf.nl/en/the-surf-cooperative/surf-members>

⁵Nouwen, N. (n.d.). SURF operating companies. SURF.nl. Retrieved 6 september 2020, from <https://www.surf.nl/en/surf-operating-companies>

Project

Context / Initial situation

SURF provides a VPN solution to universities, research labs, etc., to allow students, researchers and staff to use a VPN connection on their devices. A VPN can be used to encrypt all traffic, send it to another computer that will decrypt the traffic, and then send it to a website. This can be used to prevent people from spying on the local network to see what websites you visit. The website you visit will see the IP address of the VPN and not your original IP, which preserves your privacy. A VPN can also be used to access systems that are not public but only accessible from within a company or university. When you are in a country that blocks certain websites, you can use a VPN to access them anyway.

The VPN solution build by SURFnet (eduVPN) is currently built using OpenVPN. When using OpenVPN as your VPN software, it is necessary to import a config file on the client. eduVPN offers a website to download the right config for your institution. At some point in time, organizations started to ask for an easier way to set up a VPN connection, this resulted in the development of eduVPN clients which allows connecting to eduVPN without having to download and import configuration files manually. These configuration files are downloaded by the eduVPN client automatically.

OpenVPN is one of the existing VPN implementations. OpenVPN is a full-featured VPN solution that has been around since 2001⁶. In 2016 a new VPN implementation was released to the public: WireGuard. WireGuard aims to be simpler, faster, leaner and more useful, than other VPN implementations⁷.

Using WireGuard instead of OpenVPN could mean a faster eduVPN service. By using WireGuard instead of OpenVPN for eduVPN the performance of eduVPN should improve. WireGuard should offer better performance and improved security, because the code base is smaller which means it is easier to check for security flaws.

Project goal

The goal of the project is to have a version of eduVPN which is implemented using WireGuard. This implementation covers both the server side and the client side. This WireGuard implementation will not have all the features that the OpenVPN implementation provides. Not all clients will be modified to support WireGuard, only the Android client will be modified to create a proof of concept.

Assignment

The assignment is to create a PoC (proof of concept) in which it is possible to connect to a WireGuard eduVPN server by using a eduVPN client. There has already been done some work to support WireGuard, but at the moment this WireGuard implementation is not complete. The existing WireGuard implementation will be extended to make it possible to use a WireGuard connection using the eduVPN Android client.

⁶Wikipedia contributors. (2021, January 6). OpenVPN. Wikipedia. <https://en.wikipedia.org/wiki/OpenVPN>

⁷Donenfeld, J. A. (n.d.). WireGuard: fast, modern, secure VPN tunnel. Retrieved 4 January 2021, from <https://www.wireguard.com/>

Constraints

The existing wireguard-daemon, used by eduVPN to manage the WireGuard server, is written in Go, vpn-user-portal in PHP and the Android app in Java and Kotlin. These languages will have to be used. The eduVPN project has developing practices that have to be followed. These practices can be found on [Github](#) and include:

- PHP code should run on all PHP version ≥ 5.4 (Which means it should also run on PHP 7.x).
- Do not use frameworks.
- Minimize the number of dependencies.
- Only use high quality, stable, small libraries that follow Semantic Versioning.
- Prefer dependencies that are already properly packages in CentOS/EPEL.

Development strategy

To develop the necessary software, the Agile framework Scrum will be used. To implement WireGuard in eduVPN, multiple subtasks have to be completed. These subtasks will be done in sprints. This way a small part of the integration will be developed that can be tested without requiring a complete WireGuard integration. When not everything is implemented, it can still be shown that parts of the implementation work. There will be 5 sprints, each sprint will cover a subtask that is necessary for a complete implementation. After every sprint the project manager is informed about the progress and a demo of new functionality will be shown. In the case of a longer sprint, the project manager will be informed while the sprint is still in progress. After every sprint the planning will be updated to represent new developments (like a sprint that takes longer or shorter than expected).

Existing code for the server side of eduVPN is written in PHP and Go, because I have never used Go before, a startup phase is planned in which I will learn the basics of Go. Because I do not have a lot of experience with developing an Android app, Sprint 5 in which the existing Android eduVPN client will be updated to support WireGuard will take longer than the other sprints.

How does eduVPN work?

Dictionary

LDAP Lightweight Directory Access Protocol (LDAP)

RADIUS Remote Authentication Dial-In User Service (RADIUS)

SAML Secure Assertion Markup Language (SAML)

VPN Configuration The VPN configuration is a file which contains the keys and the hostnames of the VPN which are necessary to create a VPN connection. Both WireGuard and OpenVPN have their own configuration file format. Might be referred to as just configuration or config.

Research strategy

- Workshop, Prototyping: To understand what components are used and how eduVPN works an eduVPN server will be created. For simplicity all components will run on one computer. Most institutions allow their students to login using the same credentials they use for logging in to other university related services. To accomplish this, eduVPN can connect to servers storing users using LDAP, RADIUS, SAML and other authentication protocols. To learn how this works a [freeRADIUS](#) server will be created.
- Field, document analysis: To further understand the design of eduVPN, I will read the eduVPN documentation written by eduVPN developers.
- Library, Literature study: To understand OpenVPN, I will read the documentation and articles about OpenVPN.
- Workshop, Decomposition: A diagram with eduVPN broken into its components will be created. Arrows will indicate the communication between components.

What components are used in eduVPN and how do they communicate with each other?

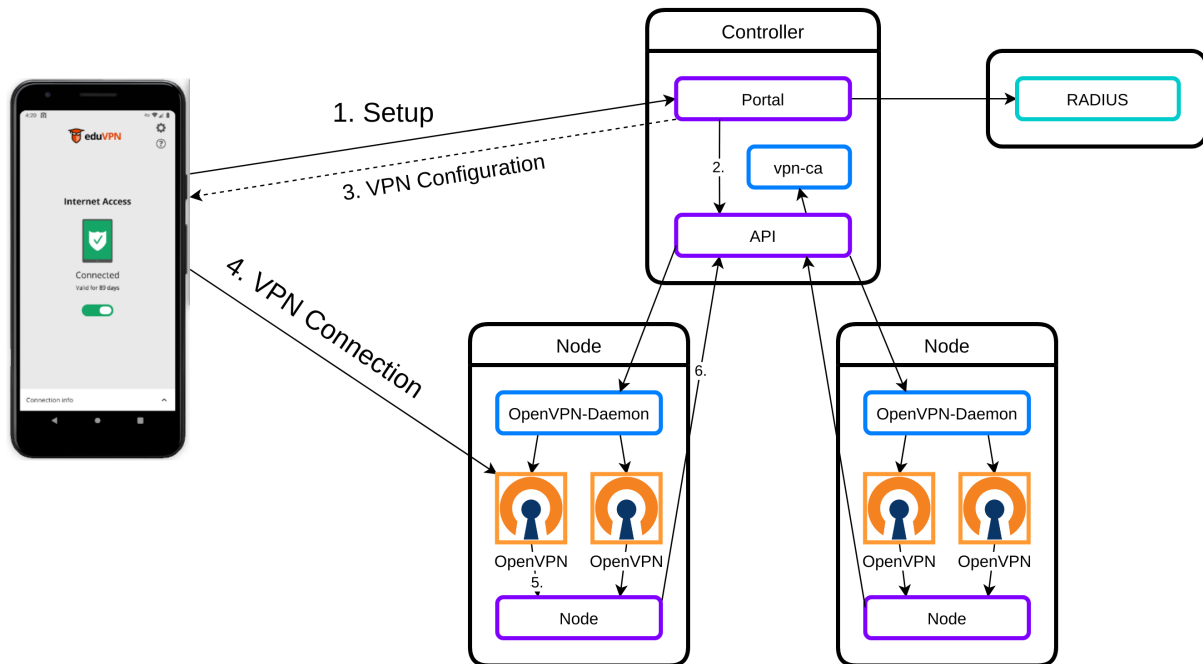


Figure 1: Network diagram eduVPN with OpenVPN. All arrows that initiate a connection also receive a response, but most arrows indicating a response have been omitted for clarity.

The device will request a configuration with a key pair from the portal. The portal will use the API to create a new configuration. This configuration is then used to connect to OpenVPN. When OpenVPN encounters a new connection it will ask the Node if the key pair used is valid, the Node will then ask the API if the user used a valid key pair. When the key pair is valid the connection will be accepted.

eduVPN Client (Phone)

This is the device of the user, a student or teacher using eduVPN.

Portal

There are 2 ways for a user to obtain a configuration necessary to connect to eduVPN. The user can use an eduVPN client, or he can go the portal to download a configuration manually. The user visits the URL of the portal to obtain a configuration, and the eduVPN clients connect to the portal to obtain a configuration.

VPN server API

The VPN server API, or just API, is used by the Portal to store and retrieve information about users and configurations. In the future the API will be merged into the portal. This is the reason that no WireGuard functionality was built into the API.

OpenVPN-Daemon

The OpenVPN-Daemon is used to retrieve a list of clients connected to OpenVPN and to disconnect clients when the system administrator disables a user.

OpenVPN

OpenVPN is the VPN that the client connects to, to set up and use a VPN connection.

Node

Every time a user connects to OpenVPN, OpenVPN asks the Node if this user has the right credentials to use the VPN service.

RADIUS, LDAP, SAML, ...

It is possible to connect eduVPN with existing authentication software like RADIUS, LDAP or SAML.

How does eduVPN with WireGuard work?

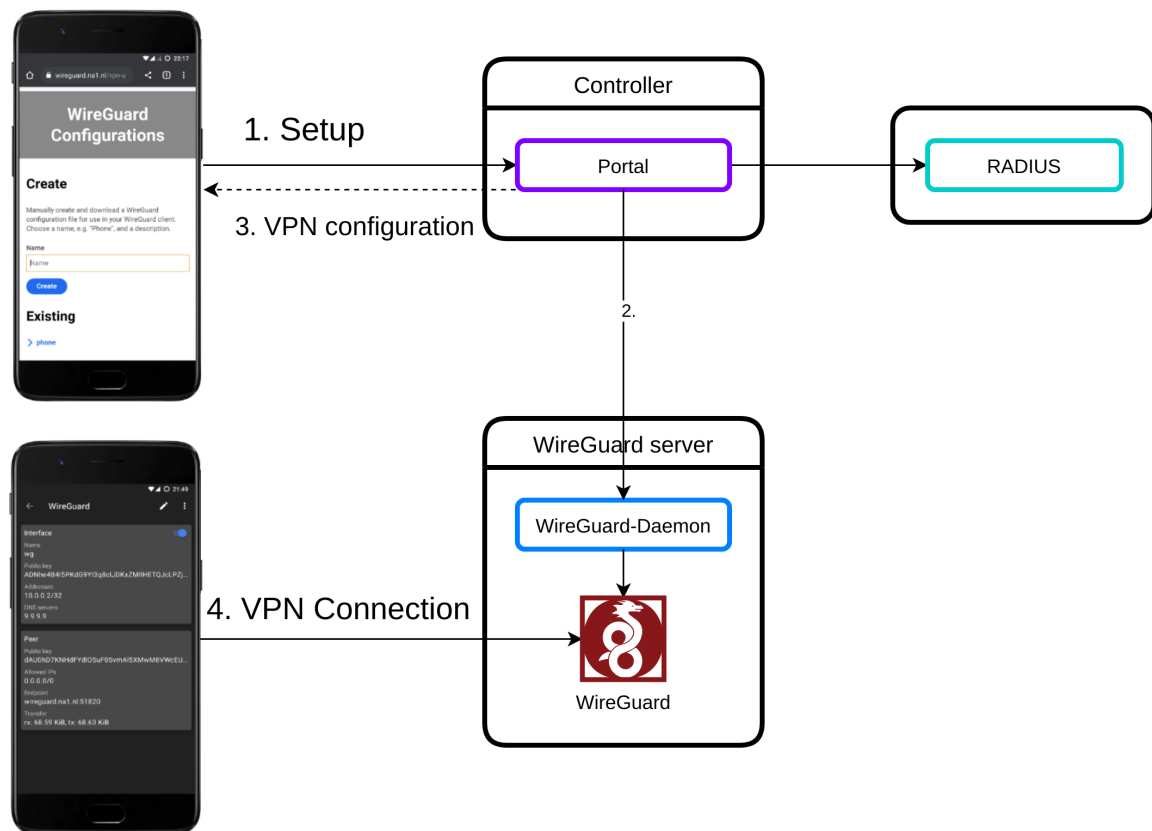


Figure 2: Network diagram eduVPN with WireGuard. All arrows that initiate a connection also receive a response, but most arrows indicating a response have been omitted for clarity.

Before this internship, there was no WireGuard support in an eduVPN client and the official WireGuard app had to be used. The user downloads a configuration in the portal and imports these configurations in the WireGuard app.

When a user creates a WireGuard configuration in the portal, the portal tells the WireGuard-Daemon to create a config. The Daemon reserves an IP address for the configuration and adds this configuration to WireGuard. Before this internship the user's browser had to connect directly to the daemon, after Sprint 1 the browser only talks to the Portal. After sprint 4, the Android eduVPN client with WireGuard support can be used instead of manually creating a configuration.

After creating a configuration the user can either download the configuration as a file or scan a QR code which contains the configuration. This QR code can be scanned by the official WireGuard app.

Process

Sprint 1 (Initial improvements)

Dictionary

WireGuard interface The WireGuard interface is a virtual network interface. Network interfaces can be hardware or software based. Hardware based network interfaces use the WiFi chip in your PC or the physical port in which you can put a cable which connects you to the internet. The WireGuard interface is a virtual interface created by software which is used for the VPN implementation.

Introduction

Multiple tasks were planned for sprint 1.

When connecting to the WireGuard VPN, it is currently not possible to access the internet, it is only possible to connect to other clients that are also using the VPN. This problem will be fixed in this sprint.

The existing WireGuard-Daemon, the component that connects the portal and the WireGuard VPN, will be improved. According to the lead developer of eduVPN the current WireGuard-Daemon has too many unnecessary dependencies that have to be removed. The functionality provided by these dependencies has to be implemented or another workaround has to be thought of.

At the moment, the QR code that can be scanned by an official WireGuard client is generated by the daemon. To accomplish this, the browser used to open the portal connects to the daemon which makes it necessary to expose the daemon to the internet which is unacceptable. This will be fixed in this sprint.

Research strategy

- Library, Literature study: To understand what the dependencies that are used do, their documentation will be read.
- Lab, Unit test: Unit tests will be added to automatically test if code still works after changes.

Implementation

Internet using WireGuard

A previous intern started the development of a WireGuard-Daemon and the necessary changes in the portal to make it possible to connect to a WireGuard VPN by scanning a QR code in the portal. At first, it did not seem to be possible to connect to the internet using the VPN connection, so it was planned to fix this issue. After doing some more research I discovered I did not read the documentation of the previous intern properly and it was already possible to connect to the internet after setting the right packet filter rules. These rules are now set automatically when deploying (since Sprint 3).

WireGuard Daemon

The daemon depends on multiple other projects which are not necessary. Using more dependencies means it is necessary to do more work when packaging the daemon, because the dependencies that are not packaged yet have to be packaged for both Debian and Fedora (Sprint 3). If a dependency is not maintained anymore, extra work is necessary to maintain this dependency. Having less code means the code is more maintainable and a security audit is easier.

Multiple dependencies were removed from the daemon.

At first `nftables` and `netns` were removed. These dependencies set up the packet filter rules. Setting up the packet filter rules did not work, so removing these dependencies did not change the behaviour of the daemon. Setting up the right packet filters is now done when deploying the daemon.

Then the `httprouter` dependency was removed. Instead of using an external library to do the routing, the routing is now done by the Go built-in `net/http` library. This library is shipped with the Go language itself. Manually routing using this library requires more code, but the routing code is now explicit which means it is more clear how HTTP requests are routed.

The daemon did not specify which version of a library should be used, it always used the latest version of a library. This is problematic as this might work today but will break in the future when the library changes in a backward incompatible way. To solve this problem I decided to convert the project to use Go modules. Go modules is a way to specify what libraries and what version of these libraries your project uses. This also made it easier to get an overview of what libraries were still used in the daemon.

Then the `go-bindata-assetfs` was removed, this dependency was used to serve static files over HTTP. This was used for the index page and was replaced by an index page which returns `404 Not found`.

Then the `netlink` dependency was removed. This dependency set up the WireGuard interface. Setting up the WireGuard interface was replaced with running a shell script, but this was not ideal either. In Sprint 3 these scripts were removed and the responsibility of setting up the WireGuard interface was moved to a config file that is installed when installing the daemon package. This config file is used by [systemd](#).

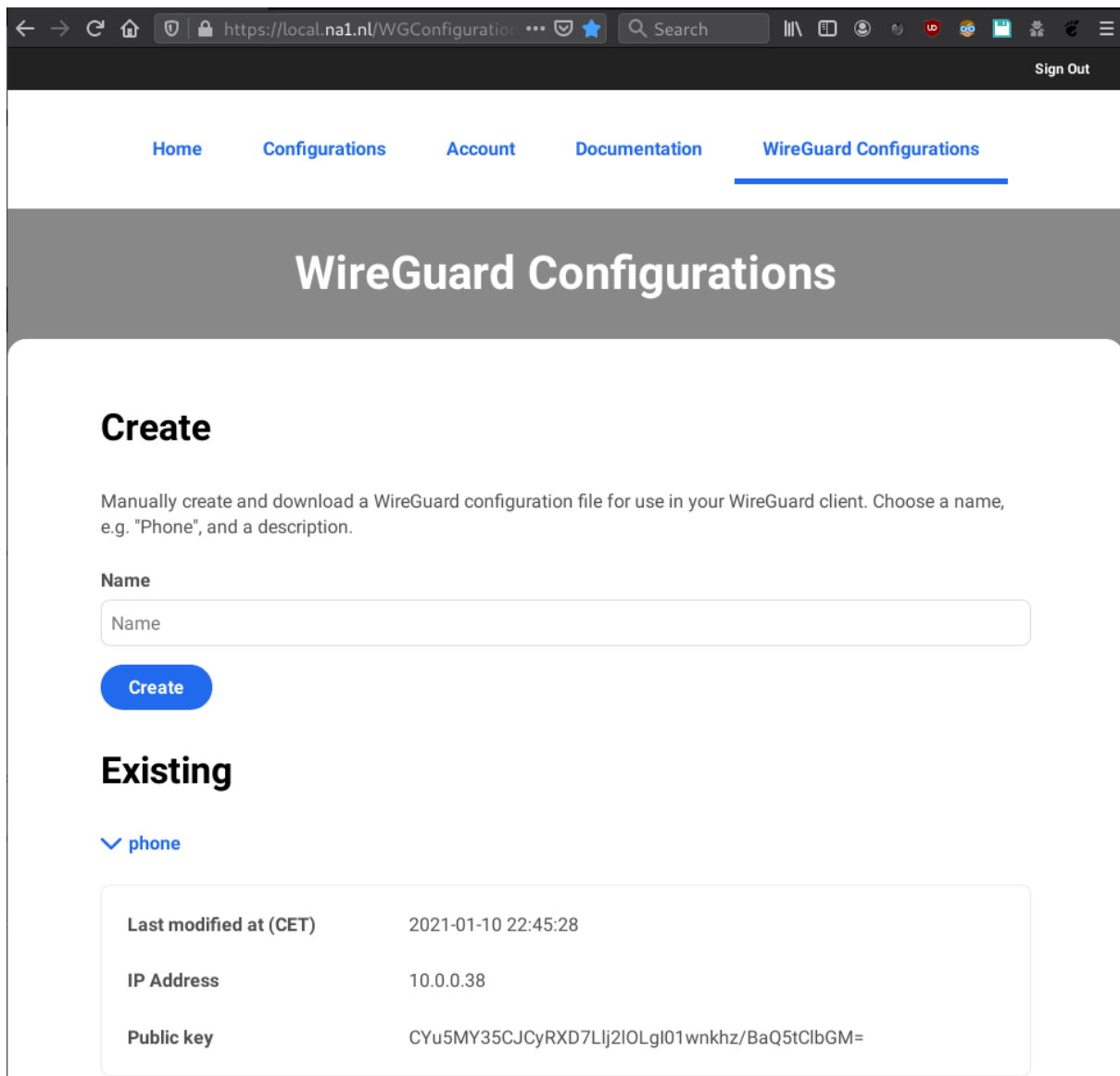
Other improvements to the daemon were made which were not planned:

- Do not store the private key of the client, this is a security risk.
- Allow clients to create a config without sending the private key (this will later be used by the Android app).
- The mutex used to secure the data from being written to and read at the same time was locked and unlocked throughout the whole application. This was moved to one file which made it much easier to prove there are no concurrency issues.
- Refactor code so tests are possible and add unit tests.

Portal

In the software that the previous intern wrote, the WireGuard-Daemon created the QR code that was shown in the portal. Creating this QR code had to be moved to the portal. Unfortunately the WireGuard portal pages were completely invalid HTML. It did work as a browser tries to do its best with invalid HTML, but it might break after a browser update. The HTML was fixed by recreating the WireGuard pages. The PHP code was also improved and the QR Code generation was moved to the portal.

Demo and review



WireGuard Configurations

Create

Manually create and download a WireGuard configuration file for use in your WireGuard client. Choose a name, e.g. "Phone", and a description.

Name

Create

Existing

▼ phone

Last modified at (CET)	2021-01-10 22:45:28
IP Address	10.0.0.38
Public key	CYu5MY35CJCyRXD7Llj2IOlG1wnkhz/BaQ5tCibGM=

Figure 3: WireGuard configurations overview

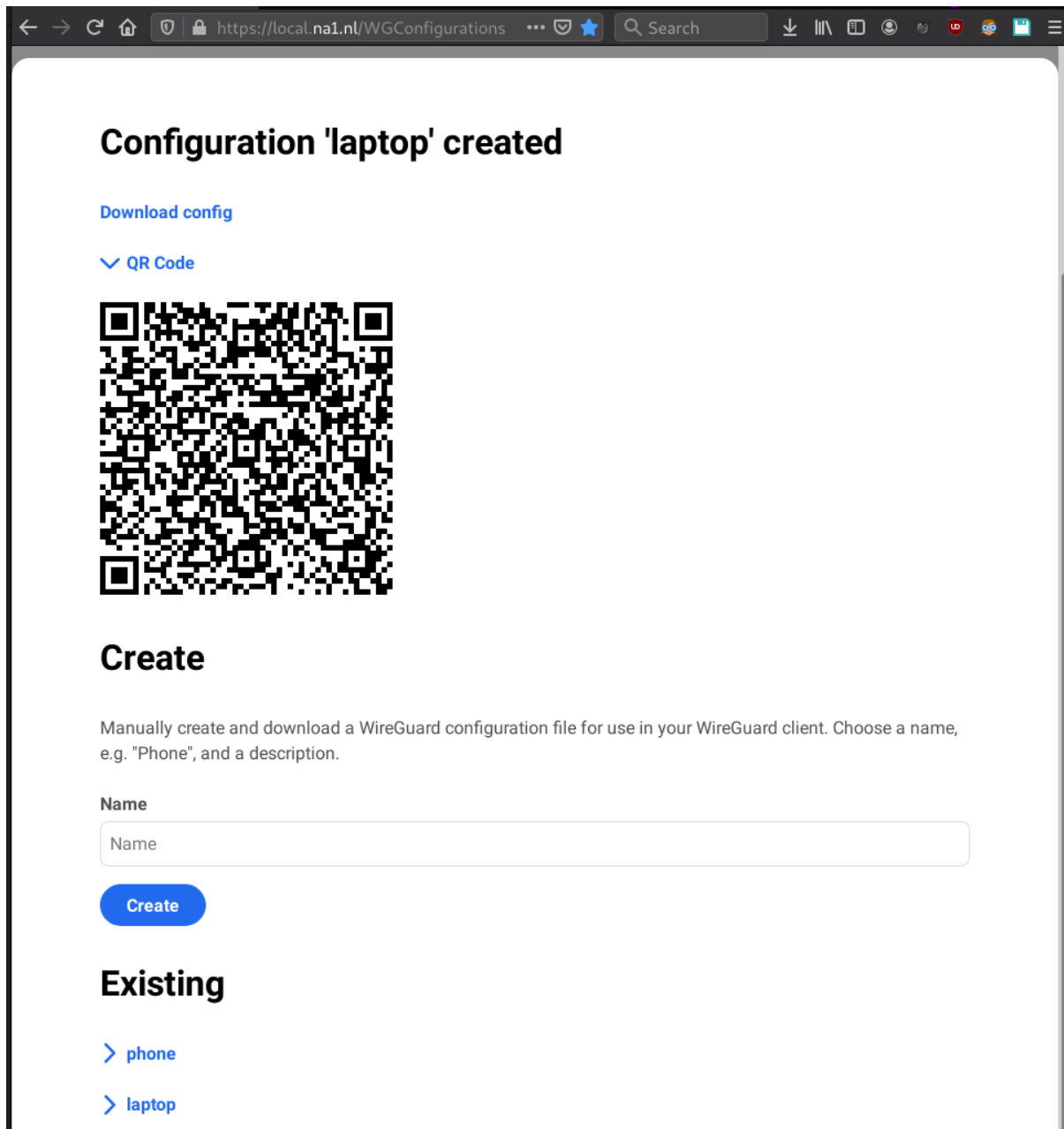


Figure 4: WireGuard configurations overview after creating a configuration

At the end of sprint one a demo was shown with the functionality that was planned. This demo showed how to create a WireGuard config in the portal and how to scan the QR code to connect to the internet using the WireGuard VPN by using the official WireGuard Android client. Using the VPN with the WireGuard Windows and Linux client was also tested and works. While implementing the features in sprint 1, some unplanned work had to be done. The code had to be changed to make it possible to test the code, Go modules are now used, and other improvements were done. In sprint 2 I will improve the code using the feedback provided by the lead eduVPN developer.

Sprint 2 (Admin features)

Introduction

In this sprint it was planned to make it possible for admins to disable users and to view active WireGuard connections. This functionality is already implemented for OpenVPN.

Research strategy

- Workshop, code review: During Sprint 1, feedback was received on the code written in Sprint 1 and the code will be improved based on this feedback.
- Library, community research: There will be searched for projects that have similarities in functionality with the daemon to understand how they implemented the functionality to get a list of the currently connected users.

Implementation

Blocking users

When blocking a user it is necessary to prevent the user from creating new configurations, to disable already created configurations, and to terminate active VPN connection. When blocking a user this user is logged out everywhere so it is not possible to create new configurations anymore. Every OpenVPN connection is also killed. Every time a user connects with OpenVPN it is checked if the user is blocked. This is not possible with WireGuard. When a user is blocked, the user is unable to create new WireGuard configurations, but the existing configurations keep working. These configurations have to be disabled by the daemon. This functionality was built and when an admin now disables a user, all the WireGuard configurations of the user are disabled instantly. When the admin enables the user again, the configurations are enabled again if they were not removed. A config might be removed when we do not have enough IP addresses available, see Sprint 5 IP management.

Viewing connected users

WireGuard is a stateless protocol, so there is no list of users that are connected. When someone is using an official WireGuard client and stops the VPN connection, the clients simply stops sending packets but there is no way for the server to know if the client is disconnected or if the client is not using his internet. WireGuard has an optional keep-alive which could be used for this, but this keep-alive is (in my opinion) a waste of internet bandwidth. A keep-alive sends an empty packet at a specified interval to tell a firewall or NAT that the connection should be kept open.

At first, I looked at projects similar to the daemon to see how these projects view connected users. There was only one project which had this functionality. This project analyzed WireGuard logs to get more information about the current connections but a simpler approach was chosen to see who is connected.

After looking at the WireGuard code (initially to find a way to get the last time the amount of bandwidth used was changed), it was discovered that WireGuard stores the last time a handshake took place.

In `wireguard/messages.h` multiple limits are hard coded. These include:

```
REKEY_AFTER_TIME = 120
REJECT_AFTER_TIME = 180
```

Which means that every 120 seconds WireGuard will attempt to perform a new handshake and will reject all packets if there was no handshake after 180 seconds. After this time the VPN can not be used anymore if there was no handshake. The user did thus not use his VPN connection for at least 3 minutes. After these 3 minutes we assume the user disabled his VPN connection, and we do not list the user anymore in the connected users list.

When the eduVPN Android app disables his WireGuard connection, the app sends a disconnect call to the server and the server will remove this user from the connected user lists instantly, see Sprint 5 IP management.

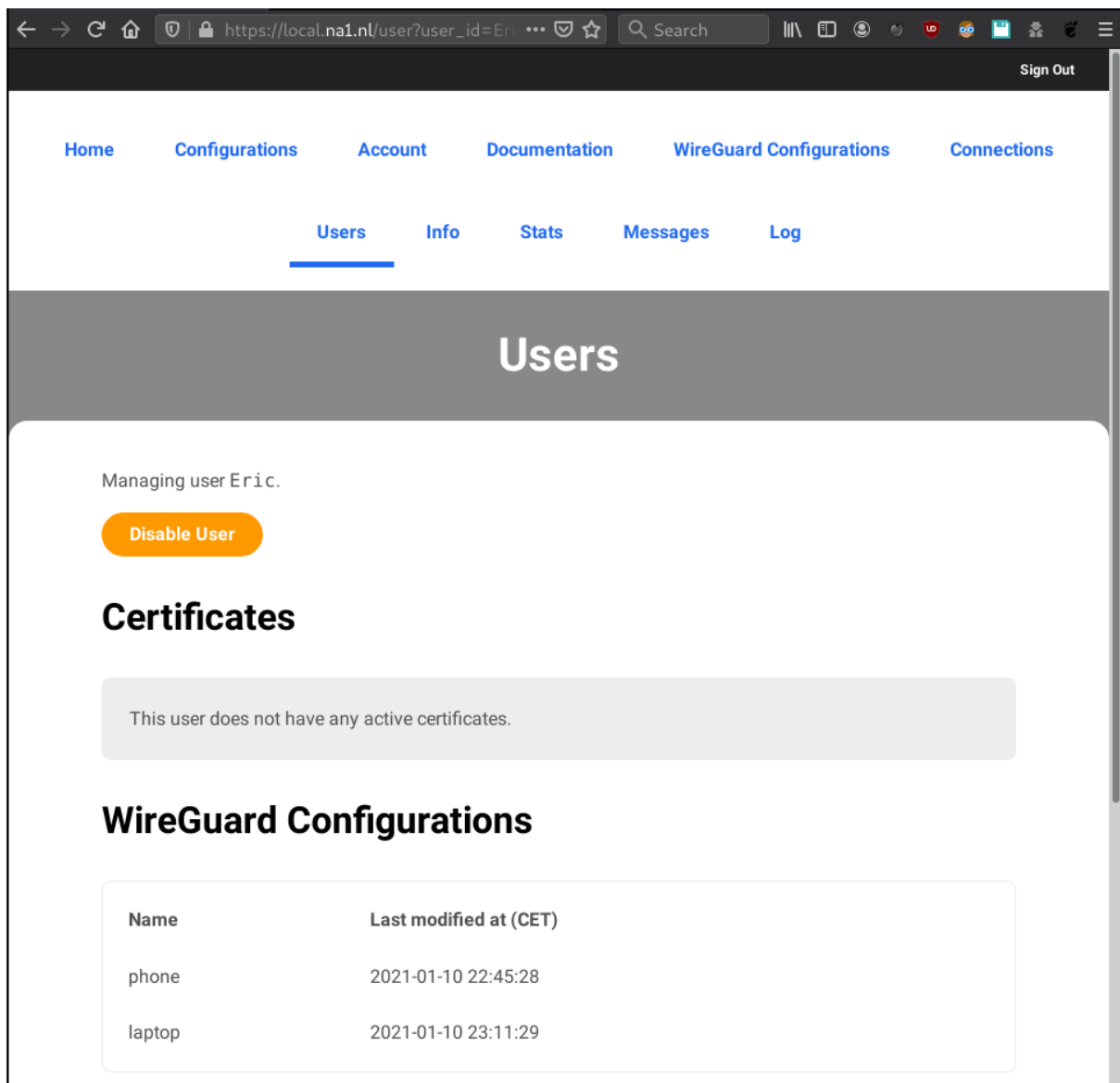


Figure 5: Admins can now view the WireGuard configuration of their users

Deleting configurations

It was not possible to delete existing configs, this was implemented this sprint. There is now a delete button.

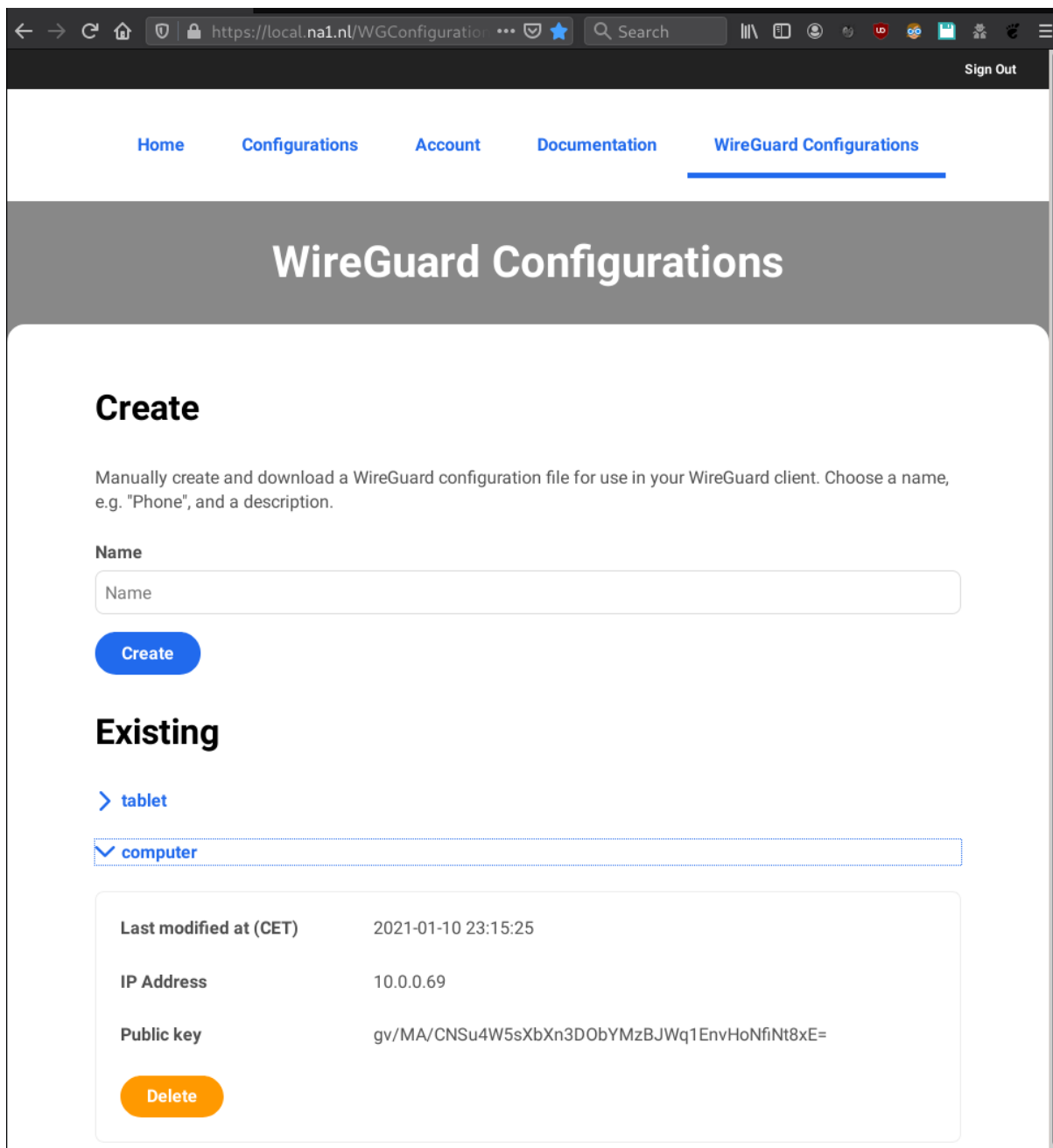


Figure 6: WireGuard configuration overview

Viewing WireGuard configuration of users

It was for an admin not possible to view the WireGuard configurations of their users. This functionality was added this Sprint.

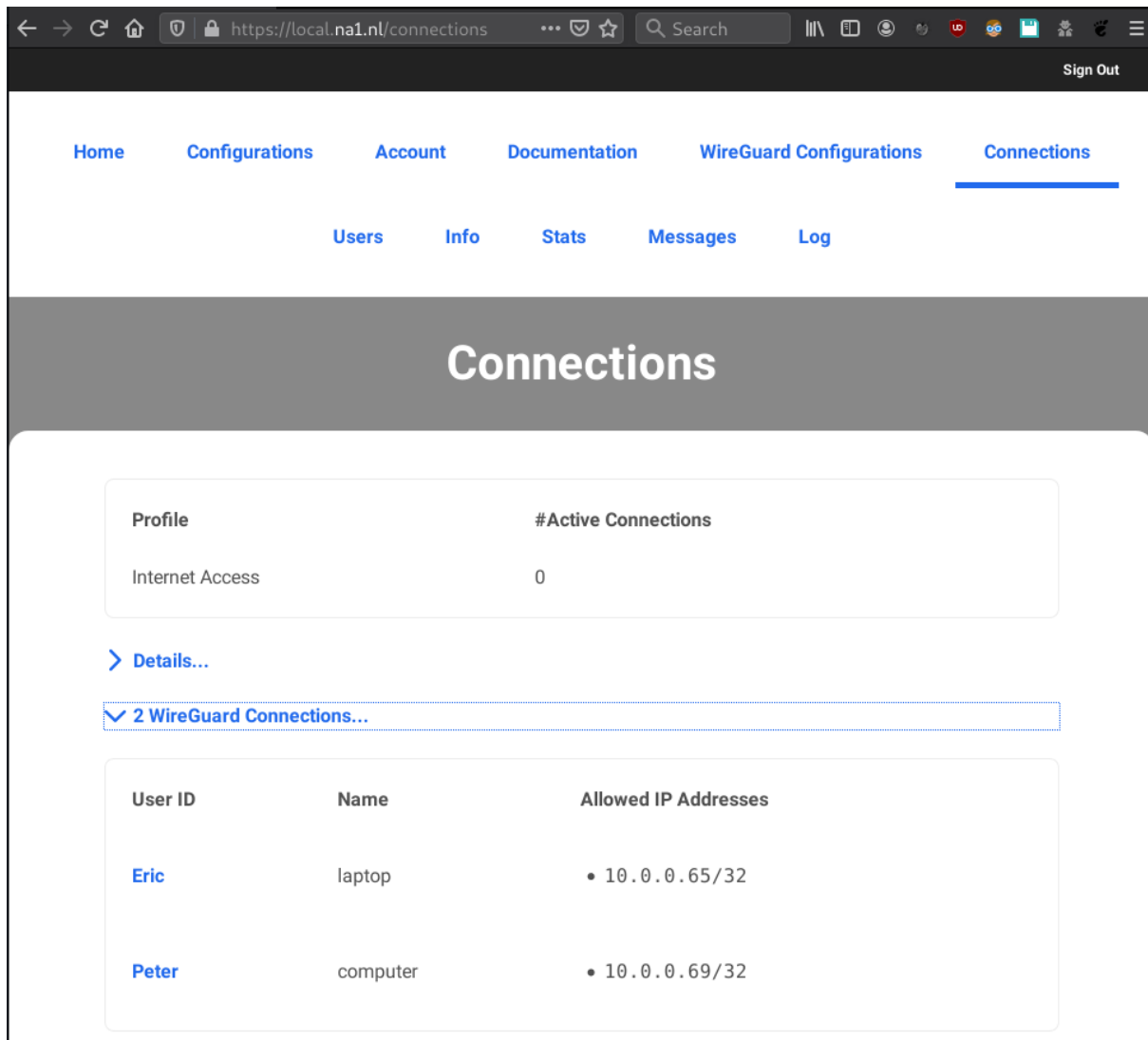


Figure 7: Admins can now view who is connected to WireGuard

Demo and review

At the end of sprint 2 the demo was only shown to the lead developer of eduVPN, because the project manager was not available at that time. The lead developer came with the idea to set up a server with WireGuard so that they could try it out themselves. I created this server in Sprint 3 and discovered some other issues which will be fixed in the next sprint. The right firewall rules have to be created and right now OpenVPN and WireGuard use IP addresses in the same range which might break if OpenVPN and WireGuard give the same IP to a different client.

Sprint 3 (Deployment and packaging)

Dictionary

Git commit hash Git is a system to control versions of software. Using Git it is possible to split software changes in separate commits which make it possible for other developers to easily see what changes were made since the previous version. To identify these commits, Git uses a Git commit hash which is the unique identifier for a commit.

Distribution packages (distro packages) Distro packages are programs which can be installed on Linux distributions. Debian uses `.deb` files and Fedora uses `.dnf` files, these are comparable to

.exe files on Windows.

Software repository (repository) A software repository on Debian is an URL which points to a location where distribution packages can be retrieved from. Debian has their own repository for software supported by Debian. Everyone can host their own repository where they can host packages.

Introduction

eduVPN can be installed using a script and distribution (distro) packages. The installation method using distro packages is available for all software necessary for running eduVPN with OpenVPN. This installation method should also be available for WireGuard. To support this an installation script (also called deployment script) has to be made and the necessary distro packages have to be created.

eduVPN is available in distro packages for Debian, Fedora and CentOS. WireGuard packages will only be made for versions of distros that have WireGuard available in their official repositories. This includes Debian 11 and Fedora 33.

Research strategy

- Library, Literature study: To learn how to package for Debian and Fedora I will read the packaging manual provided by these distributions.
- Field, expert interview: The server side developer of eduVPN already packaged existing eduVPN software. He will be asked for help when problems occur.
- Workshop, code review: During sprint 3, the code written in Sprint 2 will be improved based on the feedback received.

Implementation

Packaging

Because all unnecessary dependencies were removed in sprint 1, the WireGuard-Daemon only depends on 2 packages: wgctrl and go-cmp. wgctrl however depends on 3 other packages which depend on other packages as shown in the picture:

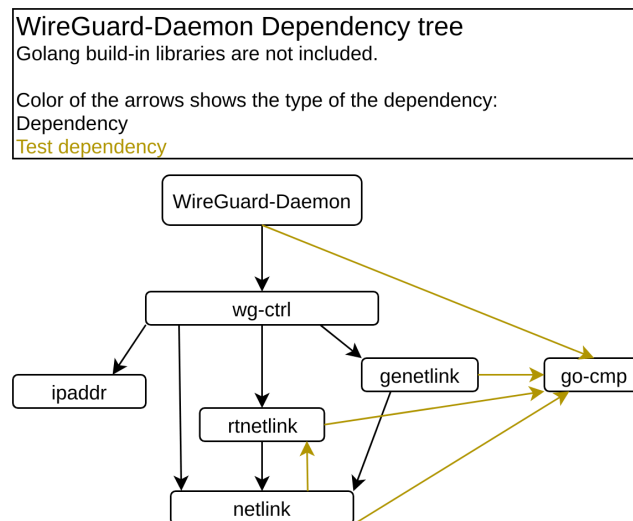


Figure 8: WireGuard Daemon dependency tree

go-cmp is already available in Debian⁸ The circular dependency between rtnetlink and netlink is problematic. To build rtnetlink, netlink is necessary, but to build and test netlink, rtnetlink is necessary. To solve this problem we first have to build netlink without running the tests that depend on rtnetlink, and then

⁸Debian – Details of package golang-github-google-go-cmp-dev in bullseye. (n.d.). Debian. Retrieved 6 November 2020, from <https://packages.debian.org/bullseye/golang-github-google-go-cmp-dev>

build netlink. Doing this manually is no problem but to automate this process is a bit harder. Currently, the tests of rtnetlink are not run at all⁹ to solve this problem.

Fedora has wg-ctrl and go-cmp available in its repositories, so it was assumed only the daemon had to be packaged, because the dependencies were already available. Unfortunately, this was not the case as wg-ctrl in Fedora was outdated, so the new version of wg-ctrl had to be packaged.

Debian only has go-cmp in its repositories which meant that all its dependencies had to be packaged for Debian. After packaging all dependencies, the existing **builder** was used to automate building the packages, and to create a Debian repository which can be added to a Debian server to install eduVPN with WireGuard support. When deploying a new version of my code to a VM I use this repository which now other people that want to deploy eduVPN with WireGuard can use. Instructions for using this repository can be found in the [README.md of the WireGuard-Daemon](#).

The existing builder had to be modified to build Go packages. Most software projects use [semantic versioning](#), but some Go projects use git commit hashes instead (ipaddr and wgctrl in our case). This makes it harder to determine what version of the project is necessary. The existing builder used **uscan** which did not support git commit hashes. I looked at the code of **uscan** to see how much work it would be to add support for git commit hashes to uscan but was unable to figure this out in a short time and decided to use the software that the Debian Go Team uses instead of uscan: **gbp export-orig**. When the builder now wants to build ipaddr or wgctrl, it uses **gbp export-orig** instead of uscan.

Deployment script

WireGuard can now be deployed by running a deployment script. This script is available in the [README.md of the WireGuard-Daemon](#).

Demo and review

A demo server was deployed using the new deployment script and the new packages. Some unexpected problems were encountered when packaging but these problems were resolved in time.

Sprint 4 (Android APP)

Dictionary

Fork To add WireGuard support to eduVPN, a copy of the eduVPN software was made with the WireGuard changes. This copy is called a fork.

Mobile deep linking A deeplink is a link to a specific location inside a mobile app.¹⁰ The eduVPN app uses the deeplink `org.eduvpn.app:/api/callback?code=eyJhbG...` to share an access token from the portal with app.

Introduction

eduVPN has multiple clients that can be used to connect to set up a VPN connection using OpenVPN. There are clients for [Windows](#), [macOS and iOS](#), [Android](#) and [Linux](#).

On all the platforms where the eduVPN client is available it is possible to download the OpenVPN software and manually import an OpenVPN configuration downloaded from the portal. It is possible to download a configuration from the portal and import it into the official WireGuard client for the same platforms, but to prevent users from having to manually download and import configurations, WireGuard support has to be added to the eduVPN clients. Because of time constraints, I was decided to add WireGuard support to only 1 client. According to the project manager Windows is the most used client, so it would make sense to start with the Windows client. However, I personally do not use Windows, and I am more interested in Kotlin (used for the Android app) than in C# (used for the Windows client) so I decided to add WireGuard support to the Android client.

⁹Aquina, N. (2020, November 5). fantostisch/deb-builder. GitHub. https://github.com/fantostisch/deb-builder/blob/2592f46c3bc85b3abec766867ce5180900be8ff4/build_packages.sh#L116

¹⁰Wikipedia contributors. (2020, December 28). Mobile deep linking. Wikipedia. https://en.wikipedia.org/wiki/Mobile_deep_linking

Research strategy

- Library, Community Research: When implementing WireGuard in the Android app, I will look at existing Android apps that use WireGuard as their VPN implementation.
- Library, Literature study: To understand how the Android WireGuard library should be used, I will read the documentation of this library.
- Workshop, Code review: All code that was written in this Sprint will be reviewed by the eduVPN Android developer and the code will be improved based on his feedback.
- Workshop, Prototyping: To understand how the current Android app works, the requests that the app makes to an eduVPN server will be examined.
- Lab, Unit test: The UI tests were fixed and new unit tests were written to test the WireGuard functionality.

Implementation

When using the eduVPN app and a server that supports WireGuard, the app now has to choose between OpenVPN and WireGuard. After a discussion with the project manager and the lead developer we choose to add a setting to the eduVPN app where the user can choose to use WireGuard when it is available. This option was added to the settings:

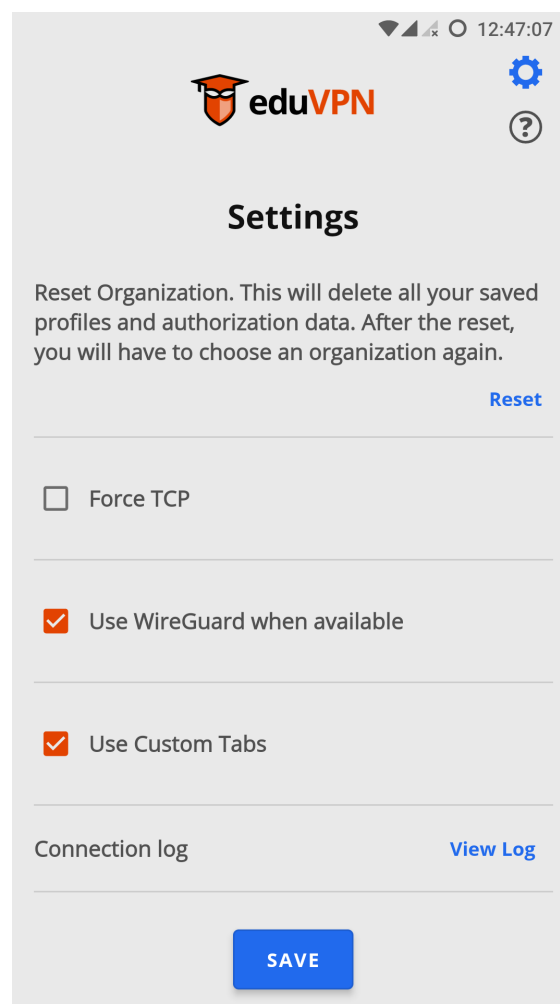


Figure 9: eduVPN Android app with WireGuard support: settings screen

When the user has enabled WireGuard, it has to be checked if the server supports WireGuard. To determine how and when to do this I had to know how the app works, and what communication already happens between the app and the server. To learn what exactly happens, I set up the eduVPN portal and API on my laptop and determined using [WireShark](#) what traffic was sent to eduVPN. I created 2

diagrams to visualize what exactly happens. The first diagram shows what happens when the user select his organization for the first time and has to log in. The second diagram show what happens when the user is already logged in and selects his organization.

Initial setup

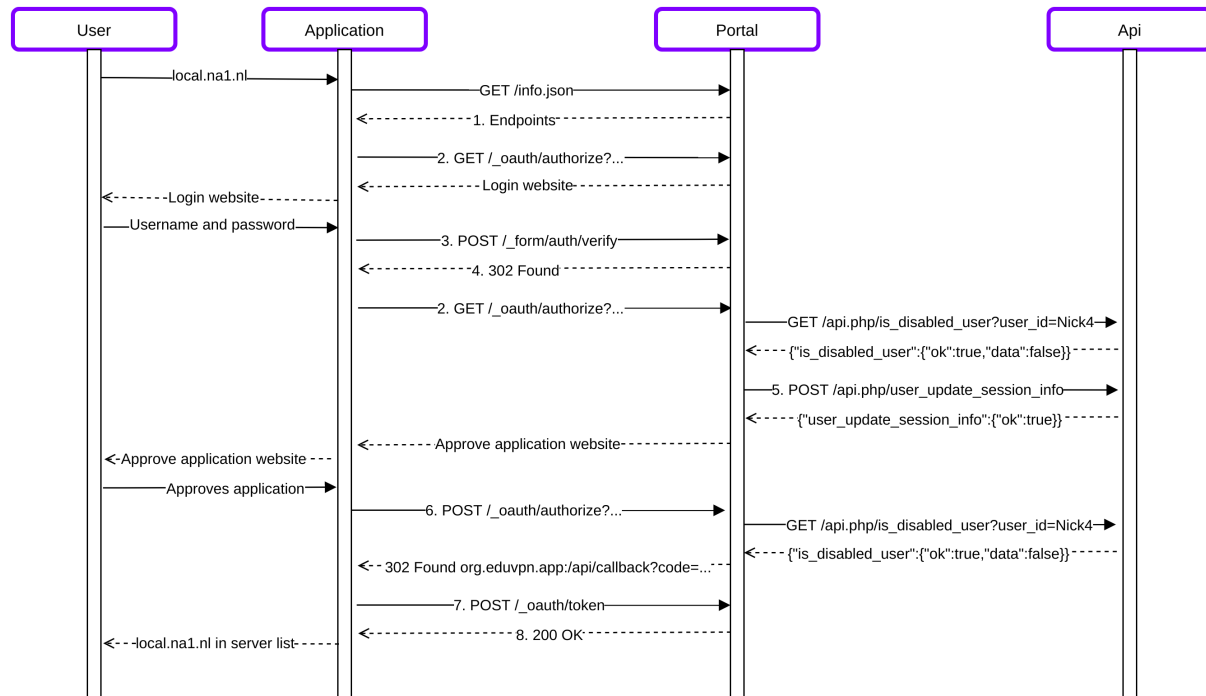


Figure 10: Sequence diagram when user uses the Android app for the first time

In the above diagram it is shown what interactions there are between the user, the application, and the eduVPN system to retrieve an authentication token that can be used to retrieve a VPN configuration. This happens when the user selects his institution for the first time. The numbers correspond to the technical details of the requests which can be found in the appendices.

Currently, there is traffic between the portal and the API as these are 2 different components in the eduVPN system. These components will be merged in the next major release of eduVPN.

After the user selects his institution, the app retrieves the endpoints where it has to connect to in the `info.json` file. It then tries to get an OAuth access token by opening the page where the user can open a deeplink which shares the access token with the eduVPN app. Before the user can open this page the user has to log in and after logging in the button can be pressed which approves to share the access token with the application.

Initialize connection

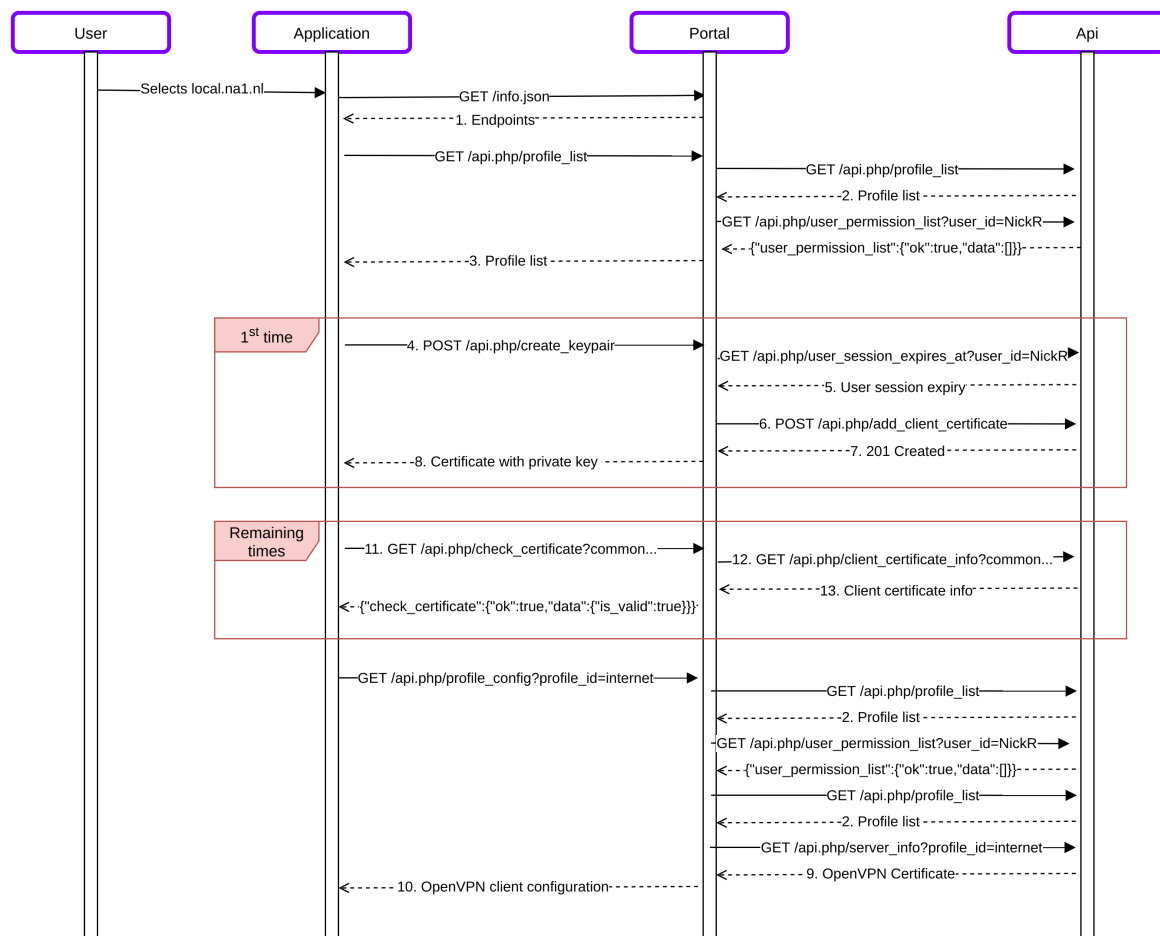


Figure 11: Sequence diagram when user starts OpenVPN connection

When the user wants to start a VPN connection for the first time, the app will request the server to create a key pair. The next time the app connects it already has a key pair and a certificate and will check if the certificate it has is still valid. The app will then retrieve the configuration that will be used to initiate the VPN connection. The numbers correspond to the technical details of the requests which can be found in the appendices.

Tests

When starting the development on the Android app, the UI tests did not work anymore. The tests logged in to eduVPN using Chrome, but Chrome changed its UI which broke the tests. These tests were fixed and a [pull request](#) was created. This pull request was merged.

Coroutines

When starting to do network requests in the Android App to make calls to the WireGuard API on the portal, I preferred to use [Kotlin Coroutines](#) which is the new and preferred way by JetBrains to do asynchronous programming. The WireGuard code depends on the existing API code which used a combination of callbacks and RxJava. Adding jet another asynchronous programming method would make the app harder to maintain. After a discussion with the Android developer, it was decided to replace RxJava with coroutines. I replaced RxJava with coroutines, this code was merged in the latest

version of eduVPN¹¹ and will be part of the next Android App release.

WireGuard

The server might enable or disable WireGuard at any time, so it would not suffice to only check for WireGuard support when doing the initial setup (logging in). When setting up the connection first the `info.json` is downloaded and then the OpenVPN specific API calls are made to the portal. One option is to sent in the `info.json` the information if WireGuard is enabled. The other option is to let the client make an API call that returns if WireGuard is enabled. After a discussion with the project manager and the lead developer it was chosen to implement an API call, so the WireGuard code is better split from the OpenVPN functionality. This makes it easier to keep the WireGuard fork of the eduVPN code up-to-date.

If the user enables WireGuard in the OpenVPN app, it will now make an API call to see if WireGuard is supported on the server:

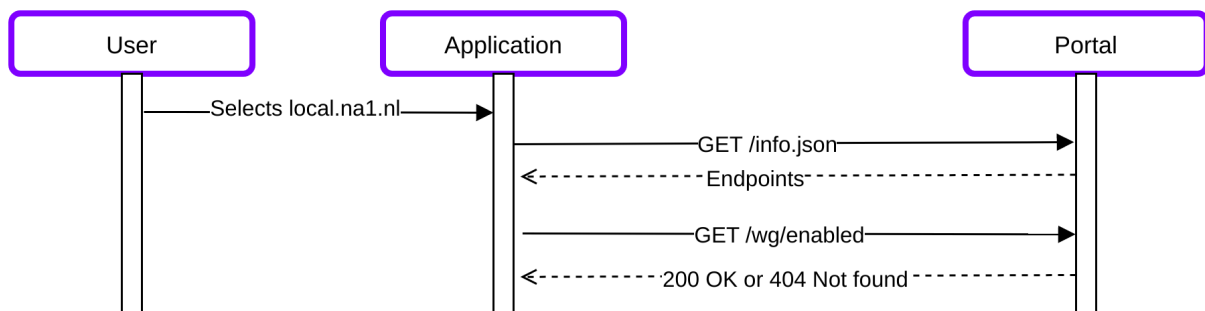


Figure 12: Sequence diagram WireGuard enabled check

When WireGuard is enabled, the server responds with a 200 OK and a body containing “Y”: a shorter version of **Yes**. Then the client creates a public private key pair and sends the public key to the server:

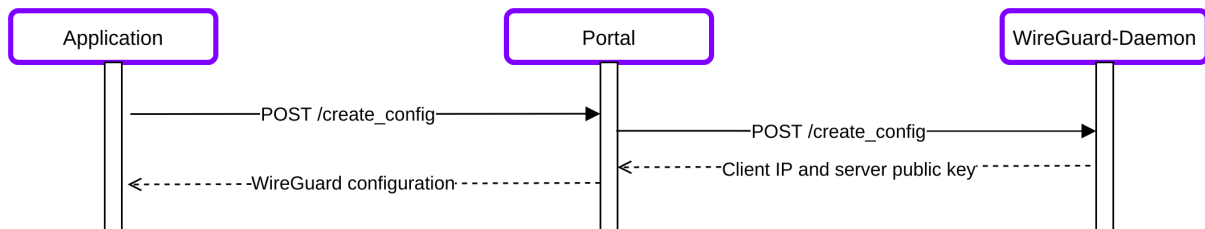


Figure 13: Sequence diagram creating WireGuard configuration

Then the client starts the VPN connection using the certificate it received. The key pair is saved and will be reused next time (this will change in Sprint 5).

After connecting, the user can see if the client is using OpenVPN or WireGuard in the **Connection info**:

¹¹Aquina, N. (2020b, December 10). Replace RxJava with Kotlin Coroutines by fantostisch · Pull Request #315 · eduvpn/android. GitHub. <https://github.com/eduvpn/android/pull/315>

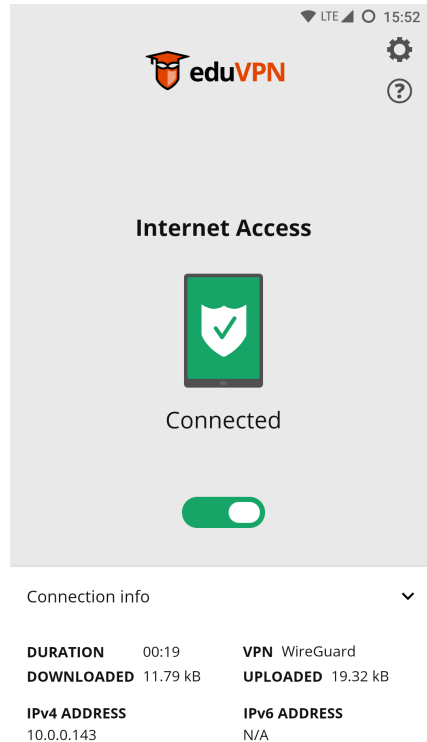


Figure 14: Screenshot Android eduVPN client connected using WireGuard

Other Android apps using WireGuard

When looking how to implement the WireGuard Android library into the eduVPN Android app, I looked at how other project implemented their WireGuard support.

The official WireGuard app was the most useful to look at as it used the Android library which I decided to use.

Mullvad is another VPN provider which published their VPN implementation online. They build their own cross-platform wrapper around the WireGuard Go implementation. Since we use the WireGuard Android library, looking further into their implementation will not help.

Another VPN provider which published their code is IVPN, they also use the WireGuard Android library, but they have copy-pasted this library into their app and have made changes to this implementation. For example, instead of implementing the necessary tunnel interface they changed this interface to a class.

Kernel space implementation

Most software runs in userspace, in userspace a program can not damage hardware or access memory from other programs. When a program runs in kernelspace, it has access to all memory and hardware.

WireGuard has a kernelspace implementation and a userspace implementation. The kernelspace implementation is faster, but is only available on Unix based systems (Linux, OpenBSD). Android uses the Linux kernel and could thus use the kernelspace implementation. Kernelspace programs can not be easily installed like an app, it requires the developers of Android to merge the WireGuard kernelspace implementation into Android's kernel. This was done in the (at the time of writing) latest unreleased Android version.¹² Currently the eduVPN Android app uses the [userspace Go implementation of WireGuard](#), but in the future it might be possible to use [the kernel implementation](#).

¹²Rahman, M. (2020, October 26). Google adds WireGuard VPN to Android 12's Linux Kernel. XDA-Developers. <https://www.xda-developers.com/google-adds-wireguard-vpn-android-12-linux-kernel-5-4/>

Code review

The WireGuard code in the Android app has been reviewed by the Android client maintainer, the code was improved based on this feedback, and then the code was merged into the [WireGuard branch](#).

Demo and review

At the end of the sprint a Demo was shown with the Android app to prove it could use WireGuard when enabled in the settings.

Sprint 5 (IP management)

Introduction

WireGuard uses static IP addresses. Because of this, an institution might run out of IPv4 addresses. This problem should be solved.

Research strategy

- Library, Community research: There will researched online if other people already solved the problems with dynamic IP allocation in WireGuard.

Implementation

An institution might want to give every VPN user a public IPv4 address. This is possible with WireGuard, but WireGuard uses static IP addresses which means that when a user connects using WireGuard it must already know the IP address it is going to use. This IP address is stored in the configuration that the eduVPN client receives from the server which is the same configuration that can be downloaded from the portal. Every time a user downloads a configuration, a public IP address will then be reserved for this configuration, but an institution might not have an IP available for every user for every device. An institution can not easily receive more public IPv4 addresses because there are not enough IPv4 addresses available for everyone¹³.

OpenVPN does not have this problem, because it uses dynamic IP allocation: when a device connects, the device receives an IP address from OpenVPN and when the device disconnects the IP becomes available for other users.

After searching online if other project already implemented dynamic IP allocation with WireGuard, [wg-dynamic](#) was discovered. wg-dynamic is an implementation of the WireGuard kernel module that uses dynamic IP allocation. wg-dynamic can unfortunately not be used in the Android client, because it is a kernel-module and it is not available in the Android Linux kernel. There is currently no userspace wg-dynamic implementation and developing this implementation might take a long time. To solve this problem it is necessary to tell the server that the user disabled the VPN and thus someone else can use our IP address. The official WireGuard Android app does not tell the server when the user disables the connection, but this disconnect functionality was implemented in the eduVPN Android app. When the user disables the connection, a request is sent to the server that we disabled the VPN.

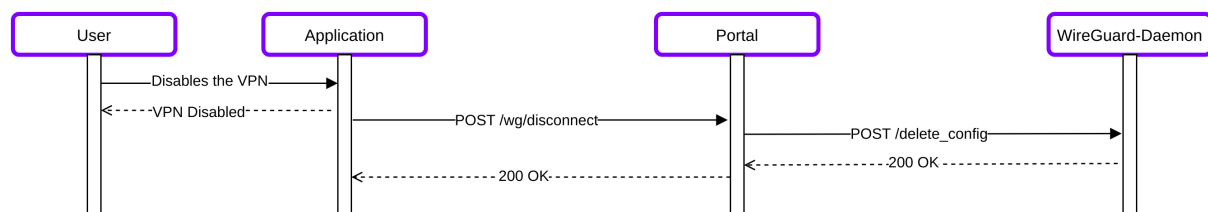


Figure 15: Sequence diagram eduVPN client WireGuard disconnect

¹³What is IPv4 Run Out? (2019, August 23). RIPE Network Coordination Centre. <https://www.ripe.net/manage-ips-and-asns/ipv4/ipv4-run-out>

Instead of creating 2 new features: release IP and request IP, it is simpler to remove the configuration when the user is done with the connection, and create a new configuration when the user starts a new VPN connection. Using this design it is not necessary to add more code to the WireGuard-Daemon, because it already has the functionality to create and delete a config. However, it should still be researched how creating a new key pair for every connection affects security.

Demo and review

At the end of Sprint 5 we were busy debugging an issue with the Android App on ChromeOS which made it impossible for ChromeOS users to use eduVPN. After this issue was resolved, I discussed the progress of IP management and how it was implemented with the project manager which agreed on the implementation.

Tasks for complete WireGuard support

This is an (incomplete) list of tasks that have to be done before WireGuard has the same functionality as OpenVPN in eduVPN.

Multiple WireGuard servers

In the original project plan it was planned to add support for multiple WireGuard servers. WireGuard should scale better than OpenVPN, because WireGuard uses multithreading and OpenVPN does not. To scale with OpenVPN, an OpenVPN process is necessary for every CPU core. Multiple WireGuard servers will thus be less quickly necessary than with OpenVPN, but when a second server is necessary in the long term to support more users, there is currently no easy way to integrate this second WireGuard server with an existing eduVPN installation.

WireGuard-Daemon security

Secure connection between portal and WireGuard-Daemon

Support for a secure connection between the portal and the WireGuard-Daemon was originally planned in the sprint where support would be added for multiple WireGuard servers. However, this sprint was replaced and currently there is no secure connection between the portal and the WireGuard-Daemon. This means that when the WireGuard server is not running on the same machine as the portal, the connection could be intercepted. This makes it impossible to have a WireGuard server and a portal that are for example located in different countries.

Since the WireGuard-Daemon uses an HTTP API this can be easily and quickly solved by adding a TLS proxy in front of the WireGuard-Daemon. Apache or NGINX can be used for this task.

WireGuard-Daemon authentication

Currently, everyone with access to the WireGuard-Daemon API can do requests, but this should be limited to the portal. When sending requests to the vpn-server-api, a **Basic Authorization** header has to be added to authenticate the request. In this header a username and password are sent to the vpn-server-api which can then check if this matches the username and password of a known component. The same method can be used for the WireGuard-Daemon.

TCP

There is currently no TCP support when using WireGuard. OpenVPN can be used over UDP and TCP. WireGuard can only be used over UDP. UDP is the preferred method as it is faster than TCP^{14,15}. The

¹⁴Why does use UDP and TCP? (n.d.). OpenVPN. Retrieved 7 January 2021, from <https://openvpn.net/faq/why-does-openvpn-use-udp-and-tcp/>

¹⁵What is TCP Meltdown? (n.d.). OpenVPN. Retrieved 7 January 2021, from <https://openvpn.net/faq/what-is-tcp-meltdown/>

disadvantage of UDP is that UDP might be blocked on restrictive networks¹⁶.

Configuration expiry

When a student leaves his university, his account might be disabled in the system the university uses to register all user accounts (e.g. RADIUS or LDAP), but the eduVPN system is not notified when this user is disabled. When a user has already downloaded an OpenVPN configuration this configuration will keep working until a specified time. WireGuard configurations do not expire at all, so unless the user is explicitly disabled using the portal, the user can still use a WireGuard VPN connection. Instead of a configuration that expires, this problem could also be solved by notifying the eduVPN system of users that are disabled or removed from the authentication system.

eduVPN Profiles

When using eduVPN with OpenVPN, different type of users (students, teachers, external users) can be assigned a different configuration. No such functionality has been implemented for WireGuard. All users get the exact same configuration.

IPv6

Currently, the WireGuard-Daemon only assigns IPv4 addresses to configurations. IPv6 support should be added.

Support in all eduVPN clients

WireGuard support was added to the Android eduVPN client, but support for the Windows, Linux, macOS and iOS client has yet to be developed.

Consider using the pre-shared symmetric key mode

WireGuard has an optional pre-shared symmetric key mode which is required for post-quantum resistance¹⁷. This mode is currently not used, but it should be used to protect against attacks using quantum computers.

¹⁶Why does use UDP and TCP? (n.d.). OpenVPN. Retrieved 7 January 2021, from <https://openvpn.net/faq/why-does-openvpn-use-udp-and-tcp/>

¹⁷Donenfeld, J. A. (n.d.-a). Protocol & Cryptography - WireGuard. WireGuard. Retrieved 8 January 2021, from <https://www.wireguard.com/protocol/>

Recommendations

Implement the tasks for complete WireGuard support

As described in the section `Tasks for complete WireGuard support`, there are still tasks that have to be done before WireGuard support is complete.

Drop support for CentOS

Because of the support for CentOS, eduVPN is stuck with PHP 5.4 (PHP 8 is available at the time of writing). eduVPN can not use libraries that most other projects use, because most projects have dropped support for PHP 5. Existing CentOS installations can be replaced with Debian.

Improve Android app test speed

Running the unit tests of the Android app takes a lot of time. Especially the UI tests are slow. Most of the time the phone is doing nothing while running the UI tests. This prevents developers from running these tests which makes them less useful. The developer experience would be greatly improved if the tests run in a few seconds.

Conclusions

eduVPN is a VPN used by institutions like universities and research labs to allow their users to securely connect to the internet and to connect to their institutions' network to access internal services.

eduVPN is built using OpenVPN. OpenVPN is a widely used VPN implementation. Since 2016 a new VPN implementation was released to the public: WireGuard. WireGuard aims to be faster, simpler, and more useful. In this internship I improved and extended the existing WireGuard support.

The planned functionality was implemented and a proof of concept showing WireGuard integration in eduVPN has been successfully delivered. The official WireGuard client can be used to set up a VPN connection to eduVPN and on Android the eduVPN app can be used. Admins can view the WireGuard configurations of their users, view who is currently using WireGuard and block users. Admins can now easily deploy eduVPN with WireGuard support on their servers.

WireGuard support in eduVPN is not yet production ready and multiple tasks have to be completed before a release to production. If you want to help complete these tasks, you can start today. The necessary code repositories are linked in the introduction.

Evaluation

I enjoyed working for SURF and working at the eduVPN project. Working on a feature that is actually going to be used at some point motivated to work on the project. It also motivates when your code is merged into the main branch of the software and will be part of the next release (Android app).

Because of an ongoing pandemic I was only at the SURF office every 1-2 weeks which required communicating using voice and video chat. This made it a little harder to communicate, but I think the impact was minimal as we had a digital meeting every week and when I had questions I could always ask for help using digital communications. Even if I could have been more at the SURF office, online communications would still be required as the lead developer and the Android app developer both live abroad. After every Sprint a demo was shown which made it clear what worked and what did not work, this way the progress made was clear.

I learned a lot while working on the eduVPN project. I learned how eduVPN works and learned about OpenVPN and WireGuard. To develop the WireGuard-Daemon I had to learn the Go programming language which I had never used before. I also learned how to package software for Debian and Fedora, in particular Go and PHP packages. Doing the packaging myself made me realize how much work it takes to keep Linux distributions up-to-date by packaging every new version of software. I also learned how Android applications work and how to develop them. Android development introduced me to new concepts like dependency injection, how front-ends can be built and how to make the front-end instantly react to changes like a VPN tunnel that gets enabled or disabled.

Appendices

Initial setup request details

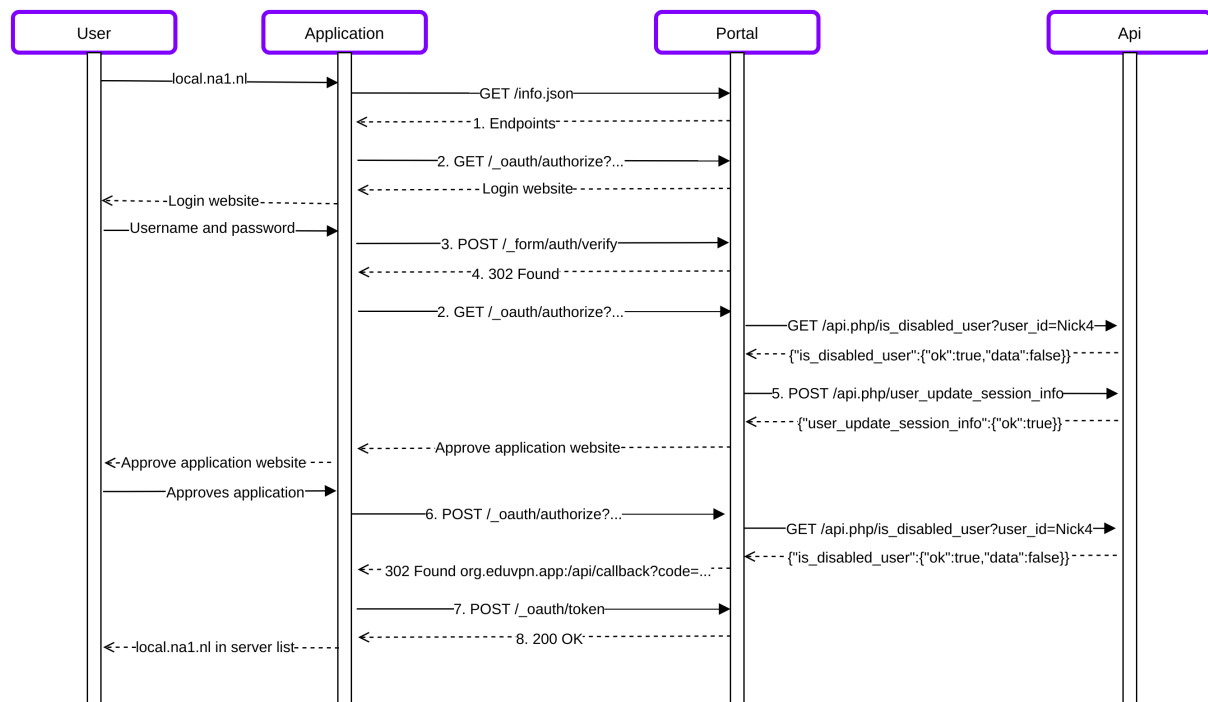


Figure 16: Sequence diagram when user uses the Android app for the first time

1. Endpoints

```

{
  "api": {
    "http://eduvpn.org/api#2": {
      "api_base_uri": "https://local.na1.nl/api.php",
      "authorization_endpoint": "https://local.na1.nl/_oauth/authorize",
      "token_endpoint": "https://local.na1.nl/oauth.php/token"
    }
  },
  "v": "2.3.4"
}

```

2. GET /_oauth/authorize?...

Parameter	Value
redirect_uri	org.eduvpn.app:/api/callback

Parameter	Value
client_id	org.eduvpn.app.android
response_type	code
state	VW9Gt+5nBACO9kLAovDYM5JMZtGLgxIw
nonce	pEeIErze__4ZgUnt0U5OSg
scope	config
code_challenge	XcrDHLbADeRk-HNSnWxo4A7xLHnYUU1PAHH0Tgx3A24
code_challenge_method	S256

3. POST /__form/auth/verify

Parameter	Value
userName	Nick4
userPass	password4
__form_auth_redirect_to	http://localhost:8082/__oauth/authorize?redirect_uri=org.eduvpn.app:/api/callback
client_id	org.eduvpn.app.android
response_type	code
state	VW9Gt+5nBACO9kLAovDYM5JMZtGLgxIw
nonce	pEeIErze__4ZgUnt0U5OSg
scope	config
code_challenge	XcrDHLbADeRk-HNSnWxo4A7xLHnYUU1PAHH0Tgx3A24
code_challenge_method	S256

4. 302 Found

todo

Location: http://localhost:8082/__oauth/authorize?redirect_uri=org.eduvpn.app%3A%2Fapi%2Fcallback&cli

5. POST /api.php/user_update_session_info

Parameter	Value
user_id	Nick4
session_expires_at	2020-12-27T10:34:29+01:00
permission_list	[]

6. POST /__oauth/authorize?...

URL

Parameter	Value
redirect_uri	org.eduvpn.app:/api/callback
client_id	org.eduvpn.app.android
response_type	code
state	VW9Gt%2B5nBACO9kLAovDYM5JMZtGLgxIw
nonce	pEeIErze__4ZgUnt0U5OSg
scope	config
code_challenge	XcrDHLbADeRk-HNSnWxo4A7xLHnYUU1PAHH0Tgx3A24
code_challenge_method	S256

Postdata

Parameter	Value
approve	yes

7. POST /__oauth/token

Parameter	Value
code	eyJhbGc...
grant_type	authorization_code
redirect_uri	org.eduvpn.app%3A%2Fapi%2Fcallback
code_verifier	W_WFD-8xo...
client_id	org.eduvpn.app.android

8. 200 Ok

```
{
  "access_token": "eyJhbGc...i7x-Cg",
  "refresh_token": "eyJhbGc...VUn-Bg",
  "token_type": "bearer",
  "expires_in": 3600
}
```

Connection setup request details

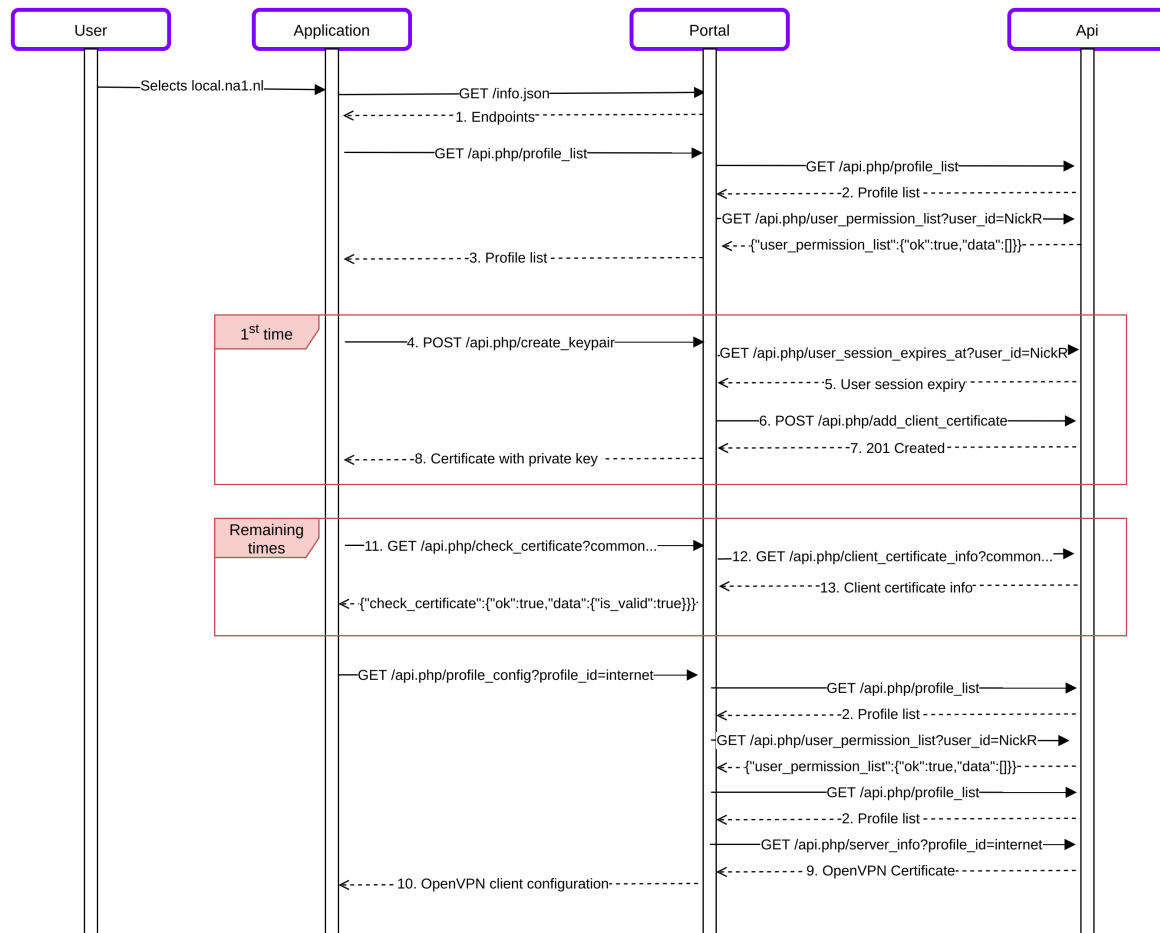


Figure 17: Sequence diagram when user starts a OpenVPN connection

1. Endpoints

```

{
  "api": {
    "http://eduvpn.org/api#2": {
      "api_base_uri": "https://local.na1.nl/api.php",
      "authorization_endpoint": "https://local.na1.nl/_oauth/authorize",
      "token_endpoint": "https://local.na1.nl/oauth.php/token"
    }
  },
  "v": "2.3.5"
}

```

2. Profile list send from API to portal

Todo: This a profile list of an incorrectly configured server, so this example is not representative.

```

{
  "profile_list": {
    "ok": true,
    "data": {
      "internet": {
        "defaultGateway": true,

```

```

    "routes": [],
    "dns": [
        "192.168.11.1"
    ],
    "dnsSuffix": [],
    "clientToClient": false,
    "listen": "::",
    "enableLog": false,
    "enableAcl": false,
    "aclPermissionList": [],
    "managementIp": "127.0.0.1",
    "vpnProtoPorts": [
        "udp/1194",
        "tcp/1194"
    ],
    "exposedVpnProtoPorts": [],
    "hideProfile": false,
    "tlsProtection": "tls-crypt",
    "blockLan": true,
    "profileNumber": 1,
    "displayName": "Internet Access",
    "range": "10.207.1.0/25",
    "range6": "fd9c:f2a2:8cbb:8b77::/64",
    "hostName": "local.na1.nl"
}
}
}
}
}

```

3. Profile list send from portal to client

```

{
  "profile_list": {
    "ok": true,
    "data": [
      {
        "profile_id": "internet",
        "display_name": "Internet Access",
        "two_factor": false,
        "default_gateway": true
      }
    ]
  }
}
}

```

4. POST /api.php/create_keypair

Parameter	Value
display_name	eduVPN for Android

5. User session expiry

```

{
  "user_session_expires_at": {
    "ok": true,
    "data": "2021-02-09T14:08:53+01:00"
  }
}

```



```
}
```

6. POST /api.php/add_client_certificate

Parameter	Value
user_id	NickR
display_name	org.eduvpn.app.android
client_id	org.eduvpn.app.android
expires_at	2021-02-09T14:08:53+01:00

7. 201 Created

```
{
  "add_client_certificate": {
    "ok": true,
    "data": {
      "certificate": "-----BEGIN CERTIFICATE-----\nMIIEHDCCA...lsdL\n-----END CERTIFICATE-----",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIG/gIBA...HyAP\n-----END PRIVATE KEY-----",
      "valid_from": 1605099991,
      "valid_to": 1612876133
    }
  }
}
```

8. Certificate with private key

```
{
  "create_keypair": {
    "ok": true,
    "data": {
      "certificate": "-----BEGIN CERTIFICATE-----\nMIIEHDCCA...lsdL\n-----END CERTIFICATE-----",
      "private_key": "-----BEGIN PRIVATE KEY-----\nMIIG/gIBA...HyAP\n-----END PRIVATE KEY-----"
    }
  }
}
```

9. OpenVPN Certificate

```
{
  "server_info": {
    "ok": true,
    "data": {
      "tls_crypt": "#\n# 2048 bit OpenVPN static key\n#\n-----BEGIN OpenVPN Static key V1-----\nndbeb",
      "ca": "-----BEGIN CERTIFICATE-----\nMIIEEDCCA...RneB\n-----END CERTIFICATE-----"
    }
  }
}
```

10 OpenVPN client configuration

```
# OpenVPN Client Configuration
dev tun
client
nobind
remote-cert-tls server
verb 3
server-poll-timeout 10
ncp-ciphers AES-256-GCM
```

```

cipher AES-256-GCM
reneg-sec 0
<ca>
-----BEGIN CERTIFICATE-----
MIIEDCCA...neB
-----END CERTIFICATE-----
</ca>
tls-version-min 1.2
tls-cipher TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384:TLS-ECDHE-ECDSA-WITH-AES-256-GCM-SHA384
<tls-crypt>
#
# 2048 bit OpenVPN static key
#
-----BEGIN OpenVPN Static key V1-----
dbeb...0fd6
-----END OpenVPN Static key V1-----
</tls-crypt>
remote local.na1.nl 1194 udp
remote local.na1.nl 1194 tcp

```

11 GET /api.php/check_certificate?common_name=b535bf74bfd3e39ba9b3a5f68f102612

12 GET /api.php/client_certificate_info?common_name=b535bf74bfd3e39ba9b3a5f68f102612

13 Client certificate info

```

{
  "client_certificate_info": {
    "ok": true,
    "data": {
      "user_id": "NickR",
      "user_is_disabled": "0",
      "display_name": "org.eduvpn.app.android",
      "valid_from": "2020-11-12T12:11:13+00:00",
      "valid_to": "2021-02-10T12:14:37+00:00",
      "client_id": "org.eduvpn.app.android"
    }
  }
}

```